

Installation Manual

Linux Ubuntu Server 16.04 LTS
on

Physical Machine (Host)
“skyqode-server1”

w/

VirtualBox (5.2.10)
Virtual Machine (Guest)
“skyqode-server1-vm1”

Compiled by
Andre N. Basson

2019.05.29
Draft 1

Featuring *Virtual Box* Sources:

-
- <https://askubuntu.com/questions/41478/how-do-i-install-the-virtualbox-version-from-oracle-to-install-an-extension-pack>
 - <https://askubuntu.com/questions/367248/how-to-install-virtualbox-from-command-line>
 - <https://www.ostechnix.com/install-oracle-virtualbox-ubuntu-16-04-headless-server/>
 - <https://sourceforge.net/p/phpvirtualbox/wiki/Common%20phpVirtualBox%20Errors%20and%20Issues/>
 - <https://www.virtualbox.org/manual/ch02.html#install-linux-host>
-

1. HDD Partitioning for Host OS & VM

As with any new Operating System installation, we are required to format storage space and assign a file system hierarchy.

As the title stipulates - a Debian based Ubuntu Server is the choice of operating system for this project. Subsequently, all commands used reflect this choice, and should be updated to the equivalent in the Linux flavour you have chosen.

The Linux file system for this project will employ the following partition layout:

- /
 - o Root partition
 - o 20GB, ext4 primary partition, at beginning of HDD free space
- /boot
 - o Boot partition (incl. Grub)
 - o 1024MB, ext4 primary partition, at beginning of HDD free space
- /vm
 - o Virtual machine partition – to store virtual operating systems
 - o Allocate hard drive space for both VMs and VM snapshot
 - Snapshots is a remarkable feature which allows you to rollback the guest system in a virtual machine to a previous time
 - o 20GB per VM (*.vdi) + 20GB per VM for snapshots, ext4 logical partition, at end of HDD free space:
 - Eg. 3 x VMs = 3 x (20GB + 20GB) = 120GB partition
 - Putting the partition at the end of HDD space also makes for easier resizing later if needed.
- /home
 - o Home partition – ie. user profile directories
 - o 5GB, ext4 logical partition, at end of HDD free space
- Swap
 - o Swap partition
 - o 1.5 x RAM GB, swap space, at beginning of HDD free space

2. Host OS Installation & Configuration

A full set of instructions for installing and configuring Ubuntu Server can be found in another of our publications.

For this project we have chosen Ubuntu Server 16.04 LTS, configured with the following user credentials:

- Ubuntu Server 16.04 LTS
 - o Host name: skycode-server¹

- 1 to refer to number in the line of servers
- User name: andre
- Password: als gaan goed + _ + admin password

3. VirtualBox Installation on Ubuntu Server

(with ref. [VirtualBox User Manual](#), chapter 02)

Ensure previous VirtualBox (if any) installation is removed (purged)

- Check if installed:
 - `sudo dpkg -l | grep -Ei "virtualbox.*"`
 - OR...
 - `sudo systemctl status virtualbox`
- Remove package:
 - `sudo apt purge <package-name>`

Add the official Oracle VB repository to your *sources list*

- With apt package manager:
 - `sudo add-apt-repository "deb http://download.virtualbox.org/virtualbox/debian $(lsb_release -cs) contrib"`
- Manually
 - Get your current Ubuntu distribution with:
 - `echo $(lsb_release -cs)`
 - `sudo nano /etc/apt/sources.list`
 - add the line at the bottom:
 - `deb http://download.virtualbox.org/virtualbox/debian distribution contrib`

Download and add public/signature key(s) to *apt* to verify new downloads

- `wget -q https://www.virtualbox.org/download/oracle_vbox_2016.asc -O-`
`| sudo apt-key add -`

- `wget -q
http://download.virtualbox.org/virtualbox/debian/oracle_vbox.
asc -O- | sudo apt-key add -`

Update the Ubuntu source package list, and upgrade outdated packages installed:

- `sudo apt-get update; sudo apt-get upgrade -y`

Install VirtualBox Prerequisites on Host System

Build-essentials

A package with reference to all packages required to compile a Debian package

- `sudo apt install build-essential-y`

Linux headers

These are the C (programming language) header files that define portions of the API exposed to developers by the Kernel.

They are necessary in order to build loadable kernel modules, or other drivers, but are not required to build application software. Run:

- `sudo apt install linux-headers-$(uname -r) -y`

DKMS (Dynamic Kernel Module Support) Package

If you don't want to compile the kernel driver every time your kernel updates it is also recommended to install DKMS.

DKMS provides support for installing supplementary versions of kernel modules. The package compiles and installs into the kernel tree. Uninstalling restores the previous modules. By default, installation is into the current kernel tree, but any kernel tree can be selected with command-line options.

Furthermore, DKMS is called automatically upon installation of new Ubuntu kernel-image packages, and therefore modules added to DKMS will be automatically carried across updates.

- **NOTE - some manuals perform this installation AFTER the VirtualBox installation**
 - **i.e. if errors appear – remove first, then install after VirtualBox installed**

- `sudo apt install dkms`

OR...

Remove, then start over

(note you may experiment with two versions as shown below)

- `sudo apt-get autoremove dkms [virtualbox-dkms]`
`sudo apt-get install dkms [virtualbox-dkms]`

A server reboot might be required now.

Install VirtualBox on the Host OS:

Run:

- `sudo apt-get install virtualbox-5.2`
 - replace '5.2' with latest version if so required

You can also try:

- `sudo apt-get install virtualbox`

Install VirtualBox kernel module *vboxdrv*

First confirm to see if not already loaded and active (running)

- `sudo systemctl status vboxdrv` # on newer Ubuntu versions
- `sudo /etc/init.d/vboxdrv status` # on older Ubuntu versions

See if you can start it (might be loaded, but not running)

- `sudo /etc/init.d/vboxdrv setup`

OR failing....

- `sudo systemctl start vboxdrv`

TBC: If none of the above works, (re-)install from scratch, and confirm as above again

- `sudo service vboxdrv setup`

Install the Oracle VB Extension Pack (includes VBoxManage)

The VirtualBox Extension pack provides the following functionalities to the VirtualBox guests:

- The virtual USB 2.0 (EHCI) device
 - VirtualBox Remote Desktop Protocol (VRDP) support
 - Host webcam passthrough
 - Intel PXE boot ROM
 - Experimental support for PCI passthrough on Linux hosts
- Note: Extension Packs always need to be for the same Virtual Box version

IMPORTANT: Must be installed AFTER *VirtualBox* and *vboxdrv* module has been installed.

Download the Extension Pack from <https://www.virtualbox.org/wiki/Downloads>

- Wget https://download.virtualbox.org/virtualbox/5.2.18/Oracle_VM_VirtualBox_Extension_Pack-5.2.18.vbox-extpack
 - o replace version (5.2.18) and package name (Oracle_VM_VirtualBox_Extension_Pack-5.2.18.vbox-extpack) with the latest found at the above *Downloads* directory

Then install Extension pack with:

- `sudo VBoxManage extpack install [--replace] <filename>`
- e.g.
 - o `sudo VBoxManage extpack install Oracle_VM_VirtualBox_Extension_Pack-5.2.18.vbox-extpack`
- use the option `--replace` only in case you have an older version of the extension pack already installed.

OR failing.... the following from the repository might also work:

- o `sudo apt install virtualbox-ext-pack`

Updating Extension Pack

Note: On upgrading Virtual Box to a newer version we also have to manually upgrade the extension pack. This will not be done automatically from the repository.

- **TBC:** One option would be not to have the VirtualBox package broken (and thus crucial VM downtime); you may want to reconfigure the `unattended-upgrades` package, by selectively disabling individual sources to auto-update. You do so by adding the package name(s) to the `Unattended-Upgrade::Package-Blacklist` object, eg.:

- o `Sudo nano /etc/apt/apt.conf.d/50unattended-upgrades`
 - Add the lines:
 - `Unattended-Upgrade::Package-Blacklist {
 "vim";
 //"libc6";
};`
- o Restart the `unattended-upgrades` package
 - `Sudo systemctl restart unattended-upgrades`

Optional: VirtualBox *Guest Additions*

Extra features to communicate and integrate better with host OS.

Guest OS Preparation: Guest Additions supporting software

(ref. <https://www.virtualbox.org/manual/ch04.html>)

First we must prepare your guest OS for building external kernel modules/drivers, similarly as done for the host OS prior to installing VirtualBox. Install the following **on the GUEST system** (not the host!):

- **Build-Essentials**

- A package with reference to all packages required to compile a Debian package.

Run:

- `sudo apt install build-essential-y`

- **Linux Headers**

- These are the C (programming language) header files that define portions of the API exposed to developers by the Kernel.
- They are necessary in order to build loadable kernel modules, or other drivers, but are not required to build application software. Run:

- `sudo apt install linux-headers-$(uname -r) -y`

- **DKMS**

- If you don't want to compile the kernel driver every time your kernel updates it is also recommended to install DKMS.

DKMS provides support for installing supplementary versions of kernel modules. The package compiles and installs into the kernel tree. Uninstalling restores the previous modules. By default, installation is into the current kernel tree, but any kernel tree can be selected with command-line options.

Furthermore, DKMS is called automatically upon installation of new Ubuntu kernel-image packages, and therefore modules added to DKMS will be automatically carried across updates.

- **NOTE - some manuals perform this installation AFTER the VirtualBox installation**
 - **i.e. if errors appear – remove first, then install after VirtualBox installed**

- `sudo apt install dkms`

OR...

Remove, then start over

(note you may experiment with two versions as shown below)

- `sudo apt-get autoremove dkms [virtualbox-dkms]`
`sudo apt-get install dkms [virtualbox-dkms]`

~~A guest system reboot might be required now.~~

If you suspect that something has gone wrong with module installation as above, check that your guest is set up correctly and run the following command as *root*:

- `rcvboxadd setup`

If you run into several issues or cannot continue, a simple remove or purge of packages installed might be in order:

- Check for package(s) installed, eg. syntax:
 - `sudo dpkg -l | grep -Ei "(packageName1|packageName2|..|packageName)"`
- Remove package:
 - `sudo apt purge <package-name>`

Install VirtualBox Guest Additions

Installed **on the Guest OS** running in the virtual machine (VM), **after** the Guest OS has been installed on the VM.

Full install instructions are available [here](#) or [here](#), of which the most pertinent are shown below. Several options exist, pick one of the following:

NOTE: Stop/shutdown the Guest OS on the VM first (will fail otherwise), then choose **one** of the following installations - statement TBC.

Install from VBoxGuestAdditions.iso CD Image

- easiest done from *VM VirtualBox Manager* GUI if available, or *phpvirtualbox* web console in case of systems using Command Line Interfaces (CLIs) like Linux Ubuntu Server
 - some functions TBC available in *phpvirtualbox* web console
- requires the `VBoxGuestAdditions.iso` image file from either install CD, LiveUSB, or download. Preferably you should download it from the [official release page](#), so that you can pick the one which corresponds to the VirtualBox version installed on the host OS.
- If you downloaded onto the host, then the ISO will have to be inserted/mounted into the virtual machine's virtual CD-ROM
 - **Menu-bar method:**

- The menu-bar at the top of the Guest OS Window has to be visible for this, so follow the instructions given in section *Connecting to Guest OS on VM: VB Guest Container Window* to enable it, if it isn't already.
 - Then, in the *Devices* menu in the menu bar, Oracle VM VirtualBox has a menu item *Insert Guest Additions CD Image*, which mounts the Guest Additions ISO file inside your virtual machine.
- **Controller method:**
 - add a SATA storage controller (if not already added) for either hard disk or optical disk to your virtual machine:
 - *settings > storage > add optical > leave empty*
 - add/mount the ISO as a CD/optical-device in the SATA controller or press `RCtrl` (aka "host") + `D` to load the ISO in your virtual machine to accomplish the same. A result similar to the next picture should be obtained.
- In the guest system you should now have access to a CD-ROM with the installer
 - **From guest OS running a GUI**
 - Guest Additions installer might autostart at the moment the mount was completed
 - If not, simply browse to the executable to the executable on the mounted image ISO, and double click to start installation
 - **From guest OS running a CLI (eg. Linux Server)**
 - Unlike GUI based systems (Windows, Mac, Ubuntu Desktop etc.), in systems like Linux Server, you still have to manually mount the `VBoxGuestAdditions.iso` CD image to the file system on the guest OS:
 - `sudo mount /dev/cdrom /media/cdrom`
 - If successful you should receive a message about the block device being mounted and that it is *read-only*.
 - `mount: block device /dev/sr0 is write-protected, mounting read-only`
 - Confirm that the software packages (eg. Linux Headers at minimum) required by the `VBoxGuestAdditions.iso` have been installed
 - First get the currently running kernel version
 - `uname -r`
 - Then a list of the software to confirm installed
 - `sudo dpkg -l | grep -Ei "(linux-headers|headers)"`
 - If the version of the software installed corresponds with the kernel version, then the software has been confirmed and you may continue.
 - If it doesn't, follow the instructions in the preceding section *Guest OS Preparation: Guest Additions Supporting Software*
 - Change to the directory where your CD-ROM drive is mounted and run the following command as *root*:

for 64bit users:

```
sudo /media/cdrom/VBoxLinuxAdditions-amd64.run
```

for 32bit users:

```
sudo /media/cdrom/VBoxLinuxAdditions-x86.run
```

```
andre@hpe-190a.vn.skygode1:/$ sudo /media/cdrom/VBoxLinuxAdditions.run
[sudo] password for andre:
Verifying archive integrity... All good.
Uncompressing VirtualBox 5.2.10 Guest Additions for Linux.....
VirtualBox Guest Additions installer
Copying additional installer modules ...
Installing additional modules ...
VirtualBox Guest Additions: Building the VirtualBox Guest Additions kernel modules.
VirtualBox Guest Additions: Look at /var/log/vboxadd-setup.log to find out what went wrong
VirtualBox Guest Additions: Running kernel modules will not be replaced until the system is restarted
VirtualBox Guest Additions: Starting.
andre@hpe-190a.vn.skygode1:/$
```

The installation will run automatically and complete with some messages (possibly) informing you that the Guest Additions kernel modules, folder support, and non-kernel Guest Additions have been built. You could potentially also receive some error messages saying the Window System drivers have failed to install, which is fine because our guest machine is not running X-Windows.

Now I recommend rebooting the guest machine and re-logging in.

- `sudo shutdown -r now`

OR

- `sudo reboot`

Install from ISO (locally mounted)

- `sudo apt-get install virtualbox-guest-additions-iso`
- The `.iso` file with an image of the OSE edition of the guest additions CD will install in the host directory `/usr/share/virtualbox/VBoxGuestAdditions.iso`.
- Mount this `.iso` file as a CD/optical-device in your virtual machine's STORAGE settings under SATA controller.
- In the guest you will then have access to a CD-ROM with the installer.

Install from repository

- `sudo apt-get install virtualbox-guest-additions`
- This will install guest additions matching the Virtual Box version as obtained from the repositories. It is not recommended to install these in newer releases of Virtual Box as obtained from the Oracle repository

Updating Guest Additions

Guest Additions can simply be updated by going through the installation procedure again with an updated CD-ROM image (ISO). This will replace the drivers with updated versions. You should reboot after updating the Guest Additions.

4. Congratulations! What's next?

You should now have successfully installed Oracle VirtualBox with the extension pack in Ubuntu server. It is almost time now to add deploy virtual machines.

Should you choose to manage your upcoming virtual machines via the *command line*, please refer to the [virtualbox official guide](#) (or downloaded manual, ch08) to start creating and managing virtual machines.

Not everyone, though, is a command line expert. Some might want to create and use virtual machines via a graphical interface. This is where `phpVirtualBox` comes in handy!!

In the following paragraphs we'll start by creating and adding a user to the `vboxusers` group. We'll install `phpVirtualBox` along with Apache web-server, make some adjustments to the local firewall to allow traffic where/how it should, and finally create a Virtual Machine with which to host a guest OS.

All of this is covered next, and is largely based on an article found [here](#).

5. Adding Users to VirtualBox Group

You can either create a *separate* user, or use an *existing* user, with which to manage your VMs.

As you will later be required to add a password for this user to a `config.php` file - **in clear text** – you may wish to create a separate user instead of using your default (admin?) account. You will be limiting access permission to this configuration file and thus should be okay. Nevertheless, the password is now ever so slightly less secure for being in clear text somewhere. So why not create a new user specifically for virtualbox.

We can create a new user with this command:

```
o sudo adduser <username>
```

The user needs to be added to the `vboxusers` group

```
o sudo usermod -aG vboxusers <username>
```

```
o eg.: sudo usermod -aG vboxusers vboxuser_andre
```

Please note that if you use a separate user for virtualbox, you must log out and log in to that user first. The rest of the steps will be followed while logged in under this user. You may have to add the user to the `sudo` group to gain the required permissions for some of the tasks to follow.

Before proceeding though, confirm the virtualbox kernel module/driver is installed and active with:

- o `sudo systemctl status vboxdrv`
- o if not, follow the installation instruction given earlier

6. Install phpVirtualBox

phpVirtualBox is a free, web-based front-end to VirtualBox. It is written using the PHP language. Using phpVirtualBox, we can easily create, delete, manage and administer virtual machines via a web browser from any remote system on the network.

We'll be installing **Apache web server** to deliver the interface to the phpVirtualBox web-service at the default TCP port 18083. Along with that, we'll also require PHP and some PHP modules.

To do so, execute the following command:

- o `sudo apt install apache2 \`
`php php-mysql libapache2-mod-php \`
`php-soap php-xml`

Now download the **phpVirtualBox** package from its official [release page](#). Note that the version downloaded (to be subsequently installed) should be of the same major release version as that of **VirtualBox**. For example: phpVirtualBox version 5.2.x must be installed with VirtualBox 5.2.

To download phpVirtualBox, execute:

- o `wget https://github.com/phpvirtualbox/phpvirtualbox/archive/5.2-0.zip`
- *where:* **5.2-0** reflects the version in question

Extract the downloaded archive with:

- o `unzip 5.2-0.zip`

This command will extract the contents of *zip* file into a folder named *phpvirtualbox-5.2-0*.

Now, copy or move the contents of this folder to your apache web server root folder.

- o `sudo mv phpvirtualbox-5.2-0/ /var/www/html/phpvirtualbox`

Assign the proper permissions to the phpvirtualbox folder (**only**).

- o `sudo chmod 777 /var/www/html/phpvirtualbox/`
- **note** that the `chmod-R` switch (recursively change permissions folder content) was **not** used, because we **donot** want to assign these permissions to the **content** under the folder as well.

Configure phpVirtualBox

Next, we configure phpVirtualBox with `config.php`. Start out by creating a new configuration based on a sample accompanying the phpVirtualBox installation:

- o `sudo cp /var/www/html/phpvirtualbox/config.php-example /var/www/html/phpvirtualbox/config.php`

Edit the phpVirtualBox `config.php` file:

- o `sudo nano /var/www/html/phpvirtualbox/config.php`
- o find the following lines and replace the *username* and *password* with the user you added to the `vboxusers` group earlier, eg.:
 - o `var $username = 'vboxuser_andre';`
`var $password = '*****';`

Save and close the file.

Next, we create a new file called `/etc/default/virtualbox`:

- o `sudo nano /etc/default/virtualbox`
- o at minimum, add the following line; replace 'vboxuser_andre' with your own username.
 - o `VBOXWEB_USER=vboxuser_andre`
- o also add these – note some commented out for inclusion when troubleshooting
 - o `VBOXWEB_HOST=127.0.0.1`
`#VBOXWEB_HOST=hostOS-IP-address`

`##autostart database directory - must have write access for users who want to be able`
`# to autostart VMs - must have 'sticky bit'`
`# ie. sudo chown -R vboxuser_andre:vboxusers /etc/vbox; sudo chmod 1775 /etc/vbox`
`#`
`#VBOXAUTOSTART_DB=/etc/vbox`

`##points service to autostartconfig file - ie. which user(s) is allowed to start a`
`# VM automatically, and how much startup delay is.`
`#`
`#VBOXAUTOSTART_CONFIG=/etc/vbox/vbox.cfg`

Finally, Reboot your system or simply restart the following services to complete the configuration.

- `sudo systemctl restart vboxweb-service`
- `sudo systemctl restart vboxdrv`
- `sudo systemctl restart apache2`

Important Side Note:

To prevent `read` access to other users of *config.php* (you are after all leaving your password exposed within), you may wish to reduce read/write/traverse permissions of it to 660 (ugo) instead of the default 664.

To do so, run this command:

- `sudo chmod 660 /var/www/html/phpvirtualbox/config.php`

Do note, however, that should this result in login issues in the phpVirtualBox web console; you'll have to revert back to the default permissions at this moment. More research required on this matter. Also see the troubleshooting section.

Adjust Firewall to Allow Access to Apache Web Server

By default, the apache web browser can't be accessed from remote systems if you have enabled the UFW firewall in Ubuntu. You must allow the http and https traffic in UFW by following the below steps.

Firewall Application Profiles

It is perfectly fine to allow traffic, via firewall ports, using conventional syntax, eg.:

- `sudo ufw allow in 443/tcp`

On the other hand, applications often install pre-set firewall profiles that we may use instead. These profiles accomplish the same as conventional syntax, but without the need of picking a port or protocol to allow.

We can view which applications have installed a profile using command:

- `sudo ufw app list`

```
Available applications:
Apache
Apache Full
Apache Secure
OpenSSH
```

As you can see, *Apache* and *OpenSSH* applications have installed UFW profiles.

If you look into the "Apache Full" profile, you will see that it enables traffic to the TCP ports 80 and 443, for HTTP and HTTPS traffic respectively:

- `sudo ufw app info "Apache Full"`

```
Profile: Apache Full
Title: Web Server (HTTP,HTTPS)
```

Description: Apache v2 is the next generation of the Apache web server.

Ports:
80,443/tcp

Security Side Note

Before we configure the firewall for incoming traffic, it is worth reflecting on some security decisions in regard to your gateway router.

If access to the Apache web server is to be conducted **across the internet**; the recommended standard for gaining secure access to the phpVirtualBox web-service would be to do so via *Virtual Private Network* (VPN). In particular, the following steps are of importance:

- avoid forwarding ports on your gateway router for HTTP (80) or HTTPS (443) traffic directly to the system hosting Apache
- use either existing VPN on same LAN as the Apache hosting server, or setup new VPN server on the Apache hosting server
 - DNS routing is preferable for extra security (all traffic diverted through VPN)
 - Clients are to connect via VPN to the hosting server, from where a webbrowser can be directed at the phpVirtualBox web-service running on the Apache web server.

If access to the Apache web server is to be conducted over the internet – **but without the wish to use a VPN** – then one should consider forwarding gateway router port 443 (HTTPS) **only**, followed by enabling the same port (or “Apache Secure” profile) on the host’s firewall.

Assigning Rules

Having discussed UFW application profiles and security concerns, we can now run the following command to allow incoming HTTP and HTTPS traffic for this profile:

- **`sudo ufw allow in "Apache Full"`**

```
Rules updated
Rules updated (v6)
```

If you want to allow http traffic (80) only, run:

- `sudo ufw app info "Apache"`

Similarly, if you want to allow https (443) traffic (only), run:

- `sudo ufw app info "Apache Secure"`

7. Preparing VM Space

Right at the start of this project, you had created a separate partition for your Virtual Machines. The size of the partition was estimated on the number of VMs you'd wish to create.

If you have not already, you'll now create a directory on this partition which to store your VM(s) within.

Create VM Directory

Assuming the VM partition is mounted on the host OS at `/vm`, we can now create a directory for the virtual machine by running this command, eg.:

- `sudo mkdir -p /vm/VirtualBox_VMs/ skyqode-server1-vm1`

The last directory in the path (in red) should reflect the name and number of the host OS and VM configuration. In this instance, the chosen 'server1' and 'vm1' scheme reserves the possible inclusion of more servers and more virtual machines per server. In this manner admins can deploy a range of servers - say, in a load-balanced cluster – without having to contemplate different (yet applicable) host names for each. Instead they can now all share the same name, differing only in number adjective.

Eg.:

- `.../skyqode-server1-vm1`
- `.../skyqode-server1-vm2`
- `.../skyqode-server1-vm3`
- `.../skyqode-server2-vm1`
- `.../skyqode-server2-vm2`
- `.../skyqode-server2-vm3`
- Etc.

Access Permissions

Permissions to the VM directory should be granted to the same user(s) added earlier to the `vboxusers` group (ref. section *Adding Users to VirtualBox Group*).

A simple way to do so would be to grant **group(g)** ownership to the `vboxusers` group:

- `sudo chown -R root:vboxusers /vm/VirtualBox_VMs`

- Note the inclusion of the `-R` switch to apply the ownership recursively to all files and folders

If the above isn't sufficient to grant access, you may wish to change the **owner(u)** and group ownership to the specific user (or group), eg.:

- `sudo chown -R vboxuser_andre:vboxuser_andre /vm/VirtualBox_VMs`
 - `vboxuser_andre` should reflect the user added to group `vboxusers` earlier.

...or create a combination of owner/group ownership, eg.:

- `sudo chown -R vboxuser_andre:vboxusers /vm/VirtualBox_VMs`

8. Access phpVirtualBox Web console

Now, go to any local or remote system with access to the system hosting *virtualbox* and Apache web server (phpVirtualBox), and open up a web browser.

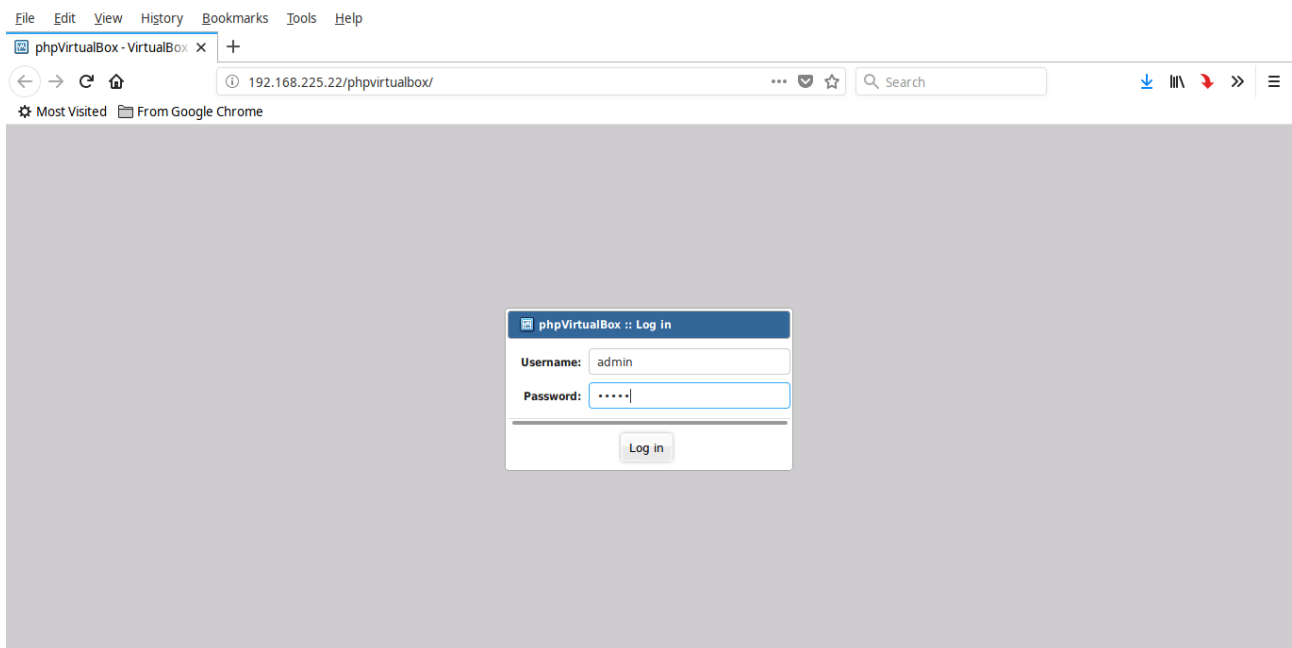
In the address bar, type: <http://IP-address-of-virtualbox-headless-server/phpvirtualbox>. This should connect to the Apache web server on the host, trigger the phpVirtualBox web-service, and respond by serving the phpVirtualBox web application to the web browser.

In my case, I navigated to this link – <http://192.168.178.35/phpvirtualbox>

If a browser was used on the same host as the one running virtualbox/Apache, then you could use:

- this link - <http://127.0.0.1/phpvirtualbox> or
- this link – <http://localhost/phpvirtualbox>

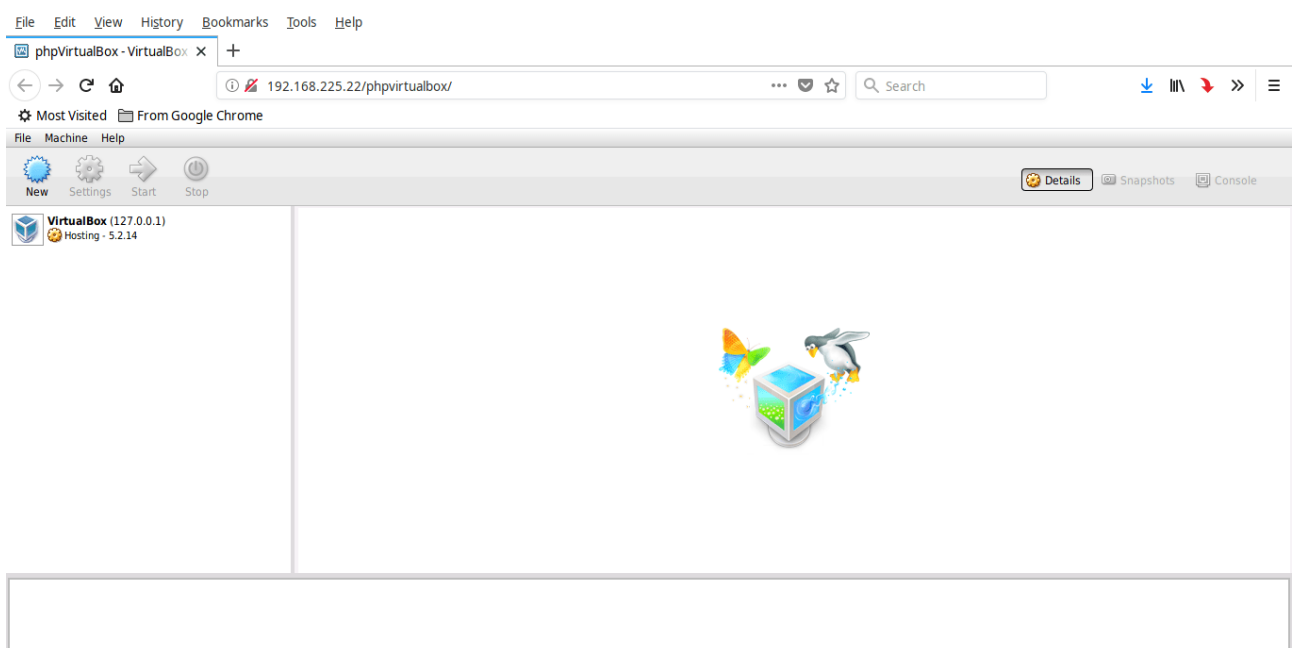
You should see a screen with dialogue window with login/password fields appears.



Enter the phpVirtualBox administrative user credentials. The initial (and default) username and password is `admin/admin`. You can add a new user under the *File > Preferences > Users* menu if you please.

NOTE: You should change the default password as soon as possible. Not doing so will give anyone with knowledge of the default credentials, and access to the network, free access to the web service and thus access to manage any and all VMs.
(see "Security Side Note" section on how to secure the host OS and services)

On successful login, you should be greeted with the phpVirtualBox dashboard.



You may now create your VMs and manage them via this dashboard.

9. Creating or Adding VMs with phpVirtualBox

If you haven't enabled virtualization support in the BIOS of the host system (not the guest), phpVirtualBox will allow you to create 32-bit guests only. To install 64-bit guest systems, you must enable virtualization in your host system's BIOS. Look for an option that is similar to "virtualization" or "hypervisor" in your BIOS and enable it.

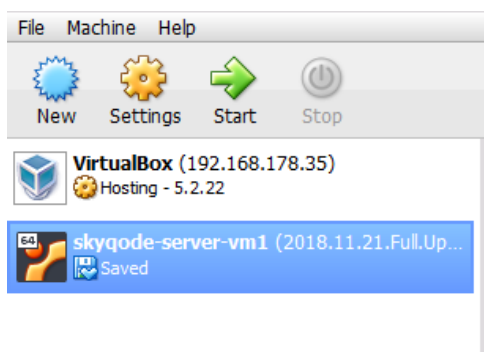
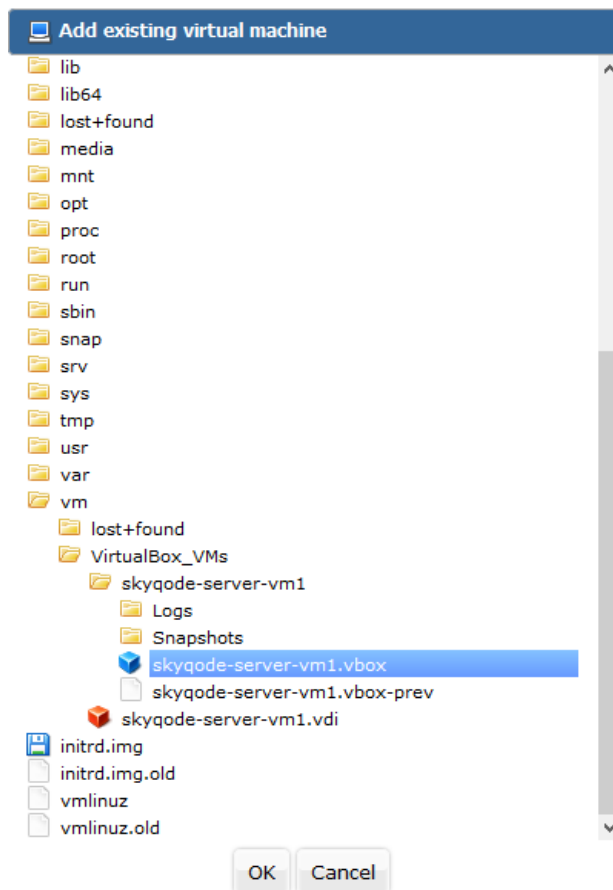
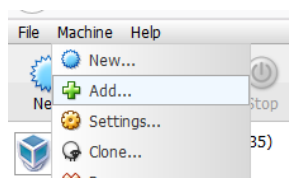
The rest of this section will cover how to add an existing VM, or create a new one from scratch. We start by opening a web browser and enter the follow in the address bar:

- `http://IP-address-of-virtualbox-server/phpvirtualbox`
 - eg.: `http://192.168.178.35/phpvirtualbox`

Adding an Existing Virtual Machine

Should you already have created a VM with guest OS on a previous occasion, simply follow these steps to add it :

- From the drop down menu at the top of the page, open the "+Add" submenu under the "Machine" menu tab heading.
- An "Add existing virtual machine" dialogue window should appear
- Finally, locate and select the virtual machine by file extension *.vbox; press OK to add the virtual machine to list of VMs.



Creating a New Virtual Machine

Side Note: VM migration between different systems

Yes, in most cases VMs **are** transferable between different, but compatible systems.

Virtualization software essentially functions on the basis of creating an abstraction layer of the hardware on the physical machine, which it then packages and presents as an environment (API) to which a guest OS can interface. Since this environment (API) mirrors that of a physical machine's hardware interface; the guest OS is unaware that it is running inside this environment.

In this way – as long as two physical machines on which you run the same virtualization software doesn't differ in terms of general compatibility (at least not to the degree that the virtualization setup cannot accommodate the differences) - **there is no reason why you cannot simply copy the VM files from one system onto another, and have it run there.** The VM on the 2nd machine simply requires a setup the same (or similar) as the one on the 1st system.

To create a virtual machine from scratch, follow these steps:

- From the drop down menu at the top of the page, open the "+Add" submenu under the "Machine" menu tab heading.



- From this point forward, there are two ways in which to configure the new VM. You can either pick the *guided* or *expert mode*; the latter gives you more direct control.
- Whichever method you choose, your VM should have the following characteristics applied where available. Take note – and with transferability in mind - try and configure the VM with hardware settings you know is likely to be available on most systems.
 - **Settings >GENERAL:**
 - **ENCRYPTION:**
 - default is to leave virtual disk **unencrypted**
 - Too many recovery issues when things go wrong!!!
 - Your system should be maintained securely otherwise both in sound configuration (firewalls, key based authentication, good passwords, permissions, etc.) and limiting physical access.
 - **Settings >SYSTEM:**

- VIRTUAL DISK:
 - Choose a **fixed**(not dynamic) sized VDI (virtual disk image)
 - I don't suspect you'll need much space (eg. 10GB) – and a fixed VDI is usually faster than dynamic sized one.
 - It also leaves more space on your HDD for VM snapshots!!
 - When having to choose the hard drive controller prior to installing an OS onto the VM, it may help running one or more of the following to find out what type of storage controller (IDE, PATA, SATA, etc.) you have:
 - `sudo lshw | less`
 - `sudo hwinfo -disk | less`
 - `sudo hwinfo --storage-ctrl | less`
 - More likely than not you're system will use a SATA hard drive controller.
- RAM / MEMORY:
 - Linux requires between 512MB and 1024MB to function properly.
 - Leave approx. 1024MB for the host (physical) system, and the same amount (or more) for any guest (VM) system you may want to create
 - The true amount is dependent on the size and/or amount of tasks you envisage could be running at any one time. In most cases 1024MB is more than sufficient.
- CPU CORES:
 - Choose a maximum of 1 core per VM to run at full speed
 - default settings otherwise will do
- **DEVICES:**
 - USB / OPTICAL / SERIAL DEVICES:
 - Best to enable the devices (eg. USB, optical) you may wish to use in the VM; especially for OS installation in the VM
 - AUDIO / DISPLAY:
 - If you're installing a headless VM (no GUI, peripherals, etc.), then assign the default minimum resources. If not, assign a bit more (but not that much!!)
- **Settings > NETWORK ADAPTER:**
 - The intention here is to have the Guest OS on the VM to believe it has its own network interface.
 - set up Adapter 1 as a '*bridged adapter*', so that host and guest OS appears to have separate network adapters to the rest of the network
 - the shell command `ip a` on the host OS will show you the name of interface you will have to bridge
 - under '*Advanced*', select '*Allow VMs*' in the '*Promiscuous Mode*' field

- 'Cable Connected' field should be ticked.
- the default NAT setting will suffice in most other scenarios. This way the host's network interface will translate all incoming addresses between itself and the child (VM) network interface.

TIPS:

- If you intend to SSH in to the guest OS running on the VM, then you should start the VM as a *headless server*.
- If later you would like to update any settings of the VM in the virtualbox manager (using phpVirtualBox web console), please make sure to *shutdown* the guest OS inside the VM first. Any other state (eg. 'saved' state) will result in all or most settings to be greyed out and not allow you to make changes.

10. Autostarting VirtualBox VMs (guest OS's) on Host Boot

WARNING: set up of *autostart* has proven problematic and has caused complete failure resulting in complete reinstallation of virtualbox. More experimenting is required. Initial centiment suggests that file/folder access permission and/or ownership might play a role.

Until a workable solution is found, it is recommended that all VMs are to be started manually from within phpVirtualBox web console.

The following is an excerpt based on official Virtual Box documentation for chapter 09 'autostart'.

On Linux, the autostart service is activated by setting two variables in `/etc/default/virtualbox`:

- VBOXAUTOSTART
- VBOXAUTOSTART_CONFIG

You might recall we already created this file back in section *Install phpVirtualBox: Configure phpVirtualBox*. We included some entries we haven't discussed yet. We will do so now.

If you haven't already added the following lines, open the file and do so now:

```
o  sudo nano /etc/default/virtualbox

##autostart database directory
# Must have write access for users who want to be able
# to autostart VMs -also must have 'sticky bit' set
#   eg.  sudochgrp vboxusers /etc/vbox
#        sudochmod 1775 /etc/vbox
```

```
#
VBOXAUTOSTART_DB=/etc/vbox

##points service to autostartconfig file
# ie. which user(s) is allowed to start a
#   VM automatically, and how much startup delay is.
#
VBOXAUTOSTART_CONFIG=/etc/vbox/vbox.cfg
```

Side note:

If for some reason the name *vbox.cfg* doesn't work, you may also try renaming it *autostart.cfg*.

The *first* variable added is VBOXAUTOSTART_DB which contains an absolute path to the autostart database directory (/etc/vbox). The directory should have write access for every user who should be able to start virtual machines automatically. For a user vboxuser_andre added to group vboxusers, you can do so by executing:

- `sudo chgrp vboxusers /etc/vbox`

or even...

- `sudo chown -R root:vboxusers /etc/vbox`

or even...

- `sudo chown -R vboxuser_andre:vboxusers /etc/vbox`

The directory should also have the sticky bit set to prevent deletions or renaming of the folder:

- `sudo chmod 1775 /etc/vbox`

Side Note:

If you have changed group permissions for the current user, log out and back in again to refresh the permissions.

The *second* variable added is VBOXAUTOSTART_CONFIG which points the service to the autostart configuration file which is used during boot to determine whether to allow individual users to start a VM automatically and configure startup delays.

The configuration file can be placed in /etc/vbox and contains several options. One is *default_policy* which controls whether the autostart service allows or denies to start a VM for users which are not in the exception list. The exception list starts with *exception_list* and contains a comma separated list with usernames.

Furthermore, a separate startup delay can be configured for every user to avoid overloading the host. A sample configuration is given below:

```
# Default policy is to deny starting a VM, the other option
# is "allow".
default_policy = deny

# vboxuser_andre is allowed to start virtual machines but
# starting them will be delayed for 10 seconds
vboxuser_andre = {
    allow = true
    startup_delay = 20
}

# Heidi is not allowed to start virtual machines: useful to
# exclude certain users if the default policy is set to allow.
#heidi = {
#    allow = false
#}
```

Side Note:

If you are the only user you can just add the line
`default_policy = allow` to the *vbox.cfg* file.

Every user (from *vboxusers* group) who wants to enable autostart for individual machines, will **have** to set the path to the autostart database directory using the *virtualbox manager (VBoxManage)* command:

- `VBoxManage setproperty autostartdbpath <Autostart directory>`
 - eg.: `VBoxManage setproperty autostartdbpath /etc/vbox`

Now the user will instruct the manager to enable autostart for each VM:

- `VBoxManage modifyvm <uuid|vmname> --autostart-enabled on`
 - eg.: `VBoxManage modifyvm skycode-server1-vm1 --autostart-enabled on`
- This will create a *username.id* file in */etc/vbox* directory.
- **Note**, the command only has to be used for the VM(s) you want enabled
 - a very easy way to select which VM(s) to autostart or not

And finally restart the `vboxautostart-service` to read in the changes:

- `sudo service vboxautostart-service restart`
- **Note**, you require *sudo* privileges, so you may have to login to another user with admin wrights if your *vbox users* user hasn't been added to the *sudo* group

The VM(s) should now start automatically after the host system has (re)booted.

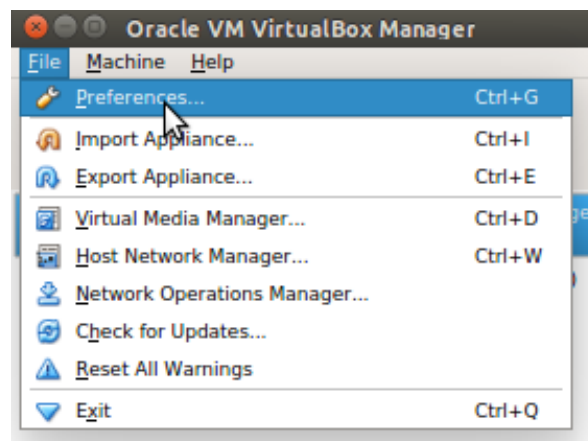
11. Connecting to Guest OS on VM

There are three ways in which to connect to a Guest OS on the virtual machine:

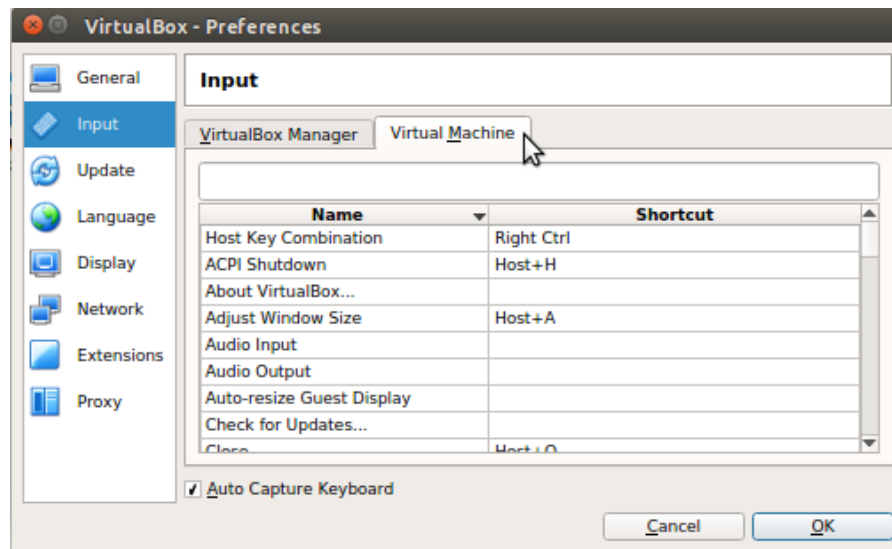
- VirtualBox Guest Window
- Secure Shell (SSH) protocol
- Remote Desktop Protocol (RDP)
- phpVirtualBox 'console' interface

VB Guest Container Window

- Part of the VB *Guest Additions* package – not to be confused with *phpVirtualBox* or its limited *console* interface – runs on the guest OS
 - Graphically simulates an interface to the guest OS in a wrapped window
 - whether text based like Ubuntu Server, or GUI based Linux Desktop ,Windows, Mac, etc.
 - Brings useful extras (eg. menu-bar or popup-window) not provided in other interfaces such as SSH.
 - **Do note**, however, that the full-blown VirtualBox Manager GUI package available for Windows and Linux Desktop releases do offer options not yet available in the phpVirtualBox web application version.
 - **At time of writing, both menu bar and popup window is NOT available in phpVirtualBox web console. TBC; is it due to VB guest additions not being installed on Ubuntu Server guest OS? Is it even possible? Or is guest additions limited to full-blown GUI VirtualBox Manager only?**
 - Admittedly, these extras are of less use once configuration of VB has been finalized. However until then, and indeed for any change in future configurations, these might prove helpful.
- At time of writing the following directions are available for the full blown VirtualBox Manager GUI package and not in phpVirtualBox web console.**
- Short-cuts to the enable/display both menu-bar and popup can be found (or indeed customized) from the VirtualBox Manager GUI or phpVirtualBox web console if available as indicated below
 - Start by clicking on *File > Preferences*

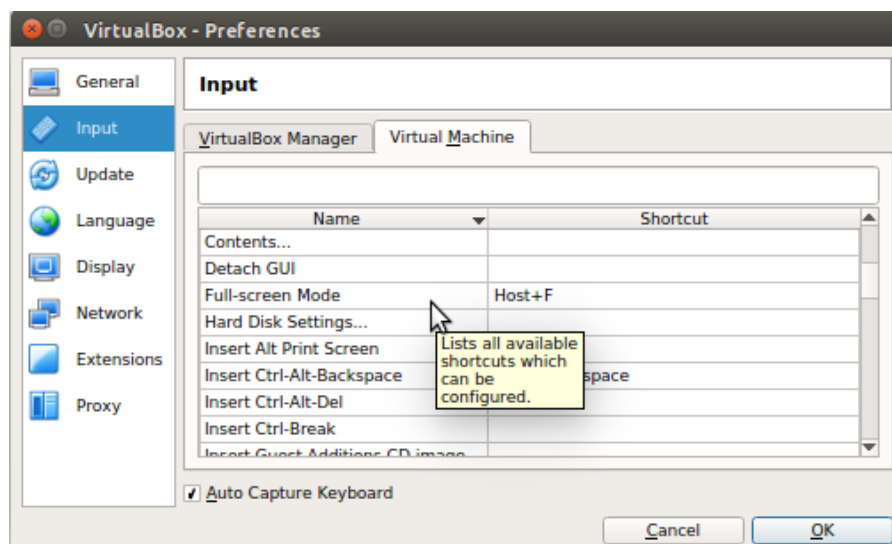


- Click then on *Input*, and *Virtual Machine*

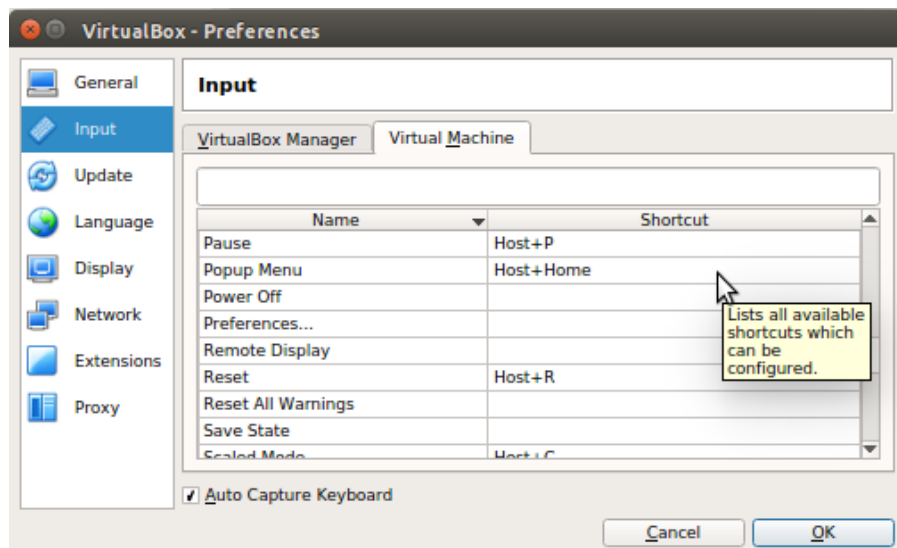


- By default the menu bar of the RDP client running the guest OS is not enabled/visible. Scrolling through the shortcut commands for the Virtual Machine, the following can be useful to help in configuration the virtual machine.

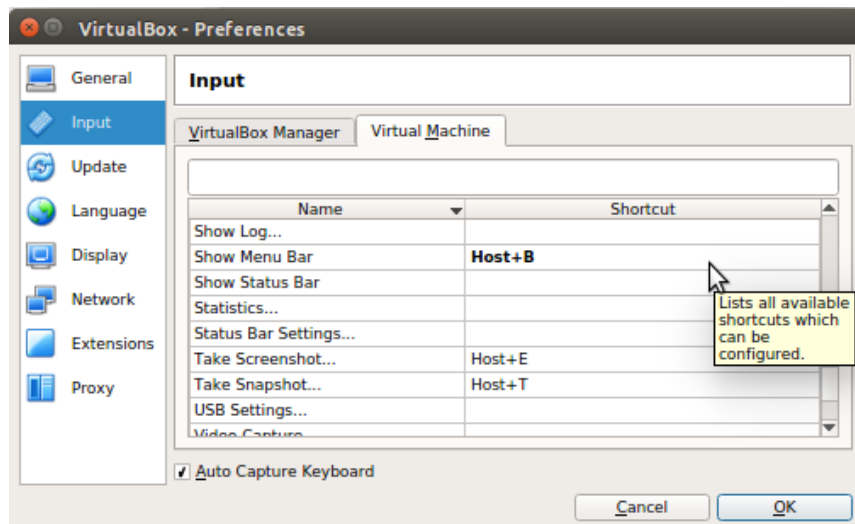
Note that in Linux parlance, the `RCtrl` is often referred to as the `host` key :



- `Host+F` will maximise the RDP window to full-screen, along with adding a menu-bar



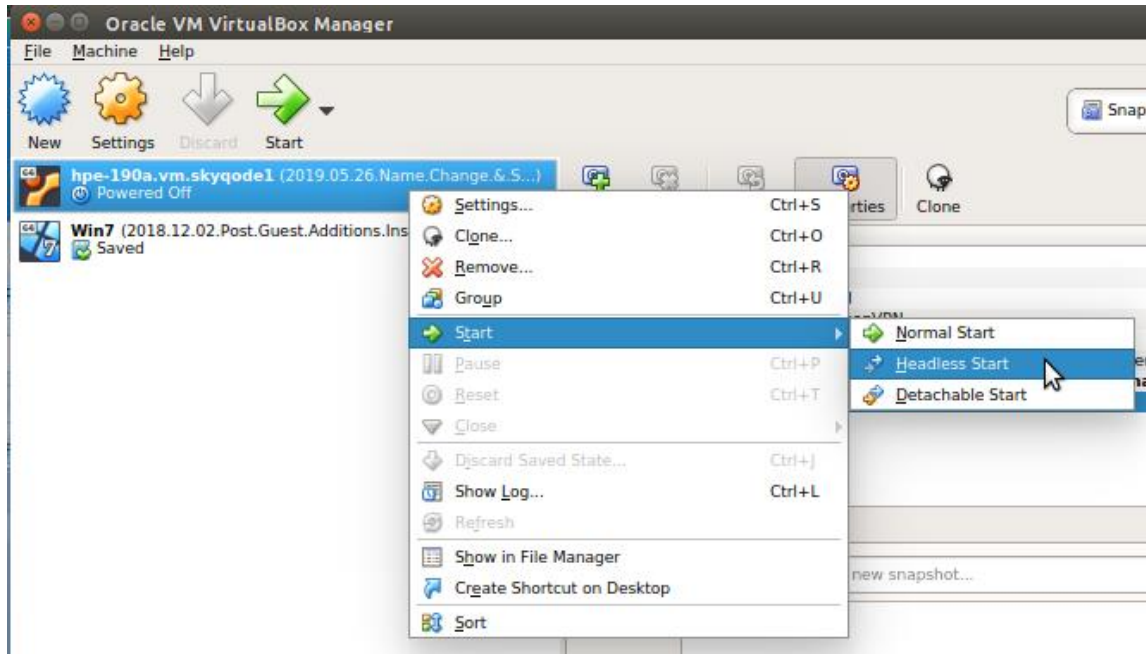
- Host+home will produce a pop-up window in the middle of the RDP window



- Host+B will toggle the menu bar at the top of the RDP window.
 - NOTE: You may have to leave scaled mode first before the toggle will work, press: Host+C
 - Simply enter <IP-or-hostname>:<port> in your web browser's address field to connect with the guest OS on the VM; you should see a BASH terminal appearing shortly:
 - Eg. 192.168.178.35:9001

SSH (Secure Shell)

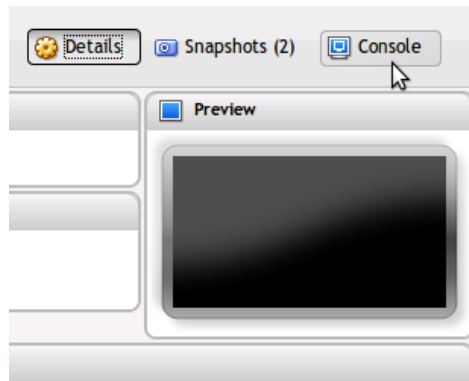
- Usually, you would use one of the other methods to establish an initial connection, makes some configuration changes, and then set up SSH connectivity to establish secure future connections with guest OS on the VM.
- If connection to the guest OS on the VM is to be established via SSH, then it is common practice to run the VM in *headless* mode.



RDP Client

- VirtualBox (on the host OS) runs an RDP server which exposes a remote connection service by which an RDP client can connect to a guest OS running on a virtual machine.
 - eg.:
 - MS Windows 'Remote Desktop Protocol Connection'
 - Linux Ubuntu Desktop 'Remmina'
- RDP clients are useful when the phpvirtualbox 'console' interface to VM guest OS is not registering some keys – as sometimes happens – or causes other hard to troubleshoot issues
- RDP clients can access the RDP server at TCP port 9001 (default)
 - Eg. `<host-name-or-IP>:9001`

phpVirtualBox 'console' interface



- bare minimum interface – bottom of the barrel stuff
- sometimes causes issues with some keys (CTRL, SHIFT, ALT) not working
 - could be time consuming to troubleshoot (and still not be working)
 - eg. to send a `ctrl+c` combo to the console, you have to press `ctrl+c` `c` the `rAlt+c` `c`
 - ie. Press `<Ctrl|rAlt>+c` then release and then press `c` immediately after
 - similar combinations for `ctrl+x`, `ctrl+o` combos etc.
- often easier to just use a dedicated RDP application running on a client to connect to guest OS on VM

12. VirtualBox Shared Folders

(ref. <https://devtidbits.com/2010/03/11/virtualbox-shared-folders-with-ubuntu-server-guest/>,
<https://askubuntu.com/questions/161759/how-to-access-a-shared-folder-in-virtualbox>,
<https://askubuntu.com/questions/30396/error-mounting-virtualbox-shared-folders-in-an-ubuntu-guest/30446#30446>,
<https://www.virtualbox.org/manual/ch04.html#sharedfolders>,
<https://www.virtualbox.org/manual/ch08.html#vboxmanage-sharedfolder>)

Introduction (and a cautionary tail)

VirtualBox's *Shared folders* facility is a quick and easy way to allow a guest operating system on the VM access to files stored on the host system.

It does come with a limitation though – **file & folder permissions (and ownership) are not carried forward to the guest system** – at least this is the belief of this author after many hours of online research at time of writing this manual (VirtualBox release 5.2.10)

Nonetheless, if read-only access or permissions are not an issue, *Shared Folders* is less risky than mounting a block device (HDD) to either host or guest system – a type of access referred to as **raw disk access**. The risk of corruption and mass data loss is a distinct possibility when attempting to mount or access the physical device when already mounted on the other system.

Caution to be practiced when *raw disk access* is used instead of using *shared folders*.

Prerequisites

Folders physically residing on the host are be shared to the guest via a special file system driver supplied by *Guest Additions*. For Windows guests, these folders are implemented as a pseudo-network interface. For Linux guests, Guest Additions provide a virtual file system – aptly called *vboxsf*.

In order to use the *shared folder* functionality, a few [prerequisites](#) need to be met. Some of the following have already been covered in earlier text; some are covered next; others will be covered in the text following where and when required (don't worry!):

- Make sure that *Guest Additions* are properly installed on the guest OS.
- Users in a guest Ubuntu must be in the group `vboxsf` to be able to access shares.
 - Only *vboxsf* users are allowed access to the shared content
 - Run:
 - `sudo usermod -aG vboxsf <username>`
 - eg.: `sudo usermod -aG ${whoami}`
- Define a directory on the host that will be used in the virtual machine using the settings dialogue of Virtual Box.

Depending on host or guest OS the following may also be needed:

- Do not share personal folders like `/home/username` or `My Documents`
- Avoid special characters or empty spaces in the path to the shared folder, especially if the host or VM is a Windows machine
- Use different names for share and mountpoint
- Create a mountpoint on the guest OS.

Type of Shares

There are two types of shares:

- *Permanent* shares, that are saved with the VM settings.
- *Transient* shares, that are added at runtime and disappear when the VM is powered off. These can be created using a checkbox in the *VirtualBox Manager* (or *phpVirtualBox* web console), or by using the `--transient` option of the `VBoxManage sharedfolder add` command (covered below).

Shared folders can either be read-write or read-only. This means that the guest is either allowed to both read and write, or just read files on the host. By default, shared folders are read-write. Read-only folders can be created using a checkbox in the *VirtualBox Manager* (or *phpVirtualBox* web console), or with the `--readonly` option of the `VBoxManage sharedfolder add` command (discussed shortly).

By default, the guest system has read/write access to the files on the host path. On Linux distributions, shared folders are mounted with 770 file permissions with *root* user and *vboxsf* as the group (*vboxsf group* not to be confused with *vboxsf virtual file system*).

Side Note: So what about shared folder access permissions and ownerships?

Access permissions and ownership attributes of files and folders shared on the host is **NOT** carried to the guest OS – at least not by time of writing this manual (VB release 5.2.10).

There are some options available when mounting the shared folder on the guest, but it is the opinion of this author that it would **not** be possible to mirror the permissions/ownerships **exactly** as on the physical drive. Either that or the effort would not be worth the effort.

Oracle VM VirtualBox shared folders also support symbolic links. For security reasons the guest OS is not allowed to create symlinks by default. For more information on how to enable for symlinks and its implementation, see [this](#) link.

Steps to Share Folders

When sharing folders, you can either decide to create a new folder, or use an existing one; just as long as it is accessible in the file system on the host system.

In this section I will be using the example of sharing a 1.8TB hard disk attached to a USB port, which is mounted on the file system of the host operating system at `/media/andre/1.8TB`.

There are 3 methods in which shared folders can be set up for a virtual machine:

1. In the guest window of a running VM, you select *Shared Folders* from the *Devices* menu, or click on the folder icon on the status bar in the bottom right corner.
 - o See section *Install from VBoxGuestAdditions.iso CD Image* for details if the menu bar is not visible.
2. If a VM is not currently running, you can configure shared folders in the virtual machine's *Settings* dialogue, as can be found in either *VM VirtualBox Manager*, or *phpVirtualBox* web console.
3. From the command line on the **host** system, you can create (add) or delete (remove) shared folders using `VBoxManage`, as follows:

Syntax:

```
o VBoxManage sharedfolder      add <uuid|vmname>
                                --name <name> --hostpath <hostpath>
                                [--transient] [--readonly] [--automount]
```

```
o VBoxManage sharedfolder remove <uuid|vmname>
  --name <name> [--transient]
```

where:

<uuid|vmname>: Specifies the UUID or name of the VM whose guest operating system will be sharing folders with the host computer. Mandatory.

--name <name>: Specifies the name of the share. Each share has a unique name within the namespace of the host operating system. Mandatory.

--hostpath <hostpath>: Specifies the absolute path on the host operating system of the directory to be shared with the guest operating system. Mandatory.

--transient: Specifies that the share is transient, meaning that it can be added and removed at runtime and does not persist after the VM has stopped. Optional.

--readonly: Specifies that the share has only read-only access to files at the host path.

--automount: Specifies that the share will be automatically mounted. On Linux distributions, this will be to either /media/USER/sf_<name> or /media/sf_<name>, where <name> is the share named. The actual location depends on the guest OS. Optional.

Example:

```
o VBoxManage sharedfolder add "hpe-190a.vm.skycode1" \
  --name "sharename-on-host" --hostpath "path-to-folder-on-host"
```

Steps to Mount the Shared Folder in the Guest

There are two ways to mount the shared folder in the guest OS:

- **Automatic mounting**
- **Manual Mounting**

Automatic Mounting

The VirtualBox Manager (and phpVirtualBox web console) provides the option to mount shared folders *automatically*. When automatic mounting is enabled for a shared folder, the *Guest Additions* service will mount it for you automatically. For Windows or OS/2, a preferred drive letter can also be specified. For Linux or Oracle Solaris, a mount point directory can also be specified.

If a drive letter or mount point is not specified, or is in use already, an alternative location is found by the *Guest Additions* service. The service searches for an alternative location depending on the guest OS, as follows:

- Windows and OS/2 guests. Search for a free drive letter, starting at Z:. If all drive letters are assigned, the folder is not mounted.

- Linux and Oracle Solaris guests. Folders are mounted under the */media* or */media/user* directory. The folder name is normalized (no spaces, slashes or colons) and is prefixed with *sf_*.

For example, if you have a shared folder called *myfiles*, it will appear as *media/sf_myfiles* or */media/andre/sf_myfiles* in the guest.

Access to an automatically mounted shared folder is granted to everyone in a Windows guest, including the guest user. For Linux and Oracle Solaris guests, access is restricted to members of the group *vboxsf* and the *root* user.

UPDATE:

It would appear that that on reboot of the host system

Manual Mounting

You can mount the shared folder from inside a VM, in the same way as you would mount an ordinary network share:

- In a **Windows guest**, shared folders are browseable and therefore visible in Windows Explorer. To attach the host's shared folder to your Windows guest, open Windows Explorer and look for the folder in *My Networking Places*, *Entire Network*, *Oracle VM VirtualBox Shared Folders*. By right-clicking on a shared folder and selecting *Map Network Drive* from the menu that pops up, you can assign a drive letter to that shared folder.
- In a **Linux guest**, use the following command:
 - `mount -t vboxsf [-o OPTIONS] <sharename> <mountpoint>`
 - eg.1:
 - `sudo mount -t vboxsf 1.8TB /mnt/host-shared/1.8TB_USB`
 - eg.2:
 - `sudo mount -t vboxsf -o uid=1000,gid=1000 \ 1.8TB /mnt/host-shared/1.8TB_USB`
 - eg.3:
 - `sudo mount -t vboxsf \ -o uid=$(id -u $whoami),gid=$(id -g $whoami) \ 1.8TB /mnt/host-shared/1.8TB_HDD`

Fixing Automatic Mount after Guest Reboot

Unfortunately, neither of the automatic or manual mounting options result in the mount to remain after the guest reboots.

The official suggestion is to include the mount command in */etc/fstab* :

- `<sharename> <mountpoint> vboxsf defaults 0 0`

But, even this doesn't guarantee that the mount will actually occur - largely because *fstab* is processed **before** the *Shared Folders* module is loaded. The solution is to execute the mount AFTER the boot has completed.

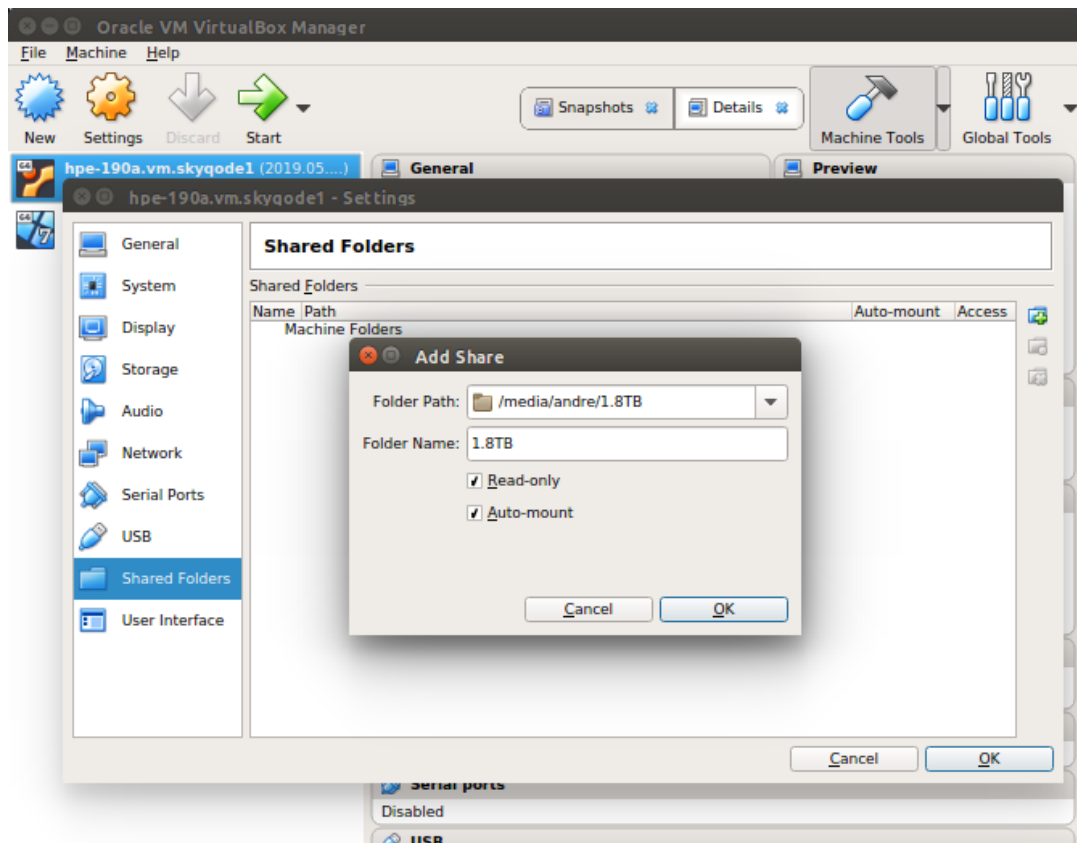
To mount a shared folder **after the system boot**, add the following entry to */etc/rc.local* file:

- `sudo mount -t vboxsf`

Example: Using VirtualBox Manager (method no.2) (and possibly phpVirtualBox web console)

To share the */media/andre/1.8TB* folder, I will take the following steps:

1. Within VirtualBox (either GUI based *VM VirtualBox Manager* or *phpVirtualBox web console*), select the guest machine you wish to have access to the shared folder.
2. Press the *Settings* button (or *mouse-rlick > settings*)
3. In the *Settings* dialogue:
 - a. Click the *Shared Folders* button
 - b. In the popup dialogue, select the directory to share with the guest system,
 - i. eg.: */media/andre/1.8TB*
 - c. Assign a short, simple (no illegal characters or white spaces), yet descriptive name in the *Folder Name* field, with which to represent the shared folder/directory name to the guest OS,
 - i. eg. *1.8TB*
 - ii. this name will be used to mount the shared folder on the guest OS
 - d. select/tick the following based on requirements
 - i. **Read-Only**
 1. when no modifications allowed by the guest system.
 2. Recommended, because by default the guest will assume root read/write permissions on the files/folders on the host – ie. not adhere to the permissions (or ownserhips) on the actual physical device on the host.
 - ii. **Auto-Mount**
 1. if guest OS should try mounting shared folder on startup automatically.
POST-EDIT: doesn't seems to work. Step 4 has a fix though!



4. On the guest OS
 - a. Mount the shared folder to access it on the guest
 - i. `sudo mount -t vboxsf 1.8TB /mnt/host-shared/1.8TB_USB`
 - b. Mount the shared folder after every system boot, every time
 - i. Add the mount command above to the `/etc/rc.local`

The shared folder should now be available at the mount point, and after every system boot.

13. Advanced Storage Configurations: Raw Hard Disk Access on Guest OS

(ref. <https://www.serverwatch.com/server-tutorials/using-a-physical-hard-drive-with-a-virtualbox-vm.html>,
<https://www.virtualbox.org/manual/ch09.html#adv-storage-config>,
<https://superuser.com/questions/467891/how-to-get-write-access-in-guest-os-linux-for-raw-disk-ext4-with-virtualbox>)

As an alternative to using virtual disk images, VirtualBox can also present either entire physical hard disks or selected partitions as virtual disks to virtual machines.

With VirtualBox, this type of access is called *raw hard disk access*. It enables a guest operating system to access its virtual hard disk without going through the host OS file system.

Warning

Raw hard disk access is for expert users only. Incorrect use or use of an outdated configuration can lead to **total loss of data** on the physical disk. Most importantly,

do not attempt to boot the partition with the currently running host operating system in a guest. This will lead to severe data corruption.

In this section, we are going to attempt gaining access to the **entire** physical hard disk, as opposed to just one or more partitions on the physical disk. The first is an option, but it has proven problematic on the occasions this author has tried (and boy have I tried).

As with the section on *Shared Folders*, in this section we will again be using the example of a 1.8TB hard disk attached to a USB port on the **host** system.

Author's IMPORTANT Note:

re. Changes done in guest system (in raw disk access mode) not visible in other systems

It would appear that changes made by the guest to a physical disk connected in *raw disk access mode*; **NOT** to be visible if the physical disk is mounted to the host or any other system.

The reverse is also true – changes made in the host (or another system) is not visible in the guest (raw disk access).

At time of writing, the author of this article has yet to find a solution, if one exists at all.

Step 1: Identify the Block Device (hard drive)

In Linux host OS, run the following command:

- `sudo fdisk -l`

You'll see a listing of the block devices (path) and partitions (number suffix). Make note of the desired drive in the format of *dev/sda0*, *dev/sda1*, etc. In the case of our USB attached 1.8TB hard disk, the pertinent output looks like this:

| Device | Start | End | Sectors | Size | Type |
|------------------|-------|------------|------------|------|------------------|
| /dev/sdj1 | 2048 | 3907028991 | 3907026944 | 1.8T | Linux filesystem |

In the above output, we can see that the block device (hard drive) is listed at **/dev/sdj** and the available partitions to be limited to (primary partition) no. 1.

Optional: You may confirm the numbers for the list of partitions on the block device (eg. */dev/sdj*), you can also run the following command:

- `VBoxManage internalcommands listpartitions -rawdisk /dev/sdj`

The output lists the partition types and sizes to give the user enough information to identify the partitions necessary for the guest.

Step 2: VMDK image for Access to Entire Physical Hard Disk

While this variant is the simplest to set up, you must be aware that this will give a guest operating system direct and full access to an entire physical disk. If your host operating system is also booted from this disk, **please take special care to not access that boot partition from the guest at all.**

Raw hard disk access, both for entire disks and individual partitions, is implemented as part of a *VMDK image* format support. As a result, you will need to create VMDK image file which defines where the data will be stored. After creating such a special VMDK image, you can use it like a regular virtual disk image.

As stated earlier; in this example, we're interested in raw access of **entire physical drive**. This image file will represent the physical drive – an interface or pointer (if you will) to the actual drive connected on the host system. As such, the physical size of the VMDK image will be negligibly small.

Important:

Unlike *Shared Folders*, we do NOT intend to have the physical disk mounted on both host and guest systems simultaneously. In fact, this might lead to some data corruption. So before we create the VMDK image; please ensure that the block device is NOT mounted to any mounting point on the host system with this command:

- `sudo df -h`

In the output you should typically **not** see the block device as listed in the output of `fdisk -l` command used in the previous section.

To create the VMDK image for raw disk access, run a command on the **host** system with the following syntax:

- `sudo VBoxManage internalcommands createrawvmdk -filename </path/to/file>.vmdk -rawdisk /dev/<block-device>`

Example:

For our 1.8TB USB drive (block device) at `/dev/sdj`, we can create the VMDK image on the host system with this command:

- `sudo VBoxManage internalcommands createrawvmdk -filename /mnt/vm_rawDiskAccess/1.8TB.vmdk -rawdisk /dev/sdj`

The example would create the image

`/mnt/vm_rawDiskAccess/1.8TB.vmdk`, which must be an absolute path.

For more information on raw disk access, please see the [Advanced Storage Configuration](#) chapter on the official Oracle VM VirtualBox website.

Step 3: Attach VMDK file to your Linux VM

Now that you've created the VMDK image, you can attach it to a controller in the VM for the guest OS to have access to it.

There are two ways to do so:

1. Using VirtualBox Manager (or phpVirtualBox web console)

- Open VirtualBox Manager (or phpVirtualBox web console)
- Select your Linux VM (mouse-*rclick*) > *Settings* > *Storage* > *SATA controller* > *Add Attachment* > *Add Hard Disk* > locate and *Open* 1.8TB_partition.vmdk image
 - if you're getting *permission* or *failed to open* type errors, close VirtualBox Manager and start it again with `sudo wrights` from the command line:
 - `gksudo virtualbox &`
 - `&` = to run in background
 - Note to self: starting with `sudo` privilege appears to change the VM *.vbox file to root:root user and group ownerships. Is this a concern? Immediate observation is that from now on I will HAVE to run virtualbox with `sudo` privileges from here on out – unless I chown those files back to andre:andre – in which case access to the VMDK image file becomes denied (irrespective of changing permissions to the image file!!)
 - if you're still getting *permission* or *failed to open* type errors when attempting to attach the VMKD image:
 - 1st restart VirtualBox as per the previous step, and attempt attaching the VMDK image again.
 - Still not attaching without errors? Then change the ownership and file permissions of VMDK image file.
- Start the Linux VM

2. Using Command Line *VBoxManage* Command

(Ref. <https://www.virtualbox.org/manual/ch08.html#vboxmanage-storageattach>)

To attach the created image to a virtual machine, use the following syntax:

- ```
VBoxManage storageattach <uuid|vmname>
 --storagectl <name>
 --port <number>
 --device <number>]
 --type dvddrive|hdd|fdd
 --medium < none|emptydrive|
 uuid|filename>|host:<drive>
 >
```

**Note:** port and device numbers are assigned on a sort of trial and error fashion, as no discernable information could be found on the official VB white pages online, and at the time of writing.

## Example:

For our 1.8TB drive (block device) attached to USB port on the host, run:

- `VBoxManage storageattach hpe-190a.vm.skygodel --storagectl "SATA Controller" --port 5 --device 5 --type hdd --medium /mnt/vm_rawDiskAccess/1.8TB.vmdk.vmdk`

Conversely, to **detach** the created image from a virtual machine, stipulate `none` for the `--medium` switch:

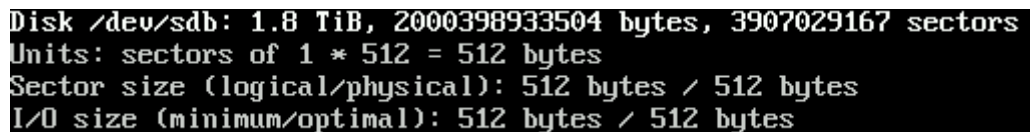
- `VBoxManage storageattach <uuid|vmname>  
--storagectl <name>  
--port <number>  
--device <number>]  
--type dvddrive|hdd|fdd  
--medium < none|emptydrive|  
uuid|filename>|host:<drive>  
>`

## Step 4: Set Permissions and Mount Partition in Guest OS

Now that you've attached VMDK image in the VM, the guest (eg. Linux Server) system running in the VM can mount it to the file system.

Confirm the raw disk is available as block device in the guest with:

- `sudo fdisk -l`



```
Disk /dev/sdb: 1.8 TiB, 2000398933504 bytes, 3907029167 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

Create a directory to server as mountpoint somewhere on the guest:

- `sudo mkdir -p <mountpoint>`
  - eg.: `sudo mkdir -p /mnt/host-rawPartitionAccess`

Set the read/write permissions and u/g/o ownerships:

- `sudo chown <user>:<group> -R <mountpoint>`
  - o eg.: `sudo chown root:andre -R /mnt/host-rawDiskAccess`
- `sudo chmod 770 -R <mountpoint>`
  - o eg.: `sudo chmod 770 -R /mnt/host-rawDiskAccess`

Mount the disk to the mountpoint:

- `sudo mount <blockdevice> <mountpoint>`
  - o eg.: `sudo mount /dev/sdb /mnt/host-rawDiskAccess`

The user(s) with the permission granted should now have access to the raw attached hard drive partition.

## Step 5: Mount Disk Automatically After Guest Boot

As discussed in the Shared Folders, adding the mount command in the `/etc/fstab` file does NOT guarantee the raw disk will be mounted to the guest at boot time.

The solution is to execute the mount **AFTER** system boot has completed – and for that we'll have to enter the mount command in the `/etc/rc.local` file.

Because we cannot assume that Linux will assign the same block-device name for the raw disk attached in the host at boot time (eg. `/dev/sdb1`); we're going to use the `--uuid` switch to mount the device by UUID number instead. The UUID number is unique, and will be the same on any system the drive is attached.

The UUID of the raw disk can be found on the guest OS with:

- `sudo fdisk -l`      #note the block-device name of the raw disk (eg. `/dev/sdb1`)
- `sudo blkid`        #note the UUID of the block-device (given inside quotes)

Then enter this command to the `/etc/rc.local` file to finalize the after-boot mount:

- `sudo mount --uuid <uuid> <mountpoint>`
- eg.:
  - `sudo mount --uuid 6d3032e5-a7e4-45e2-86cb-40d72938346b /mnt/host-rawDiskAccess`

The raw disk will now be mounted automatically to the guest system, after it has booted.

## 14. Troubleshooting:

The following list of actions is to be trialled on a one-by-one case. That is to say, after each change made, the aim should be to restart all related services as follows, and then (r)evaluate if the amendment was successful:

**\*Note: all services must be started with `sudo` privileges.**

- `sudo systemctl restart vboxweb-service && \`  
`sudo systemctl restart vboxdrv && \`  
`sudo systemctl restart apache2`

## VirtualBox Issues

A lot of the virtualbox problems originate from access permissions.

Try the following:



- change ownership (group and/or user) of the directory and files where the virtual machine is located, to the virtual box user you added earlier to the group `vboxusers`
  - eg.: `sudo chown vboxuser_andre:vboxusers -R <path-to-VM-dir>`
- change permission of some of the directory and files (eg. `config.php`) where the virtual machine is located, to the virtual box user you added earlier to the group `vboxusers`
  - eg.: `sudo chmod 664 -R <path-to-config.php>`

## PhpVirtualbox Web Console Issues

### Connection Problems

- Open a port (eg. 18083) on the HOST firewall (UFW) you would like to assign to `phpvirtualbox`
  - `sudo ufw allow 18083/tcp`
- changes to `/var/www/html/phpvirtualbox/config.php`:
  - from: `#var $enableAdvancedConfig = true;`  
to: `var $enableAdvancedConfig = true;`
    - enables you to see (among other items) the 'Net Address' field in the 'Remote Display' section.
  - from: `#var $vrdeports = '9000-9100';`  
to: `var $vrdeports = '9001';`
    - this should coincide with opening port 9001 on UFW for TCP traffic:
      - `sudoufw allow 9001/tcp`
  - from: `var $location = 'http://127.0.0.1:18083/';`  
to: `var $location = 'http://host-ip-address:18083';`  
eg.: `var $location = 'http://192.168.178.35:18083';`
- changes to `/etc/default/virtualbox`:
  - `VBOXWEB_USER=<virtualbox-username>`
  - `VBOXWEB_HOST=<ip-address-of-host-OS>`
    - eg.: `VBOXWEB_HOST=192.168.178.35`
    - Only do this if the default 127.0.0.1 (localhost) doesn't work
    - Host IP must remain static

## login error (api.php)

### If secure authentication and access is not a requirement

- comment in (ie. enable) the following line in *config.php*.
  - `var $noauth = true;`
  - **NOTE:** if commented out, this will give anyone access to your localprivate network access to your VM (and guest OS) without requiring a password!!!!
    - So always make sure you are logged out on the guest OS if you follow this setup
    - As long as you don't open on the gateway router the port virtualbox uses for RDP connections to running guest OS, no direct access from the internet is possible (pending a private host machine has not been compromised of course!!)

### If secure authentication and access is a must

- TBC

## Updating phpVirtualBox (from within phpvirtualbox)

- **settings > display**
  - port 9001 (to match UFW rule)
  - remote display > net address: change from 127.0.0.1 to blank
    - or, if both client and server on same network, you can enter here the IP of the server (host)
    - if client located remotely, ie. NOT on same network, you should really instead leave it blank (so as to default to the localhost/server IP), and connect via VPN to the server or another host (with DNS relay capabilities set up) to get behind the router's firewall and into the private network domain.

## RDP Connection Issues

### RDP client struggles to connect to the Guest OS on the VM

VirtualBox runs an RDP server for RDP clients to connect to the Guest OS on the VM. Connections are by default received at TCP port 9001.

- Open a port (eg. 9001) on the HOST firewall (UFW) you would like to assign to phpvirtualbox
  - `sudo ufw allow 9001/tcp`

