Open Edge

Intelligent Desktop Virtualization (IDV)

Revision 0.4.0

Date: 8/12, 2020

RBHE Transactional

Philip Park

Joko Sastriawan

Contents

[1 Intelligent Desktop Virtualization (IDV) 5](#_Toc48118916)

[1.1 History 5](#_Toc48118917)

[1.2 Architecture overview 5](#_Toc48118918)

[1.3 Pre-requisite 6](#_Toc48118919)

[1.3.1 Ubuntu 18.04 6](#_Toc48118920)

[1.3.2 Centos 7 7](#_Toc48118921)

[2 Build Base OS 8](#_Toc48118922)

[2.1 Build kernel with IDV patch 8](#_Toc48118923)

[2.1.1 Build the Initrd (Initial Ramdisk) 8](#_Toc48118924)

[2.1.1.1 Ubuntu OS 8](#_Toc48118925)

[2.1.1.2 Centos OS 9](#_Toc48118926)

[2.1.2 Build kernel source 9](#_Toc48118927)

[2.1.2.1 With idv3.0 patches 9](#_Toc48118928)

[2.1.2.2 With idv-2.1 patches 12](#_Toc48118929)

[2.2 Install and configure kernel 14](#_Toc48118930)

[2.2.1 Installation of the kernel 14](#_Toc48118931)

[2.2.2 Configure grub to boot newly installed kernel 14](#_Toc48118932)

[2.2.3 GVTg Verification 15](#_Toc48118933)

[2.2.3.1 Check for “mdev\_supported\_types “ 16](#_Toc48118934)

[2.2.3.1.1 i915\_GVTg\_V5\_1 16](#_Toc48118935)

[2.2.3.1.2 i915\_GVTg\_V5\_2 17](#_Toc48118936)

[2.2.3.1.3 i915\_GVTg\_V5\_4 17](#_Toc48118937)

[2.2.3.1.4 i915\_GVTg\_V5\_8 17](#_Toc48118938)

[3 QEMU build 17](#_Toc48118939)

[3.1 Install qemu on LTS 20.04 17](#_Toc48118940)

[3.2 Install QEMU on LTS 18.04 18](#_Toc48118941)

[3.2.1 Install Package to build QEMU 18](#_Toc48118942)

[3.2.2 Download, configure QEMU, compile to install 18](#_Toc48118943)

[4 Putting it together 20](#_Toc48118944)

[4.1 Recommended directory structure 20](#_Toc48118945)

[4.1.1 “disk” directory 20](#_Toc48118946)

[4.1.2 “fw” directory 21](#_Toc48118947)

[4.1.3 “iso” directory 22](#_Toc48118948)

[4.2 Available GVTg control node 22](#_Toc48118949)

[4.2.1 Explanation of gvt\_disp\_ports\_status 22](#_Toc48118950)

[4.2.2 Explanation of gvt\_disp\_ports\_mask 23](#_Toc48118951)

[4.2.3 Explanation of gvt\_disp\_ports\_owner 24](#_Toc48118952)

[4.3 Network Preparation 24](#_Toc48118953)

[4.4 Graphics pass through 25](#_Toc48118954)

[4.4.1 Create UUID (GUID) 25](#_Toc48118955)

[4.4.2 Masking the ports for VGPU 25](#_Toc48118956)

[4.4.3 Assigning the port/s to vGPU using UUID 25](#_Toc48118957)

[4.4.4 De-coupling the ports 26](#_Toc48118958)

[5 Installing guest OS 26](#_Toc48118959)

[5.1 Start VM with USB device 26](#_Toc48118960)

[5.2 Advanced topic 27](#_Toc48118961)

[5.2.1 Network in NAT mode 27](#_Toc48118962)

[5.2.2 Utilization of system 27](#_Toc48118963)

[5.2.3 Display Switch between Host and VM 28](#_Toc48118964)

[5.2.4 Device pass-through/assignment to guest OS 29](#_Toc48118965)

[5.2.4.1 Passing USB by Bus and Device 29](#_Toc48118966)

[5.2.4.2 Passing USB by vendor and product ID 29](#_Toc48118967)

[5.2.4.3 Passing by USB port 30](#_Toc48118968)

[5.2.4.4 Passing by PCI device using vfio-pci 31](#_Toc48118969)

[5.2.4.5 Passthrough MISC devices 32](#_Toc48118970)

[5.2.5 Headless VM setup for Spice 32](#_Toc48118971)

[5.2.6 Automating VM deployment using Portainer 33](#_Toc48118972)

[5.2.7 Automating VM deployment using LibVirt 33](#_Toc48118973)

[6 Remote management using MeshCentral 34](#_Toc48118974)

[6.1 Basic usage of MeshCentral 34](#_Toc48118975)

[6.2 Advance features with MeshCentral 36](#_Toc48118976)

[Addendum 38](#_Toc48118977)

[Scripts to pull and build kernels 38](#_Toc48118978)

[Script to pull and build qemu 39](#_Toc48118979)

[VFIO PCI unbind and bind script 40](#_Toc48118980)

[Gvt.sh (create-vgpu, setup directory) 41](#_Toc48118981)

[Use case 43](#_Toc48118982)

[QEMU parameters 43](#_Toc48118983)

# Intelligent Desktop Virtualization (IDV)

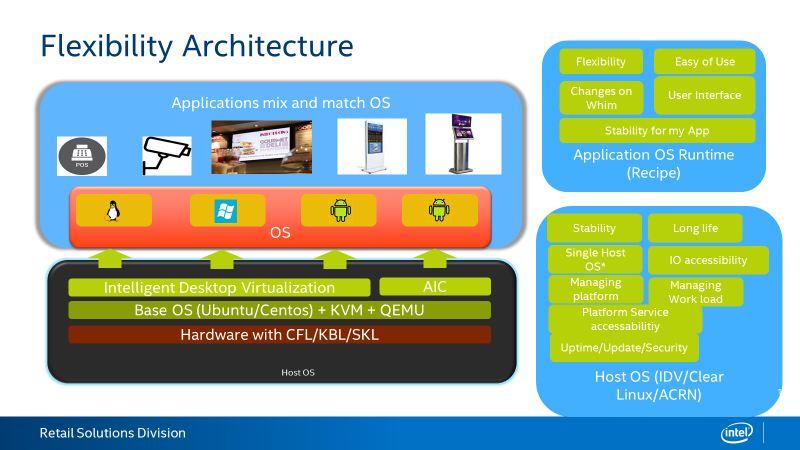
TBD Introduction: IDV, VTx, VTd,

## History

|  |  |  |
| --- | --- | --- |
| Revision | Date | Note |
| 0.2 | 2/27, 2020 | Draft for Ubuntu 18.04 |
| 0.2.1 | 2/28, 2020 | Update the document per Jimmy’s input (70% done) |
| 0.2.2 | 3/2, 2020 | Update the document per Jimmy’s input-done. |
| 0.2.3 | 3/3, 2020 | Update USB assignment method in Advance section  Add MeshCentral with advance features |
| 0.3.0 | 3/10, 2020 | Add gvt.sh setup file to automate the create-vgpu and vm directory structure |
| 0.4.0 | 8/12, 2020 | Add section 3.1 to support LTS 20.04  Add IDV 3.0 patch support (section 2, Build Base OS)  Add MISC device passthrough bkm |

## Architecture overview

IDV solution can co-exist AIC-CIC and AIC-CIV.



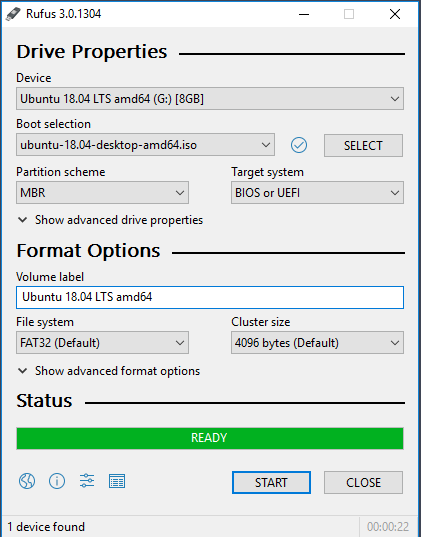
## Pre-requisite

IDV has been tested with Ubuntu 18.04 and Centos7

**Bootable USB**

Make bootable USB thumb drive with ISO. Rufus can be used to make bootable drive. Recommended size for USB thumb drive is greater than 8G.

Download Rufus from <https://rufus.ie/>.



### Ubuntu 18.04

Ubuntu 18.04 text mode is recommended. However, Ubuntu 18.04 minimal GUI is alternative.

Mini.iso

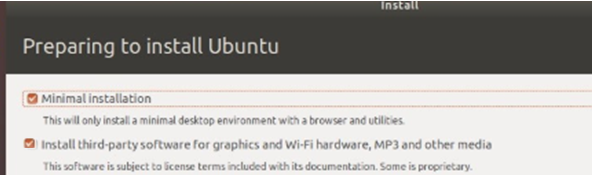
User can download the Ubuntu 18.04 text mode “mini.iso” from [mini.iso](http://archive.ubuntu.com/ubuntu/dists/bionic/main/installer-amd64/current/images/netboot/)

Minimal GUI installation option with normal Ubuntu 18.04 distro

User can download the Ubuntu 18.04 desktop ISO file from the following link.

<https://ubuntu.com/download/desktop>

Please select “Minimal Installation” and “development package” option while installing Ubuntu.



### Centos 7

Download the ISO image from <http://mirror.centos.org/centos/7/os/x86_64/>.

Tested with minimal installation with development package.

TBD

**Hardware requirement**

The IDV package currently supports **KBL/SKL/CFL**. Number of VM is depends on the memory size, number of threads in CPU, and application you want to run on each VM.

Supported graphics port is : HDMI/DP/DVI

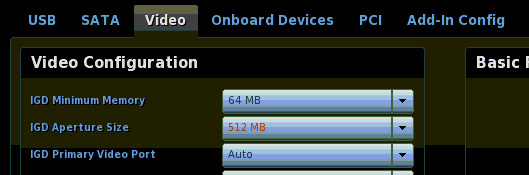
VGA port is not supported.

General rule of thumb is as below. Please note that implementer can experiment the allocation as 1 thread/2G memory also works depends on your application.

|  |  |  |
| --- | --- | --- |
| **OS** | **Thread** | **Memory** |
| Windows 10 | 2 | > 4G |
| Ubuntu | 2 | > 2G |
| Android | 2 | > 3G |

**Aperture size in BIOS setup**

Aperture size in BIOS setup needs to be at least 512. The recommend setting will be maximum size available above 512.



# Quick start guide

The semi-automated script will be used in this section for quick start. Please contact your local field intel engineers to get reference scripts.

|  |  |
| --- | --- |
| Build-kernel.sh | Script to build the kernel and install. Modifies kernel environment to accommodate gvt-g environment with newly build kernel |
| Config.sh | Configure guest OS in /var/vm directory |
| Install-guest-os.sh | Help to install the guest OS |
| Start-geust-os.sh | Help to validate installed OS |

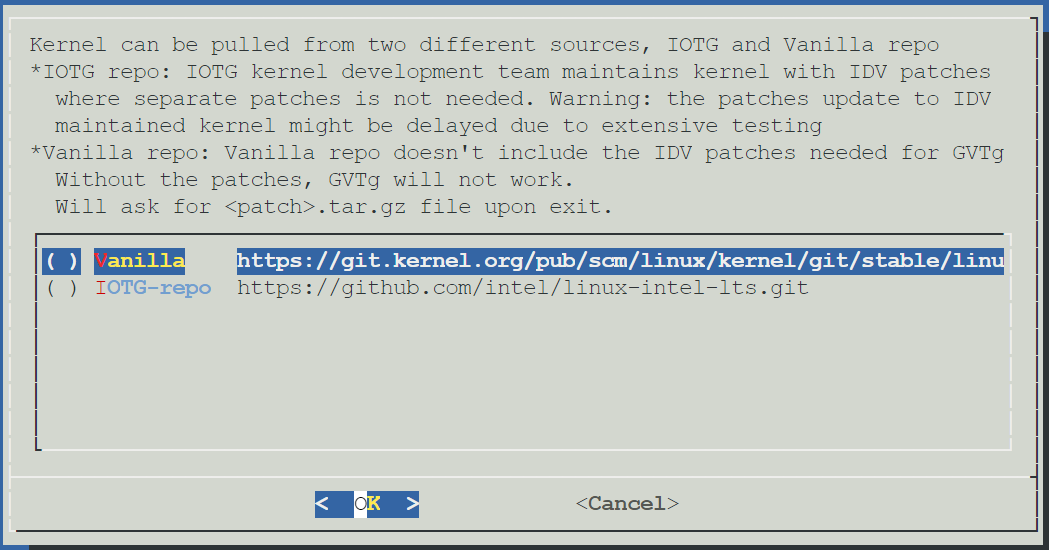
## Build kernel using script

When this script ran first time, it will go through the menu to establish the environment and source of the kernel.

The “.idv\_config” file will be created in “idv” directory. You can manually delete the “.idv\_config” file to start over.

The following screen will select the appropriate kernel repo. Recommended to use the **IOTG-repo** which available as public access.

The script will automatically pull the kernel source and start building in docker image.

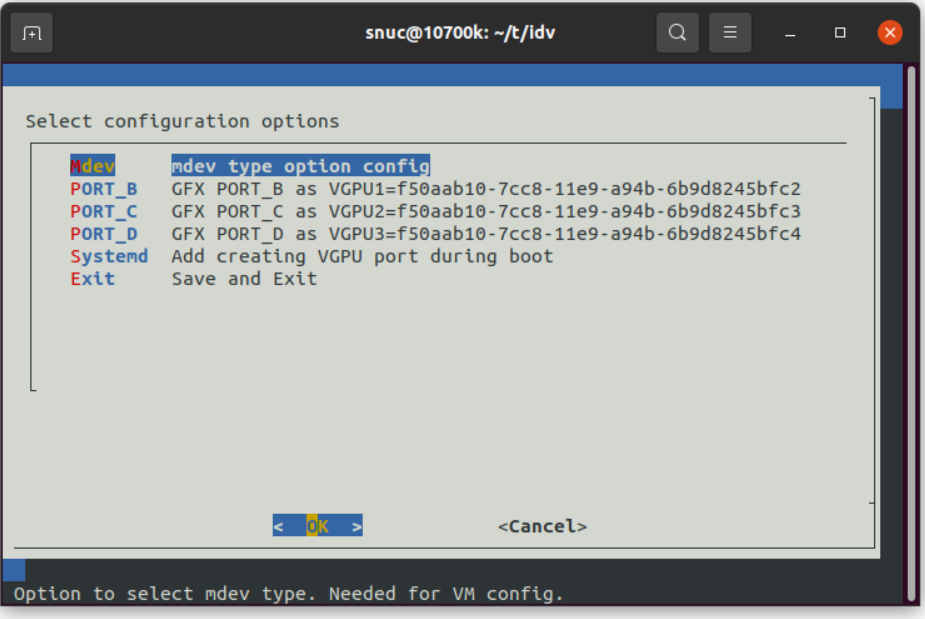


## Install kernel

Please install kernel using the option provide once kernel build successfully. It will configure the grub as part of the installation.

## Configuration using config.sh file

The following screen shows up when you run the “config.sh” file.



### mdev option

The options showing in this step is based on the memory allocated for GFX aperture. If enough memory is allocated, higher resolution option should be available. The default selection is for “1920x1080” resolution (I915-GVTg\_V5\_4).

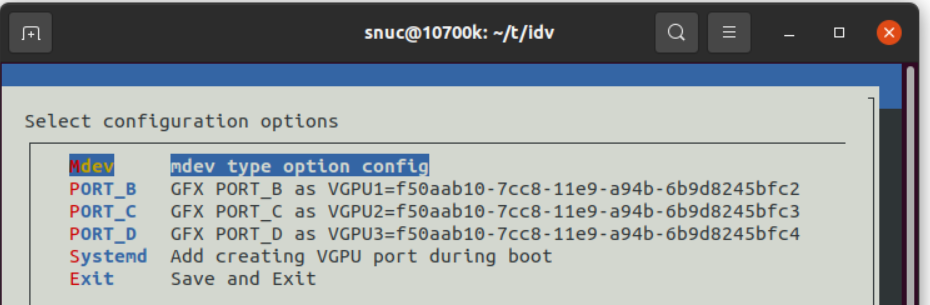
If available aperture size is set to small then this option will not show.



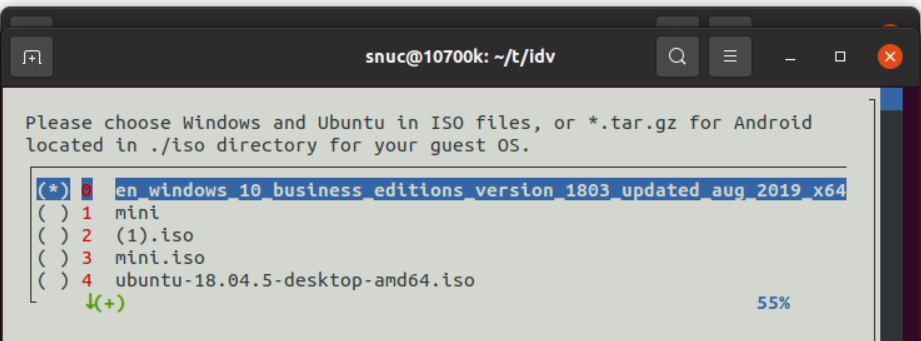
### Assigning each GFX port to guest OS

The port options will show up depends on how monitors are connected to the system. The following “PORT\_B”, “PORT\_C”, and “PORT\_D” for a system with 3 monitors are connected. If no monitor is connected, the “config.sh” script will abort with message.

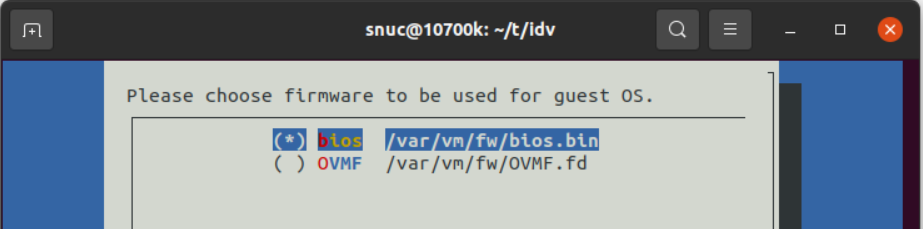
1. Select desired port (e.g. PORT\_B, PORT\_C, or PORT\_D)



1. Select guest OS ISO file. In this example, Windows are selected.



1. Select the firmware option
   1. For Windows, please select bios.bin file for stability



1. J
2. K
3. L

# Build GVTg enable OS and Configure for VM

Automated scripts are provided to aid

Please install Ubuntu LTS 18.04/20.04 in either text/minimal GUI mode on bare metal.

## Build kernel with IDV patch

Provided “build-kernel.sh” supports two kernel repositories. For now, it is recommended to use the Vanilla repository. When executed first time, the following dialog open for user to select source of kernel repository.

What it does:

Install docker

Create docker image

Install packages needed to compile the kernel in docker image

Modify the /etc/initramfs-/modules

Modify the grub file when installing newly built kernel

1. Vanilla

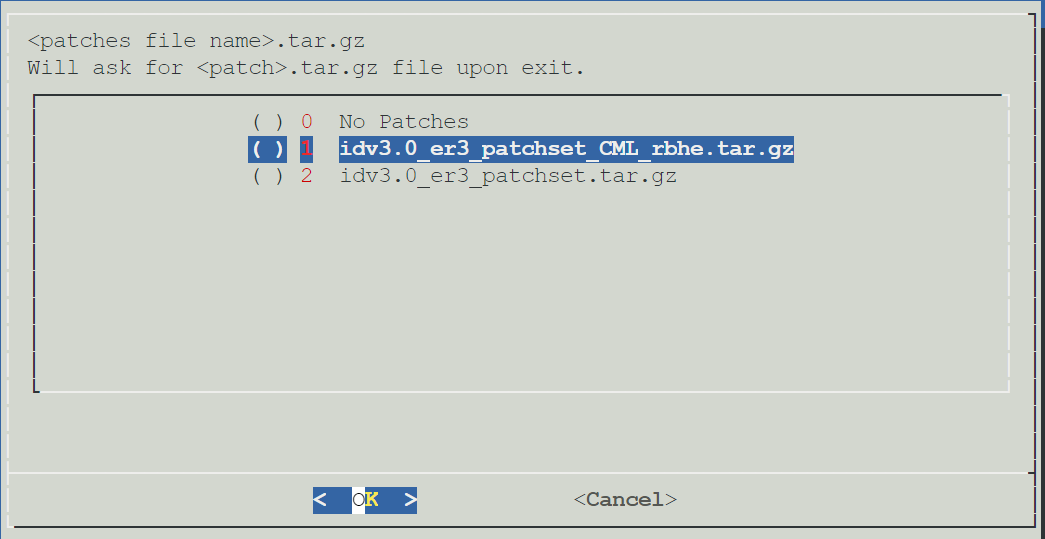
Vanilla is public kernel source repo. You can access this repo from any IP address.

1. IOTG-repo

IOTG-repo is internal to intel. Can’t access from public IP address. No patches is supplied if this option is selected.

|  |
| --- |
|  |

Select patches



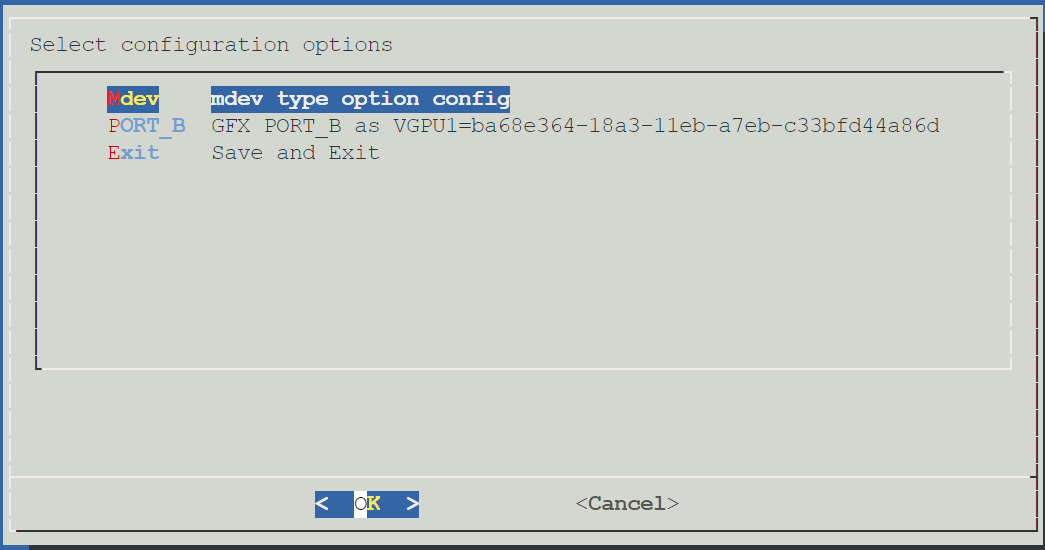
### 

Click <OK> to start build the kernel

## Configure the VM

Provided “config.sh” will configure your system to be able to run VM.

The config.sh file expects all guest OS ISO file located in “iso” directory. Download the guest OS ISO files before running “config.sh”.



/var/vm/ directory gets populated with scripts and files.

At this point, the guest OS disk is not created.

Need to run the install-guest-os.sh <option>

Need to reboot

Cd /var/vm/scripts/create-vgpu.sh

Sudo ./create-vgpu.sh

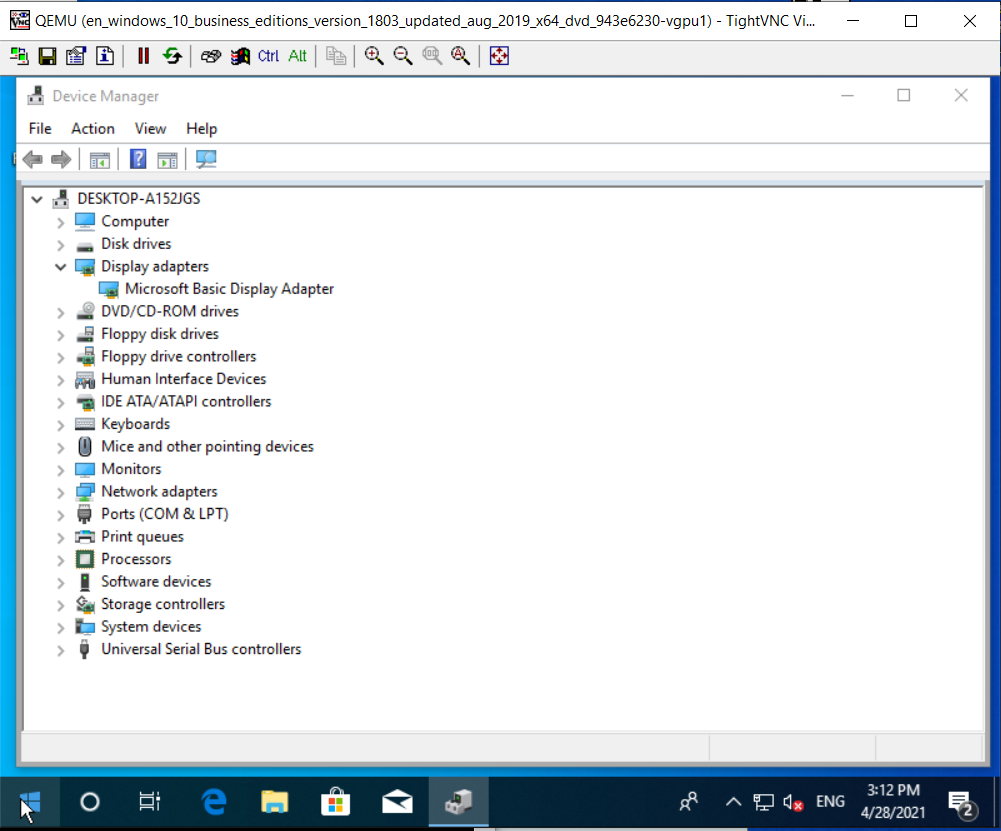
Install-guest.sh vgpuX

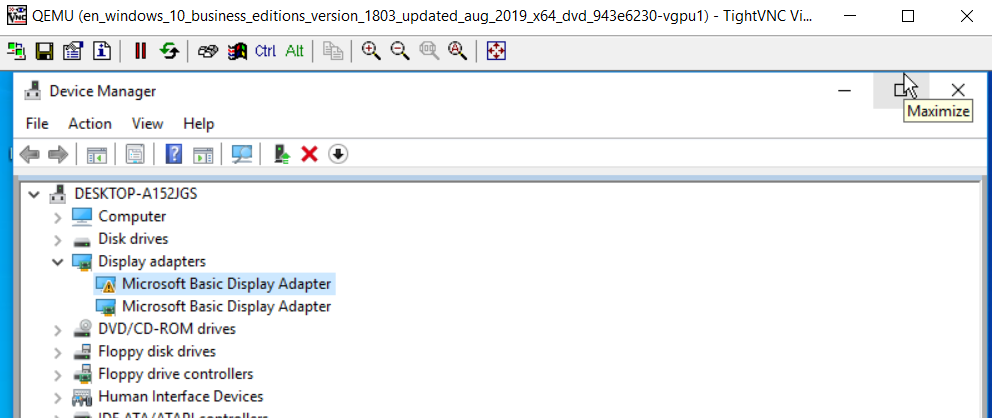
Start the VNC viewer to anticipate the user options during guest OS installation

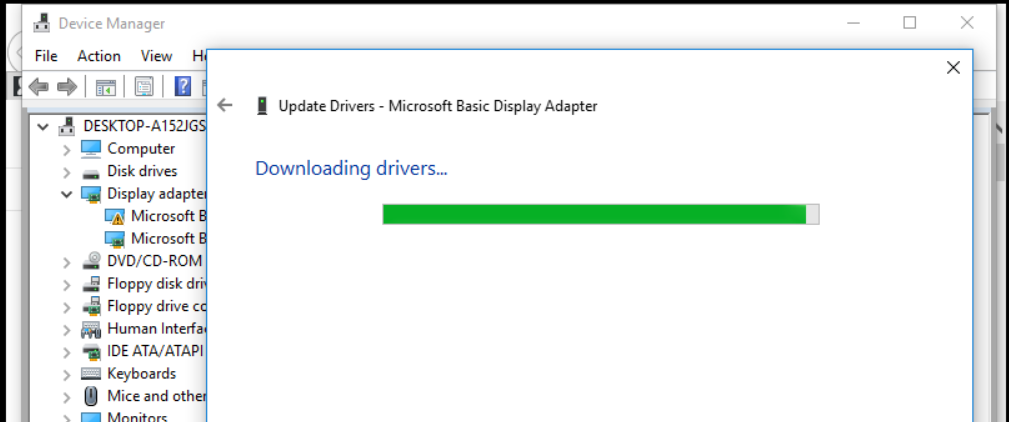
Please note that display driver is not loaded. It only shows MS basic display adapter driver.

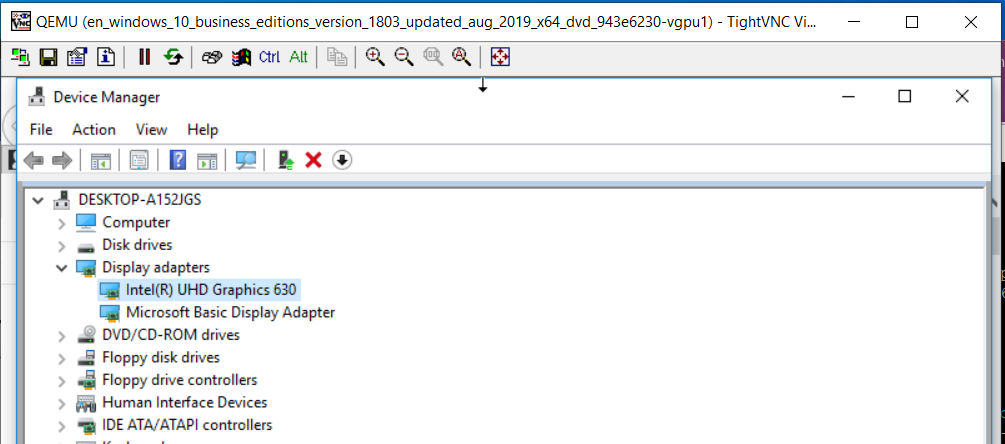
This is because the installation script is not connected to a GFX port via uuid. It only shows up on vncviewer.

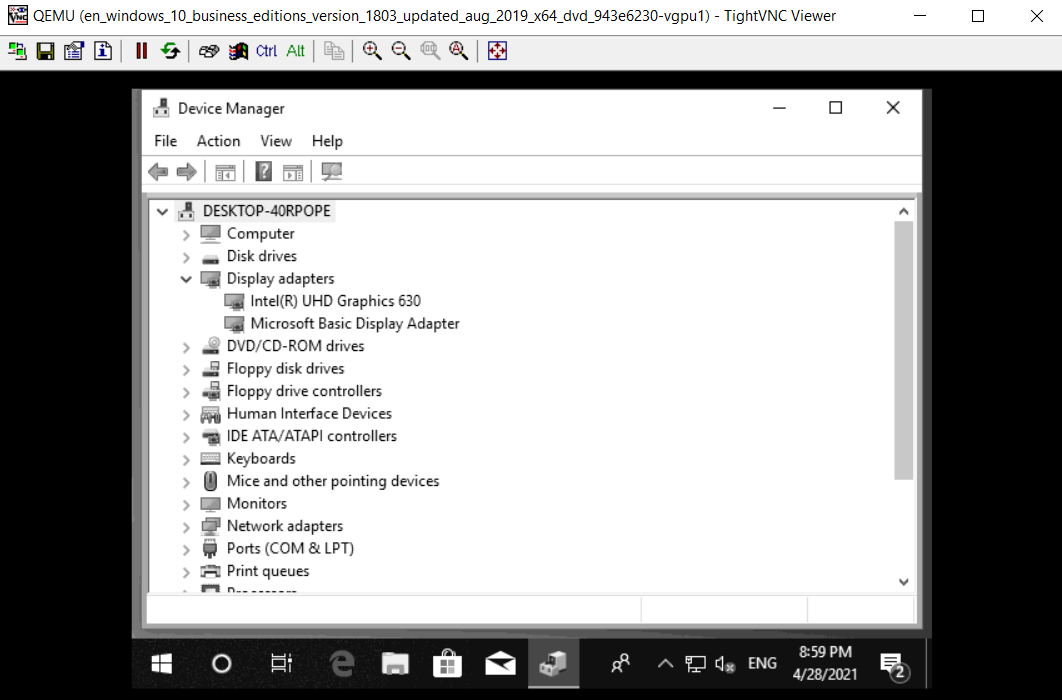
Now you can shutdown the guest OS.

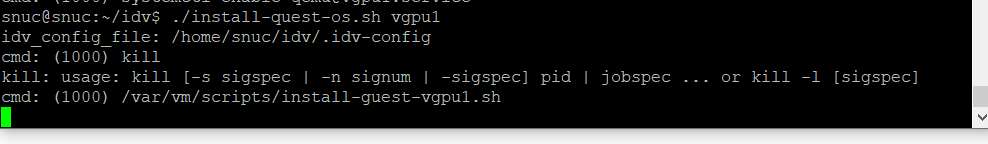


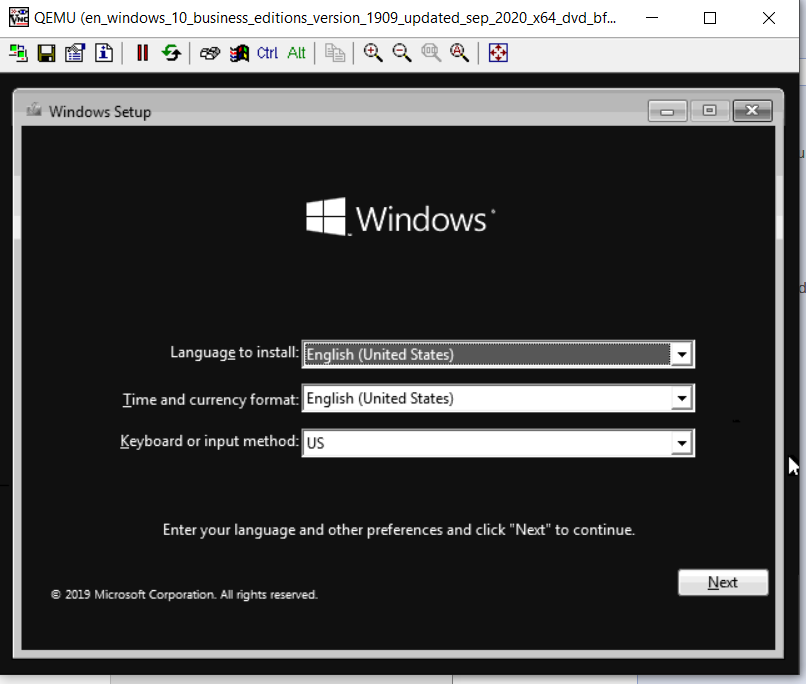




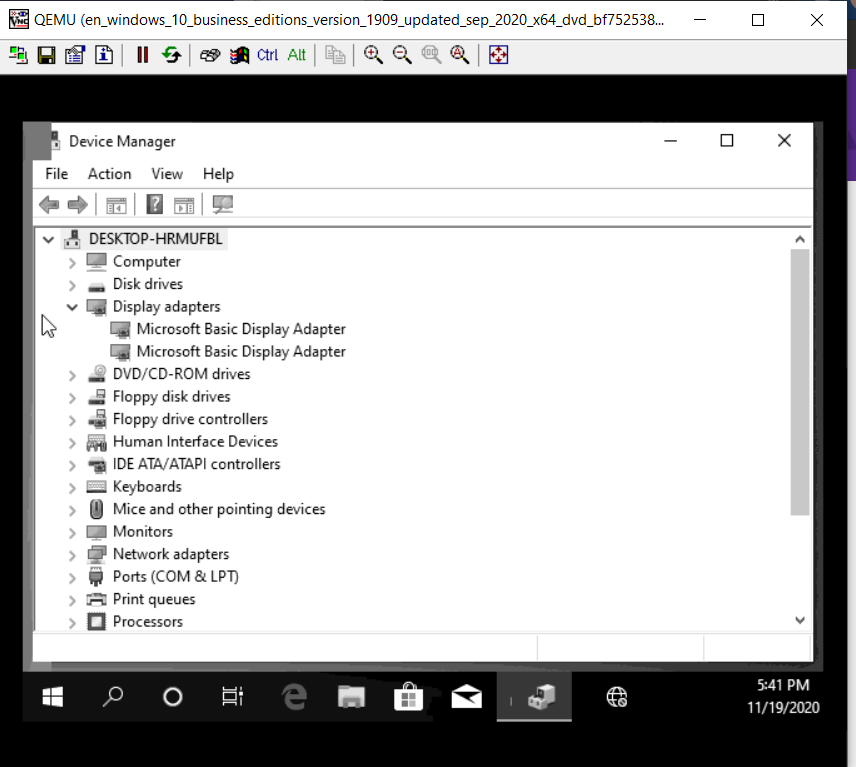








Note: you might want to use keyboard through-out the windows installation if mouse is not working



# Manual Build

Please install Ubuntu LTS 18.04/20.04 in either text/minimal GUI mode on bare metal.

## Build Kernel

### Build the Initrd (Initial Ramdisk)

The following is the list of modules needs to be added to kernel

* kvmgt
* vfio-iommu-type1
* vfio-mdev
* vfio-pci (optional. It is for PCI->USB cards pass through)

#### Ubuntu OS

Modify the /etc/initramfs-tools/modules as root to add the following modules.

Please refer to the following example.

|  |
| --- |
| Example of modules file  # List of modules that you want to include in your initramfs.  # They will be loaded at boot time in the order below.  #  # Syntax: module\_name [args ...]  #  # You must run update-initramfs(8) to effect this change.  #  # Examples:  #  # raid1  # sd\_mod  kvmgt  vfio-iommu-type1  vfio-mdev  vfio\_pci |

After saving /etc/initramfs-tools/modules, please execute the following command.

|  |
| --- |
| # execute following command to update  **$ sudo** update-initramfs -u |

#### Centos OS

Modify the /etc/dracut.conf file to add the modules before building the kernel

|  |
| --- |
| # additional kernel modules to the default  Add\_drivers+**=**"kvmgt vfio-iommu-type1 vfio-mdev" |

TBD

### Build kernel source

The following will pull the appropriate kernel branch and start kernel deb package.

For easy of documenting purpose, the following alias is used.

#### With idv3.0 patches

|  |  |
| --- | --- |
| **Alias** | **Note** |
| CDIR | current working root directory (recommend ~/build as your working directory)  CDIR**=$(pwd)** |
| URL | URL where kernel source can be download  URL**=**"https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git" |
| BRANCH | tag or branch for kernel tree in git  BRANCH**=**"v5.4.54" |
| WDIR | working directory where kernel source will be downloaded to  WDIR**=**"idv-3" |
| PATCHES | name of the patches you download from [here.](https://platformsw.intel.com/ddrivers.aspx?kitnumber=135055)  https://platformsw.intel.com/ddrivers.aspx?kitnumber=135055  PATCHES**=**"idv3\_er3\_patchset" |

**Note: subject to change as URL/branch/patch files are changing**

Building kernel involves following steps.

1. Install the packages

The following package is needed to build kernel. It is possible that additional package is needed depends on the installed OS. The following is tested with Ubnuntu 18.04 installation with “minimal GUI” option.

|  |
| --- |
| Example of modules file Packages needed  # Install packages needed  **$ sudo** apt-get install **-**y liblz4-tool kernel-package libelf-dev build-essential git libfdt-dev libpixman-1-dev libssl-dev vim bc socat libsdl1.2-dev libspice-server-dev autoconf libtool xtightvncviewer tightvncserver x11vnc uuid-runtime uuid uml-utilities bridge-utils python-dev liblzma-dev libc6-dev libegl1-mesa-dev libepoxy-dev libdrm-dev libgbm-dev spice-client-gtk libgtk2.0-dev libusb-1.0-0-dev bison flex openssh-server net-tools |

1. Update the package and upgrade

It is recommended to update and upgrade the packages with latest.

|  |
| --- |
| # Update the packages and upgrade  **$ sudo** apt-get update **&&** **sudo** apt-get upgrade |

1. Pull the kernel tree with correct tag or branch

The following command will connect to $URL git kernel repository to pull the specific tag or branch indicated in $BRANCH into $WDIR. The following command will create $WDIR that user doesn’t have to create.

|  |
| --- |
| ### IDV 2.1 section 3.2.2 Build Kernel Source  **git** clone **--**depth 1 **$URL** **--**branch **$BRANCH** **--**single-branch **$WDIR** |

1. Apply the patches

|  |
| --- |
| # unzip download patched  # Unzip downloaded patches  **tar** xzvf **$CDIR/$PATCHES.**tar.gz  **cd** **$WDIR**  **git** apply **$CDIR/$PATCHES/\*** **&>/**dev**/**null |

1. Configure the kernel

|  |  |
| --- | --- |
| # Create configuration file  **echo** ""**|make** oldconfig  # Verify the kernel configuration is done correct or check the  # contents by opening the .config file using text editor  **egrep** 'CONFIG\_DRM\_I915\_GVT|CONFIG\_VFIO\_MDEV' **.**config  # Make sure you see the following output   |  | | --- | | Table Output of "egrep 'CONFIG\_DRM\_I915\_GVT|CONFIG\_VFIO\_MDEV' .config  egrep 'CONFIG\_DRM\_I915\_GVT|CONFIG\_VFIO\_MDEV' .config  dev@build:~/work/idv21/idv-2.1$ egrep 'CONFIG\_DRM\_I915\_GVT|CONFIG\_VFIO\_MDEV' .config  CONFIG\_DRM\_I915\_GVT=y  CONFIG\_DRM\_I915\_GVT\_KVMGT=m  CONFIG\_VFIO\_MDEV=m  CONFIG\_VFIO\_MDEV\_DEVICE=m  dev@build:~/work/idv21/idv-2.1$ | |

1. Compile the kernel source with overlay method

The following command will generate deb package with version and revision code.

|  |
| --- |
| # The following will compile the code can create deb package  **$ CONCURRENCY\_LEVEL=`nproc`** fakeroot make-kpkg **--**initrd **--**append-to-version**=-**intelgvt **--**revision 2**.**1.0 **--**overlay-dir**=$cdir/**ubuntu-package kernel\_image kernel\_headers |

linux-image-4.19.65-intelgvt+\_2.1.0\_amd64.deb

linux-headers-4.19.65-intelgvt+\_2.1.0\_amd64.deb

* 1. –append-to-version=-intelgvt
  2. –revisioin 2.1.0

Where 2.1 is IDV package version and 0 is engineering release version

#### With idv-2.1 patches

|  |  |
| --- | --- |
| **Alias** | **Note** |
| CDIR | current working root directory (recommend ~/build as your working directory)  CDIR**=$(pwd)** |
| URL | URL where kernel source can be download  URL**=**"https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git" |
| BRANCH | tag or branch for kernel tree in git  BRANCH**=**"v4.19.65" |
| WDIR | working directory where kernel source will be downloaded to  WDIR**=**"idv-2.1" |
| PATCHES | name of the patches you download from [here.](https://platformsw.intel.com/ddrivers.aspx?kitnumber=135055)  https://platformsw.intel.com/ddrivers.aspx?kitnumber=135055  PATCHES**=**"idv2.1\_er5\_patchset" |

**Note: subject to change as URL/branch/patch files are changing**

Building kernel involves following steps.

1. Install the packages

The following package is needed to build kernel. It is possible that additional package is needed depends on the installed OS. The following is tested with Ubnuntu 18.04 installation with “minimal GUI” option.

|  |
| --- |
| Example of modules file Packages needed  # Install packages needed  **$ sudo** apt-get install **-**y liblz4-tool kernel-package libelf-dev build-essential git libfdt-dev libpixman-1-dev libssl-dev vim bc socat libsdl1.2-dev libspice-server-dev autoconf libtool xtightvncviewer tightvncserver x11vnc uuid-runtime uuid uml-utilities bridge-utils python-dev liblzma-dev libc6-dev libegl1-mesa-dev libepoxy-dev libdrm-dev libgbm-dev spice-client-gtk libgtk2.0-dev libusb-1.0.0-dev bison flex openssh-server net-tools |

1. Update the package and upgrade

It is recommended to update and upgrade the packages with latest.

|  |
| --- |
| # Update the packages and upgrade  **$ sudo** apt-get update **&&** **sudo** apt-get upgrade |

1. Pull the kernel tree with correct tag or branch

The following command will connect to $URL git kernel repository to pull the specific tag or branch indicated in $BRANCH into $WDIR. The following command will create $WDIR that user doesn’t have to create.

|  |
| --- |
| ### IDV 2.1 section 3.2.2 Build Kernel Source  **git** clone **--**depth 1 **$URL** **--**branch **$BRANCH** **--**single-branch **$WDIR** |

1. Apply the patches

|  |
| --- |
| # unzip download patched  # Unzip downloaded patches  **tar** xzvf **$CDIR/$PATCHES.**tar.gz  **cd** **$WDIR**  **git** apply **$CDIR/$PATCHES/\*** **&>/**dev**/**null |

1. Configure the kernel

|  |  |
| --- | --- |
| # Create configuration file  **echo** ""**|make** oldconfig  # Verify the kernel configuration is done correct or check the  # contents by opening the .config file using text editor  **egrep** 'CONFIG\_DRM\_I915\_GVT|CONFIG\_VFIO\_MDEV' **.**config  # Make sure you see the following output   |  | | --- | | Table Output of "egrep 'CONFIG\_DRM\_I915\_GVT|CONFIG\_VFIO\_MDEV' .config  egrep 'CONFIG\_DRM\_I915\_GVT|CONFIG\_VFIO\_MDEV' .config  dev@build:~/work/idv21/idv-2.1$ egrep 'CONFIG\_DRM\_I915\_GVT|CONFIG\_VFIO\_MDEV' .config  CONFIG\_DRM\_I915\_GVT=y  CONFIG\_DRM\_I915\_GVT\_KVMGT=m  CONFIG\_VFIO\_MDEV=m  CONFIG\_VFIO\_MDEV\_DEVICE=m  dev@build:~/work/idv21/idv-2.1$ | |

1. Compile the kernel source with overlay method

The following command will generate deb package with version and revision code.

|  |
| --- |
| # The following will compile the code can create deb package  **$ CONCURRENCY\_LEVEL=`nproc`** fakeroot make-kpkg **--**initrd **--**append-to-version**=-**intelgvt **--**revision 2**.**1.0 **--**overlay-dir**=$cdir/**ubuntu-package kernel\_image kernel\_headers |

linux-image-4.19.65-intelgvt+\_2.1.0\_amd64.deb

linux-headers-4.19.65-intelgvt+\_2.1.0\_amd64.deb

* 1. –append-to-version=-intelgvt
  2. –revisioin 2.1.0

Where 2.1 is IDV package version and 0 is engineering release version

## Install and configure kernel

### Installation of the kernel

The following commands will install “linux-image-4.19.65-intelgvt+\_2.1.0\_amd64.deb” and “

linux-headers-4.19.65-intelgvt+\_2.1.0\_amd64.deb”.

|  |
| --- |
| # the following command will install the kernel  **cd** **$CDIR**  **sudo** dpkg **-**i **\*.**deb |

### Configure grub to boot newly installed kernel

In Linux system, you can have multiple revisions of OS that you can selectively boot from. This example shows how GRUB can be used to boot. The default GRUB file is located in “/etc/default/grub”. We are interested in two options.

|  |  |
| --- | --- |
| **GRUB\_DEFAULT** | Default OS version to be boot from |
| **GRUB\_CMDLINE\_LINUX\_DEFAULT** | Options to pass to OS |

1. Edit the “/etc/default/grub” as root and make modification to the file as below.

|  |
| --- |
| # to modify the /etc/default/grub as root  **$ sudo** nano **/**etc**/**default**/**grub |

Modify the “GRUB\_DEFAULT” and “GRUB\_CMDLINE\_LINUX\_DEFAULT”

|  |
| --- |
| GRUB\_DEFAULT**=**"Advanced options for Ubuntu>Ubuntu, with Linux 4.19.65-intelgvt+"  GRUB\_CMDLINE\_LINUX\_DEFAULT**=**"splash quiet i915.enable\_gvt=1 i915.enable\_fbc=0 kvm.ignore\_msrs=1 intel\_iommu=on,igfx\_off drm.debug=0" |

Note: 4.19.65-intelgvt is part of the kernel deb file name. Upon installation of the kernel, this string should be in /boot/grub/grub.cfg file.

1. Apply the default grub setting

Must run the following command to have it take new rules. It will take effect on next reboot

|  |
| --- |
| # to modify the /etc/default/grub as root  **$ sudo** update-grub2 |

1. Verification of the new kernel after **reboot**

Upon reboot, all GRUB setting should take effect and boot to new kernel

|  |
| --- |
| # reboot system  **$ sudo** reboot |

|  |
| --- |
| # Check the kernel and options  **$ cat** **/**proc**/**cmdline  # Please note that “BOOT\_IMAGE” reflects the compiled and installed image and options  $ cat /proc/cmdline  BOOT\_IMAGE=/boot/vmlinuz-4.19.65-intelgvt+ root=UUID=baea869f-283c-483e-b29e-8ea87927b48c ro splash quiet i915.enable\_gvt=1 i915.enable\_fbc=0 kvm.ignore\_msrs=1 intel\_iommu=on,igfx\_off drm.debug=0 vt.handoff=1 |

### GVTg Verification

#### Check for “mdev\_supported\_types “

Once reboot with correct build, grub configuration, and reboot, “medv\_supported\_types” directory should exist. Depends on your aperture size, you will see one or more of the following directories under “mdev\_supported\_types”.

* i915\_GVTg\_V5\_1
* i915\_GVTg\_V5\_2
* i915\_GVTg\_V5\_4
* i915\_GVTg\_V5\_8

|  |  |
| --- | --- |
| # Check for mdev\_supported\_types directory  **$ ll** **/**sys**/**bus**/**pci**/**devices**/**0000**:**00**:**02**.**0**/**mdev\_supported\_types**/**  # The output of “ll /sys/bus/pci/devices/0000:00:02.0/mdev\_supported\_types/”   |  | | --- | | Table Output of “ll” command for mdev\_supported\_type directory  dev@gvt:~$ ll /sys/bus/pci/devices/0000:00:02.0/mdev\_supported\_types/  total 0  drwxr-xr-x 6 root root 0 Feb 28 13:09 ./  drwxr-xr-x 14 root root 0 Feb 24 14:55 ../  drwxr-xr-x 3 root root 0 Feb 24 14:55 i915-GVTg\_V5\_1/  drwxr-xr-x 3 root root 0 Feb 24 14:55 i915-GVTg\_V5\_2/  drwxr-xr-x 3 root root 0 Feb 24 14:55 i915-GVTg\_V5\_4/  drwxr-xr-x 3 root root 0 Feb 24 14:55 i915-GVTg\_V5\_8/  dev@gvt:~$ | |

##### i915\_GVTg\_V5\_1

|  |  |
| --- | --- |
| # Check for mdev\_supported\_types directory  **$ cat** **/**sys**/**bus**/**pci**/**devices**/**0000**:**00**:**02**.**0**/**mdev\_supported\_types**/i915\_GVTg\_V5\_1**  # Output of above command   |  | | --- | | Table Output of Description of node i915\_GVTg\_V5\_1  giga@giga-0t:~$ cat /sys/bus/pci/devices/0000:00:02.0/mdev\_supported\_types/i915\_GVTg\_V5\_1/description  low\_gm\_size: 512MB  high\_gm\_size: 2048MB  fence: 4  resolution: 1920x1200  weight: 16 | |

##### i915\_GVTg\_V5\_2

|  |  |
| --- | --- |
| # Check for mdev\_supported\_types directory  **$ cat** **/**sys**/**bus**/**pci**/**devices**/**0000**:**00**:**02**.**0**/**mdev\_supported\_types**/i915\_GVTg\_V5\_2**  # Output of above command   |  | | --- | | Table Output of Description of node i915\_GVTg\_V5\_2  giga@giga-0t:~$ cat /sys/bus/pci/devices/0000:00:02.0/mdev\_supported\_types/i915\_GVTg\_V5\_2/description  low\_gm\_size: 512MB  high\_gm\_size: 2048MB  fence: 4  resolution: 1920x1200  weight: 16 | |

##### i915\_GVTg\_V5\_4

|  |  |
| --- | --- |
| # Check for mdev\_supported\_types directory  **$ cat** **/**sys**/**bus**/**pci**/**devices**/**0000**:**00**:**02**.**0**/**mdev\_supported\_types**/i915\_GVTg\_V5\_4**  # Output of above command   |  | | --- | | Table Output of Description of node i915\_GVTg\_V5\_4  giga@giga-0t:~$ cat /sys/bus/pci/devices/0000:00:02.0/mdev\_supported\_types/i915\_GVTg\_V5\_4/description  low\_gm\_size: 512MB  high\_gm\_size: 2048MB  fence: 4  resolution: 1920x1200  weight: 16 | |

##### i915\_GVTg\_V5\_8

|  |  |
| --- | --- |
| # Check for mdev\_supported\_types directory  **$ cat** **/**sys**/**bus**/**pci**/**devices**/**0000**:**00**:**02**.**0**/**mdev\_supported\_types**/i915\_GVTg\_V5\_8**  # Output of above command   |  | | --- | | Table Output of Description of node i915\_GVTg\_V5\_8  giga@giga-0t:~$ cat /sys/bus/pci/devices/0000:00:02.0/mdev\_supported\_types/i915\_GVTg\_V5\_8/description  low\_gm\_size: 512MB  high\_gm\_size: 2048MB  fence: 4  resolution: 1920x1200  weight: 16 | |

# QEMU build

## Install qemu on LTS 20.04

If you have LTS 20.04 as base OS then use method in this section.

1. Install qemu package for LTS 20.04. It will install qemu and BIOS.bin file.

|  |
| --- |
|  |

1. Copy the bios.bin and OVMF.fd file to “vm” work direcotry

|  |
| --- |
| **cp** **/**usr**/**share**/**qemu**/**bios.bin **~/**vm**/**fw  **cp** **/**usr**/**share**/**qemu**/**OVMF.fd **~/**vm**/**fw |

## Install QEMU on LTS 18.04

If you are using 18.04 and follow the method in this section.

### Install Package to build QEMU

|  |
| --- |
| # Install packages to build QEMU  **$ sudo** apt install **-**y git libfdt-dev libpixman-1-dev libssl-dev vim socat libsdl2-dev libspice-server-dev autoconf libtool xtightvncviewer tightvncserver x11vnc uuid-runtime uuid uml-utilities bridge-utils python-dev liblzma-dev libc6-dev libegl1-mesa-dev libepoxy-dev libdrm-dev libgbm-dev libaio-dev libusb-1.0.0-dev libgtk-3-dev bison |

### Download, configure QEMU, compile to install

|  |
| --- |
| # Pull the tree  **$ wget** https://download.qemu.org/**qemu-4.2.0.**tar.xz  # Un-compress the tar ball file  **$ tar** xzvf qemu-4.2.2.tar.gz  # change directory to qemu source tree  **$ cd** qemu-4.2.2  # pull submodules <optional>  **$ git** submodule update **--**init roms**/**seabios  # configure the QEMU  **$ ./**configure **--**prefix**=/**usr **\**  **--**enable-kvm **\**  **--**disable-xen **\**  **--**enable-libusb **\**  **--**enable-debug-info **\**  **--**enable-debug **\**  **--**enable-sdl **\**  **--**enable-vhost-net **\**  **--**enable-spice **\**  **--**disable-debug-tcg **\**  **--**enable-opengl **\**  **--**enable-gtk **\**  **--**target-list**=**x86\_64-softmmu **\**  **--**audio-drv-list**=**pa  # make source tree  **$ make** **-**j **`nproc`**  # build bios.bin (optional)  **$ cd** roms**/**seabios  **$ make** **-**j**`nproc`**  **$ cd** **-**  # Install the QEMU  **sudo** make install  # copy the bios.bin to vm/fw directory (optional)  **$ cp** roms**/**seabios**/**out**/**bios.bin **~/**vm**/**fw |

-------------------------------------------------------------------------------

The kernel was built with IDV patches. User can verify the GVTg.

Once should be able to see “gvt\_disp\_auto\_switch”, “gvt\_disp\_edid\_filter”, “gvt\_disp\_ports\_mask”, “gvt\_disp\_ports\_owner”, and “gvt\_disp\_ports\_status”.

# Putting it together

## Recommended directory structure

Recommend the following directory setup as below for easy maintenance. This document and scripts as example assume the following directory is exist and refers to them. However, one can change.

|  |  |
| --- | --- |
| **Directory** | **Note** |
| vm | parent directory consisting fw, disk, iso, and script |
| fw | directory where firmware is located |
| disk | directory where collection of qcow2 files |
| iso | directory where IOS image |
| script | directory where all scripts are located |

|  |
| --- |
| The recommended directory structures. Please see rest of document for explanation. |

### “disk” directory

The “disk” directory is where guest OS is located in “qcow2” file format.

|  |
| --- |
| # create qcow2 file where guest OS to be installed  **$ qemu-**img create -f qcow2 ubuntu18.qcow2 60G  **$ qemu-**img create -f qcow2 win10.qcow2 60G |

### “fw” directory

The “fw” directory is where BIOS and UEFI firmware is located.

The bios.bin is results of part of QEMU build. Please see 3.2.2.

The requirements of bios.bin is depends on the guest OS. If the guest OS is needs UEFI, then the bios.bin is not needed.

|  |
| --- |
| # 1) The following link has the pre-build OVMF files where you can download  # <https://www.kraxel.org/repos/jenkins/edk2/>    #-----------------------------------------------------  # 2) Download the pre-build OVMF rpm package. Please note that the file name  # changes as it gets updated. If it fails, you can select the correct file name # by viewing following link on web  # “https://www.kraxel.org/repos/jenkins/edk2/”  #-----------------------------------------------------  **$ wget** https://www.kraxel.org/repos/jenkins/edk2/edk2.git-ovmf-x64-0-20191118.1337.g2be4828af1.noarch.rpm  #-----------------------------------------------------  # 3) Unpack the rpm package to current direction without installing them  #-----------------------------------------------------  **$ rpm2cpio** edk2.git-ovmf-x64-0-20191118.1337.g2be4828af1.noarch.rpm **|** **cpio** **-**idmv  $ rpm2cpio edk2.git-ovmf-x64-0-20191118.1337.g2be4828af1.noarch.rpm | cpio -idmv  ./usr/share/doc/edk2.git-ovmf-x64  ./usr/share/doc/edk2.git-ovmf-x64/README  ./usr/share/edk2.git  ./usr/share/edk2.git/ovmf-x64  ./usr/share/edk2.git/ovmf-x64/OVMF-need-smm.fd  ./usr/share/edk2.git/ovmf-x64/OVMF-pure-efi.fd 🡨 Tried and worked  ./usr/share/edk2.git/ovmf-x64/OVMF-with-csm.fd 🡨 Tried and worked  ./usr/share/edk2.git/ovmf-x64/OVMF\_CODE-need-smm.fd  ./usr/share/edk2.git/ovmf-x64/OVMF\_CODE-pure-efi.fd  ./usr/share/edk2.git/ovmf-x64/OVMF\_CODE-with-csm.fd  ./usr/share/edk2.git/ovmf-x64/OVMF\_VARS-need-smm.fd  ./usr/share/edk2.git/ovmf-x64/OVMF\_VARS-pure-efi.fd  ./usr/share/edk2.git/ovmf-x64/OVMF\_VARS-with-csm.fd  ./usr/share/edk2.git/ovmf-x64/UefiShell.iso  ./usr/share/qemu/firmware/80-ovmf-x64-git-needs-smm.json  ./usr/share/qemu/firmware/81-ovmf-x64-git-pure-efi.json  ./usr/share/qemu/firmware/82-ovmf-x64-git-with-csm.json  $  #-----------------------------------------------------  # 4) Copy the OVM files to “fw” directory  #-----------------------------------------------------  **$ cp** **./**usr**/**share**/**edk2.git**/**ovmf-x64**/**OVMF-pure-efi.fd **~/**vm**/**fw/OVMF.fd  **$ cp** **./**usr**/**share**/**edk2.git**/**ovmf-x64**/**OVMF-with-csm.fd **~/**vm**/**fw/OVMF-csm.fd |

### “iso” directory

Please consult your local IT for a copy of Windows 10 ISO file

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Please where you can download the IOS file.   |  |  | | --- | --- | | **OS** | **Link** | | **Ubuntu 18.04** | <https://ubuntu.com/download/desktop> | | **Android x86** | <https://osdn.net/projects/android-x86/releases> | |

## Available GVTg control node

|  |  |
| --- | --- |
| # Check the GVTg control node.  **$ cat** **/sys/class/drm/card0/gvt\***  # Output should be as below   |  | | --- | | Table Output of "ls /sys/class/drm/card0/gvt\*"  $ ls /sys/class/drm/card0/gvt\*  /sys/class/drm/card0/gvt\_disp\_auto\_switch  /sys/class/drm/card0/gvt\_disp\_edid\_filter  /sys/class/drm/card0/gvt\_disp\_ports\_mask  /sys/class/drm/card0/gvt\_disp\_ports\_owner  /sys/class/drm/card0/gvt\_disp\_ports\_status | |

### Explanation of gvt\_disp\_ports\_status

Shows current status of GVTg.

The following example indicates there are 3 ports available and mapped.

|  |
| --- |
| # Check the port status of GVTg  **$ cat** **/**sys**/**class**/**drm**/**card0**/**gvt\_disp\_ports\_status  $ cat /sys/class/drm/card0/gvt\_disp\_ports\_status  Auto host/vGPU display switch: Y  Available display ports: 0x00004320 **🡨 Ports available**  ( PORT\_B(2) )  ( PORT\_C(3) )  ( PORT\_D(4) )  Display ports assignment: 0x0000000000080402 **🡨 Currently mapped**  Each byte represents the bit mask of assigned port(s) to vGPU 1-8 (low to high)  Bit mask is decoded from LSB to MSB (PORT\_A to PORT\_A+7):  0: This port isn't assigned to that vGPU.  1: This port is assigned to that vGPU.  vGPU-1 port mask: (0x02)  ( PORT\_B(2) )  vGPU-2 port mask: (0x04)  ( PORT\_C(3) )  vGPU-3 port mask: (0x08)  ( PORT\_D(4) )  Display ports ownership: 0x00000000 **🡨 Ownership of the display port**  x on n-th hex-digit: (port\_name(port\_id#) <---> owner)  n: port id, PORT\_A(1), PORT\_B(2), ...  x: owner: host (0), vGPU id(x)  ( PORT\_B(2) <---> Host )  ( PORT\_C(3) <---> Host )  ( PORT\_D(4) <---> Host ) |

### Explanation of gvt\_disp\_ports\_mask

“gvt\_disp\_ports\_mask” can accommodate up to 8 physical monitor assignment each with 8 bits to indicates graphics ports. The vGPU(x) is mapped using bitmapped definition.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| IDV 2.1 gvt\_disp\_ports\_mask definition – each xx is OR-able 8bit   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **GPU index** | **vGPU7** | **vGPU6** | **vGPU5** | **vGPU4** | **vGPU3** | **vGPU2** | **vGPU1** | **vGPU0** | | **Pipe ownership mapping** | xx | xx | xx | xx | xx | xx | xx | xx |   Where xx is defined as below and can be OR-ed   |  |  |  | | --- | --- | --- | | **PORT** | **BITMASK** | **HEX** | | **PORT A (EDP)** | 0000 0001 | 01 | | **PORT B** | 0000 0010 | 02 | | **PORT C** | 0000 0100 | 04 | | **PORT D** | 0000 1000 | 08 |   Mapping port B to vGPU0, port C to vGPU1, and port D to vGPU2   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **GPU index** | **vGPU7** | **vGPU6** | **vGPU5** | **vGPU4** | **vGPU3** | **vGPU2** | **vGPU1** | **vGPU0** | | **Pipe ownership mapping** | xx | xx | xx | xx | xx | 08 | 04 | 02 |   Mapping port B & C to vGPU0, and port D to vGPU2   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **GPU index** | **vGPU7** | **vGPU6** | **vGPU5** | **vGPU4** | **vGPU3** | **vGPU2** | **vGPU1** | **vGPU0** | | **Pipe ownership mapping** | xx | xx | xx | xx | xx | xx | 08 | 06 | |

### Explanation of gvt\_disp\_ports\_owner

This feature is implemented to accommodate single monitor with VM and AIC-CIC running at same time by giving framebuffer switching. The detail will be TBD.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Gvt\_disp\_port\_ownership is total 32bit field with each nibble represents on vGPU.   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **GPU index** | **vGPU7** | **vGPU6** | **vGPU5** | **vGPU4** | **vGPU3** | **vGPU2** | **vGPU1** | **vGPU0** | | **gvt\_disp\_port\_ownership** | x | x | x | x | x | x | x | x |   Each nibble can be either 1 or 0. The nibble will be set to “1” to guest OS while guest OS is running. It will be set to “0” when guest OS exist. |

## Network Preparation

For network configuration for VM, NAT method will used in this case.

**NAT mode**

Create a “**vm/script/qemu-ifup-nat**” file and past the script indicated in the <https://wiki.qemu.org/Documentation/Networking/NAT>.

## Graphics pass through

### Create UUID (GUID)

Need to use unique ID to logically connect to virtual graphics pipe. “uuidgen” will be used. Generate as many as UUID needed. One UUID for one graphics port. If the system has 3 port/monitor, you will need 3 UUID. Repeat the “uuidgen” to get additional UUID (GUID).

|  |
| --- |
| # Generate unique id with uuidgen util  **$ uuidgen**  $ uuidgen  d09054ed-30cd-474f-b662-10be479ac004 |

### Masking the ports for VGPU

The port needs to be logically assigned by writing “gvt\_disp\_ports\_mask”. The 0x0000000000080402 correspondent to the UUID writes to the “create”.

|  |
| --- |
| # for IDV 2.1 or later. See section 4.2.2  **$ sudo** echo 0x00000000000080402 **>** **/**sys**/**class**/**drm**/**card0**/**gvt\_disp\_ports\_mask |

### Assigning the port/s to vGPU using UUID

The lspci shows that the system has GFX on pci bus:0, dev:0, func:2.

Please refer to 4.2.2.

|  |
| --- |
| # The first occurrence of echoing UUID to create will assign UUID to port number 2 in this example (per “gvt\_disp\_ports\_mask” value)  **$ sudo** echo "d09054ed-30cd-474f-b662-10be479ac004" **>** **/**sys**/**bus**/**pci**/**devices**/**0000\:00\:02**.**0**/**mdev\_supported\_types**/**i915-GVTg\_V5\_4**/**create  # The second occurrence of echoing UUID to create will assign UUID to port number 4 in this example (per “gvt\_disp\_ports\_mask” value)  **$ sudo** echo "e49e8cb2-0720-47d0-b035-081ac135d216" **>** **/**sys**/**bus**/**pci**/**devices**/**0000\:00\:02**.**0**/**mdev\_supported\_types**/**i915-GVTg\_V5\_4**/**create  # The third occurrence of echoing UUID to create will assign UUID to port number 8 in this example (per “gvt\_disp\_ports\_mask” value)  **$ sudo** echo "6f620412-8c42-42df-a798-a150c97bcfc1" **>** **/**sys**/**bus**/**pci**/**devices**/**0000\:00\:02**.**0**/**mdev\_supported\_types**/**i915-GVTg\_V5\_4**/**create |

### De-coupling the ports

The ports assigned to vGPU can be de-assigned by using the following command.

|  |
| --- |
| # de-couple ports from the vGPU by writing "1" to "remove" node as below  **$ sudo** echo 1 **>** **/**sys**/**bus**/**pci**/**devices**/**0000\:00\:02**.**0**/**d09054ed-30cd-474f-b662-10be479ac004**/**remove  **$ sudo** echo 1 **>** **/**sys**/**bus**/**pci**/**devices**/**0000\:00\:02**.**0**/**e49e8cb2-0720-47d0-b035-081ac135d216**/**remove  **$ sudo** echo 1 **>** **/**sys**/**bus**/**pci**/**devices**/**0000\:00\:02**.**0**/**6f620412-8c42-42df-a798-a150c97bcfc1**/**remove |

# Installing guest OS

Need to clarify (TBD)

The following script will start the guest OS on assigned ports per UUID (GUID) when done correctly. In this example, note the high-lighted UUID (GUID). While the OS is installing, you can connect to the installation process using VNC using port 5900.

1. Run the following script to install guest OS

|  |
| --- |
| #!/bin/bash -x  /usr/bin/qemu-system-x86\_64 \  -m 4096 -smp 1 -M q35 \  -enable-kvm \  -name ubuntu-guest \  -hda ~/vm/disk/ubuntu.qcow2 \  -cdrom ~/vm/iso/mini.iso \  -nic user,model=e1000,mac=00:DE:AD:BE:EF:47 \  -vga qxl \  -k en-us \  -vnc :0 \  -machine q35,kernel\_irqchip=on \  -cpu host -usb -device usb-tablet \  -device vfio-pci,sysfsdev=/sys/bus/pci/devices/0000:00:02.0/d09054ed-30cd-474f-b662-10be479ac004,display=off,x-igd-opregion=on |

1. Start VNC client port 5900 to interact with installation process

## Start VM with USB device

Script here is example where you use it as template. Need to make modification to the script per your system specific.

**-**device usb-host**,**vendorid**=**0x045e**,**productid**=**0x0800 **\**

vendorid and productid is your USB device to assign to guest OS such as keyboard.

Type “lsusb” to find the vendorid and productid.

|  |
| --- |
| #!/bin/bash -x  **/**usr**/**bin**/**qemu-system-x86\_64 **\**  **-**m 4096 **-**smp 1 -M q35 **\**  **-**enable-kvm **\**  **-**name ubuntu-guest **\**  **-**hda .**/**vm**/**disk**/**ubuntu.qcow2 **\**  **-**nic user**,**model**=**e1000**,**mac**=**00**:**DE**:**AD**:**BE**:**EF**:**47 **\**  **-**vga none **\**  **-**display egl-headless **\**  -k en-us **\**  **-**vnc **:**0 **\**  **-**machine q35**,**kernel\_irqchip**=**on **\**  **-**cpu host **-**usb **-**device usb-tablet **\**  **-**device usb-host**,**vendorid**=**0x045e**,**productid**=**0x0800 **\**  **-**device usb-host**,**vendorid**=**0x045e**,**productid**=**0x07b2 **\**  **-**audiodev pa**,**id**=**snd0**,**server**=/**run**/**user**/**1000**/**pulse**/**native **\**  **-**soundhw hda **\**  **-**device vfio-pci**,**sysfsdev**=/**sys**/**bus**/**pci**/**devices**/**0000**:**00**:**02**.**0**/**404d5798-c07d-11e8-89ef-63dff7a20fd0**,**display**=**off**,**x-igd-opregion**=**on |

## Advanced topic

TDB

### Network in NAT mode

Formatting option for QEMU to add custom network. Make sure you have qemu-ifup-nat file available.

Refer to 4.3.

|  |
| --- |
| MAC**=**"00:DE:AD:BE:EF:F1"  IFUP**=./**script**/**qemu-ifup-nat  **-**net nic**,**macaddr**=$MAC** **-**net tap**,**script**=$IFUP,**downscript**=**no |

### Utilization of system

TBD

### Display Switch between Host and VM

By default, the connected display monitor will show content of assigned VM once VGPU  
is activated and switch back to host after VGPU is deactivated. To switch display  
between host and VM on-the-fly while VM is still running, “gvt\_disp\_port\_owner” can be used. The input value must be in hexadecimal. Refer to section 4.2.3.

Feature for IDV 3.0 only

TBD, add example.

### Device pass-through/assignment to guest OS

#### Passing USB by Bus and Device

The USB device passed to guest OS using Bus and Device is not hot pluggable. The Bus and Device gets assigned whenever USB enumeration happens.

1. Find the USB device to pass through/assign to guest OS

|  |
| --- |
| #--------------------------------------------------------------  # "lsusb" command will list the USB device has been enumerated  #--------------------------------------------------------------  **$ lsusb**  $ lsusb  Bus 003 Device 002: ID 046d:c52b Logitech, Inc. Unifying Receiver  $ |

1. Formation option for QEMU

Based on the information from “lsusb”, the Unifying Receiver (keyboard and mouse) is enumerated with Bus: 3 and Device 2. The following QEMU option needs to be added as parameter.

|  |
| --- |
| **-**device usb-host**,hostbus=3,hostaddr=2** |

#### Passing USB by vendor and product ID

This method provides limited hot pluggable. When there is multiple device in a system with same vendor and product ID (e.g. same manufacture and model), the behavior is not guaranteed in such case.

1. Find the USB vendor and product ID

|  |
| --- |
| #--------------------------------------------------------------  # "lsusb" command will list the USB device has been enumerated  #--------------------------------------------------------------  **$ lsusb**  $ lsusb  Bus 003 Device 002: ID 046d:c52b Logitech, Inc. Unifying Receiver  $ |

1. Formation option for QEMU

Based on the information from “lsusb”, the device being pass through/assign to guest OS has vendor and product id as 046d:c52b. The following QEMU option needs to be added as parameter.

|  |
| --- |
| **-**device usb-host**,**vendorid**=**0x046d**,**productid**=**0xc52b |

#### Passing by USB port

This method provides full hot pluggable feature. It passes through the USB port to guest OS rather than USB device.

1. Find USB port definition

|  |
| --- |
| #--------------------------------------------------------------  # "lsusb -t" command will show USB port information  # The port information shows how hardware is wired.  #--------------------------------------------------------------  **$ lsusb** -t  $ lsusb -t  /: Bus 01.Port 1: Dev 1, class="root\_hub", Driver=xhci\_hcd/12p, 480M  |\_\_ Port 3: Dev 3, If 0, class="Hub", Driver=hub/4p, 480M  |\_\_ Port 4: Dev 18, If 0, class="Human" Interface Device, Driver=usbhid, 1.5M  |\_\_ Port 3: Dev 17, If 0, class="Human" Interface Device, Driver=usbhid, 1.5M  $ |

1. Formation option for QEMU

Based on the information from “lsusb -t”, each USB port has assigned port number. The given USB port is passed and permanently assigned to guest OS for duration of its life. The following QEMU option needs to be added as parameter.

Note: the hostbus and hostport is based on the “lsusb -t” output.

|  |
| --- |
| **-**device usb-host**,**hostbus**=**1**,**hostport**=**3**.**3  **-**device usb-host**,**hostbus**=**1**,**hostport**=**3**.**4 |

#### Passing by PCI device using vfio-pci

QEMU options allow entire PCI device to be pass-through/assigned to guest OS. This process requires binding the PCI devices to IOMMU. Make sure the system BIOS has IOMMU support for VT-x and VT-d. This method need PCI vendor and device identification specified by PCI-SIG organization. Make sure “intel\_iommu=on” in the command line option in GRUB and “vfio-pci” is in the module (2.1.1).

1. Find PCI device to pass-through

|  |
| --- |
| #--------------------------------------------  # "lspci" command to find Bus/Dev/Func  #--------------------------------------------  **$ lspci**  $ lspci  01:00.0 USB controller: Renesas Technology Corp. uPD720201 USB 3.0 Host Controller (rev 03)  $  Note: The particular PCI device is Bus:01, Dev:00, and Func:0. |

1. Find the PCI unique device identification for Bus/Dev/Func

|  |
| --- |
| #------------------------------------------  # Read the PCI device unique vendor ID  #------------------------------------------  **$ cat** **/**sys**/**bus**/**pci**/**devices**/**"0000:01:00.0"**/vendor**  $ cat /sys/bus/pci/devices/"0000:01:00.0"/vendor  0x1912  $  #------------------------------------------  # Read the PCI device unique device ID  #------------------------------------------  **$ cat** **/**sys**/**bus**/**pci**/**devices**/**"0000:01:00.0"**/device**  $ cat /sys/bus/pci/devices/"0000:01:00.0"/device  0x0014  $  Note: The PCI device vendor and device ID is for Bus:01, Dev:00, and Func:0. |

1. Bind the PCI device to IOMMU

|  |
| --- |
| #------------------------------------------  # Bind PCI to IOMMU controller  # Note: $vendor is 0x1912, and $device is 0x0014  #------------------------------------------  **$** echo **$vendor** **$device** **|** **sudo** tee **/**sys**/**bus**/**pci**/**drivers**/**vfio-pci**/**new\_id |

1. Formation option for QEMU

Based on the information above, the Bus/Dev/Func will be used to assign.

|  |
| --- |
| **-**device vfio-pci**,**sysfsdev**=/**sys**/**bus**/**pci**/**devices**/**0000**:**00**:**02**.**0**/**01**:**00**.**0**,**x-igd-opregion**=**on |

#### Passthrough MISC devices

Collection of information from other team cross the division

|  |
| --- |
| sudo usermod -aG kvm <user>  sudo usermod -aG dialout <user> |

|  |
| --- |
| -audiodev pa,id=snd0,server=/run/user/1000/pulse/native \  -device ich9-intel-hda \  -device hda-duplex,audiodev=snd0 \  *-parallel /dev/parportN*                # replace N with the right number  -serial /dev/ttyS0 \  -serial /dev/ttyS1 \  -chardev tty,id=com4,path=/dev/ttyS4 \  -chardev tty,id=com5,path=/dev/ttyS5 \  -chardev tty,id=com6,path=/dev/ttyS6 \  -chardev tty,id=com7,path=/dev/ttyS7 \  -chardev tty,id=com8,path=/dev/ttyS8 \  -chardev tty,id=com9,path=/dev/ttyS9 \  -chardev tty,id=com10,path=/dev/ttyS10 \  -device pci-serial-4x,chardev1=ser2,chardev2=ser2,chardev3=ser3,chardev4=ser4 \  -device usb-serial,chardev=com4 \  -device usb-serial,chardev=com5 \  -device usb-serial,chardev=com6 \  -device usb-serial,chardev=com7 \  -device usb-serial,chardev=com8 \  -device usb-serial,chardev=com9 \  -device usb-serial,chardev=com10 \ |

### Headless VM setup for Spice

Example script to start the headless guest OS

|  |
| --- |
| #! /bin/bash -x  NAME**=**"Ubuntu-Guest"  #VGPU="f50aab10-7cc8-11e9-a94b-6b9d8245bfc1"  MAC**=**"00:DE:AD:BE:EF:20"  HDD**=./**disk**/**ubuntu-1.qcow2  ISO**=./**iso**/**ubuntu.iso  BIOS**=./**fw**/**bios.bin  IFUP**=./**script**/**qemu-ifup-bridge  DISPLAY**=**"on"  VGA**=**"qxl"  **$EGL=**"-display egl-headless"  **/**usr**/**bin**/**qemu-system-x86\_64 **\**  **-**m 2048 **-**smp 2 -M pc **\**  **-**name **$NAME** **\**  **-**hda **$HDD** **\**  **-**cdrom **$ISO** **\**  **-**bios **$BIOS** **-**enable-kvm **\**  **-**net nic**,**macaddr**=$MAC** **-**net tap**,**script**=$IFUP,**downscript**=**no **\**  **-**vga **$VGA** **$EGL** **\**  -k en-us **\**  **-**serial stdio **\**  **-**spice port**=**5900**,**disable-ticketing **\**  **-**machine kernel\_irqchip**=**on **\**  **-**global PIIX4\_PM.disable\_s3**=**1 **-**global PIIX4\_PM.disable\_s4**=**1 **\**  **-**cpu host **-**usb **-**device usb-tablet **\** |

### Automating VM deployment using Portainer

TBD

### Automating VM deployment using LibVirt

TBD

# Remote management using MeshCentral

This section is used to remotely manage the computer might be at remote location such as customer site.

## Basic usage of MeshCentral

1. Create account with MeshCentral

Create an account by visiting [MeshCentral](http://www.meshcentral.com)

|  |
| --- |
|  |

1. Create a group

Using icons on left column, go to My Account and click “new”

|  |
| --- |
|  |

1. Add machine you want to gain access

|  |
| --- |
| # click "My Device" icon on left column    # Look for newly created "Sample-Group" and click "Add Agent"    # Look for newly created "Sample-Group" and click "Add Agent"  # select the Operating System installed on remote computer  # Copy the high-lighted command and paste at remote machine to run |

Note: If done correctly, remote machine name should pop-up under the “Sample-Group”. Click it to explore (e.g. Terminal or Files)

## Advance features with MeshCentral

Information was distributed by Joko Sastriawan to gain access to remote machine via MeshCentral using commonly known tools such as “SSH” and “SFTP”.

|  |
| --- |
|  |

1. Install node.js (<https://www.guru99.com/download-install-node-js.html>)
2. Download mc2router-master from github as zip file (<https://github.com/jsastriawan/mc2router>)
3. Unzip the downloaded file
4. Type "npm install nw -g"
5. npm i nw -g --nwjs\_build\_type=sdk
6. Type "nw" to start
7. Navigate to “File->Configure”
   1. Click “proxy”
   2. Enter your local proxy information if needed
   3. Select the “SSH” tool location
   4. Select the “SFTP” tool location
   5. Select the “RDP” tool location
   6. Select the “Label” in “Custom tool” section (optional)
   7. Select the “Path” to custom VNC viewer
   8. Add “Arguments”
   9. Add “Port”
8. Click “Save”
9. Click “File->Reset” to log on with your MeshCentral.com account

Note: If done correctly, you should be able to log on to remote machine using SSH. Make sure SSH server is running at remote machine.

# Addendum

Addendum is subject to change/delete. It should be treated as template/example. Customer supporting team should make appropriate change if chose to use.

## Scripts to pull and build kernels

“idv.sh script”

|  |
| --- |
| #!/bin/bash -e  ###################################################################  # version : version of the idv.sh file  # cdir : pionts to current working directory where idv.sh file runs  # URL : URL to kernel source  # BRANCH : tag or branch of the kernel source  # PATCHES : idv patch file name with our ".tar.gz" extention  ###################################################################  version**=**"0.5"  cdir**=$(pwd)**  URL**=**"https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git"  BRANCH**=**"v4.19.65"  W\_DIR**=**"idv-2.1"  PATCHES**=**"idv2.1\_er5\_patchset"  REVISION**=**"2.1"  # Display version of idv.sh file  **echo** "idv.sh - version ${version}"  # delete all files directorys created by executing idv.sh  # Usage: $ ./idv.sh clean  **function** clean**()** **{**  **rm** **-**rf **$cdir/$W\_DIR**  **rm** **-**rf **$cdir/$PATCHES**  **rm** **-**rf **$cdir/**ubuntu-package  **rm** **-**rf **$cdir/\*.**deb  **}**  **[[** "$1" **==** "clean" **]]** **&&** clean **&&** **exit** 0  #================================================  # Install packages to compile the kernel  #================================================  **sudo** -s <<RUNASSUDO\_PACKAGE  **apt-get** autoremove **-**y **&>/**dev**/**null  **apt-get** install **-**y liblz4-tool kernel-package libelf-dev build-essential git libfdt-dev libpixman-1-dev libssl-dev vim bc socat libsdl1.2-dev libspice-server-dev autoconf libtool xtightvncviewer tightvncserver x11vnc uuid-runtime uuid uml-utilities bridge-utils python-dev liblzma-dev libc6-dev libegl1-mesa-dev libepoxy-dev libdrm-dev libgbm-dev spice-client-gtk libgtk2.0-dev libusb-1.0.0-dev bison flex openssh-server net-tools **&>/**dev**/**null  RUNASSUDO\_PACKAGE  #================================================  # Source of Information  # https://wiki.ubuntu.com/KernelTeam/GitKernelBuild  # IDV 2.1 section 3.2.1 Build Initial Ramdisk  #================================================  modules**=(**kvmgt vfio-iommu-type1 vfio-mdev vfio-pci**)**  **for** i **in** "${modules[@]}"  **do**  **echo** **$i**  **sudo** -s <<RUNASSUDO\_MODULES  **grep** **-**qxF **$i** **/**etc**/**initramfs-tools**/**modules **||** **echo** **$i** **>>** **/**etc**/**initramfs-tools**/**modules  RUNASSUDO\_MODULES  **done**  **cp** -a **/**usr**/**share**/**kernel-package **$cdir/**ubuntu-package  ### IDV 2.1 section 3.2.2 Build Kernel Source  **[[** **!** **-d** "$cdir/$W\_DIR" **]]** **&&** **git** clone **--**depth 1 **$URL** **--**branch **$BRANCH** **--**single-branch **$W\_DIR**  **tar** xzvf **$cdir/$PATCHES.**tar.gz  **cd** **$W\_DIR**  **git** reset **--**hard  **git** apply **$cdir/$PATCHES/\*** **&>/**dev**/**null  **echo** ""**|make** oldconfig  CONCURRENCY\_LEVEL**=`nproc`** fakeroot make-kpkg **--**initrd **--**append-to-version**=-**intelgvt **--**revision **$REVISION** **--**overlay-dir**=$cdir/**ubuntu-package kernel\_image kernel\_headers |

## Script to pull and build qemu

|  |
| --- |
| #!/bin/bash -x  cdir**=$(pwd)**  W\_DIR**=**"qemu"  URL**=**"https://git.qemu.org/git/qemu.git"  BRANCH**=**"v4.2.0"  **[[** "$1" **==** "clean" **]]** **&&** **rm** **-**rf **$W\_DIR** **&&** **exit** 0  **function** ubu\_install\_qemu**(){**  **sudo** -s <<RUNASSUDO\_INSTALL  apt purge **-**y "qemu\*" **&>/**dev**/**null  apt autoremove **-**y **&>/**dev**/**null  apt install **-**y git libfdt-dev libpixman-1-dev libssl-dev vim socat libsdl2-dev libspice-server-dev autoconf libtool xtightvncviewer tightvncserver x11vnc uuid-runtime uuid uml-utilities bridge-utils python-dev liblzma-dev libc6-dev libegl1-mesa-dev libepoxy-dev libdrm-dev libgbm-dev libaio-dev libusb-1.0.0-dev libgtk-3-dev bison **&>/**dev**/**null  RUNASSUDO\_INSTALL  **if** **[** **!** **-d** **$W\_DIR** **];** **then**  **git** clone **--**depth 1 **$URL** **--**branch **$BRANCH** **--**single-branch **$W\_DIR**  **fi**  **cd** **$W\_DIR**  **rm** **-**rf **./**roms**/**seabios  **git** submodule update **--**init roms**/**seabios  **./**configure **--**prefix**=/**usr **\**  **--**enable-kvm **\**  **--**disable-xen **\**  **--**enable-libusb **\**  **--**enable-debug-info **\**  **--**enable-debug **\**  **--**enable-sdl **\**  **--**enable-vhost-net **\**  **--**enable-spice **\**  **--**disable-debug-tcg **\**  **--**enable-opengl **\**  **--**enable-gtk **\**  **--**target-list**=**x86\_64-softmmu **\**  **--**audio-drv-list**=**pa  **if** **make** **-**j**`nproc`;** **then**  **echo** "Make successful"  **else**  **echo** "Make failed"  **exit** 0  **fi**  **cd** **./**roms**/**seabios  **make** **-**j **`nproc`**  # make install  # cd ../  **}**  ubu\_install\_qemu |

## VFIO PCI unbind and bind script

|  |
| --- |
| #!/bin/bash  # Usage: pci-vfio.sh <Bus:Dev.Func>  **if** **[** "$#" **-**eq 0 **];** **then**  **echo** "Usage: pci-vfio.sh Bus:Dev.Func"  **exit**  **fi**  **for** pcie **in** "$@"**;** **do**  vendor**=$(cat /sys/bus/pci/devices/"0000:$pcie"/vendor)**  device**=$(cat /sys/bus/pci/devices/"0000:$pcie"/device)**  **echo** "unbinding -> $pcie ($vendor $device)"  **if** **[** **-e** **/**sys**/**bus**/**pci**/**devices**/**"0000:$pcie"**/**driver **];** **then**  **echo** "0000:$pcie" **|** **sudo** tee **/**sys**/**bus**/**pci**/**devices**/**"0000:$pcie"**/**driver**/**unbind  **fi**  **echo** "binding -> $pcie ($vendor $device)"  **echo** **$vendor** **$device** **|** **sudo** tee **/**sys**/**bus**/**pci**/**drivers**/**vfio-pci**/**new\_id  **done** |

## Gvt.sh (create-vgpu, setup directory)

|  |
| --- |
| #!/bin/bash -e  #cdir=$(pwd)  VM\_WORK\_DIR**=**${PWD}  cdir**=$VM\_WORK\_DIR**  #grub\_file=/etc/default/grub  grub\_file**=$(pwd)/**grub  option\_GRUB\_DEFAULT**=**'"Advanced options for Ubuntu>Ubuntu, with Linux 4.19.65-intelgvt+"'  option\_CMDLINE**=**'"quiet splash vt.handoff=7 i915.enable\_gvt=1 i915.enable\_fbc=0 kvm.ignore\_msrs=1 intel\_iommu=on,igfx\_off pcie\_aspm=off drm.debug=0"'  **function** clean**()** **{**  **rm** **-**rf **$cdir/**vm  **}**  **[[** "$1" **==** "clean" **]]** **&&** clean **&&** **exit** 0  #--------------------------  # 1) Create working directory  #--------------------------  **mkdir** -p **$cdir/**vm**/**disk  **mkdir** -p **$cdir/**vm**/**fw  **mkdir** -p **$cdir/**vm**/**iso  **mkdir** -p **$cdir/**vm**/**script  #--------------------------  # 2) Modify GRUB  #--------------------------  # setup grub  **sudo** sed **-**i 's/^GRUB\_DEFAULT=.\*/GRUB\_DEFAULT=\'"$option\_GRUB\_DEFAULT"'/' **$grub\_file**  **sudo** sed **-**i 's/^GRUB\_CMDLINE\_LINUX\_DEFAULT=.\*/GRUB\_CMDLINE\_LINUX\_DEFAULT=\'"$option\_CMDLINE"'/' **$grub\_file**  #----------------------------  # Uncomment once testing is done  #sudo update-grub2  #----------------------------  #network bridge setup  # setup auto start for create vgpu  #----------------------------  # prepare the vgpu setup  #--------------------------  # 3) Start vgpu-list  #--------------------------  #--------------------------  VGPU\_LIST**=$VM\_WORK\_DIR/**vm**/**script**/**vgpu-list  **touch** **$VGPU\_LIST**  **echo** "Adding Mask and GUID, $VGPU\_LIST"  **cat** **<<** EOF **|** **sudo** tee **$VGPU\_LIST**  MASK**=**0x000000000080402  VGPU0**=**37414358**-**bd2b-11e8-8fa3-7bde838751b0  VGPU1**=**37414358**-**bd2b-11e8-8fa3-7bde838751b1  VGPU2**=**37414358**-**bd2b-11e8-8fa3-7bde838751b2  EOF  #--------------------------  #--------------------------  # 4) Start vgpu.service  #--------------------------  #--------------------------  SYSTEMD\_VGPU**=$VM\_WORK\_DIR/**vm**/**script**/**vgpu.service  **touch** **$SYSTEMD\_VGPU**  **echo** "Adding VGPU Service $SYSTEMD\_VGPU"  **cat** **<<** EOF **|** **sudo** tee **$SYSTEMD\_VGPU**  **[**Unit**]**  Description**=**"Start/Stop VGPU service"  **[**Service**]**  WorkingDirectory**=/**VM\_WORK\_DIR**/**  Type**=**oneshot  ExecStart**=/**bin**/**bash create-vgpu start  ExecStop**=/**bin**/**bash create-vgpu stop  RemainAfterExit**=**yes  **[**Install**]**  WantedBy**=**multi-user.target  EOF  **sed** **-**i "s|/VM\_WORK\_DIR|$VM\_WORK\_DIR/vm/script|g" **$SYSTEMD\_VGPU**  #######-------- sudo systemd enable $SYSTEMD\_VGPU  #--------------------------  # 5) Start create-vgpu  #--------------------------  # /sys/class/drm/card0  CREATE\_VGPU**=$VM\_WORK\_DIR/**vm**/**script**/**create-vgpu  **touch** **$CREATE\_VGPU**  **echo** "Adding VGPU Service $CREATE\_VGPU"  **cat** << EOF **|** **sudo** tee **$CREATE\_VGPU**  #!/bin/bash -x  **[[** **-f** **$VGPU\_LIST** **]]** **&&** source **$VGPU\_LIST**  **echo** "mask: \$MASK"  **echo** "vgpu0: \$VGPU0"  **echo** "vgpu1: \$VGPU1"  **echo** "vgpu2: \$VGPU2"  **echo** \$MASK **>** **/**sys**/**devices**/**pci0000**:**00**/**0000**:**00**:**02**.**0**/**drm**/**card0**/**gvt\_disp\_ports\_mask  **if** **[** **!** **-d** "/sys/devices/pci0000:00/0000:00:02.0/\$VGPU0" **];** **then**  **echo** \$VGPU0 **>** "/sys/devices/pci0000:00/0000:00:02.0/mdev\_supported\_types/i915-GVTg\_V5\_4/create"  **fi**  **if** **[** **!** **-d** "/sys/devices/pci0000:00/0000:00:02.0/\$VGPU1" **];** **then**  **echo** \$VGPU1 **>** "/sys/devices/pci0000:00/0000:00:02.0/mdev\_supported\_types/i915-GVTg\_V5\_4/create"  **fi**  **if** **[** **!** **-d** "/sys/devices/pci0000:00/0000:00:02.0/\$VGPU2" **];** **then**  **echo** \$VGPU2 **>** "/sys/devices/pci0000:00/0000:00:02.0/mdev\_supported\_types/i915-GVTg\_V5\_4/create"  **fi**  EOF  **chmod** **+**x **$CREATE\_VGPU**  **exit** 0  **sudo** systemd enable vgpu.service  **sudo** systemd start vgpu.service  #--------------------------  **exit** 0 |

## Use case

TBD

## QEMU parameters

TBD