

aleister888 dotfiles

Documentación e información útil

Índice

1. Estructura del repositorio	3
1.1. Entorno	3
1.2. Instalador	3
2. Compartir audio	4
3. Scripts útiles	4
3.1. comic-reduce	4
3.2. compressed-backup	4
3.3. exif-remove	4
3.4. flac-corruption-check	4
3.5. gallery-rename	5
3.6. music-convert	5
3.7. video-transform	5
3.8. wakeat	5
3.9. wakeme	5
4. Guías	6
4.1. VFIO GPU passthrough	6
4.1.1. Pasos Iniciales	6
4.1.2. Pre-configurar el gestor de arranque	6
4.1.3. Identificar el ID de nuestra gráfica y los grupos IOMMU	6
4.1.4. Módulos del kernel	7
4.1.5. Instalación del sistema operativo	8
4.1.6. Looking Glass en el Host	9
4.1.7. Añadir Gráfica y Looking Glass en el Guest	9
4.2. VirtioFS	10
4.2.1. Compartir más de una carpeta	11
4.3. Conexión bridge y RDP	11
4.3.1. Creación de nuestra red puente	11
4.3.2. Invitado con conexión puenteada	11
4.3.3. Configurar Escritorio remoto	12
4.4. SSH	12
4.5. Firewall	12
5. Aviso de Uso	13

1. Estructura del repositorio

La información contenida en este documento es *resumida* y tiene únicamente la intención de proporcionar una *visión general* del proyecto. Este documento *no se actualiza con la misma frecuencia que el repositorio*, por lo que es posible que *no refleje el estado actual* del mismo. Para información precisa y actualizada, se recomienda consultar directamente el repositorio.

1.1. Entorno

Tanto el instalador como los distintos scripts incluidos con el repositorio utilizan funciones contenidas en el archivo assets/shell/shell-utils-sh. Este provee funciones para facilitar el logging y la creación de directorios temporales. De modo que todos los scripts depositan sus logs en \$HOME/.cache/dotfiles_log/ y sus archivos temporales en /tmp/00_dotfiles/.

Además, los scripts que utilizan bash pueden importar shell-utils, que facilita el depurado guardando en los logs los distintos fallos de los scripts, especificando el número de linea en el que se dio el error.

1.2. Instalador

El instalador se divide en varias etapas, cada una implementada en un script que llama a la siguiente al completarse:

- `install.sh`: Se encarga de instalar las dependencias necesarias para el instalador y de procesar los argumentos de instalación.
- `stage1.sh`: Se ejecuta desde la ISO de Arch. Utiliza whiptail para preguntar por las opciones de configuración, formatea los discos, instala los paquetes básicos mediante basstrap, crea los usuarios y copia el repositorio al entorno chroot.
- `stage2.sh`: Se ejecuta dentro del chroot como root, configurando el sistema base.
- `stage3.sh`: Se ejecuta dentro del chroot como el usuario no privilegiado creado previamente. Durante su ejecución utiliza una configuración temporal de sudo sin contraseña para realizar tareas que requieren permisos administrativos. Configura el entorno de usuario, los servicios y otros ajustes, y al finalizar aplica una configuración de sudo más restrictiva.

Si `install.sh` se ejecuta con el flag `-d` el instalador se detendrá ante cualquier error.

2. Compartir audio

Tanto el instalador como los distintos scripts incluidos en el repositorio utilizan funciones definidas en assets/shell/shell-utils-sh. Este archivo proporciona utilidades para facilitar el *logging* y la creación de directorios temporales, de modo que todos los scripts registran sus logs en \$HOME/.cache/dotfiles_log/ y guardan archivos temporales en /tmp/00_dotfiles/.

Además, los scripts que utilizan bash pueden importar shell-utils, lo que permite un depurado más sencillo: los errores de los scripts se registran en los logs, indicando el número de línea en el que ocurrió cada fallo.

3. Scripts útiles

El repositorio incluye una variedad de scripts para automatizar tareas. Por defecto no se usan, pero podemos hacer que se ejecuten de forma automática con [crond](#).

3.1. comic-reduce

Parecido a music-convert, crea un mirror de una biblioteca de comics (archivos .cbz y .cbr) en un formato reducido y optimizado. Para esto reduce la calidad de las imágenes y las convierte en .jpg.

3.2. compressed-backup

Este script genera un archivo comprimido tar.gz que contiene una copia de seguridad del directorio especificado como primer argumento, y lo guarda en el directorio indicado como segundo argumento. Además, elimina las copias de seguridad antiguas, conservando únicamente aquellas cuya antigüedad no supere un número máximo de días.

Por defecto, este límite es de 7 días, aunque puede modificarse proporcionando dicho valor como tercer argumento al ejecutar el script.

3.3. exif-remove

Elimina todos los metadatos EXIF de las imágenes contenidas en el directorio especificado como primer argumento.

3.4. flac-corruption-check

Este script comprueba que no haya archivos .flac corruptos en nuestra biblioteca de música, corrige falsos positivos de corrupción y crea una lista con los archivos que no se pueden reproducir correctamente. Necesita como argumento el directorio cuyos archivos de audio queremos analizar.

3.5. gallery-rename

Renombra todas las imágenes del directorio especificado como primer argumento siguiendo el esquema: <año>-<mes>-<dia>_<hora>-<minuto>-<segundo>. *

3.6. music-convert

Este script convierte toda la música del directorio que damos como primer argumento a formato *.m4a*, *.mp3* o *.flac* en un mirror que replica al estructura del directorio original. Si ejecutamos:

```
music-convert -m4a /musica/biblioteca /musica/m4a
```

Toda la música de */musica/biblioteca* se convertira en *m4a* en la carpeta */musica/m4a*

- Si no se pone nada, el formato usado por defecto es mp3. También se puede especificar este comportamiento explícitamente con *-mp3*,
- El script también puede usarse con el flag *-l* lo que hará que además de convertir las canciones, se obtenga la letra de la canción (del archivo original, y si no está ahí, de internet) y se incluya dentro del archivo de audio.

3.7. video-transform

Recodifica el video proporcionado como primer argumento a h.264 con la opción de rotarlo. El archivo de salida se proporciona como segundo argumento y la rotación como el tercero.

3.8. wakeat

Comprueba si hay alguna máquina virtual en ejecución, y si no encuentra ninguna, suspende nuestro equipo y lo reanuda a la hora especificada como argumento (7 de la mañana del día siguiente si no se pasa ningún argumento). Útil para ahorrar energía y no tener que preocuparte por suspender tu equipo, ni de encenderlo por las mañanas.

El script te avisa de que el sistema se va suspender y pasados 10 minutos desde dicho aviso, suspende el sistema. *Si el script se ejecuta pasándole el argumento "now", entonces el sistema se suspenderá inmediatamente.*

3.9. wakeme

Este script funciona como un despertador, hace sonar el archivo de audio especificado hasta que le damos al un botón que apaga nuestra alarma.

4. Guías

4.1. VFIO GPU passthrough

Con VFIO (Virtual Function I/O) GPU passthrough podemos pasarle una tarjeta gráfica física a una máquina virtual. Lo que nos permite tener gráficos acelerados dentro de dicha máquina virtual.

4.1.1. Pasos Iniciales

- Primero activamos **VT-d** o **AMD-v** dependiendo de si tenemos un procesador Intel o AMD.
- Debemos de tener **IOMMU** activado. En la mayoría de placas bases activar VT-d o AMD-v, también activa IOMMU.
- Debemos tener desactivado **CSM (Compatibility Support Module)** en los ajustes de arranque de nuestra placa base.

4.1.2. Pre-configurar el gestor de arranque

El gestor de arranque **GNU GRUB**, es el programa que se encarga de cargar el kernel de nuestro sistema operativo.

Para usar VFIO necesitamos configurar el arranque del kernel. Las opciones de configuración globales de GRUB están en `/etc/default/grub`

Este archivo contiene las opciones de arranque del kernel en: `GRUB_CMDLINE_LINUX`.

```
GRUB_CMDLINE_LINUX_DEFAULT="loglevel=3 quiet"
GRUB_CMDLINE_LINUX=""
```

Tendremos que añadir las siguientes opciones:

- `iommu=pt`
- `amd_iommu=on` o `intel_iommu=on`, dependiendo de si tenemos una cpu Intel o AMD.
- `video=efifb:off`

```
GRUB_CMDLINE_LINUX="intel_iommu=on iommu=pt video=efifb:off"
GRUB_CMDLINE_LINUX="amd_iommu=on iommu=pt video=efifb:off"
```

Una vez editado el archivo, debemos actualizar nuestra configuración de GRUB con:

```
# grub-mkconfig -o /boot/grub/grub.cfg
```

4.1.3. Identificar el ID de nuestra gráfica y los grupos IOMMU

Después de esto ya podemos asignar el driver VFIO a nuestra tarjeta gráfica para poder usarla en nuestra máquina virtual.

Necesitamos conocer el identificador de nuestra gráfica, podemos obtener esta información ejecutando en BASH:

```
shopt -s nullglob
for g in /sys/kernel/iommu_groups/*; do
    echo "IOMMU Group ${g##*/}:"
    for d in $g/devices/*; do
        echo -e "\t$(lspci -nns ${d##*/})"
    done;
done;

IOMMU Group 15:
08:00.0 VGA compatible controller [0300]: NVIDIA TU116 [GeForce GTX 1660] [10de:21c4] (rev a1)
08:00.1 Audio device [0403]: NVIDIA TU116 High Definition Audio Controller [10de:1aeb] (rev a1)
08:00.2 USB controller [0c03]: NVIDIA Device [10de:1aec] (rev a1)
08:00.3 Serial bus controller [0c80]: NVIDIA TU116 [GeForce GTX 1650] [10de:1aed] (rev a1)
```

Debemos añadir cada uno de los IDs de los dispositivos que se encuentran en el mismo grupo que nuestra gráfica (en nuestro ejemplo: 10de:21c4, 10de:1aeb, 10de:1aec y 10de:1aed) a nuestro archivo de configuración de GRUB, diciéndole que le asigne a estos dispositivos el driver vfio.

```
GRUB_CMDLINE_LINUX_DEFAULT="loglevel=3 quiet"
GRUB_CMDLINE_LINUX="amd_iommu=on iommu=pt video=efifb:off
vfio-pci.ids=10de:21c4,10de:1aeb,10de:1aec,10de:1aed"
```

Ejecutamos de nuevo:

```
# grub-mkconfig -o /boot/grub/grub.cfg
```

4.1.4. Módulos del kernel

Ahora cada vez que iniciemos nuestro ordenador el kernel intentará asignarle a esos dispositivos PCI el driver VFIO, pero el driver VFIO por defecto no se carga durante el arranque del sistema.

Debemos de configurar nuestro kernel para que nuestra imagen de arranque incluya el driver VFIO, para poder asignarselo a nuestra gráfica.

Para esto debemos editar /etc/mkinitcpio.conf y añadir los módulos del kernel que vamos a usar.

```
MODULES=( vfio_pci vfio vfio_iommu_type1 )
```

Una vez editado nuestro archivo debemos regenerar el **initramfs** con:

```
# mkinitcpio -P
```

Para comprobar que hemos seguido los pasos correctamente reiniciamos nuestro ordenador y cuando arranque de nuevo, ejecutamos:

```
# dmesg | grep -i vfio
```

Si tenemos una salida parecida a:

```
[3.416692] vfio_pci: add [10de:21c4[ffffffff:ffffffff]] class 0x000000/00000000
[3.433353] vfio_pci: add [10de:1aeb[ffffffff:ffffffff]] class 0x000000/00000000
[3.450019] vfio_pci: add [10de:1aec[ffffffff:ffffffff]] class 0x000000/00000000
[3.466953] vfio_pci: add [10de:1aed[ffffffff:ffffffff]] class 0x000000/00000000
```

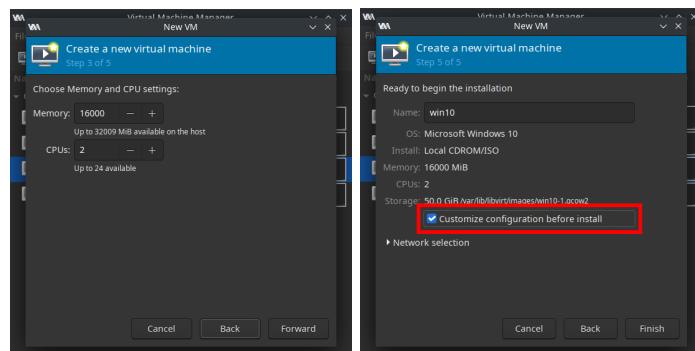
donde podemos ver que el driver vfio_pci se ha cargado correctamente para nuestros dispositivos, entonces hemos realizado correctamente todos los pasos.

4.1.5. Instalación del sistema operativo

Ya tenemos lista nuestra gráfica para ser usada por nuestra máquina virtual, queda instalar nuestra máquina virtual y configurarla para usarla cómodamente.

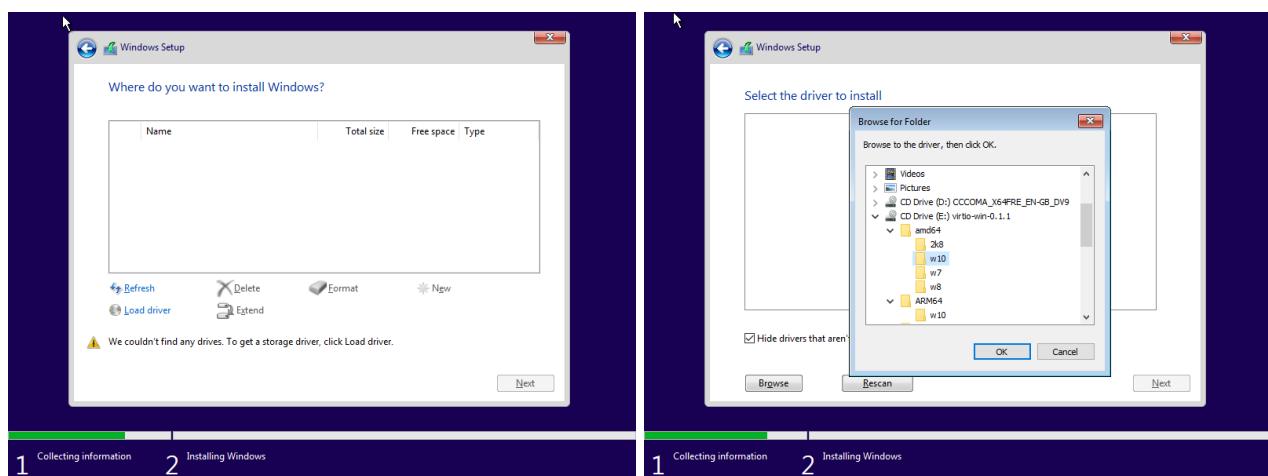
Descargamos la ISO de Windows 10 desde <https://www.microsoft.com>, y los drivers que necesitará Windows desde <https://fedorapeople.org>. Una vez descargadas las ISO, abrimos *virt-manager* y creamos una máquina virtual:

Asignamos la memoria RAM (8GB como mínimo) a nuestra máquina virtual. Y en el último paso, elegimos *customizar nuestra máquina virtual* antes de la instalación.

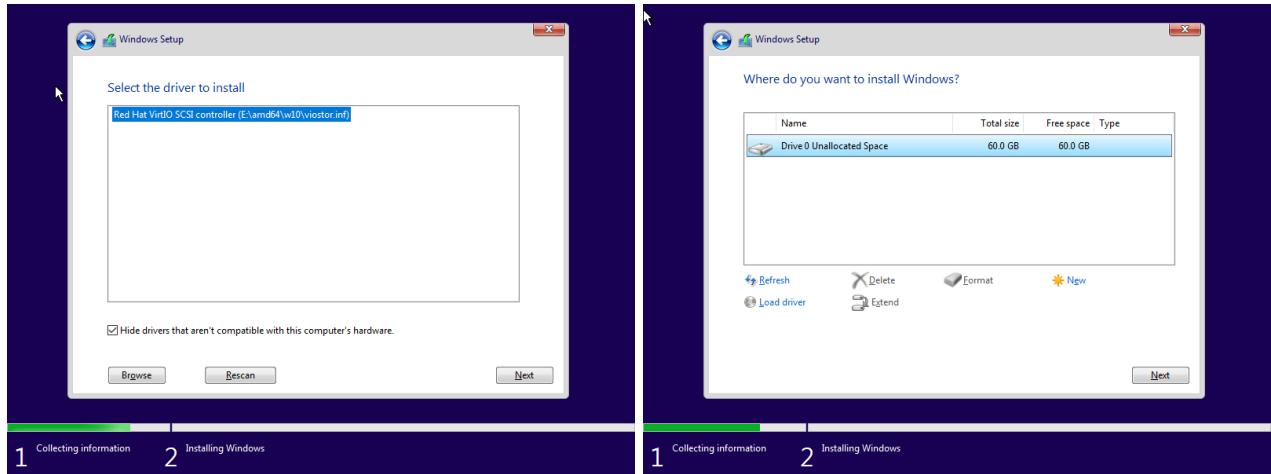


Ahora deberás hacer las siguientes modificaciones:

- Cambiar el chipset a Q35 y elegir el firmware UEFI:x86_64:/usr/share/edk2/x64/OVMF_CODE.fd
- Ajustar la *topología* del procesador de la siguiente manera:
 - 1 *socket*, tantos *centros* como núcleos tenga tu procesador y tantos *hilos* como hilos tenga tu procesador por núcleo (ej. para un procesador de 8 núcleos, 16 hilos, elegiríamos: 1 *Socket*, 8 *Centros*, 2 *Hilos*).
- Cambiar el bus de nuestro disco virtual de SATA a VirtIO.
- Cambiar el modelo del NIC a virtio
- Añadir el ISO con los drivers virtio.



Durante la instalación debemos cargar nuestro driver *virtio* para que nuestro disco virtual sea detectado. *Cuando la instalación de windows finalize, apaga la máquina virtual.*



4.1.6. Looking Glass en el Host

Para usar looking glass sin tener dos monitores, necesitamos un [display dummy](#).

Looking Glass nos muestra una el output de nuestra gráfica con una latencia muy baja. Logra esto mandando la información a través de un archivo compartido entre máquina virtual y host.

Necesitamos configurar *tmpfilesd* para que cada vez que iniciamos el sistema, se cree el archivo que compartido que Looking Glass necesita. Para eso ejecutamos los siguientes comandos:

```
echo "f /dev/shm/looking-glass 0660 $USER kvm -" | \
sudo tee -a /etc/tmpfiles.d/looking-glass.conf
```

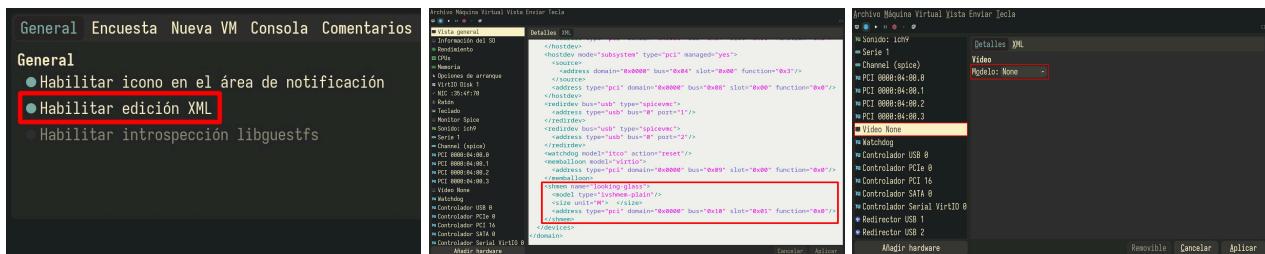
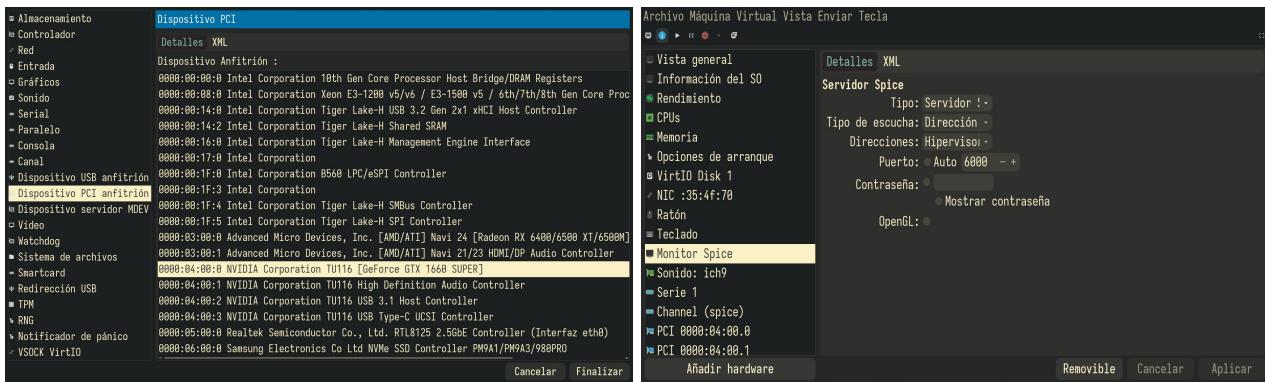
4.1.7. Añadir Gráfica y Looking Glass en el Guest

Ya tenemos casi todo listo, nos queda instalar los drivers de vídeo y Looking Glass en nuestra máquina virtual.

Tenemos que añadir nuestra tarjeta gráfica a la máquina virtual. En “Añadir Dispositivo”, buscamos la pestaña para añadir dispositivos PCI y añadimos la tarjeta gráfica. Después debemos de configurar el servidor Spice para que use el puerto 6000 en vez de asignar un puerto automáticamente.

Ahora tenemos que activar la edición de XML en Virt-Manager (*Editar, Preferencias*) y editar el código XML de nuestra máquina virtual añadiendo estas líneas justo después de las sección </memballoon>:

```
<shmem name='looking-glass'>
  <model type='ivshmem-plain' />
  <size unit='M'>32</size>
</shmem>
```



Iniciamos nuestra máquina virtual e instalamos nuestros drivers de [NVIDIA](#) o [AMD](#) y [Looking Glass Host](#).

Finalmente cambiamos el driver de vídeo de QXL a *None* y borramos el dispositivo de tableta táctil. La próxima vez que iniciemos la máquina virtual desde Virt-Manager podremos usarla através de Looking Glass con gráficos acelerados por hardware. Para que esto funcione debemos reiniciar nuestro ordenador o ejecutar el comando.

```
sudo install -g kvm -o $(whoami) -m 0660 /dev/null /dev/shm/looking-glass
```

Nota

Para más información sobre como configurar un VFIO passthought puede consultar:

- <https://qqq.ninja/blog/post/vfio-on-arch/>
- <https://gitlab.com/risingprismtv/single-gpu-passthrough/-/wikis/home>
- https://wiki.archlinux.org/title/PCI_passthrough_via_OVMF

4.2. VirtioFS

Podemos usar virtiofs para compartir directorios del host con nuestra máquina virtual. Para ello, añadimos los directorios deseados desde Virt-Manager (Añadir hardware, Sistema de archivos) eligiendo la *ruta de origen* (la carpeta que queremos compartir) y la *ruta objetivo* (el nombre con el que aparecerá en nuestra máquina invitada).

Necesitamos también los controladores VirtIO y el software de integración con el host. Podemos instalarlos desde la ISO que descargamos: [virtio-win.iso](#). Esta contiene los instaladores necesarios (*virtio-win-gt-x64.msi* y *virtio-win-guest-tools.exe*)

Después de instalar los controladores, debemos instalar [WinFSP](#) para poder montar sistemas de archivos VirtioFS. Podemos activar el servicio *virtiofs* para que windows detecte nuestra carpeta compartida.

4.2.1. Compartir más de una carpeta

Para compartir múltiples carpetas con nuestra máquina virtual, en lugar de utilizar el servicio *virtiofs*, es necesario crear un script que monte las carpetas deseadas directamente en la máquina virtual.

Para ello, primero ejecutamos desde cmd como Administrador:

```
"C:\Program Files (x86)\WinFsp\bin\fsreg.bat" virtiofs "C:\Program Files\Virtio-Win\VioFS\virtiofs.exe" "-t %1 -m %2"
```

Una vez que tenemos todo instalado, creamos un script ‘.bat’ que monte nuestras carpetas en la máquina invitada. Si tenemos dos carpetas cuyas rutas objetivo son “documentos” y “videos” y queremos montarlas en “Y:” y “Z:”, respectivamente, podemos crear un script para montar ambas carpetas:

```
"C:\Program Files (x86)\WinFsp\bin\launchctl-x64.exe" start virtiofs documentos documentos Y:  
"C:\Program Files (x86)\WinFsp\bin\launchctl-x64.exe" start virtiofs videos videos Z:
```

4.3. Conexión bridge y RDP

Adicionalmente, podemos configurar nuestra máquina virtual para, a efectos prácticos, comportarse como un ordenador distinto en la red. En vez de usar un NAT y que nuestro host se encargue del re-direccionamiento al guest, podemos usar una conexión puente y hacer que nuestro router reconozca nuestra máquina virtual como un dispositivo distinto y le asigne su propia IP. De esta forma podemos abrir servicios desde nuestro guest y poder acceder a ellos.

4.3.1. Creación de nuestra red puente

Para crear nuestra conexión de puente, utilizaremos *NetworkManager*:

- Ejecutamos *nmtui*, seleccionamos *Modificar una conexión* y elegimos añadir una nueva conexión (*de tipo Puente*).
- Debemos elegir que tarjeta de red queremos *añadir* para la conexión puenteada. Elegiremos *Ethernet*.
- Debemos eliminar la conexión *Ethernet* original de la lista de conexiones en (el host seguirá teniendo acceso a Internet; tanto la máquina anfitriona como la máquina invitada usarán la conexión de puente para acceder a Internet).

4.3.2. Invitado con conexión puenteada

Ahora tenemos que decirle a *virt-manager* que queremos usar nuestra conexión puenteada para la máquina virtual, podemos incluso borrar la red NAT ‘*default*’ si no tenemos ninguna otra máquina virtual que queramos conectar a Internet.

Para usar la conexión puenteada en nuestra máquina virtual; en las propiedades de nuestra máquina virtual, cambiamos la ‘*Fuente de red*’ de nuestro NIC a la conexión puenteada.

Adicionalmente, podemos ajustar la dirección MAC y establecer una IP estática asociada a esa dirección MAC en nuestro router. Lo cual facilitará configurar servicios y acceder a nuestra máquina invitada virutal la red.

4.3.3. Configurar Escritorio remoto

Ahora podemos configurar nuestra máquina invitada para acceder remotamente a ella mediante RDP, para ello en Windows (*Sólo si tenemos Windows 10/11 Pro*) en Ajustes/Escritorio Remoto, activaremos el escritorio remoto.

Para conectaros a la máquina virtual de nuestra red debemos configurar el Firewall de nuestro guest para permitir las conexiones entrantes de RDP, para ello ejecutaremos este comando en cmd como Administrador: `netsh advfirewall firewall add rule name="RDP" protocol=TCP dir=in localport=3389 action=allow`

Podemos conectarnos a nuestra máquina usando el script `rdp-connect`

4.4. SSH

Para conectarnos a nuestro equipo remotamente debemos configurar [OpenSSH](#). Podemos configurar SSH de forma muy básica con el script:

```
ssh-configure
```

Mi recomendación es no conformarse con esta configuración básica. Desactiva el login con el usuario root, desactiva el login por contraseña y usa claves públicas para conectarte.

4.5. Firewall

Para instalar el firewall ejecuta:

```
sudo sh -c 'pacman -Sy ufw; systemctl enable ufw; ufw enable'
```

Podemos añadir unas reglas muy básicas a nuestro firewall (*que se aplicarán cuando reiniciemos nuestro equipo*) con:

```
ufw allow 80/tcp
ufw allow 443/tcp
ufw allow 3020      # Redmine
ufw allow in on virbr0 # Libvirt (NAT)
ufw allow out on virbr0 # Libvirt (NAT)
ufw limit ssh
ufw default deny incoming
ufw default allow outgoing
```

Nota

Si configuraste ssh con el script `bin/utils/ssh-configure` el firewall ya se configuró automáticamente

5. Aviso de Uso

Este proyecto es para uso personal. Se comparte con la intención de que pueda ser útil para otros, pero se proporciona tal cual, sin ninguna garantía. No se asume responsabilidad por ninguna pérdida de datos o problemas que puedan surgir del uso de este proyecto.