PRACTICA FINAL PRIMER PARCIAL:

SCRIPT PARA LA CREACIÓN AUTOMÁTICA DEL ESCENARIO DEL BALANCEADOR DE LA P3

Última actualización: 10 de noviembre de 2015

La práctica final del primer parcial que será evaluada en el examen oral consistirá en el desarrollo de un script en Python (pfinalp1) que automatice en la medida de lo posible la creación del escenario de pruebas del balanceador de tráfico de la segunda parte de la práctica 3 (opcionalmente se podrá trabajar con otros escenarios virtuales de complejidad similar previo acuerdo con los profesores).

El script deberá partir del fichero con la imagen base del la MV (cdps-vm-base-p3.qcow2) y de la plantilla de MVs (plantilla-vm-p3.xml) disponibles en el directorio /mnt/vnx/repo del laboratorio, que pueden estar ya copiados al directorio de trabajo (no es necesario que el script los copie).

El script *pfinalp1* se ejecutará pasándole un parámetro que definirá la operación a realizar:

 # pfinalp1 <orden>

- *<orden>=create*, para crear los ficheros .qcow2 de diferencias y de especificación XML de cada MV, así como los bridges que soportan las LAN del escenario.
- *<orden>=start*, para arrancar las máquinas virtuales y mostrar su consola.
- *<orden>=stop*, para parar las máquinas virtuales.
- *<orden>=destroy*, para liberar el escenario, borrando todos los ficheros creados.

El número de servidores web a arrancar deberá ser configurable (de 1 a 5). Se podrá especificar mediante un segundo parámetro o especificándolo en un fichero de configuración que lea el script.

Como funcionalidades adicionales se propone incluir:

- La configuración del *hostname* y de las direcciones IP de los interfaces de red de las máquinas virtuales del escenario.
- La monitorización del escenario mediante, por ejemplo, una orden adicional (monitor) que presente el estado de todas las máquinas virtuales del escenario.

- Otras funcionalidades a definir 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

- Para la gestión de las máquinas virtuales (arranque, parada, etc.) se utilizará el commando "sudo virsh create|shutdown …" ya utilizado en la P3. Consulte en los tutoriales de Python cómo ejecutar comandos externos desde un script.
- Para crear y modificar los ficheros XML de definición de las MVs, se utilizará alguna de las librerías disponibles en Python para 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..
- El script debe mostrar las consolas de las máquinas virtuales del escenario. 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>**"

  Tenga en cuenta que los comandos anteriores hay que ejecutarlos en *background* para que el script no se detenga.
- A la hora de detener las máquinas virtuales deberán pararse de forma ordenada usando "virsh shutdown ...". Opcionalmente se podrá incluir un parámetro adicional o una nueva orden para parar las máquinas de forma destructiva utilizando "virsh destroy ...".
- Para la parte opcional de configuración del *hostname* y la red de las maquinas virtuales, se puede seguir el procedimiento siguiente:
  - Antes de arrancar cada máquina virtual se monta su sistema de ficheros en un directorio del host para poder modificarlo directamente desde el host:

    mkdir mnt
    sudo vnx_mount_rootfs -s -r s1.qcow2 mnt
  - Una vez ejecutados los comandos anteriores, los ficheros de la maquina virtual estarán accesibles en el directorio *mnt/*.
  - La configuración de la maquina virtual se debe realizar modificando los ficheros /etc/hostname y /etc/network/interfaces (consulte en Internet el formato de este último).

- Terminadas las modificaciones, se debe desmontar la imagen de la maquina virtual con:

  sudo vnx_mount_rootfs -u mnt

Nota: en caso de realizar esta parte en su propio ordenador, debe instalar manualmente el comando **vnx_mount_rootfs**:

wget idefix.dit.upm.es/download/cdps/vnx_mount_rootfs
mv vnx_mount_rootfs /usr/bin

- 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").

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>p3</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/python

from lxml import etree

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

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

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

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

# Buscamos la etiqueta 'nombre' bajo el nodo 'parte3' e imprimimos su valor y luego lo
cambiamos
nombre_parte3=root.find("./parte3/nombre")
```

```
print(nombre_parte3.text)
nombre_parte3.text='veneno'
print(nombre_parte3.text)

# Buscamos el nodo 'cacho' bajo 'parte3' con nombre 'c1' e  imprimimos su valor y
luego 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"))

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