Virtualización e hipervisores

PID_00286213

Remo Suppi Boldrito

Tiempo mínimo de dedicación recomendado: 6 horas

Universitat Oberta de Catalunya

Remo Suppi Boldrito

Ingeniero de Telecomunicaciones. Doctor en Informática por la UAB. Profesor del Departamento de Arquitectura de Computadores y Sistemas Operativos en la Universidad Autónoma de Barcelona.

La revisión de este recurso de aprendizaje UOC ha sido coordinada por el profesor: Josep Jorba Esteve

Segunda edición: febrero 2022

Av. Tibidabo, 39-43, 08035 Barcelona
Autoría: Remo Suppi Boldrito
Producción: FUOC

Índice

Objetivos

Los objetivos principales de este módulo son los siguientes:

1. Conocer los diferentes tipos de virtualización y saber cuál es el más adecuado para cada tipo de infraestructura.

2. Analizar los diferentes hipervisores y conocer sus ventajas y limitaciones.

3. Analizar las características de los contenedores y sus ventajas en relación con las máquinas virtuales.

Además, el estudiante deberá centrar su atención en los siguientes conceptos fundamentales de este módulo:

- Virtualización y virtualización soportada por hardware
- Hipervisor
- KVM, VirtualBox, Hyper-V, Vmware ESX
- Contenedores
- Docker, LXC

1. Hipervisores, máquinas virtuales y contenedores

Como se ha mencionado anteriormente, la virtualización es el gran actor que permite que el *cloud* exista y sea posible. De la misma forma que un usuario (no experto) puede utilizar un ordenador, ya que el SO actuará como una capa de abstracción que esconderá todas las complejidades de los dispositivos físicos presentando una interfaz gráfica y cercana a las acciones que desea hacer el usuario, la virtualización será lo mismo para el *cloud*.

En cierta forma, podemos decir que esta es el equivalente al SO del *cloud* ya que permitirá disponer de recursos virtualizados (máquinas, servidores, redes y almacenamiento) que harán un trabajo real pero que compartirán recursos físicos y esconderán toda esta complejidad al usuario que los utilizará.

La pregunta que puede surgir es: ¿para qué necesito la virtualización si con un sistema operativo ejecutándose en un hardware (*metal-base*) ya tengo esto? Es evidente que sobre una máquina podemos poner un servicio y podremos utilizar los recursos para un usuario, y esto tendrá una carga sobre el sistema físico determinada que en la mayoría de los casos no llega al 10 % (se puede analizar cuántos recursos en momentos de funcionamiento estable está dedicando nuestro ordenador).

¿Qué pasa si ahora, en función de un ahorro de costos y para diez usuarios más, deseo aprovechar los recursos físicos y debo poner diez servicios (que pueden ser iguales al que estoy dando para el primer usuario, pero también pueden ser diferentes y con diferentes SO y con diferentes datos)? Podría utilizar algunas estrategias de compartición de recursos, librerías, hacer espacio controlado para los datos de cada una de ellas, pero nunca serían sistemas aislados/privados con sus configuraciones y sus especificidades, ni tampoco sería ágil, ni eficiente, sería complicado de mantener, monitorizar y administrar.

Pensemos ahora en una empresa realizando esta tarea sobre cien servicios/servidores y mil clientes que tienen una dinámica y servicios que entran y salen, *apps* que se inician y terminan, y un montón de configuraciones, compatibilidades, librerías y un largo etcétera de cuestiones específicas. Todo esto haría que fuera extremadamente difícil o imposible a un costo razonable, con eficiencia y, además, que fuera escalable (que es uno de los objetivos vitales del *cloud* como negocio; es decir, si tengo diez usuarios y vienen diez más, debería poder ampliar la infraestructura en forma simple y poder darles servicio en un tiempo aceptable).

La virtualización es el gran posibilitador tecnológico del *cloud* ya que se podrán crear los recursos virtuales necesarios y el usuario «verá» (con una eficiencia cercana como si tuviera un dispositivo hardware) su infraestructura (máqui-

nas, servidores, discos, red) y será solo SU infraestructura y lo mismo los N clientes del *cloud*. Con esto se aprovecharán todos los recursos físicos existentes, maximizando la eficiencia y con la consiguiente reducción de costos.

En cuanto al ahorro de recursos, ¿por qué es mejor virtualizado que físico? Si tenemos un requerimiento de disco, por ejemplo de 1 Tbyte para un usuario, y el recurso es físico, el proveedor deberá asignarle, tanto si usa como si no, 1 TByte, ya que el usuario pagará por ello y el costo estará en relación con el coste del recurso; si tengo mil usuarios con los mismos requerimientos, se deberán disponer de 1 Pbyte para tal fin. De valores estadísticos de utilización real se sabe que estos no superan el 50 % (el cliente siempre sobreaprovisiona), y en valores reales se encuentra dentro de un 10-15 %, por lo cual si el recurso es virtualizado y la provisión es dinámica, el proveedor no debe disponer del 1 Pbyte para dar inicio a los clientes en el servicio; de esta forma, si las necesidades van aumentando con el tiempo y el sistema es escalable, se podrán ir aumentando los recursos en función de las necesidades reales de los clientes y no tener «atrapado» 1 Pbyte de disco. Lo mismo se puede decir con el cómputo y la red.

Por ello la virtualización es la tecnología base (y esencial) para el *cloud computing* ya que abstrae al usuario de forma total de lo que hay «abajo»; de hecho, el usuario tampoco sabe dónde se estará ejecutando su servidor y lo mejor de ello es que tampoco le hace falta, ya que habrá pedido unas prestaciones y un servicio, garantizado por una SLA, y lo tendrá a un precio aceptable. Para el proveedor será un negocio con una cuenta de resultados positiva, será eficiente, reducirá costos que podrá trasladar a sus clientes, reducirá espacio y consumo de energía, mantendrá los recursos aislados (privacidad), será ágil en la gestión y aprovisionamiento y será escalable (premisa de que si «necesito más», puedo «agregar más»).

Es por ello que la virtualización nos permite:

- **Incrementar la utilización de los recursos físicos**: el espacio de disco es un caso, pero también si un servidor está poco utilizado y otro mucho, puedo mover carga de un servidor a otro y en algunos casos «en ejecución», cosa impensable sobre un servidor físico.

- **Consolidación de recursos y escalabilidad**: más allá de los discos, que es un caso habitual, se podrá hacer lo mismo con servidores de todo tipo, pero también estar abierto a que, si se necesita más, se podrán anexar más recursos, cosa imposible con un superordenador.

- **Menor espacio**: vinculado al punto anterior, menos servidores = menos espacio, y esto está relacionado con la escalabilidad, ya que si el negocio crece el ampliar un centro de datos, suele ser una inversión muy grande, por lo cual todo espacio que se pueda optimizar es bueno.

- **Ahorro energético y eficiencia**: normalmente, en centros de datos la relación suele ser de que de cada KWatt gastado en cómputo se gasta otro KWatt en refrigeración, por lo cual reducir el número de servidores se traducirá directamente en reducción de costos.

- **Agilidad y reducción de administración**: tener MV preparadas para las diferentes opciones o configurarlas en línea reduce notablemente las tareas de instalación y administración –ya que será centralizada–, generando generalmente un autoservicio por parte del usuario, con la consiguiente reducción de atención por parte del proveedor, lo cual genera satisfacción en el cliente y reducción de RR. HH. en el proveedor.

- **Alta disponibilidad**, **resguardos y recuperación**: en esencia, un servidor virtualizado no es nada más que un archivo, el cual se puede clonar, copiar, o incluso en hipervisores modernos, mover en caliente, lo cual permite estrategias de diferentes tipos para la alta disponibilidad, QoS y resguardo.

- Y, finalmente, todo ello repercute en la **reducción de los costos** de gestión, mantenimiento y escalabilidad y las mejoras en el TCO (*total cost of ownership*) y el ROI (*return on investment*).

No obstante, la virtualización no está libre de riesgos que pueden provenir de:

- Las limitaciones de hardware: se deberá analizar muy bien la carga y monitorizar para evitar su sobreutilización, ya que se deberá cumplir con una QoS acordado en una SLA.

- Plataforma de virtualización (sobre todo de tipo 1 *bare-metal*) que no son compatibles con todo el hardware (se deberá consultar para cuál está certificada, y así evitar tener problemas de controladores o funcionalidades no permitidas.

- Necesidades tecnológicas: que los sistemas virtualizados necesiten determinados «nuevos» recursos tecnológicos que la plataforma de virtualización no esté preparada para habilitar.

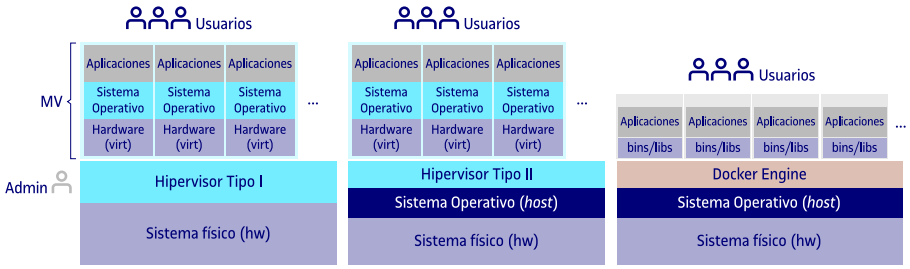
- Y, finalmente, la más probable/habitual: un fallo del hardware del servidor físico (lo cual afecta a todos los servidores virtualizados) y que se deberá solventar con alta disponibilidad de los servidores físicos (redundancia). En este último caso incrementará el coste, pero se podrá repercutir al cliente a través de la SLA correspondiente.

Como ya se mencionó anteriormente, el **hipervisor** (o monitor de máquina virtual, VMM, *virtual machine monitor*) es la capa de abstracción para la virtualización que podrá actuar, conjuntamente, como un núcleo indivisible con el sistema operativo o separado de él como una aplicación más (pero trabajando conjuntamente con él). El hipervisor mostrará a las máquinas virtuales (*guest*) una infraestructura virtualizada que tendrá su contrapartida en la máquina sobre la cual se ejecuta (*host*) y que permitirá que el *guest* «vea» esta infraestructura como si del hardware físico se tratara. El hipervisor tendrá como tarea, además, gestionar las máquinas virtuales, asignar los recursos, permitir el acceso al hardware equivalente, monitorizar/contabilizar los recursos gastados por cada MV y todo un conjunto de acciones que serán necesarias para la gestión y administración del sistema.

En la actualidad podemos contar con hipervisores de **tipo 1** (también llamados nativos, *unhosted* o *bare-metal*), por ejemplo, VMware ESXi, Xenserver, Xen entre otros, donde el hipervisor forma un conjunto indivisible con el SO o **tipo 2** (también llamado *hosted*), por ejemplo, VMware Workstation Player, VirtualBox, Qemu, KVM (aunque algunos autores lo consideran de tipo 1), Hyper-V, donde el hipervisor se ejecuta como una capa que interactúa con el SO, pero no está incluido en él (excepto KVM).

Es importante recordar que la mayoría de ellos solo pueden ejecutarse en procesadores con extensiones hardware VT-x/AMD-V y estas serán necesarias para que los *guests* puedan ser de 64 bits (además, en el tipo 2 el SO deberá ser de 64 bits también). Por otro lado, es importante destacar el papel que juega la virtualización del SO, ya que hoy en día es una tendencia al alza (muchas PaaS y SaaS ya solo trabajan con *containers*) y con el papel predominante de Docker y LXC/LXD en este contexto.

Se debe tener en cuenta que, así como con hipervisores tipo 1 o 2, el sistema operativo *guest* puede ser de cualquier otro tipo (y podrá ser de 32 bits o 64 bits siempre y cuando el *host*, si es de tipo 2, sea de 64 bits), en el caso de la virtualización de SO (contenedores), el *guest* solo podrá ser del mismo tipo (excepto en Docker para Windows), aunque podrán ser diferentes distribuciones (es decir, que no podemos tener un *host* Linux y un *guest* Windows, pero sí tener un *host* Debian y un *guest* Fedora). Como se ha mencionado, un caso especial de contenedores es el de Docker para Windows, ya que este permite tener contenedores Windows, y como W10 incluye *kernel* de Linux, también permite tener contenedores de Linux. La figura siguiente muestra las capas de la virtualización, donde ocurre:



A continuación, se realizarán diferentes experiencias de instalación y configuración de los hipervisores más habituales ya que estos son la base para todo sistema de *cloud computing* con el objetivo de analizar sus prestaciones y facilidades en la configuración/gestión. La virtualización de SO y contenedores se dejarán para siguientes capítulos, donde se describirán con detalle.

2. Casos de uso

2.1. KVM

Kernel-based Virtual Machine (KVM) es un proyecto *open source* que implementa sobre Linux (desde la versión 2.6.20/2007) una infraestructura de virtualización permitiendo que Linux actúe como hipervisor (según algunos autores, de tipo 2, pero cuando ya está instalado es de tipo 1, según otros) a través de un módulo (*kvm-intel/amd.ko*) cargable en el *kernel* y herramientas en el espacio de usuario para su gestión.

Este hipervisor permite ejecutar máquinas virtuales a partir de imágenes de disco que contienen sistemas operativos sin modificar y cada máquina verá su propio hardware virtualizado (tarjeta de red, discos duros, tarjeta gráfica, etc.). Solo necesita un procesador x86/64, con soporte para virtualización (VT-X/AMD-V) y puede ejecutar diferentes SO *guest* como Linux/Unix/OSX/Darwin/Windows, entre otros, tanto de 32 como de 64 bits. Además, soporta un conjunto de dispositivos paravirtualizados para Linux, openBSD y FreeBSD, Windows, utilizando la API VirtIO [Kvm].

2.1.1. Instalación y configuración

Su instalación se puede realizar sobre una máquina física (baremetal) o incluso, como prueba funcional, sobre una MV pero es necesario que el hipervisor soporte virtualización anidada, ya que KVM necesita «ver» las extensiones hardware del procesador. Las pruebas para este apartado se han realizado sobre una máquina con procesador Intel i7 620M (64 bits, 2 *cores* + *Hyper-Threading* = 4 procesos) y sistema operativo Ubuntu 21.04 LTS. Ubuntu utiliza KVM como tecnología para virtualización de *back-end,* principalmente para servidores no gráficos, la librería libvirt como *toolkit/API*. Libvirt actúa como *front end* para gestionar las máquinas virtuales a través de `virt-manager` (`GUI`) o `virsh` (`CLI`).

A continuación, se describen los pasos para verificar si el *host* soporta virtualización y si dispone del SO adecuado. Para verificar si el procesador soporta las extensiones hardware se debe ejecutar:

```
egrep -c '(vmx|svm)' /proc/cpuinfo
```

Si es 0 significa que la CPU no soporta virtualización, y si es 1 o más (4 en nuestro caso) sí la soporta (en caso de 0 se debe verificar que estas extensiones no estén deshabilitadas desde la BIOS, ya que muchos fabricantes ponen esta opción por defecto). También se puede instalar el paquete `apt-get install`

`cpu-checker` y ejecutar `kvm-ok`; con ello nos indicará si es posible o no ejecutar KVM sobre esta arquitectura. Cabe tener en cuenta que una arquitectura que no soporte KVM podrá ejecutar VM, pero sin aceleración, solamente por emulación.

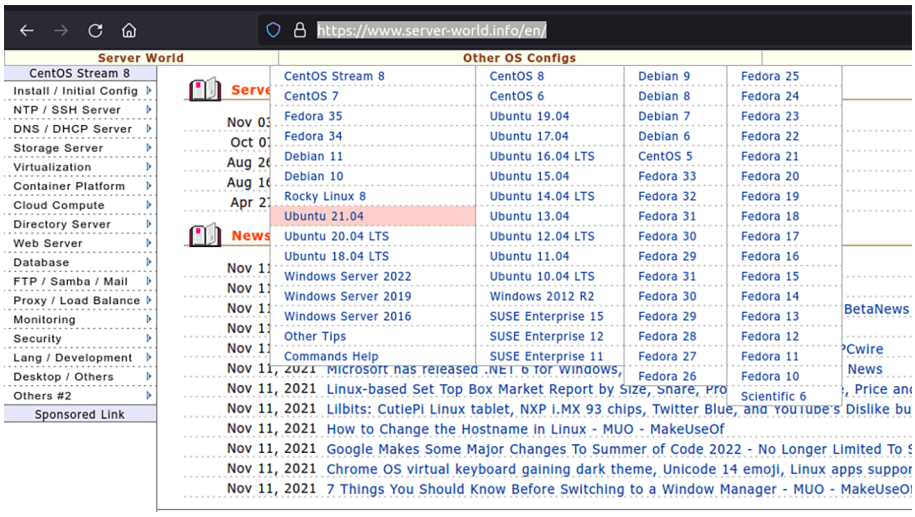
Si se necesita rendimiento es aconsejable tener un *kernel* del SO de 64 bits (y si si se desea MV con más de 2 GB de RAM, el *kernel* deberá ser de 64 bits). Un *kernel* de 64 bits podrá albergar SO *guest* de 32 o 64 bits, mientras que si el *host* es de 32 bits, el *guest* solo podrá ser de 32 bits. Para verificar si el procesador es de 64 bits, cabe ejecutar:

```
egrep -c ' lm ' /proc/cpuinfo
```

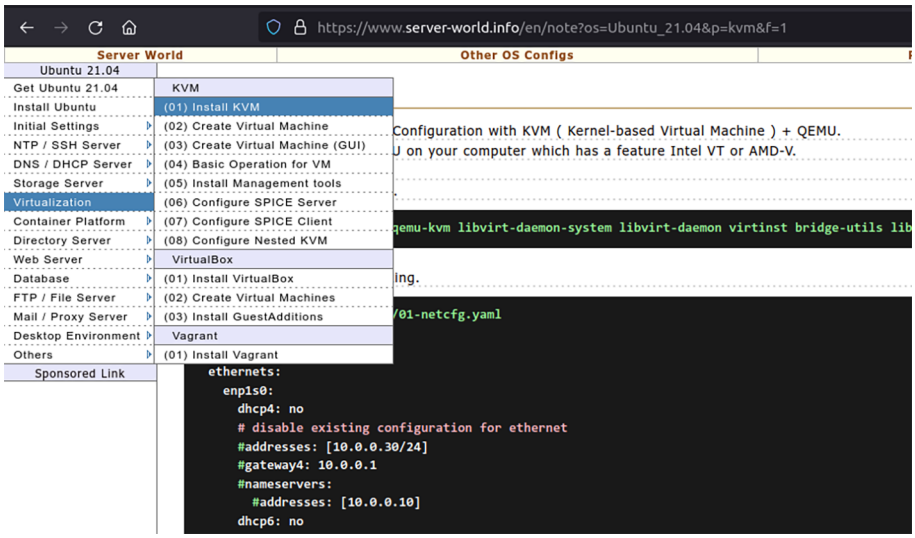
Si el resultado es 0 = no, 1 o mayor = si; lm significa *long mode* equivalente a CPU de 64 bit si es > 0.

Para verificar si el SO *host* es de 64 bits hay que ejecutar `uname -m` que indicará x86_64 (o amd64). Si indica i386, i486, i586, i686, el *kernel* es de 32 bits, y si el procesador es de 64, se debería cambiar.

Dado que existen diferencias entre las instalaciones de KVM para las diferentes distribuciones, recomendamos documentarse sobre el proceso, en función de la distribución y versión que se disponga del SO *host*, en Server World. Los autores de Server World mantienen actualizados para las principales distribuciones y últimas versiones los procedimientos de instalación de una forma detallada y bien documentada, por lo cual es recomendable seguir los pasos de instalación realizados por estos expertos. Cuando se accede a la página se muestran las principales notificaciones y SO actualizados y desde el selector de *Other* OS *Configs* se puede seleccionar la versión adecuada de SO disponible en nuestro host como se muestra en la figura siguiente.

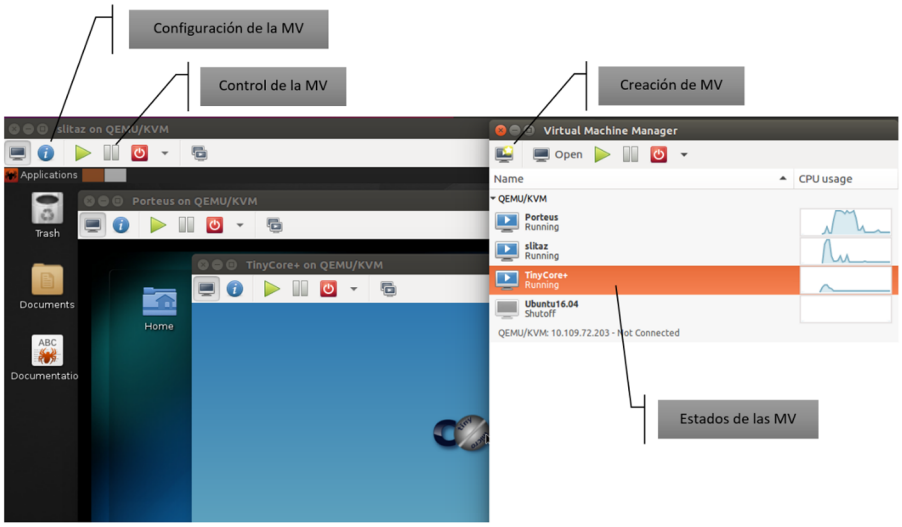


En la figura siguiente se puede observar para Ubuntu 21.04 el selector del menú en *Virtualization* -> *Install* KVM y el proceso a seguir para instalar KVM sobre este SO.



Donde se instalan las diferentes librerías de libvirt i qemu necesarias para administrar instancias qemu y kvm y bridge-utils permite configurar un *bridge* desde la red hacia las VM. En las opciones siguientes del menú se puede acceder a los procesos paso a paso para crear un MV desde un terminal o con una interfaz gráfica, gestionar un MV, instalar herramientas complementarias para gestionar las MV o configurar un servidor/cliente SPICE para acceder remotamente a las MV. Por último, cómo configurar KVM para que las MV vean las extensiones hardware del procesado (lo que se conoce como *virtualización anidada*).

La figura siguiente muestra el `virt-manager` ejecutando tres máquinas virtuales con diferentes instalaciones de Linux (TinyCore+, Proteus, SliTaz). A través de esta aplicación o con `virsh` se podrán crear, gestionar (poner marcha, reiniciar, parar, etc.) y administrar todas las opciones de la máquina virtual (cpu, red, pantalla, memoria, discos, etc.).



Además de las opciones antes mencionadas de SPICE, el `virt-manager` permite gestionar la MV tanto en local como **en remoto** (es decir, desde otra máquina diferente a donde se están ejecutando las MV). Para ello, en la máquina que se desea conectar a la MV se debe instalar:

```
apt-get install virt-manager ssh-askpass-gnome --no-install-recommends
```

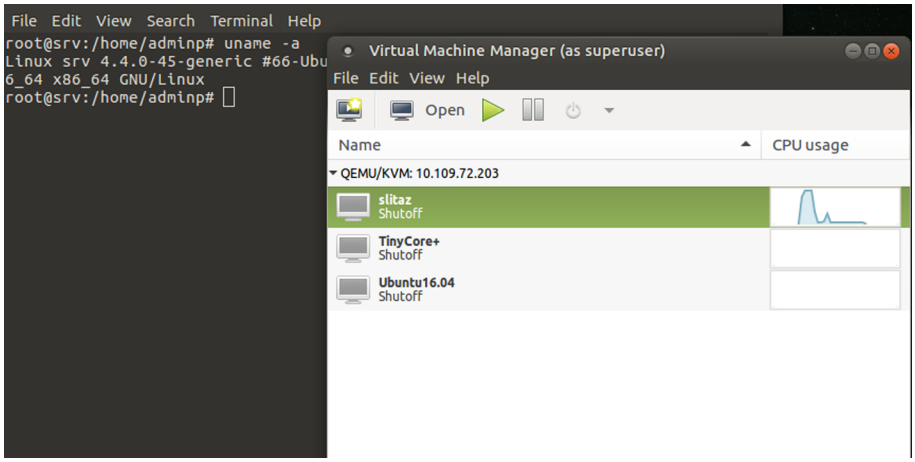
Es importante el paquete `ssh-askpass-gnome` o cualquiera de los `ssh_askpass*` ya que la conexión con el servidor será a través de `ssh` y se necesita validar la sesión con el usuario y *passwd* del servidor libvirt, que se realizará a través de este paquete.

Se inicia el `virt-manager` y en el menú *File* → *Add Connection* → *Hypervisor=QEMU/KVM, Username* = usuario remoto dentro del grupo de *libvirt* en *host* remoto, *Hostname* = IP del *host* remoto.

Con ello se verá la conexión, se solicitará el *passwd* para el usuario indicado y ya se podrá trabajar exactamente como si fuera en local, pero desde el *host* remoto. La figura siguiente muestra la conexión remota que tiene el formato:

```
quemu+ssh://user_libvirt@ip_host_remoto/system
```

sobre una máquina diferente a la que están ejecutando las máquinas virtuales.



Esto mismo se puede hacer en la CLI ejecutando en un terminal:

```
virsh -c quemu+ssh://user_libvirt@ip_host_remoto/system
```

donde pedirá el *passwd* y podremos acceder a la administración remota (mostrará el *prompt virsh #* donde el comando *help* nos dará todas las posibilidades de gestión del `virsh`).

2.1.2. Networking

Hay dos formas diferentes de permitir que una MV acceda a una red externa. El modo por defecto, conocido como *usermode networking*, que realiza un NAT a través de la interfaz del *host* hacia la red externa. Si se desea acceder a servicios sobre SO *guest* de una MV, es necesario configurar una *bridged networking*.

En el ***usermode networking*** (por defecto), las máquinas se configuran para que puedan salir hacia la red externa (internet) y los SO *guest* obtendrán IP en el rango 192.168.122.0/24 y el SO *host* tendrá 192.168.122.1. Desde el SO *guest* se puede acceder al *host* (por ejemplo, para compartir archivos o cualquier otra acción) por medio de ssh 198.168.122.1 o desde el *host* a la IP asignada dentro de 192.168.122.0/24 al *guest*. Si los SO *guests* no tienen acceso al *host* o la red externa, cabe verificar con `sudo iptables -n -t nat -L` si se está haciendo el NAT, donde se deberá ver (entre otros):

Chain POSTROUTING (policy ACCEPT)
target prot opt source destination
MASQUERADE all -- 192.168.122.0/24 !192.168.122.0/24

Si esta regla no existe, se deberán parar todas las MV y recrear las reglas con

```
virsh net-destroy default
virsh net-start default
```

Si se pierde la conectividad durante grandes transferencias (por ejemplo, durante un `rsync`), una posible solución es habilitar el *driver* de red virtio (*driver* paravirtualizado y necesario para Windows).

Se puede obtener más información sobre este modo y cómo funciona sobre la página de libvirt [Lib].

El ***bridged networking*** permite conectarse, a través de la interfaces virtuales, a la interfaz física y, por lo tanto, ser visibles desde afuera como si de una máquina habitual se tratara. Se debe tener cuidado, ya que si el dispositivo físico sobre el cual se hace el *bridge* es inalámbrico (wifi), este no funcionará (muchos dispositivos *wireless* no soportan el *bridging*). Sobre las distribuciones que ejecutan el NetworkManager es necesario deshabilitar este procedimiento para evitar que interfiera en la configuración (si es necesario se puede hacer con `systemctl stop NetworkManager; systemctl disable NetworkManager`). Si se ejecuta `brctl show` veremos:

bridge name bridge id STP enabled interfaces
virbr0 8000.5254002bc4ce yes virbr0-nic

Si da errores, cabe verificar que se dispone del paquete bridge-utils (y si no, se deberá instalar). La ejecución de `ip address` dará:

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1 inet 127.0.0.1/8 scope host lo ...
2: enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br0 state UP group default qlen 1000 ..
4: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000 link/ether 52:54:00:2b:c4:ce brd ff:ff:ff:ff:ff:ff inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
5: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast master virbr0 state DOWN group default qlen 1000 link/ether 52:54:00:2b:c4:ce brd ff:ff:ff:ff:ff:ff

Donde virbr0 es la interfaz NAT de KVM y enp0s25 la primera interfaz física del *host*. Para instalar el *bridge* de forma manual (recordar que en las instalaciones de Server World se configura el Netplan habitual en la distribución Ubuntu pero se puede mirar en Debian 11 por ejemplo donde la instalación es manual) y se pueden encontrar más detalles en [Lib,Kne]). En el caso de la configuración manual, primero se bajará la interfaz hardware (*enp0s25* en este caso de estudio) `ifdown enp0s25` y a continuación se modificará el archivo */etc/network/interfaces* como:

```
auto lo auto br0
iface lo inet loopback


iface enp0s25 inet manual
```

```
iface br0 inet static
    address 158.109.65.67
    netmask 255.255.255.0
    gateway 158.109.64.1
    dns-nameservers 158.109.0.1 158.109.0.9
    bridge_ports enp0s25
    bridge_stp on
    bridge_fd 0
    bridge_maxwait 0
```

En este caso se ha utilizado una red con IP públicas, pero puede ser perfectamente una red con IP privadas y que después haya una máquina que haga NAT hacia la red pública. Los parámetros de *bridge* indicados son: *bridge_stp off|on* es la configuración para el *spanning tree* (aunque puede causar errores en la asignación de dhcp a los *guests*, dado este caso poner en *off* u omitirlo), *bridge_fd 0* indica *turns off all forwarding delay y bridge_maxwait 0* es el tiempo que el sistema esperará que los puertos estén disponibles (0 = no espera).

A continuación, ejecutar `ifup br0` y con `brctl show` veremos la nueva interfaz (y que también podremos verificar con `ip address`):

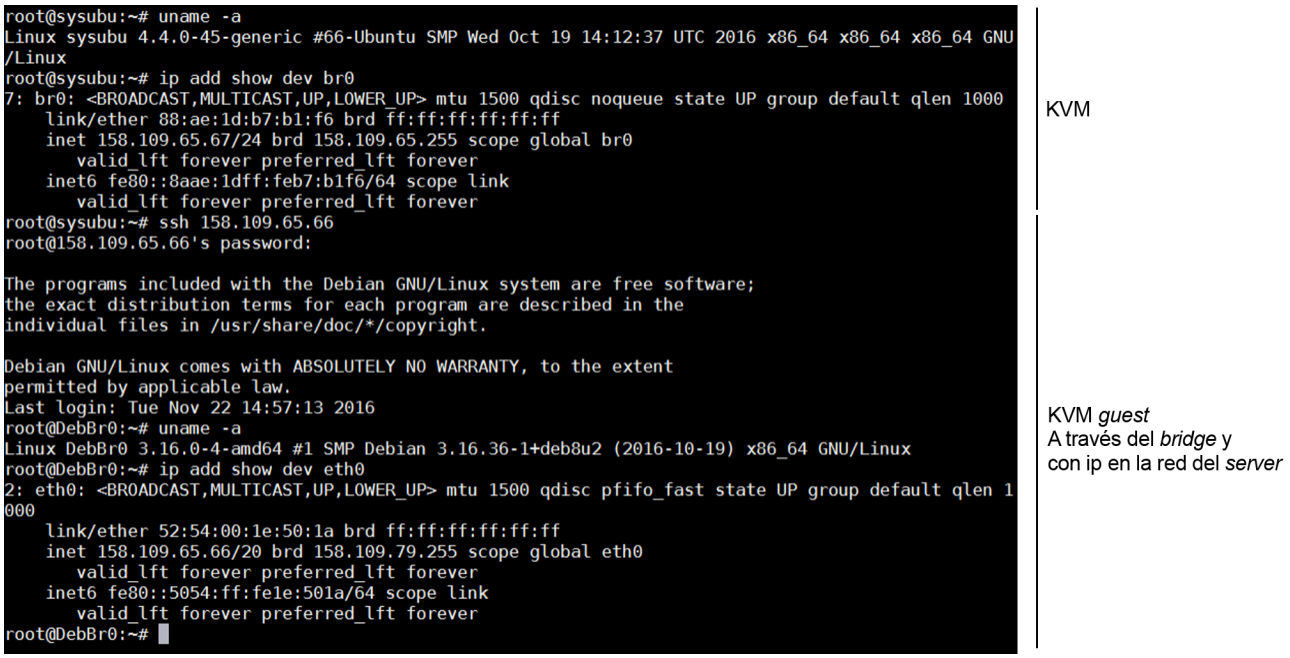
bridge name bridge id STP enabled interfaces
br0 8000.88ae1db7b1f6 yes enp0s25
virbr0 8000.5254002bc4ce yes virbr0-nic

A continuación se puede verificar la conectividad y comenzar a crear MV utilizando esta interfaz, tanto en el modo gráfico como en el modo CLI (si no se dispone de DHCP en el servidor la asignación de ip y demás parámetros en el *guest* deberá ser manual). Si se desea modificar una máquina ya creada, se puede hacer con el
`virsh edit nombre-MV`, por ejemplo, para indicar (dentro de *devices*, y la mac se puede omitir):

```
<interface type='bridge'>
   <mac address='52:54:00:1e:50:1a'/>
   <source bridge='br0'/>
   <model type='virtio'/>
   <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
```

Libvirt dispone de muchas opciones para la configuración de la red, además de las mostradas, como asignar una tarjeta directamente a un *guest*, hacer una red *routed*, una red NAT específica o múltiples redes como se puede consultar en [Vnh].

La figura siguiente muestra la conexión desde una máquina externa al servidor (Ubuntu sysubu) y luego al *guest* (Debian DebBr0).



2.1.3. Creación de máquinas virtuales (*guests*)

Es un procedimiento que se realiza básicamente con la GUI, como se ha visto anteriormente, o con CLI (`virt-install`).

Como prueba de concepto se instalará una distribución Windows por lo cual se debe contar con la ISO o el DVD de la distribución. En este ejemplo se dispone del DVD de Windows 8.1 y se utilizará este, pero es el mismo procedimiento si se dispone del DVD de Windows 10. Por ello se crea la imagen con:

```
dd if=/dev/dvd of=/var/lib/libvirt/images/win.iso
```

Luego se utiliza el comando `virsh-install` para instalar la máquina virtual (es muy potente y se deben consultar las opciones que dispone) introduciendo en una única línea (antes de cada opción indicar '- -'):

```
virt-install --name=win --ram=2000 --cpu=host --vcpus=2
--os-type=windows
--os-variant=win8.1
--disk
/var/lib/libvirt/images/w8.qcow2,bus=sata,size=20,format=qcow2
--disk /var/lib/libvirt/images/win.iso,device=cdrom,bus=ide --network bridge=virbr0
--graphics vnc,listen=0.0.0.0
```

Se pueden utilizar las opciones - *-acelerate* y mejora mucho buscando los *virtio drivers* que se pueden bajar y compilar u obtener la iso para la distribución deseada.

En Ubuntu, se bajaría la última (virtio-win-drivers-xxxx-yyyy.iso) y cambiando en la creación la opción del disco poniendo *bus=virtio* y proseguir con la instalación, cuando la instalación se atasque en la página de almacenamiento, debemos cargar los drivers scsi del DVD. Dentro de `virsh` se ejecuta:

Se obtiene el dominio ID:

```
virsh# list
```

Unidades cargadas:

```
virsh# domblklist ID
```

Se extrae el DVD:

```
virsh# change-media win81 hdc --eject
```

Se inserta el nuevo:

```
virsh# change-media win81 hdc /var/lib/libvirt/images/virtio-win-0.1-81.iso --insert
```

Se verán tres drives (*network*, *scsi* y *balloon driver*), escoger scsi driver.

Ahora se deberá volver a poner el disco de instalación:

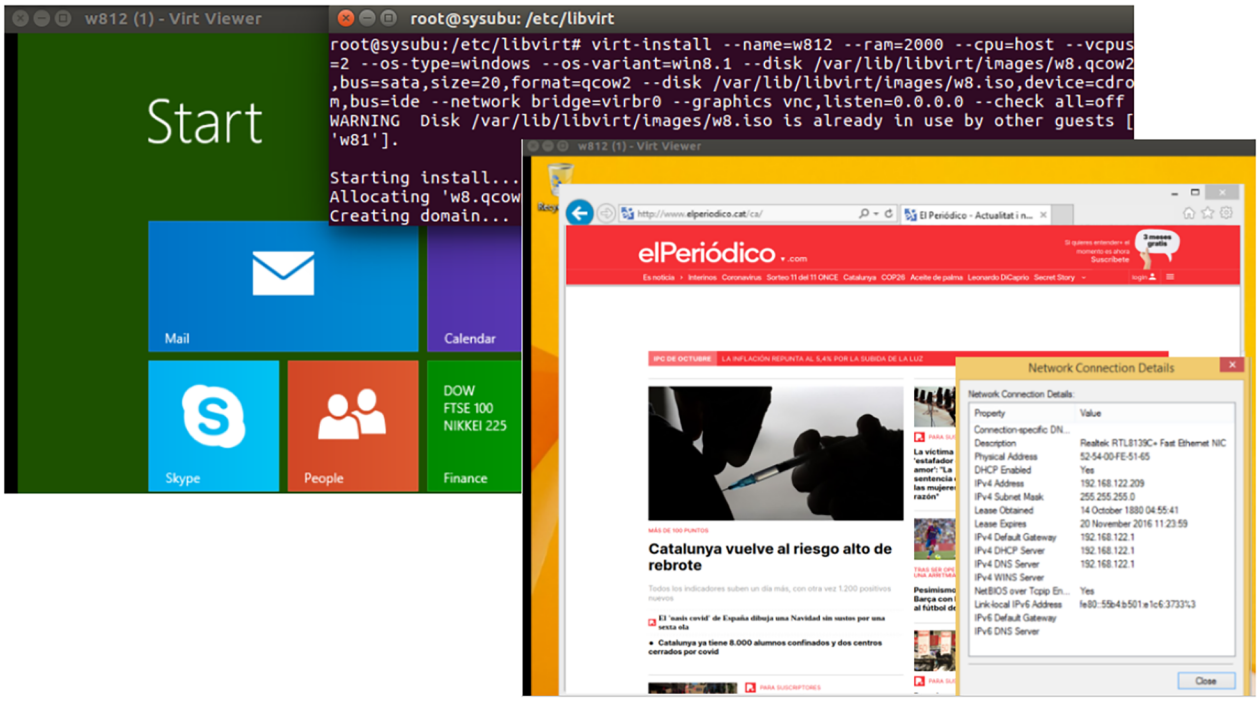
```
virsh# change-media win81 hdc /var/lib/libvirt/images/virtio-win-0.1-81.iso --eject
```

Se inserta el de Windows:

```
virsh # change-media win81 hdc /var/lib/libvirt/images/win8.iso --insert
```

Debería repetirse lo mismo para actualizar el driver de red.

Las dos figuras siguientes muestran la ejecución de W8.1 sobre Ubuntu y las configuraciones de red (en este caso se ha utilizado NAT).



Otros **comandos útiles con virsh** (CLI):

Para conectarse:

```
virsh --connect qemu:///system
```

Listar las máquinas:

```
virsh# list --all
```

Iniciar la máquina Windows (también se puede utilizar *suspend, resume, shutdown, destroy* –apaga MV por la fuerza– para gestionar las MV):

```
virsh# start win81
```

Desde otra terminal se puede acceder a su interfaz con:

```
virt-viewer --connect qemu:///system win81
```

Desde otra máquina:

```
virt-viewer --connect qemu+ssh://ip_servidor/system win81
```

Para clonar una MV desde original a copia:

```
virt-clone --connect qemu:///system -o original -n copia -f /var/lib/libvirt/copia.qcow2
```

La configuración de cada máquina virtual se almacena en un archivo XML en */etc/libvirt/qemu/* con el nombre de la máquina; se puede modificar primero exportando el archivo de la MV (por ejemplo, nteum):

```
virsh dumpxml nteum > /tmp/nteum.xml
```

Editar el archivo modificando las opciones (CPU, RAM, discos, etc.) y luego importarlo

```
virsh define /tmp/nteum.xml
```

Otros **comandos útiles**
(paquetes `apt-get -y install libguestfs-tools virt-top`) [Ksw]:

```
virt-ls -l -a /var/lib/libvirt/images/debina8.qcow2
virt-ls -l -d Ubuntu20.04
virt-cat -a /var/lib/libvirt/images/debina8.qcow2 /etc/passwd
virt-edit -d Ubuntu20.04 /etc/fstab (la máquina apagada para evitar incoherencias en el disco)
virt-df -d Ubuntu20.04
virt-top
guestmount -d ubuntu -i /mnt; ll /mnt (montar el disco de la VM)
```

Para hacer una **migración en «vivo»** desde un servidor KVM a otro se requiere que ambos compartan un disco con las imágenes de la MV (por ejemplo, en este caso, por NFS, pero puede ser iSCSI o GlusterFS). Los servidores en este ejemplo son *syskvm1.nteum.org* y *syskvm2.nteum.org* y además *nfs.nteum.org*, que está montado sobre los dos anteriores en */var/lib/libvirt/images*. En este ejemplo se migrará la máquina Ubuntu20.04 desde *syskvm1 a syskvm2*. Para ello se debe ejecutar:

```
virsh migrate --live Ubuntu20.04 qemu+ssh://syskvm2.nteum.org/system
```

La ejecución pedirá el *passwd* de root de *syskvm2* y moverá la máquina de un entorno a otro, lo cual se podrá verificar con `virsh list` en cada máquina (sobre *syskvm1* se verá el mensaje *just migrated* y sobre *syskvm2* la máquina en ejecución).

KVM también puede hacer **migración del almacenamiento** cuando migra una máquina virtual con la ventaja de que no es necesario tener un disco compartido como en el caso anterior. Para ello solo con los dos servidores (*syskvm1* y *syskvm2*) ya hay suficiente. Para ello, primero se debe generar el mismo espacio en el destino que en el origen ejecutando sobre *syskvm1*.

```
ll /var/lib/libvirt/images/deb*
-rw------- 1 root 3221946368 dic 2 12:28 debian8.img
```

Sobre *syskvm2* se crea el espacio para alojar esta imagen:

```
fallocate -l 3221946368 /var/lib/libvirt/images/debian8.img
```

Se verifica sobre *syskvm2* que se ha creado:

```
ll /var/lib/libvirt/images/
-rw------- 1 root root 3221946368 dic 2 12:34 debian8.img
```

Ya desde *syskvm1* se puede hacer la migración:

```
virsh migrate --live --copy-storage-all debian8 qemu+ssh://syskvm2.nteum.org/system
```

Pedirá el *passwd* de root sobre syskvm2 y se podrá comprobar la migración. Si se desea revertir y volver la máquina a *syskm1* solo es necesario ejecutar una migración en «vivo» solamente:

```
virsh migrate --live debian8 qemu+ssh://syskvm1.nteum.org/system
```

2.1.4. Virtualización anidada (*Nested KVM*)

En la virtualización el *host* Linux físico (*bare-metal*) tiene instalado KVM como hipervisor y ejecuta varios sistemas operativos. Si se analizan las máquinas virtuales por defecto, estas no disponen de las extensiones hardware del procesador, por lo cual no se podría instalar otro hipervisor que lo requiriera sobre esta MV; se puede comprobar ejecutando:

```
grep --color vmx /proc/cpuinfo
```

Sobre una máquina que las disponga se verá la palabra **vmx** de color, mientras que si no están activas no se verá nada. En ciertas ocasiones, es útil disponer de estas extensiones sobre las máquinas virtualizadas y KVM permite lo que se denomina **virtualización anidada** (*nested virtualization*), lo cual habilita a ejecutar un *guest* dentro de un hipervisor virtualizado que se está ejecutando sobre el hipervisor base (*bare-metal*). Esto es útil cuando en un servidor se presta servicio a diferentes usuarios que, a su vez, desean tener diferentes MV (por ejemplo, en el *cloud*) o cuando se desea probar o dar servicio de diferentes hipervisores sobre un mismo hardware, o cuando se desea depurar/analizar diferentes configuraciones sobre un determinado hipervisor. En las últimas versiones de KVM este soporte viene activado por defecto, pero se puede comprobar con:

```
cat /sys/module/kvm_intel/parameters/nested
```

Se verá si la respuesta es **Y** (está activado) y en */etc/modprobe.d/qemu-system-x86.conf* se podrá observar que hay una opción *options kvm_intel nested=1*. Si la respuesta es **N** (desactivado) se puede activar con:

```
echo 'options kvm_intel nested=1' >> /etc/modprobe.d/qemu-system-x86.conf
```

Y luego reiniciar el sistema. Para modificar una MV (por ejemplo, la máquina Ubuntu20.04) que vea las extensiones hardware se ejecuta (en el directorio */etc/libvirt/qemu* se pueden consultar los nombres de las máquinas ya que es donde se encuentra un archivo con su nombre que contiene las definiciones XML de cada una de ellas):

```
virsh edit Ubuntu20.04
```

Y se cambia *cpu mode='host-passthrough'* (por defecto, se configuran las MV con *cpu mode='custom'*). Una vez arrancada, se podrá ver que la máquina ahora dispone de la extensión **vmx** cuando se mira */proc/cpuinfo*.

2.1.5. Administración remota

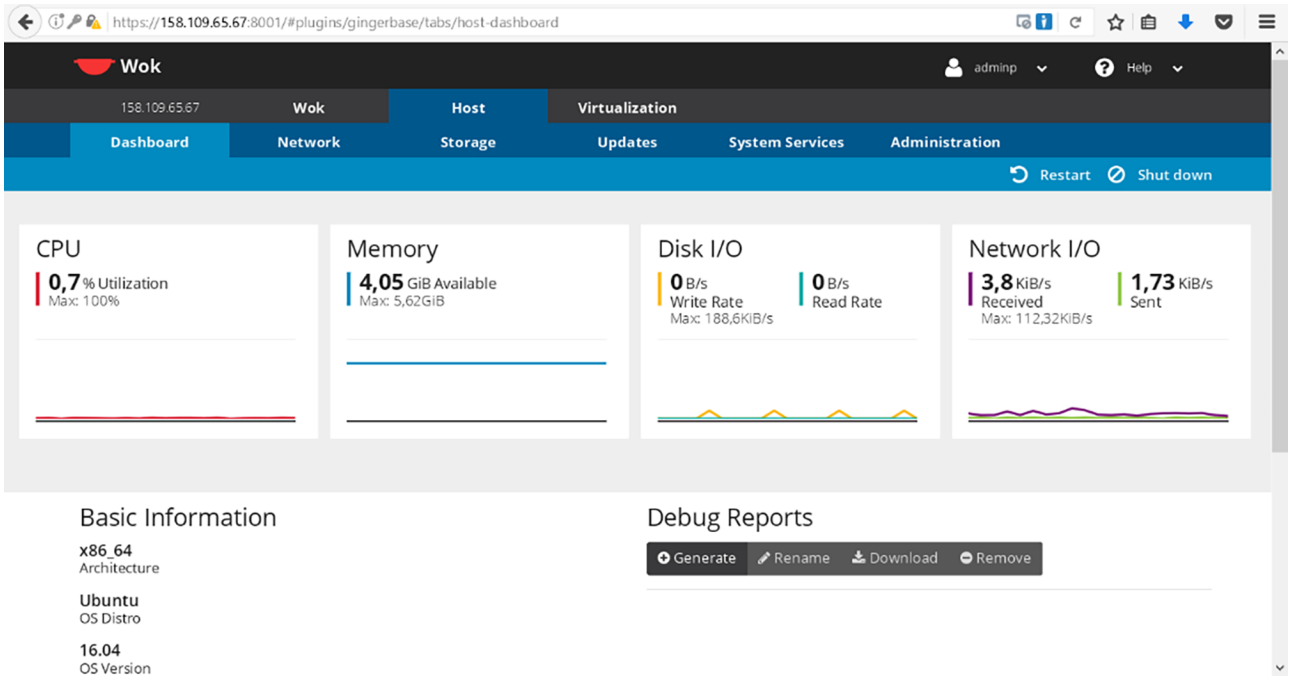
Existe una gran cantidad de herramientas (además de la CLI y `virt-manager`) para gestionar arquitecturas basadas en KVM (si bien muchas de ellas también pueden gestionar la infraestructura IaaS completa, otras ya están obsoletas). Los usuarios que deseen hacer una experiencia con este tipo de herramientas pueden realizar una experiencia con Kimchi.

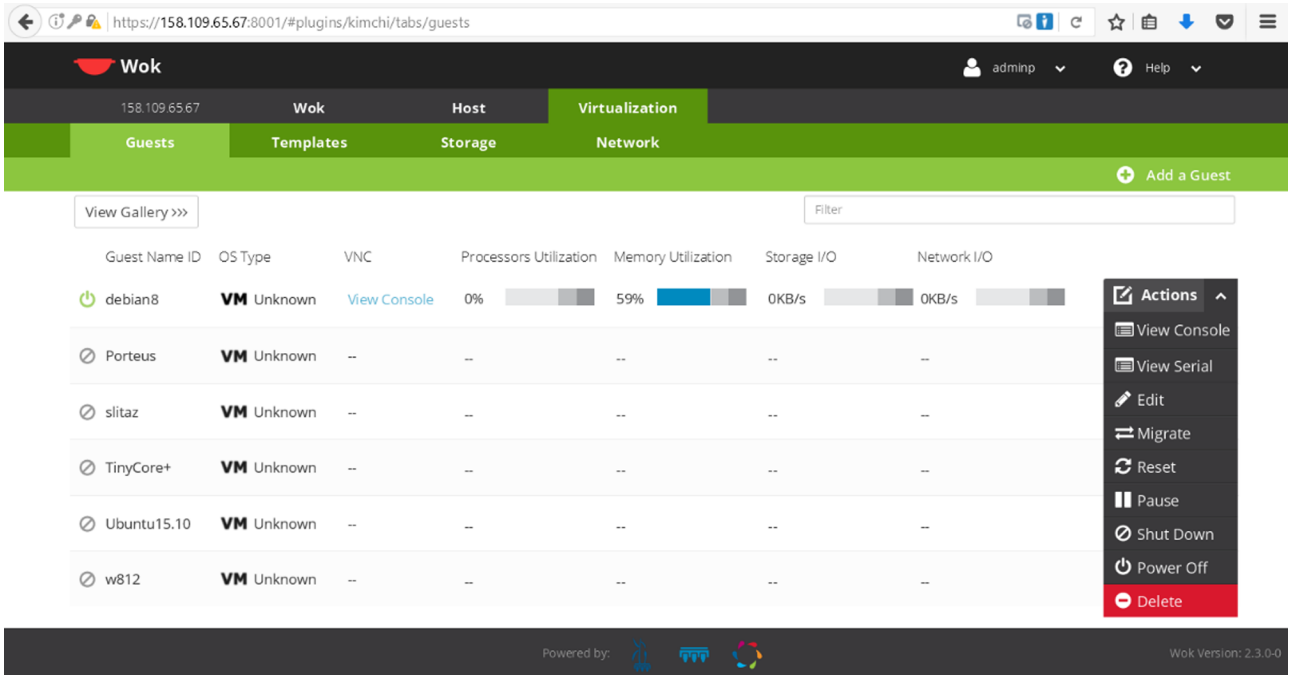
Kimchi es una aplicación excelente para la administración y monitorización de KVM remota en HTML5 (así como del propio *host*). Los objetivos de diseño son que la gestión tanto del *host* como de KVM sean simples y fáciles y desde cualquier lugar. Kimchi se ejecuta como un *plugin* de Wok y maneja los *guests* KVM a través de libvirt. A la interfaz de gestión se accede mediante un navegador que soporte HTML5. La instalación requiere un poco de atención pero es factible siguiendo los pasos indicados en la página web del proyecto que disponen de los paquetes en diferentes distribuciones ya compilados (primero instalar Wok y después Kimchi).

Si el sistema sobre el que se está instalando no tiene todas las dependencias necesarias, fallará; se deberá ejecutar (también se pueden descargar las dependencias para cada uno de los paquetes ejecutando `apt-get install -f`). También se debe recordar que después de instalar se deberá reiniciar el servicio con `systemctl restart wokd.service`.

Para acceder al entorno se deberá poner en la URL de navegador *<https://machine-ip o machine.name.domain:8001>* donde deberá introducir el usuario Linux y el *passwd* de Linux para este usuario. Si esta operación se realiza una vez instalado Wok solo mostrará las propiedades del *host* y permitirá administrar todos sus recursos y cuando se haya instalado Kimchi (y reiniciado el servicio)

se podrá ver tanto la parte del *host* como la parte de KVM y gestionar todas las máquinas que estén bajo su dominio. Este paquete es de muy alta calidad, su visualización es muy buena a igual que su estabilidad y es recomendable en entornos tanto medianos como grandes. Los paquetes están en desarrollo activo (existen actualizaciones en el 2020) y se puede encontrar más información en la web del proyecto. En las figuras siguientes se muestra el *dashboard* (carga, recursos, configuración del *host*) y, en la segunda, el entorno de virtualización con sus cargas y recursos sobre KVM+libvirt.





Otra opción simple es acceder a la máquina remotamente (y si no se desean instalar clientes VNC) o se puede acceder desde un dispositivo móvil; es posible acceder a ella desde un simple navegador utilizando noVNC; noVNC es un cliente VNC basado en un *browser* desarrollado sobre HTML5-Canvas/WebSockets y que funciona tanto sobre un servidor VNC que soporte WebSockets (tal como `x11vnc/libvncserver`) o puede utilizar el propio *websockify* que hará de *bridge* ente el navegador y el VNC server. NoVNC soporta los navegadores habituales (incluido los de iOS, Android), diferentes codificaciones de VNC (*raw, copyrect, rre, hextile, tight, tightPNG*), WebSocket SSL/TLS encryption (por ejemplo, wss://), redimensionamiento de la pantalla de acuerdo al tamaño del cliente (navegador), cursor remoto o local, copiar y pegar, desplazamientos verticales y horizontales, entre otras características. Para instalarlo simplemente en la máquina, descargar la última versión (sobre Ubuntu está como paquete, pero no incluye la última versión):

```
git clone https://github.com/novnc/noVNC.git
```

En este caso, se utilizará el servidor por defecto de Ubuntu, que es `vino`, y que no soporta *WebSockets* por lo cual se utilizará el propio (*websockify*), pero antes debemos deshabilitar la encriptación de `vino` (las propiedades se pueden modificar con `vino-preferences` y, para iniciarlo sobre Ubuntu simplemente `/usr/lib/vino-vino-server`), ya que el algoritmo implementado por este no es compatible con el noVNC; para ello se debe ejecutar desde un terminal (del usuario que tiene el entorno):

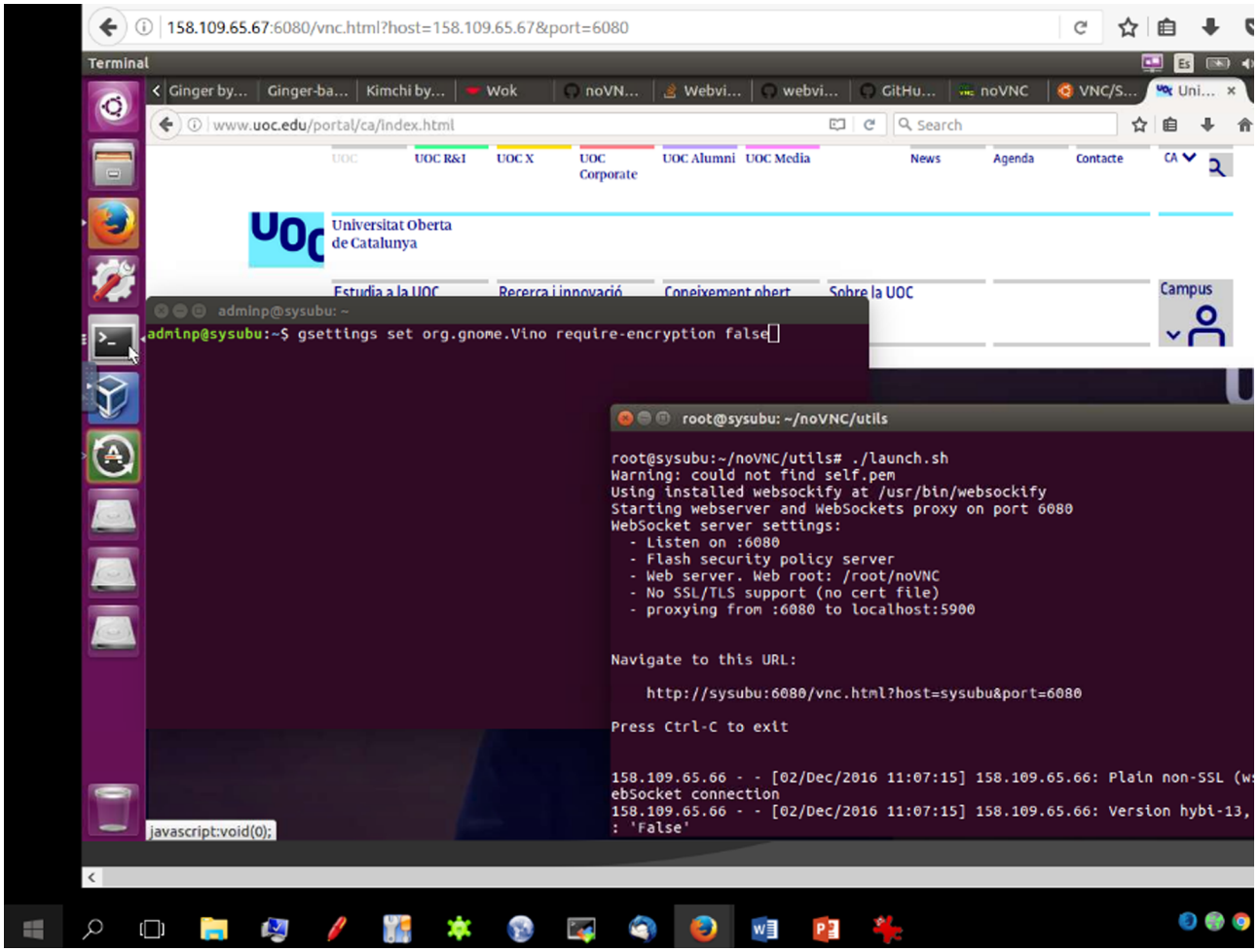
```
gsettings set org.gnome.Vino require-encryption false
```

Luego se ejecuta un *script* que pone en marcha el servidor (y provee la URL de conexión; cabe mirar los parámetros del *script* ya que se pueden cambiar los puertos y otras configuraciones).

```
cd ./noVNC/utils
./launch.sh
```

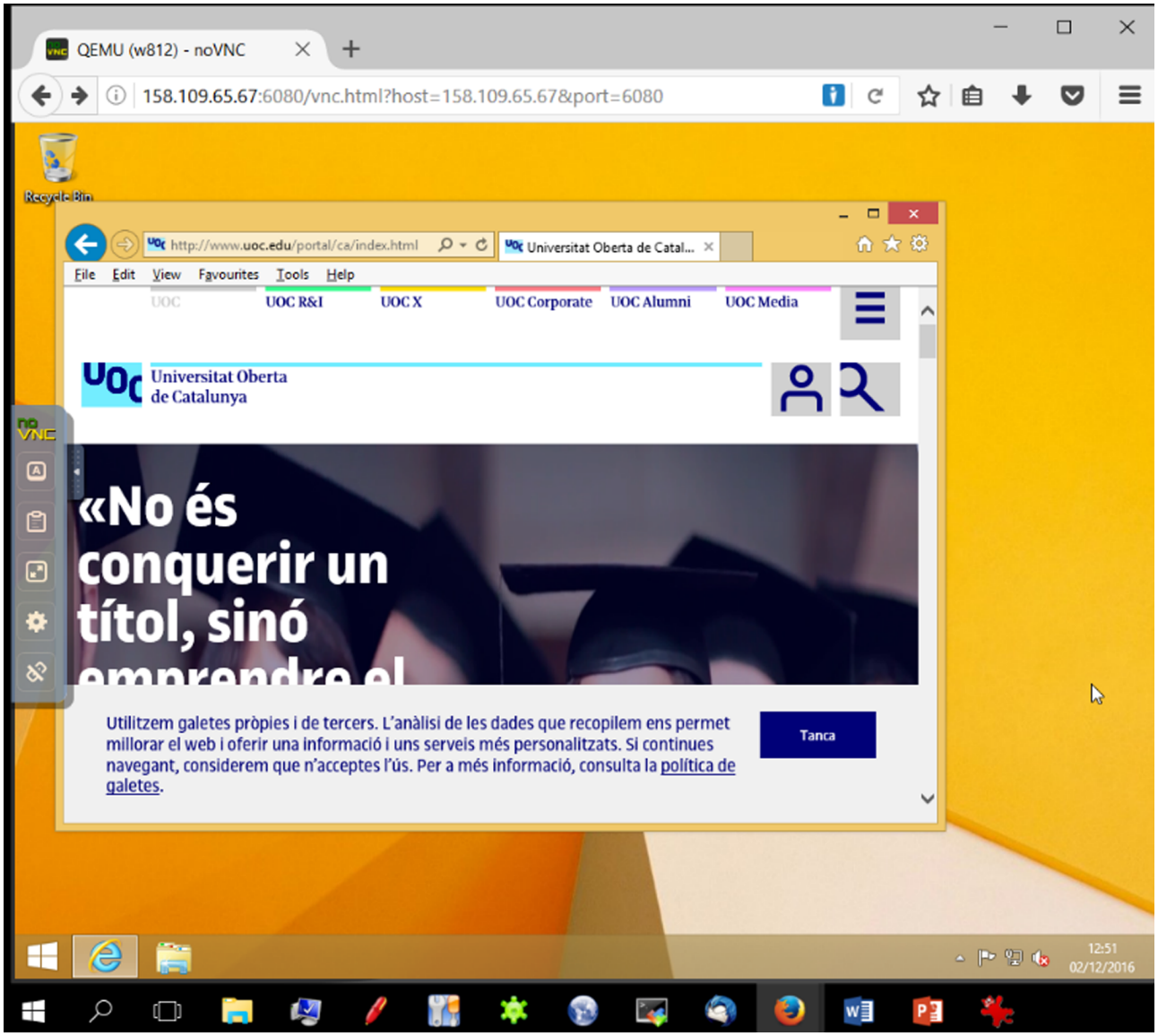
Dado que nuestro interés es acceder remotamente, ejecutamos (ver la figura siguiente donde se muestra una conexión entre una máquina W10 y el servidor Ubuntu):

http://158.109.65.67:6080/vnc.html?host=158.109.65.67&port=6080



También se puede ejecutar una MV con KVM y al *guest* activarle el VNC server teniendo en cuenta que, si se desea conectar una máquina remota, es necesario cambiar las configuraciones de */etc/libvirt/qemu.conf*: `vnc_listen = "0.0.0.0"`

Aunque debe tener cuidado con ello, ya que cualquier cliente se podrá conectar al *guest* remotamente y es necesario considerar aspectos de seguridad de `libvirt`. Finalmente, reiniciando `libvirt` y poniendo en marcha el *guest* se podrá acceder como anteriormente a través de noVNC. La figura siguiente muestra la conexión con un *guest* W8.1 desde un W10 utilizando noVNC sobre un navegador Firefox.



También es posible conectarse a los *guest* a través de un Spice [Spi], seleccionando este en la MV y utilizando el comando `remote-viewer`, indicándole la url *spice://localhost:5900*. Si no se dispone de este comando, instalar el paquete `virt-viewer` (`apt-get install virt-viewer`). Al igual que para VNC, si se desea conectar remotamente a un *guest* por Spice, es necesario modificar la configuración de */etc/libvirt/qemu.conf* para quitar el comentario a *spice_listen = 0.0.0.0* reiniciando el servicio y la conexión.

En [Ksw] se pueden encontrar referencias adicionales sobre la instalación de Spice *server*/clientes.

2.2. VirtualBox

Oracle VM VirtualBox es una infraestructura *free & open source* de virtualización (tipo 2) para x86/64, creado por la empresa Innotek GmbH (adquirida en 2008 por Sun Microsystems, la cual fue adquirida por Oracle en 2010) y que puede ser instalado sobre diversos *hosts* (Linux, OS X, Windows, OpenSolaris,

FreeBSD y Genode, entre otros). Soporta la creación y gestión de MV pudiendo utilizar como *guest* diferentes SO (Linux, Windows, BSD, OS/2, Solaris, Haiku y OSx86, entre otros) y utiliza (para algunos SO) un paquete adicional (en muchas distribuciones de Linux ya está incluido en el repositorio, por ejemplo, en el de *non-free* y en otras en el *contribution*) llamado *Guest Additions* con controladores que mejoran las prestaciones, funcionalidades y los gráficos.

A partir de la versión 4.0, el núcleo de VBox es GPLv2, pero el *Oracle VM VirtualBox extension pack* (para soporte USB 2.0/3.0, RDP, PXE) tiene licencia propietaria *Personal Use and Evaluation License* (PUEL) que permite el uso del software para uso personal, educación o evaluación sin cargo. Para su gestión dispone de una GUI, pero todo se puede hacer desde CLI a través del comando `VBoxManage` (incluso algunas opciones/configuraciones solo desde este).

Como formato de disco utiliza VDI (VBox Disk Image), VMDK (Virtual Machine Disk), VHD (Virtual HD), HDD (Parallels HD), QDE (Qemu Enhanced Disk), QCOW (Qemu Copy-on-write) y OVF (para exportar e importar *appliances*) y dispone de comandos para cambiar entre formatos, permite montar ISO (tanto de unidades virtuales ópticas como físicas de CD/DVD) y soporta aceleración en 3D, 32 vCPU (CPU virtuales), dispositivos IDE, SATA, SCSI, o conexión a iSCSI, soporte ACPI, pantalla completa, cuatro tarjetas de Ethernet (o 36 si se utiliza CLI), USB, integración con teclado/ratón y admite para cada máquina virtual que pueda ser configurada mediante *software-based virtualization* (en este modo soporta *guests* de 32 bits ejecutándose en *rings* 0 y 3 de la arquitectura *ring* de Intel) o *hardware assisted virtualization* (si se dispone de las extensiones hardware) y desde la versión 6 soporta virtualización anidada (es decir, las MV pueden ver las extensiones hardware del procesador).

En cuanto a la gestión de máquinas permite realizar *snapshots* (congela el estado de la MV y es posible volver a la configuración anterior), soporta *Remote Desktop Extension* –VRDE– (para conexión remota por RDP con mapeo del USB local, USB *over* RDP), configuración de las secuencias de teclas sobre el *guest* (por ejemplo, Crtl+Alt+Del), exportar e importar MV en formato OVF1.0/2.0, permite agrupaciones para aplicar comandos a todas las MV del grupo (iniciar, pausar, reiniciar, salvar estado, enviar señal de apagado, apagar, descartar estado salvado, mostrar en sistema de archivo, ordenar), cifrado mediante AES, dispone de un administrador de medios virtuales, y con el *host* puede: compartir carpetas, USB, portapapeles, y *Drag & Drop*.

En aspectos de red soporta: NAT, *Network Address Translation* (permite utilizar el NIC del *host* creando un *router*, por defecto en 10.0.2.0/24, pero se puede cambiar con CLI, y reenvía los paquetes por el *host*, por lo cual si el *host* tiene conexión la MV también lo tendrá); NetWork NAT (funciona como un *router* doméstico, donde las máquinas que estén en esta red se podrán conectar entre

sí); *bridged* (IP propia en la red del *host*, la máquina será visible igual que el *host*); red interna (red aislada a través de un *switch* virtual); y *host-only* (red interna pero que compartirá con el *host*).

También dispone de un modo *Generic Driver*, incluye otros *drivers* que vengan con VirtualBox o en un paquete de extensión y donde hay dos modos disponibles: UDP *Tunnel* (permite interconectar las máquinas virtuales que se ejecutan en distintos *hosts* de forma directa, fácil y transparente, sobre la infraestructura de red existente) y VDE, *Virtual Distributed Ethernet* (para conectarse a un switch Virtual Distributed Ethernet en un *host* Linux o FreeBSD) [Vir].

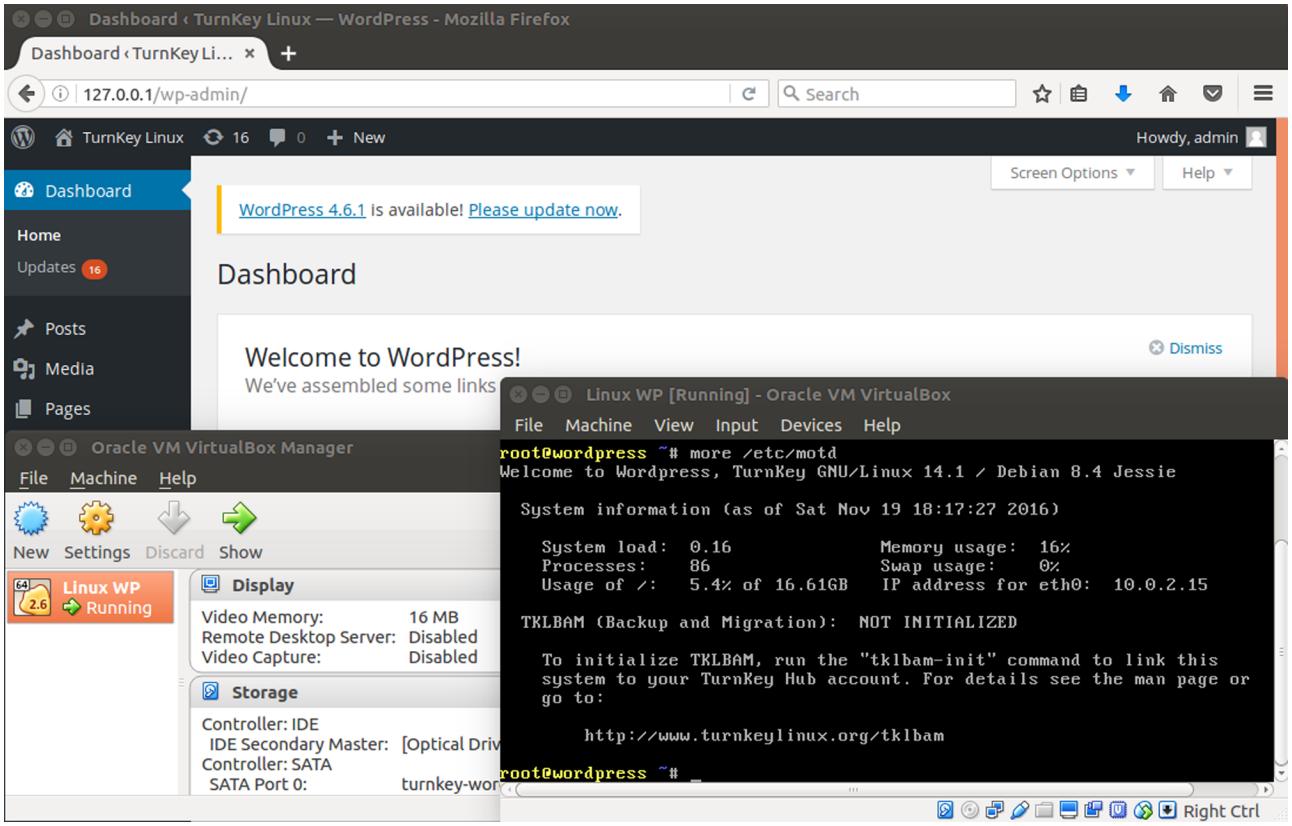
Tened en cuenta que, si se desea instalar VirtualBox manteniendo KVM, se recomienda quitar (temporalmente) los módulos kvm y kvm_intel ya que podrían interactuar entre ellos y generar algún tipo de error (aunque en las distribuciones actuales, p. ej. Ubuntu20.04, es posible ejecutar VirtualBox teniendo los módulos KVM instalados sin problemas).

2.2.1. Instalación y configuración

Su instalación se realizará sobre una máquina con procesador Intel i7 620M (64 bits, 2 *cores* + *Hyper-Threading* = 4 procesos) y sobre el SO Ubuntu. El repositorio de Ubuntu dispone de una versión (6.1.26 para Ubuntu 21.10) que se puede ver con `apt-get search virtualbox | grep ^virtualbox` en cualquier distribución y con `apt-get install virtualbox` se podría instalar directamente. Pero si se desea la última versión (6.1.28 en el momento de escribir este documento) se pueden bajar del repositorio seleccionando la distribución, versión y arquitectura que se desee. Para instalarlo se deben seguir las indicaciones de los desarrolladores pero básicamente es:

```
dpkg -i virtualbox-xxx-yyy_amd64.deb
```

En caso que dé problemas de dependencias estas se pueden resolver fácilmente con `apt install -f` o si no es posible instalando los paquetes indicados. La figura siguiente muestra una imagen de la interfaz del VBoxManager (abajo izquierda), la MV (abajo derecha) ejecutando una appliance de WordPress (disco en formato vmdk) y un navegador sobre el *host* visualizando una sesión de WordPress (para ello la MV tiene NAT en el adaptador de red y se ha hecho un *port forwarding* de puerto 80 de *host* al puerto 80 del *guest*).



2.2.2. Administración y ejecución remota

Una característica de VirtualBox es que incluye una API que permite su gestión y administración remota. Existen diversos paquetes y se comentarán a continuación algunas de estas opciones.

RemoteBox es un cliente de VirtualBox que puede administrar de forma remota (es decir, a través de la red) una instalación de VirtualBox en un servidor, incluidas sus máquinas virtuales (*guests*), e interactuar con ellos como si se estuvieran ejecutando localmente. Funciona en modo cliente-servidor donde VirtualBox se encuentra instalado en la máquina «servidor» y RemoteBox se ejecuta en la máquina «cliente» proporcionando una interfaz gráfica completa (basada en GTK2) con una apariencia muy similar a la de la GUI nativa de VirtualBox y que permite controlar todo el entorno de forma remota.

El funcionamiento es seguro ya que se accede a través de la API de VirtualBox (a través de un protocolo SOAP) permitiendo acceder de forma estándar y segura independientemente de la versión de VirtualBox. Para acceder a las MV se puede utilizar el protocolo de escritorio remoto (RDP) y RemoteBox usa un cliente RDP para mostrar la pantalla de un invitado, localmente en la máquina cliente y completamente interactivo. RemoteBox se ejecuta en Linux, Solaris, Mac OS X, Windows y versiones actuales de BSD y permite su conexión con

VirtualBox en cualquiera de los SO en los cuales se ejecute (Linux, Mac OS X, Windows...) por lo cual se recomienda como herramienta de gestión remota de VirtualBox.

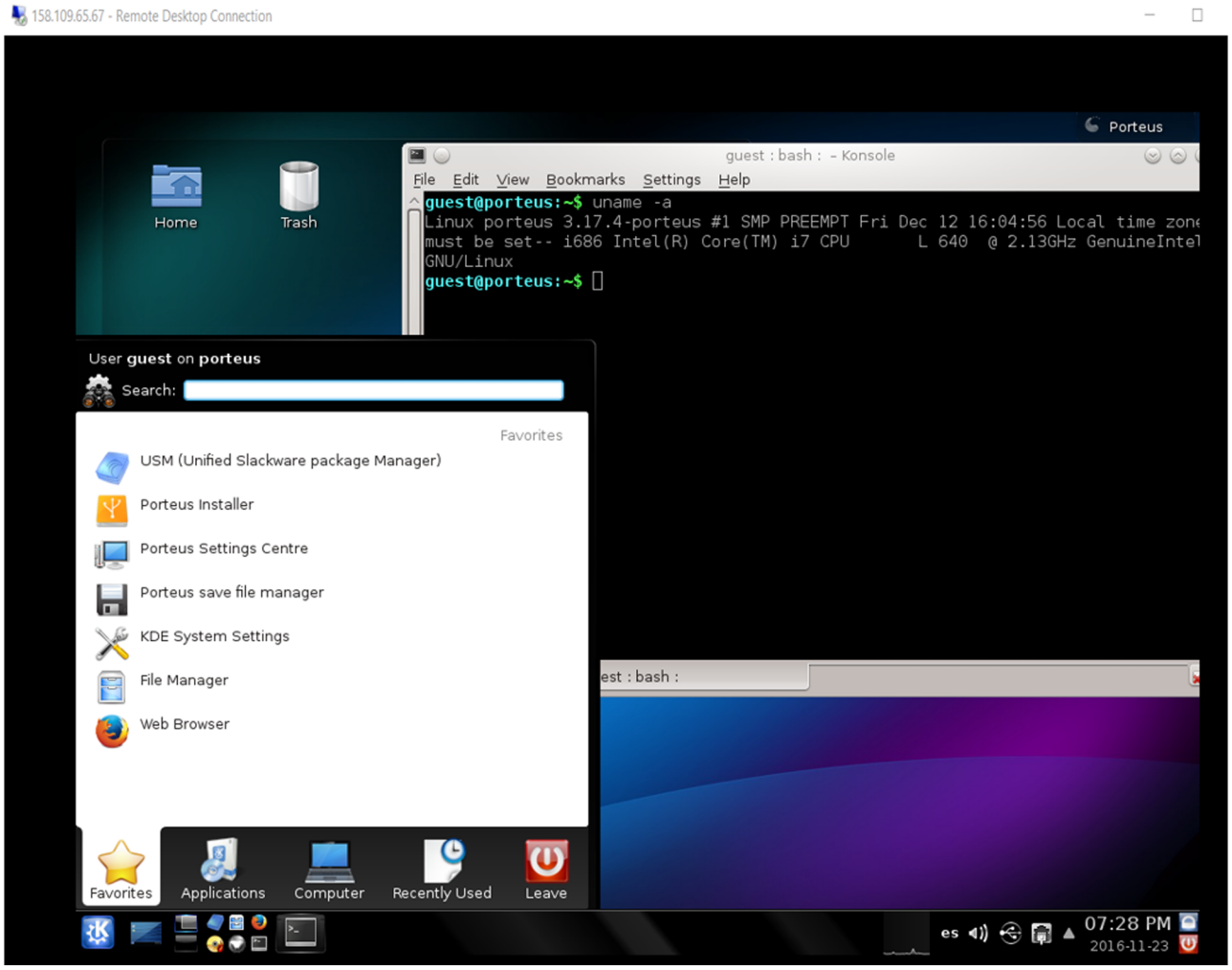
Otro paquete interesante como gestores de infraestructura es Hyperbox, que es una alternativa *open-source* a productos comerciales como *VMware vCenter/ESXi* y *Citrix XenCenter/XenServer,* basado en una arquitectura de cliente-servidor y que se integra muy bien con VirtualBox (y con otros hipervisores según el autor).

Dentro de este apartado no se puede dejar de mencionar, si bien hoy ya es obsoleto, phpVirtualbox que fue una implementación muy popular de *open-source* AJAX de la interfaz de usuario de VirtualBox en PHP para acceder y controlar remotamente las instancias de VirtualBox. Si bien ha sido un software muy popular, en este momento se encuentra muy desactualizada y solo se debería utilizar en casos muy específicos, ya que depende mucho de la distribución de VirtualBox.

Si solo se desea acceder a las MV remotamente, la forma más simple es a través de una interfaz de *VirtualBox Remote Desktop Extension* (VRDE). Incluida en las extensiones, Oracle proporciona una implementación de *VirtualBox Remote Display Protocol* (VRDP) que, dado que no es *open source* (aunque es gratuito), se deben aceptar las licencias cuando se instala. VRDP es una extensión compatible con Microsoft *Remote Desktop Protocol* (RDP) y con ello se puede utilizar cualquier cliente RDP estándar para controlar la MV remota. Para instalar las extensiones, simplemente hay que acceder a la página web de Virtualbox y bajar el archivo de extensiones para la versión instalada (VirtualBox 5.1.10 Oracle VM VirtualBox *Extension Pack*), que ya lo abrirá VirtualBox y lo instalará (si no, bajar el archivo y agregarlo *File→Perferences→Extensions*). Cuando se crea al MV, el servidor VRDP está desactivado y puede activarse en el menú de *Display* de la MV o con:

```
VBoxManage modifyvm "nombre de la máquina virtual" --vrde on
```

De forma predeterminada, el servidor VRDP utiliza el puerto TCP 3389 y, si se desea utilizar más de un servidor, se deberán cambiar los puertos ya que el puerto solo puede utilizarse por un servidor a la vez (en la CLI con *--vrdeport* o en el menú de Display) o también se pueden utilizar rangos de puertos (consultar el manual). La figura siguiente muestra la ejecución de una MV con Linux Proteus por VRDP utilizando el cliente *Remote Desktop Display* sobre W10.



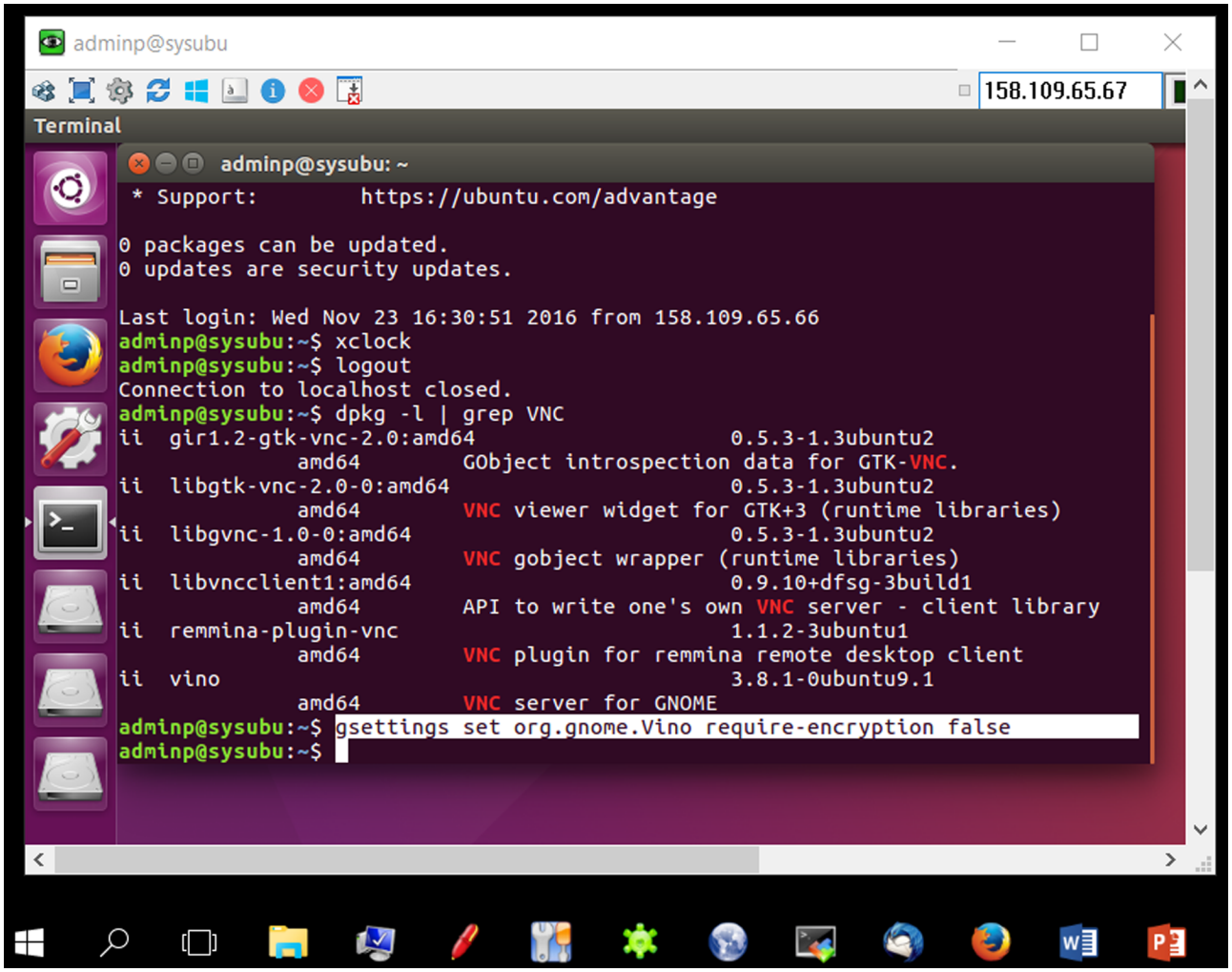
No obstante, y es válido en cualquier hipervisor, si la máquina tiene IP en la red (o bien hay una regla de *forward* en la máquina que hace de *gateway*), la forma más simple de conectarse a una MV es a través de `ssh -X ip_MV`, con lo cual se habilita con el parámetro -X un X11 *forwarding* mediante el cual se puede acceder al *display* remoto (consultar documentación del `ssh` sobre cuestiones de seguridad y la diferencia con el parámetro -Y) y las aplicaciones que se ejecuten remotamente se visualizarán en el *display* local (deberá ser *display* compatible X11). Para Windows se puede utilizar la aplicación (gratuita) MobaTerm que ya incluye un servidor de X11.

Otra opción, como ya se mencionó en el apartado de KVM, para conectarse y acceder a un escritorio remoto es a través de *Virtual Network Computing* (VNC). A igual que en la sección de KVM, se debe tener cuidado con el servidor/cliente que se utilice y las opciones, ya que en algunos casos la comunicación irá sin encriptar o ambos (servidor y cliente) deben soportar los algoritmos de encriptación para poder conectarse. Como ya se ha mencionado, en Ubuntu el servidor por defecto es `vino` (y corre en el puerto 5900 por defecto), el cual

utiliza un algoritmo de encriptación por defecto (TLS, *type 18*) y que no es soportado por los clientes habituales de Windows, pero se puede deshabilitar la encriptación mediante (desde el usuario que tiene abierta la pantalla)

```
gsettings set org.gnome.Vino require-encryption false
```

Se debe tener en cuenta que, con esta configuración, toda la comunicación va en claro por la red, por lo cual se recomienda utilizar un túnel `ssh` o cambiar a otro servidor. Sobre Linux existen gran cantidad de clientes (por ejemplo, Ubuntu) o uno que se utiliza habitualmente como Remmina y que está para la mayoría de distribuciones. La figura siguiente muestra UltraVNC sobre W10 y Servidor Vino-Ubuntu1 6.04 con la encriptación desactivada (opción **solo** para redes seguras).



Determinadas veces se desea tener en la misma máquina VirtualBox y KVM, pero según la versión del *kernel* (si es +5.8 no sale este error) se obtendrá un error (similar a *can't operate in VXM root mode*) si se desea ejecutar Virtualbox

con KVM instalado. Para ello, se deben desinstalar (temporalmente) los módulos kvm y kvm_intel para que se pueda arrancar las MV en Virtualbox. Para ello se comprueban primero los módulos:

```
lsmod | grep kvm
kvm_intel       172032 0
kvm          540672 1 kvm_intel
...
```

Luego se desinstalan con:

```
/sbin/rmmod kvm_intel
/sbin/rmmod kvm
```

Y ya se podrán ejecutar las máquinas virtuales. Cuando se desee nuevamente iniciar las máquinas en KVM, se deben cargar los módulos nuevamente con:

```
cd /lib/modules/'uname -r'/kernel/arch/x86/kvm/
insmod kvm.ko; insmod kvm-intel.ko
```

2.3. VMware Workstation Player

Es uno de los paquetes de virtualización producidos por VMware (empresa cuyo accionista mayoritario es la empresa EMC –83 % de las acciones– y que a su vez esta ha sido adquirida por Dell en 2016) para equipos x64 que utiliza como *host* Windows o Linux; es gratuito para uso personal, doméstico y no comercial. VMWPlayer puede ejecutar *appliances* existentes y crear y gestionar sus propias máquinas virtuales utilizando el mismo núcleo de virtualización que VMware Workstation Pro (no gratuito) pero con algunas limitaciones (probablemente la mayor es que en la versión Player solo deja ejecutar una máquina virtual por vez, restricción que no existe en la Pro) [Vwp].

Entre sus características permite *multiple-monitor display*, USB 3.0, soporte hasta 16 vCPU × 64 GB RAM, discos de hasta 8 TB, gráficos con aceleración, ejecución de MV cifradas, entre otras características.

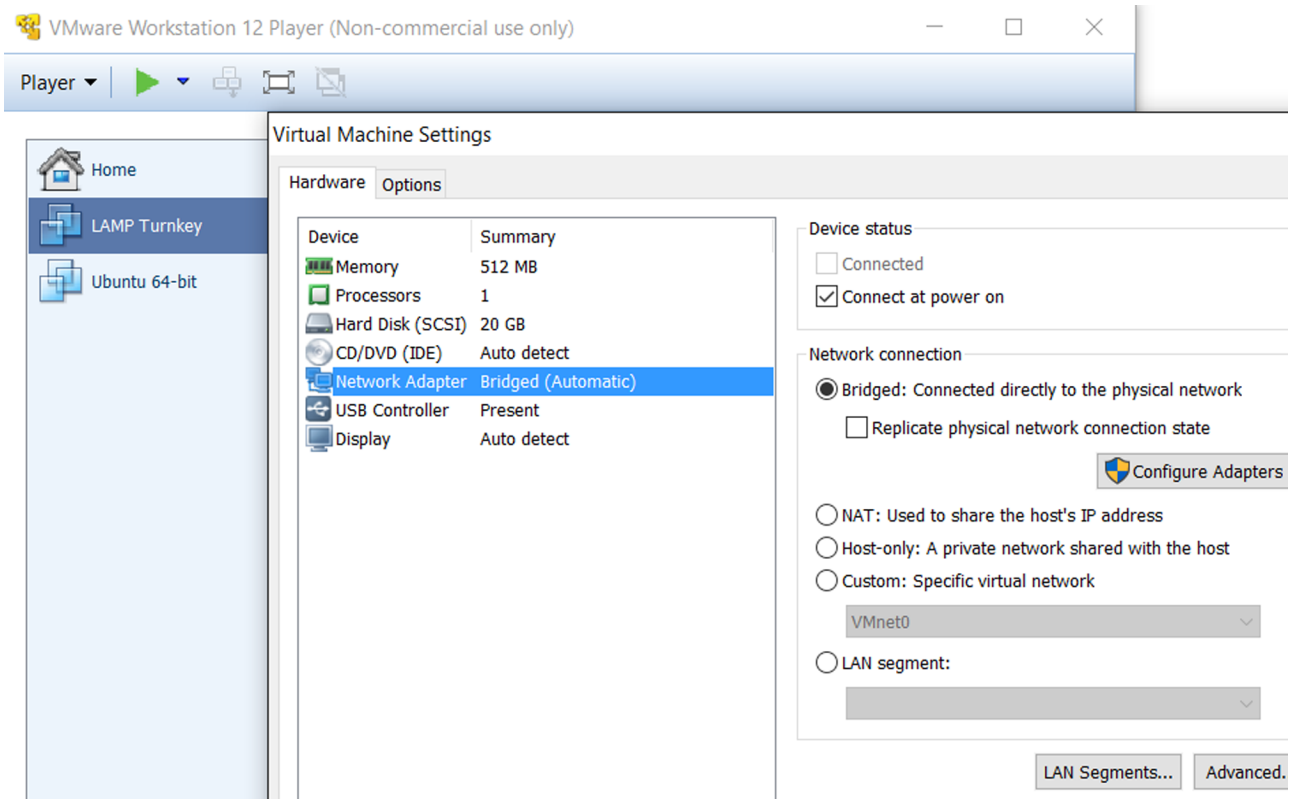
Su instalación es sumamente fácil tanto para Windows como para Linux (instalación) y los pasos para instalar una MV son los equivalentes a cualquier otra aplicación ya que puede hacerlo desde el CD/DVD o desde la ISO (también utiliza Crtl-Alt para pasar del *guest* al *host*). Cuando detecta que el sistema que se va a instalar es Linux, descarga un conjunto de herramientas (*VMware tools*) para mejorar el rendimiento, movimiento del ratón y gráficos. Sobre Linux (el paquete tiene extensión *.bundle*) se puede instalar sobre una consola o con un terminal gráfico (en este caso con un doble clic sobre el archivo es el gestor de archivos). El programa de instalación en Linux se puede poner en marcha con:

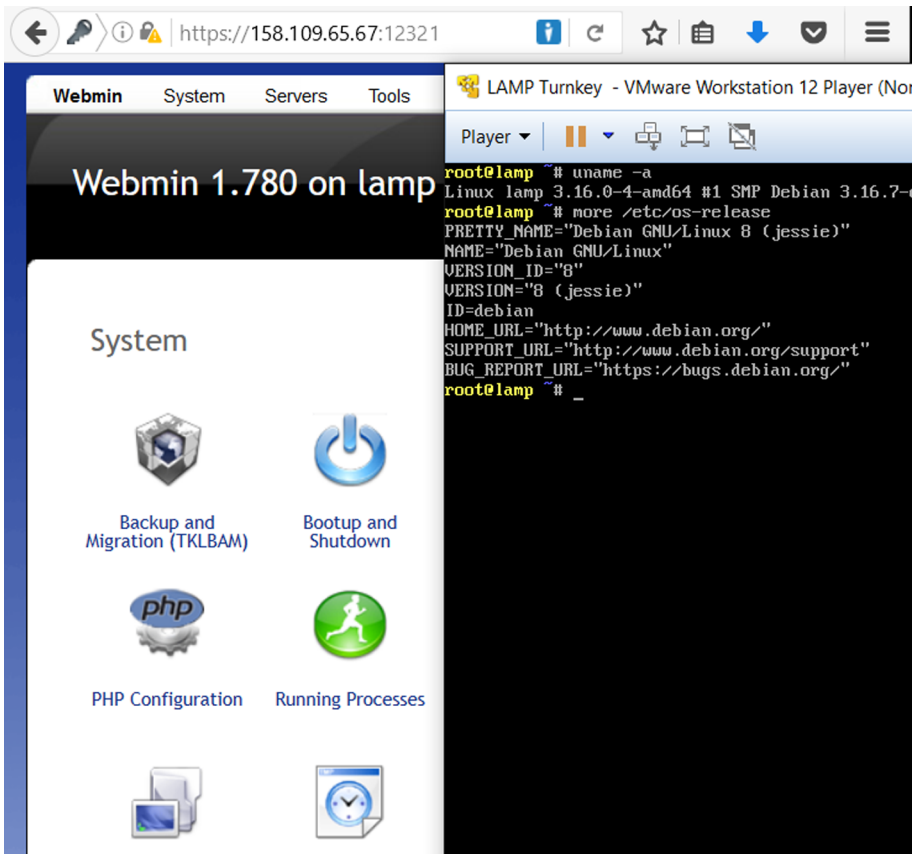
```
sh VMware-Player-xxx-yyy.x86_64.bundle --opción
```

Donde se debe reemplazar *xxx-yyy* por la versión adecuada (la arquitectura solo puede ser de 64 bits) y las opciones son *gtk* (interfaz gráfica), *console* (terminal texto) y *custom* (permite escoger diferentes opciones de directorio/límites).

Después de instalada la MV (Linux en este caso), y después de haber instalado *VMware tolos*, indicará que se haga clic en un botón de *Install Tools*, lo cual montará el CD/DVD (virtualizado) con el software. Después de montar el dispositivo (`sudo mount /dev/cdrom /media`) encontraremos el paquete VMware-Tools-xyz.tar.gz), que se podrá copiar al /tmp por ejemplo, extractar (`tar xzvf file.tgz`) y dentro del directorio ejecutar `vmware-install.pl`. Se debe tener en cuenta que algunas distribuciones (Ubuntu, por ejemplo) ya cuentan con estas herramientas como `open-vm-tools` y que solo se deben instalar.

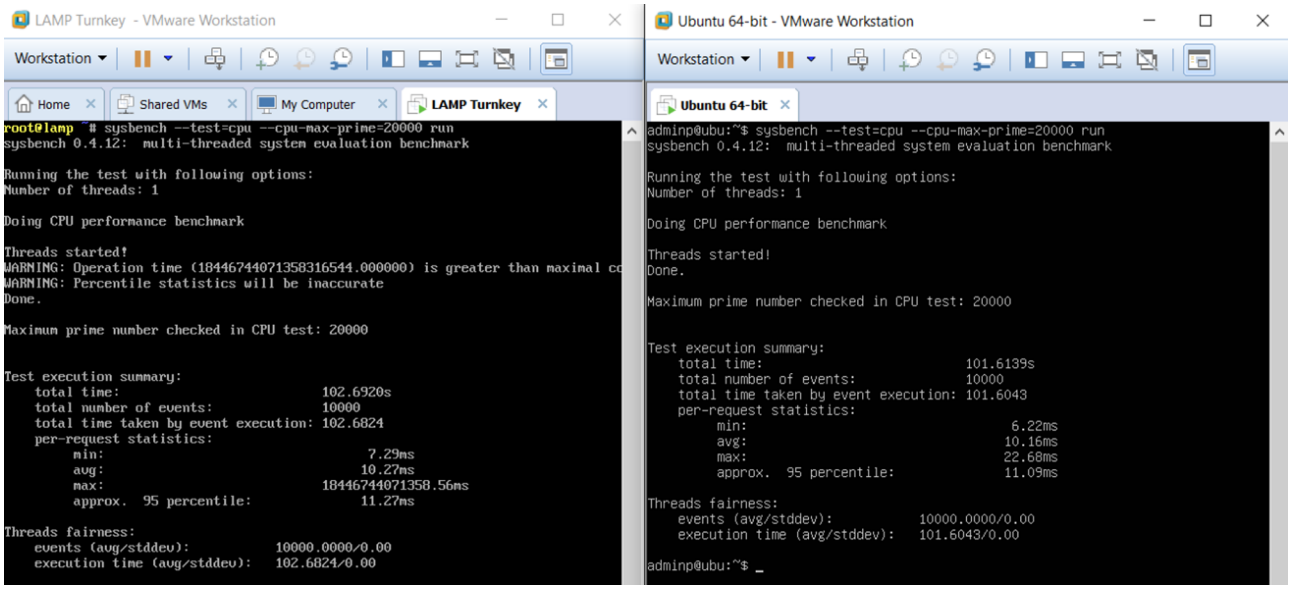
Las figuras a continuación muestran la interfaz de VMware WP con dos MV instaladas (y con las opciones habituales para configurar los recursos) y la ejecución de un *appliance* (*vmdk*) LAM (Turnkey) con la interfaz de webmin.





En la parte derecha se puede ver la MV sobre la consola del VMware WP, y en la izquierda, el acceso al MV desde una máquina remota y a la aplicación Webmin para administrar la *appliance*.

La instalación de VMware Workstation Pro (versión de pago pero permite una versión de prueba de 30 días) incorpora una serie de herramientas adicionales como editor de redes y permite ejecutar todas las VM que se desee y la conexión con otros productos VMware, entre otras opciones. La figura siguiente muestra dos MV ejecutando el *benchmark sysbench*.



Más allá de que VirtualBox es *open source*, como se ha podido observar, las opciones y posibilidades entre VirtualBox y VMware Workstation Player en sus últimas versiones son similares, excepto que en este último solo se puede abrir una MV solamente (limitación que no tienen los otros productos de VMware, pero es de pago). En cuanto a prestaciones, son similares, y se recomienda, sobre el hardware disponible, ejecutar diversos *benchmarks* (`sysbench` o también Phoronix, por ejemplo) para obtener cuál es la que mejores prestaciones obtiene (en muchos aspectos dependerá de los *drivers* utilizados y si se utilizan *drivers* paravirtualizados *virtio*) y en aspectos de facilidad y flexibilidad, según muchos expertos VirtualBox provee prestaciones y opciones similares a los producos de VMware con la ventaja de ser *open-source* (aunque no dispone de soporte empresarial).

2.4. Proxmox

Proxmox *Virtual Environment* es una solución (basada en Debian) *open source* (GPL) para la gestión de servidores virtualizados con QEMU/KVM y LXC que permite gestionar máquinas virtuales, contenedores, clústeres de alta disponibilidad, almacenamiento y redes con una interfaz web muy simple o a través de la CLI. Un aspecto importante en su diseño (*multi-master*) es que no es necesario un servidor de administración adicional ahorrando recursos y permitiendo la alta disponibilidad (HA), lo cual posibilita desplegar entornos de virtualización de clase empresarial en un centro de datos. Permite, además, múltiples fuentes de autenticación combinadas con la gestión de roles y permisos de usuario dando un control total al administrador del clúster virtualizado de HA y dispone, además, de una API web RESTful que permite la integración de herramientas de gestión de terceros [Pve].

Entre las principales características se pueden contar:

- **Tecnología**: soporta *guest* Linux y Windows (32/64 bits) e incorpora las últimas especificaciones de Intel/AMD para mejorar las prestaciones de las MV, soportando cargas de trabajo dentro de una empresa.

- **Administración**: contiene todas las herramientas necesarias para la gestión de una infraestructura virtualizada en un único entorno con API RESTful y con verificación automática de parámetros para reducir errores en la definición/gestión.

- **Arquitectura**: basada en ROA (*resource oriented architecture*) que permite *high availability cluster con no SPOF* (*no single point of failure*) y con *multi-master cluster* (para garantizar la disponibilidad) y todo gestionado desde la interfaz GUI (basada en AJAX).

- **Sistema de archivos**: basado en pmxcfs (*Proxmox VE Cluster File System*) orientado a una base de datos para el almacenamiento de los archivos de configuración y que son replicados sobre todos los nodos utilizando Corosync.

- **HA**: basado en Linux HA para proveer un sistema fiable con alta disponibilidad.

- **Agentes**: soporte para KVM y Linux Containers (LXC).

- **Seguridad**: MV/contenedores aislados con soporte seguro (SSL) para consola VNC en HTML5 y con gestión basada en roles para el tratamiento de permisos para todos los objetos (MV, contenedores, almacenamiento...) y autentificación multimodo (por ejemplo, local, MS ADS, LDAP...). Además, incorpora un *firewall* integrado que permite filtrar paquetes sobre cualquier interfaz tanto de una MV como de un contenedor, y aplicar las reglas por grupos denominados *security groups*.

- **Migración**: «en vivo» que permite mover MV desde una máquina física a otra sin tiempo de apagado/recuperación.

- ***Backup*/recuperación**: incorpora una herramienta (`vzdump`) para la creación de *snapshots* de los contenedores o MV que permite salvar (y posteriormente recuperarlos) estos en diferentes tipos de almacenamiento (como NFS, iSCSI LUN, Ceph RBD or Sheepdog) en un formato optimizado y efectivo.

- ***Bridged Networking***: todas las MV comparten un *bridge* dando la conectividad entre las MV y el exterior a través de una interfaz de red y se permite la generación de VLAN (IEEE 802.1q) y el *network bonding/aggregation*, permitiendo construir redes complejas adecuadas a las necesidades de conexión de los *guests*.

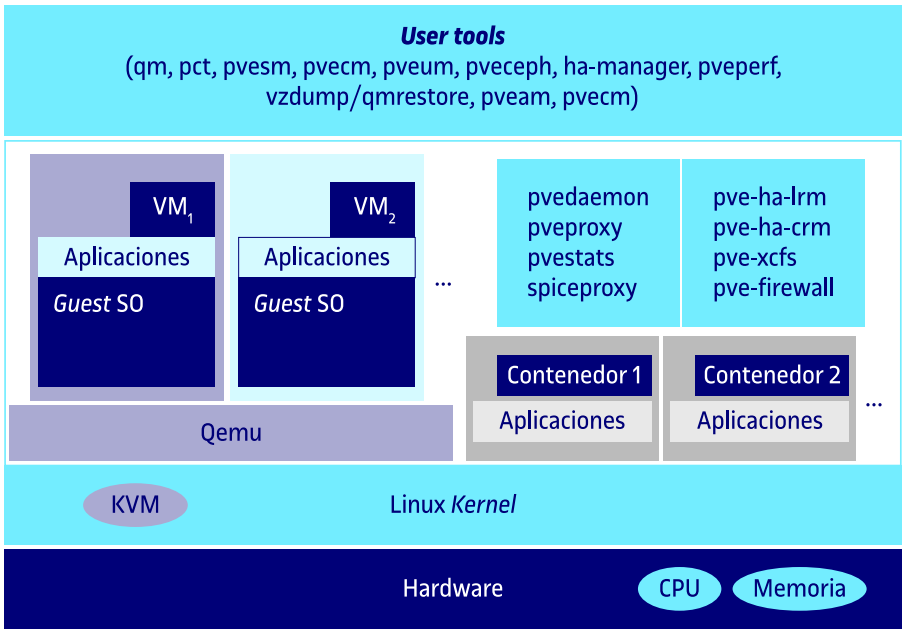
- **Almacenamiento**: es flexible y permite que las MV puedan ser almacenadas en local o compartido por NFS o SAN sin grandes restricciones y permitiendo la migración en vivo si están en sistemas compartidos. Los sistemas de archivos en red soportados son los habituales en Linux LVM sobre iSCSI targets, iSCSI target, NFS, Ceph RBD, Direct to iSCSI LUN o GlusterFS y como locales LVM Group (sobre dispositivos de bloques, DRBD...), *local file system*, ZFS.

Proxmox VE comparte características similares a VMware Sphere, Hyper-V o Citrix-XenServer en cuanto a que es un *bare-metal-hypervisor*, soporta migración «en vivo» y HA, *Snapshots* y *backup* de MV/*Containers*, pero además de ser *open source* es el único que posee soporte para contenedores y dispone de una gestión centralizada integrada. En cuanto a sus limitaciones, son las menos restrictivas e iguales que las de VMWare, que están en 160 CPU/2 TB RAM por *host*.

Como se puede observar en la documentación [Pve], este proyecto es muy reciente (comienza en 2007 y su primera versión estable es de 2008). En la primera versión se utilizó OpenVZ para contenedores y KVM para máquinas virtuales, y en sucesivas versiones, se agrega Corosync y *pmxcfs* (nuevo sistema de archivos de clúster), con lo cual se logra un gran avance en la gestión de clúster donde administrar múltiples nodos requería la misma complejidad que hacerlo con uno. También se agrega una API REST (escrita en JSON-Schema) para integrar Proxmox VE con otras herramientas y con lo cual luego se reemplaza la interfaz de usuario por una moderna aplicación HTML5 + JavaScript, y la original consola VNC (basado en Java) con noVNC para administrar las máquinas virtuales.

No obstante, los cambios más importantes (según los desarrolladores) fueron el soporte para ZFS (primera distribución en ofrecerlo en Linux en 2014) y la gestión de almacenamiento Ceph en los nodos del hipervisor, las copias de seguridad «en vivo» de KVM, y el cambio de OpenVZ a LXC para que los contenedores estén plenamente integrados y puedan utilizar las mismas funciones de almacenamiento y red que las máquinas virtuales.

La figura siguiente muestra la arquitectura de Proxmox VE.



En la capa de usuario se pueden apreciar las siguientes herramientas:

- **qm** (Qemu/KVM *Virtual Machine Manager*): ejecuta comandos en Qemu (*Guest Agent*).
- **pct** (*Container Toolkit*): crea o clona un *container*.
- **pvesm** (*Storage Manager*): crea un nuevo espacio de almacenamiento.
- **pveum** (*User Manager*): gestión de usuarios.
- **pveceph** (*Manage CEPH Services on Proxmox VE Nodes*): crea un monitor de Ceph.
- **ha-manager** (HA): gestión de la infraestructura de alta disponibilidad (HA).
- **pveperf** (*Benchmark Script*): análisis de prestaciones.
- **vzdump/qmrestore** (*Backup Utility for VMs and Containers /Restore Backups*): crea y recupera copias de resguardo.
- **pveam** (*Appliance Manager*): gestiona las *appliances*.
- **pvecm** (*Cluster Manager*): gestión del clúster.

Entre los principales *daemons*/servicios se puede enumerar:

- **pve-firewall** (*Firewall*)

- **pvedaemon** (API)
- **pveproxy** (*Proxy*)
- **pvestatd** (*Status*)
- **spiceproxy** (SPICE *Proxy Service*).
- **pmxcfs** (*Cluster File System*)
- **pve-ha-crm** (*Cluster Resource Manager*)
- **pve-ha-lrm** (*Local Resource Manager*)

2.4.1. Instalación y creación de máquinas virtuales y contenedores

Para la instalación de este hipervisor se utilizará, como prueba de concepto para ver su potencialidad solamente (no para una máquina de servicios), como una máquina virtual de KVM. Para ello se necesitará activar la virtualización anidada, como se mencionó en el apartado de KVM (ya que si no las MV que ponga en marcha solo se iniciarán en modo emulación, con la consiguiente pérdida de prestaciones). Este hipervisor está basado en Debian y la imagen incluye todos los paquetes necesarios, solo se debe crear la máquina virtual en KVM, insertar esta imagen en la unidad de CD-Rom e iniciarla, seleccionar *Install Proxmox VE* del menú de arranque y responder a las preguntas de configuración que realizará:

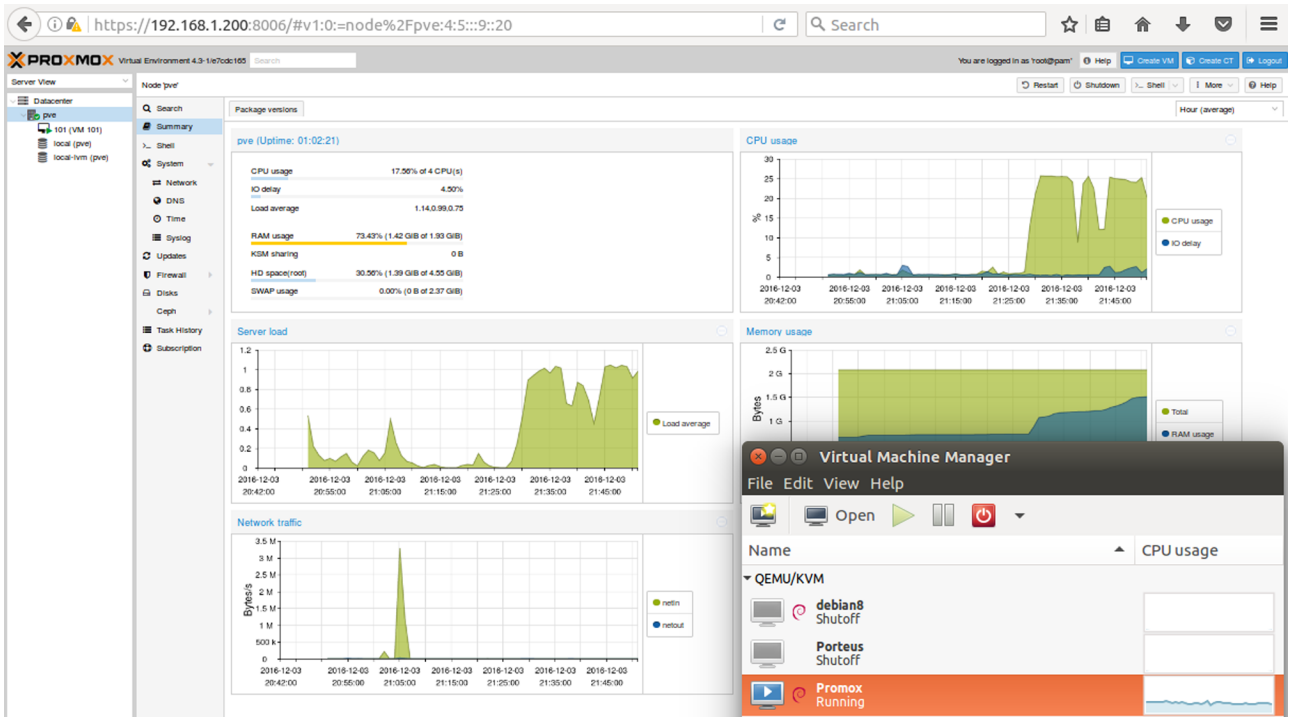
- discos a utilizar: particiones que en el presente caso es simple ya que es todo el disco virtual y formatos *ext3/4, xfs, ZFS*,
- paquetes: todos los paquetes para gestionar el entorno incluyendo máquinas virtuales KVM y contenedores LXC, además del entorno de administración web,
- configuración básica de la red: en este caso es necesario darle un IP de nuestro segmento de red KVM (no obstante, después se podrá reconfigurar).

Luego de unos minutos tendremos el *login* de PVE y se podrá acceder a la consola. Una alternativa (consultar la documentación) es instalarlo sobre un Debian previamente instalado (ya sea virtual o sobre *bare-metal*), pero solo se recomienda a usuarios avanzados en Debian. Cabe recordar que si se está sobre un sistema físico, el disco seleccionado para la instalación será formateado y por lo tanto perderá todo su contenido.

Una observación importante, cuando se escoge el formato de los discos, es, por defecto, *ext4*, pero se pueden seleccionar los otros disponibles y ofrece opciones adicionales para configurar el LVM. El instalador creará un *volume group* –*VG*– llamado **pve**, y *logical volumes* –*LV*– llamados ***root***, ***data*** y ***swap*** donde el tamaño podrá ser controlado por el parámetro *hdsize* (define el tamaño total del disco a ser utilizado, lo cual permite reservar espacio para acciones futuras como un PV/VG adicional y podrá anexarse al LVM posteriormente), *swapsize* (por defecto, el mismo tamaño de la RAM y hdsize/8 como máximo), *maxroot* (tamaño máximo para el volumen de *root*), *maxvz* (tamaño del volumen de datos), y *minfree* (define la cantidad mínima de espacio libre en el volumen *pve*, que será de 16 GB si la partición es superior a 128 GB o *hdsize*/8 en otro caso).

Si se dispone de más de un disco se puede utilizar ZFS (*Zettabyte File System*) como sistema de archivo, que destaca por el soporte a archivos de gran tamaño, la unión de dos conceptos (separados por otros sistemas de archivos) como sistema de ficheros y administrador de volúmenes lógicos y nueva estructura optimizada sobre el disco que permite archivos ligeros y una gestión del espacio simple. Para tener una idea sobre los tamaños que maneja, ZFS permite 2 10^{14} entradas en un directorio (2^{48}), 16 exbibytes (2^{64} bytes) como tamaño máximo de un único archivo, así como el tamaño máximo de cualquier atributo, 256 zebibytes (2^{78} bytes) el tamaño máximo de un *zpool* (3×10^{23} petabytes) y de los cuales se admiten 2^{64}. Una comparación frecuentemente utilizada por sus desarrolladores (Sun Microsystems en 2004, que se integró en OpenSolaris en 2005, el cual fue discontinuado en 2010, aunque hay diferentes *forks* como Illumos que continúan con sus avances) es que, si un usuario crease mil ficheros por segundo, tardaría más de nueve mil años en alcanzar el límite impuesto por el número de ficheros. En Proxmox VE, ZFS soporta varios niveles de RAID, lo cual permite adaptarse si no se dispone de una controladora RAID hardware, pero es necesario tener en cuenta que ZFS utiliza una cantidad muy grande de memoria; es necesario memoria adicional si se selecciona esta opción, por ejemplo, 4 GB más 1 GB RAM por cada terabyte de espacio de disco para ZFS (*raw*), es decir, si se dispone de dos discos de 1 TB en Raid 1 sobre ZFS, se recomiendan 6 GB de RAM.

Toda la configuración de PVE será realizada por la interfaz web, a la cual se accederá por la IP del servidor (la indicada durante la instalación) poniendo en un navegador ***https://ip-servidor:8006*** (tener en cuenta que se deberá aceptar el certificado propio de PVE antes de acceder, y accederemos con el usuario *root* y *passwd* indicados durante la instalación). La siguiente imagen nos muestra una pantalla de la administración de PVE ejecutándose como MV de KVM con la opción de virtualización anidada.



Para instalar una MV/contenedor es necesario disponer de las ISO o de los archivos de los contenedores que pueden ser gestionados desde la propia interfaz web. Para ello se debe seleccionar el *datastore* local y, en *Content*, acceder a *Templates* (para los contendores); saldrá una lista de todos los disponibles tanto del sistema como los contenedores que provee Turnkey (los del sistema también están en http://download.proxmox.com/appliances/system/) que se pueden descargar directamente. Para las ISO se puede utilizar la opción de *Upload* desde el mismo menú y subir una ISO previamente descargada. También estas operaciones se pueden hacer desde la CLI, que se deberán guardar en el *datastore* (*/var/lib/vz/template/iso* para las ISO y */var/lib/vz/template/cache* para los contenedores). Para una ISO, de Debian, por ejemplo, se puede hacer:

```
cd /var/lib/vz/template/iso
wget http://cdimage.debian.org/debian-cd/8.6.0/amd64/iso-cd/debian-8.6.0-amd64-netinst.iso
```

Las cuales ya se verán en la interfaz web cuando se actualice. Para los contenedores, en formato *xz*, una forma simple es a través de la CLI con el comando *pveam* (*appliance manager*).

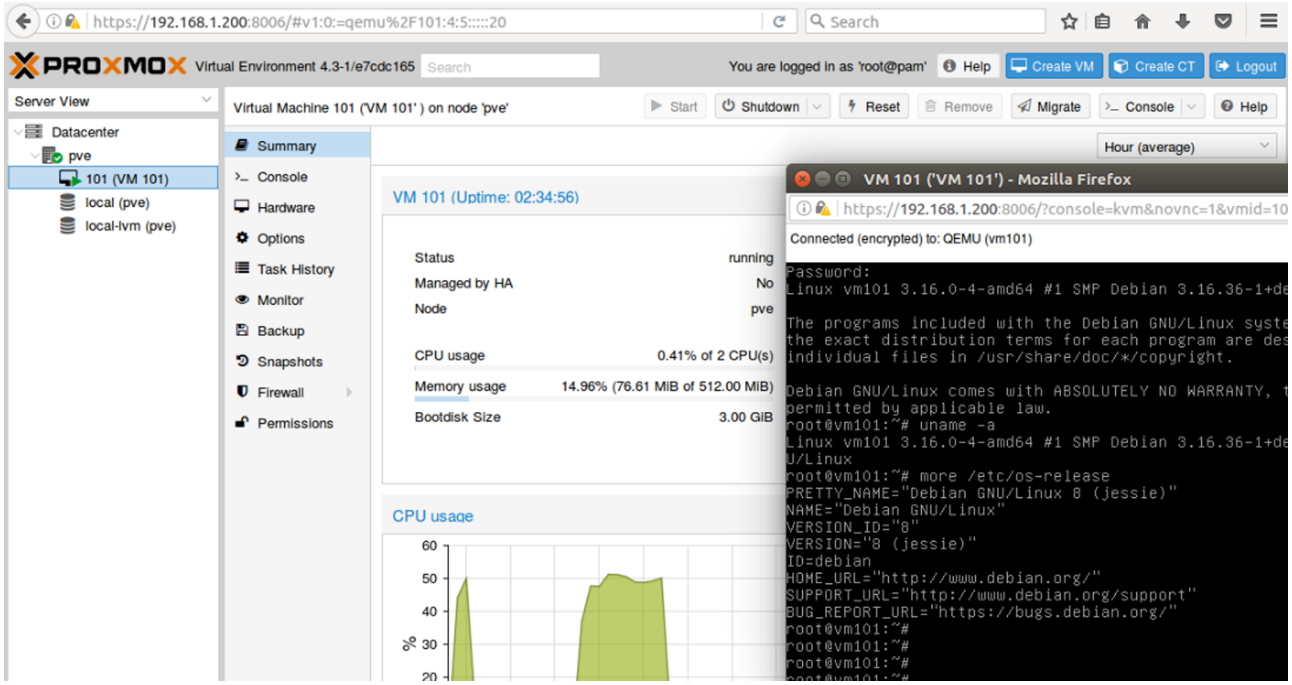
```
pvme update
pveam available
...
system      centos-x-default_xxxx_amd64.tar.xz
...
turnkeylinux  debian-x-turnkey-lamp_xx.yy_amd64.tar.gz
...
pveam download local debian-x-turnkey-lamp_xx.yy_amd64.tar.gz
```

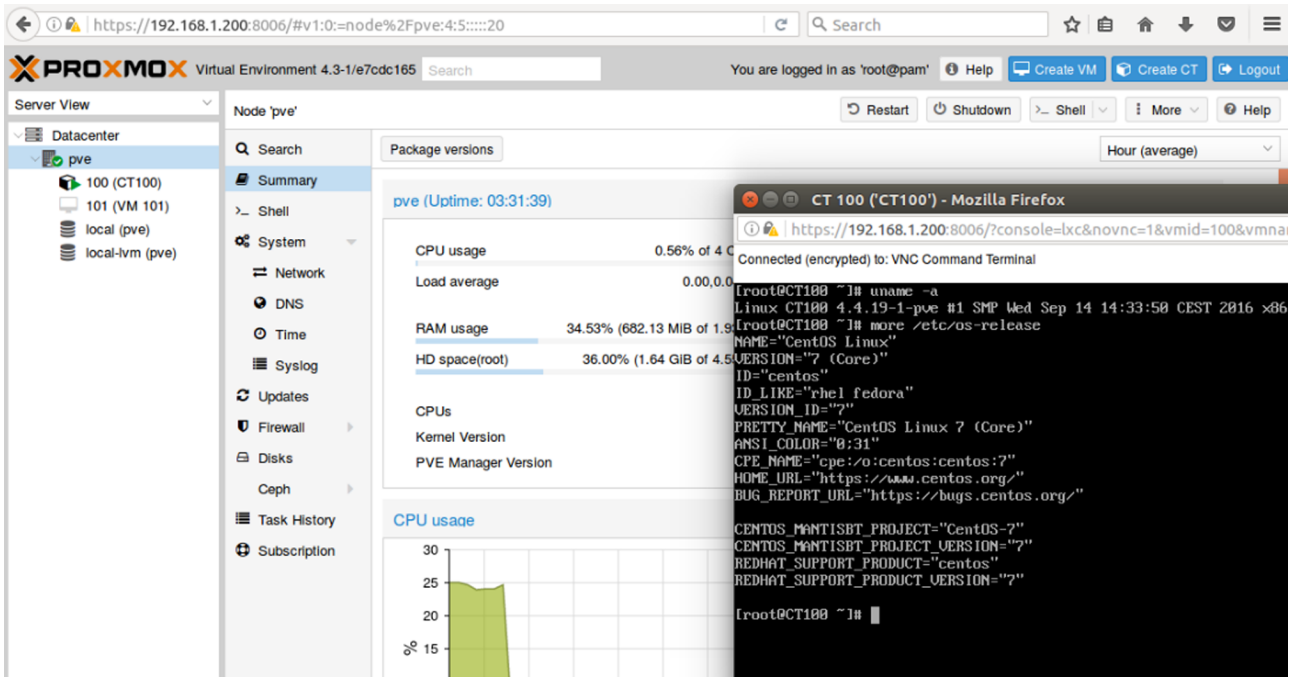
```
pveam download local centos-x-default_xxxx_amd64.tar.xz
```

Con ello ya se está en disposición de crear tanto una máquina virtual como un contenedor, seleccionando las opciones indicadas en la derecha de la interfaz.

Es importante tener en cuenta que en la presente prueba se está utilizando un servidor Proxmox virtualizado sobre una máquina KVM, por lo cual las máquinas virtuales dentro de este, si no se dispone de la virtualización anidada, deberán ser emuladas a través de Qemu. Es por ello que no se debe seleccionar esta opción durante su creación o deshabilitarla en el menú *Options KVM hardware Virtualization* de la máquina virtual ya que, si no, esta no arrancará (*error: no accelarator found!*). Si se dispone de la virtualización anidada (ver apartado KVM) en la MV de Promox, las MV sobre este podrán ejecutarse con la opción KVM activada. A las máquinas se podrá acceder a través de la consola noVNC o a través de ssh.

Las figuras siguientes muestran la ejecución de una MV Debian 8 instalada a partir de la ISO y un contenedor CentOS7.





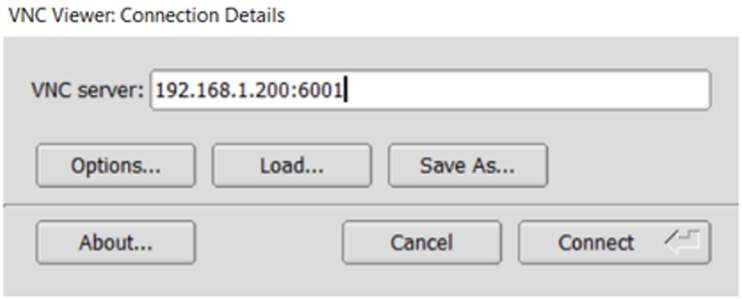
Es importante destacar la reducción significativa entre el despliegue de un contenedor y la reducción en el uso de recursos (como se puede apreciar en la interfaz gráfica de cada uno) en relación con las máquinas virtuales.

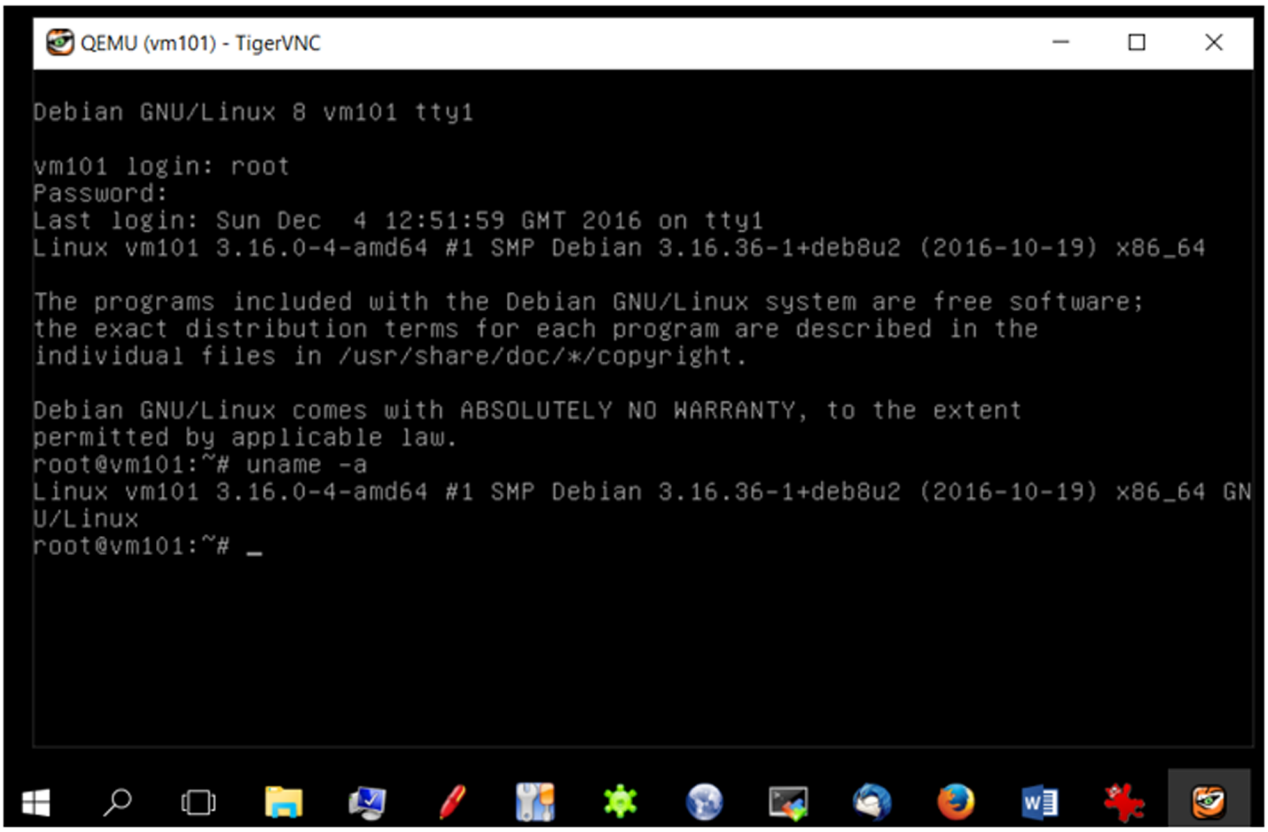
En [Pwi] se pueden encontrar una serie de manuales (*HowTo*) de cómo ajustar determinados parámetros o instalar/administrar determinados aspectos del entorno.

Un aspecto interesante es poder conectarse a las MV externamente, para lo cual la forma más simple es hacerlo a través de VNC (o por SPICE). Para ello, simplemente se accede a la opción de monitor en la máquina virtual y se le agrega:

```
change vnc 0.0.0.0:101
```

Donde 101 es el número que se debe agregar al puerto base de VNC (5900) para la conexión a esta máquina, quedando en este caso 6001. Luego, por ejemplo, con TigerVNC (incluye el cliente y el servidor y soporta diferentes algoritmos de encriptación, pero en este caso solo se ha descargado el cliente de VNC), se ejecuta:





Es importante tener en cuenta que en este caso la MV 101 solo tiene NAT en su red y que en esta prueba el servidor Proxmox se está ejecutando como MV de KVM, pero con *bridge* de su conexión (IP 192.168.1.200). En la documentación se indica cómo introducir *passwd* y cómo hacer esta configuración definitiva (ya que con *Monitor* solo es temporal).

También existen otras formas de conexión y es a través de un servicio en el propio servidor Proxmox. Para ello, se deberá activar el `vnc proxy` en el servidor instalando `openbsd-inetd` (o también se puede hacer con `xinetd`) y habilitarlo para la máquina en cuestión, sobre el servidor se ejecutará:

```
apt-get update
apt-get install openbsd-inetd
```

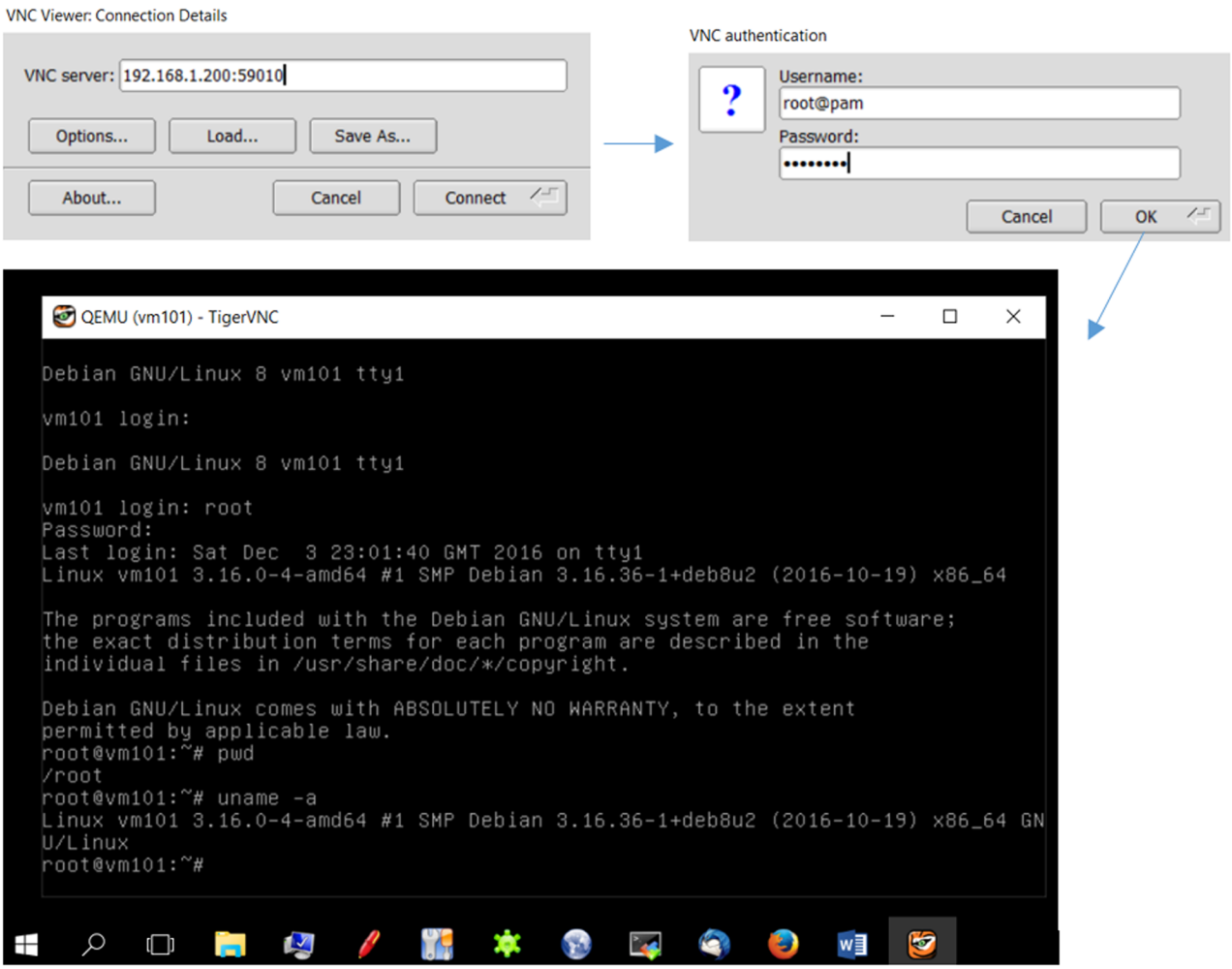
Se edita */etc/inetd.conf* y se agrega una línea con:

```
59010 stream tcp nowait root /usr/sbin/qm qm vncproxy 101
```

Donde se le indica a *Qemu/KVM virtual machine manager* (`qm`) que haga un *proxy* de la comunicación VNC (`vncproxy`) para la MV 101 (este es el nombre de la MV en Proxmox y no tiene nada que ver con el 101 utilizado en el puerto en el ejemplo anterior). Luego se reinicia el servicio:

```
service inetd restart
```

Para conectarse de una máquina externa se utiliza el mismo Tiger VNCviewer, aunque en este caso se debe autentificar con un usuario definido en el servidor.



Finalmente, se debe tener en cuenta que aquí solo se han mostrado básicamente las posibilidades del hipervisor, pero es importante considerar que este dispone de características de nivel avanzado como, por ejemplo, montar una arquitectura de alta disponibilidad con clústeres de servidores, conexión a diferentes servicios de almacenamiento, gestión de la seguridad a través del *firewall* interno, *backup* y restauración en diferentes modos o migración «en caliente» entre otras y que escapan al objetivo de este apartado.

2.4.2. Conexión *bridged* sobre una wifi

Para que las máquinas virtuales tengan dirección propia dentro de la red es necesario poder realizar un *bridge* {Dbr] para que diferentes IP salgan por el mismo NIC físico, por ejemplo nuestro servidor (KVM) y las máquinas virtuales (por ejemplo, Promox), pero si se desea montar sobre una conexión wifi existen diversos problemas (consultar la documentación en *bridged connections*, que si bien se indica en Ubuntu16.04 es lo mismo para versiones posteriores). El principal problema es que la mayoría de *access points* (AP) rechazan, por seguridad, los *frames* que tienen una dirección origen diferente que la que se ha autentificado con el AP. Debido a que Linux hace el *bridging* en forma transparente (no modifica ni los *frames* de salida ni de entrada), es necesario modificar esto para que cada paquete pueda provenir desde la misma IP origen.

Para ello, siguiendo la documentación [Ufk], se ha construido un *script* que permite realizar esto sobre una máquina que dispone de KVM y su conexión es a través de la wifi NIC (wlo1). El archivo */etc/network/interfaces* solo tiene la configuración de conexión a la wifi como se muestra a continuación (es importante desactivar/desinstalar el *NetworkManager* para evitar que haya conflictos):

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
auto wlo1
iface lo inet loopback
#adecuar si se dispone de otro método de conexión WEP p.ej.
iface wlo1 inet dhcp
wpa-ssid SSID-de-la-red-Wifi
    wpa-psk PASSWD-de-la-red-Wifi
    dns-nameserver 8.8.8.8
    dns-nameserver 8.8.4.4
#no se utilizará ethernet wired
iface enp0s25 inet manual
```

Luego se crea un *script* con el siguiente contenido (en esta configuración se utiliza una conexión *tap*, que es un dispositivo de red virtual que simula un dispositivo en la capa de *link* y opera con los paquetes igual que si fueran *ethernet frames* y se utilizan normalmente para crear un *network bridge*):

```
#!/bin/bash
# Create br0 device
brctl addbr br0
# Create the tap device:
tunctl -t tap0
# Add tap0 to the bridge:
brctl addif br0 tap0
# IP to Br0
```

```
ip addr add 172.16.1.1/32 dev br0
ip link set br0 up
#Use parprouted to perform voodoo magic on the routing tables.
parprouted wlo1 br0
#start bcrelay - this will make sure that broadcast traffic
#(like the dhcp stuff) gets pushed through the tap as well.
bcrelay -d -i br0 -o wlo1
# It is necessary to add a routing rule between the ip assigned to the VM (in this case 10.0.0.100)
#for the packets can back to br0. Replace with the ip of the virtual machine (which is an IP
#inside the wifi network).
route add -host 10.0.0.100 dev br0
```

Una vez ejecutado el *script* y asignadas las IP a las máquinas virtuales, se podrá verificar que las mismas tienen conexión a la red, y si la wifi está conectada a internet, también lo estarán las máquinas virtuales y se podrá acceder a ellas a través de una IP en la misma red.

2.5. Hyper-V

Esta plataforma permite crear y administrar un entorno informático virtualizado mediante la tecnología de virtualización integrada en Windows®. La arquitectura software está formada por el propio hipervisor, el servicio administración de MV, el control de instrumentación (WMI), el bus de máquina virtual (*VMbus*), el proveedor de servicios de virtualización (VSP) y el controlador de infraestructura virtual (VID). La administración se realiza mediante una herramienta (GUI) administrador de Hyper-V, un complemento Microsoft Management Console (MMC) y la conexión a máquina virtual, que da acceso a la salida de una MV para poder interactuar externamente. Además, es posible interactuar mediante comandos específicos (llamados *cmdlets*) que se despliegan sobre CLI y PowerShell.

La tecnología incluida en Hyper-V virtualiza el hardware para proporcionar un entorno que permite ejecutar diferentes sistemas operativos al mismo tiempo en un equipo físico y administrar las MV, así como sus recursos. El objetivo de Hyper-V es proveer un entorno de *cloud* privado y adaptarlos al uso en función de los cambios en la demanda, con el fin de prestar unos servicios de TI más flexibles, con la consiguiente reducción del hardware (por la consolidación de servidores y las cargas de trabajo en un menor número de equipos físicos de mayor potencia) que reducen en forma global el consumo de recursos como la energía y el espacio físico. Con esta infraestructura es posible disponer de una infraestructura de escritorio virtual (VDI) que contribuye a aumentar la agilidad empresarial y la seguridad de los datos y, al mismo tiempo, simplifica la administración del sistema operativo y las aplicaciones del escritorio.

Como requisitos de hardware requiere un procesador de 64 bits con soporte hardware para la virtualización (Intel VT-x o AMD-V) y con soporte hardware para DEP (*Data Execution Prevention*) habilitada. Como SO *guests*, Hyper-V

da soporte en máquinas de 32/64 bits a Windows. En *guests* Linux y FreeBSD da soporte (ver enlace anterior para consideraciones particulares) para CentOS/RHEL, Debian, SUSE, Oracle Linux, Ubuntu y FreeBSD. Hyper-V sobre Windows 10 Enterprise/ Professional es el reemplazo de Microsoft Virtual PC, el cual tuvo EOL en abril de 2017.

Entre las razones para utilizar Hyper-V sobre entornos Windows, se pueden enumerar:

- Ejecución de múltiples SO, configuraciones software/hardware diferentes con distintas versiones de SO sobre la misma máquina física, con el consiguiente aislamiento en entornos de prueba o desarrollo y con una considerable reducción de la inversión, energía y espacio.

- Desplegar un entorno de pruebas sobre un equipo de desarrollo (portátil y de escritorio) para luego exportarlo a sistemas de producción sobre los sistemas reales o sobre Azure. También es posible hacer la operación inversa sobre una MV en producción sobre la cual se detectan problemas, exportarla e importarla sobre el entorno local para analizarlos en un entorno de depuración.

- Analizar las posibilidades del *virtual networking* para generar desarrollos similares a los entornos reales, pero sin afectar a su continuidad o insertar máquinas que puedan interferir en su comunicación.

Existen diferencias entre Hyper-V, ejecutándose en versiones Wserver o Windows 10. Una de ellas es que tienen un modelo de memoria diferente (sobre *server* solo MV, sobre W10 comparten espacio con otras aplicaciones), y la otra es que solo existe una serie de características que solo son soportadas por WServer (*Virtualizing GPUs using RemoteFX, Live migration*, Hyper-V *Replica, Virtual Fiber Channel*, SR-IOV *Networking, Shared* VHDX).

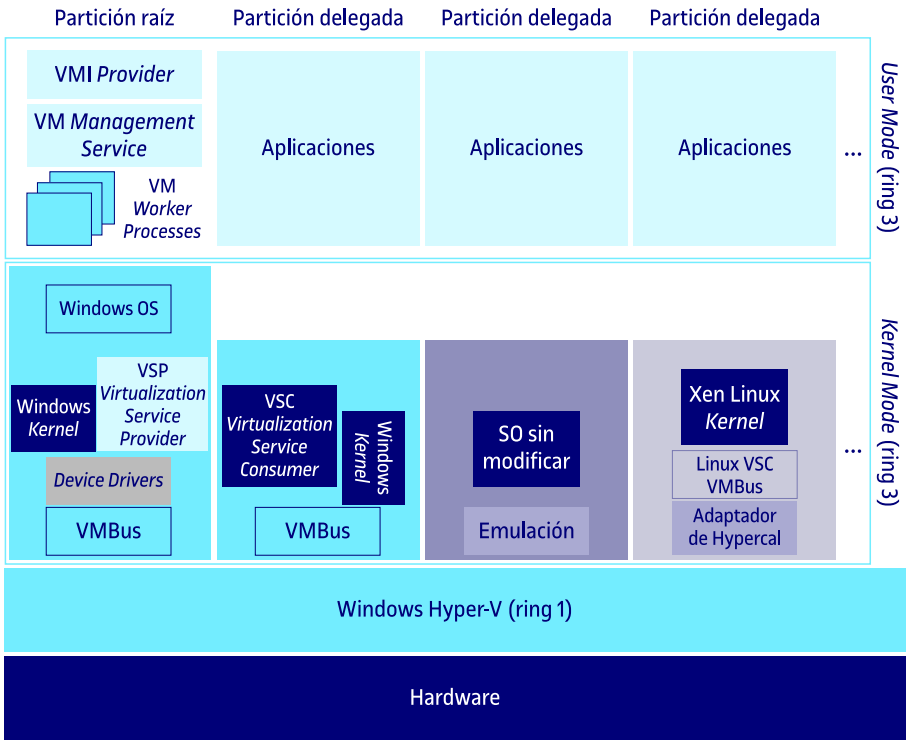
Es conveniente tener en cuenta que existen algunas limitaciones en la virtualización, por ejemplo, en aplicaciones que dependen de hardware específico y que no tendrán un buen rendimiento sobre una MV (juegos, aplicaciones que requieran uso intenso de GPU, o temporizadores que trabajen por debajo de los 10 ms, como *live music mixing*). Asimismo, aplicaciones que se ejecuten sobre el *host* y que sean sensibles a la latencia se verán afectadas cuando se ejecuten sobre el *host*, ya que el propio SO *host* se ejecuta sobre la capa de virtualización de Hyper-V.

Dada la relevancia de los sistemas Windows en el ámbito empresarial, Microsoft ha desarrollado una estrategia frente a la competencia para cubrir una parte importante del mercado. De hecho, en 2016, aparece en el cuadrante mágico de Gartner como uno de los proveedores líderes en soluciones de virtualización sobre x86 (junto con VMware) y ha mantenido sus posiciones en entornos empresariales básicamente por el dominio de Windows Server. Dentro de

esta estrategia, Microsoft publica (sin restricciones temporales) una plataforma denominada Hyper-V Server (última versión disponible en Enero 2022 = 2019). Hyper-V Server, que es una plataforma dedicada (*stand-alone*) que contiene el hipervisor, el Windows *Server Driver Model*, el soporte para la virtualización y componentes para la clusterización de alta disponibilidad, pero no contiene las restantes características de un SO Windows Server.

Esta plataforma es una solución importante, ya que produce una imprenta muy pequeña y requieren mínimos recursos que lo hacen adecuado para empresas o instituciones que requieren consolidar servidores sin nuevas licencias o necesitan ejecutar otros SO junto con Windows. La última versión incluye un conjunto de características adicionales han sido añadidas para dar soporte a la virtualización (de categoría empresarial) e integración con el *cloud* híbrido, así como mejoras en el escalado y adaptación a diferentes tipos de carga para ajustarse a las necesidades de prestaciones de sistemas críticos. Además, Hyper-V se puede activar en Windows 10 (*desktop*) y en las versiones *Windows servers*.

La figura siguiente muestra la arquitectura de Hyper-V [Hay].



Donde **VMBus** es un mecanismo dentro de la arquitectura Hyper-V que permite la comunicación lógica entre las particiones y el hipervisor, es decir, es el canal de comunicaciones interno para redirigir las peticiones a los dispositivos virtuales al hardware y viceversa; **VSC** (*Virtualization Service Client/Consumer*) es un dispositivo sintético que reside en las particiones delegadas/hijas que utiliza los recursos provistos por los *Virtualization Service Providers* (VSP) en la partición raíz (*parent*) y que se comunica con VMBus para satisfacer las

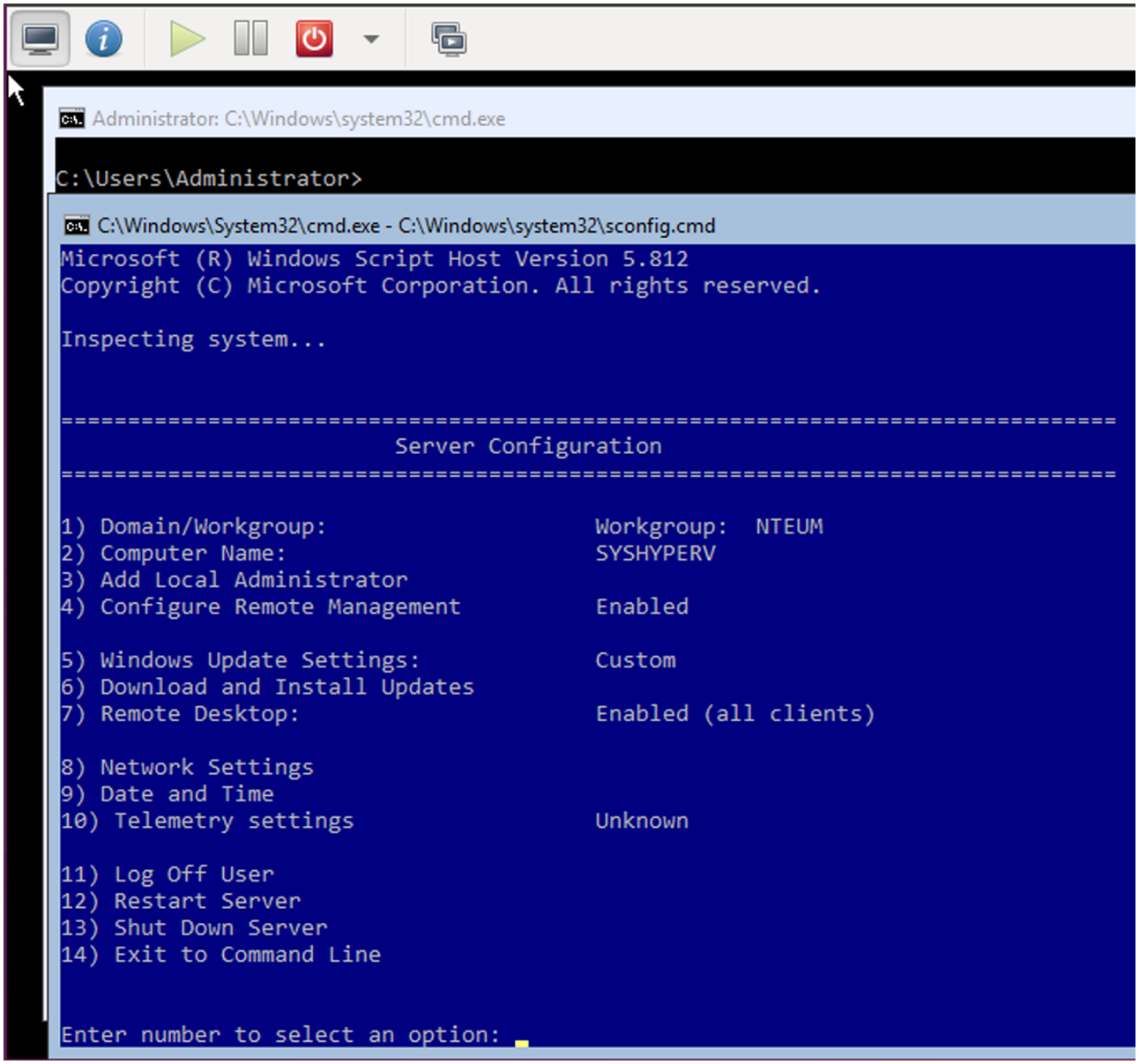
peticiones de E/S; **VMMS** (*Virtual Machine Management Service*) responsable de manejar el estado de la MV en las particiones delegadas/hijas; **VMWP** (*Virtual Machine Worker Process*) componente que se ejecuta dentro del espacio de usuario y que provee los servicios de gestión de la MV al SO de la partición raíz (uno por cada MV en una partición delegada).

2.5.1. Instalación de Hyper-V Server

Para la instalación de Hyper-V Server [Hyv] se descargará la ISO del repositorio del Microsoft y, en nuestro caso, como prueba de concepto, se utilizará una máquina virtual de KVM con 3 VCPU (pero pueden ser menos), 4 GB de Ram, 40 GB de disco y un dispositivo de red de tipo *shared* apuntando a dispositivo de *bridge* de KVM (br0 en nuestro caso, hay que recordar configurar el *bridged*, como se indicó en el apartado de Promox, si se desea trabajar con un wifi como conexión *bridged*). Después de la creación de la máquina y antes de la instalación, es necesario modificar sobre la configuración creada (*/etc/libvirt/qemu/*) el *cpu mode*:

```
virsh edit nombre_máquina_hyperV
Modificar la línea de cpu mode a: <cpu mode='host-passthrough'/>
```

Luego ya se puede arrancar la máquina e instalar Hyper-V respondiendo a las preguntas que realizará (disco, *passwd*, que deberá de tipo seguro con *Mayúsculas-minúsculas-dígitos...*). Finalmente, antes de reiniciar el sistema, hay que quitar la ISO e iniciar la MV, que podrá arrancar desde el disco duro. Durante el arranque solicitará que se introduzca *Crtl+Alt+Supr* para acceder a la consola, quedando como la de la figura:



En la cual se podrán ajustar los parámetros del *Workgroup*, *Computer Name*, habilitar el *Remote Management* y el *Remote Display*, modificar *Network Settings* para darle una IP dentro de nuestra red (además de *netmask*, *Gateway* y *DNS*), configurar *Date/Time* y finalmente reiniciar el Hyper-V *Server*. Dado que es un servidor complejo, sobre todo de las partes de administración y gestión remota, se recomienda consultar la documentación [Hyv].

Para la gestión remota del servidor es necesario disponer de Windows 10 (o de otro Windows *server*); en nuestro caso se realizará sobre W10: activar en *Programs & Features → Turn Windows Features On-Off → Hyper-V → Hyper-V Management Tools*. Seguir los pasos indicados en [Mrs] para activar los permisos según se tenga dominio o no.

En nuestro caso, no se dispone de dominio ni DNS para el segmento, por lo cual lo primero será agregar en *C:/Windows/System32/drivers/etc/hosts* la máquina remota y su FDQN. Para ello se debe desactivar el antivirus si se dispone de

él (normalmente evitan que este archivo pueda ser modificado), luego abrir una ventana de `cmd` a través de «buscar» (Cortana, por ejemplo), pero abrirla con el botón derecho seleccionando la ejecución con permisos de administrador. Luego ejecutar:

```
notepad C:/Windows/System32/drivers/etc/hosts
```

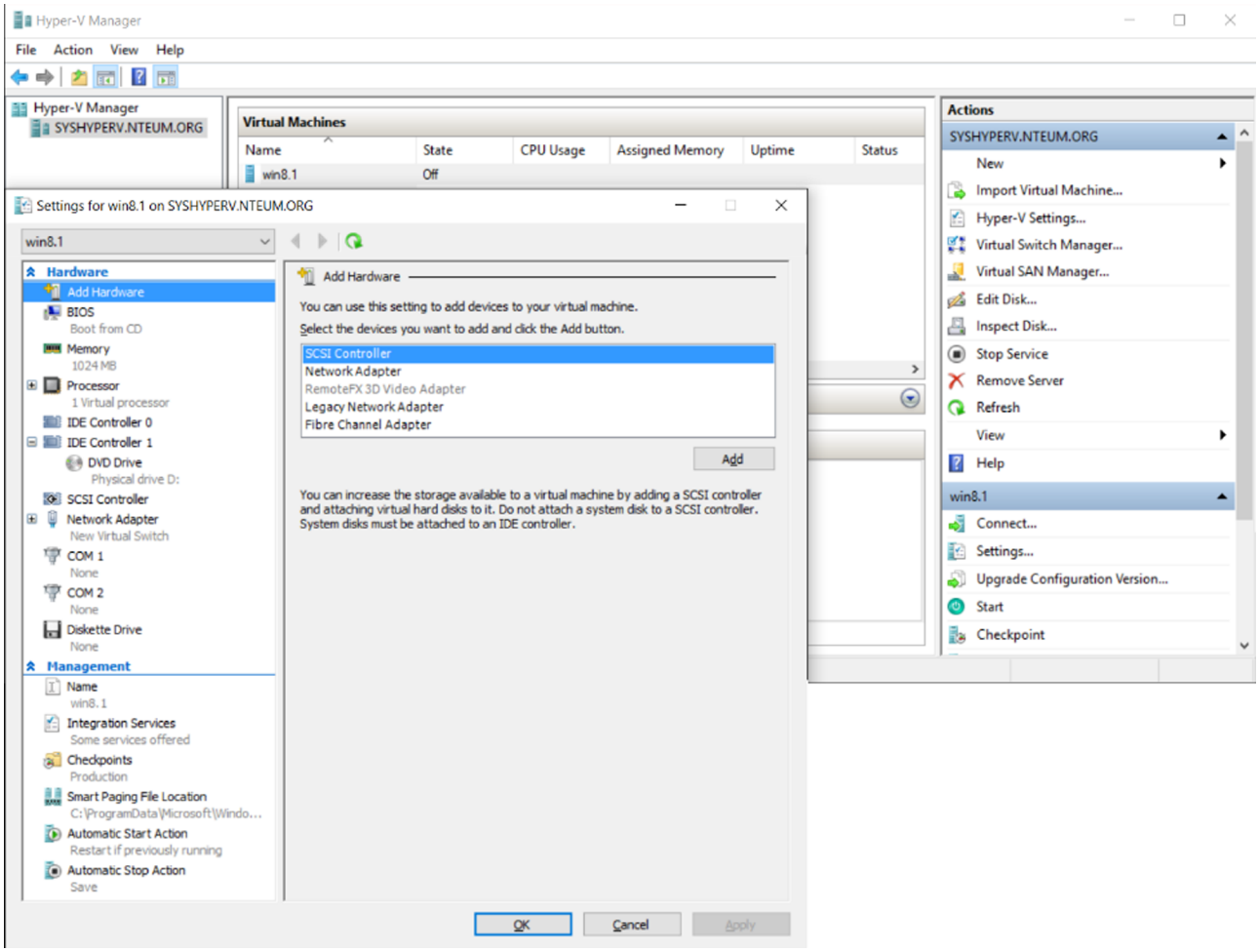
Y sobre este archivo incluir la IP y el FDQN del servidor Hyper-V.

Luego se debe, sobre el servidor Hyper-V, activar la opción de administración remota y sobre la máquina desde la cual se desea administrar el servidor remoto y, en una ventana de PowerShell, ejecutar (reemplazando: *fqdn-of-hyper-v-host* por el nombre FDQN de la máquina remota).

```
Set-Item WSMan:\localhost\Client\TrustedHosts -Value "fqdn-of-hyper-v-host"
Enable-WSManCredSSP -Role client -DelegateComputer "fqdn-of-hyper-v-host"
```

Adicionalmente es necesario configurar el *group policy*, llamado desde una ventana de `cmd`, ejecutar el comando `gpedit` e ir a *Computer Configuration* → *Administrative Templates* → *System* → *Credentials Delegation* → *Allow delegating fresh credentials with NTLM-only server authentication*, habilitarlo e insertar: `wsman/fqdn-of-hyper-v-host` (en nuestro caso `wsman/syshyperv.nteum.org`). Cabe recordar que se deberá disponer de un usuario con el mismo nombre y *passwd* en las dos máquinas y que la máquina gestora será la que inicie el Hyper-V Manager.

Luego se podrá ejecutar el Hyper-V Manager [Mrs]; agregando un nuevo servidor, ya podremos tener la conexión al servidor Hyper-V y realizar todas las tareas de administración y gestión de la infraestructura. La figura siguiente muestra la interfaz de esta herramienta:



Es importante comentar (como ya se ha hecho en el caso de Promox) que Hyper-V es un entorno complejo y con muchas opciones, escenarios y configuraciones posibles. Se necesita tiempo y dedicación para conocer la infraestructura, los modelos de gestión, la seguridad, los permisos, los atributos, etc. que son necesarios para tener todo correctamente configurado y ejecutándose. En este apartado solo se ha hecho una prueba funcional sobre el hipervisor para mostrar sus posibilidades (que son muchas y variadas).

2.5.2. Instalar Hyper-V sobre W10

Sobre una máquina local Windows10, instalar Hyper-V no tiene ninguna dificultad. Los requisitos son un procesador de 64 bits con traducción de direcciones de segundo nivel (SLAT), extensión hardware (VT-x), mínimo de 4 GB de memoria y desde el BIOS; además de las extensiones de virtualización, debe estar habilitada la prevención de ejecución de datos (*Hardware Enforced Data Execution Prevention*). Esta información se puede obtener ejecutando el PowerShell o consola (`cmd.exe`) y a continuación `systeminfo.exe` donde, si todos los requisitos que se enumeran en Hyper-V tienen un valor de Yes, el sistema puede ejecutar el rol de Hyper-V. Si algún elemento devuelve No, se deberán comprobar y realizar los ajustes necesarios.

```
Hyper-V Requirements:      VM Monitor Mode Extensions: Yes
                           Virtualization Enabled In Firmware: Yes
                           Second Level Address Translation: Yes
                           Data Execution Prevention Available: Yes
```

Para instalar Hyper-V se debe ir a *Programs & Features → Turn Windows Features on-off → Hyper-V* y activarlas todas haciendo luego *OK*. Luego de reiniciar la máquina en las herramientas de administración, podremos encontrar *Hyper-v Management* e iniciar la consola de administración, la cual detectará la propia máquina local y la agregará a la lista de servidores (en caso de que no lo detecte, hacer clic en *add new server*).

Como paso previo a crear una MV de Hyper-V, será necesario que esta pueda conectarse a la red, por lo cual se creará un *switch* virtual. Hyper-V tiene tres tipos de *switchs*:

- **Externo**: conectividad entre la red virtual y un determinado NIC físico que puede ser el del *host* u otro NIC si se desea aislar el tráfico de las MV.

- **Interno**: la red no está conectada a un NIC, pero sí existe comunicación entre el *host* y la MV conectadas al *switch*.

- **Privado**: no está conectado a un NIC ni existe conectividad con el *host* ni las MV conectadas a este *switch*.

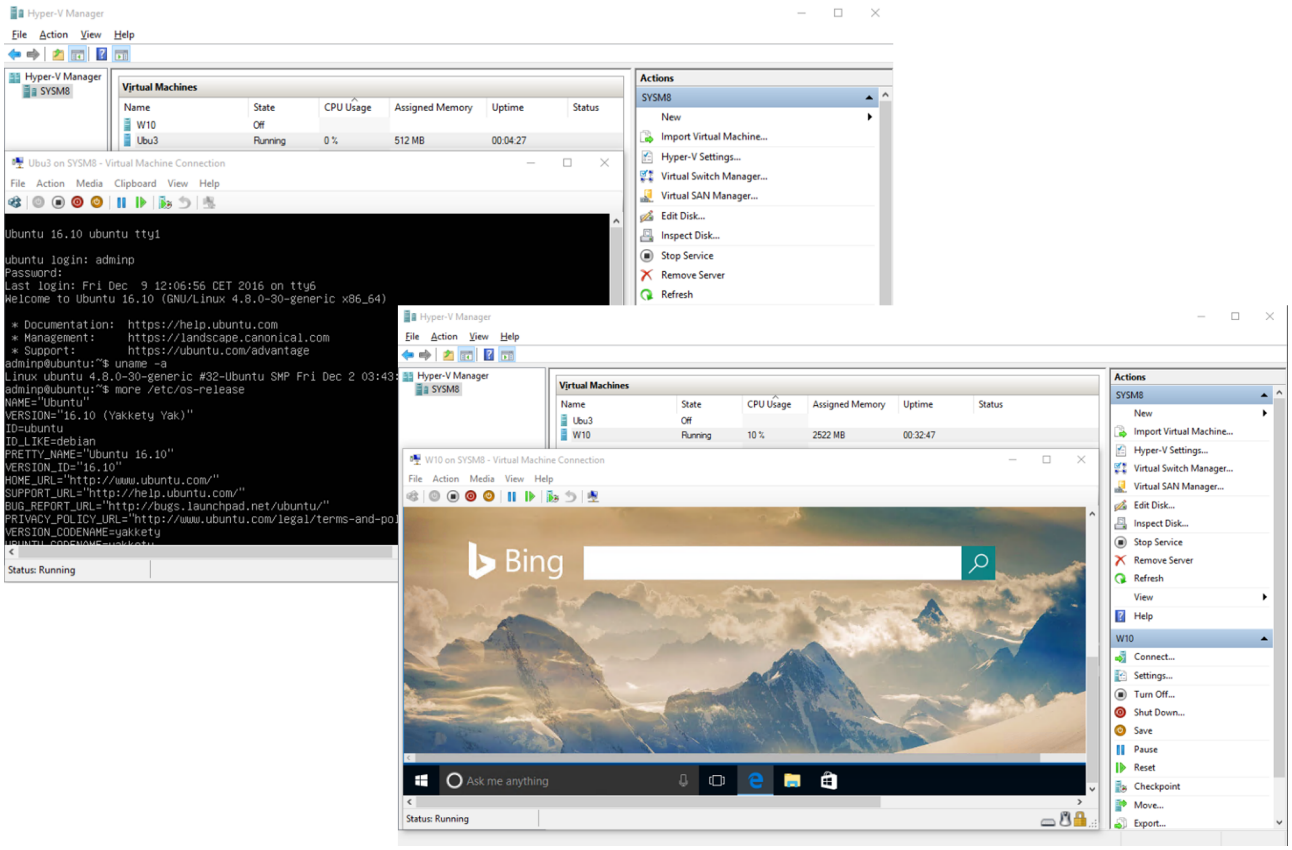
Para ello acceder a *Virtual Switches → New virtual network switch → type of virtual switch → seleccionar External → Create Virtual Switch*, indicar un nombre y verificar las propiedades seleccionando el NIC sobre el cual se desea conectar y finalmente *OK*.

Para crear una MV se puede hacer de muchas formas, mediante Windows *Deployment Services*, con un disco duro virtual preparado, o manualmente, mediante una ISO, por ejemplo. En este caso se instalará Ubuntu16.10 (Yakkety), que previamente se habrá descargado la ISO, y un W10. En este último caso se pueden bajar las ISO de prueba del TechNet Evaluation Center pero se ha optado por buscar una MV ya instalada de W10 y se utilizará una preparada para Hyper-V desde el sitio de Test Microsoft Edge (para ello, seleccionar la versión y el tipo de máquina virtual Hyper-V y descargar el zip que se deberá descomprimir).

Para crear una MV, seleccionar *New → Virtual Machine* y responder a todas las preguntas para configurar la máquina virtual. Se recomienda seleccionar máquinas de Generation 1, la cantidad de memoria y disco adecuadas (4 GB para Linux), conectarlas al *switch* creado anteriormente y determinar dónde se almacenarán las MV y las propiedades de estas. En el caso de Linux, indicar

que la instalación se realizará una ISO (indicándole dónde está) y, en caso de W10, seleccionar que se utilizará un disco con formato *vhdx* (que se encontrará donde anteriormente se ha descomprimido el archivo zip descargado).

Luego se podrán poner en marcha las MV y abrir una ventana para acceder a la MV (connect, si se utilizan las máquinas de Test Microsoft Edge, el usuario es **IEUser** y el *passwd* **Passw0rd!**) y acceder a través de RDP desde otra máquina (para ello, en la máquina cliente, se deberá permitir la conexión a escritorio remoto dentro de las opciones de sistema). Las dos imágenes a continuación muestran las pantallas de la ejecución de Ubuntu Yakkety y de W10.



Se debe tener en cuenta que W10 es muy exigente con los recursos; considerar una buena cantidad de memoria para los clientes W10 en MV (2 GB mínimo para un funcionamiento aceptable).

La gestión y administración de la plataforma y los detalles avanzados se pueden realizar a través de PowerShell. El comando que dará una lista de todos los cmdlets cuya funcionalidad se puede consultar en las Referencias de Microsoft es `Get-Command -Module hyper-v | Out-GridView`.

2.6. ESXI

VMware ESXi es un hipervisor de clase empresarial de tipo 1 desarrollado y mantenido por VMware y que el producto que lo contiene (VMware vSphere) figura como líder empresarial en Server Virtualization Software Reviews and Ratings de Gartner. Como todos los supervisores de tipo 1, incluye su propio sistema operativo; esta es una de las piezas principales de toda la infraestructura/productos de VMware [Ves].

Los dos componentes principales de vSphere Hypervisor son ESXi y vCenter Server. ESXi es la plataforma de virtualización que puede crear y ejecutar máquinas virtuales y *appliances* virtuales. VCenter Server es un servicio que actúa como administrador central de los *hosts* ESXi conectados en una red, permitiendo agrupar y administrar los recursos de múltiples *hosts* y que puede ser instalado en una máquina virtual Windows o servidor físico, o desplegar VCenter Server Appliance (VCenter preconfigurado sobre una MV Linux) sobre el propio ESXi (5.5 o posterior).

Entre las principales características de este hipervisor (en su versión 6.5) y toda la infraestructura que lo rodea, se puede enumerar entre otras [Wne]:

- Escalabilidad probada: número de *cores* por CPU física/CPU por *host* = sin límites, número de VCPU = 480, máximo de vCPU por MV = 8.
- VMware vCenter Server® Appliance: centro de control único y eje central de vSphere.
- VCenter Server® Alta disponibilidad: solución de HA para centros críticos.
- Copia de seguridad y restauración en «caliente».
- Migración de un solo paso y actualizaciones sin interrupción del servicio en forma dirigida o automática bajo condiciones de QoS.
- API REST: para integración con otras herramientas.
- VSphere Client: cambio de la antigua aplicación cliente a una GUI basada en HTML5 que asegura mejores prestaciones y servicios homogéneos entre las plataformas.
- Seguridad escalable: políticas de seguridad diseñadas para proveer el nivel de protección deseado sin complicar la gestión o administración.
- Cifrado a nivel de MV para la protección contra accesos no autorizados.

- Auditoría activa para un control total de los usuarios y acciones tanto activas como pasivas, incluyendo información para el análisis forense.

- Replicación de volúmenes virtuales.

- *Secure boot* para la prevención de inyecciones de componentes o modificaciones *on-fly*.

- Soporte para MV y contenedores en forma eficiente, integrada y sin modificaciones sobre el *guest*.

La arquitectura VMware [EAr] ESXi incluye el sistema operativo, llamado VMkernel, y los procesos que se ejecutan en la parte superior de la misma. VMkernel proporciona los medios para ejecutar todos los procesos del sistema, incluidas las aplicaciones y agentes de gestión, así como las máquinas virtuales y contenedores, controla todos los dispositivos hardware en el servidor y administra recursos para las aplicaciones. Los principales procesos que se ejecutan en la parte superior de VMkernel son:

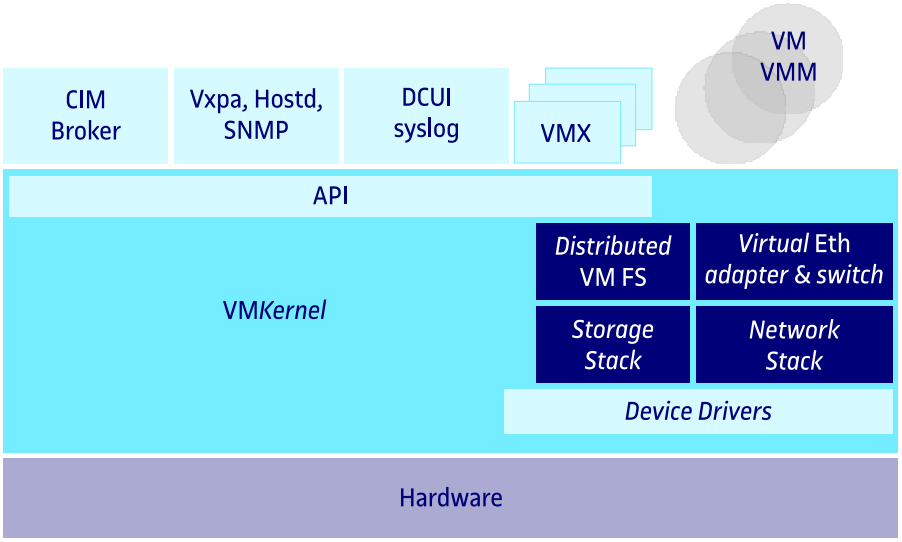
- Interfaz de usuario de consola directa (DCUI): interfaz de configuración y administración de bajo nivel, accesible a través de la consola del servidor, utilizada principalmente para la configuración básica inicial.

- Monitor de máquina virtual: es el proceso que proporciona el entorno de ejecución para cada máquina virtual, así como un proceso auxiliar conocido como VMX. Cada máquina virtual en ejecución tiene su propio proceso VMM y VMX.

- Agentes de administración: utilizados para poder realizar la administración y monitorización de alto nivel y desde aplicaciones remotas.

- Sistema de información común (CIM): es la interfaz que permite la administración a nivel de hardware desde aplicaciones remotas a través de un conjunto de API estándar.

La figura siguiente muestra los principales componentes de la arquitectura ESXi.



Uno de los aspectos, muy esperado por los usuarios, a partir de la versión 6.0u2, es una nueva herramienta (gratuita) de VMware que permite administrar los *hosts* ESXi a través de un cliente web sin necesidad de servidor vCenter (como es necesario para las versiones anteriores). La interfaz de cliente de ESXi Free Web (basada en HTML5) permite administrar un *host* sin necesidad del cliente de Windows y se incorpora con las actualizaciones o se puede instalar fácilmente. Con esta herramienta se puede gestionar todo el *host* tanto en tareas básicas como avanzadas, crear MV (desde cero o desde OVF/OVA), configuración de los parámetros del *host*, visualización de resúmenes, eventos, tareas y notificaciones/alertas, consola a MV y configuración de red, entre otras; según los expertos, aporta más flexibilidad y facilidad a la hora de gestionar el *host* que su predecesor. Esta opción, es la alternativa al vSphere Web Client basado en tecnología Flash/Flex que se tenía en las versiones anteriores, el cual se debía instalar en otra máquina (Windows o Linux) que tuviera acceso a la red donde se encontraba el servidor ESXi.

Una de las dudas habituales es en relación con los nombres de los productos y qué integran: ESX y ESXi son en esencia el mismo hipervisor con diferentes servicios y módulos que han evolucionado hacia la línea de ESXi (aunque coexistieron y VMware sigue dando soporte sobre ESX). Además, VMware tiene una línea de productos llamada VSphere donde se integran diferentes productos de virtualización, y *cloud*, que desde la versión de vSphere 4.1 (actualmente VSphere 7.0) ya no contiene la versión «clásica» de ESX sino la nueva versión de llamada ESXi (así como todas las subsiguientes versiones). Una de las grandes diferencias entre ESX y ESXi es un entorno de usuario llamado *Console Operating System* (COS) o también *Service Console* que se basaba en una distribución de Linux y que se utilizaba como interfaz de administración e interacción con el *vmkernel*. En el modelo ESXi desaparece el COS y su funcionalidad, ya sobre *vmkernel*, la provee un nuevo módulo llamado *PowerCLI + vCLI + ESXi Shell*, por lo cual el hipervisor reduce su huella, mejora las prestaciones, incrementa su seguridad y mejora la interacción con una nueva API. Si se consulta en los productos a descargar y se selecciona el ESXi de Vmware, cuando se

accede realmente a la descarga aparece como *VMware vSphere Hypervisor 7.0 U3b*. Existen otros productos gratuitos de VMware como *VCenter Converter* que permite transformar máquinas físicas (basadas en Windows y Linux) y los formatos de imágenes de terceros en máquinas virtuales de VMware, la versión de *Workstation Player* mencionada en apartados anteriores, y una aplicación de gestión de software de VMware llamada *Software Manager*.

Cuando se descarga v7.x o v6.x (disponibles en el momento de escribir este documento) pedirá una estimación de servidores físicos (limitado a 100) y con una licencia para probar todos los productos (HA, vMotion...) y su integración con otros (por ejemplo, vCenter) durante sesenta días, pero en el momento de hacer la descarga VMware, provee la licencia definitiva, que se deberá incluir sobre ESXi para que pase a estado de *Expiration date: Never*.

2.6.1. Instalación de ESXi 6.5

En cuanto a requerimientos, ESXi necesita un mínimo de 2 *cores* en un *socket* (recomendado *dualsocket* y 4 o más *cores*) y 6 GB (recomendado 8 GB) de RAM ya que, si no, no se instalará. Además, un adaptador de red 1 Gbit/s (recomendado 2) y mínimo 35 GB de disco (se recomiendan discos redundantes). También se recomienda (aunque no es necesario en primera instancia) espacio compartido por los servidores por NFS, iSCSI o Fibre Channel para el almacenamiento de las MV. ESXi tiene como valores máximos: *cores* por CPU física = sin límite, CPU por *host* = sin límite, si bien existen dependiendo de las versiones límites de vCPU por *host* y límites de vCPU por MV, pero son valores suficientemente grandes para su instalación, e incluso de alto rendimiento.

En la siguiente prueba de concepto se instalará (solo para analizar la funcionalidad y no apta para producción) y probarán las características de ESXi como una MV de KVM con 4 GB de RAM, 3 vCPU, con virtualización anidada (*host-passthrough* en el *cpu-mode* de la MV KVM), 50 GB de espacio de disco (ya que no se utilizará espacio compartido y se almacenarán las MV en el propio disco), y un dispositivo de red en modo *bridged* y de tipo *vmxnet3* (ya que, si no, durante la instalación dará un error que el dispositivo no es físico). Para cambiar este dispositivo, antes de instalar la máquina, es necesario editar su definición (no se puede hacer desde el `virt-manager`, por lo cual se puede escoger uno cualquiera, por ejemplo, e1000, y luego reemplazarlo en la definición de la máquina):

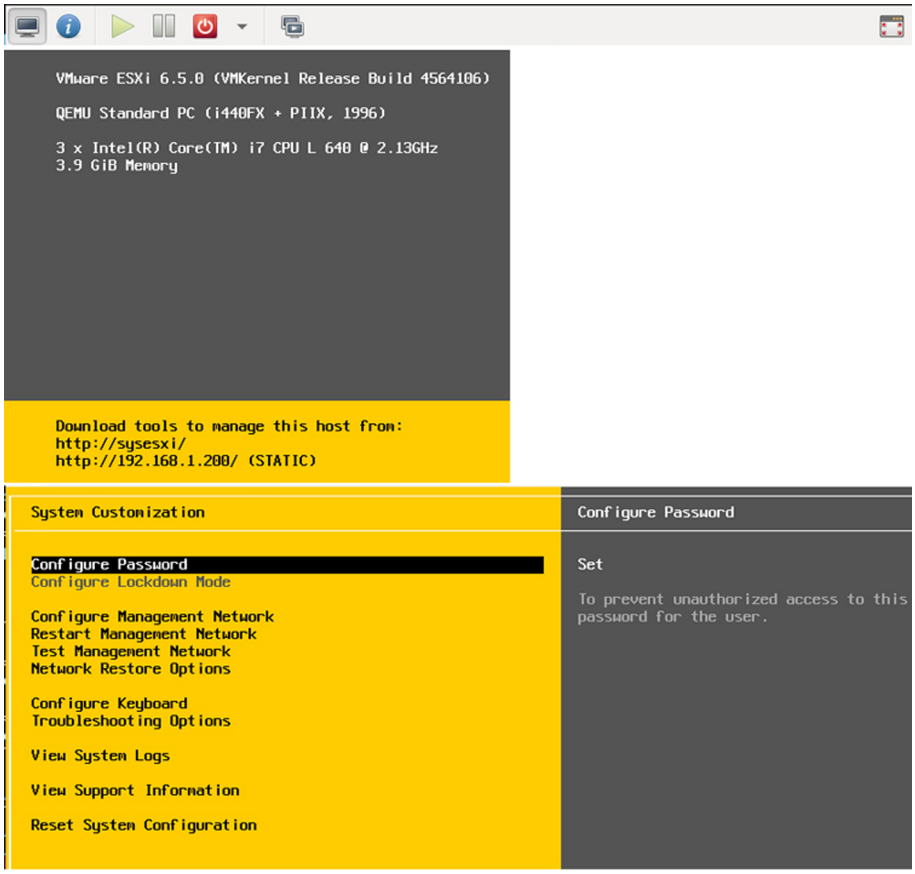
```
virsh edit nombre_maquina_esxi
```

Cambiar la interfaz de red donde indica:

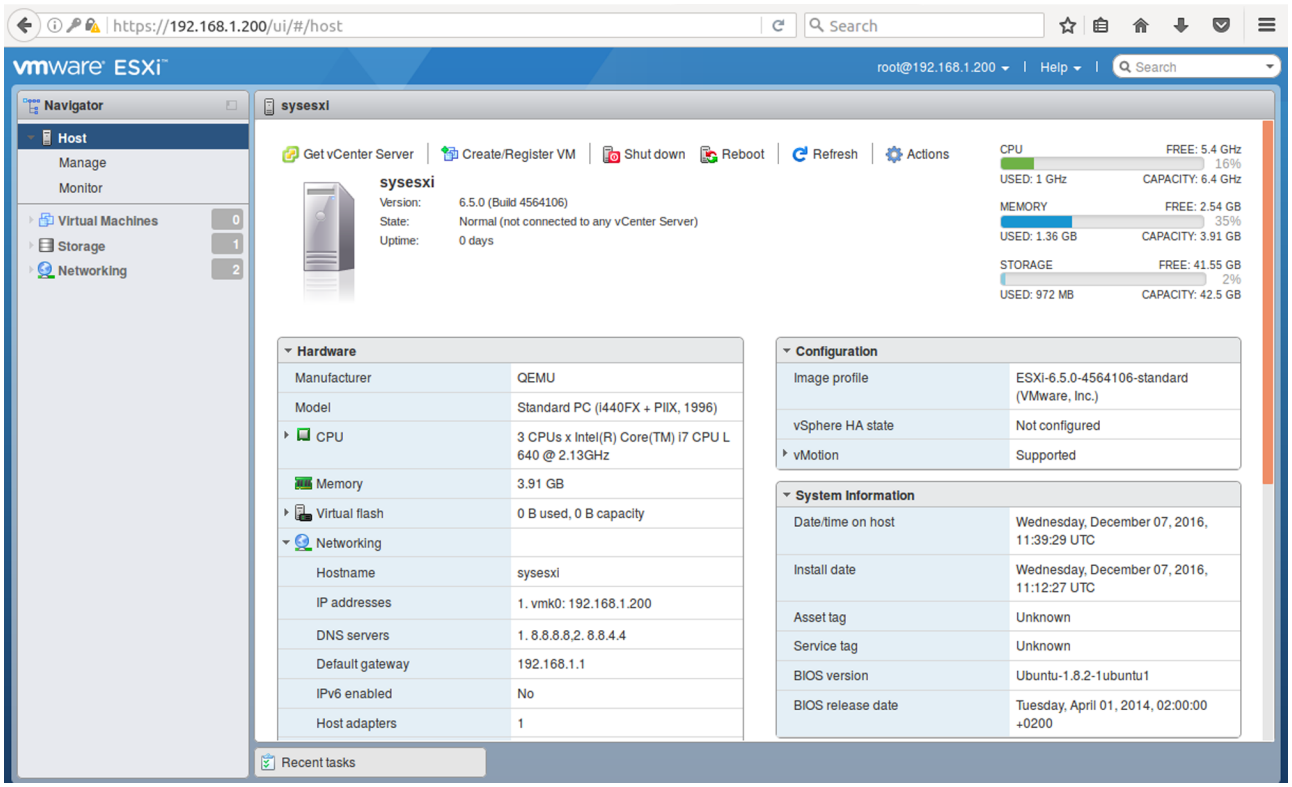
```
<interface type='bridged'>
   <mac address='aa:bb:cc:dd:ee:ff'/>
   <source bridge='dispositivo_bridge'/>
   <model type='vmxnet3'/>
```

```
    ...
</interface>
```

Con ello ya se podrá insertar la ISO descargada de VMware (luego de hacer una cuenta, verificarla y registrarse para descargar la versión deseada) e iniciar la instalación. Luego de algunas preguntas (disco, *passwd*...) tendremos la versión instalada con la habitual consola texto (gris y amarillo de Vmware) y la URL para conectarse a la interfaz de administración. En primera instancia, la IP las gestionará por DHCP, pero si no hay un servicio disponible asignará una interna, por lo cual será necesario acceder a la consola del ESXi y configurarla. Para ello, presionar F2, autentificarse y ya se dispondrá de la consola para hacer los cambios necesarios (y otros comandos de gestión mínima del servidor). Las figuras siguientes muestran la interfaz de ESXi y la de consola una vez presionado F2 y autentificado.

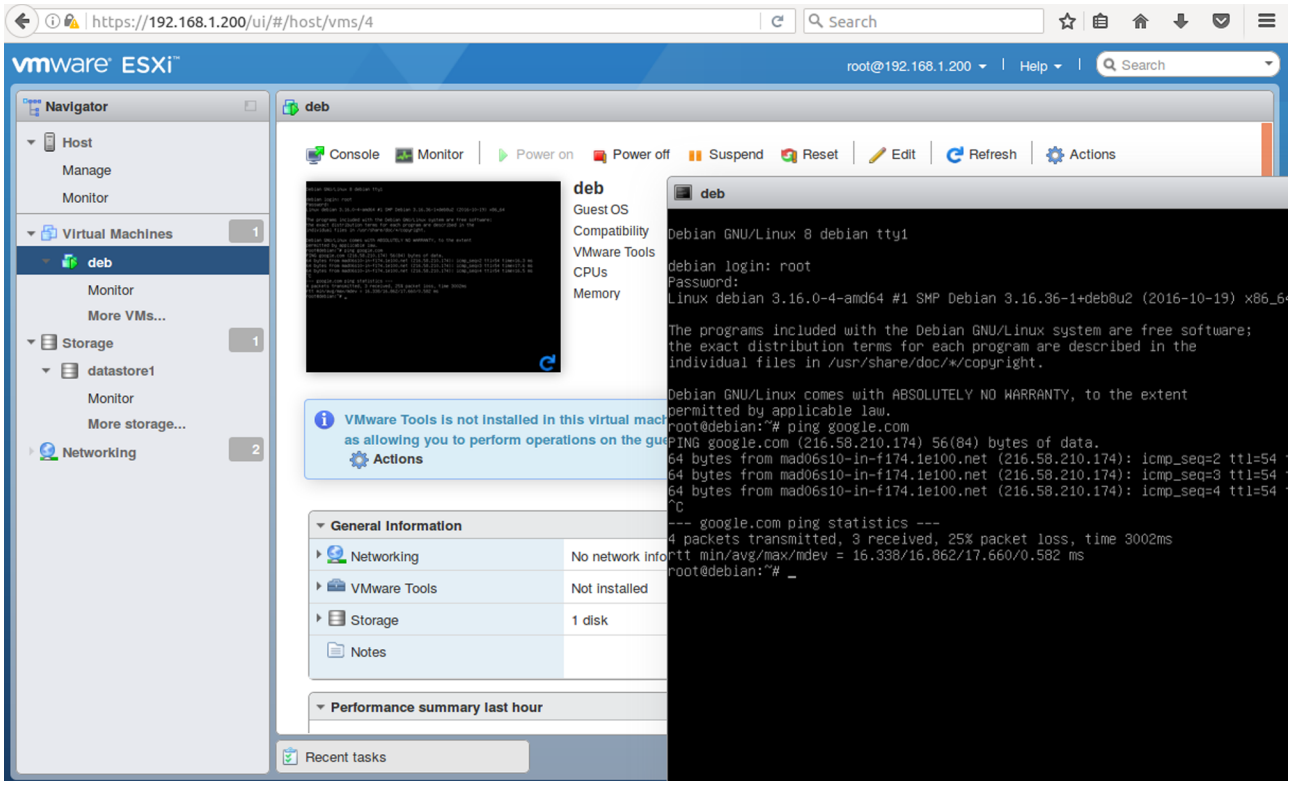


Una vez configurado se puede acceder desde la URL (después de aceptar el certificado de https y autentificarse) indicada mostrando la interfaz de gestión y administración de ESXi, como muestra la figura siguiente:



Es importante tener presente la inclusión de la licencia definitiva antes de los sesenta días, la cual se obtiene de la misma página que se ha hecho la descarga. Para ello acceder a *Host* → *Manage* → en el *tab Licensing* e introducir la licencia obtenida en la página de descarga y se observará que el estado pasa a *Expiration date: Never* y con las X vCPU por MV (*up to X-way virtual SMP*). Cabe tener en cuenta que ya no es necesario el antiguo cliente VSphere sobre Windows para conectarse al servidor ESXi, ya que este ha sido reemplazado por el nuevo que funciona sobre la interfaz web y es mucho más ágil y flexible.

Una vez disponible la interfaz web, el funcionamiento es similar a los otros hipervisores ya vistos y se podrá crear una red asociada al NIC, y crear una MV cargando la ISO desde el disco local al servidor, definir todos sus parámetros y configuración y ponerla en marcha. Las posibilidades de conexión podrán ser a través del navegador (incluso desde una máquina remota), `ssh`, o utilizando un producto VMware *Remote Console* (VMRC de pago tanto para W, Linux o MacOS). Sobre la MV instalada, se recibirá un mensaje que las VMware Tools no están instaladas y se recomienda hacerlo para mejorar la interacción ESXi-MV (básicamente información, encendido, apagado, reinicio, entre otras). La figura siguiente muestra una imagen de ESXi ejecutando una Debian con red en modo *bridged* y saliendo por el dispositivo de red (y a la cual se puede acceder externamente mediante la URL indicada, en este caso *https://192.168.1.200/ui/#/console/4*).



Una máquina virtual en VMware ESXi, como se puede apreciar en el *datastore*, es básicamente un directorio con el nombre de la máquina con un conjunto de ficheros (texto y binarios) que conforman la MV con el sistema operativo y aplicaciones incluidas. Las extensiones de los ficheros más importantes que componen una máquina virtual son: vmx (fichero de configuración), vmdk (fichero de disco virtual), nvram (fichero BIOS -*Basic Imput/Output System*-), log (fichero de registro de eventos y errores de la MV). En relación con el hardware disponible, se puede configurar de 1 a 8 VCPus, 1 TB de RAM, controlador Scsi, IDE, SSD (NVMe), CD/DVD, USB, controlador gráfico, controlador de red, puertos series y paralelos, controlador de sonido, ratón y teclado (para todos ellos, utilizando los diferentes controladores incluidos o pudiendo además agregar controladores de terceros compatibles).

Otro modo de interacción con la plataforma es a través de la línea de comandos (CLI) utilizando los comandos ESXCLI (`esxcli`), `vmk*` y Datacenter CLI (DCLI, para VCenter), entre otros, que permiten realizar todas las tareas de administración necesarias (sobre ESXi: `excli y vmk*`) con gran detalle y eficiencia para administradores expertos [Cli].

Es importante tener en cuenta que en esta prueba todo es virtualizado, ya que la Debian accede al NIC virtual de ESXi (en modo *bridged*) que, a su vez, accede a NIC virtual de KVM (en modo *bridged*), la cual está asociada al dispositivo de red real; sin embargo, como se puede ver, su funcionalidad es probada y no habrá diferencia entre esta instalación y una que se realice sobre *bare-metal*. Como se podrá apreciar, no se le pueden pedir demasiadas prestaciones;

no obstante, la fluidez y ejecución son más que aceptables (a pesar de todas las virtualizaciones que se realizan) y demuestra una plataforma consolidada, eficiente y simple de administrar y gestionar.

Como ya se ha mencionado en otros hipervisores, las potencialidades de ESXi son muy grandes y en este apartado solo se ha realizado una pequeña demostración de las posibilidades de virtualización que ofrece, pero es necesario consultar la documentación [Ves] (y la propia interfaz de administración) para tener en cuenta las prestaciones y características que puede ofrecer la plataforma (incluso la gratuita) que la convierten en un candidato de excepción para diferentes escenarios y configuraciones, ya sean para pequeñas, medianas o grandes instalaciones.

3. Contenedores

Cuando la virtualización se realiza a nivel del sistema operativo se obtienen los contenedores (o virtualización basada en contenedores) que son el resultado de aplicar una capa de virtualización sobre el núcleo del sistema operativo que permite que existan múltiples instancias aisladas de espacios de usuario, en lugar de solo uno. Estas instancias se denominan *contenedores* (utilizado LXC, Docker), pero también reciben otros nombres, como *zonas* (Solaris), *servidores privados virtuales* (OpenVZ), *particiones/entornos virtuales* (VEs), *núcleo virtual* (DragonFly BSD) o *jaulas* (*jail* FreeBSD).

La primera referencia que se tiene como virtualización a nivel del sistema operativo es la utilización del chroot el cual recibe los siguientes argumentos `chroot [OPTION] NEWROOT [COMMAND [ARG]...` y permite ejecutar el comando pasado como parámetro impidiendo que él y sus procesos hijos accedan a cualquier ruta fuera del directorio pasado como argumento. Esta forma de ejecutarse es conocida como *jaula chroot*. Basado en estas ideas FreeBSD, a principio de los años 90, introduce jaulas (*jails*) con el objetivo de solucionar los problemas/limitaciones del chroot, ya que este solo limita el sistema de archivos a los que pueden acceder los procesos pero no el resto de recursos. Las jaulas BSD extienden este modelo virtualizando, además del acceso al sistema de ficheros, al resto de recursos (usuarios, subsistema de red, procesos, etc.) incluso el rol del propio usuario *root* dentro y fuera de la jaula, lo cual permite crear un entorno virtual igual a la máquina real (o máquina virtual real) con la única limitación que, a diferencia de las máquinas virtuales, ejecutan la misma versión del núcleo del SO.

Estas ideas generan nuevos desarrollos, como los *namespaces* (2002) que permiten particionar los recursos del *kernel* de modo que un grupo de procesos vea un conjunto de recursos diferente a los que vea otro conjunto de procesos y, en 2006, Google publica para el SO Linux, *Process Containers* que permite limitar y aislar los accesos a recursos de la máquina (CPU, memoria, I/O de disco, red, etc.) a un grupo de procesos (posteriormente, por cuestiones de nombres, los *Process Containers* fueron renombrados como *Control Groups* o simplemente cgroups (2007)) integrándose con el kernel de Linux. Si se ejecuta `systemd-cgls` sobre Linux, mostrará el árbol de grupos y sus dependencias.

En 2008 se ofrece para sistemas Linux, LinuX Containers (LXC), que utilizando *namespaces* y *cgroups* del kernel, permite crear entornos de gestión de contenedores y, en 2013 y después de algunos años de desarrollo, se publica Docker, que permite gestionar el ciclo de vida de los contenedores de forma sencilla en comparación con las herramientas anteriores teniendo gran aceptación por los desarrolladores y un crecimiento exponencial en su uso, ya que permite dentro de los contenedores, empaquetar aplicaciones aisladas entre sí con ca-

pacidad total de ejecución y, además, permite que puedan ser integradas en los flujos de integración/distribución continua generando un único flujo desde el desarrollo de aplicaciones hasta su puesta en producción.

Docker usó como entorno de ejecución por defecto LXC. Sin embargo, más tarde fue reemplazado por libcontainer permitiendo utilizar otras tecnologías de aislamiento (distintas a las de LXC) y poder acceder directamente a las API del kernel del sistema operativo reduciendo sus dependencias de librerías y aumentando la eficiencia del conjunto.

No son los únicos sistemas basados en la virtualización del SO. Apache Mesos abstrae la CPU, la memoria, el almacenamiento y otros recursos informáticos de las máquinas (físicas o virtuales), lo que permite que se pueda construir un sistema distribuido elástico y tolerante a fallas y que se ejecuten fácilmente de manera efectiva. Este sistema está especialmente orientado a entornos de Big Data.

Windows también ha desarrollado su propio ecosistema (*Windows Server Container/Hyper V Container*). No obstante, la versión de Docker para Windows que utiliza la propia infraestructura de Windows es la referencia actual en el SO Windows.

Probablemente, en el mundo de los contenedores, el hecho más significativo después de la creación de estos fue (en 2014) la apertura por parte de Google del proyecto Kubernetes (también conocida como K8s) como una plataforma portátil, extensible y de código abierto que permite administrar cargas de trabajo y servicios en contenedores y facilita una configuración declarativa como la automatización. Esta plataforma se ha convertido en el referente en la industria para el despliegue basado en contenedores, la gestión de cargas y la escalabilidad horizontal y es uno de los entornos más utilizados en los procesos de integración continua del desarrollo de software.

Los expertos clasifican los sistemas de contenedores como:

- **Contenedores de infraestructura**, como LXC/LXD, donde se permite ejecutar múltiples instancias de sistemas operativos de manera aislada permitiéndole acceder a los recursos (CPU, network, I/O) en forma similar a una máquina virtual pero más rápido y ligero. Ejemplos de estos sistemas son LXC y LXD.

- **Contenedores de procesos o aplicaciones**, como `systemd-nspawn` o Docker (por defecto en todo Linux), que ofrecen un sistema de encapsulado de aplicaciones, permiten acelerar su desarrollo y distribución eliminando las diferencias de entorno entre producción y desarrollo y facilitan la migración del entorno.

- **Contenedores *sandbox***, como FireJail, Bubblewrap, nsroot, nsjail, que permiten el aislamiento mediante un entorno donde se tiene acceso restringido a recursos del sistema operativo o datos del usuario (*sandbox*).

3.1. Linux Containers, LXC

LXC (Linux Containers) es un método de virtualización en un nivel del sistema operativo para ejecutar múltiples sistemas Linux aislados (llamados contenedores) sobre un único *host*. El *kernel* de Linux utiliza `cgroups` para poder aislar los recursos (CPU, memoria, E/S, *network*, etc.), lo que no requiere iniciar ninguna máquina virtual. `Cgroups` también provee aislamiento de los espacios de nombres para aislar por completo la aplicación del sistema operativo, incluidos el árbol de procesos, la red, los ID de usuarios y los sistemas de archivos montados. Mediante una API muy potente y herramientas simples, permite crear y gestionar contenedores de sistema o aplicaciones.

LXC utiliza diferentes módulos del *kernel* (ipc, uts, mount, pid, network, user) y de aplicaciones (p. ej. Apparmor, SELinux profiles, Seccomp policies, *chroots -pivot_root-* y *control groups –cgroups–*) para crear y gestionar los contenedores. Se puede considerar que LXC está a medio camino entre un «potente» chroot y una máquina virtual, y ofrece un entorno muy cercano a un Linux estándar pero sin necesidad de tener un *kernel* separado. Esto es más eficiente que utilizar virtualización con un hipervisor (KVM, VirtualBox) y más rápido de (re)iniciar, sobre todo si se están haciendo desarrollos y es necesario hacerlo frecuentemente, y su impacto en el rendimiento es muy bajo (el contenedor no ocupa recursos) y todos se dedicarán a los procesos que se estén ejecutando.

Dado que LXC está soportado por Ubuntu, la instalación y configuración es muy simple, como se puede observar en la *Getting Started*.

Los comandos son muy simples y para crear un contenedor Debian simplemente se ejecuta: `sudo lxc-create -t download -n u1 -- --dist ubuntu --release DISTRO-SHORT-CODENAME --arch amd64`

Se podrá observar que después de unos instantes, en el directorio */var/lib/xlc* se tiene un directorio con el nombre del contenedor y con no más de 300 MB de tamaño (dependerá del tipo de contenedor pero son valores habituales). Es importante observar la información que da del contenedor cuando se crea, ya que proporcionará información útil sobre usuarios y *passwd*. Los comandos más útiles para trabajar con contenedores son:

- `lxc-ls` para mostrar los contenedores y `lxc-info` para obtener información sobre ellos. (los que se están ejecutando se deben agregar --active).

- `lxc-start -n debian8 -d` para poner en marcha el contenedor en background (`-d = daemon`).

- `lxc-console -n debian8` para conectarse al contenedor (con el *passwd* generado para el *root* durante su creación). Para volver al SO del *host* presionar las teclas Ctrl+a y después q.

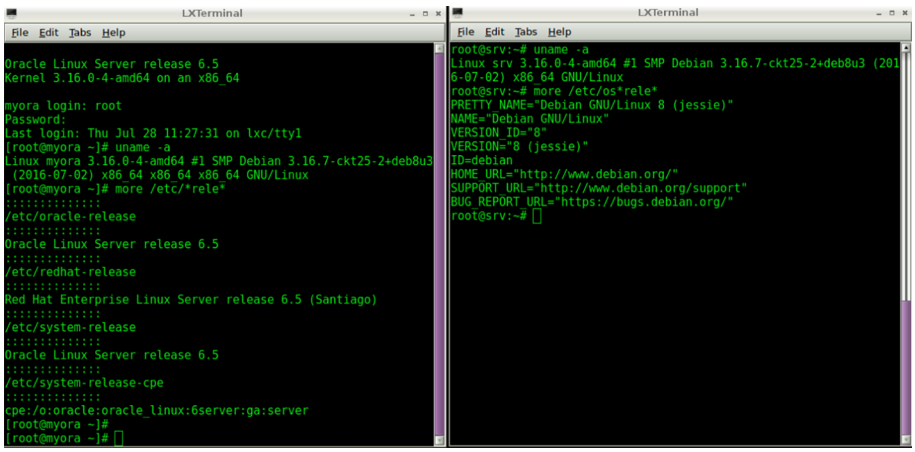
- `lxc-stop -n debian8` para la ejecución del contenedor.

- `lxc-destroy -n debian8` elimina el contenedor.

- `lxc-clone debian8 debian8V2` clona un contenedor.

- Para montar un directorio del *host* en un contenedor, agregar en */var/lib/lxc/containername/config* la sentencia *lxc.mount.entry = /usr/bin /mnt none ro, bind 0.0* que montará /usr/bin de *host* en el directorio /mnt del contenedor.

La figura siguiente muestra el *host* a la derecha (Debian) y el contenedor a la izquierda (Oracle), funcionando sobre el mismo núcleo (`uname -a` desde cada uno).



Es interesante instalar el paquete `lxctl` (`apt-get install lxctl`) ya que permite gestionar de forma más simple los contenedores LXC.

3.2. Docker

Docker es una plataforma abierta para el desarrollo, empaquetado y ejecución de aplicaciones de forma que se puedan poner en producción o compartir más rápidamente y que separa estas de la infraestructura. Es por ello que el costo es mucho menor en recursos (básicamente espacio de disco, CPU y arranque) y permite que los desarrolladores puedan disponer de todo tal y como lo ha dejado el anterior desarrollador sobre su infraestructura. Esto significará menos tiempo para probar y acelerará el despliegue acortando de forma significativa el ciclo entre que se escribe el código y pasa a producción. Docker emplea una plataforma (contenedor) de virtualización ligera con flujos de trabajo

y herramientas que la ayudan a administrar e implementar las aplicaciones y proporcionar una forma que casi cualquier aplicación se podrá ejecutar de forma segura en un contenedor aislado.

Este aislamiento y seguridad permiten ejecutar muchos contenedores de forma simultánea en el *host* y, dada la naturaleza (ligera) de los contenedores, todo ello se ejecuta sin la carga adicional de un hipervisor (que sería la otra manera de gestionar estas necesidades compartiendo la VM), lo que significa se pueden obtener mejores prestaciones y utilización de los recursos. Es decir, con una VM, cada aplicación virtualizada incluye no solo la aplicación, que puede ser decenas de Mbytes –binarios + bibliotecas–, sino también el sistema operativo *guest*, que pueden ser varios Gbytes; en cambio, en Docker solo será la aplicación y sus dependencias, que se ejecutan como un proceso aislado en el espacio de usuario del SO *host*, compartiendo el *kernel* con otros contenedores. Por tanto, tiene el beneficio del aislamiento y la asignación de recursos de las máquinas virtuales, pero es mucho más portátil y eficiente transformándose en el entorno perfecto para apoyar el ciclo de vida del desarrollo de software, test de plataformas, entornos, etc. [d1].

El entorno está formado por dos grandes componentes: Docker [d2] (es la plataforma de virtualización –contenedor–) y Docker Hub [d4] (una plataforma SasS que permite obtener y publicar/gestionar contenedores ya configurados). En su arquitectura, Docker utiliza una estructura cliente-servidor en la que el cliente (CLI Docker) interactúa con el *daemon* Docker, que hace el trabajo de la construcción, ejecución y distribución de los contenedores de Docker. Tanto el cliente como el *daemon* se pueden ejecutar en el mismo *host*, o se puede conectar un cliente a un *daemon* de Docker remoto. El cliente Docker y el servicio se comunican mediante *sockets* o una API RESTful. Para tener más detalles sobre la arquitectura se puede consultar https://docs.docker.com/get-started/overview/.

Es importante no confundir una aplicación llamada docker en Debian con la Plataforma Docker (para evitar confusiones en Debian se llama docker.io).

Su instalación es sumamente fácil, ya que Docker mantiene los binarios para cada distribución y el proceso de instalación se puede consultar de la página web del desarrollador ya sea para Linux, MacOS o Windows.

La forma más simple de verificar su funcionamiento es ejecutar un contenedor llamado hello-world simplemente poniendo `docker run hello-world` que se descargará una imagen del contenedor del Docker-Hub, lo pondrá en marcha y mostrará una pantalla de bienvenida. También se puede hacer una prueba con contenedor Ubuntu `docker run -it ubuntu bash` (como el contenedor no existe lo descargará y lo ejecutará, por lo que veremos el *prompt* del *bash* de Ubuntu). Si hacemos `more /etc/os-release`, se podrá apreciar la versión del Ubuntu instalado. Con `exit` (o Ctrl-D) se sale al *host*.

Es importante el concepto de *imagen de un contenedor* y el *contenedor* propiamente dicho: la imagen contiene todos los elementos y configuraciones para crear el contenedor pero no es el contenedor. El contenedor se crea a partir de la imagen y puede ser modificado, reiniciado o parado, pero los cambios que se realicen en él no se reflejarán en la imagen que solo es de lectura. Si se quieren guardar los cambios en un contenedor se deberá hacer una nueva imagen con el nuevo contenedor (docker commint)

Las órdenes iniciales para trabajar con contenedores son:

- `docker`: muestra todas las opciones.
- `docker images`: lista las imágenes localmente.
- `docker search patrón`: busca contenedores/imágenes que tengan este patrón.
- `docker pull nombre`: obtiene imágenes del *hub*. El nombre puede ser `<username>/<repository>` para los particulares.
- `docker run name cmd`: ejecutará el contenedor '*name*' y dentro, el comando indicado por cmd.
- `docker ps -l`: obtiene la info de un contenedor (--all de todos los contenedores incluso los que están parados).
- `docker commit ID name`: crea una del contenedor con todas la modificaciones que se han hecho (con 3-4 números del ID es suficiente).
- `docker inspect`: permite ver la configuración exhaustiva de un contenedor/imagen.
- `docker push`: sube una copia de la imagen en el Docker-hub (se debe tener un usuario –gratuito–) y estará disponible para otros usuarios/*hosts* que quieran instalarla.
- `docker cp`: copia archivos/directorios desde el contenedor al *host*.
- `docker export/import`: exporta/importa el contenedor en un archivo tar.
- `docker history`: muestra la historia de una imagen.
- `dockerinfo`: muestra información general (imágenes, directorios, etc.).
- `docker restart`: reinicia un contenedor.
- `docker rm ID`: elimina un contenedor.
- `docker rmi ID`: elimina imágenes.
- `docker start/stop`: inicia/para un contenedor.

Como prueba funcional se realizará la instalación de un servidor Apache sobre un contenedor Docker. Para ello comenzaremos con un contenedor base Ubuntu, se instalará Apache2 y, después, accederemos desde el *host* haciendo un mapeo de los puertos del contenedor. Para ello:

1) Se ejecuta `docker run -it ubuntu /bin/bash` para acceder al contenido. Si no está instalado, se descargará y se instalará.

2) Dentro del contenedor ejecutamos `apt-get update` y, a continuación, se instala Apache2 (también se instalará el editor Vi) con `apt-get install apache2 vim`. Se puede probar que Apache2 se arranca sin problemas con `systemctl restart apache2` (normalmente solo mostrará una advertencia, que no tiene el ServerName configurado). También se puede ver con los comandos `ps -edaf` o `systemctl status apache2`.

3) Se modifica la página inicial para poner algo relativo a Docker y, de este modo, poder identificar la página en /var/www/html/index.html.

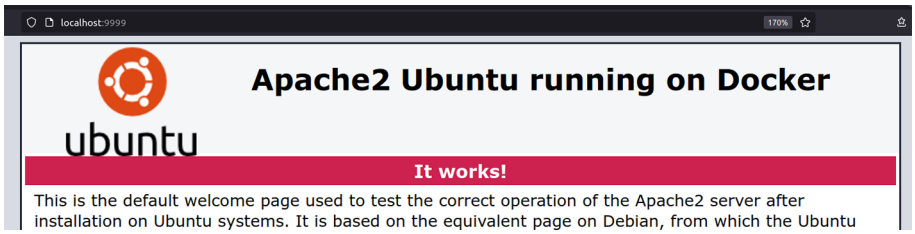
4) Es importante no salir del contenedor, sino hacer Ctrl+P y, después Ctrl+Q para salir al *host*, ya que, si hace Ctrl+D, el contenedor terminará su ejecución (también se puede realizar desde otro terminal).

5) Con el contenedor en marcha, verificamos que está en UP con `docker ps` y se ejecuta `docker commit ID_contenedor apache`, donde `ID_contenedor` es el número que muestra `docker ps`, y solo se deben poner los 3-4 primeros números. Este comando salvará una nueva imagen llamada apache, que podemos verificar con `docker images`.

6) Ahora ya es posible salir del contenedor (Ctrl+D o exit) o pagarlo con el comando `docker stop ID_contenedor`; `docker ps` deberá mostrarnos que no hay ningún contenedor Ubuntu en ejecución.

7) Para iniciar un contenedor a partir de la imagen creada se ejecuta: `docker run -d -p 9999:80 apache /usr/sbin/apache2ctl -D FOREGROUND` donde con -d indica que se ejecute en modo *daemon*, con -p se mapean el puerto del contenedor (80) al del host(9999); y luego se ejecuta el comando `apache2ctl` indicándole que ponga en marcha Apache2 en modo *Daemon* y en *FOREGROUND*.

8) Finalmente desde el *host* se puede abrir un navegador con localhost: 9999 o con el nombre de dominio del *host* (si se ha actualizado /etc/hosts como por ejemplo srv.nteum.org:9999), y veremos la página modificada del contenedor.



Glosario

API web RESTful Forma de proveer interoperabilidad entre servicios y clientes en internet.

appliances Aplicación software que puede ser combinada con JeOS (*just enough operating system*) para ejecutarse sobre un servidor o una máquina virtual.

bare-metal Máquina hardware «desnuda», es decir, sin núcleo (SO) instalado.

benchmarks Programa o conjunto de ellos que permiten evaluar determinadas prestaciones de un dispositivo.

bridged networking Entorno de red configurado para que los múltiples clientes puedan tener un IP dentro de la red sobre un mismo dispositivo físico.

CLI (*command line interface*) Línea de comandos o de consola texto.

cmdlets Comandos utilizados en Hyper-V.

cloud, cloud computing Conocida también como servicios en la nube, informática en la nube o nube de cómputo, es un paradigma que permite ofrecer servicios IT a través de una red, que usualmente es internet.

containers Entorno software virtualizado a nivel del SO.

control de instrumentación (**WMI**), **bus de máquina virtual** (**VMbus**), **proveedor de servicios de virtualización** (**VSP**), **controlador de infraestructura virtual** (**VID**), **Microsoft Management Console** (**MMC**), módulos que conforman la infraestructura del hipervisor Hyper-V.

dashboard Panel de control/instrumentos de un entorno.

DHCP Servicio que permite asignar los parámetros de red e información relativa en un esquema cliente servidor.

Docker y LXC/LXD Plataformas que permiten la virtualización a nivel del SO.

ESXi y vCenter *Server* Hipervisor y entorno de gestión de VMware.

ESXCLI `(esxcli)`, **`vmk*`** y **Datacenter CLI** (DCLI, para VCenter) Diferentes modos de interacción entre el administrador y un host ESXi.

extensiones hardware VT-x/AMD-V Conjunto de instrucciones introducidas por Intel y AMD para dar soporte a la virtualización que permite acceder a los recursos del procesador ganando en prestaciones y eficiencia.

front ends Entorno software que interacciona con los usuarios.

GlusterFS Sistema de archivo distribuido escalable.

guest Cliente (SO) de un sistema virtualizado.

hardware assisted virtualization Técnica de virtualización que utiliza las extensiones hardware del procesador para mejorar las prestaciones.

high availability cluster Conjunto de servidores que pueden prestar un servicio ininterrumpido y sincronizado.

hipervisores (*hypervisors*) Plataforma que permite aplicar diversas técnicas de control de virtualización para utilizar, al mismo tiempo, diferentes sistemas operativos.

host Máquina (SO) que integra el hipervisor y ofrece un servicio de virtualización.

host-passthrough Parámetro utilizado por libvirt para permitir la virtualización anidada.

Hyper-Threading (H Technology, SMT) ® de la empresa Intel que permite a programas preparados para ejecutar múltiples hilos (*multi-threaded*) procesarlos en paralelo dentro de un único procesador, incrementando el uso de las unidades de ejecución del procesador.

Hyper-V Hipervisor sobre SO Windows.

interfaz de usuario de consola directa (DCUI) Interacción entre el usuario y el SO/ hipervisor.

IP address Número o dirección (lógica) que identifica un dispositivo en una red.

iSCSI (internet SCSI) Estándar que permite el uso del protocolo SCSI sobre redes TCP/IP definido dentro de la capa de transporte.

ISO Archivo que contiene una copia o imagen exacta de un sistema de archivos. Utilizado para la instalación de software o SO.

JSON-Schema Provee las reglas para un archivo en formato JSON requerido por una aplicación o servicio y define cómo este puede ser modificado.

kernel Núcleo del SO.

Kernel-based Virtual Machine (KVM) Infraestructura de virtualización que permite transformar a Linux en un hipervisor de altas prestaciones.

KWatt (KiloWatt o kilovatio) Unidad de medida de potencia.

micronúcleo (*microkernel*) Código software esencial (mínimo) que puede proporcionar los mecanismos necesarios para implementar un sistema operativo (SO).

Microsoft Remote Desktop Protocol (RDP) Protocolo de Microsoft para la comunicación en la ejecución de una aplicación entre un terminal y un servidor.

monitor de máquina virtual (VMM, VMX) Proceso que proporciona el entorno de ejecución para cada máquina virtual, así como un proceso auxiliar conocido como VMX sobre ESXi.

multi-master cluster (para garantizar la disponibilidad) Sistema multiservidor para garantizar la alta disponibilidad.

NAT (*Network Address Translation*) Mecanismo que permite que un dispositivo en una red se pueda conectar a otra red (mecanismo implementado por los *routers*).

nested virtualization Virtualización anidada que permite a un hipervisor poder «traspasar» la interfaz hardware a su MV permitiendo que, por ejemplo, se pueda ejecutar otro hipervisor.

networking Entorno de red en un SO.

NFS, iSCSI LUN, Ceph RBD, Sheepdog Tecnología de acceso a dispositivos de almacenamiento distribuido.

NIC Network Interface Card.

no SPOF (*no single point of failure*) Evitar que cuando una parte del sistema falle este haga que todo el sistema falle.

open source Software distribuido y desarrollado libremente que se centra en los beneficios prácticos de acceso al código fuente más que en cuestiones éticas o de libertad que son la esencia del software libre.

overhead Combinación de tiempo de cálculo excesivo o indirecto, memoria, ancho de banda u otros recursos que se requieren para alcanzar una meta en particular.

paravirtualización Técnica que permite mostrar una interfaz virtualizada a un SO o parte de él similar al hardware virtualizada, pero no idéntica.

password Palabra clave.

personal use and evaluation license (PUEL) Licencia de uso libre a determinados productos.

plugins Complemento que permite agregar una funcionalidad a un software o a un SO.

PowerShell Shell de Windows similar a los intérpretes de comandos de Linux.

QEMU Emulador de procesadores basado en la traducción dinámica de binarios (conversión del código binario de la arquitectura fuente en código entendible por la arquitectura huésped) y con posibilidades de virtualización.

QoS (*quality of service*) Índice que mide la calidad de un servicio.

red host-only Red que permite comunicar un guest y el host a través de una interfaz virtual.

ROI (*return on investment*) Relación que compara el beneficio o la utilidad obtenida en relación con la inversión realizada.

root Usuario administrador en Unix. Directorio principal en sistemas *Nix identificado por el símbolo '/'.

servicios IT Se denomina así a todos los servicios de las tecnologías de la información que comprende un conjunto de actividades (que responden a necesidades de un cliente) utilizando sistemas informáticos. Algunos autores utilizan servicio TI o simplemente TI.

sistema de información común (CIM) Módulo de comunicación entre diferentes módulos en ESXi.

sistemas virtualizados Son aquellos que se ejecutan sobre un hipervisor.

SLA (*service level agreement*) Contrato escrito entre un proveedor de servicio y su cliente con objeto de fijar el nivel acordado para la calidad de dicho servicio.

snapshots Estado/imagen congelada de una MV y que es posible recuperar al punto en que se realizó.

SSD (NVMe) *Non-Volatile Memory Host Controller Interface Specification* (NVMHCI) Es una especificación para el acceso a las unidades de estado sólido (SSD) conectadas a través del bus PCI Express (PCIe).

TCO (*total cost of ownership*) Método de cálculo para determinar los costes directos e indirectos, así como los beneficios, relacionados con la compra de equipos o programas informáticos.

Turnkey Repositorio de *appliances*.

vCPUs Cpus virtuales.

VirtIO *Paravirtualized drivers* for kvm/Linux.

virtualización Capa de software que ofrece una versión virtual de los recursos hardware/software subyacente.

Virtual Network Computing (VNC) Programa basado en una estructura cliente-servidor que permite tomar el control del ordenador servidor remotamente a través de un ordenador cliente.

VirtualBox Hipervisor *open source* desarrollado y mantenido por Oracle (antes Sun Microsystems).

VirtualBox *Remote Display Protocol* (VRDP) Implementación de Virtualbox de RDP.

VMware ESXi Hipervisor de VMware integrado dentro de la línea de productos VSphere.

VMware *Remote Console* Aplicación Windows que provee acceso a la consola de las máquinas virtuales sobre un host remoto.

VMware Workstation Player Hipervisor para instalar sobre un SO (*hosted*).

Volume Group/Logical Volumes Parte del Logical Volume Manager (LVM) que es un módulo que permite la gestión de volúmenes sobre el *kernel* de Linux.

Xen Proyecto de virtualización orientado a la eficiencia mediante una técnica llamada paravirtualización.

Xenserver Hipervisor *open source* desarrollado y mantenido por Citrix.

WebSockets Canal de comunicación bidireccional sobre un *socket* TCP.

wireless Tecnología de red inalámbrica.

ZFS Sistema de archivo que permite gran tamaño, número de archivos y de gran tamaño.

Bibliografía

Bibliografía básica

Gavanda, M. *et al.* (2019). *Mastering VMware vSphere 6.7: Effectively deploy, manage, and monitor your virtual datacenter with VMware vSphere 6.7*. Packt Publishing

Ivanov, K. (2017). *KVM Virtualization Cookbook*. Packt Publishing.

Nickoloff, J.; Kuenzli, S. (2019). *Docker in Action*. Manning.

Portnoy, M. (2016). *Virtualization Essentials*. Sybex (Wiley).

Webgrafía

Todos los enlaces han sido visitados en noviembre de 2021.

[ApM] Apache Mesos. <http://mesos.apache.org/>

[Cli] VSphere CLI. <https://pubs.vmware.com/vsphere-60/topic/com.vmware.ICbase/PDF/vsphere-esxi-vcenter-server-60-command-line-interface-concepts-examples-guide.pdf>

[Cpv] Comparison of platform virtualization software. <https://en.wikipedia.org/wiki/Comparison_of_platform_virtualization_software>

[Cwp] Computing with a price tag: VM cost calculation guide. S. Bigelow. 2016. Techtarget.com. <http://searchservervirtualization.techtarget.com/feature/Computing-with-a-price-tag-VM-cost-calculation-guide>

[Dbr] Bridging Network Connections. Debian. <https://wiki.debian.org/BridgeNetworkConnections>

[EAr] Architecture of VMware ESXi. <http://www.vmware.com/techpapers/2007/architecture-of-vmware-esxi-1009.html>

[Hya] Arquitectura de Hyper-V. Microsoft. <https://msdn.microsoft.com/en-us/library/cc768520(v=bts.10).aspx https://www.microsoft.com/en-us/download/details.aspx?id=29189>

[Hyp] Virtualization using Hyper-V on Windows 10. <https://msdn.microsoft.com/virtualization/hyperv_on_windows/index>

[Hvs] Hyper-V Server Evaluations. <https://www.microsoft.com/en-in/evalcenter/evaluate-hyper-v-server-2016 https://technet.microsoft.com/library/mt169373.aspx>

[Hyv] Hyper-V. <https://technet.microsoft.com/library/mt169373.aspx>

[Iar] Iconos con licencia de uso libre. <http://www.iconarchive.com> <http://www.customicondesign.com> <http://icons8.com>

[Kvm] Kernel Virtual Machine. 2016. <http://www.linux-kvm.org/>

[Kne] KVM Networking Ubuntu. <https://help.ubuntu.com/community/KVM/Networking>

[Ksw] Kvm on Server World. <https://www.server-world.info/en/note?os=Ubuntu_16.04&p=kvm&f=5>

[Lib] Libvirt Networking. 2016. <https://wiki.libvirt.org/page/Networking>

[Lvh] Libvirt Networking Handbook. Version 1.0.1. Jamie Nguyen. 2015. <https://jamielinux.com/docs/libvirt-networking-handbook/>

[NoV] NoVNC. <https://kanaka.github.io/noVNC/>

[Mrs] Manage Remote Hyper-V Hosts with Hyper-V Manager. <https://msdn.microsoft.com/en-us/virtualization/hyperv_on_windows/user_guide/remote_host_management>

[Pve] Proxmox Virtual Environment. <http://pve.proxmox.com/pve-docs/pve-admin-guide.html>

[Pwi] Proxmox Wiki (HOWTOs). <https://pve.proxmox.com/wiki/Main_Page>

[Sht] Spice html5 client. <https://www.spice-space.org/page/Html5>

[Spi] Spice Server & Clients. <https://www.spice-space.org/>

[Ufk] Virt-Manager Bridged Wireless Network. Ubuntu Forums. <https://ubuntuforums.org/showthread.php?t=1766674>

[Ves] VMware vSphere 6.5 Documentation Center. <https://pubs.vmware.com/vsphere-65/index.jsp#com.vmware.vsphere.doc/GUID-1B959D6B-41CA-4E23-A7DB-E9165D5A0E80.html>

[Vir] VirtualBox. <https://www.virtualbox.org/>

[VSO] Virtualización a nivel de sistema operativo. <https://es.wikipedia.org/wiki/Virtualización_a_nivel_de_sistema_operativo>

[Vwp] VMware Workstation Player. <http://www.vmware.com/products/workstation.html>

[Xag] Citrix Xen Server 7.0. Administration Guide. <http://docs.citrix.com/content/dam/docs/en-us/xenserver/xenserver-7-0/downloads/xenserver-7-0-administrators-guide.pdf>

[Wne] What's New in VMware vSphere®.6.5. <http://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/whitepaper/vsphere/vmw-white-paper-vsphr-whats-new-6-5.pdf>