ANEXO 0: TESTBED

Índice general

Índice de códigos fuente

TODO: actualizar a Debian 7 Wheezy las localizaciones de las imágenes.

Capítulo 1

Entorno de referencia ("testbed")

Contents

Llamamos “testbed”[1] a un par de imágenes con Debian 6 "Squeeze" preinstalado. El testbed permitirá que las condiciones en que se despliegue este proyecto sean idénticas a en las que (nosotros) desarrollamos este documento: debido a su complejidad, cualquier diferencia entre el equipo de despliegue y nuestro testbed puede dar lugar a errores difíciles de depurar, por lo que este documento no podría ser útil si no se ofrece este entorno de referencia.

Con ello, el despliegue ideal contemplaría seguir estrechamente todo lo que se indicará en posteriores capítulos en el testbed que se describe ahora y, entonces, teniendo a ése como referencia, inténtese hacerlo en los equipos en producción. Es más, el testbed propuesto es usado utilizando técnicas de virtualización[2] tal que se pueda desplegar, a falta de otros recursos, en las propias máquinas que albergarán su versión en producción (con el suficiente aislamiento como para ir desplegando el proyecto en el uno paralelamente al otro).

[1] Del inglés “mesa de pruebas”.

[2] Las imágenes se han arrancado utilizando ténicas de virtualización (Qemu), y de contextualización (contenedores LXC). Sin embargo, como se expondrá en el apartado para las conclusiones del testbed, no es posible usar el cliente AFS (para nuestro sistema de archivos distribuído AFS) con LXC. El cliente AFS está implementado como un módulo en el kernel, y no pueden usarlo todos los sistemas a la vez (sólo puede cargarse y usarse una vez: o por el host, o por el primer invitado, o por el segundo invitado). Ésto haría que, usando LXC, toda la parte del proyecto que dependa del acceso al sistema de archivos AFS, sólo podría realizarse en un invitado, con lo que el despliegue queda, aunque bien esbozado, incompleto. Por ello hemos decidido no hacer mención alguna a LXC más que una valoración en el apartado sobre las conclusiones.

1.1. Imágenes Debian gnu/linux 6.0 "Debian Squeeze"; overlays

A continuación se dan instrucciones para obtener las imágenes y, más adelante, para arrancarlas. Se explicará como si el SO host, aquel donde arrancamos las máquinas virtuales, fuese igualmente un sistema Debian gnu/linux. Ésto puede no ser así, de forma que en aquel caso remitimos a http://wiki.qemu.org/Links

```
apt-get install qemu
mkdir testbed
cd testbed
wget \
 http://people.debian.org/~aurel32/qemu/i386/debian_squeeze_i386_standard.qcow2
if ! which qemu-img; then apt-get install qemu-utils; fi
qemu-img info debian_squeeze_i386_standard.qcow2
```

Para crear la imagen overlay de la máquina que llamaremos dklab1:

```
qemu-img create -b debian_squeeze_i386_standard.qcow2 -f qcow2 dklab1.ovl
```

Para crear la imagen overlay de la máquina que llamaremos dklab2:

```
qemu-img create -b debian_squeeze_i386_standard.qcow2 -f qcow2 dklab2.ovl
```

Descripción de los overlays:

```
qemu-img info dklab1.ovl
qemu-img info dklab2.ovl
cd ..
```

1.1.1. Anotación: hacer a los overlays disponibles como si fueran unidades físicas

Es posible hacer las imágenes disponibles como dispositivos de disco Convenientemente, se hace aquí esta anotación porque facilitaría:

- Hacer que otros sistemas de virtualización que no conozcan el formato de imagen qcow2, puedan cargar las imágenes como si fuesen un dispositivo de disco real. De esta forma, si el lector está habituado a otros sistemas de virtualización, podría usarlos en lugar de los propuestos aquí.
- Adicionalmente, nos permitiría montar las imágenes y explorarlas:

```
mkdir -p /var/lib/lxc/dklab1/rootfs /var/lib/lxc/dklab2/rootfs
#... o cualquier otros puntos de montaje.

modprobe nbd max_part=8

## Montaje:

N=0 # Nota: cuando ha habido necesidad de desmontar y volver a montar, hemos
    #       tenido que incrementar en 2 el valor usado de N la vez anterior
    #       para evitar un error desconocido.
img=testbed/dklab1.ovl
dir=/var/lib/lxc/dklab1/rootfs
dev=/dev/nbd${N}

qemu-nbd --connect=${dev} ${img}
sleep 3
fdisk -l ${dev}
mount ${dev}p1 ${dir}

let N=N+1
img=testbed/dklab2.ovl
dir=/var/lib/lxc/dklab2/rootfs
dev=/dev/nbd${N}

qemu-nbd --connect=${dev} ${img}
sleep 3
fdisk -l ${dev}
mount ${dev}p1 ${dir}

ls ${dir}
```

```
## Desmontaje:
 N=0
 umount /var/lib/lxc/dklab1/rootfs
 umount /var/lib/lxc/dklab2/rootfs
 qemu-nbd --disconnect /dev/nbd${N}; let N=N-1
 qemu-nbd --disconnect /dev/nbd${N}
 modprobe -v -r nbd
```

1.2. Password cuenta root

Sea cual sea el método de virtualización con que logremos arrancar las imágenes, el proceso de carga terminará con el programa login pidiéndonos el nombre de usuario y contraseña. Durante el proyecto usaremos la cuenta de root, sabiendo que:

El password para el usuario root es root igualmente.

1.3. Uso de los overlays como Qemu virtual machines; acceso SSH

```
qemu ~/testbed/dklab1.ovl \
     -net nic,vlan=1,macaddr=F0:FF:FF:11:22:34 \
     -net socket,vlan=1,listen=:3333,id=mynetcard0 \
     -net nic,vlan=0 -net user,vlan=0,hostfwd=tcp::22222-10.0.2.15:22 \
     -monitor telnet:127.0.0.1:33333,server,nowait \
     -k es -vga cirrus  # -soundhw sb16,pcspk
```

Código fuente 1: Cómo lanzar la imágen virtual dklab1.ovl usando qemu

```
qemu ~/testbed/dklab2.ovl \
     -net nic,vlan=1,macaddr=F0:FF:FF:11:22:35 \
     -net socket,vlan=1,connect=127.0.0.1:3333,id=mynetcard0 \
     -net nic,vlan=3 -net user,vlan=3,hostfwd=tcp::22223-10.0.2.15:22 \
     -monitor telnet:127.0.0.1:33334,server,nowait \
     -k es -vga cirrus # -soundhw sb16,pcspk
```

Código fuente 2: Cómo lanzar la imágen virtual dklab2.ovl usando qemu.

Qemu es un software potente, y a su vez no exento de dificultades y, en ocasiones, comportamientos erráticos. A continuación vamos tanto a justificar los flags pasados como a dar algunas recomendaciones de uso.

- Las líneas que comienzan por "-net", configuran las interfaces de red que tendrá la máquina virtual. La primera representa la tarjeta enlazada entre las dos máquinas, de forma que se puedan comunicar; precisamente por esa interconexión, especificamos una dirección MAC distinta en cada una. La segunda aparición de "-net"depende de la anterior, pues especifica cómo se va a hacer efectiva la comunicación entre las dos instancias de qemu para permitir el enlace de red virtual: la primera instancia escucha en el puerto tcp 3333 y la segunda se conectará a éste. En una red virtual distinta, la última aparición de "-net"crea una segunda tarjeta de red que conecta con el SO host, de forma que pueda acceder a internet a través de éste, pero además:
- Nótese que con la subopción

  `,hostfwd=tcp::2222X-10.0.2.15:22`

  qemu hará una redirección de un puerto del SO host al puerto 22 de la máquina virtual. En concreto el puerto 22222 del SO host se redirige al 22 de dklab1, y el 22223 del SO host al 22 de dklab2. Estas redirecciones nos permiten usar un cliente SSH para iniciar una shell en la máquina virtual, conservando la configuración de teclado, localización etc que tenemos en el SO host. Por tanto es una excelente idea instalar ahora un servidor SSH tal como el del proyecto OpenSSH: cuando una máquina virtual termine de arrancar y nos presente el login, introducimos las credenciales del usuario root que se dieron y ejecutamos:

```
dhclient eth1
apt-get update
apt-get install openssh-server openssh-client
invoke-rc.d ssh status || invoke-rc.d ssh start
```

y en dklab2, idénticamente:

```
dhclient eth1
apt-get update
apt-get install openssh-server openssh-client
invoke-rc.d ssh status || invoke-rc.d ssh start
```

Y de ahora en adelante podremos usar un cliente SSH como ssh -también del proyecto OpenSSH- o putty -del proyecto Putty- para acceder a las máquinas virtuales. Con ssh sería:

```
ssh -o "UserKnownHostsFile /dev/null" root@127.0.0.1 -p 22222 # -X
#... para acceder a dklab1.
ssh -o "UserKnownHostsFile /dev/null" root@127.0.0.1 -p 22223 # -X
#... para acceder a dklab2.
```

A la vez, tener a openSSH nos permite, usando software como SSHfs/Macfusion/WinSCP etc transferir ficheros a la máquina virtual mientras está activa. Con sshfs, del proyecto SSHfs, sería:

```
apt-get install sshfs
mkdir ./dklab1 ./dklab2
sshfs root@127.0.0.1:/ ./dklab1 -o port=22222
sshfs root@127.0.0.1:/ ./dklab2 -o port=22222
ls ./dklab1
ls ./dklab2
# Para desmontar:
fusermount -u ./dklab1
fusermount -u ./dklab2
```

Continuamos exponiendo los flags pasados a qemu 1:

- Con la opción "-monitor"podemos conectarnos por telnet y tener un prompt del supervisor de qemu. Desde éste podemos comprobar el estado de la máquina, crear nuevas redirecciones de puertos, dispositivos... Algunos ejemplos:

```
# Dependen de que socat este' instalado y accesible:
echo help | socat - tcp4:localhost:3333
echo info | socat - tcp4:localhost:3333
echo info network | socat - tcp4:localhost:3333
echo hostfwd_add tcp::55555-10.0.2.15:5222 | socat - tcp4:localhost:3333
echo 'hostfwd_add tcp::55432-10.0.2.15:5432' | socat - tcp4:localhost:33333
```

- "-k es -vga cirrus'éspecifica el tipo de teclado (español) y de tarjeta de video emulada (cirrus logic GD5446). Además "-soundhw"permite especificar una tarjeta de sonido emulada, pero nuestra versión de qemu tenía problemas y por ello aparece comentado, más detalles al respecto se darán después.
- Qemu facilita las opciones "-curses"y "-vnc display[,option[,...]]"para entornos no gráficos, si bien una vez pudiésemos instalar Openssh, sería preferible acceder a través de SSH.
- Para parar una máquina virtual es preferible, en lugar de cerrar qemu directamente, ordenarle al SO invitado que pare:

```
shutdown -h now
```

Problemas conocidos Qemu

- En la versión de debian 6, arrancar con soporte de tarjeta de sonido ("-soundgw sb16", o ac97, o es1370) acaba haciendo que qemu consuma todos los recursos de tiempo de procesador. Suponemos que este bug estará resuelto en otras versiones.
- Procesadores en SO host lentos, pueden producir:
  - Al levantar la segunda instancia de qemu e interaccionar con la primera, pueden copar la velocidad de procesamiento sin aparente fin; suele mejorar la situación si limitamos el uso de procesador de la primera instancia mientras carga la segunda.

- A veces qemu tiene algún problema estableciendo la redirección de puertos y ssh/putty en el SO host no conectan con el SO virtual. Revísese con el supervisor de qemu, o reiníciese.
- A veces no todos los servicios parecen estar en marcha tras el arranque:

```
ps aux | (sleep 1; egrep '(
 openvpn
|ntpd
|krb5kdc
|kadmin
|slapd
|bosserver
|fileserver
|volserver
|vlserver
|ptserver
|salvager
|buserver
|upclient
|upserver
|afsd
|jabberd2-.*xml
|postgres.*conf
|exim4
|dovecot)')
```

Si alguno no aparece en la salida, se lanza manualmente su script de inicio con el argumento start:

```
/etc/init.d/<service-script-name> start
```

- Nosotros solemos usar algo casi equivalente:

```
invoke-rc.d <service-script-name> start
```

- Si se hacen redirecciones de la salida gráfica de aplicaciones del SO virtual al SO host, puede bloquearse el servidor gráfico en el SO host en algún momento posterior. Ésto sólo nos ha ocurrido al usar el viejo y discontinuado gestor de ventanas "PWM", de forma que al cambiar (en nuestro caso a "Fluxbox") el problema desapareció. Se recomienda por tanto utilizar sistemas de escritorios o gestores de ventanas recientes.

1.4. Algunas modificaciones convenientes en las imágenes

1.4.1. Configuración de red y del nombre de la máquina

En dklab1:

```
echo dklab1 > /etc/hostname
vim /etc/network/interfaces
```

```
auto lo
iface lo inet loopback

allow-hotplug eth0
iface eth0 inet static
     address 192.168.1.10
     netmask 255.255.255.0
     #post-up ifup ovpnCASAFX-SRV # relacionado con la VPN y DNS

allow-hotplug eth1
iface eth1 inet dhcp
```

Y hacemos que se sigan los cambios (puede interrumpir las conexiones SSH activas):

```
ifdown -v -a
ifup -v -a
```

En dklab2:

```
echo dklab2 > /etc/hostname
vim /etc/network/interfaces
```

```
auto lo
iface lo inet loopback


allow-hotplug eth0
iface eth0 inet static
     address 192.168.1.20
     netmask 255.255.255.0
     #post-up ifup ovpnCASAFX # relacionado con la vpn y dns


allow-hotplug eth1
iface eth1 inet dhcp
```

Y hacemos que se sigan los cambios (puede interrumpir las conexiones SSH activas):

```
ifdown -v -a
ifup -v -a
```

1.4.2. Configuración de debconf y de la localización

Tanto en dklab1 como en dklab2:

```
dpkg-reconfigure -plow debconf
```

```
-"Interface to use:"
Dialog
-"Ignore questions with a priority less than:"
Low
```

```
dpkg-reconfigure tzdata
```

```
-"Geographic area:"
Europe
-"Time zone:"
Madrid
#... o cualquier otro par pertinente.
```

```
dpkg-reconfigure locales
```

```
-"Locales to be generated:"
es_ES.UTF-8 UTF-8, en_GB.UTF-8 UTF-8, en_US.UTF-8 UTF-8
#... o cualquier otra combinacio'n pertinente.
-"Default locale:"
en_US.UTF-8 UTF-8
#... o cualquier otro de los escogidos.
#    Se hara' efectivo en el siguiente login.
```

Repítase para dklab2.

1.4.3. Configuración de bash, vim

En dklab1:

```
rojo='\[\033[2;35m\][\D{%H:%M}]\[\033[1;31m\]
\u@\h:\[\033[0;37m\]\w\[\033[1;31m\][$WINDOW\$\[\033[0;37m\] '

echo "export PS1='$rojo'"  >> ~/.bashrc

cat >> ~/.bashrc <<EOF
export HISTCONTROL=ignoredups
export HISTSIZE=1000000
export HISTFILESIZE=1000000
export HISTTIMEFORMAT=
export GREP_OPTIONS="--color=auto"
export LS_OPTIONS='--color=auto --time-style=long-iso'
eval "\`dircolors\`"
alias ls='ls \$LS_OPTIONS'
alias l='less -R'
# make less more friendly for non-text input files, see lesspipe(1)
[ -x /usr/bin/lesspipe ] && eval "\$(lesspipe)"
EOF
```

```
cat >> ~/.vimrc <<EOF
set background=dark
au BufReadPost \
* if line("'\"") > 0 && line("'\"") <= line("$") | exe "normal g'\"" | endif
syntax on
set encoding&           " terminal charset: follows current locale
set termencoding=
set fileencodings=      " charset auto-sensing: disabled
set fileencoding&       " auto-sensed charset of current buffer
map <F12> 0xk0
map <F11> o####fx:<ESC>o####endfx<ESC>0
EOF

cat >> /usr/local/bin/catc <<EOF
#!/bin/sh
cat "\$@" | ccze -A
EOF
chmod a+x /usr/local/bin/catc
```

En dklab2 (es idéntico a dklab1 excepto el color del prompt bash):

```
azul='\[\033[2;35m\][\D{%H:%M}]\[\033[1;34m\]
\u@\h:\[\033[0;37m\]\w\[\033[1;34m\][$WINDOW\$\[\033[0;37m\] '

echo "export PS1='$azul'"  >> ~/.bashrc

cat >> ~/.bashrc <<EOF
export HISTCONTROL=ignoredups
export HISTSIZE=1000000
export HISTFILESIZE=1000000
export HISTTIMEFORMAT=
export GREP_OPTIONS="--color=auto"
export LS_OPTIONS='--color=auto --time-style=long-iso'
eval "\`dircolors\`"
alias ls='ls \$LS_OPTIONS'
alias l='less -R'
# make less more friendly for non-text input files, see lesspipe(1)
[ -x /usr/bin/lesspipe ] && eval "\$(lesspipe)"
EOF
```

```
cat >> ~/.vimrc <<EOF
set background=dark
au BufReadPost \
* if line("'\"") > 0 && line("'\"") <= line("$") | exe "normal g'\"" | endif
syntax on
set encoding&           " terminal charset: follows current locale
set termencoding=
set fileencodings=      " charset auto-sensing: disabled
set fileencoding&       " auto-sensed charset of current buffer
map <F12> 0xk0
map <F11> o####fx:<ESC>o####endfx<ESC>0
EOF

cat >> /usr/local/bin/catc <<EOF
#!/bin/sh
cat "\$@" | ccze -A
EOF
chmod a+x /usr/local/bin/catc
```

1.4.4. Software inicial adicional

En Debian, todo software complejo deja un documento en el que el desarrollador orienta sobre cómo trabajar con dicho software. Ese documento se encuentra en /usr/share/doc/<nombrepaquete>/README.Debian. Adicionalmente, se tiende a que todo ejecutable tenga una página de manual.

La sentencia anterior resume con qué mentalidad nos debemos enfrentar a nuevo software en Debian en lo que sigue de este documento.

```
apt-get update
apt-get install bash screen vim ccze man less file iputils-ping \
                traceroute socat nmap iproute bridge-utils tcpdump tshark\
                strace iptables aptitude apt tree netpipes rlwrap
                #etckeeper
update-alternatives --set editor /usr/bin/vim.basic
update-alternatives --display editor
```

bash	shell de acceso al SO en modo texto. En realidad ya presente en el sistema.
screen	multiplexor de terminales (permite usar una como si fueran varias)
vim	editor de texto típico
ccze	filtro coloreador de texto en la terminal
man	visor para el manual de los programas
less	paginador de texto en la terminal
file	inspector de ficheros
iputils-ping	comando de red ping
traceroute	trazador de rutas de red traceroute
socat	interconector de sockets y procesos (“netcat++”)
nmap	escáner de puertos
iproute	interfaz avanzada de configuración de red
bridge-utils	interfaz para la creación de switch de red por software
tcpdump	trazador de red en modo texto
tshark	trazador de red en modo texto del proyecto Wireshark
iptables	interfaz al sistema cortafuegos de red
strace	trazador de llamadas al sistema
aptitude	interfaz aptitude para la gestión de paquetes en debian
apt	interfaz apt para la gestión de paquetes en debian, ya instalado.

tree	lista ficheros y directorios en formato de árbol
netpipes	provee utilidad de limitación de tiempo de vida de un proceso
rlwrap	provee una interfaz gnu readline (historial...) a programas en modo texto sin ella
etckeeper	Es siempre una buena idea mantener /etc bajo un sistema de control de versiones (como git). Para no hacer más denso este documento, obbiaremos esa posibilidad, pero puede verse un ejemplo de uso, por ejemplo, aquí.

Caso de Xorg

Los clientes de nuestros servicios suelen ser gráficos, luego necesitan un servidor gráfico (Xorg) y adicionalmente un gestor de ventanas (fluxbox, icewm...) o incluso un entorno de escritorio (gnome, kde, lxde...). Para probar dichos clientes, es necesario instalar:

Nada más, usamos reenvío del protocolo X11 desde un SO host que cuenta con un servidor X: El flag -X de ssh (putty también tiene un menú al respecto) crea un túnel para reenviar la salida gráfica de los comandos en la máquina virtual, de vuelta al servidor X que detecte ssh en el SO host cuando es lanzado. Por tanto, con sólo añadir el flag -X, se usa el servidor X en el SO host y nos ahorra su instalación en las máquinas virtuales.[3]

```
ssh -X -o "UserKnownHostsFile /dev/null" root@127.0.0.1 -p 22222 # dklab1
ssh -X -o "UserKnownHostsFile /dev/null" root@127.0.0.1 -p 22223 # dklab2
#... simplemente an~adimos, a la sentencia que se dio anteriormente,
#    el flag -X cuando ssh se lanza en un SO host con servidor X.
```

Evidentemente, éste es el método preferido siempre que nos sea posible. En cualquier otro caso, necesitamos instalar un servidor Xorg en las máquinas virtuales.

Instalación mínima del servidor Xorg (10-15 MB) Se explica esta posibilidad para ayudar a economizar espacio de almacenamiento. En otro caso, sígase la instalación típica que explicamos después.

Instalamos xserver-xorg-core; éso instala todos los drivers de video y de métodos de entrada, a no ser que especifiquemos alguno:

- Puesto que el hardware de video es el emulado por qemu, es seguro instalar sólo el xserver-xorg-video-cirrus.

[3]Los usuarios habituados a exportar manualmente la variable DISPLAY y evitar la ineficiencia del reenvío de puertos por SSH, pueden hacerlo también. Concretamente en el SO host hay que permitir conexiones desde localhost con "xhost +127.0.0.1", y en la sesión ssh en la máquina virtual hay que configurar: "export DISPLAY=10.0.2.2:0" previamente a lanzar los comandos con salida gráfica.

- Y en el método de entrada, puesto que vamos a utilizar ratón y teclado, instalamos xserver-xorg-input-evdev.

Ante la duda, no se indicaría nada adicional a xserver-xorg-core para que se instale soporte para todo, decíamos. Si todo está claro:

```
apt-get install --no-install-recommends \
        xserver-xorg-core \
        xserver-xorg-input-evdev \
        xserver-xorg-video-cirrus


Xorg -configure
#...genera un ~/xorg.conf.new
Xorg -config ~/xorg.conf.new
#... pueden an~adirse: -dpi 100 y -nolisten a la li'nea anterior.
```

Éso es suficiente, para que los programas le manden su salida gráfica:

```
DISPLAY=:0 <programa>
#...donde programa podri'a ser xterm ti'picamente:
#   apt-get --no-install-recommends install xterm
```

Si instalamos xinit, podremos iniciar el servidor con:

```
apt-get install --no-install-recommends xinit
# Lanzamos Xorg y xterm:
xinit //usr/bin/xterm -- /usr/bin/X :0  &
# O bien Xorg y el gestor de ventanas por defecto:
xinit //usr/bin/x-window-manager -- /usr/bin/X :0  &
# O bien Xorg y el cliente/gestor/escritorio por defecto:
DISPLAY=:0 startx
```

Instalación típica Instalará una amplia variedad de paquetes. Normalmente se proveen varios métodos que controlan qué subconjunto de paquetes se utilizará:

```
apt-get install fluxbox  # o cualquier otro gestor de ventanas.


 #Si queremos un pesado escritorio:
apt-get install kde-full # o kde-standard, o mi'nimamente kde-core
apt-get install gnome   # o gnome-session-bin mi'nimo, sin scripts inicio


 #Otra forma ti'pica en debian de instalar especi'ficamente Gnome o Kde es:
tasksel install gnome-desktop --new-install
tasksel install kde-desktop   --new-install
```

1.5. Uso básico de las utilidades base: screen/bash/vim

Para desplegar el proyecto necesitamos introducir comandos y modificar ficheros. Teniendo ésto en cuenta, las utilidades que serán la base de nuestro trabajo son screen, bash y vim.

1.5.1. Screen y Bash

Entre una larga lista de funcionalidades, screen nos permite tener varias terminales donde sólo hay una (e incluso separarlas de ella y volver a tomarlas más tarde). Con ello, si necesitamos dejar una utilidad en una terminal e interaccionar con ella desde otra, usamos screen. Para lanzar screen, lo lanzamos con:

```
screen
```

Éste nos muestra entonces una terminal y su shell. Si queremos añadir otra, cambiar a ella y después deshacernos de ella, pruébese:

Control-a c (crea)

Control-a n (next)

Control-a k (kill; efecto parecido hace salir de su shell con "exit").

Con ésto nos bastará. Consúltese "man screen'ó http://aperiodic.net/screen/ para más información.

Respecto de bash, es la shell de usuario por defecto y nos permite lanzar todos los demás comandos. Adicionalmente tiene modificadores de control de flujo ("if", "for"...)

y variables pre-construídas que mejoran esa funcionalidad base. Podemos configurar su comportamiento en ~/.bashrc, consúltese "man bash'ó http://wiki.bash-hackers.org/start; en principio no parece estrictamente necesario conocer nada más.

1.5.2. Vim

Vim es una implementación de editor de ficheros de la familia de vi/ed. Si para desplegar el proyecto necesitamos introducir comandos y modificar ficheros, bash se ocupaba de lo primero y vim, finalmente, de lo segundo. Para lanzar vim:

```
vim <ruta al fichero que editar>
```

Ya dentro, admite:

cursores (nos permiten movernos libremente por el buffer - también otras: h,j,k,l...-)

a (nos permite añadir nuevo texto - también otras: i, o, O...-).

ESC (nos permite salir del "modo de añadir nuevo texto'ál inicial "modo comandos").

:w (nos permite salvar el buffer desde el modo de comandos).

:q (nos permite salir - y ":q!'ños permite salir sin guardar cambios).

A veces nos interesa que vim divida la pantalla en dos partes para poder comparar dos ficheros. En ese caso lanzamos a vim así:

```
vim -O <ruta al primer fichero> <ruta al segundo fichero>
```

Para movernos de un fichero a otro:

Control-w l (permite llevar el cursor al fichero de la derecha).

Control-w h (permite llevar el cursor al fichero de la izquierda).

Con ésto será suficiente. Puede ejecutarse

```
vimtutor
```

para ejercitar los comandos anteriores y aprender otros nuevos. Más información (por ejemplo sobre copiar y pegar) de vim en
http://www.swaroopch.com/notes/Vim_en:Table_of_Contents

1.6. Conclusiones

- *Imágenes*: nos basamos en una imagen de debian 6 aka "squeeze", descargable desde la web, para proporcionar un entorno de referencia en que desplegar el proyecto. Además aprovechamos el formato qcow2 de esa imagen para generar la de cada máquina virtual como "qcow2 overlays", es decir ficheros que contienen las diferencias respecto de la imagen base, permitiéndonos ahorrar espacio de almacenamiento en el equipo, o hacer eficientes backups en distintos momentos del despliegue en el testbed. Los overlays pueden ser presentados por el SO host como dispositivo de disco convencionales, por lo que pueden ser usados por soluciones de virtualización distintas a la propuesta aquí, o simplemente ser montados para explorar su contenido.

- *Qemu*: podemos levantar los overlays con la máquina virtual proveída por qemu (véase cómo en 1). Decantarse por qemu tiene ventajas e inconvenientes:

  - ventajas: nos permite hacer el despliegue sobre las máquinas virtuales desde casi cualquier SO/HW: windows, osx, linux, *bsd... todos tienen un porte de qemu.
  - desventajas: sin embargo, puede ser ineficiente en tiempo de procesador (técnica de virtualización "full translation"). Ésto no representa incompatibilidad alguna, y es por tanto Qemu la solución definitiva para arrancar las máquinas virtuales.

- *LXC (desechado)*: si estamos trabajando en una máquina con linux (>2.6.29), tenemos la oportunidad de correr las máquinas virtuales tan eficientemente como el SO host sobre el que se ejecutan éstas. Tras un proceso no trivial (montado de las imágenes overlay, declación de cierta configuración, realización de pequeños cambios -compatibles con qemu- en los overlays) la llamada a lxc-start arranca los contenedores.

  - ventajas: las máquinas virtuales se ejecutan con la máxima eficiencia posible en el sistema host (técnicas de contextualización, en lugar de paravirtualización o virtualización).
  - desventajas: tenemos que llamar a chroot al punto de montaje de los overlays para instalar software desde los repositorios de debian (si bien ésto puede verse como una ventaja, la de ahorrarnos la configuración de red entre el contenedor y la ruta por defecto del SO host, además de proveer aislamiento entre ambos). Finalmente, lxc sólo puede usarse cuando el kernel del SO usa linux >2.6.29. Por ejemplo el mismo debian 6 "squeeze" que se usa en los overlays, sería válido como SO host para lxc.
  - incompatibilidades: pero los contenedores son incompatibles con el cliente AFS pues, implementado como un módulo del kernel, sólo puede ser usado por una de las máquinas virtuales.

- *Paravirtualización KVM, XEN... (desechado)*: estas técnicas se sitúan, en cuanto a rapidez, entre las dos tecnologías anteriores y requieren soporte específico a nivel hardware (Intel VT, AMD V). Nosotros no disponíamos de un procesador así como tampoco queríamos centrarnos en soluciones con requerimientos hardware específicos, por lo que no se valoró nada más.

En los siguientes capítulos se considerarán las máquinas virtuales arrancadas y configuradas según lo expuesto en este capítulo. LyX