VirtualBox/Tips and tricks

< VirtualBox

See **VirtualBox** for the main article.

Contents

Import/export VirtualBox virtual machines from/to other hypervisors

If you plan to use your virtual machine on another hypervisor or want to import in VirtualBox a virtual machine created with another hypervisor, you might be interested in reading the following steps.

Remove additions

Guest additions are available in most hypervisor solutions: VirtualBox comes with the Guest Additions, VMware with the VMware Tools, Parallels with the Parallels Tools, etc. These additional components are designed to be installed inside a virtual machine after the guest operating system has been installed. They consist of device drivers and system applications that optimize the guest operating system for better performance and usability **by providing these features (https://www.virtualbox.org/manual/ch04.html)**.

If you have installed the additions to your virtual machine, please uninstall them first. Your guest, especially if it is using an OS from the Windows family, might behave weirdly, crash or even might not boot at all if you are still using the specific drivers in another hypervisor.

Use the right virtual disk format

This step will depend on the ability to convert the virtual disk image directly or not.

Automatic tools

Some companies provide tools which offer the ability to create virtual machines from a Windows or GNU/Linux operating system located either in a virtual machine or even in a native installation. With such a product, you do not need to apply this and the following steps and can stop reading here.

- *Parallels Transporter (http://www.parallels.com/products/transporter)* which is non free, is a product from Parallels Inc. This solution basically consists in an piece of software called *agent* that will be installed in the guest you want to import/convert. Then, Parallels Transporter, which only works on OS X, will create a virtual machine from that *agent* which is contacted either by USB or Ethernet network.
- *VMware vCenter Converter (https://www.vmware.com/products/converter/)* which is free upon registration on the VMware webiste, works nearly the same way as Parallels Transporter, but the piece of software that will gather the data to create the virtual machine only works on a Windows platform.

Manual conversion

First, familiarize yourself with the **formats supported by VirtualBox** and **those supported by third-party hypervisors**.

- Importing or exporting a virtual machine from/to a VMware solution is not a problem at all if you use the VMDK or OVF disk format, otherwise converting **VMDK to VDI and VDI to VMDK**[**broken link**: invalid section] is possible and the aforementioned VMware vCenter Converter tool is available.

- Importing or exporting from/to QEMU is not a problem neither: some QEMU formats are supported directly by VirtualBox and conversion between **QCOW2 to VDI and VDI to QCOW2**[**broken link**: invalid section] is still available if needed.

- Importing or exporting from/to Parallels hypervisor is the hardest way: Parallels does only support its own HDD format (even the standard and portable OVF format is not supported!).

  - To export your virtual machine to Parallels, you will need to use the Parallels Transporter tool described above.
  - To import your virtual machine to VirtualBox, you will need to use the VMware vCenter Converter described above to convert the VM to the VMware format first. Then, apply the solution to migrate from VMware.

Create the VM configuration for your hypervisor

Each hypervisor have their own virtual machine configuration file: `.vbox` for VirtualBox, `.vmx` for VMware, a `config.pvs` file located in the virtual machine bundle (`.pvm` file), etc. You will have thus to recreate a new virtual machine in your new destination hypervisor and specify its hardware configuration as close as possible as your initial virtual machine.

Pay a close attention to the firmware interface (BIOS or UEFI) used to install the guest operating system. While an option is available to choose between these 2 interfaces on VirtualBox and on Parallels solutions, on VMware, you will have to add manually the following line to your *.vmx* file.

```
ArchLinux_vm.vmx
```

```
firmware = "efi"
```

Finally, ask your hypervisor to use the existing virtual disk you have converted and launch the virtual machine.

Tip:

- On VirtualBox, if you do not want to browse the whole GUI to find the right location to add your new virtual drive device, you can **#Replace a virtual disk manually from the .vbox file**[**broken link**: invalid section], or use the `VBoxManage storageattach` described in **#Increase virtual disks**[**broken link**: invalid section] or in the **VirtualBox manual page (https://www.virtualbox.org/manual/ch08.html#vboxmanage-storageattach)**.
- Similarly, in VMware products, you can replace the location of the current virtual disk location by adapting the *.vmdk* file location in your *.vmx* configuration file.

Virtual machine launch management

Starting virtual machines with a service (autostart)

Find hereafter the implementation details of a systemd service that will be used to consider a virtual machine as a service.

```
/etc/systemd/system/vboxvmservice@.service
--------------------------------------------------------------------------------
[Unit]
Description=VBox Virtual Machine %i Service
Requires=systemd-modules-load.service
After=systemd-modules-load.service

[Service]
User=username
Group=vboxusers
ExecStart=/usr/bin/VBoxManage startvm %i --type startmode
ExecStop=/usr/bin/VBoxManage controlvm %i stopmode
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

Note:

- Replace `username` with a user that is a member of the `vboxusers` group. Make sure the user chosen is the same user that will create/import virtual machines, otherwise the user will not see the VM appliances.
- Replace `startmode` with a VM frontend type, usually `gui` , `headless` or `separate`

- Replace `stopmode` with desired state switch, usually `savestate` or `acpipowerbutton`

Note: If you have multiple virtual machines managed by Systemd and they are not stopping properly, try to add `KillMode=none` and `TimeoutStopSec=40` at the end of `[Service]` section.

Enable the `vboxvmservice@your_virtual_machine_name` systemd unit in order to launch the virtual machine at next boot. To launch it directly, simply **start** the systemd unit.

VirtualBox 4.2 introduces **a new way (http://lifeofageekadmin.com/how-to-set-your-virtualbox-vm-to-automatically-startup/)** for UNIX-like systems to have virtual machines started automatically, other than using a systemd service.

Starting virtual machines with a keyboard shortcut

It can be useful to start virtual machines directly with a keyboard shortcut instead of using the VirtualBox interface (GUI or CLI). For that, you can simply define key bindings in `.xbindkeysrc`. Please refer to **Xbindkeys** for more details.

Example, using the `Fn` key of a laptop with an unused battery key (`F3` on the computer used in this example):

```
"VBoxManage startvm 'Windows 7'"
m:0x0 + c:244
XF86Battery
```

Note: If you have a space in the name of your virtual machine, then enclose it with single apostrophes like made in the example just above.

Use specific device in the virtual machine

Using USB webcam / microphone

Note: You will need to have VirtualBox extension pack installed before following the steps below. See **#Extension pack**[**broken link**: invalid section] for details.

1. Make sure the virtual machine is not running and your webcam / microphone is not being used.
2. Bring up the main VirtualBox window and go to settings for Arch machine. Go to USB section.
3. Make sure "Enable USB Controller" is selected. Also make sure that "Enable USB 2.0 (EHCI) Controller" is selected too.
4. Click the "Add filter from device" button (the cable with the '+' icon).
5. Select your USB webcam/microphone device from the list.
6. Now click OK and start your VM.

Note: If your Microphone does not show up in the "Add filter from device" menu, try the USB 3.0 and 1.1 options instead (In Step 3).

Detecting web-cams and other USB devices

Note: This will not do much if you are running a *NIX OS inside of your VM, as most do not have autodetection features.

If the device that you are looking for does not show up on any of the menus in the section above and you have tried all three USB controller options, boot up your VM three seperate times. Once using the USB 1.1 controller, another using the USB 2.0 controller, etc. Leave the VM running for at least 5 minutes after startup. Sometimes Windows will autodetect the device for you. Be sure you filter any devices that are not a keyboard or a mouse so they do not start up at boot. This ensures that Windows will detect the device at start-up.

Access a guest server

To access Apache server on a Virtual Machine from the host machine **only**, simply execute the following lines on the host:

```
$ VBoxManage setextradata GuestName "VBoxInternal/Devices/pcnet/0/LUN#0/Config/Apache/HostPort" 8888
$ VBoxManage setextradata GuestName "VBoxInternal/Devices/pcnet/0/LUN#0/Config/Apache/GuestPort" 80
$ VBoxManage setextradata GuestName "VBoxInternal/Devices/pcnet/0/LUN#0/Config/Apache/Protocol" TCP
```

Where 8888 is the port the host should listen on and 80 is the port the VM will send Apache's signal on.

To use a port lower than 1024 on the host machine, changes need to be made to the firewall on that host machine. This can also be set up to work with SSH or any other services by changing "Apache" to the corresponding service and ports.

Note: `pcnet` refers to the network card of the VM. If you use an Intel card in your VM settings, change `pcnet` to `e1000`.

To communicate between the VirtualBox guest and host using ssh, the server port must be forwarded under Settings > Network. When connecting from the client/host, connect to the IP address of the client/host machine, as opposed to the connection of the other machine. This is because the connection will be made over a virtual adapter.

D3D acceleration in Windows guests

Recent versions of Virtualbox have support for accelerating OpenGL inside guests. This can be enabled with a simple checkbox in the machine's settings, right below where video ram is set, and installing the Virtualbox guest additions. However, most Windows games use Direct3D (part of DirectX), not OpenGL, and are thus not helped by this method. However, it is possible to gain accelerated Direct3D in your Windows guests by borrowing the d3d libraries from Wine, which translate d3d calls into OpenGL, which is then accelerated. These libraries are now part of Virtualbox guest additions software.

After enabling OpenGL acceleration as described above, reboot the guest into safe mode (press F8 before the Windows screen appears but after the Virtualbox screen disappears), and install Virtualbox guest additions, during install enable checkbox "Direct3D support". Reboot back to normal mode and you should have accelerated Direct3D.

Note: This hack may or may not work for some games depending on what hardware checks they make and what parts of D3D they use.

Note: This was tested on Windows XP, 7 and 8.1. If method does not work on your Windows version please add data here.

VirtualBox on a USB key

When using VirtualBox on a USB key, for example to start an installed machine with an ISO image, you will manually have to create VDMKs from the existing drives. However, once the new VMDKs are saved and you move on to another machine, you may experience problems launching an appropriate machine again. To get rid of this issue, you can use the following script to launch VirtualBox. This script will clean up and unregister old VMDK files and it will create new, proper VMDKs for you:

```
#!/bin/bash

# Erase old VMDK entries
rm ~/.VirtualBox/*.vmdk

# Clean up VBox-Registry
sed -i '/sd/d' ~/.VirtualBox/VirtualBox.xml

# Remove old harddisks from existing machines
find ~/.VirtualBox/Machines -name \*.xml | while read file; do
  line=`grep -e "type\=\"HardDisk\"" -n $file | cut -d ':' -f 1`
  if [ -n "$line" ]; then
    sed -i ${line}d $file
    sed -i ${line}d $file
    sed -i ${line}d $file
  fi
  sed -i "/rg/d" $file
done

# Delete prev-files created by VirtualBox
find  ~/.VirtualBox/Machines -name \*-prev -exec rm '{}' \;

# Recreate VMDKs
ls -l /dev/disk/by-uuid | cut -d ' ' -f 9,11 | while read ln; do
  if [ -n "$ln" ]; then
    uuid=`echo "$ln" | cut -d ' ' -f 1`
    device=`echo "$ln" | cut -d ' ' -f 2 | cut -d '/' -f 3 | cut -b 1-3`

    # determine whether drive is mounted already
    checkstr1=`mount | grep $uuid`
    checkstr2=`mount | grep $device`
    checkstr3=`ls ~/.VirtualBox/*.vmdk | grep $device`
    if [[ -z "$checkstr1" && -z "$checkstr2" && -z "$checkstr3" ]]; then
      VBoxManage internalcommands createrawvmdk -filename ~/.VirtualBox/$device.vmdk -rawdisk /dev/$device -register
    fi
  fi
done

# Start VirtualBox
VirtualBox
```

Note that your user has to be added to the "disk" group to create VMDKs out of existing drives.

Run a native Arch Linux installation inside VirtualBox

If you have a dual boot system between Arch Linux and another operating system, it can become rapidly tedious to switch back and forth if you need to work in both. Also, by using virtual machines, you just have a tiny fragment of your computer power, which can cause issues when working on projects requiring performance.

This guide will let you reuse, in a virtual machine, your native Arch Linux installation when you are running your second operating system. This way, you keep the ability to run each operating system natively, but have the option to run your Arch Linux installation inside a virtual machine.

Make sure you have a persistent naming scheme

Depending on your hard drive setup, device files representing your hard drives may appear differently when you will run your Arch Linux installation natively or in virtual machine. This problem occurs when using **FakeRAID** for example. The fake RAID device will be mapped in `/dev/mapper/` when you run your GNU/Linux distribution natively, while the devices are still accessible separately. However, in your virtual machine, it can appear without any mapping in `/dev/sdaX` for example, because the drivers controlling the fake RAID in your host operating system (e.g. Windows) are abstracting the fake RAID device.

To circumvent this problem, we will need to use an addressing scheme that is persistent to both systems. This can be achieved using **UUIDs**. Make sure your **boot loader** and **fstab** file is using UUIDs, otherwise fix this issue. Read **fstab** and **Persistent block device naming**.

Warning:

- Make sure your host partition is only accessible in read only from your Arch Linux virtual machine, this will avoid risk of corruptions if you were to corrupt that host partition by writing on it due to lack of attention.
- You should NEVER allow VirtualBox to boot from the entry of your second operating system, which, as a reminder, is used as the host for this virtual machine! Take thus a special care especially if your default boot loader/boot manager entry is your other operating system. Give a more important timeout or put it below in the order of preferences.

Make sure your mkinitcpio image is correct

Make sure your **mkinitcpio** configuration uses the **HOOK** `block`:

```
/etc/mkinitcpio.conf
```

```
[...]
HOOKS="base udev autodetect modconf block filesystems keyboard fsck"
[...]
```

If it is not present, add it and **regenerate your initramfs**:

```
# mkinitcpio -p linux
```

Create a VM configuration to boot from the physical drive

Create a raw disk .vmdk image

Now, we need to create a new virtual machine which will use a **RAW disk (https://www.virtualbox.org/manual/ch09.html#rawdisk)** as virtual drive, for that we will use a ~ 1Kio VMDK file which will be mapped to a physical disk. Unfortunately, VirtualBox does not have this option in the GUI, so we will have to use the console and use an internal command of `VBoxManage`.

Boot the host which will use the Arch Linux virtual machine. The command will need to be adapted according to the host you have.

On a GNU/Linux host

There is 3 ways to achieve this: login as root, changing the access right of the device with `chmod`, adding your user to the `disk` group. The latter way is the more elegant, let us proceed that way:

```
# gpasswd -a your_user disk
```

Apply the new group settings with:

```
$ newgrp
```

Now, you can use the command:

```
$ VBoxManage internalcommands createrawvmdk -filename /path/to/file.vmdk -rawdisk /dev/sdb -register
```

Adapt the above command to your need, especially the path and filename of the VMDK location and the raw disk location to map which contain your Arch Linux installation.

On a Windows host

Open a command prompt must be run as administrator.

Tip: On Windows, open your start menu/start screen, type `cmd`, and type `Ctrl+Shift+Enter`, this is a shortcut to execute the selected program with admin rights.

On Windows, as the disk filename convention is different from UNIX, use this command to determine what drives you have in your Windows system and their location:

```
# wmic diskdrive get name,size,model
```

```
Model                              Name                Size
WDC WD40EZRX-00SPEB0 ATA Device    \\.\PHYSICALDRIVE1  4000783933440
KINGSTON SVP100S296G ATA Device    \\.\PHYSICALDRIVE0  96024821760
Hitachi HDT721010SLA360 ATA Device \\.\PHYSICALDRIVE2  1000202273280
Innostor Ext. HDD USB Device       \\.\PHYSICALDRIVE3  1000202273280
```

In this example, as the Windows convention is `\\.\PhysicalDriveX` where X is a number from 0, `\\.\PHYSICALDRIVE1` could be analogous to `/dev/sdb` from the Linux disk terminology.

To use the `VBoxManage` command on Windows, you can either, change the current directory to your VirtualBox installation folder first with `cd C:\Program Files\Oracle\VirtualBox\`

```
# .\VBoxManage.exe internalcommands createrawvmdk -filename C:\file.vmdk -rawdisk \\.\PHYSICALDRIVE1
```

or use the absolute path name:

```
# "C:\Program Files\Oracle\VirtualBox\VBoxManage.exe" internalcommands createrawvmdk -filename C:\file.vmdk -rawdisk \\.\PHYSICALDRIVE1
```

On another OS host

There are other limitations regarding the aforementioned command when used in other operating systems like OS X, please thus **read carefully the manual page (https://www.virtualbox.org/manual/ch09.html#rawdisk)**, if you are concerned.

Create the VM configuration file

Note:

- To make use of the VBoxManage command on Windows, you need to change the current directory to your VirtualBox installation folder first: cd C:\Program Files\Oracle\VirtualBox\.
- Windows makes use of backslashes instead of slashes, please replace all slashes / occurrences by backslashes \ in the commands that follow when you will use them.

After, we need to create a new machine (replace the *VM_name* to your convenience) and register it with VirtualBox.

```
$ VBoxManage createvm -name VM_name -register
```

Then, the newly raw disk needs to be attached to the machine. This will depend if your computer or actually the root of your native Arch Linux installation is on an IDE or a SATA controller.

If you need an IDE controller:

```
$ VBoxManage storagectl VM_name --name "IDE Controller" --add ide
$ VBoxManage storageattach VM_name --storagectl "IDE Controller" --port 0 --device 0 --type hdd --medium /path/to/file.vmdk
```

otherwise:

```
$ VBoxManage storagectl VM_name --name "SATA Controller" --add sata
$ VBoxManage storageattach VM_name --storagectl "SATA Controller" --port 0 --device 0 --type hdd --medium /path/to/file.vmdk
```

While you continue using the CLI, it is recommended to use the VirtualBox GUI, to personalise the virtual machine configuration. Indeed, you must specify its hardware configuration as close as possible as your native machine: turning on the 3D acceleration, increasing video memory, setting the network interface, etc.

Finally, you may want to seamlessly integrate your Arch Linux with your host operating system and allow copy pasting between both OSes. Please refer to **#Install the Guest Additions**[**broken link**: invalid section] for that, since this Arch Linux virtual machine is basically an Arch Linux guest.

Warning: For **Xorg** to work in natively and in the virtual machine, since obviously it will be using different drivers, it is best if there is no `/etc/X11/xorg.conf`, so Xorg will pick up everything it needs on the fly. However, if you really do need your own Xorg configuration, maybe is it worth to set your default systemd target to `multi-user.target` with `systemctl isolate graphical.target` as root (more details at **Systemd#Targets table**[**broken link**: invalid section] and **Systemd#Change current target**). In that way, the graphical interface is disabled (i.e. Xorg is not launched) and after you logged in, you can `startx`} manually with a custom `xorg.conf`.

Install a native Arch Linux system from VirtualBox

In some cases it may be useful to install a native Arch Linux system while running another operating system: one way to accomplish this is to perform the installation through VirtualBox on a **raw disk (http://www.virtualbox.org/manual/ch09.html#rawdisk)**. If the existing operating system is Linux based, you may want to consider following **Install from existing Linux** instead.

This scenario is very similar to **#Run a native Arch Linux installation inside VirtualBox**, but will follow those steps in a different order: start by **#Create a raw disk .vmdk image**, then **#Create the VM configuration file**.

Now, you should have a working VM configuration whose virtual VMDK disk is tied to a real disk. The installation process is exactly the same as the steps described in **#Installation steps for Arch Linux guests**[**broken link**: invalid section], but **#Make sure you have a persistent naming scheme** and **#Make sure your mkinitcpio image is correct**.

Warning:

- For BIOS systems and MBR disks, do not install a bootloader inside your virtual machine, this will not work since the MBR is not linked to the MBR of your real machine and your virtual disk is only mapped to a real partition without the MBR.
- For UEFI systems without **CSM** and GPT disks, the installation will not work at all since:

  - the **ESP** partition is not mapped to your virtual disk and Arch Linux requires to have the Linux kernel on it to boot as an EFI application (see **EFISTUB** for details);
  - and the efivars, if you are installing Arch Linux using the EFI mode brought by VirtualBox, are not the one of your real system: the bootmanager entries will hence not be registered.

- This is why, it is recommended to create your partitions in a native installation first, otherwize the partitions will not be taken into consideration in your MBR/GPT partition table.

After completing the installation, boot your computer natively with an GNU/Linux installation media (whether it be Arch Linux or not), **chroot**[**broken link**: invalid section] into your installed Arch Linux installation and **#Install and configure a bootloader**[**broken link**: invalid section].

Move a native Windows installation to a virtual machine

If you want to migrate an existing native Windows installation to a virtual machine which will be used with VirtualBox on GNU/Linux, this use case is for you. This section only covers native Windows installation using the MSDOS/Intel partition scheme. Your Windows installation must reside on the first MBR partition for this operation to success. Operation for other partitions are available but have been untested (see **#Known limitations** for details).

Warning: If you are using an OEM version of Windows, this process is unauthorized by the end user license license. Indeed, the OEM license typically states the Windows install is tied with the hardware together. Transferring a Windows install to a virtual machine removes this link. Make thus sure you have a full Windows install or a volume license model before continuing. If you have a full Windows license but the latter is not coming in volume, nor as a special license for several PCs, this means you will have to remove the native installation after the transfer operation has been achieved.

A couple of tasks are required to be done inside your native Windows installation first, then on your GNU/Linux host.

Tasks on Windows

The first three following points comes from **this outdated VirtualBox wiki page (https://www.virtualbox.org/wiki/Migrate_Windows#HAL)**, but are updated here.

- Remove IDE/ATA controllers checks (Windows XP only): Windows memorize the IDE/ATA drive controllers it has been installed on and will not boot if it detects these have changed. The solution proposed by Microsoft is to reuse the same controller or use one of the same serial, which is impossible to achieve since we are using a Virtual Machine. **MergeIDE (https://www.virtualbox.org/wiki/Migrate_Windows#HardDiskSupport)**, a German tool, developped upon another other solution proposed by Microsoft can be used. That solution basically consists in taking all IDE/ATA controller drivers supported by Windows XP from the initial driver archive (the location is hard coded, or specify it as the first argument to the `.bat` script), installing them and registering them with the regedit database.

- Use the right type of Hardware Abstraction Layer (old 32 bits Windows versions): Microsoft ships 3 default versions: `Hal.dll` (Standard PC), `Halacpi.dll` (ACPI HAL) and `Halaacpi.dll` (ACPI HAL with IO APIC). Your Windows install could come installed with the first or the second version. In that way, please **disable the *Enable IO/APIC* VirtualBox extended feature (https://www.virtualbox.org/manual/ch08.html#idp56927888)**.

- Disable any AGP device driver (only outdated Windows versions): If you have the files `agp440.sys` or `intelppm.sys` inside the `C:\Windows\SYSTEM32\drivers\` directory, remove it. As VirtualBox uses a PCI virtual graphic card, this can cause problems when this AGP driver is used.

- Create a Windows recovery disk: In the following steps, if things turn bad, you will need to repair your Windows installation. Make sure you have an install media at hand, or create one with *Create a recovery disk* from Vista SP1, *Create a system repair disc* on Windows 7 or *Create a recovery drive* on Windows 8.x).

Using Disk2vhd to clone Windows partition

Boot into Windows, clean up the installation (with CCleaner (http://www.piriform.com/ccleaner) for example), use disk2vhd (https://technet.microsoft.com/en-us/library/ee656415.aspx) tool to create a VHD image. Include a reserved system partition (if present) and the actual Windows partition (usually disk C:). The size of Disk2vhd-created image will be the sum of the actual files on the partition (used space), not the size of a whole partition. If all goes well, the image should just boot in a VM and you will not have to go through the hassle with MBR and Windows bootloader, as in the case of cloning an entire partition.

Tasks on GNU/Linux

Tip: Skip the partition-related parts if you created VHD image with Disk2vhd.

- Reduce the native Windows partition size to the size Windows actually needs with `ntfsresize` available from `ntfs-3g (https://www.archlinux.org/packages/?name=ntfs-3g)`. The size you will specify will be the same size of the VDI that will be created in the next step. If this size is too low, you may break your Windows install and the latter might not boot at all.

  Use the `--no-action` option first to run a test:

  ```
  # ntfsresize --no-action --size 52Gi /dev/sda1
  ```

  If only the previous test succeeded, execute this command again, but this time without the aforementioned test flag.

- Install VirtualBox on your GNU/Linux host (see **#Installation steps for Arch Linux hosts**[**broken link**: invalid section] if your host is Arch Linux).

- Create the Windows disk image from the beginning of the drive to the end of the first partition where is located your Windows installation. Copying from the beginning of the disk is necessary because the MBR space at the beginning of the drive needs to be on the virtual drive along with the Windows partition. In this example two following partitions `sda2` and `sda3` will be later removed from the partition table and the MBR bootloader will be updated.

  ```
  # sectnum=$(( $(cat /sys/block/''sda/sda1''/start) + $(cat /sys/block/''sda/sda1''/size) ))
  ```

  Using `cat /sys/block/sda/sda1/size` will output the number of total sectors of the first partition of the disk `sda`. Adapt where necessary.

  ```
  # dd if=''/dev/sda'' bs=512 count=$sectnum | VBoxManage convertfromraw stdin ''windows.vdi'' $(( $sectnum * 512 ))
  ```

  We need to display the size in byte, `$(( $sectnum * 512 ))` will convert the sector numbers to bytes.

- Since you created your disk image as root, set the right ownership to the virtual disk image:

  ```
  # chown your_user:your_group windows.vdi
  ```

- Create your virtual machine configuration file and use the virtual disk created previously as the main virtual hard disk.

- Try to boot your Windows VM, it may just work. First though remove and repair disks from the boot process as it may interfere (and likely will) booting into safe-mode.

- Attempt to boot your Windows virtual machine in safe mode (press the F8 key before the Windows logo shows up)... if running into boot issues, read **#Fix MBR and Microsoft bootloader**. In safe-mode, drivers will be installed likely by the Windows plug-and-play detection mechanism **view (http://i.imgur.com/hh1RrSp.png)**. Additionally, install the VirtualBox Guest Additions via the menu *Devices* > *Insert Guest Additions CD image...*. If a new disk dialog does not appear, navigate to the CD drive and start the installer manually.

- You should finally have a working Windows virtual machine. Do not forget to read the **#Known limitations**.

- Performance tip: according to **VirtualBox manual (http://www.virtualbox.org/manual/ch05.html#harddiskcontrollers)**, SATA controller has a better performance than IDE. If you cannot boot Windows off virtual SATA controller right away, it is probably due to the lack of SATA drivers. Attach virtual disk to IDE controller, create an empty SATA controller and boot the VM - Windows should automatically install SATA drivers for the controller. You can then shutdown VM, detach virtual disk from IDE controller and attach it to SATA controller instead.

Fix MBR and Microsoft bootloader

If your Windows virtual machine refuses to boot, you may need to apply the following modifications to your virtual machine.

- Boot a GNU/Live live distribution inside your virtual machine before Windows starts up.
- Remove other partitions entries from the virtual disk MBR. Indeed, since we copied the MBR and only the Windows partition, the entries of the other partitions are still present in the MBR, but the partitions are not available anymore. Use `fdisk` to achieve this for example.

```
fdisk ''/dev/sda''
Command (m for help): a
Partition number (''1-3'', default ''3''): ''1''
```

- Write the updated partition table to the disk (this will recreate the MBR) using the `m` command inside `fdisk`.
- Use **testdisk (https://www.archlinux.org/packages/?name=testdisk)** (see **here (http://www.cgsecurity.org/index.html?testdisk.html)** for details) to add a generic MBR:

```
# testdisk > Disk /dev/sda...' > [Proceed] >  [Intel] Intel/PC partition > [MBR Code] Write TestDisk MBR to first sector > Write a new copy of
 MBR code to first sector? (Y/n) > Y > Write a new copy of MBR code, confirm? (Y/N) > A new copy of MBR code has been written. You have to reb
oot for the change to take effect. > [OK]
```

- With the new MBR and updated partition table, your Windows virtual machine should be able to boot. If you are still encountering issues, boot your Windows recovery disk from on of the previous step, and inside your Windows RE environment, execute the commands **described here (http://support.microsoft.com/kb/927392)**.

Known limitations

- Your virtual machine can sometimes hang and overrun your RAM, this can be caused by conflicting drivers still installed inside your Windows virtual machine. Good luck to find them!
- Additional software expecting a given driver beneath may either not be disabled/uninstalled or needs to be uninstalled first as the drivers that are no longer available.
- Your Windows installation must reside on the first partition for the above process to work. If this requirement is not met, the process might be achieved too, but this had not been tested. This will require either copying the MBR and editing in hexadecimal see **VirtualBox: booting cloned disk (http://superuser.com/questions/237782/virtualbox-booting-cloned-disk/253417#253417)** or will require to fix the partition table **manually (http://gparted.org/h2-fix-msdos-pt.php)** or by repairing Windows with the recovery disk you created in a previous step. Let us consider our Windows installation on the second partition; we will copy the MBR, then the second partition where to the disk image. `VBoxManage convertfromraw` needs the total number of bytes that will be written: calculated thanks to the size of the MBR (the start of the first partition) plus the size of the second (Windows) partition.

```
{ dd if=/dev/sda bs=512 count=$(cat /sys/block/sda/sda1/start) ; dd
if=/dev/sda2 bs=512 count=$(cat /sys/block/sda/sda2/size) ; } |
VBoxManage convertfromraw stdin windows.vdi $(( ($(cat
/sys/block/sda/sda1/start) + $(cat /sys/block/sda/sda2/size)) * 512 ))
```

.

Run a Windows partition in VirtualBox

Note: The technique outlined in this section only applies to **UEFI** systems.

In some cases, it is useful to be able to **dual boot with Windows** *and* access the partition in a virtual machine. This process is significantly different from **#Move a native Windows installation to a virtual machine** in several ways:

- The Windows partition is not copied to a virtual disk image. Instead, a raw VMDK file is created
- Changes in the VM will be mirrored in the partition, and vice versa
- OEM licenses should still be satisfied, since the Windows partition still boots directly on the hardware

Warning: Some of the commands used here can corrupt either the Windows partition, the Arch Linux partition, or both. Use extreme caution when executing commands, and double check that they are being run in the correct shell. It would be a good idea to have a backup of the entire drive ready before beginning this process.

Tip: It will be useful to have the **Arch install ISO (https://www.archlinux.org/download/)** readily available, as it will be used the configure the **EFI System Partition** in the VM. It is also worth having access to Windows installation media (such as the **Windows 10 ISO (https://www.microsoft.com/en-us/software-download/windows10ISO)**) in case the Windows partition is corrupted.

Configure udev to give VirtualBox raw access to Windows partitions

VirtualBox must have **raw disk access (https://www.virtualbox.org/manual/ch09.html#rawdisk)** in order to run a Windows partition. Normally, this would require VirtualBox to be run with full root privileges, but **udev** can be configured to restrict access to certain partitions. One convenient way to do this assign all Windows-related partitions to the *vboxusers* group, which can be done with the following udev rules:

Note: The UUIDs in these rules correspond to particular **GPT partition types**. In this case, the UUIDs correspond to all **Microsoft Reserved** (*msftres*) and **Microsoft basic data** (*msftdata*) partitions.

```
/etc/udev/rules.d/99-vbox.rules

# Rules to give VirtualBox users raw access to Windows partitions

# Microsoft Reserved partitions (msftres)
SUBSYSTEM=="block", ENV{ID_PART_ENTRY_TYPE}=="e3c9e316-0b5c-4db8-817d-f92df00215ae", GROUP="vboxusers"

# Windows partitions (msftdata)
SUBSYSTEM=="block", ENV{ID_PART_ENTRY_TYPE}=="ebd0a0a2-b9e5-4433-87c0-68b6b72699c7", GROUP="vboxusers"
```

Tip: Useful udev environment variables and attributes corresponding to a particular partition can be found by running `udevadm info /dev/sdXX`.

VirtualBox configuration

A VirtualBox VM must be manually created with a custom configuration.

Note: Do not add a virtual disk during the initial creation of the VM.

Configure the VM with the following settings:

- System
  - Motherboard
    - *Enable I/O APIC*
    - *Enable EFI*
    - *Hardware Clock in UTC Time*
  - Acceleration
    - Paravirtualization Interface: *Hyper-V*
    - *Enable VT-x/AMD-V*
    - *Enable Nested Paging*
- Display > Screen > Acceleration
  - *Enable 3D Acceleration*
  - *Enable 2D Acceleration*
- USB > *Enable USB Controller*

Note: The *Hyper-V* setting is not required in order for the system to operate correctly, but it may help avoid licensing issues.

Create virtual disk image files

To access the Windows partitions, create a **raw VMDK file (https://www.virtualbox.org/manual/ch09.html#rawdisk)** pointing to the relevant Windows partitions and set the ownership:

```
# vboxmanage internalcommands createrawvmdk -filename /path/to/vm/folder/windows.vmdk -rawdisk /dev/sdx -partitions res,data -relative
# chown user:group /path/to/vm/folder/windows.vmdk
```

Note: When creating the raw VMDK file...

- Root access because VirtualBox must read the partition table
- The `-rawdisk` parameter should point to a block device, not a partition
- *res* is the partition number of the Microsoft Reserved Partition
- *data* is the partition number of the Microsoft basic data partition (i.e. the Windows installation)
- An extra file (*windows-pt.vmdk*) will also be created
- *windows.vmdk* will need to be re-created if the partition table is changed

After creating the image, change the ownership so that the desired user/group has access:

In order to boot the virtual machine in UEFI mode, a dedicated virtual disk for the **EFI System Partition** must be created:

```
$ vboxmanage createmedium disk --filename /path/to/vm/folder/esp.vmdk --size 512 --format VMDK
```

Attach virtual disk images to the VM

Set up a separate ESP in VirtualBox

Configure the virtual UEFI firmware to use the Windows bootloader

Retrieved from "https://wiki.archlinux.org/index.php?title=VirtualBox/Tips_and_tricks&oldid=497579"