Despliegue Automatizado sobre contenedores

ANSIBLE

LXC

Asir
Álvaro Rodríguez Márquez

Indice

Introducción

La idea automatizar el escenario usado principalmente en servicios surgió cuando se nos enseñó algo de **ansible** , ya que esto me permitiría poder tener el escenario de practicas del cloud replicado en mi casa, así podría seguir trabajando con la última versión de este aunque el **cloud** estuviese "caido" o inaccesible ,o para aquellos casos en los que la prueba requería de la instalación de paquetes que pudiesen afectar al funcionamiento de las máquinas, dandonos un escenario de pruebas en el que probar antes de pasar a "*producción*".

Objetivo del proyecto

Despliegue de forma automática de un escenario de pruebas para el curso.

El escenario podría ser utilizado para la realización de prácticas , disponiendo de un escenario previamente configurado desde el que partir, reciclable en caso de tener que volver a empezar o requerir un nuevo lanzamiento de las máquinas en "limpio".

El despliegue tendría que ser sencillo y con una necesidad mínima de que el usuario intervenga en la configuración de las máquinas, permitiéndole centrase en la práctica y no en la preparación del escenario para esta.

Descripción del escenario

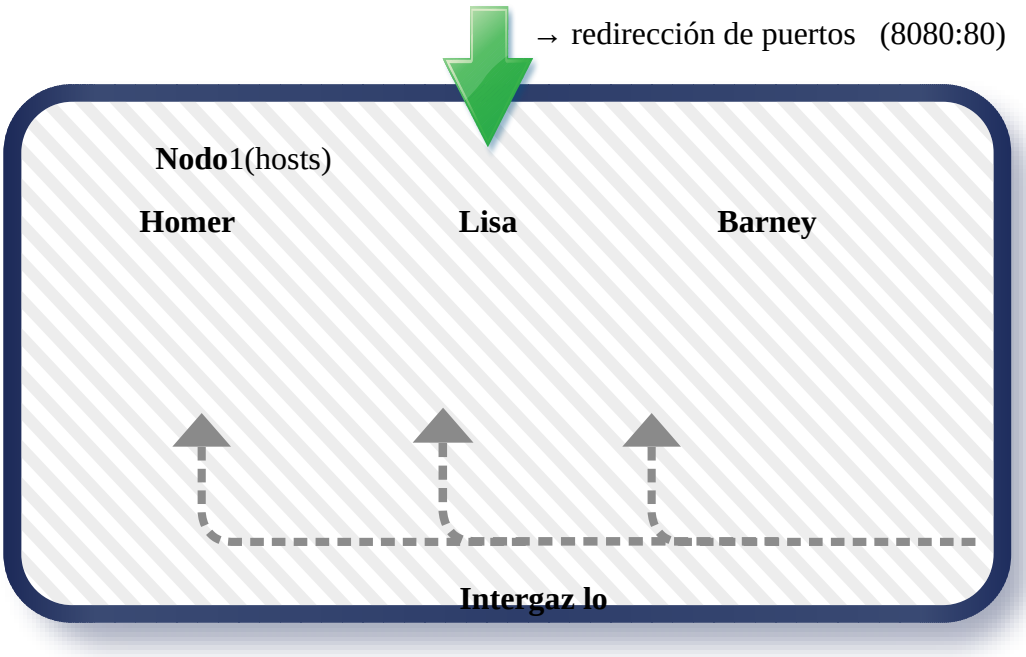
El que se desplegará constará de tres contenedores hospedados en la misma máquina desde la que operaremos, lanzandose las ordenes por **lo**.

Las máquinas usaran los mismos nombres que las tres usadas este año, Homer, Lisa y Barney.

Cada uno tendrá su servicio principal: Barney → DNS

Lisa → Servidor Web

Homer → Servidor MySQL



Introducción a LXC

Linux Containers o LXC es una tecnología de virtalización a nivel del SO, lo que implica que no virtualiza del hardware, llegando a usar el mismo **kernel** del “anfitrión”.

¿Por que lxc?

Por que LXC puede desplegarse en cualquier **Gnu/linux** , no requiere de mucha potencia, ni aceleración por hardware, lo cual permite desplegarlo en cualquier máquina, incluso en una virtual.

Además de que no tenemos que crear las imágenes para trabajar sobre ellas, como pasaría con **KVM** .

Su instalación es simple:

```
apt install lxc
```

O en caso de ser centos:

```
yum install lxc
```

Ejemplos de lxc

Creación de contenedor en LXC:

```
lxc-create -t debian -n ejemplo-proyecto
```

Podemos ver las máquinas creadas con:

```
lxc-ls
ejemplo-proyecto maquinaorigen    sandbox
```

(Para poder hacer un **lxc-ls** en centos necesitamos el siguiente paquete: lxc-extra.x86_64)
E Iniciamos cualquiera de ellas con:

```
lxc-start -n ejemplo-proyecto
```

Tras eso unimos una consola al contenedor con:

```
lxc-attach -n ejemplo-proyecto
root@ejemplo-proyecto:/# hostname
ejemplo-proyecto
```

O podemos iniciar la consola al la vez que lanzamos el contenedor:

```
lxc-start -n lisa -F
```

Podemos parar cualquier contenedor con:

```
lxc-stop -n ejemplo-proyecto
```

Y eliminarlo con esta orden:

```
lxc-destroy -n ejemplo-proyecto
Destroyed container ejemplo-proyecto
```

Introducción a Ansible

Es una herramienta de orquestación que permite configurar y administrar de forma automática .
Ansible maneja otras máquinas mediante el protocolo **ssh** por lo que no requiere la instalación de un agente propio.
Cuando se quiere manejar una máquina con **Ubuntu**, con este sistema tendremos que instalar **sshpass** o nos dará problemas de conexión.

Ansible se sirve de sus **playbook** para realizar tareas realizándolas sobre máquinas, grupos de máquinas o aprovechando las funcionalidades extras que ofrecen los roles.

Ejemplos con Ansible

Podemos probar la conexión desde nuestra máquina a otra (a si misma, ya sea por **lo** o una interfaz externa dará problemas en debian Jessie).

Si lo hicieramos por comandos sería así:

```
ansible matrix -m ping -u usuario -k
SSH password:
192.168.0.61 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

El host **Matrix** tendremos que haberlo añadido previamente a un fichero llamado **hosts** que estará en el directorio desde el que lanzaremos las ordenes.

Ejemplo de host para la máquina **matrix**:

```
[matrix]
192.168.0.61
```

También podemos lanzar esta operación con un **playbook** , el cual se llamaría , por ejemplo, **ping.yml** y tendría una sintaxis como esta:

```
---
 - name: prueba ping
   hosts: matrix
   vars:
    - ansible_user: usuario
 - tasks:
    - action: ping
```

Y se usaría así:

```
ansible-playbook ping.yml -k
SSH password:

PLAY [prueba ping] ************************************************************

TASK [setup] *******************************************************************
ok: [192.168.0.61]
```

Elevación de privilegios

Para algunas tareas como la instalación de paquetes necesitaremos permisos de root.

Para conseguir esos privilegios podemos hacerlo de las siguientes formas:

Sudo_user

Con este módulo pedimos la elevación del usuario actual a root durante la tarea si va emplazado tras **tasks** o de forma global al playbook si está en la parte de **hosts**.

Si el usuario no puede elevar sus privilegios sin contraseña tendremos que pasarla, esto podemos hacerlo de varias formas.

Si queremos que la contraseña no aparezca en ningún fichero podemos pasarla directamente en el comando con el que lanzamos el **playbook**, para ello añadiremos al playbook las siguientes opciones:

```
---
 - hosts: local
   sudo_user: root ← indicaríamos el usuario al que queremos “elevarnos”
   tasks:

    - name: Prueba sudo_user
      lxc_container:
        name: ubuntu-temporal
        container_log: true
        template: ubuntu
        state: started
        template_options: --release trusty
```

Y lo lanzamos de la siguiente manera:

```
root@nodo-1:/etc/ansible# ansible-playbook prueba-sudo_user.yml -u usuario -k
SSH password:
[DEPRECATION WARNING]: Instead of sudo/sudo_user, use become/become_user and make sure become_method is 'sudo'
(default). This feature will be removed in a future release. Deprecation warnings can be disabled by setting
deprecation_warnings=False in ansible.cfg.
```

Aparecerá un **warning** tal como podemos ver, el cual nos indica que estamos usando una opción en estado **deprecated** .

Una alternativa menos segura sería dejar la contraseña como variable dentro de un fichero, ya sea el propio **playbook** , el archivo de **hosts** u otro al que se llame desde alguno de los anteriores.

Aunque **sudo_user** sigue funcionando hay alternativas mejores, con funcionalidades extra implementadas, por ello he preferido usar **become**.

become

Become se usa de foma similar a sudo_user , podemos conseguir la elevación global para un playbook, colocando las opciones de become en el apartado de **hosts** o de forma individual por tarea, poniendolo tras el **name** de cada tarea .

Un ejemplo de **become** sería un playbook para levantar una máquina en lxc, ya que por defecto esto solo puede hacerlo root:

```
---
 - hosts: local
   become: yes              ← Activa la opción para cambiarte de usuario.
   become_user: root    ← Es la variable a la que  añadimos el nombre de usuario al
que queremos pasar.

   tasks:

    - name: prueba contenedor prueba_become
      lxc_container:
        name: ubuntu-prueba_become
        container_log: true
        template: ubuntu
        state: started
        template_options: --release trusty
```

Para lanzarlo necesitaríamos pasar la variable "**ansible_sudo_pass**" con la opción –**extra-vars** .

Quedaría de la siguiente forma:

```
root@nodo-1:/etc/ansible# ansible-playbook ubuntu-prueba.yml -u usuario -k --extra-vars "ansible_sudo_pass=usuario"
SSH password:

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [127.0.0.1]

TASK [prueba contenedor prueba_become] *****************************************
changed: [127.0.0.1]

PLAY RECAP *********************************************************************
127.0.0.1                  : ok=2    changed=1    unreachable=0    failed=0
```

Como podemos ver ya no salta ningún **warning.**

El gran inconveniente de lanzarlo así es que dejamos la contraseña en claro, no solo durante el tiempo que esté en pantalla, se quedará en el historial.

La alternativa a esto sería permitir al usuario una elevación de privilegios sin contraseña o guardar su contraseña en un fichero, con los permisos adecuados.

Playbook con lxc

Una vez conozcamos algunos conceptos de lxc podremos empezar a crear contenedores de forma automática con ansible

En primer lugar tendríamos que añadir una entrada hosts con la dirección de la máquina en la que vamos a desplegar :

```
[local]
127.0.0.1
```

Tras eso crearíamos el playbook, en este caso lo he llamado **ubuntu-prueba.yml** para que el nombre sea claro, el playbook contiene las opciones básicas para un despliegue de lxc:

```
---
 - hosts: local
   become: yes
   become_user: root

   tasks:

    - name: prueba contenedor
      lxc_container:
        name: ubuntu-prueba
        container_log: true
        template: ubuntu
        state: started
        template_options: --release trusty
```

Y lo usaríamos de estaforma:

```
root@nodo-1:/etc/ansible# ansible-playbook ubuntu-prueba.yml -u root -k
SSH password:

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [127.0.0.1]

TASK [prueba contenedor] *******************************************************
changed: [127.0.0.1]

PLAY RECAP *********************************************************************
127.0.0.1                  : ok=2    changed=1    unreachable=0    failed=0

root@nodo-1:/etc/ansible# lxc-ls
jean          ubuntu-prueba
```

La opción “**-u <nombre de usuario>**” sirve para declarar el nombre del usuario con el que accederemos a la máquina, lo uso en este caso ya que estoy en el directorio original de ansible y

este tiene los permisos de root. Con un usuario común podría ejecutar los playbook pero en caso de error no se crearía el .retry para continuar la operación ya que no tendría permisos.

Tal como está el playbook podemos ejecutarlo como un usuario normal, siempre que tenga permisos de root, para ello lo haríamos de la siguiente forma:

```
root@nodo-1:/etc/ansible# ansible-playbook ubuntu-prueba.yml -u usuario -k --extra-vars "ansible_sudo_pass=usuario"
SSH password:

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [127.0.0.1]

TASK [prueba contenedor] *******************************************************
ok: [127.0.0.1]

PLAY RECAP *********************************************************************
127.0.0.1                  : ok=2    changed=0    unreachable=0    failed=0
```

La contraseña puede almacenarse en un fichero al que se llame desde el playbook o en el mismo.

Ya podríamos acceder a la máquina con lxc-console:

```
Ubuntu 14.04.5 LTS ubuntu-prueba tty1

ubuntu-prueba login: root
Password:
Last login: Sat Jun 10 16:04:55 CEST 2017 on lxc/tty1
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 4.4.0-77-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
root@ubuntu-prueba:~#
```

Preparación del anfitrión

Como anfitrión he usado ubuntu Xenial tras haber probado con centos y debian.

Comencé probando con debian pero la versión que traía de ansible no daba soporte para lxc(creación de playbook que levantasen contenedores).

Tras eso probé con centos, pero la instalación del componente lxc-python2 daba problemas hasta tal punto que me topaba con errores constantes.

Instalación de componentes - Manual

Los componentes a instalar son :

- Ansible
- LXC
- Bridge-utils
- libvirt
- debootstrap
- perl
- pip
- python-lxc (desde apt en ubuntu o desde pip si intentas hacerlo en centos o debian)
- yum (para la instalación del contenedor con centos)

Podemos instalarlo directamente con las siguientes órdenes:

```
apt install ansible lxc bridge-utils libvirt debootstrap perl python-lxc yum
```

Si decidimos hacerlo en centos (**NO RECOMENDADO**) tendremos que activar lxc y libvirt, podemos hacerlo de la siguiente forma:

```
systemctl start lxc.service
systemctl start libvirtd
```

Con eso ya tendríamos suficiente para comenzar los despliegues.

Despliegue del escenario

El escenario está compuesto por máquinas montadas sobre Linux Container, las cuales se instalaran y configurarán mediante “recetas” con unos servicios concretos, permitiéndonos usarlas para hacer pruebas sobre ellas, por ejemplo para tareas de clase.

Creación de contenedores

Comenzamos creando los contenedores de las máquinas,

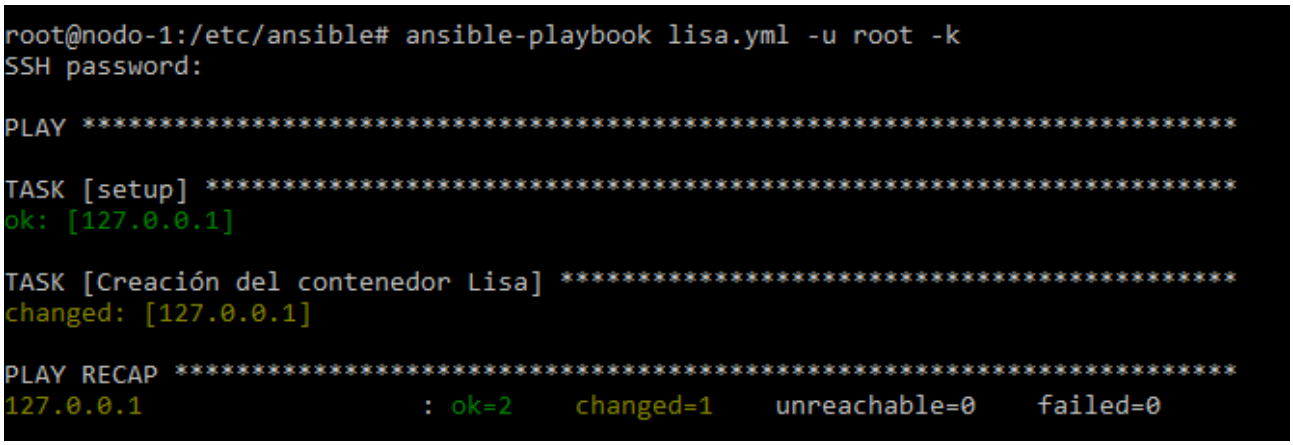
Lisa

Comenzamos con la creación del contenedor **Lisa** , el cual es un centos7 que albergará un servidor web , para este necesitábamos tener instalado el paquete **yum** , sin el no podremos instalarlo.

Su playbook sería algo así:

```
---
 - hosts: local
   become: yes
   become_user: root

   tasks:

    - name: Creación del contenedor Lisa
      lxc_container:
        name: lisa
        container_log: true
        template: centos
        state: started
```

Y lo lanzamos:



Homer

Homer es una máquina con **Ubuntu** 16.04 , el cual tenía una base de datos.

El playbook para lanzar el contenedor sería este:

```
---
 - hosts: local
   become: yes
   become_user: root

   tasks:

    - name: Creación del contenedor Homer
      lxc_container:
        name: homer
        container_log: true
        template: ubuntu
        state: started
        template_options: --release trusty
```

Lo lanzamos igual que el playbook para lisa:

```
root@nodo-1:/etc/ansible# ansible-playbook homer.yml -u root -k
SSH password:

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [127.0.0.1]

TASK [Creación del contenedor Homer] *******************************************
changed: [127.0.0.1]

PLAY RECAP *********************************************************************
127.0.0.1                  : ok=2    changed=1    unreachable=0    failed=0
```

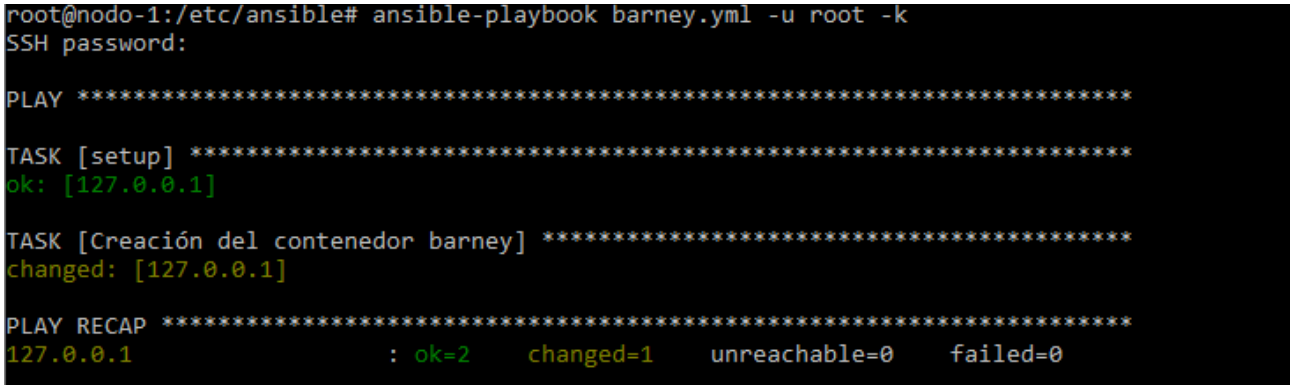
Barney

Esta máquina **debian jessie** tenía varios servicios, aunque el que mas se usó en las distintas asignaturas es el **DNS**.

Su playbook sería así:

```
---
 - hosts: local
   become: yes
   become_user: root

   tasks:

    - name: Creación del contenedor Barney
      lxc_container:
        name: barney
        container_log: true
        template: debian
        state: started
        template_options: --release jessie
```

Lo lanzamos como los dos anteriores:



Comunicación con las máquinas

Tras crearlas podremos acceder a ellas directamente con lxc-console pero esto nos obliga a realizar las tareas de forma manual, teniendo que repetir las tareas entre las distintas máquinas .

Queremos ahorrarnos eso, para ello tenemos que permitir a ansible llegar a las máquinas.

Ansible nos permite ejecutar las tareas directamente en los contenedores con el módulo **lxc** pero de una forma ineficiente ya que lo único que hace es pasarle una o varias ordenes para que las ejecute el contenedor directamente.

Lo que vamos a hacer es crear una conexión por ssh para poder operar con ansible en los contenedores.

Primero tenemos que saber las ips de las máquinas, para ello podemos usar **lxc-ls** con la opción

"**--fancy"** nos mostrará las distintas máquinas con sus estados y correspondientes ips:

```
root@nodo-1:/etc/ansible# lxc-ls --fancy
NAME                STATE   AUTOSTART GROUPS IPV4       IPV6
barney              RUNNING 0         -      10.0.3.185 -
cenots              STOPPED 0         -      -          -
centos-prueba       STOPPED 0         -      -          -
centos-rebirth      RUNNING 0         -      10.0.3.251 -
debian              STOPPED 0         -      -          -
homer               RUNNING 0         -      10.0.3.55  -
jean                STOPPED 0         -      -          -
lisa                RUNNING 0         -      10.0.3.163 -
ubuntu-prueba       STOPPED 0         -      -          -
ubuntu-prueba_become RUNNING 0         -      10.0.3.128 -
```

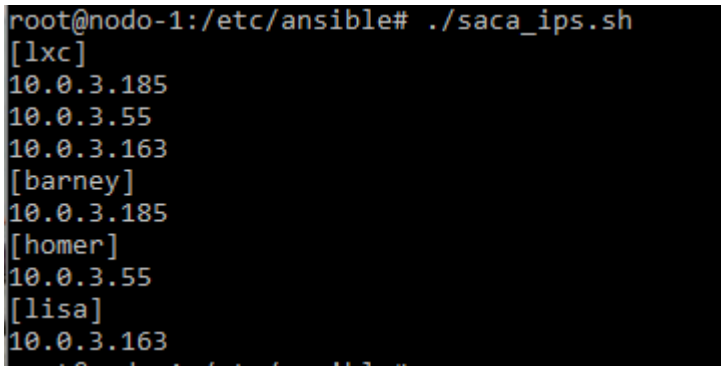
Ahora tendríamos que añadirlas al archivos **hosts** dejándolo de este modo:

```
[lxc]
10.0.3.163
10.0.3.55
10.0.3.185

[lisa]
10.0.3.163

[homer]
10.0.3.55

[barney]
10.0.3.185
```

O podríamos pasar el siguiente script, que nos da directamente las ips dentro de sus correspondientes campos

```
#!/bin/bash

barney="$(lxc-ls --fancy |grep barney |cut -d '-' -f 2)"
homer="$(lxc-ls --fancy |grep homer |cut -d '-' -f 2)"
lisa="$(lxc-ls --fancy |grep lisa |cut -d '-' -f 2)"
echo "[lxc]"
echo $barney
echo $homer
echo $lisa
echo "[barney]"
echo $barney
echo "[homer]"
echo $homer
echo "[lisa]"
echo $lisa
#inserción de datos
echo -e "[lxc] \n" $barney "\n"$homer "\n" $lisa "\n [barney] \n" $barney "\n
[homer] \n" $homer "\n [lisa] \n"  $lisa >> hosts

echo -e "\nbarney:"$barney"\nhomer:"$homer"\nlisa:"$lisa >> roles/dns/files/ips.yml
```

Podemos coger los campos ya preparados y copiarlos al fichero:



Este “programita” también guarda las ips en un archivo **.json** que aprovecharemos después para configurar nuestro **DNS**.

Ahora cambiaríamos la contraseña del usuario root e instalaríamos **openssh-server** y **sshpass**, para ello también he preparado unos playbooks:

Playbook para barney (También utilizable para **homer**)

```
---
 - hosts: local
   become: yes
   become_user: root
   tasks:
    - name: instalación ssh barney
      lxc_container:
        name: barney
        container_command: apt-get update | apt-get install openssh-server sshpass
sudo  python-minimal apt-utils python-apt -y

    - name: creación de usuario
      lxc_container:
        name: barney
        container_command: useradd usuario -s /bin/bash -m

    - name: configuración del usuario
      lxc_container:
        name: barney
        container_command: |
          echo -e "usuario\nusuario" | passwd usuario

    - name: configuración de sudo
      lxc_container:
        name: barney
        container_command: |
          echo "usuario ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers
```

Playbook para homer

```
---
 - hosts: local
   become: yes
   become_user: root
   tasks:
    - name: instalación ssh homer
      lxc_container:
        name: homer
        container_command: apt-get update | apt-get install openssh-server sshpass
sudo python-minimal -y

    - name: creación de usuario
      lxc_container:
        name: homer
        container_command: useradd usuario -s /bin/bash -m

    - name: configuración del usuario
      lxc_container:
        name: homer
        container_command: |
          echo -e "usuario\nusuario" | passwd usuario

    - name: configuración de sudo
      lxc_container:
        name: homer
        container_command: |
          echo "usuario ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers
```

Playbook para lisa

```
---
 - hosts: local
   become: yes
   become_user: root
   tasks:
    - name: instalación ssh barney
      lxc_container:
        name: lisa
        container_command: yum update | yum install openssh-server sshpass sudo -y

    - name: creación de usuario
      lxc_container:
        name: lisa
        container_command: useradd usuario -s /bin/bash -m

    - name: configuración del usuario
      lxc_container:
        name: lisa
        container_command: |
          echo -e "usuario\nusuario" | passwd usuario

    - name: configuración de sudo
      lxc_container:
        name: lisa
        container_command: |
          echo "usuario ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers
```

Con esos playbook podríamos preparar una configuración de las máquinas tras concluir la instalación, para ello solo tendríamos que usar la siguiente línea al final del playbook donde desplegábamos el contenedor:

```
 include: configurar-lisa.yml
```

De no haberlo puesto en esos playbook previamente podríamos ejecutarlo con una sencilla receta:

```
---
 - include: configurar-homer.yml
 - include: configurar-barney.yml
 - include: configurar-lisa.yml
```

Podemos ejecutarlo como los playbook anteriores:

```
root@nodo-1:/etc/ansible# ansible-playbook configurar-contenedores.yml -u root -k
SSH password:

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [127.0.0.1]

TASK [instalación ssh homer] ***************************************************
changed: [127.0.0.1]

TASK [creación de usuario] *****************************************************
changed: [127.0.0.1]

TASK [configuración del usuario] ***********************************************
changed: [127.0.0.1]

TASK [configuración de sudo] ***************************************************
changed: [127.0.0.1]

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [127.0.0.1]

TASK [instalación ssh barney] **************************************************
changed: [127.0.0.1]

TASK [creación de usuario] *****************************************************
changed: [127.0.0.1]

TASK [configuración del usuario] ***********************************************
changed: [127.0.0.1]

TASK [configuración de sudo] ***************************************************
changed: [127.0.0.1]

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [127.0.0.1]

TASK [instalación ssh lisa] ****************************************************
changed: [127.0.0.1]

TASK [creación de usuario] *****************************************************
changed: [127.0.0.1]

TASK [configuración del usuario] ***********************************************
changed: [127.0.0.1]

TASK [configuración de sudo] ***************************************************
changed: [127.0.0.1]

PLAY RECAP *********************************************************************
127.0.0.1                  : ok=15   changed=12   unreachable=0    failed=0
```

Podemos probar la conexión de ansible con el cliente de forma “artesanal”:

```
root@nodo-1:/etc/ansible# ansible  barney -m ping  -u usuario -k
SSH password:
10.0.4.185 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

O crear un playbook con el que probar los tres contenedores:

```
---
 - hosts: lxc
   become: yes
   become_user: root

   tasks:
    - name: prueba de ping + ssh
      ping:
```

```
root@nodo-1:/etc/ansible# ansible-playbook prueba-ssh.yml -u usuario -k
SSH password:

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [10.0.4.185]
ok: [10.0.4.55]
ok: [10.0.4.163]

TASK [prueba de ping + ssh] ****************************************************
ok: [10.0.4.55]
ok: [10.0.4.185]
ok: [10.0.4.163]

PLAY RECAP *********************************************************************
10.0.4.163                 : ok=2    changed=0    unreachable=0    failed=0
10.0.4.185                 : ok=2    changed=0    unreachable=0    failed=0
10.0.4.55                  : ok=2    changed=0    unreachable=0    failed=0
```

Tras dejar lista la conexión entre ansible y las máquinas podremos comenzar a instalar los servicios.

Instalación de servicios

En este punto, puede que nos interese aprender a usar los **roles** , lo cual nos permitirá mantener un orden ya que la cantidad de playbooks hasta ahora es alta y aun quedan por crear, para ello tendremos que crear un arbol de directorios como el siguiente:

```
roles
├── bd
│   ├── files
│   │   └── my.cnf
│   └── tasks
│       └── main.yml
├── dns
│   ├── files
│   │   ├── db.10.0.3
│   │   ├── db.10.0.4
│   │   ├── db.arodriguez
│   │   ├── db.named.conf.local
│   │   └── ips.yml
│   └── tasks
│       └── main.yml
└── httpd
    ├── files
    │   └── index.html
    └── tasks
        └── main.yml
```

Dentro de task pondriamos un fichero llamdo **main.yml** , el cual tiene las tareas que va a realizar cuando se llame a ese rol.

Un ejemplo de archivo main sería el siguiente:

```
---
 - name: instalación de httpd (apache)
   yum: name=httpd state=present
 - name: instalación de php
   yum: name=php state=present

 - name: Copiar la página de presentación
   copy: src=index.html dest=/var/www/html/index.html
```

Instala el servidor web, con un servidor para aplicaciones php y copia una página previamente hecha a /var/www/.

El playbook que llamaría a ese rol sería algo así:

```
---
 - hosts: lisa
     become: yes
   become_user: root
   roles:
    - httpd
```

Servidor web

Empezamos con el servidor web en lisa, el mas sencillo de los que tenemos que instalar.

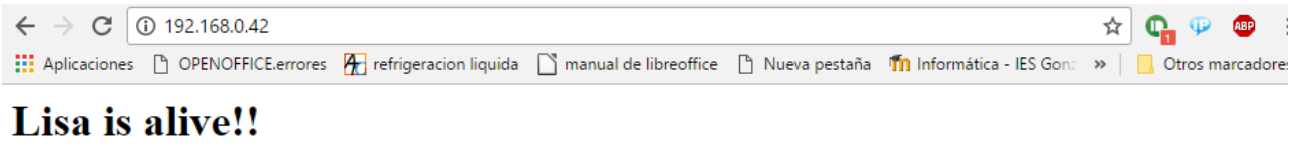
Aprovechamos el playbook que ya tenemos y lo lanzamos:

```
root@nodo-1:/etc/ansible# ansible-playbook httpd.yml -u usuario -k --extra-vars "ansible_sudo_pass=usuario"
SSH password:

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [10.0.4.163]

TASK [httpd : instalación de httpd (apache)] ***********************************
ok: [10.0.4.163]

TASK [httpd : instalación de php] **********************************************

changed: [10.0.4.163]

TASK [httpd : Copiar la página de presentación] ********************************
changed: [10.0.4.163]

PLAY RECAP *********************************************************************
10.0.4.163                 : ok=4    changed=2    unreachable=0    failed=0
```

Ahora tendremos que abrir puertos para poder ver el contenido web:

```
iptables -t nat -A PREROUTING -p tcp --dport 8080 -j DNAT --to-destination
10.0.4.163:80
```

Ya podremos acceder a la página :



Lisa is alive!!

DNS

Para el DNS necesitaremos instalar bind9 y configurarlo, esto lo haremos a base de archivos preconfigurados.

Comenzamos con el playbook:

```
---
 - hosts: barney
   become: yes
   become_user: root
   roles:
    - dns
```

Y creamos un main.yml para la instalación y configuración:

```
---
 - name: instalaciÃ³n de dns
   apt: name=bind9 state=present

 - name: configurando zona directa
   template:
     src: roles/dns/files/db.arodriguez
     dest: /var/cache/bind/db.arodriguez
     owner: root
     group: root
     mode: 0644

 - name: configurando conf.local
   template:
     src: roles/dns/files/db.named.conf.local
     dest: /etc/bind/named.conf.local
     owner: root
     group: root
     mode: 0644


 - name: configurando zona inversa
   template:
     src: roles/dns/files/db.10.0.4
     dest: /var/cache/bind/db.10.0.4
     owner: root
     group: root
     mode: 0644

 - name: reinicio de bind
   service:
     name: bind9
     state: restarted
```

Tendremos que tener también plantillas de los archivos con los que configuraremos el servidor; zona directa, inversa y el archivo de configuración principal:

Zona directa:

```
$TTL    86400
@       IN      SOA     barney.arodriguez.gonzalonazareno.org.
doommaster.arodriguez.gonzalonazareno.org. (
                              1         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                          86400 )       ; Negative Cache TTL

      IN      NS      barney.arodriguez.gonzalonazareno.org.



barney.arodriguez.gonzalonazareno.org.        IN      A       10.0.4.185
homer.arodriguez.gonzalonazareno.org.         IN      A       10.0.4.55
lisa.arodriguez.gonzalonazareno.org.          IN      A       10.0.4.163


www                     IN      CNAME   lisa
```

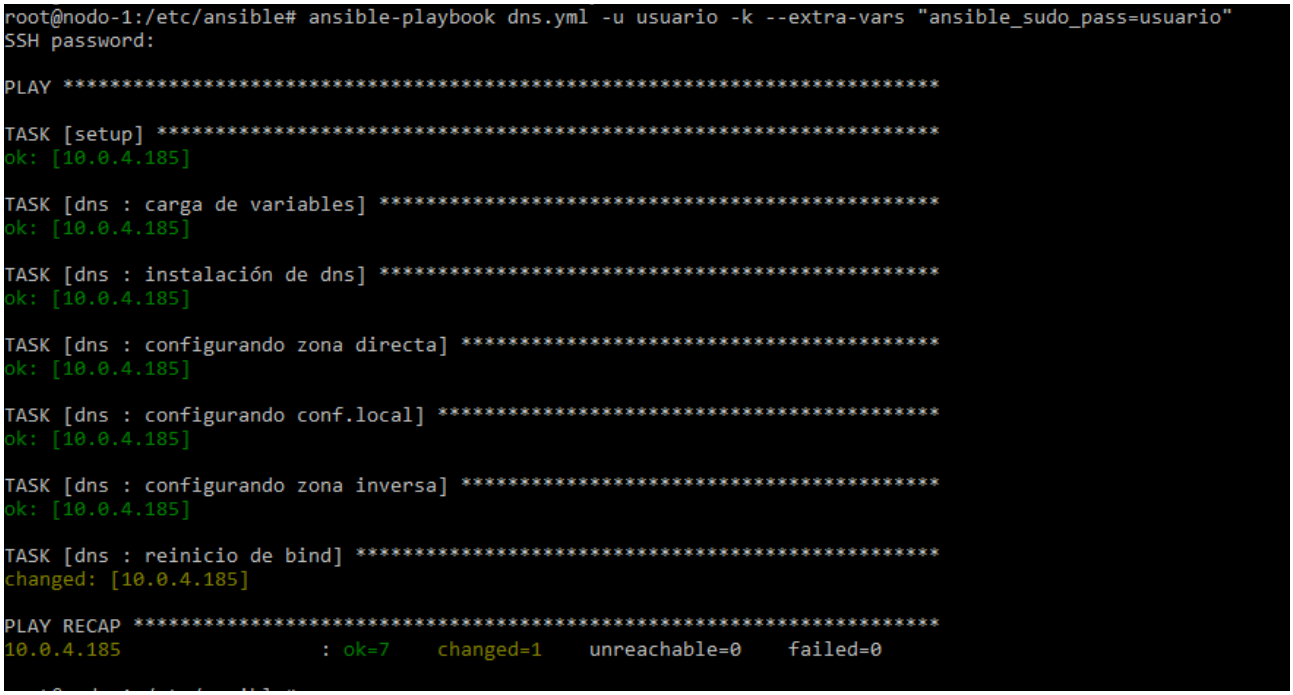
Zona inversa:

```
$TTL    86400
@       IN      SOA    barney.arodriguez.gonzalonazareno.org.
doommaster.arodriguez.gonzalonazareno.org. (
                              1         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                          86400 )       ; Negative Cache TTL

     IN      NS      barney.arodriguez.gonzalonazareno.org.

185       IN      PTR     barney.arodriguez.gonzalonazareno.org.
55        IN      PTR     homer.arodriguez.gonzalonazareno.org.
163       IN      PTR     lisa.arodriguez.gonzalonazareno.org.
```

Named.conf.local:

```
zone "arodriguez.gonzalonazareno.org"{
 type master;
 file "db.arodriguez";
# allow-query {10.0.4.0/24;};
};

zone "4.0.10.in-addr.arpa" {
 type master;
 file "db.10.0.4";
# allow-query {10.0.4.0/24;};
};
```

Y lo lanzamos así:



Podemos usar **dig** del paquete **dnsutils** para ver que funciona el servidor dns:

```
root@nodo-1:/etc/ansible# dig a  @10.0.4.185 barney.arodriguez.gonzalonazareno.org.

; <<>> DiG 9.10.3-P4-Ubuntu <<>> a @10.0.4.185 barney.arodriguez.gonzalonazareno.org.
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19323
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;barney.arodriguez.gonzalonazareno.org. IN A

;; ANSWER SECTION:
barney.arodriguez.gonzalonazareno.org. 86400 IN A 10.0.4.185

;; AUTHORITY SECTION:
arodriguez.gonzalonazareno.org. 86400 IN NS     barney.arodriguez.gonzalonazareno.org.

;; Query time: 0 msec
;; SERVER: 10.0.4.185#53(10.0.4.185)
;; WHEN: Wed Jun 14 16:13:39 CEST 2017
;; MSG SIZE  rcvd: 96

root@nodo-1:/etc/ansible#
```

```
root@nodo-1:/etc/ansible# dig a  @10.0.4.185 www.arodriguez.gonzalonazareno.org.

; <<>> DiG 9.10.3-P4-Ubuntu <<>> a @10.0.4.185 www.arodriguez.gonzalonazareno.org.
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14820
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.arodriguez.gonzalonazareno.org. IN A

;; ANSWER SECTION:
www.arodriguez.gonzalonazareno.org. 86400 IN CNAME lisa.arodriguez.gonzalonazareno.org.
lisa.arodriguez.gonzalonazareno.org. 86400 IN A 10.0.4.163

;; AUTHORITY SECTION:
arodriguez.gonzalonazareno.org. 86400 IN NS     barney.arodriguez.gonzalonazareno.org.

;; ADDITIONAL SECTION:
barney.arodriguez.gonzalonazareno.org. 86400 IN A 10.0.4.185

;; Query time: 1 msec
;; SERVER: 10.0.4.185#53(10.0.4.185)
;; WHEN: Wed Jun 14 16:15:51 CEST 2017
;; MSG SIZE  rcvd: 135

root@nodo-1:/etc/ansible#
```

También podemos probar con la resolución inversa:

```
root@nodo-1:/etc/ansible# dig -x 10.0.4.185 @10.0.4.185

; <<>> DiG 9.10.3-P4-Ubuntu <<>> -x 10.0.4.185 @10.0.4.185
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57786
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;185.4.0.10.in-addr.arpa.       IN      PTR

;; ANSWER SECTION:
185.4.0.10.in-addr.arpa. 86400  IN      PTR     barney.arodriguez.gonzalonazareno.org.

;; AUTHORITY SECTION:
4.0.10.in-addr.arpa.    86400   IN      NS      barney.arodriguez.gonzalonazareno.org.

;; ADDITIONAL SECTION:
barney.arodriguez.gonzalonazareno.org. 86400 IN A 10.0.4.185

;; Query time: 0 msec
;; SERVER: 10.0.4.185#53(10.0.4.185)
;; WHEN: Wed Jun 14 16:23:30 CEST 2017
;; MSG SIZE  rcvd: 133

root@nodo-1:/etc/ansible#
```

```
root@nodo-1:/etc/ansible# dig -x 10.0.4.55 @10.0.4.185

; <<>> DiG 9.10.3-P4-Ubuntu <<>> -x 10.0.4.55 @10.0.4.185
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15894
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;55.4.0.10.in-addr.arpa.                IN      PTR

;; ANSWER SECTION:
55.4.0.10.in-addr.arpa. 86400   IN      PTR     homer.arodriguez.gonzalonazareno.org.

;; AUTHORITY SECTION:
4.0.10.in-addr.arpa.    86400   IN      NS      barney.arodriguez.gonzalonazareno.org.

;; ADDITIONAL SECTION:
barney.arodriguez.gonzalonazareno.org. 86400 IN A 10.0.4.185

;; Query time: 0 msec
;; SERVER: 10.0.4.185#53(10.0.4.185)
;; WHEN: Wed Jun 14 16:24:15 CEST 2017
;; MSG SIZE  rcvd: 138

root@nodo-1:/etc/ansible#
```

BD

Como base de datos instalaremos **MySQL server** sobre homer, para ello creamos el siguiente playbook:

```
---
 - hosts: homer
   become: yes
   become_user: root
   roles:
    - bd
```

Su archivo **main.yml** sería algo así:

```
---
 - name: instalacin de python-mysqlb
   apt: name=python-mysqldb state=present
 - name: instalacin de MySQL server
   apt: name=mysql-server state=present
 - name: creacin de usuario remoto
   mysql_user: user="usuario" host="%" password=usuario priv=*.*:ALL,GRANT
 - name: Copiar la configuracion
   copy: src=my.cnf dest=/etc/mysql/my.cnf

 - name: reinicio de mysql
   service:
     name: mysql
     state: restarted
```

Y lo lanzamos como los anteriores:

```
root@nodo-1:/etc/ansible# nsible-playbook mysql.yml -u usuario -k --extra-vars "ansible_sudo_pass=usuario"
SSH password:

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [10.0.4.55]

TASK [bd : instalacin de python-mysqlb] ****************************************
ok: [10.0.4.55]

TASK [bd : instalacin de MySQL server] *****************************************
ok: [10.0.4.55]

TASK [bd : creacin de usuario remoto] ******************************************
ok: [10.0.4.55]

TASK [bd : Copiar la configuracion] ********************************************
changed: [10.0.4.55]

TASK [bd : reinicio de mysql] **************************************************
changed: [10.0.4.55]

PLAY RECAP *********************************************************************
10.0.4.55                  : ok=6    changed=2    unreachable=0    failed=0
```

Y a podremos conectarnos a la base de datos con el usuario que hemos creado:

```
root@nodo-1:/etc/ansible# mysql -u usuario -p -h 10.0.4.55
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 38
Server version: 5.5.55-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Actualización de contenedores

Una tarea que podríamos lanzar de forma grupal entre los contenedores es una actualización de paquetes, esta tarea podríamos realizarla con un playbook simple.

```
---
 - include: update-homer.yml
 - include: update-barney.yml
 - include: update-lisa.yml
```

```
---
 - hosts: homer
     become: yes
   become_user: root
   tasks:
    - name: Actualizacin de homer
      apt:
        name: '*'
        state: latest
```

```
---
 - hosts: barney
   become: yes
   become_user: root
   tasks:
    - name: Actualización de barney
      apt:
        name: '*'
        state: latest
```

```
---
 - hosts: lisa
   become: yes
   become_user: root
   tasks:
    - name: Actualización de lisa
      yum:
        name: '*'
        state: latest
```

Ejecución del playbook global

Para facilitar las cosas de todo aquel que quiera utilizar el escenario he creado un playbook que llama al resto en orden, de tal forma que realice la configuración e instalación máquinas y servicios partiendo de la base de tener un host configurado.

```
---
 - include: homer.yml
 - include: lisa.yml
 - include: barney.yml
 - include: configurar-contenedores.yml
 - include: httpd.yml
 - include: mysql.yml
 - include: dns.yml
```

Puede verse la ejecución en el video adjuntado, no pude meterlo en una captura debido al largo del contenidor.

Demo de Lisa

La demostración con la máquina lisa es la principal que llevaré a cabo durante la exposición, debido a que es la más rápida de desplegar .

Para el despliegue he creado un playbook general paralelo al usado en el despliegue del escenario completo.

La demostración cuenta con el despliegue completo de la máquina, creación del contenedor, preparación para el acceso remoto e instalación del servicio httpd(apache).

Si quedase mas tiempo precedería a mostrar también el despliegue de otra de las máquinas.