
acontis technologies GmbH

**SOFTWARE**

# acontis Hypervisor

Real-time Hypervisor

Product Manual
Edition: 2021-02-04

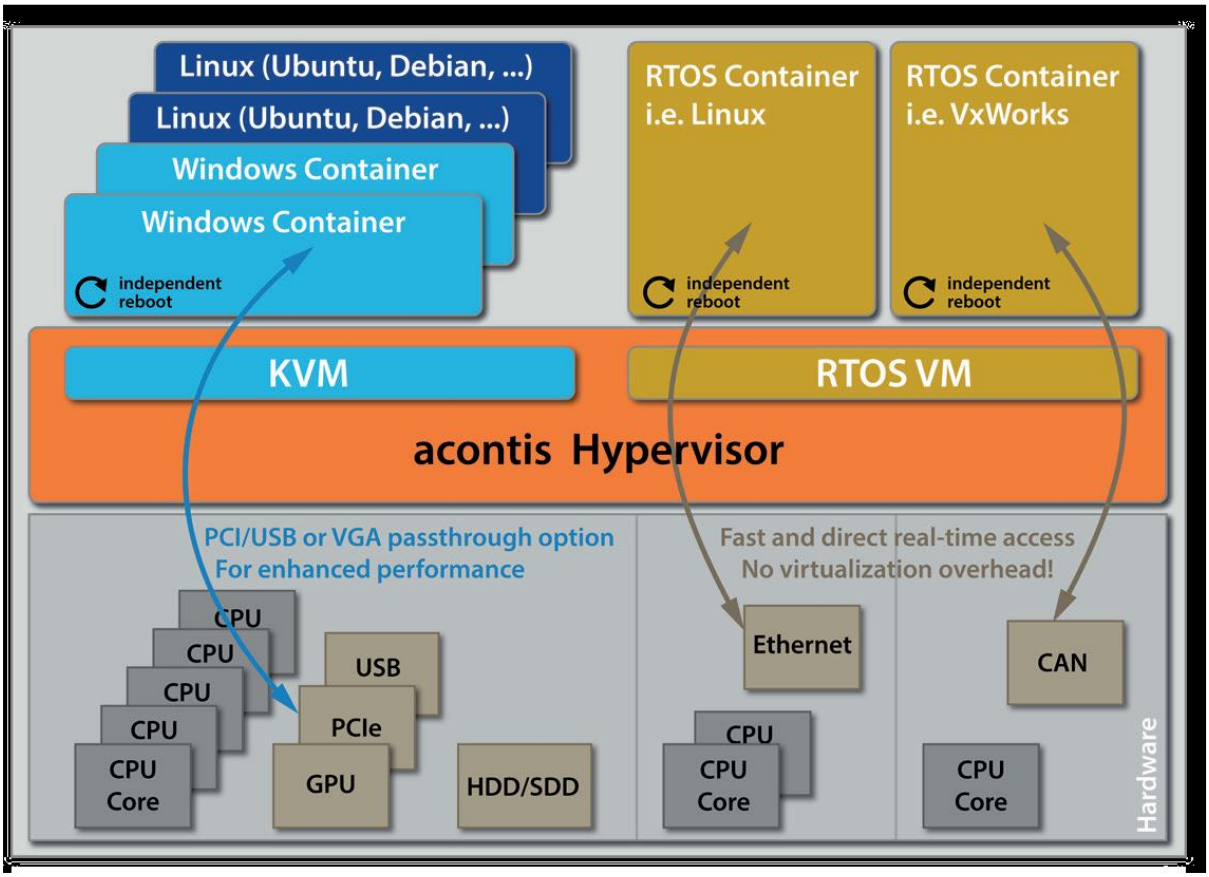
# Compact Table of Contents

# Table of Contents

# 1 Introduction

## 1.1 Overview

Using the existing, industry proven acontis real-time RTOS-VM virtualization technology, multiple hard real-time operating systems (Real-time Linux, VxWorks, etc.) can run in native speed. In addition, based on the well-known and proven KVM virtualization solution, multiple standard operating systems like Windows and Linux (Ubuntu, Debian, Fedora, etc.) operate in parallel. Besides virtual hardware, KVM provides para-virtualization, PCI/USB and VGA pass-through for highest possible performance. Each guest OS is fully independent and separated and can be rebooted or shutdown while the other guests continue without being affected.




**RTOS Virtual Machine Hypervisor**

The RTOS Virtual Machine hypervisor technology provides an independent layer to run any RTOS in native speed. No virtualization overhead is introduced and all RTOS drivers as well the operating systems and applications have direct and fast hardware access. The same technology is successfully used by customers all over the world since more than 10 years in the existing acontis real-time solution. The new acontis Hypervisor can utilize this technology without modification so existing customer applications can be re-used without modification.

## KVM Hypervisor

KVM is one of the most popular Type 1 hypervisors used in many high-availability and security critical environments like the cloud solutions from Google, Amazon or Oracle.

To increase performance, for example for fast USB, Ethernet or Graphics operation, the respective devices can be completely passed through to Windows or Linux guests. Alternatively, para-virtualized devices for the hard disk, the Ethernet or graphics controller reduce the amount of necessary hardware without compromising throughput.

# Key technical features

The acontis Hypervisor is a perfect symbiosis between the wide-spread KVM virtualization solution and the industry proven RTOS Virtual Machine hypervisor for real-time operating systems.

**General**

- Supports Multiple OSes: VxWorks® RTOS, real-time Linux, Standard Linux, Microsoft® Windows®, proprietary Roll-your-own, Bare metal, any unmodified OS
- RTOS containers including applications run on bare metal core with no virtualization overhead and direct hardware access
- Fully separated and independent guest operation
- User defined guest startup sequence
- Utilize any number of CPU cores per single guest
- Independent reboot of all guests while others continue operation
- Virtual Network between all guests
- Inter-OS Communication: Shared Memory, Events, Interlocked data access, Pipes, Message Queues and Real-time sockets for high speed application level communication
- Hypervisor provided fileserver for all guests

**KVM**

Windows and Standard Linux operating systems like Ubuntu or Debian run under control of the KVM hypervisor. This hypervisor provides plenty of sophisticated features:

- Multiple Windows and/or standard Linux instances
- Windows/Linux containers with snapshot support to easily switch between different application situations without the need to install multiple OS instances. Snapshots create a view of a running virtual machine at a given point in time. Multiple snapshots can be saved and used to return to a previous state, in the event of a problem.
- Pass-through support: To increase performance, for example for fast USB, Ethernet or Graphics operation, the respective devices can be completely passed through to Windows or Linux guests.
- Paravirtualization: Para-virtualized devices for the hard disk, the ethernet or graphics controller reduce the amount of necessary hardware without compromising throughput.
- Graphics virtualization to provide 3D accelerated graphics to multiple guests.

**RTOS-VM**

- Multiple real-time Operating Systems (Linux, VxWorks, On Time RTOS-32, etc.)
- Fast real-time interrupt handling and short thread latencies
- Direct hardware access with no virtualization overhead
- Compatible with the existing acontis Windows real-time extension (applications can be shared or cross-migrated between both solutions)

# 2 System Setup

## 2.1 Installation and basic configuration

The hypervisor has to be installed onto an empty installation media. It is directly booted from the BIOS. Side-by-side installation with an existing Windows or Linux is not recommended.
If an existing Windows or Linux system shall be used, assure a free partition for booting the hypervisor is required. The minimum required size for the installation media is 10 GByte.

### *2.1.1 BIOS settings*

In the BIOS, VT-x as well as VT-d has to be enabled, Secure Boot must be disabled.
To assure real-time behavior, the following additional settings have to be accomplished:

- USB Legacy mode has to be disabled
- Hyper-Threading shall be disabled
- Power saving settings have to disabled (C-States, Speedstepping, …)

### *2.1.2 Installation*

Currently, a light-weight Ubuntu derivate (Xubuntu) is used for hypervisor installation.
Select “Install Xubuntu” to install the Hypervisor, other options are not supported.
A user account must be created while installing the hypervisor. Please note, you MUST define a password.
After the installation has finished, you have to reboot the system; do not remove the boot media when rebooting.
Follow the installation instructions.

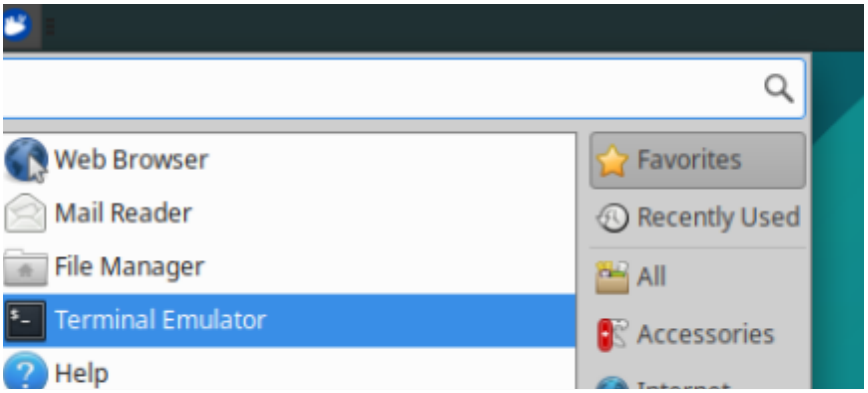
After reboot the default entry “Ubuntu” is selected, keep this and do not boot the “Hypervisor” entry. This entry will be activated automatically after the first-time configuration.

### *2.1.3 **Important note:** privileged commands*

Most of the configuration and launch commands require root privilege. This can be achieved by using the `sudo` prefix before each command or switch once into root mode using the

`sudo -s` command. The password for the user you created on setup has to be entered.

Commands can be entered from the hypervisor shell. Open the shell via the top left button and start the Terminal Emulator:




Alternatively, you can launch the Terminal emulator by typing Ctrl-Alt-T

## *2.1.4 Basic system configuration*

Basic configuration settings including helper scripts are stored in directory /hv/config. Use the cd command to switch into this directory:

```
cd /hv/config
```

If the hypervisor has been installed in parallel to Windows or Linux, the file setrootuuid.sh has to be adjusted. The defaultBoot values need to be set to 3 and 5, respectively.

Enter `gedit setrootuuid.sh` to adjust these values, they are near the end of the file:

```
# update default boot configuration
#####################################
defaultbootRE='\(GRUB_DEFAULT=\)[0-9]*'
[ -d /sys/firmware/efi ] && defaultBoot=3 || defaultBoot=5 # 2 for UEFI, 4 for Legacy
```

Next, you need to run the basic system and memory configuration. The RAM size for the RTOS as well as the memory pool size for all shared memory areas need to be configured. This has to be accomplished using the autoconf.sh script. The following example shows how a system with 64 Mbyte RTOS RAM size and 8 Mbyte shared memory pool size has to be configured:

```
sudo ./autoconf.sh 64 8
```

If you get the following message “WARNING: CPU frequency not stable…”, you did not properly disable power settings in the BIOS. Please follow the respective instructions above.

Now, reboot the system:

```
sudo reboot
```

A second run of the autoconf script may help in case the above CPU frequency warning was generated.

# 2.2 Automatic PCI device assignment to RTOS

## *2.2.1 Ethernet device identification*

Before assigning the device to the RTOS you should identify its name.

Connect a cable with the Ethernet port that shall be used by the RTOS. Disconnect all other ports. The related device name must be determined using the command `ifconfig -a`:

```
ifconfig -a
       ens34: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
              inet 192.168.80.128  netmask 255.255.255.0  broadcast 192.168.80.255
              inet6 fe80::b70:3228:6113:2f1a  prefixlen 64  scopeid 0x20<link>
              ether 00:0c:29:5c:6b:15  txqueuelen 1000  (Ethernet)
              RX packets 12143  bytes 1425085 (1.4 MB)
              RX errors 0  dropped 0  overruns 0  frame 0
              TX packets 544  bytes 99892 (99.8 KB
                TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Several devices may be listed. The ones where an Ethernet cable is connected gets a link (see the <link> string above). In the above case, the device name is ens34.

## *2.2.2 Ethernet device assignment*

Next step is to call the assignment script. Using device name ens34 from the previous chapter you should call the script like:

```
./addeth.sh ens34 rtos_eth 1
```

The script creates 2 files located in /hv/config: rtos_eth.sh, rtos_eth.config.
Calling convention for the addeth.sh script file:

addeth.sh <device name> <RTOS device name> <device number>

Parameters:

device name (mandatory)

This parameter is the Linux device name of the Ethernet device.

RTOS device name (mandatory)

This parameter must be a unique name that is used to identify the device. This name will also be used in filenames that are created by the addeth.sh script:

<RTOS device name>.config - contains the information, required by the RTOS for a proper device configuration.

<RTOS device name>.sh – a script that unbinds the device from the host and binds it to RTOS.

device number (mandatory)

This parameter is a sequential number for each assigned device, starting with 1.
The numbers must be unique. Currently no sanity check will be made.

## *2.2.3 PCI device assignment*

To assign PCI device by PCI ID you should use `addpcidev.sh` script. Calling convention for the script:

```
addpcidev.sh <PCI ID> <RTOS device name> <device number>
```

<RTOS device name> <device number> are the same as for `addeth.sh` for script.

<PCI ID> is in form of `domain:bus:device.function` or `bus:device.function`.

## *2.2.4 Adjust main configuration file*

After creating the device configuration file <RTOS device name>.config, it needs to be included in the main configuration file.
In case of VxWorks, the main configuration file is the file /hv/vx/vxworks.config
In case of On Time RTOS-32, the main configuration file is /hv/rtos-32/realtimedemo.config
In case of Real-time Linux, the main configuration file is /hv/lx/hv.config
Add the corresponding <RTOS device name>.config entries to the includes section of the file.
The example below shows, how rtos_eth device configuration file is included:

```
;--------------------------------------

; includes

;--------------------------------------

#include "../config/membase.config"

#include "../config/memory.config"

#include "../config/cpu.config"

#include "../config/hwbase.config"

#include "../config/hwdevbase.config"

#include "../config/vmf.config"

#include "../config/debug.config"

#include "../config/windowsvm.config"

#include "../config/rtos_eth.config"
```

## *2.2.5 Adjust system startup script /etc/rc.local*

The device assignment scripts <RTOS device name>.sh usually shall be executed automatically on system startup. To accomplish this, add the respective <RTOS device name>.sh calls in the file /etc/rc.local.

The example below shows, how the device with the unique name rtos_eth is assigned.
Please reboot the system to make the change effective.

```
#!/bin/bash
source /hv/config/rtos_eth.sh add
```

## *2.2.6 How to return devices from the RTOS*

You should remove include of <RTOS device name>.config from the main configuration file (/hv/vx/vxworks.config, etc).

Remove reference of <RTOS device name>.sh file in /etc/rc.local

After system restart device will be automatically used by the host. If you want to return the device to the host without rebooting you should call <RTOS device name>.sh script with parameter `delete`:

```
rtos_eth.sh delete
```

Now you can delete <RTOS device name>.sh and <RTOS device name>.config files.

### *2.2.7 Legacy Ethernet PCI devices*

If you have legacy PCI device (not PCIe) it will be required to assign device in another way. A fourth, optional parameter of addeth.sh script shall be used then.

```
addeth.sh <device name> <RTOS device name> <device number> <interrupt type>
```

Possible values are “legacy” or “no_interrupt”. If this parameter is not provided, standard MSI interrupt is configured.

# 2.3 Manual PCI/PCIe device assignment to RTOS

All PCI/PCIe devices that shall be used by the RTOS have to be assigned first. The below steps show how to accomplish this.

## *2.3.1 Ethernet controller*

Connect a cable with the Ethernet port that shall be used by the RTOS. Disconnect all other ports. The related device name must be determined using the command `ifconfig -a`:

```
ifconfig -a
       ens34: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
              inet 192.168.80.128  netmask 255.255.255.0  broadcast 192.168.80.255
              inet6 fe80::b70:3228:6113:2f1a  prefixlen 64  scopeid 0x20<link>
              ether 00:0c:29:5c:6b:15  txqueuelen 1000  (Ethernet)
              RX packets 12143  bytes 1425085 (1.4 MB)
              RX errors 0  dropped 0  overruns 0  frame 0
              TX packets 544  bytes 99892 (99.8 KB
              TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Several devices may be listed. The ones where an Ethernet cable is connected gets a link (see the <link> string above). In the above case, the device name is ens34.

Additional information about the device resources are need for assignment. These will be determined using the command `lshw -class network`:

```
lshw -class network
*-network:1
       description: Ethernet interface
       product: 82545EM Gigabit Ethernet Controller (Copper)
       vendor: Intel Corporation
       physical id: 2
       bus info: pci@0000:02:02.0
       logical name: ens34
       version: 01
       serial: 00:0c:29:5c:6b:15
       size: 1Gbit/s
       capacity: 1Gbit/s
       width: 64 bits
       clock: 66MHz
       capabilities: pm pcix bus_master cap_list rom ethernet physical logical tp
10bt 10bt-fd 100bt 100bt-fd
       resources: irq:16 memory:fd580000-fd59ffff memory:fdfe0000-fdfeffff
ioport:2040(size=64) memory:fd510000-fd51ffff
```

Several devices may be listed. Look for the previously determined device name (in our example: ens34). The resource information marked in red color are needed in further steps.

The ones where an Ethernet cable is connected get a link (see the <link> string above in the ifconfig -a result). In this example case, the device name is ens34.

Determine the original driver, currently used by the hypervisor and the PCI vendor and device id using the previously determined bus info:

```
lspci -s 02:02.0 -v -nn
02:02.0 Ethernet controller [0200]: Intel Corporation 82545EM Gigabit Ethernet
Controller [8086:100f] (rev 01)
        Subsystem: VMware PRO/1000 MT Single Port Adapter
        Physical Slot: 34
        Flags: 66MHz, medium devsel, IRQ 16
        Memory at fd580000 (64-bit, non-prefetchable) [size=128K]
        Memory at fdfe0000 (64-bit, non-prefetchable) [size=64K]
        I/O ports at 2040 [size=64]
        [virtual] Expansion ROM at fd510000 [disabled] [size=64K]
        Capabilities: <access denied>
        Kernel driver in use: e1000
        Kernel modules: e1000
```

## 2.3.2 Other PCI/PCIe devices

Use the lshw command shown in the Ethernet example above and search for the appropriate device. Determine the respective parameters in a similar way as for Ethernet.

## 2.3.3 Assignment script

A template assignment script pcidev1.sh is located in /hv/config.
Open the script using the gedit editor and adjust the following settings according to the values determined before:

```
vendor_id=8086
device_id=100f
businfo=0000:02:02.0
origdriver=e1000
```

If more than one device shall be assigned, duplicate this script and adjust the entries:

```
cp pcidev1.sh pcidev2.sh
```

## 2.3.4 Configuration file

A template configuration file pcidev1.config is located in /hv/config.
Note: this configuration file has to be included in the main configuration file to become effective!
Open the script using the gedit editor and adjust the following settings according to the values determined before.

Replace the template device name PCI Device Name with an appropriate unique name (caveat: replace in ALL places!):

```
[Devices\Rtos\Intel Gigabit Controller1]
```

Adjust the bus information values (above example: 0000:02:02.0). The first number 0000 can be ignored, the following numbers are related to PCI Bus, Device and Function:

```
[Devices\Rtos\Intel Gigabit Controller1]
"PciBus"=dword:2
"PciDevice"=dword:2
"PciFunction"=dword:0
```

Adjust the device IO and memory resources (note: hex values are to be written in reverse order!!!):

```
[Devices\Rtos\Intel Gigabit Controller1\IoPort\1]
"Address"=hex:40,20,00,00,00,00,00,00
"Length"=dword:40
[Devices\Rtos\Intel Gigabit Controller1\Memory\1]
"Address"=hex:00,00,58,FD,00,00,00,00
"Length"=dword:20000
[Devices\Rtos\Intel Gigabit Controller1\Memory\2]
"Address"=hex:00,00,FE,FD,00,00,00,00
"Length"=dword:10000
```

Adjust the device interrupt resources.

PCIe card
In case it is a PCIe card, the MSI entries must be kept uncommented.
The Id value of the first PCI card to be assigned shall be 20, for the second card it shall be 21 and so on.
The destination value determines the CPU where the interrupts have to be delivered. The value is a bit mask where all bits related to the RTOS CPU have to be set.
Examples:

- Destination 2 = 00000010: CPU core 1 --> dual core system
- Destination 8 = 00001000: CPU core 3 --> quad core system
- Destination 80 = 10000000: CPU core 7 --> octal core system

The Vector value of the first PCI card to be assigned shall be F4, for the second card it shall be F5 and so on.
In the below example the values given are related to the first PCI device that is assigned to the RTOS on a quad core system:

```
; MSI
[Devices\Rtos\Intel Gigabit Controller1\Interrupt\1]
"Type"=dword:4
"Id"=dword:20
"DestinationFormat"=dword:0
"Destination"=dword:8
"Vector"=dword:F4
```

In the below example the values given are related to a second PCI device that is assigned to the RTOS on a quad core system:

```
; MSI
[Devices\Rtos\Intel Gigabit Controller2\Interrupt\1]
"Type"=dword:4
"Id"=dword:21
"DestinationFormat"=dword:0
"Destination"=dword:8
"Vector"=dword:F5
```

PCI card
In case it is a PCI card, the MSI entries must be commented out and the Legacy Interrupt entries must be uncommented.
The Id, Destination and Vector values have to be determined in the same way as for PCIe devices.
The IoApicIntIn value can be determined using the file /hv/config/hwbase.config. Search for the entry [IoApics\##]. The number ## is used for the IoApicIntIn value.
Examples:

- ## == 01: IoApicId=dword:1
- ## == 02: IoApicId=dword:2

The value for the IoApicIntIn can be determined by the resource information gotten via the lshw command described before (in the example above it is irq 16). The value in the configuration file is hexadecimal!
In the below example the values given are related to the first PCI device that is assigned to the RTOS on a dual core system where the number found for the IoApicId is 02:

```
; Legacy
[Devices\Rtos\Intel Gigabit Controller1\Interrupt\1]
"Type"=dword:1
"Id"=dword:20
"IoApicId"=dword:2
"IoApicIntIn"=dword:10
"DestinationFormat"=dword:0
"Destination"=dword:2
"Vector"=dword:F4
```

Note: do not forget to include this configuration file in the main configuration file to become effective!

### 2.3.5 Adjust main configuration file

After creating the device configuration files (pcidev1.config etc.), they need to be included in the main configuration file.
In case of VxWorks, it is the file /hv/vx/vxworks.config
In case of On Time RTOS-32, it is /hv/rtos-32/rtos-32.config
In case of Real-time Linux, the main configuration file is /hv/lx/linux.config
Uncomment or add the respective pcidev#.config entries.
The example below shows, how two PCI device configuration files are included:

```
;--------------------------------------

; includes

;--------------------------------------

#include "../config/membase.config"

#include "../config/memory.config"

#include "../config/cpu.config"

#include "../config/hwbase.config"

#include "../config/hwdevbase.config"

#include "../config/vmf.config"

#include "../config/debug.config"

#include "../config/windowsvm.config"

#include "../config/pcidev1.config"

#include "../config/pcidev2.config"
```

### 2.3.6 Adjust system startup script /etc/rc.local

The device assignment scripts pcidev#.sh usually has to be executed automatically on system startup.
To accomplish this, uncomment or add the respective pcidev#.sh calls in the file /etc/rc.local.
The example below shows, how three PCI devices are assigned.

```
#!/bin/bash

source /hv/config/pcidev1.sh add

source /hv/config/pcidev2.sh add

source /hv/config/pcidev3.sh add
```

# 2.4 Device assignment to Windows VM

## *2.4.1 Why passthrough*

The typical “use-case” of this feature is running a latency-sensitve application in a acontisHypervisor container with a hard realtime OS (such as Rt Linux, VxWorks, RTOS-32) and a separate user-friendly GUI application in a Windows VM (KVM) that heavily uses hardware graphics accelaration to drive high-qualtity output. Both applications (real-time and GUI) communicate to each other via a well-defined Rtos Library interface. In this Scenario the Windows machine can safely install security updates and perform reboot without affecting the critical real-time part.

It is madatory to have hardware support for IOMMU (VT-d). VT-d should be supported by your Processor, your motherboard and should be enabled in the BIOS.

Virtual Function I/O (VFIO) allows a virtual machine to access a PCI device, such as a GPU or Network Card, directly and achieve close to bare metal performance.

The setup used for this guide is:

- Intel Core I5-8400 or I3-7100 (with integrated Intel UHD 630 Graphics) – this integrated graphics adapter will be assigned to the Windows VM.
- AMD/ATI RV610 (Radeon HD 2400 PRO) – optional, as a second GPU, only needed to have display output for Linux Host. Later, the host is reached only via SSH.
- Intel I210 Gigabit Network Card – optional, to demonstrate how to pass through a simple PCI device to a Windows VM

## *2.4.2 Ethernet PCI Card/Custom PCI device assignment*

Some manual work is required to pass through the PCI Device to a Windows VM.

### *2.4.2.1 Understanding IOMMU Groups*

In order to activate the hardware passthrough we have to prevent the ownership of a PCI device by its native driver and assign it to the vfio-pci driver instead.

In a first step, an overview of the hardware and related drivers is required. In the below list 2 devices are marked with red color, they should be used for passthrough to the guest VM..

```
rte@rte-System-Product-Name:~$ lspci
00:00.0 Host bridge: Intel Corporation 8th Gen Core Processor Host Bridge/DRAM Registers
00:01.0 PCI bridge: Intel Corporation Xeon E3-1200 v5/E3-1500 v5/6th Gen Core Processor
00:02.0 VGA compatible controller: Intel Corporation Device 3e92
00:14.0 USB controller: Intel Corporation Cannon Lake PCH USB 3.1 xHCI Host Controller
```

```
00:14.2 RAM memory: Intel Corporation Cannon Lake PCH Shared SRAM (rev 10)
00:16.0 Communication controller: Intel Corporation Cannon Lake PCH HECI Controller
00:17.0 SATA controller: Intel Corporation Cannon Lake PCH SATA AHCI Controller (rev 10)
00:1c.0 PCI bridge: Intel Corporation Device a33c (rev f0)
00:1c.5 PCI bridge: Intel Corporation Device a33d (rev f0)
00:1c.7 PCI bridge: Intel Corporation Device a33f (rev f0)
00:1f.0 ISA bridge: Intel Corporation Device a303 (rev 10)
00:1f.4 SMBus: Intel Corporation Cannon Lake PCH SMBus Controller (rev 10)
00:1f.5 Serial bus controller [0c80]: Intel Corporation Cannon Lake PCH SPI Controller
01:00.0 VGA compatible controller: [AMD/ATI] RV610 [Radeon HD 2400 PRO]
01:00.1 Audio device: [AMD/ATI] RV610 HDMI Audio [Radeon HD 2400 PRO]
03:00.0 Ethernet controller: Intel Corporation I210 Gigabit Network Connection (rev 03)
04:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express
Gigabit Ethernet Controller (rev 15)
```

Make sure, IOMMU is activated on your Linux host:

**dmesg | grep -e "Directed I/O"**
**DMAR: Intel(R) Virtualization Technology for Directed I/O**

When IOMMU is activated, all devices are divided into groups. The IOMMU Group is an indivisible unit. All devices in the same group must be passed thrrough together.

```
for a in /sys/kernel/iommu_groups/*; do find $a -type l; done | sort
--version-sort

/sys/kernel/iommu_groups/0/devices/0000:00:00.0
/sys/kernel/iommu_groups/1/devices/0000:00:01.0
/sys/kernel/iommu_groups/1/devices/0000:01:00.0
/sys/kernel/iommu_groups/1/devices/0000:01:00.1
/sys/kernel/iommu_groups/2/devices/0000:00:02.0
/sys/kernel/iommu_groups/3/devices/0000:00:14.0
/sys/kernel/iommu_groups/3/devices/0000:00:14.2
/sys/kernel/iommu_groups/4/devices/0000:00:16.0
/sys/kernel/iommu_groups/5/devices/0000:00:17.0
/sys/kernel/iommu_groups/6/devices/0000:00:1c.0
/sys/kernel/iommu_groups/7/devices/0000:00:1c.5
/sys/kernel/iommu_groups/8/devices/0000:00:1c.7
/sys/kernel/iommu_groups/9/devices/0000:00:1f.0
/sys/kernel/iommu_groups/9/devices/0000:00:1f.4
/sys/kernel/iommu_groups/9/devices/0000:00:1f.5
/sys/kernel/iommu_groups/10/devices/0000:03:00.0
/sys/kernel/iommu_groups/11/devices/0000:04:00.0
```

It is important to know – all devices in a single group should be shared together. In the above case the group 10 has the PCI Card, and no other devices are in this group.

We need to determine more details about the hardware we are interested in:

rte@rte-System-Product-Name:~$ lspci -s 03:00.0 –vvn
03:00.0 0200: 8086:1533 (rev 03)
…
Kernel driver in use: igb
Kernel modules: **igb**

Add to the linux kernel command line parameters the following: **vfio-pci.ids=8086:1533**. To edit kernel parameters edit **/etc/grub.d/40_custom** file and then execute **update-grub** and reboot.

Normally no other steps are needed to assign a device to a vfio-pci driver instead of a its native driver. If you have conflicts between these two drivers, it would be required to disable the loading of a native driver. Add a parameter **module_blacklist=igb** to a kernel command line.

#### *2.4.2.2 IOMMU Interrupt Remapping and RTOS*

As mentioned before, device passthrough requires IOMMU support by the hardware and the OS. It is needed to add “intel_iommu=on iommu=pt” to your kernel command line. These parameters are automatically added by Hypervisor when executing autoconf.sh script.

But PCI devices can be assigned not only to Windows VM but also to a RTOS VM, for ex. RTOS-32. Usually, network cards can be used by a realtime application, for ex. to communicate with EtherCAT slaves.

If you want to use Windows VM with devices passed through and a RTOS application with a assigned PCI device, it is required to deactivate IOMMU Interrupt Remapping in Linux Kernel.

The following kernel command line parameters should be manually added to the GRUB Entry “Hypervisor” by editing **/etc/grub.d/40_custom** file and then executing **update-grub**.

intremap=off vfio_iommu_type1.allow_unsafe_interrupts=1

#### *2.4.2.3 Config.sh*

Last step is to edit **/hv/VMs/VM1/config.sh** file and add the PCI Etherner Card information here.

Uncomment #export OTHER_HW variable and set it to:

```
export OTHER_HW=” -device vfio-pci,host=03:00.0”
```

## *2.4.3 Intel Integrated Graphics (iGVT-d) assignment*

To use graphics passthrough, less steps compared to standard PCI hardware passthrough are required, because Hypervisor automates most of the steps.

### *2.4.3.1 Creating a Windows VM*

Before we learn how to assign an Integrated Intel Graphics to a Windows VM, we should create this VM.

It is important to create this machine with OVMF UEFI, because we want to use it for graphics passthrow. If you have already created the VM using legacy Seabios, it will not work for this example.

Download a Windows 10 .iso file or create it using Media Creation Tool, which is available on a Microsoft site.

Create a hdd image:

```
qemu-img create -f qcow2 win-vm1.qcow2 50G
```

Now create a script vm-ovmf.sh:

```
#!/bin/bash

qemu-system-x86_64 -enable-kvm \
  -machine pc,accel=kvm,igd-passthru=on -m 8g \
  -enable-kvm \
  -nodefaults \
  -cpu host,hv_relaxed,hv_vapic \
  -smp cpus=3,cores=3,threads=1,sockets=1 \
  -drive if=pflash,format=raw,readonly,file=/hv/bin/OVMF_CODE.fd \
  -drive if=pflash,format=raw,file=/hv/bin/OVMF_VARS.fd \
  -drive file=/hv/VMs/vm1/Windows10.iso,index=0,media=cdrom \
  -drive file=/hv/VMs/virtio/virtio-win-0.1.171-
acontis.iso,index=1,media=cdrom  \
  -drive file=/hv/VMs/vm1/win-vm1.qcow2,index=1,media=disk,if=virtio
\
  -vga std
```

Launch this script and start VM and perform full Windows 10 installation.

Then configure Windows for Remote Desktop access, because next step is deactivating standard vga graphics. Then shutdown VM. You have to remember the IP address or computer name of this VM to later connect via RDP.

You do not need this script anymore.

Edit vmconfig.sh in /hv/VMs/vm1 and uncomment and change a variable:

```
export vmimg=win-vm1.qcow2
```

This drive image with installed Windows 10 will be required later.

#### 2.4.3.2 Understanding GPU modes UPT and Legacy

There are two modes „legacy“ and “Universal Passthrough” (UPT).

Hypervisor uses only Legacy mode, but it could be important to understand the difference.

UPT is available for Broadwell and newer processors. Legacy mode is available since SandyBridge. If you are unsure, which processor you have, please check this link https://en.wikipedia.org/wiki/List_of_Intel_CPU_microarchitectures

In Legacy it is meant that IGD is a primary and exclusive graphics in VM. Additionally the IGD address in the VM must be PCI 00:02.0, only 440FX chipset model (in VM) is supported and not Q35. The IGD must be the primary GPU for host as well (please check your BIOS settings).

In UPT mode the IGD can have another PCI adress in VM and the VM can have a second graphics adapter (for example qxl, or vga).

Please read here more about legacy and UPT mode:
https://git.qemu.org/?p=qemu.git;a=blob;f=docs/igd-assign.txt

There a lot of other little things, why IGD Passthrough could not work. For ex. In legacy mode it expects a ISA/LPC Bridge at PCI Adress 00:1f.0 in VM and this is a reason, why Q35 chip does not work, because it has another device at this adress.

In UPT mode, there is no output support of any kind. So the UHD graphics can be used for accelerating (for ex. Decoding) but the Monitor remains black and there is a non-standard experimental qemu vfi-pci command line parameter x-igd-opregion=on, which can work.

#### 2.4.3.3 Blacklisting i915 driver on Host

The standard Intel Driver i915 is complex and it is not always possible to safely unbind the device from this driver, that is why this driver is blacklisted by Hypervisor when executing autoconf.sh script.

#### 2.4.3.4 Deactivating Vesa/EFI Framebuffer on Host

Please also know, when i915 driver is disabled, there are other drivers which are ready to jump on the device to keep the console working. Depend on your BIOS settings (legacy or UEFI) two other drivers can occupy a region of a video memory: efifb or vesafb.

Hypervisor blacklists both by adding the following command line parameter:
**video=vesafb:off,efifb:off**.

Please also check if it works: **cat /proc/iomem**. If you still see that one of this driver still occupies a part of a video memory, please try manually another combination:
video=efifb:off,vesafb:off.

#### 2.4.3.5 Legacy BIOS, pure UEFI and CSM+UEFI in Host

It plays also significant role, in which mode your machine is booted: Legacy BIOS, pure UEFI or UEFI with CSM support. In pure UEFI (on host) QEMU cannot read video ROM. In this case you could extract it manually (for ex. Using Cpu-Z utility or just boot in CSM mode, when iGPU is a primary GPU in BIOS), patch it with correct device id and provide it to qemu as *romfile*= parameter for vfi-pci. Please google for *rom-parser* and *rom-fixer* for details.

#### 2.4.3.6 SeaBIOS and OVMF (UEFI) in VM

It also plays role which BIOS you use in the VM itself. For QEMU there are two possibilities: SeaBIOS (legacy BIOS, which is default for qemu) and OVMF (UEFI). You can download and build OVMF itself, but easier to install precompiled binaries from here:
https://www.kraxel.org/repos/.

For your convenience, Hypervisor installs precompiled OVMF binaries to /hv/bin directory:
**OVMF_CODE.fd**
**OVMF_VARS.fd**

By default, OVMF UEFI does not support OpRegion Intel feature, which is required to have a graphics output to a real display. There are three possibilities how to solve this problem and the easiest one seems to be the using special vbios rom **vbios_gvt_uefi.rom**, please read more here https://wiki.archlinux.org/index.php/Intel_GVT-g.

For your convenience, we have already included this file into the Hypervisor package and it can also be located in **/hv/bin** directory.

Hypervisor uses OVMF (UEFI) for graphics pass-through.

#### 2.4.3.7 How to do it in Hypervisor

The final working configuration which we consider here:

- CPU Graphics is a primary GPU in BIOS (host)

- Host boots in pure UEFI mode
- OVMF is used as BIOS in Windows VM
- vbios_gvt_uefi.rom is used as VBIOS in romfile parameter for vfio-pci
- Legacy mode for IGD, so the Windows VM has only one graphics card Intel UHD 630

Which commands should be executed in Hypervisor to do a pass-through of a Intel Integrated Graphics to a Windows VM?

None! Almost everything is done automatically. When executing autoconf.sh script (which is required to install real-time linux kernel and to reserve kernel memory for hypervisor needs), a separate GRUB entry “Hypervisor + iGVT-d” is created.

This entry contains already all necessary linux kernel parameters required to do a graphics pass-through: blacklisting intel driver, disabling interrupt remapping, assigning a VGA device to a vfio-pci driver and other stepts.

But one step should be done once, configuring your VM.

Change **/hv/VMs/VM1/vmconfig.sh** script. Two variables should be uncommented and activated:

```
export uefi_bios=1
export enable_vga_gpt=1
```

It should be enough.

Just reboot your machine, choose “Hypervisor+iGVT-d” menu item. If everything is correct, the display of your Host should remain black. Connect to the machine using SSH connection or use a second graphics card for Host (read next chapter) and then execute /hv/VMs/vm1/vmrun.sh

Wait 30-60 seconds and.. display remains black? Of course. Windows does not have Intel Graphics drivers .

Remember we configured Windows for a Remote Desktop Access in previous steps? Connect to Windows VM via RDP and install latest Intel Drivers https://downloadcenter.intel.com/product/80939/Graphics

If everyting is done correctly, your display should now work and display a Windows 10 Desktop.

#### *2.4.3.8 Using Second GPU Card for Host*

X-Windows on the hypervisor host does not work properly, when the primary GPU in the System is occupied by the vfi-pci driver. It detects the first GPU, tries to acquire it, fails and then aborts. We should let it know, that it should use our second GPU card instead.

Log in to the hypervisor host (Press Ctrl-Alt-F3, for example), shutdown LightDM manager sudo service lightdm stop.

Execute sudo X –configure, it creates xorg.conf.new file in the current directory. When this command is executed, X server enumerates all hardware and creates this file . By default, in modern systems, Xserver does not need the xorg.conf file, because all hardware is detected quite good and automatically. But the config file is still supported.

Look at this file, find a section “Device” with your second GPU (look at the PCI Adress). Copy content of this section to a separate file /etc/X11/xorg.conf.d/secondary-gpu.conf. Save and reboot.

```
Section "Device"

    ### Available Driver options are:-
    ### Values: <i>: integer, <f>: float, <bool>: "True"/"False",
    ### <string>: "String", <freq>: "<f> Hz/kHz/MHz",
    ### <percent>: "<f>%"
    ### [arg]: arg optional
    #Option     "Accel"                # [<bool>]
    #Option     "SWcursor"             # [<bool>]
    #Option     "EnablePageFlip"       # [<bool>]
    #Option     "SubPixelOrder"        # [<str>]
    #Option     "ZaphodHeads"          # <str>
    #Option     "AccelMethod"          # <str>
    #Option     "DRI3"                 # [<bool>]
    #Option     "DRI"                  # <i>
    #Option     "ShadowPrimary"        # [<bool>]
    #Option     "TearFree"             # [<bool>]
    #Option     "DeleteUnusedDP12Displays"  # [<bool>]
    #Option     "VariableRefresh"      # [<bool>]
    Identifier  "Card0"
    Driver      "amdgpu"
    BusID       "PCI:1:0:0"
EndSection
```

## *2.4.4 Keyboard and Mouse assignment*

Normally your Linux Host with acontisHypervisor with Windows VM and Integrated Graphics passed throw works in head-less mode. Windows VM outputs to a Monitor durch DVI-D/HDMI connection and the Linux Host is controlled via SSH connection. Windows has a look and feel as it works without an intermediate hypervisor layer.

So, Windows needs a keyboard and mouse.

Go **to /dev/input/by-id/** and find something that looks like a keyboard and mouse and it should containt “-event-“ in its name.

```
ls -la

usb-18f8_USB_OPTICAL_MOUSE-event-if01 -> ../event5
usb-18f8_USB_OPTICAL_MOUSE-event-mouse -> ../event3
usb-18f8_USB_OPTICAL_MOUSE-if01-event-kbd -> ../event4
usb-18f8_USB_OPTICAL_MOUSE-mouse -> ../mouse0
usb-SEM_USB_Keyboard-event-if01 -> ../event7
usb-SEM_USB_Keyboard-event-kbd -> ../event6
```

configure your VM with these parameters by editing vmconfig.sh:

```
export vga_gpt_kbd_event=6
export vga_gpt_mouse_event=3
```

Please also note, disconnecting evdev devices, such as keyboard or mouse, can be problematic when using qemu and libvirt, because it does not reopen device when the device reconnects.

If you need to disconnect/reconnect your keyboard or mouse, there is a workaround, create a udev proxy device and use its event device instead. Please read more hier https://github.com/aiberia/persistent-evdev.

If everything works, you’ll find new devices like uinput-persist-keyboard0 pointing to /dev/input/eventXXX use use these ids as usual in:

```
export vga_gpt_kbd_event=XXX
export vga_gpt_mouse_event=ZZZ
```

## 2.5 Bridge virtual and physical network

If the virtual machine shall be accessed via TCP/IP over the virtual network from an external system, the virtual network and the respective physical network has to be bridged.
In the folder /hv/hvctl you can find the template configuration file brvnetconfig.sh. Enter gedit brvnetconfig.sh and adjust all the parameters according to the description in that file.
Note, the IP address of the virtual network inside the RTOS guest need to be adjusted appropriately.

- Create the bridge
  First, start the RTOS to assure the virtual network is available.
  Change into the guest directory (/hv/hvctl) and execute the brvnetset.sh script.

- Remove the bridge
  Change into the guest directory (/hv/hvctl) and execute the brvnetclr.sh script.

# 2.6 Windows or Linux Guest basic settings

Before installing Windows or Linux guests, some configuration settings have to be performed. The guests will get access to the network via the hypervisor network.

## *2.6.1 Guest configuration*

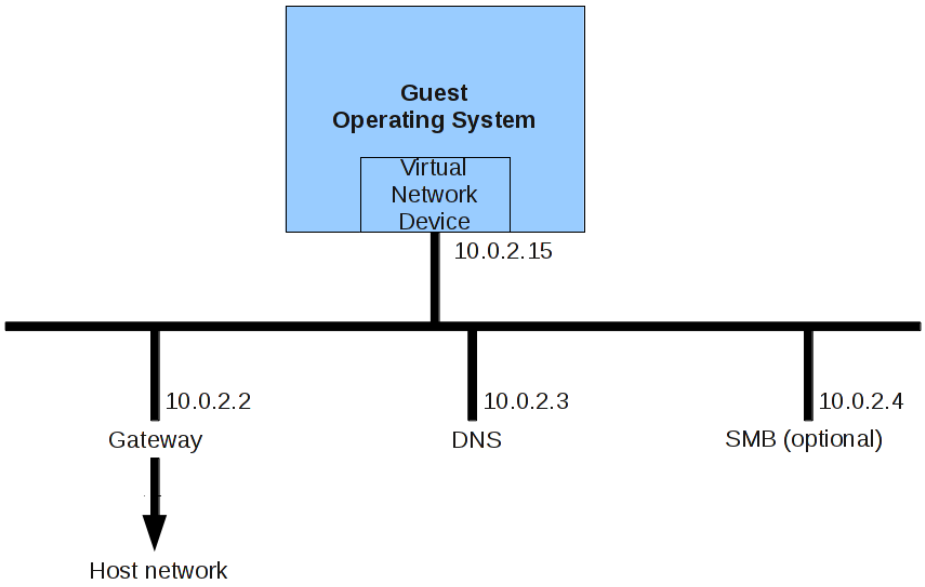
In the folder /hv/VMs/vm1 you can find the template configuration file vmconfig.sh. Enter gedit vmconfig.sh and adjust all the parameters according to the description in that file.
Please note, when installing the guest, the number of CPUs for the guest has to be set to one below the number of physical available CPUs, otherwise performance while installing significantly slows down!
If you want to use an ISO file as installation media, the setting for cdrom_iso in vmconfig.sh has to be adjusted appropriately.
For example, if the ISO file is stored on an USB stick, the following entry will use the ISO file as installation media:

cdrom_iso=/media/MyUserName/UsbStickName/IsoFileName.iso
where MyUserName is the user you are logged in, UsbStickName is the name of the USB stick and IsoFileName is the filename of the ISO file.

## *2.6.2 Guest Ethernet*

In the vmconfig.sh script, two guest Ethernet controllers can be configured.
One is used for directly connecting to the external physical network via network bridging.
To enable this network, set external_nw to a value of 1.

The second is used as a private network, which is connected to the external physical network via NAT. This connection is safer, but much slower. It is also used for file sharing with the hypervisor host. The architecture of the private network looks as follows:




### *2.6.2.1 Guest Ethernet MAC address*

In the folder /hv/VMs/vm1 the script vm1_setmac.sh defines the Ethernet MAC address for the virtual network adapters inside the guest. By default, if this file does not exist, this script file is automatically generated using random local administered addresses. You will have to change these addresses by an official address related to your company.
The content of this file looks as follows:

```
export ethmacVM1=0A:C0:FD:20:39:01
```

```
export ethmacVM2=0A:C1:FD:20:39:01
```

The ethmacVM1 address belongs to the bridged Ethernet controller (bridged to the external network).
The ethmacVM2 address belongs to the private (internal) Ethernet controller (using NAT).

## *2.6.3 Access to the hypervisor host filesystem*

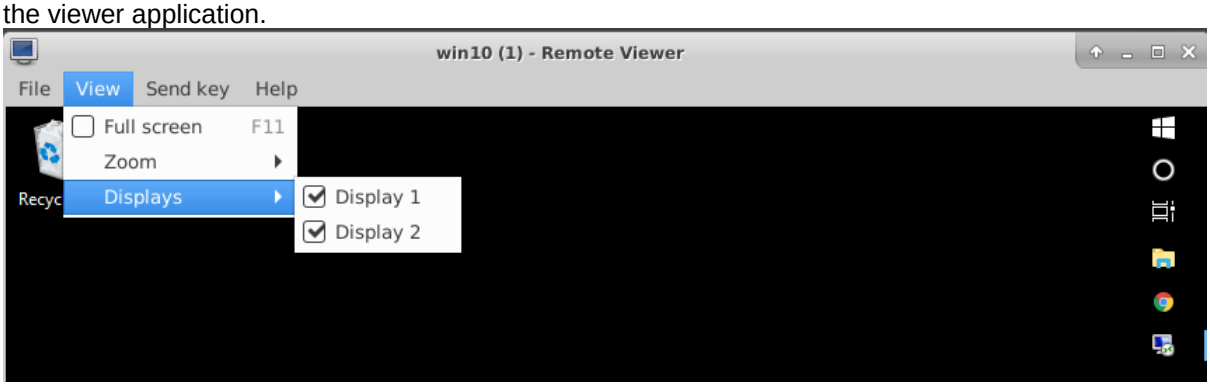
Access to the hypervisor host filesystem (/hv/VMs/share) is available using a CIFS network share.
The private network is used for this purpose. The IP address of the network share server is 10.0.2.4.
Windows: access via \\10.0.2.4\qemu
Linux: use SAMBA to mount the network share at 10.0.2.4 with the name qemu

## *2.6.4 Guest Multiple monitors*

If more than one monitor is connected to the system, these can also be used for VM guests (up to a maximum of 4 monitors).
For Windows guests, the following settings are required in vmconfig.sh:

```
export windows_guest=1
export num_monitors=# (where # is the number of monitors)
```

To enable additional monitors being displayed, select the displays via the View – Displays menu in the viewer application.

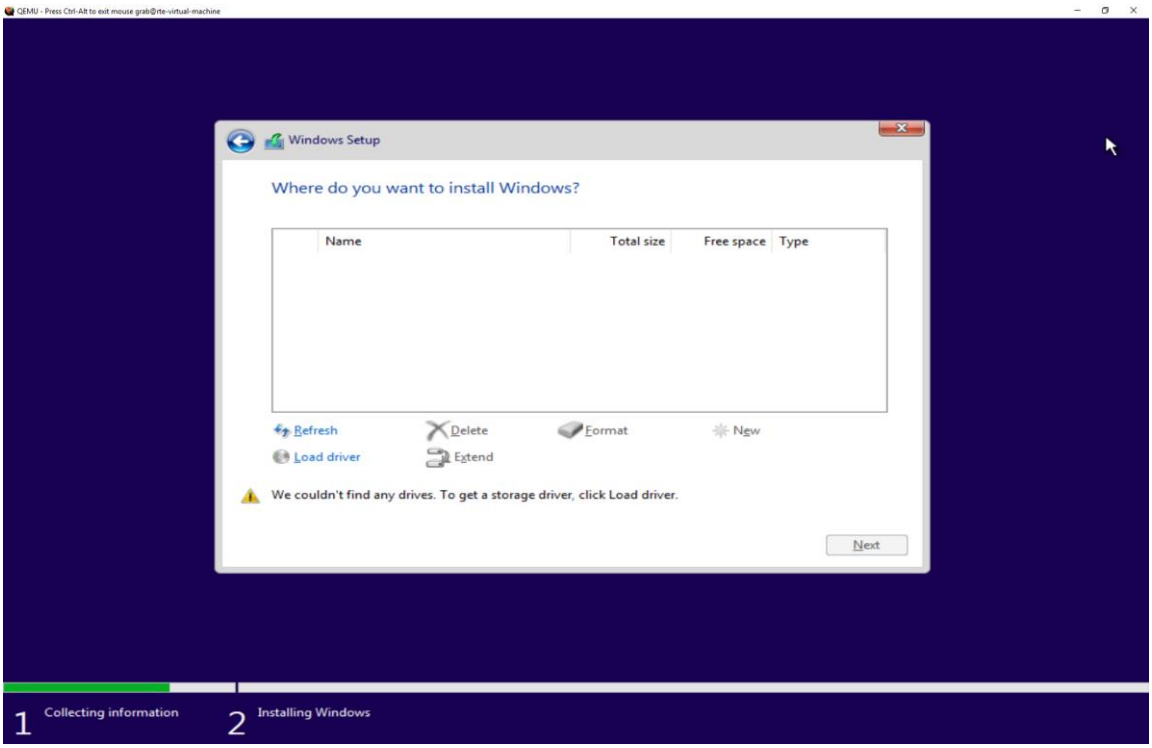


## 2.7 Windows installation

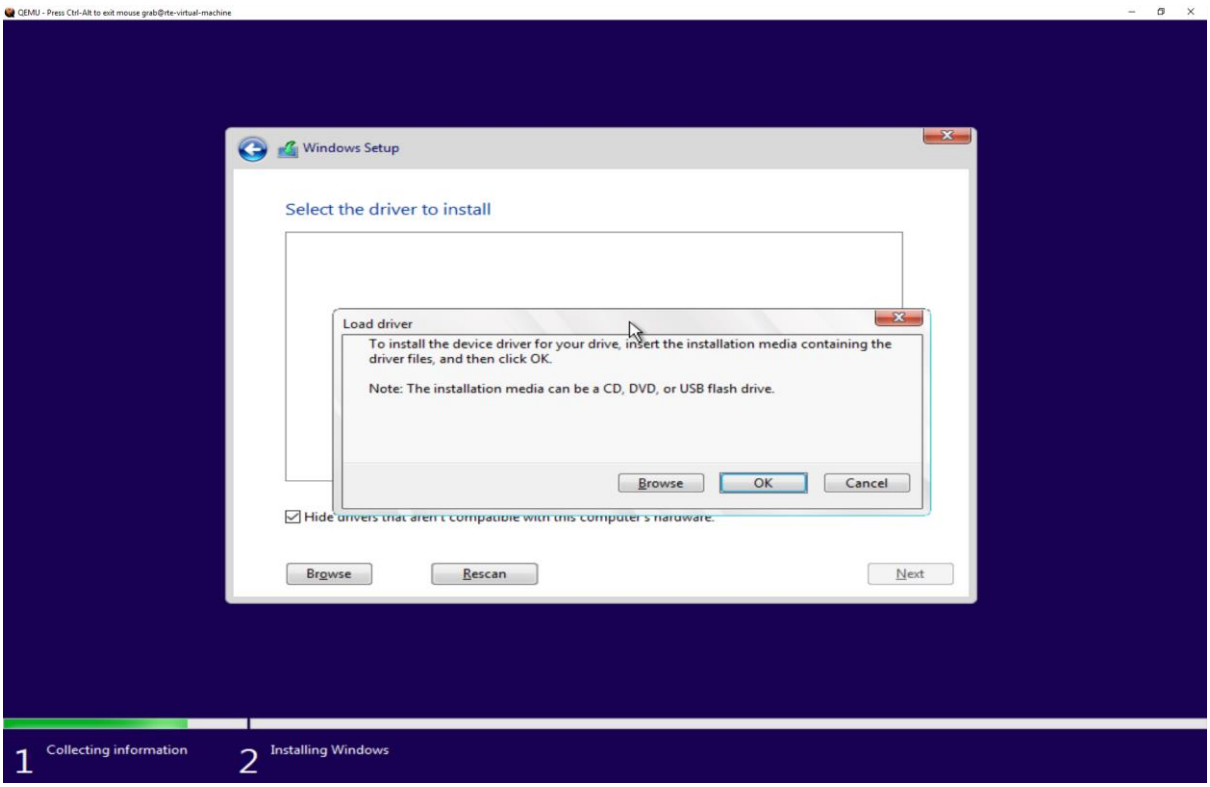
A fresh Windows installation can be accomplished by traditional methods using a bootable DVD or USB device. Alternatively, an ISO CD image can be used (see section 2.6.1).
After configuring the VM according to section 2.6.1 you must boot the VM. Execute the vmrun.sh script with option -nospice to accomplish this:

```
./vmrun.sh -nospice
```

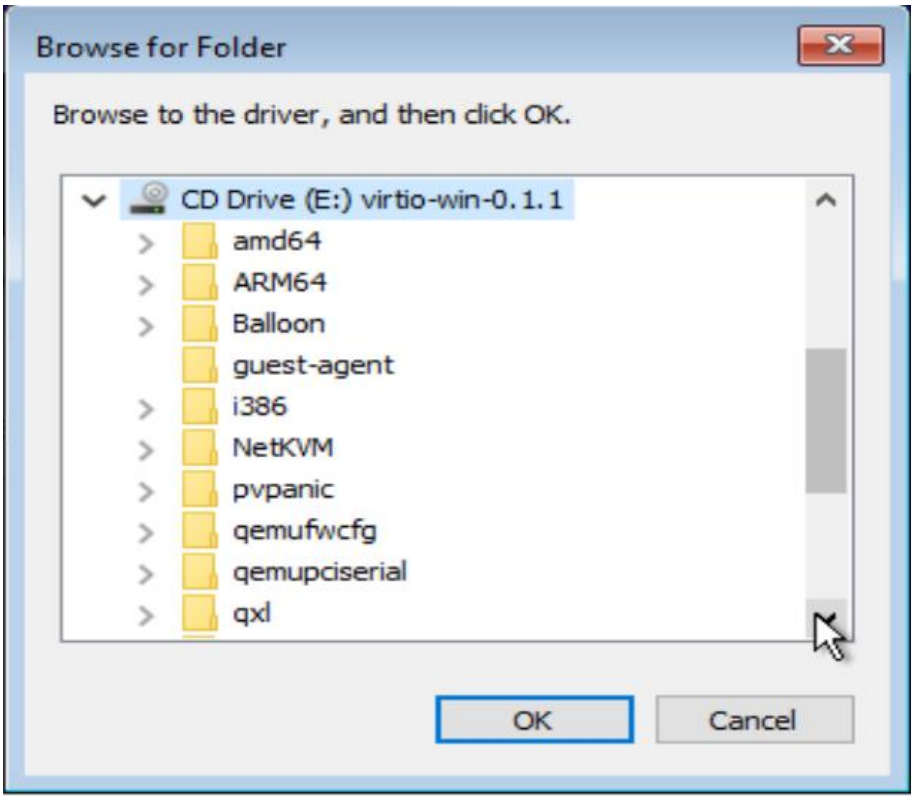
In case starting the installation process fails with the message “Could not access KVM kernel module…”, you did not enable Intel or AMD virtualization technology in the BIOS.
Please note, setting up Windows may take significanlty longer than used with real physical machines. Later on, after installation of para-virtualized drivers, Windows will become significantly faster, very close to native performance.
Before the setup can access the hard disk, the respective driver has to be installed.
Please follow the screenshots below:



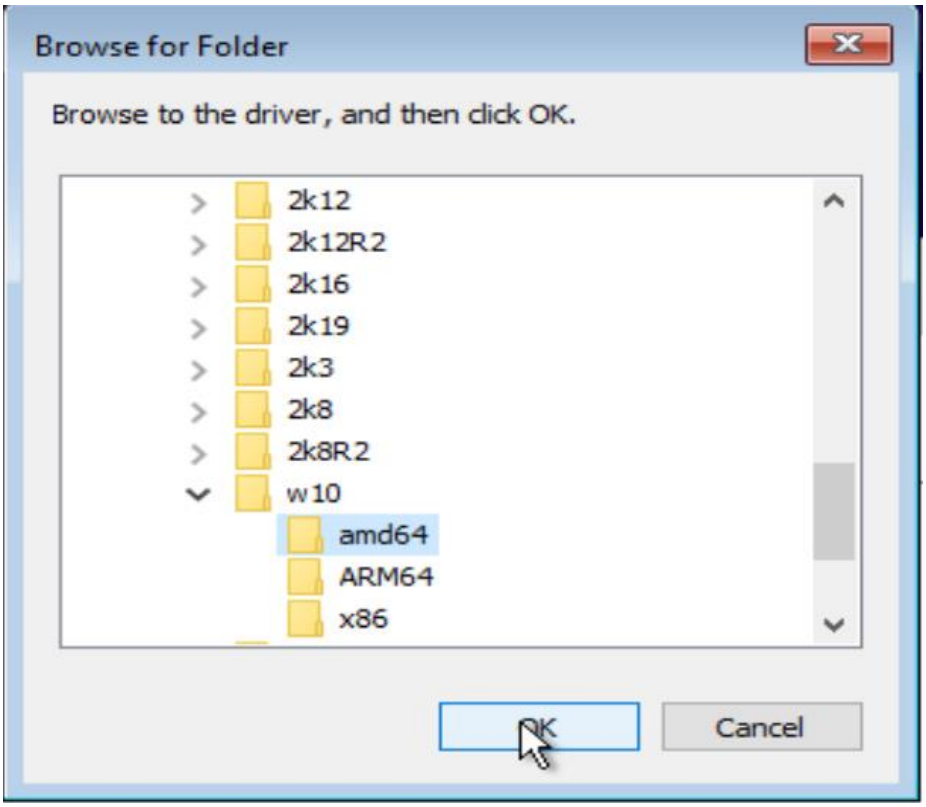

Select “Load driver”



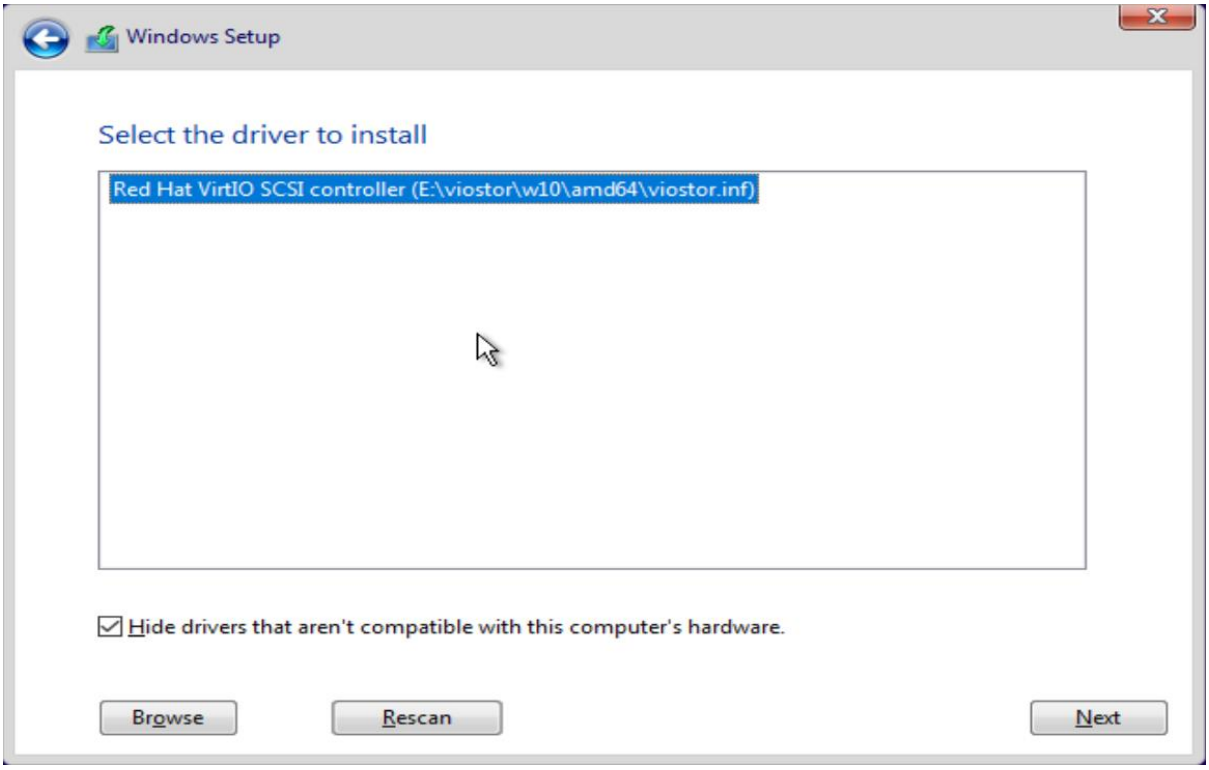

Select “Browse”




Select the CD Drive E:



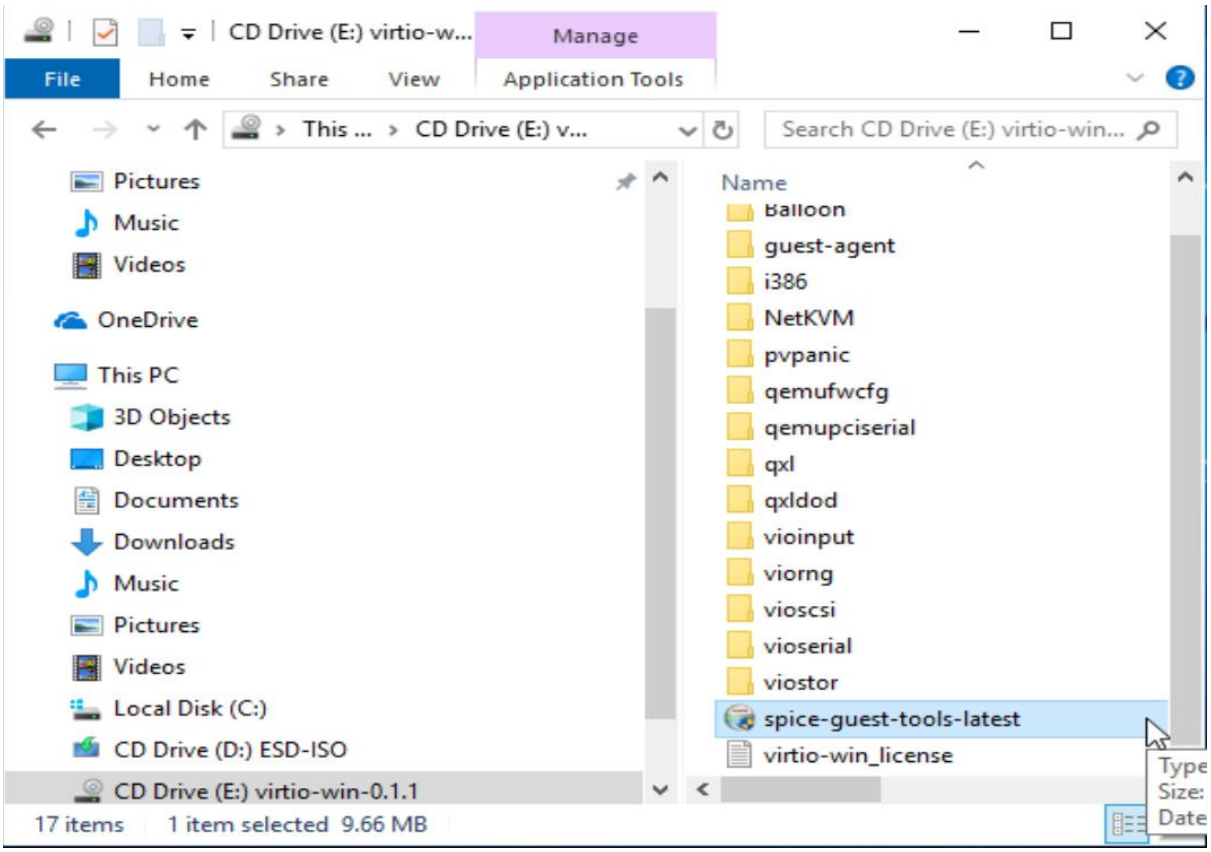

Select the appropriate directory, for Windows 10 64 Bit, it is amd64




Press Next and continue setup as usual.
Important note: skip the network setup (in this stage, there is not network driver available)

After successfully installing Windows, additional drivers need to be installed.
Follow the instructions below.

Open the Windows Explorer and install the spice guest tools.
It is located in CD Drive (E:) virtio-win-0.1.1: spice-guest-tools-latest. Select and start to install these tools. In case a message box appears to confirm driver installation, please confirm install the driver(s).

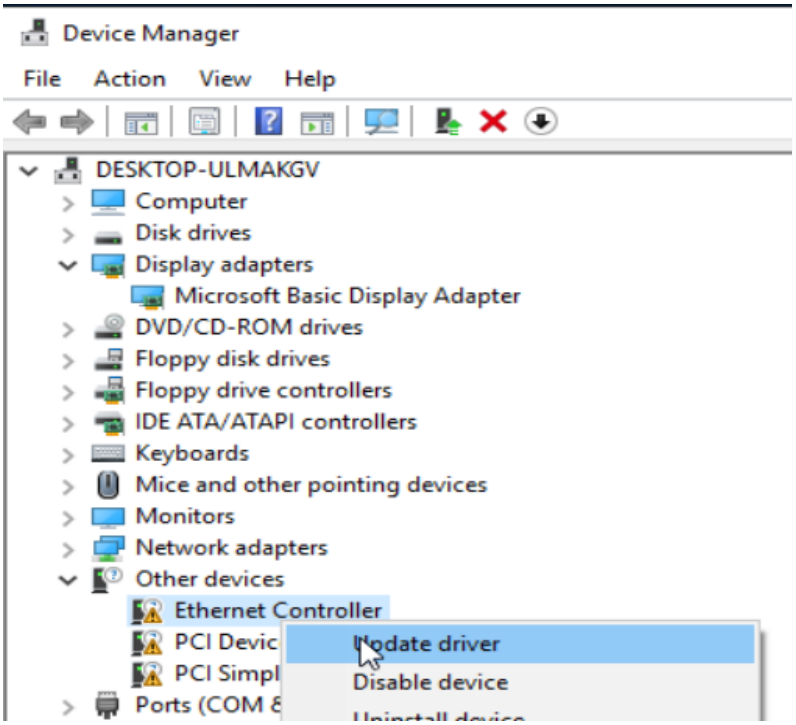



After the setup finished, you may not be able to use the mouse anymore. This is related to a new driver installed which is not supported by the viewer application.
In any case, close the setup message box (type enter, while the focus is on the finish box).
You need to shut down the guest, if not possible inside the guest, please use the ./vmshtdn.sh script.
If shutdown is blocked, you need to close the setup windows before (e.g. using tab keys to select the finish button).
After shutdown of the guest, please start again using the script ./vmrun.sh without any additional parameter:

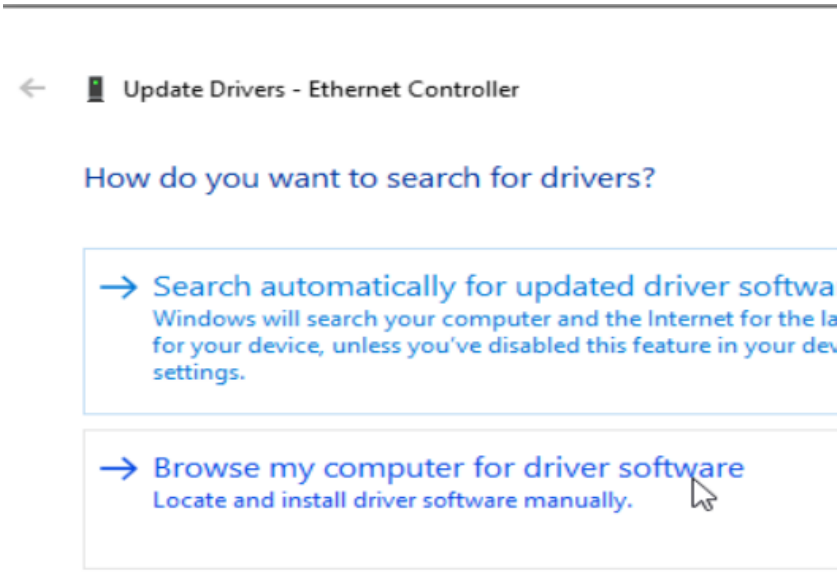
./vmrun.sh
Due to hardware changes, Windows may automatically reboot once.
Mouse and desktop may still not work properly. In this case, please install all the latest Windows updates.

## 2.7.1 Missing Drivers?

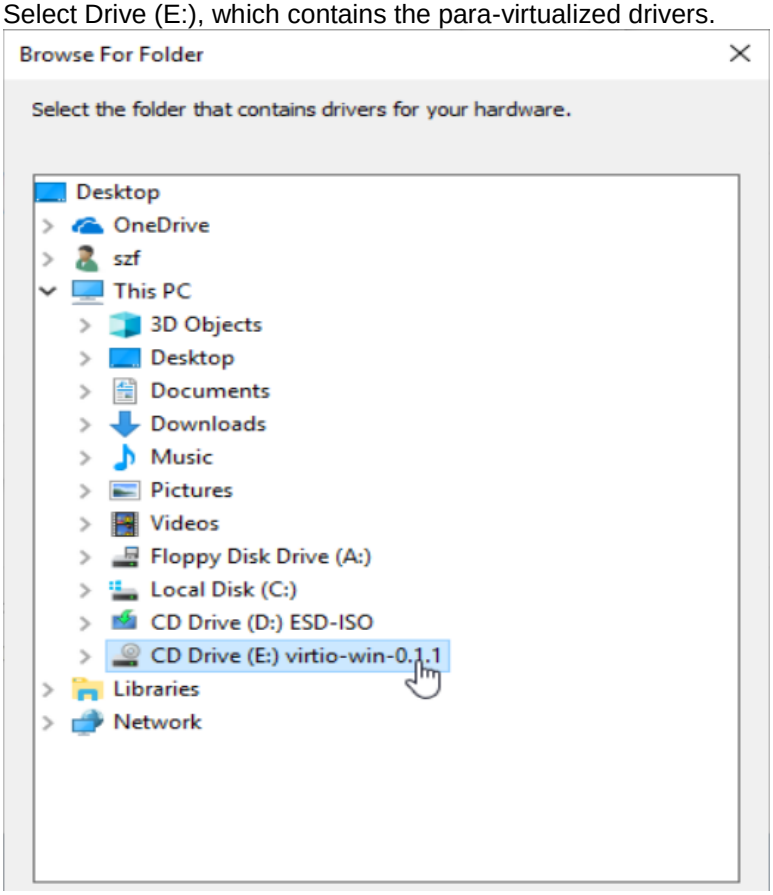
Check for missing drivers:
Open the Windows Device Manager and look for missing drivers. If any drivers are missing, follow the instructions below.
Select the driver using the right mouse key and press “Update driver”.



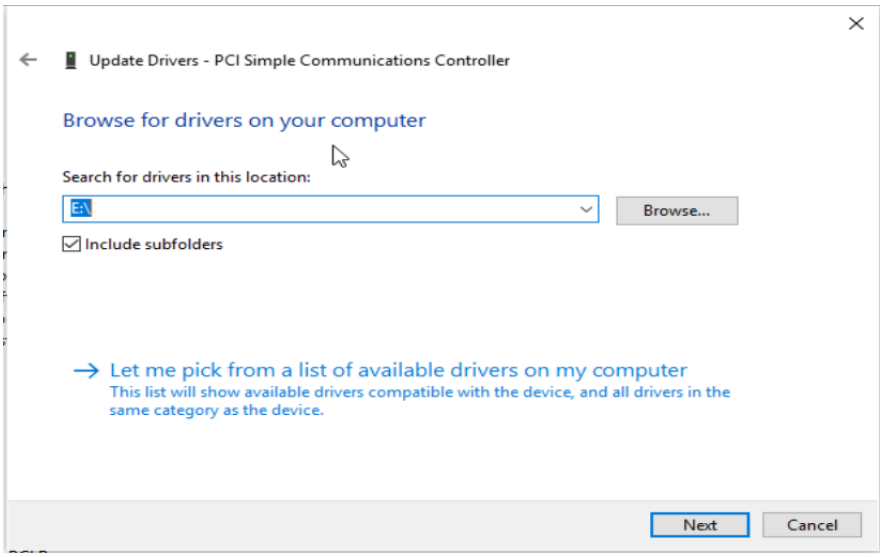

Select “Browse my computer…”



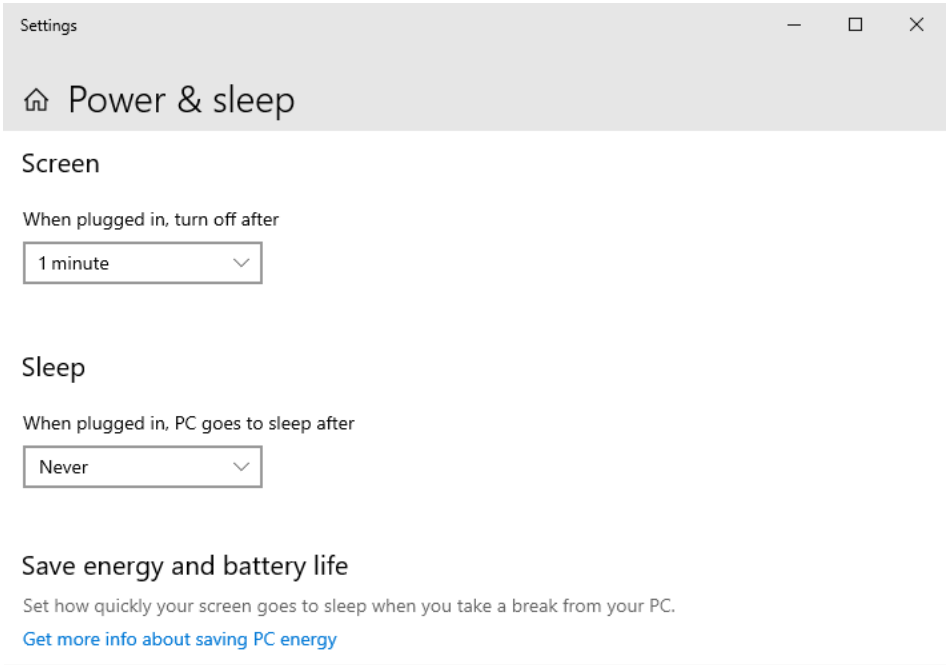
Select Drive (E:), which contains the para-virtualized drivers.




Assure “Include subfolders” is selected and press “Next” to install the driver.



Sleep and Hibernate are currently not supported (S3 or S4 mode).
Disable Sleep (Windows Settings – System – Power and sleep):



# 2.8 Windows or Linux Guest Operation

## *2.8.1 General Guest control*

Boot
Change into the guest directory (/hv/VMs/vm1) and execute the vmrun.sh script. When executed the very first time, the virtual hard disk image is created (e.g. file vm1.qcow2).

- Shutdown
  Change into the guest directory (/hv/VMs/vm1) and execute the vmshtdn.sh script.

- Reset
  Change into the guest directory (/hv/VMs/vm1) and execute the vmrst.sh script.

- Destroy the guest (in case it crashed)
  Change into the guest directory (/hv/VMs/vm1) and execute the vmkill.sh script.

- View guest output (in case the output Windows crashed or was closed)
  Change into the guest directory (/hv/VMs/vm1) and execute the vmview.sh script.

- Other control and monitoring commands
  Change into the guest directory (/hv/VMs/vm1) and execute the vmmon.sh script.
  Monitor commands are described in here: https://en.wikibooks.org/wiki/QEMU/Monitor

## *2.8.2 USB guest access*

If an USB device shall be used in the guest, the respective settings in vmconfig.sh must be adjusted. In case the USB device is plugged in while the guest is already running, you need to start the guest monitor using the vmmon.sh command in the vm directory.
Using the info command, a list of connected USB devices can be found:
Example:

```
info usbhost
  Bus 1, Addr 11, Port 1, Speed 480 Mb/s
    Class 00: USB device 0951:1665, DataTraveler 2.0
  Bus 2, Addr 3, Port 4.2, Speed 5000 Mb/s
    Class ff: USB device 0b95:1790, AX88179
  Bus 1, Addr 8, Port 8, Speed 12 Mb/s
    Class ff: USB device 06cb:009a
  Bus 1, Addr 7, Port 7, Speed 480 Mb/s
    Class ef: USB device 04f2:b604, Integrated Camera
  Bus 1, Addr 9, Port 6, Speed 12 Mb/s
    Class e0: USB device 8087:0a2b
  Bus 1, Addr 4, Port 5, Speed 12 Mb/s
    Class 00: USB device 058f:9540, EMV Smartcard Reader
  Bus 1, Addr 2, Port 2, Speed 12 Mb/s
    Class 00: USB device 046d:c52b, USB Receiver
```

The following command will dynamically connect the USB device on host bus 1 and address 11 (an USB stick) with the guest:

```
device_add usb-host,id=MyUsbDevice,hostbus=1,hostaddr=11
```

The id value has to be unique in case multiple USB devices are connected.

More information can be found here:

- https://github.com/qemu/qemu/blob/master/docs/usb2.txt
- https://unix.stackexchange.com/questions/426652/connect-to-running-qemu-instance-with-qemu-monitor/476617
- https://wiki.archlinux.de/title/QEMU#USB_Peripherie

# 3 Tutorials

## 3.1 Run shipped RTOS container

### *3.1.1 General*

RTOS images (containers) are located in the appropriate directory beyond /hv (e.g. /hv/lx for RT Linux containers).

The first time, when a RTOS container is started, the RTOS Virtual Machine is loaded and the configuration for the RTOS container(s) are loaded. In case a configuration entry has changed, all RTOS containers need to be stopped and the RTOS Virtual Machine to be reloaded.

Configuration files are stored in config files. The following operations are supported via calling the appropriate scripts:

- start RTOS (e.g. lx.sh to start the RT Linux image)
- stop RTOS: rtosstop.sh
- stop RTOS and RTOS Virtual Machine: stopall.sh
- open Debug Console: dbgcon.sh

### *3.1.2 Real-time Linux container*

Important note: you need at least 384 Mbyte RAM for the shipped Linux image. Assure the autoconf.sh script was executed with the appropriate RAM size.
Real-time Linux containers are stored in /hv/lx
Start the container using the lx.sh script.
From within the real-time Linux OS you may access the Hypervisor filesystem via the /mnt/rtfiles mount point. The root directory will point to the Hypervisor directory /hv/lx/rtfiles.

### *3.1.3 VxWorks container*

VxWorks containers are stored in /hv/vx
Start a VxWorks 6.9 container using the vx69.sh script.

#### *3.1.3.1 File system access*

To access the Hypervisor filesystem from within VxWorks, you may use FTP.
For security reasons, by default, there is no FTP server installed.
Follow these instructions to install an FTP server in the Hypervisor.

- sudo apt-get install vsftpd
- activate the entry `write_enable=YES` in /etc/vsftpd.conf
- Restart the FTP server: /etc/init.d/vsftpd restart
- Create an FTP user target with password vxworks:
  - adduser target
  - passwd vxworks
  - You may use visudo to add the user target to the sudoers group (duplicate the root entry)
- After executing the above steps, from within VxWorks, you should be able to access the directory /home/target

### *3.1.4 On Time RTOS-32 container*

RTOS-32 from "On Time" company is a hard real-time operation system. It is one of the Operating systems supported by acontisHypervisor.

Directories:
**/hv** - this root directory contains all acontisHypervisor files and executables.
**/hv/rtos-32** - RTOS32 configuration files and start/stop scripts as well as OS binaries.
**/hv/rtos-32/rtfiles** - directory for your .dlm files.

Most important bash scripts:
**/hv/rtos-32/realtimedemo.sh**
starts RTOS32 VM and the acontis tool measuring context switch and interrupt latencies.

**/hv/rtos-32/realtimedemo-debug.sh**
starts debug monitor that awaits requests from a remote debugger on a Windows machine with installed EcWin, Visual Studio, and RTE Plugin.

**/hv/rtos-32/ecmasterdemo.sh**
starts acontisHypervisor RTOS32 VM and EtherCAT MasterStack demo.

**/hv/rtos-32/stopall.sh**
stops VM and acontisHypervisor.

**/hv/rtos-32/dbgcon.sh**
opens interactive RTOS32 VM console, so you can interact with your app here.

Specific scripts used for debugging:
**/hv/hvctl/brvnetset.sh**
creates a virtual network bridge on Linux host to forward debugger TCP/IP/UDP packets from LAN1 to RTOS32 VM. It is required to start this script if you need to perform remote debugging of a RTOS32 app from another machine.

**/hv/hvctl/brvnetclr.sh**
deletes bridge, after the RTOS32 VM has been stopped.

Start a RTOS-32 container using the rtos-32.sh script.

#### *3.1.4.1 PC Configuration Prerequisites*

It is assumed that you already have installed acontisHypervisor and configured it as is is described in acontisHypervisor.pdf.

It is also assumed, that your HypervisorPC has two LAN ports and you already have assigned LAN2 to acontisHypervisor, as it is described in acontisHypervisor.pdf in the

"PCI/PCIe device assignment" section. Ths step is only required if you want to start EcMasterDemo, because it requires one LAN Port to communicate with EtherCAT slaves.

If you want just run other demoes (ex. realtimedemo), or want to remotely debug your RTOS32 app, you can skip this step.

#### *3.1.4.2 Achieving Hard Real-Time Capabilities*

It is important to understand, which factors have influence to the real-time capabilities of your hardware. Most important of them are BIOS Settings (no CPU power saving modes should be activated, no CPU throttling, no variable CPU Frequency, USB Legacy support should also be disabled). Please refer to the acontisHypervisor documentation). SMI must be also avoided.

Video Card has usually a huge impact on the real-time capabilities of the real-time system, because it can not only generate SMI but also perform huge background DMA transfers from/to DRAM memory.

For example, if you have Intel i915 video card on your Linux host, its Linux driver can produce significant interrupt/context latencies.

First step, in order to achieve better latencies, is to disable LightDM graphics manager in your Xubuntu Installation of acontisHypervisor. Simply run the following command "**sudo apt-get remove lightdm**" and the reboot your system.

This step converts your system into console only mode, no GUI. But it is not enough, because a video card can do DMA Transfers even without GUI.

The easiest and fastest solution here is to kill its driver as follows:

- establish connection with HypervisorPC via SSH (port 22)
- kill video driver "sudo rmmod -f i915. Now the Computer Monitor cannot be used anymore.
- start realtime demo and make sure, that context switch/interrupt delays much better now (200%+):
  *cd /hv/rtos32*
  *sudo /realtimedemo.sh*

#### *3.1.4.3 How to run a sample preconfigured RTOS32 App (Realtimedemo)*

AcontisHypervisor has a special tool ("RealtimeDemo"), which can accuratelly measure the real-time capabilities of a machine and the hypervisor. This tool can be also used as a first sample realtime app, to play with the acontisHypervisor.

To start it, execute /hv/rtos-32/realtimedemo.sh script.
This script loads/starts acontisHypervisor VM, RTOS32 OS Image and starts RealtimeDemo.dlm.
Execute /hv/rtos-32/stopall.sh script to stop everything.

Most important parameters here are: Interrupt Delay and Task Delay (in microseconds). These parameters show the realtime capabilities of the HypervisorPC and acontisHypervisor.

You can press Ctrl-C to exit from the console into Linux Terminal. Please execute dbgcon.sh script to return to the RTOS32 console again.

It is also easily possible to run the realtime demo in the background, completelly detached from the console. So, use dbgcon.sh script in this case, to open interactive RTOS32 terminal from a new console.

#### *3.1.4.4 How to run EcMasterDemo that uses a network card and the EthetCAT stack*

It is assumed that your HypervisorPC has two Ethernet ports LAN1 and LAN2. The first one can be used for a TCP/IP traffic and/or to estalbish a debug connection with a DevPC.

It is also assumed, that your second LAN port was already assigned to acontisHypervisor, as it is described in acontisHypervisor.pdf in the "PCI/PCIe device assignment" section.

Execute the following steps:

1. Connect LAN2 port to a EtherCAT slave.
2. cd /hv/rtos32

3. sudo ./ecmasterdemo.sh

All required files like EcMasterDemo.dlm, EcMaster.dlm, emllRTL8169.dlm are already in the rtfiles/ecmaster subdirectory.

#### *3.1.4.5 How to remotely debug a RTOS32 App from a Windows Machine with Visual Studio*

It is assumed, that you also have a second Windows PC (DevPC), where you installs Visual Studio + EcWin + RTE Plugin. DevPC is used to develop your RTOS32 Application and perform a remote debugging of this app on the HypervisorPC.

Acontis has developed a special Visual Studio plugin that provides possibility to create a RTOS32 Project, compile, debug it (local or remote) and configures a Visual Studio environment for you.

Requirements for a Windows Machine:

* Visual Studio 2015 should be installed

* acontis Rtos32Win/EcWin should be installed

Topics like creating a new RTOS32 Project are out of scope of this document. Please refer to the official Acontis RTOS32 documentation to find details.

Before we start, it is important to understand, how RTOS32 Apps work in the RTOS32 VM. Every App consists of a Loader part (/hv/rtos-32/Loader.bin) and a dynamically loaded module in a .dlm format (/hv/rtos-32/rtfiles/yourapp.dlm). Loader.bin is provided with full source code and comes as a separate Visual Studio project. It is automatically generated by the RTE VS Plugin using the method described above.

But an application, that is is supposed to be debugged, should use a separate bootloader called "RTOS-32 Monitor" (/hv/rtos-32/debug/Monvmf.bin).

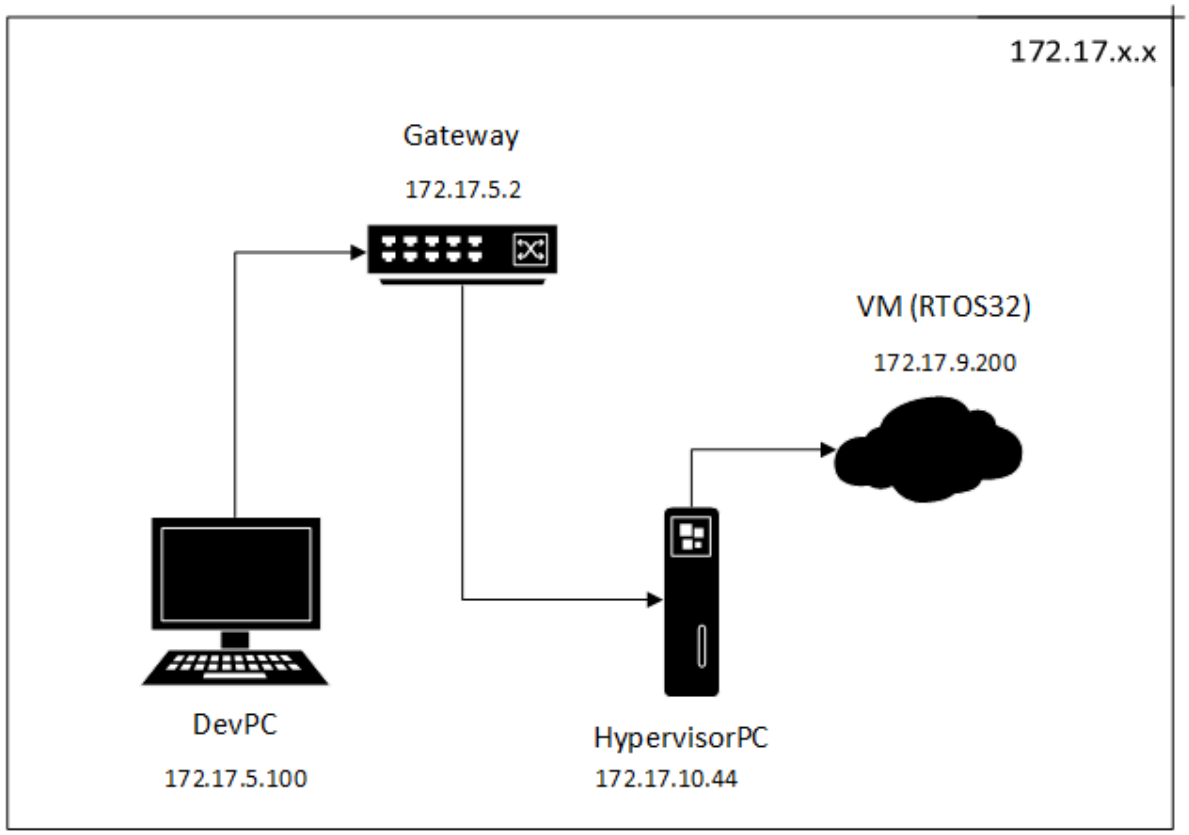
The monitor starts in RTOS32 VM and listens to a TCP/IP trafic from the remote Visual Studio debugger.

The second important thing: all Monitor settings (like an IP address to listen to) are always embedded into the Monvmf.bin file. So, copy Monvmf.bin to the HypervisorPC from DevPC every time when you change remote IP address in the System Manager (the part of EcWin/Rtos32Win).

Network Configuration:

The HypervisorPC should be configured to peform the remote debugging; scripts and configuration files should contain specific IP addresses/masks/subnets. The requirement is that DevPC, HypervisorPC and RtosVM are in the same subnet, in our example 172.17.x.x.

Our sample network configuration:



All these IP adresses should be set here:

/hv/hvctl/brvnetconfig.sh

In our example, the Gateway is a simple Router with running DHCP Server, that assigns static 172.17.5.* adresses to development PCs in the office and dynamic 172.17.10.* adresses to all other computers connected to the router.

VM with address 172.17.9.200 is in a separate subnet, not phisically connected to the router. But HypervisorPC accepts MASK 172.17.255.255 on its virtual network bridge (read below) and can forward all traffic to and from a VM from the 172.17.x.x network.

By simple words, the DevPC communicates always with VM. In the VM a special Debug Stub is started (called “RTOS32 Debug Monitor”) that listents to the 172.17.9.200 IP address.

Open project of a sample demo app (we describe here how to debug "RealtimeDemo"):

1. DevPC: open System Manager and right click "RTOS #1\Application" and choose "Create New Application Project (Debug Only)"

2. Choose "RealtimeDemo" and click OK button. Visual Studio projects are generated now and copied to RtFiles and other subsirectories of your System Manager Workspace.

Prepare a Debugger Bootloader (RTOS-32 Monitor):

1. DevPC: Select "Remote Debugging" and click "Settings" button near to a grayed IP text field.

2. Enter "172.17.9.200" in the "IP" text field and press "OK" button. Please note, now the new RTOS-32 Monitor binary (Monvmf.bin) is generated and the target IP addresss is embedded into the binary.

Compile the .dlm project:

1. DevPC: Click "Open Project with Visual Studio" button.

2. In the Visual Studio two projects are available in the Solution Explorer: "Loader" and "RealtimeDemo".

3. Right click every project and click "Build"

Copy binaries to the target (HypervisorPC):

1. Copy RealtimeDemo.dlm from the RtFiles directory in your workspace on DevPC to /hv/rtos-32/rtfiles/debug/ directory.

2. Copy projects/monvmf/Monvmf.bin to /hv/rtos-32/debug/ directory

3. Make sure /hv/rtos-32/realtimedemo-debug.config file has correct settings. It should have a valid fileserver directory (hv/rtos-32/rtfiles/debug/) and it sould of course include .config file for a PCI Card for LAN2 port, that is assigned to the RTOS VM.

Start RTOS-32 VM and the RTOS-32 Monitor (HypervisorPC):

1. Execute /hv/rtos-32/realtimedemo-debug.sh. Please note, the .dlm module was

already added to the realtimedemo-debug.config in the "Rtos\Loader" section.

2. You should see a RTOS-32 Monitor output. It listens to the IP address

172.17.9.200 to receive the remote debugger traffic.

Please check all your network setings here: /hv/hvctl/brvnetconfig.sh

Please note, vnet0 virtual network adapter is created on Linux Host, when the VM is started. Please consider this adapter as a virtual PCI Network Card inside the RTOS32 VM, that is visible in Linux Hoast as vnet0.

By default the 192.168.178.1 address is assigned to this adapter on a Linux side.

Configure a virtual bridge network adapter

Configure a virtual bridge network adapter to forward all incoming debugger traffic from LAN1 to the virtual network adapter "vnet" in the RTOS VM.

Please execute /hv/hvctl/brvnetset.sh script. The virtbr network adapter is created on the linux side.

Start debugging (DevPC):

1. In Visual Studio open file Loader.cpp.

2. Locate main() function

3. Set a breakpoint.

4. Press F5 key to start debug.

How to repeat/stop debugging:

1. There is no need to restart the RTOS-32 Monitor if you want to stop the debugging.

   Simply click Stop in the Visual Studio and then press F5 again.

2. It is not needed to copy the RTOS32 Monitor binary (Monvmf.bin) every time you

   make changes in the Loader or a .dlm project. Only when the IP address is changed.

3. If you found a bug in your .dlm module and need to upload a new verison into the

   HypervisorPC, please do the folliwing:

   a. compile project

   c. copy changed .dlm binary to the /hv/rtos-32/rtfiles/debug/ directory.

3. If you want completely stop your VM, make it in the following sequence:

   a. Execute /hv/hvctl/brvnetclr.sh to remove the bridge

   b. Execute /hv/rtos-32/stopall.sh

## 3.2 Install Windows or Linux guest

See chapter 2.7

### *3.2.1 UEFI and Legacy BIOS*

### *3.2.2 UEFI*

#### 3.2.2.1 What is SecureBoot

Secure Boot is an interface between UEFI and Operating System. When SecureBoot is activated, it prevents the loading of unsigned boot loaders or drivers

Read More:
https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface

#### 3.2.2.2 Preparing Keys for SecureBoot

The key thing in SecureBoot is a Platform Key (Platform). It establishes relationship between a platform owner and a platform firmware. PK is a self-generated certificated owned by OEM.

Another important key is a KEK key, This key is obtained from an OS Manufacturer (for ex. Microsoft) and is used to establish trust relationship between the Firmware and OS.

Generating the Platform Key:

```
openssl req -newkey rsa:2048 -nodes -keyout PKpriv.key -x509 -days 365 -out PK.crt
Generating a 2048 bit RSA private key
....+++
.+++
writing new private key to 'PKpriv.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:Bayern
Locality Name (eg, city) []:Munic
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Beer Inc
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []: BayernBeer
Email Address []:
```

OVMF supports keys in DER format only. So we need to convert t:

```
openssl x509 -in PK.crt -outform der -out PK.der
```

Download Key Exchange Key (KEK)
MicCorKEKCA2011_2011-06-24.crt
https://go.microsoft.com/fwlink/p/?linkid=321185

Download Signature Database (allows Windows to boot):
MicWinProPCA2011_2011-10-19.crt
https://go.microsoft.com/fwlink/?LinkId=321192

Download Microsoft signer for third party UEFI binaries via DevCenter:
MicCorUEFCA2011_2011-06-27.crt
https://go.microsoft.com/fwlink/p/?LinkID=321194

#### 3.2.2.3 Building OVMF with SecureBoot support

By default, OVMF is built without SecureBoot support.

So it is recommended to fetch this project from its repository and build ovmf yourselves.

Install required packages:

```
sudo apt-get install build-essential git uuid-dev iasl nasm -y
sudo apt-get install iasl -y
```

git clone git://github.com/tianocore/edk2.git

cd edk2

Prepare build tools:

git submodule update –init

make -C BaseTools

. edksetup.sh

make -C ./BaseTools

export EDK_TOOLS_PATH=/home/rte/edk2/BaseTools

. edksetup.sh BaseTools

Edit Conf/target.txt:

ACTIVE_PLATFORM = OvmfPkg/OvmfPkgX64.dsc

TARGET_ARCH = X64

TOOL_CHAIN_TAG = GCC5

Build OVMF with SecureBoot support:

```
OvmfPkg/build.sh \
   -a IA32 -a X64 \
   -D SMM_REQUIRE -D SECURE_BOOT_ENABLE \
   -D FD_SIZE_2MB -D EXCLUDE_SHELL_FROM_FD
```

Binarties can be found in the Build directory.

#### 3.2.2.4 Embedding SecureBoot keys to OVMF

Create a OVMF-SecureBoot directory and copy Build/OvmfX64/DEBUG_GCC5/FV/OVMF_CODE.fd and Build/OvmfX64/DEBUG_GCC5/FV/OVMF_VARS.fd to this directory.

Create a hda subdirectory and copy all generated and downloaded keys to this subdirectory.

Run qemu:

cd OVMF-SecureBoot

```
qemu-system-x86_64 -L . \
  -drive if=pflash,format=raw,readonly,file=OVMF_CODE.fd \
  -drive if=pflash,format=raw,file=OVMF_VARS.fd \
  -hda fat:hda \
  -net none
```

After booting you get to a UEFI shell. Type “exit”.

Go to Device Manager / Secure Boot Configuration / Secure Boot Mode and change from “Standard Mode” to “Custom Mode”.

PK Options / Enroll PK / Enroll PK Using File and choose PK.der

KEK Options / Enroll KEK / Enroll KEK Using File and choose MicCorKEKCA2011_2011-06-24.crt

DB Options / Enroll Signature / Enroll Signature Using File and choose MicWinProPCA2011_2011-10-19.crt

Repeat last step and choose MicCorUEFCA2011_2011-06-27.crt

The Secure Boot Mode should be “Enabled” now.

Exit from BIOS, shutdown the machine.

## 3.3 Windows and Virtualization Based Security

<<EXPERIMENTAL – NOT FULLY WORKS IN HV YET>>

Microsoft provides role called the Hyper-V role, that allows Microsoft with activated VBS to isolate some sensitive information, such cached credential, in not-accessble places.

When VBS is activated, Windows self works in a separate VM called “Root Partition”. All sensitite Credential are located in a separate quasi VM and the communication is performed via RPC.

Because Hyper-V means activation of a Microsoft Hypervisor (Type 1), we talked here about *Nested Virtualisation.*

VBS and Hyper-V have some requirements to a Hardware Layer:

- 64-bit PC
- IOMMU Support
- TPM 2.0 Device
- UEFI with activated SecureBoot

Acontis Hypervisor provides a Hardware Layer which is compatible with these requirements, so VBS (both CredentialGuard and HVCI) can be activated.

Info vmconfig.sh Adaption

# Enable VBS

export enabe_vbs=1

<<DEVELOPERS INFO BEGIN >>
When export enabe_vbs=1, the following thing must be done:

- UEFI BIOS should be used
- Separate OVMF Code und Data binaries should be used to start qemu machine. They should be included into the .iso package. It is prioncipally possible to generate own platform keys and embedd them into the Data OVMF file. Of course, a customer should be able to generate and embed own keys.
- A separate flag should be used to speicfy if TPM device needs to be passthrough or a emulator server should be started
- Specific –host flags should be used to hide the underlying hypervisor from Windows Hypervisor to work properly

<<DEVELOPERS INFO END >>

<<DEVELOPERS INFO BEGIN >>

Current version of qemu 2.12 in acontis Hypervisor does not properly support TPM – Windows can not detect TPM provided by old qemu. Please build qemu self by fetching it from a git repository. Currently version 5.0.1 was tested and Windows recognizes its TPM device. Or we need to build it self and include to a .iso

<<DEVELOPERS INFO END >>

CredentialGuard needs a TPM Device to store user credentials. There are two possibilities to provide such device to a Windows Root Partition – use a hardware chip on your host PC directly by the Windows VM (please note, TPM device is not designed to share its data, so, it should not be used by Linux Host) or you can use so called TPM Emulator. There is a IBM Open Source project SWTPM, which can emulate TPM 1.2 and TPM 2.0 and QEMU can work with this emulator using sockets. But in the latter case you should find the way, how to encrypt TPM Emulator files from the Linux host administrator. The whole concept of VBS is to protect VM’s data from Host,

<<DEVELOPERS INFO BEGIN >>

Unfortunatelly, I was not able to get it working – TPM passthrough to qemu. Qemu expects some files provided in sysfs of tpm driver, which do not exists. More time should be invested if this feature is needed.

In short, the kernel should be built with TPM support and then /dev/tpm0 device should be mentioned in qemu paremetes

```
-tpmdev passthrough,id=tpm0,path=/dev/tpm0 \
```

-device tpm-tis,tpmdev=tpm0 test.img

MLV

<<DEVELOPERS INFO END >>

<<DEVELOPERS INFO BEGIN >>

Please fetch and build the project according to the projects documentation.
***https://github.com/stefanberger/swtpm***

to use TPM:
mkdir /tmp/mytpm0

swtpm socket --tpmstate dir=/tmp/mytpm0 --tpm2 --ctrl type=unixio,path=/tmp/mytpm0/swtpm-sock --log level=20 &

/home/rte/qemu/x86_64-softmmu/qemu-system-x86_64 -enable-kvm \
-machine q35,accel=kvm \

…
-chardev socket,id=chrtpm,path=/tmp/mytpm0/swtpm-sock \
-tpmdev emulator,id=tpm0,chardev=chrtpm \
-device tpm-tis,tpmdev=tpm0 \

Another important thing is also TPM support in OVMF. By default this option is DEACTIVATED.

Please check previous section how to build OVMF and add two additional macros:

OvmfPkg/build.sh \
….
-D TPM2_ENABLE -D TPM2_CONFIG_ENABLE

If course it also makes sense, to pre-build own OVMF Code and Data files with TPM and SecureBios support and even embed demo Platform Keys and supply these files with .iso
MLV

<<DEVELOPERS INFO END >>

To check if TPM works, open Device Manager and find TPM 2.0 in Security Devices section.

Next thing, which is needed for VBS aktivation is SecureBoot, activate it and inject keys how it was described in previous sections.

<<DEVELOPERS INFO BEGIN >>

Unfortunatelly, it is not enough to activate the features described above.

It is interesting, that Microsoft Hypervisor has different requirements to Hardware, when using UEFI or SeaBIOS.

For some strange reason, Hyper-V can be easily activated with default CPU passthrogh settings for qemu:

-cpu host

For for UEFI OVMF it is not so! To get Hyper-V activated, it is needed to perform two additional steps: to hide persistence of an underlying qemu hypervisor and disabe CPU feature called MPX (some kind of a dead intel technology)

-cpu host,-hypervisor,-mpx

We should add this logic to our bash scripts und use these flags when enable_vbs=1 is specified in vmconfig.sh

<<DEVELOPERS INFO END >>

Next step we need to do is activate Hyper-V Feature (it is assumed that you have installed Windows 10 Pro to your VM). Windows Server 2016+ should be also supported.
Please refer this document and use one of the methods
https://docs.microsoft.com/de-de/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v

for example.:

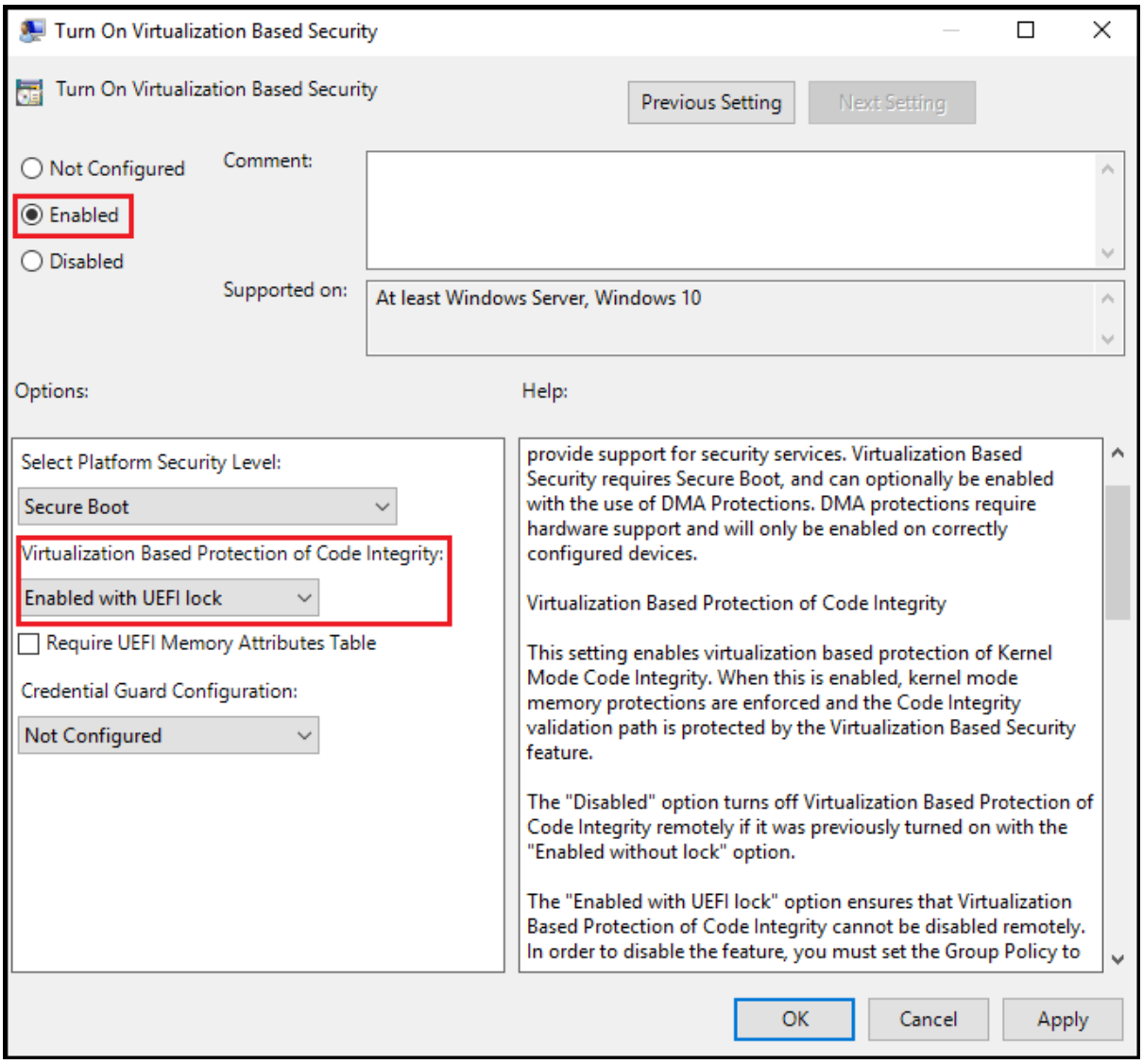
DISM /Online /***Enable-Feature*** /All /FeatureName:***Microsoft-Hyper***-V

Reboot and check if hypervisor is activated.

The best method to really check if hyper-v is activated is to use Hyper-V Manager to create a Virtual Machine and start it.

Next step is to activate VBS.

Run gpedit.msc and activate all options then reboot.




After reboot open System Information (msinfo32.exe) and find “Virtualisation Based Security: On”

<<DEVELOPERS INFO BEGIN >>

Unfortunatelly, at this point the research was suspended, because Windows VM crashes with BSOD WHEA Incorrectable Error. During debugging it was discovered, that with activated Hyper-V it is impossible to execute code from a PCI Bar (we pass a VMF.bin to Windows VM as a PCI Bar of a Memory Controller). It is possible to read this code but not execute.

Underlying Microsoft Hypervisor generates INT18 with MCE Exception (Machine Check Exception). This exception is used in bare-metal hardware to notify about serious hardware problems.

My Idea, is that a “shadow” Page Directory copy in Hyper-V (EPT) does not have “execute” permission for this memory block. Than happends VmExit from Windows Root Partition to the hypervisor and it signals with INT18 to the VM.

<<DEVELOPERS INFO END >>

# 4 Example Applications

## 4.1 General

## 4.2 Building a Windows application

## 4.3 Building a RTOS (VxWorks) application

# 5 User's Guide

## 5.1 N/A

# 6 Built-in tools and utilities

## 6.1 RtosService

# 7 Version History

N/A