Centros de Datos y de Provisión de Servicios

Curso 2021-22

PRACTICA CREATIVA 1:

DESARROLLO DE UN SCRIPT PARA LA CREACIÓN AUTOMÁTICA DEL ESCENARIO DEL BALANCEADOR DE LA PRÁCTICA 2

Última actualización: 18 de noviembre de 2021

Objetivos

La práctica final del primer parcial consistirá en el desarrollo de un script en Python de que automatice parcialmente la creación del escenario de pruebas del balanceador de tráfico de la segunda parte de la práctica 2 (Figura 1). Opcionalmente se podrá trabajar con otros escenarios virtuales de complejidad similar o que utilicen contenedores LXC o docker en vez máquinas virtuales KVM. Si va a trabajar sobre un escenario alternativo, consulte previamente a los profesores de la asignatura.

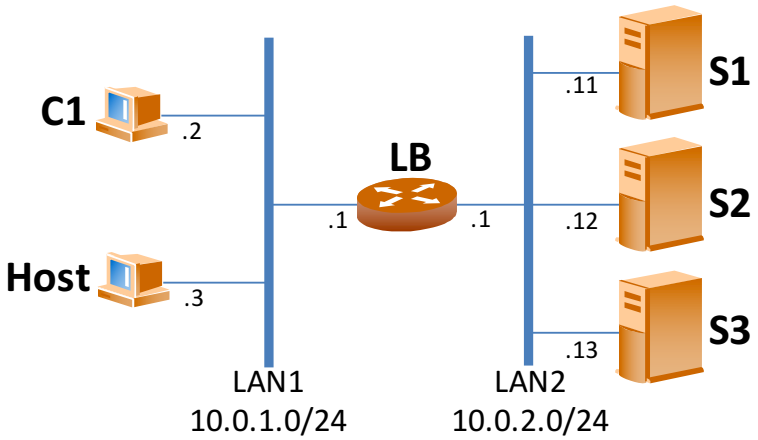

Figura 1: escenario virtual P2b

Requisitos funcionales mínimos del script

El script, de nombre '**auto-p2**', deberá ejecutarse en un directorio en el que inicialmente solo esté el fichero con la imagen base que utilizarán las máquinas virtuales (MV) del escenario (*cdps-vm-base-pc1.qcow2*) y la plantilla de MVs (*plantilla-vm-pc1.xml*), disponibles ambos en el directorio */lab/cdps/pc1* del laboratorio y también por web en: https://idefix.dit.upm.es/download/cdps/pc1/. El script no debe copiar los ficheros; éstos deben estar ya copiados al directorio de trabajo.

El script *auto-p2* se ejecutará pasándole un parámetro obligatorio que definirá la operación a realizar:

```
#  auto-p2 <orden> <otros_parámetros>
```

donde el parámetro *<orden>* puede tomar los valores siguientes:

- ***prepare***, para crear los ficheros .qcow2 de diferencias y los de especificación en XML de cada MV, así como los bridges virtuales que soportan las LAN del escenario.
- ***launch***, para arrancar las máquinas virtuales y mostrar su consola.
- ***stop***, para parar las máquinas virtuales (sin liberarlas).
- ***release***, para liberar el escenario, borrando todos los ficheros creados.

El número de servidores web a arrancar deberá ser configurable (de 1 a 5). Este número se deberá especificar mediante un segundo parámetro de la línea de comandos que será opcional: si se proporciona, se tomará el valor especificado; y si no, se asignará el valor por defecto de 3. El script deberá comprobar que el valor del número de servidores es correcto, dando un error en caso contrario.

El valor del número de servidores sólo se especificará con el comando "prepare". Ese número se almacenará en un fichero de configuración en el directorio de trabajo (auto-p2.json) y el resto de los comandos (launch, stop, release) lo leerán de ese fichero. El formato con el que se almacenará será JSON:

```
{
    "num_serv": 2
}
```

El script deberá realizar la configuración mínima de las MVs antes de arrancarlas, modificando los ficheros */etc/hostname* y */etc/network/interfaces* de cada MV, incluyendo en ellos el nombre de la máquina y la configuración de red respectivamente. También deberá configurar el balanceador para que funcione como router.

Las imágenes de las MVs deben crearse mediante ficheros de diferencias (qcow2) con respecto de la imagen base.

El script debe ser "no interactivo", esto es, cualquier parámetro necesario se debe pasar por la línea de comandos; el script no debe preguntar nada interactivamente.

Partes opcionales

Como funcionalidades adicionales se propone incluir:

- La monitorización del escenario mediante, por ejemplo, una orden adicional (monitor) que presente el estado de todas las máquinas virtuales del escenario. Esta orden se puede ejecutar con el comando *watch* para monitorizar periódicamente el escenario.
- La configuración y arranque automático del balanceador de tráfico HAproxy, de manera que cuando se arranque la MV esté disponible automáticamente el servicio de balanceo de tráfico entre servidores web.
- El acceso al interfaz de gestión de HAproxy a través del web o mediante comandos (ver https://www.haproxy.com/blog/dynamic-configuration-haproxy-runtime-api/).

- Funcionalidad para parar y/o arrancar MVs individualmente.
- Otras funcionalidades a proponer por el alumno.

En la evaluación de la práctica se valorará la generalidad de los scripts y el grado de automatización alcanzado, así como las partes opcionales implementadas.

Recomendaciones

- Dado que el script tiene que ejecutar múltiples comandos, se recomienda probar estos comandos a mano desde la línea de comandos y solo integrarlos y probarlos en el script cuando se esté seguro de que funcionan. Asimismo, se recomienda dividir las funcionalidades del script en partes y probarlas por separado en scripts de prueba (p.e.: el paso de parámetros por la línea de comandos, la modificación de los ficheros XML, el montaje de las imágenes de las máquinas virtuales y la modificación de ficheros, etc.). Una vez que cada parte esté probada, se puede integrar el código de la prueba como una función en el script principal.
- Los mensajes que se quiera que el programa siempre muestre en pantalla se pueden imprimir utilizando la función print/printf. Por el contrario, para la depuración del script es útil que el programa muestre trazas de información en pantalla que se puedan eliminar fácilmente una vez se haya depurado el programa. Se recomienda utilizar el módulo "logging" de Python. Por ejemplo, si se ejecuta este script:

```
#!/usr/bin/python3

import logging

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger('auto-p2')

logger.debug('mensaje debug1')
logger.debug('mensaje debug2')
```

  aparecerán los mensajes de depuración en pantalla, pero si se sustituye 'DEBUG' por 'INFO' desaparecerán. Más información en https://goo.gl/sSDT8Y.
- Para la gestión de las máquinas virtuales (arranque, parada, etc.) se recomienda utilizar el comando "sudo virsh define|start|shutdown|destroy" ya utilizado en prácticas anteriores. Consulte en los tutoriales de Python cómo ejecutar comandos externos desde un script. Recuerde que para las máquinas virtuales sean persistentes debe utilizar los comandos "virsh define" y "virsh start".
- Para crear y modificar los ficheros XML de definición de las MVs, se utilizará alguna de las librerías disponibles en Python para leer y modificar ficheros XML. Se recomienda utilizar la librería *lxml.etree* (http://lxml.de) disponible en el laboratorio y sobre la que se adjunta un ejemplo de uso al final de este enunciado.
- Para leer y escribir en formato JSON en el fichero de configuración se debe utilizar el modulo 'json' de Python.
- El script debe mostrar las consolas de las máquinas virtuales del escenario cuando éste se arranque. Para mostrarlas existen dos métodos según se quiera mostrar la consola gráfica o textual:

- Consola gráfica: ejecutar "**virt-viewer <nombre_mv>"**
- Consola textual: arrancar un nuevo terminal "**xterm**" o "**gnome-terminal**" con la opción -e para que ejecute el comando "**sudo virsh console <nombre_mv>**". Por ejemplo:

```
xterm -e "sudo virsh console s1"
```

Se recomienda arrancar la consola textual. Tenga en cuenta que los comandos anteriores deben ser ejecutarlos en *background* para que el script no se quede detenido cuando se ejecuta el comando (añada un "&" al final del comando para lograrlo).

- El comando "stop" debe detener las MVs de forma ordenada usando "virsh shutdown" y deberá conservar el contenido de sus imágenes, de forma que toda modificación realizada en las MVs se conserve si se paran y arrancan de nuevo.
- El comando "release" debe liberar las MVs utilizando "virsh destroy" y borrar todos los ficheros creados durante el arranque.
- Para la parte de configuración del *hostname*, los interfaces de red de las máquinas virtuales y la configuración como router del balanceador, se recomienda seguir el procedimiento siguiente:
  - Crear los ficheros de configuración de las máquinas virtuales en el host en un directorio temporal. La configuración de la máquina virtual se debe realizar modificando los ficheros /etc/hostname y /etc/network/interfaces (consulte en Internet el formato de este último). No hace falta conservar el contenido original de esos ficheros; lo más sencillo es que el script escriba su contenido completo.
  - Copiar los ficheros de configuración a las máquinas virtuales utilizando el comando "virt-copy-in". Por ejemplo, para copiar el fichero "interfaces" al directorio "/etc/network" de imagen s1.qcow2 utilizada en la MV s1:

```
sudo virt-copy-in -a s1.qcow2 interfaces /etc/network
```

  - IMPORTANTE: para modificar ficheros en la imagen de una máquina virtual es necesario que esta esté parada.
  - Para comprobar que un fichero se ha modificado correctamente, se puede utilizar el comando "virt-cat", que nos permite ver el contenido del fichero almacenado en la imagen. Por ejemplo:

```
sudo virt-cat -a s1.qcow2 /etc/network/interfaces
```

  - También es posible copiar ficheros desde la imagen de la máquina virtual al host mediante el comando "virt-copy-out". Por ejemplo, para copiar el fichero /etc/network/interfaces de la imagen al directorio actual:

```
sudo virt-copy-out -a s1.qcow2 /etc/network/interfaces .
```

  - Finalmente, para ver los ficheros que hay dentro de una imagen se puede usar el comando "virt-ls". Por ejemplo, para ver el contenido del directorio /etc de una imagen:

```
sudo virt-ls -a s1.qcow2 -l /etc
```

- Para que el balanceador de tráfico funcione como router al arrancar, se recomienda editar el fichero /etc/sysctl.conf tal como se describe en http://www.ducea.com/2006/08/01/how-to-enable-ip-forwarding-in-linux/. Para ello puede utilizar el siguiente comando:

```
sudo virt-edit -a lb.qcow2 /etc/sysctl.conf \
   -e 's/#net.ipv4.ip_forward=1/net.ipv4.ip_forward=1/'
```

- Para la parte opcional de monitorización del escenario se pueden utilizar algunos de los comandos que proporciona *virsh* para obtener información sobre las máquinas virtuales, por ejemplo, los comandos *domstate*, *dominfo*, *cpu-stats*, etc (consultar el manual de virsh mediante "man virsh"). También se puede utilizar un ping desde el host para chequear la conectividad con las MVs.

- Para la parte opcional de arranque del balanceador de tráfico durante el arranque de LB, se recomienda utilizar el fichero /etc/rc.local. Recuerde que la imagen de las máquinas virtuales utilizada tiene instalado el servidor apache2. Para que el HAproxy pueda ejecutarse, es necesario parar el servidor apache2 ("service apache2 stop").

Ejemplo de uso de la librería lxml

Fichero XML de ejemplo:

```
<objeto tipo='A'>
  <nombre>objeto1</nombre>
  <parte1>
    <nombre>p1</nombre>
  </parte1>
  <parte2>
    <nombre>p2</nombre>
  </parte2>
  <parte3>
    <nombre>p2</nombre>
    <cacho nombre='c1'>
      <cachito nombre='c11'/>
    </cacho>
    <cacho nombre='c2'>
    </cacho>
  </parte3>
</objeto>
```

Programa de ejemplo que procesa y modifica el fichero XML anterior:

```
#!/usr/bin/python3

from lxml import etree

def pause():
    p = input("Press <ENTER> key to continue...")

# Cargamos el fichero xml
tree = etree.parse('test.xml')

# Lo imprimimos en pantalla
print(etree.tounicode(tree, pretty_print=True))
pause()

# Obtenemos el nodo raiz e imprimimos su nombre y el valor del atributo 'tipo'
root = tree.getroot()
print(root.tag)
print(root.get("tipo"))
pause()

# Buscamos la etiqueta 'nombre' imprimimos su valor y luego lo cambiamos
name = root.find("nombre")
print(name.text)
name.text = 'kiko'
print(name.text)
pause()

# Buscamos la etiqueta 'nombre' bajo el nodo 'parte3', imprimimos su valor y lo cambiamos
nombre_parte3=root.find("./parte3/nombre")
print(nombre_parte3.text)
nombre_parte3.text='veneno'
print(nombre_parte3.text)
pause()

# Buscamos el nodo 'cacho' bajo 'parte3' con nombre 'c1', imprimimos su valor y lo cambiamos
cachito=root.find("./parte3/cacho[@nombre='c1']/cachito")
print(cachito.get("nombre"))
cachito.set("nombre", "de hierro y cromo")
print(cachito.get("nombre"))
pause()

# Imprimimos el xml con todos los cambios realizados
print(etree.tounicode(tree, pretty_print=True))
```