**Installation Manual - KVM Hypervisor**

Project Name: **Cyber Threat Automation and Monitoring System (CTAM)**

Contract Number: **W900KK-16-C-0043 CDRL: A00X CLIN: 000X EWT Domain: EO/RF**

Contractor Name: **Florida International University, Miami, FL**

Principle Investigator: **Dr. Leonel Lagos,**

**305-348-1810,**

[**Lagosl@fiu.edu**](mailto:Lagosl@fiu.edu)

Co-Principle Investigator: **Dr. Himanshu Upadhyay,**

**305-348-6603,**

[**Upadhyay@fiu.edu**](mailto:Upadhyay@fiu.edu)

RevisionHistory**:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 06/12/2020 | 1.0 | Initial Version | FIU |

1.0 KVM Installation Procedure:

1.1. KVM Introduction

Kernel-based Virtual Machine (KVM) is an open source full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). KVM supports multiple virtual machines running unmodified Linux or Windows images. Each virtual machine has private virtualized hardware resources such as network card, disk, graphics adapter etc.

1.2 Operating system requirements to install KVM hypervisor:

To install the KVM hypervisor, we need to install a Linux operating system (Ubuntu 18.04) on the bare metal hardware.

Please follow the below mentioned instructions to install Ubuntu 18.04.

* Please follow the below mentioned link to create a bootable USB drive for Ubuntu 18.04 installation : [https://ubuntu.com/tutorials/tutorial-create-a-usb-stick-on-windows#1-overview](https://ubuntu.com/tutorials/tutorial-create-a-usb-stick-on-windows)
* Please follow the below mentioned link to install Ubuntu 18.04 : [https://ubuntu.com/tutorials/tutorial-install-ubuntu-desktop#1-overview](https://ubuntu.com/tutorials/tutorial-install-ubuntu-desktop)

1.2.1 Source list requirements:

To install KVM hypervisor, the “source.list” file of the Linux operating system needs to be updated. Please overwrite the file contents with the below mentioned information. The file is located at “/etcp/apt/sources.list” and contains all the necessary repositories for the dependencies that will be used for the KVM installation.

|  |
| --- |
| deb-src <http://us.archive.ubuntu.com/ubuntu/> bionic main restricted  deb <http://us.archive.ubuntu.com/ubuntu/> bionic main restricted  deb <http://us.archive.ubuntu.com/ubuntu/> bionic-updates main restricted  deb-src <http://us.archive.ubuntu.com/ubuntu/> bionic-updates main restricted  deb <http://us.archive.ubuntu.com/ubuntu/> bionic universe  deb-src <http://us.archive.ubuntu.com/ubuntu/> bionic universe  deb <http://us.archive.ubuntu.com/ubuntu/> bionic-updates universe  deb-src <http://us.archive.ubuntu.com/ubuntu/> bionic-updates universe  deb <http://us.archive.ubuntu.com/ubuntu/> bionic multiverse  deb-src <http://us.archive.ubuntu.com/ubuntu/> bionic multiverse  deb <http://us.archive.ubuntu.com/ubuntu/> bionic-updates multiverse  deb-src <http://us.archive.ubuntu.com/ubuntu/> bionic-updates multiverse  deb <http://us.archive.ubuntu.com/ubuntu/> bionic-backports main restricted universe multiverse  deb-src <http://us.archive.ubuntu.com/ubuntu/> bionic-backports main restricted universe multiverse  deb <http://archive.canonical.com/ubuntu> bionic partner  deb-src <http://archive.canonical.com/ubuntu> bionic partner  deb <http://security.ubuntu.com/ubuntu> bionic-security main restricted  deb-src <http://security.ubuntu.com/ubuntu> bionic-security main restricted  deb <http://security.ubuntu.com/ubuntu> bionic-security universe  deb-src <http://security.ubuntu.com/ubuntu> bionic-security universe  deb <http://security.ubuntu.com/ubuntu> bionic-security multiverse  deb-src <http://security.ubuntu.com/ubuntu> bionic-security multiverse |

Please execute the below mentioned command for the changes to take effect:

|  |
| --- |
| sudo apt-get update |

1.3 Installing QEMU:

If a .tar file containing the sources of local copies is given, unzip it by executing the following command:

|  |
| --- |
| sudo tar –xpvzf CTAM-KVM-Sources.tar.gz  cd CTAM-KVM-Sources |

Before installing QEMU, the following dependencies need to be installed by executing the commands below:

|  |
| --- |
| sudo apt-get install qemu-kvm  sudo apt-get build-dep qemu |

If downloading QEMU emulator from the official repository, please execute the following commands:

|  |
| --- |
| sudo wget <https://download.qemu.org/qemu-4.1.0.tar.xz>  tar xvJf qemu-4.1.0.tar.xz |

For QEMU to work along with LibVMI, it needs to be patched. The patch file can be found inside the LibVMI source files directory.

If downloading LibVMI from the official repositories, please execute the commands below:

|  |
| --- |
| sudo git clone <https://github.com/libvmi/libvmi.git>  cd libvmi  sudo git checkout 8a03f89127559c06e7a204dbad5b8642df557e5e  cd .. |

If working from a local copy of LibVMI, just make sure that the LibVMI folder and QEMU folders reside inside the same parent folder.

To apply the patch, perform the following commands:

|  |
| --- |
| cd qemu-4.1.0  sudo patch -p1 < ../libvmi/tools/qemu-kvm-patch/kvm-qemu-v4.1.0-libvmi.patch |

To install QEMU, perform the following commands:

|  |
| --- |
| ./configure --prefix=/usr  sudo make -j  sudo make install -j |

Validate the QEMU installation by running the following command:

|  |
| --- |
| qemu-system-x86\_64 --version |

The QEMU version **4.1.0** should be displayed on the terminal screen.

1.4 Opening network ports and creating network bridge:

Please execute below mentioned command to open the communication ports that are required for the application to run. These ports can be changed on the introspector.conf file, and the ones shown below are shown as examples:

|  |
| --- |
| sudo apt-get install firewalld  sudo systemctl start firewalld  sudo systemctl enable firewalld  sudo firewall-cmd --permanent --zone=public --add-port=3655/udp  sudo firewall-cmd --permanent --zone=public --add-port=29171/tcp  sudo firewall-cmd --permanent --zone=public --add-port=29171/udp  sudo firewall-cmd --permanent --zone=public --add-port=3350/tcp  sudo firewall-cmd --permanent --zone=public --add-port=3350/udp  sudo firewall-cmd --permanent --zone=public --add-port=5700-5799/tcp  sudo firewall-cmd --permanent --zone=public --add-port=5700-5799/udp  sudo firewall-cmd --reload |

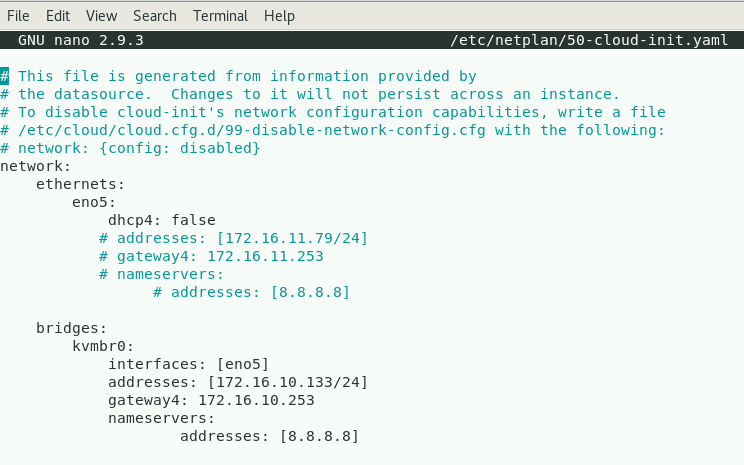
To verify that the ports have been successfully opened, please execute the command below:

|  |
| --- |
| sudo firewall-cmd --list-all |

For the hypervisor to provide internet access to the guest machines, a network bridge needs to be created by following the steps below:

* Remote login to the hypervisor and perform the following command to open the file.

|  |
| --- |
| sudo nano /etc/netplan/50-cloud-init.yaml |

* Update the contents of the file as shown in the Figure 1.
* The important parameters are the name of ethernets and bridges.
* In this case, **ethernets has name eno5**, and **bridges has kvmbr0.**
* Update the addresses to the IP of hypervisor **(172.16.10.133 in this example)**

***Figure 1: File (etc/netplan/50-cloud-init.yaml) Configuration***

* Save the file and execute the following command to apply the configuration:

|  |
| --- |
| sudo netplan try  sudo netplan apply |

2.0 Libvirt Installation:

The libvirt project is an open source toolkit to manage [virtualization platforms](https://libvirt.org/platforms.html). Libvert supports [KVM](https://libvirt.org/drvqemu.html), [QEMU](https://libvirt.org/drvqemu.html), [Xen](https://libvirt.org/drvxen.html), [Virtuozzo](https://libvirt.org/drvvirtuozzo.html), [VMWare ESX](https://libvirt.org/drvesx.html), [LXC](https://libvirt.org/drvlxc.html), [BHyve](https://libvirt.org/drvbhyve.html) hypervisor platforms and used to manage virtual machines with Linux, FreeBSD, [Windows](https://libvirt.org/windows.html) and OS-X operating systems.

2.1 Download Libvirt library from the repository or local copy:

If downloading Libvirt from the official repositories, please execute the command below:

|  |
| --- |
| sudo git clone <https://github.com/libvirt/libvirt.git>  cd libvirt  sudo git checkout v5.4.0 |

If working with a local copy, locate the Libvirt folder and cd into it.

2.2 Installation of Libvirt library:

Before installing Libvirt, the following dependencies need to be installed. Please execute the commands below:

|  |
| --- |
| sudo apt-get install autoconf  sudo apt-get install automake  sudo apt-get install libtool  sudo apt-get install libxml2-utils  sudo apt-get install xsltproc  sudo apt-get install libxml2-dev  sudo apt-get install libpciaccess-dev  sudo apt-get install libdevmapper-dev  sudo apt-get install gnutls-bin  sudo apt-get install libgnutls28-dev  sudo apt-get install libnl-3-dev  sudo apt-get install libnl-route-3-dev  sudo apt-get install yajl-tools  sudo apt-get install libssl-dev  sudo apt-get build-dep libvirt  sudo apt-get install golang-go |

Please execute below mentioned commands to install and configure Libvirt library.

|  |
| --- |
| cd libvirt  sudo ./autogen.sh –prefix=/usr  sudo make -j  sudo make install -j |

2.3 Start Libvirtd services:

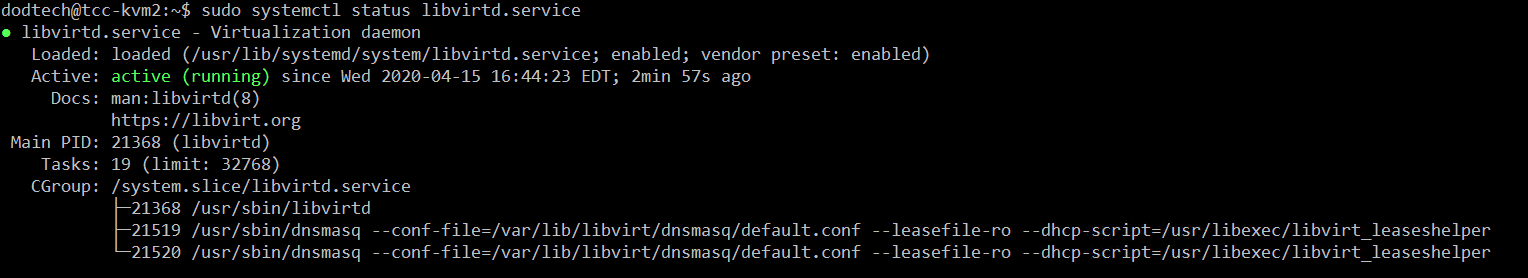
To start the libvrtd services, execute the following commands:

|  |
| --- |
| sudo systemctl enable libvirtd.service  sudo systemctl start libvirtd.service |

To verify that libvirtd services are running, execute the following command:

|  |
| --- |
| sudo systemctl status libvirtd.service |

The sample output displayed on the terminal will be shown below.



2.4 Enable remote connection to libvirt:

To enable remote connection to libvirt, the following configuration file needs to be modified.

Open the file by running the below mentioned command.

|  |
| --- |
| sudo nano /etc/libvirt/libvirtd.conf |

Following lines needs to be uncommented by removing the ‘#’ symbol from the beginning of the line:

|  |
| --- |
| unix\_sock\_group = “libvirt”  unix\_sock\_rw\_perms = “0770” |

Save the file by pressing ctrl+x, and then pressing ‘Y’ to overwrite the changes to the file.

After the file has been successfully modified, execute the following commands:

|  |
| --- |
| sudo groupadd libvirt  sudo adduser $USER libvirt |

To verify that the user was added successfully to the group, execute the following commands:

|  |
| --- |
| sudo apt-get install members  members libvirt |

Running the “members libvirt” command will show all users on the libvirt group.

Note: If working through an SSH connection, the user needs to close and open a new SSH connection for the group changes to be reflected.

In order to allow remote users to connect to the hypervisor, a rule needs to be added to the polkit of the server. This is done by running the following commands:

|  |
| --- |
| sudo su  sudo nano /etc/polkit-1/localauthority/50-local.d/50-libvirt-remote-access.pkla |

The contents of this file are as shown below:

|  |
| --- |
| [Remote libvirt SSH access]  Identity=unix-group:libvirt  Action=org.libvirt.unix.manage  ResultAny=yes  ResultInactive=yes  ResultActive=yes |

Exit nano by pressing ctrl+x and then ‘Y’ to overwrite the file.

In order to exit the sudo command prompt, execute the following command:

|  |
| --- |
| exit |

After editing the file, restart the libvirtd services for the changes to take effect by executing the following commands:

|  |
| --- |
| sudo systemctl stop libvirtd  sudo systemctl start libvirtd |

To verify that the settings are working properly, please perform Step 3 (Figure 5) of Virtual Image Creation section.

3.0 LibVMI library Installation:

The LibVMI is a virtual machine memory introspection library that provides capability to extract the low-level details of a running virtual machine by introspecting the memory, trapping hardware events and accessing the vCPU registers.

Please refer: <http://libvmi.com/> for more details about LibVMI.

3.1 Install LibVMI dependencies:

Please execute below mentioned command to install the required dependency packages for the LibVMI library.

|  |
| --- |
| sudo apt-get install cmake  sudo apt-get install flex  sudo apt-get install bison  sudo apt-get install libglib2.0-dev  sudo apt-get install libjson-c-dev  sudo apt-get install libyajl-dev  sudo apt-get install doxygen |

3.2 Download and install LibVMI library from the repository or local copy:

LibVMI’s source folder should already be available by following the instructions on section 1.3. Locate the LibVMI folder in order to proceed with the LibVMI’s installation and cd into it.

3.3 Configuration of LibVMI:

Once inside the LibVMI folder, please execute below mentioned commands to configure LibVMI library.

|  |
| --- |
| sudo mkdir build  cd build  sudo cmake -DENABLE\_XEN=OFF -DENABLE\_KVM=ON ..  sudo make -j  sudo make install -j  cd ..  cp libvmi/events.h /usr/local/include/libvmi/events.h  sudo ldconfig |

4.0 Virtual Machine Image Creation:

**Note:** Install Virtual Machine Manager (VMM) on the client machine which will be used to create virtual machine images.

VMM download link: <https://virt-manager.org/>.

Please follow the below mentioned steps to create virtual machine image from the iso.

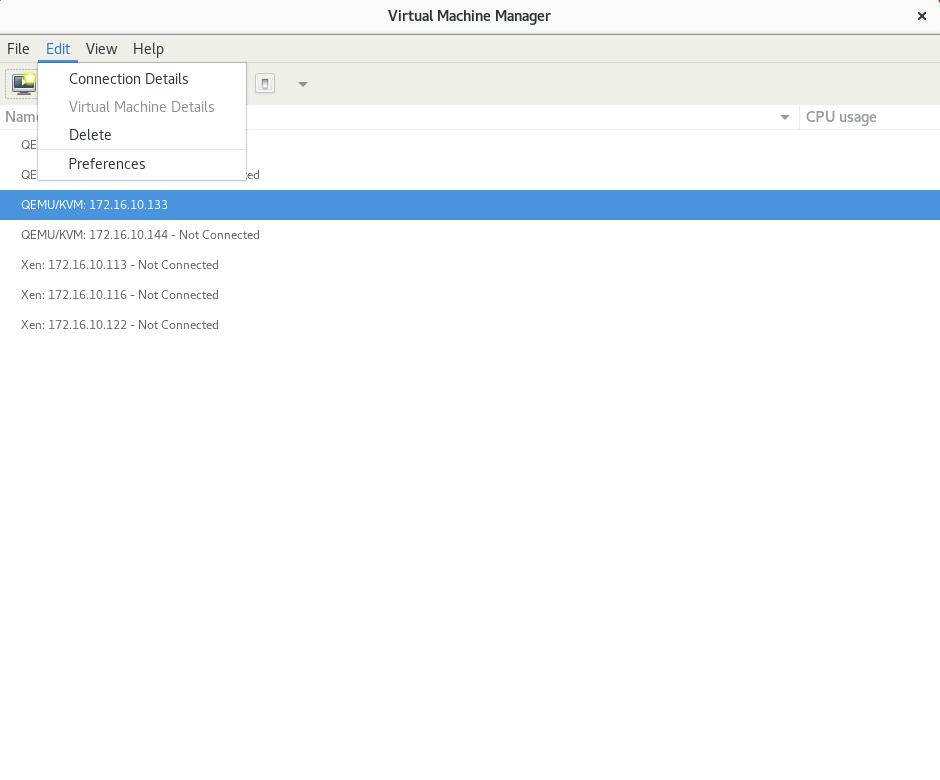
**Step 1:**

* Make sure that Hypervisor’s bridge is properly set up by executing the command below:

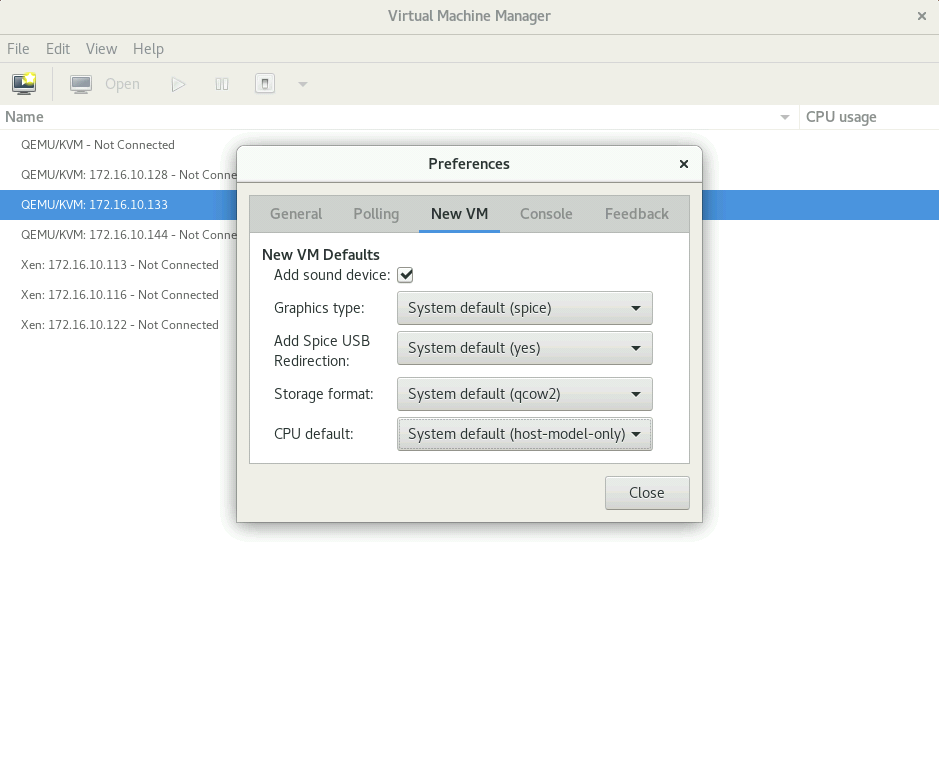
|  |
| --- |
| sudo brctl show |

**Step 2:**

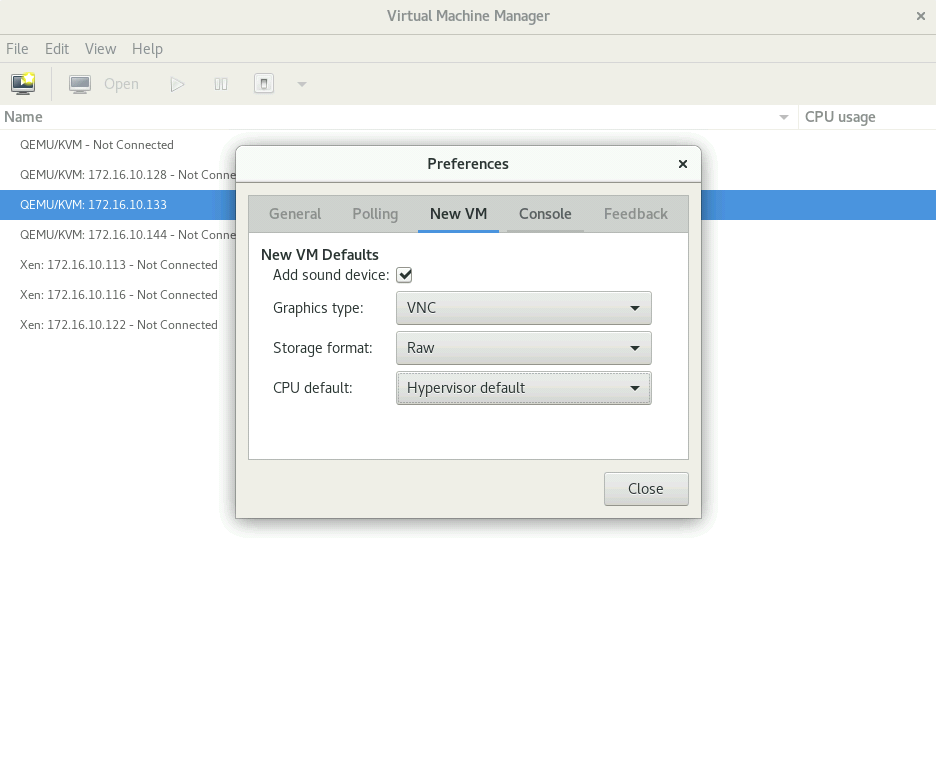
* Open the Virtual Machine Manager on the Client’s Machine.
* From the top menu click on Edit and select preferences, as shown in Figure 2.
* In Preferences section click on the New VM option.
* Settings with default are shown in the Figure 3.
* Change the Graphics Type to VNC, Storage format to RAW, and CPU default to Hypervisor Default, as shown in Figure 4.
* Close the preferences section.



***Figure 2: VMM – Preferences Settings***

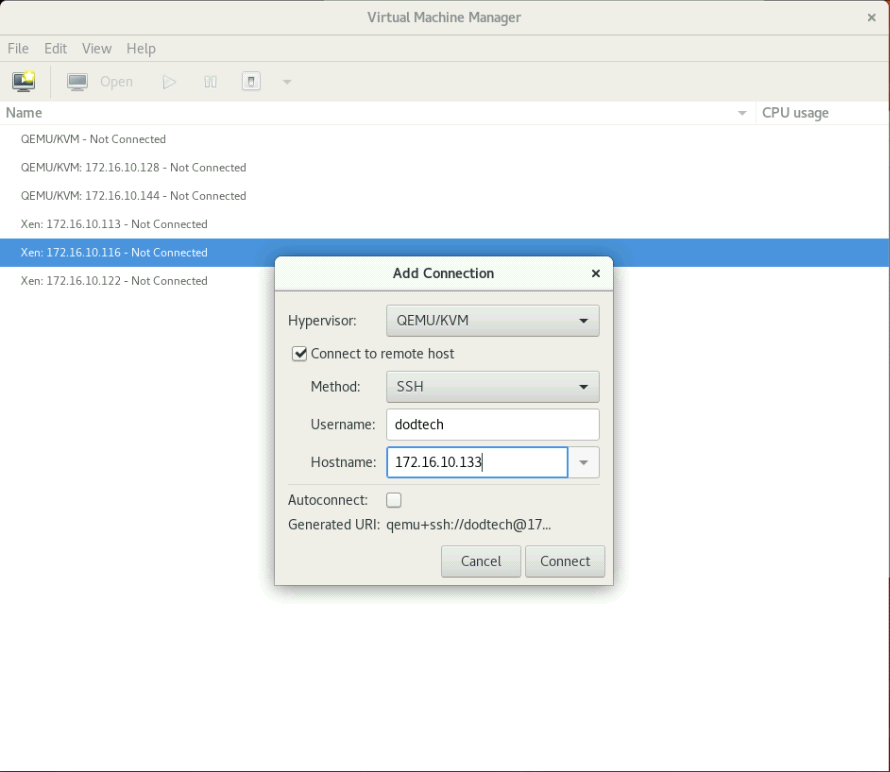


***Figure 3: VMM – View Default Preferences Settings***



***Figure 4: VMM – Update Preferences Settings***

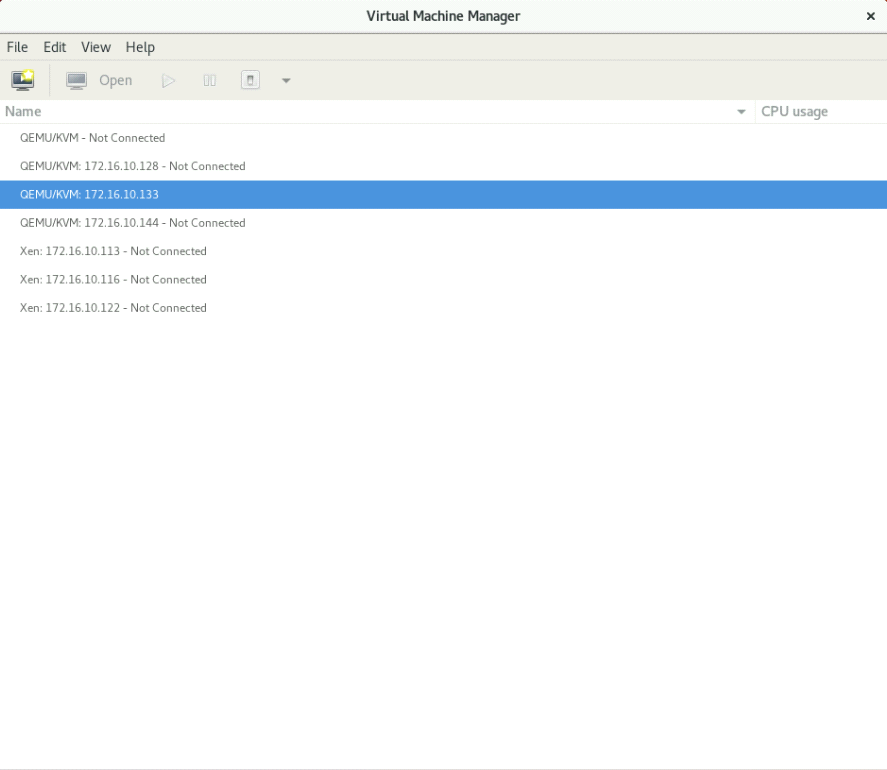
**Step 3:**

* Open the Virtual Machine Manager on the Client’s Machine.
* Select the appropriate Hypervisor (KVM in this example).
* Enter the username and hostname (IP address) of the hypervisor as shown in Figure 5.
* Click on Connect button and then enter the password.

***Figure 5: VMM – Add Connection***

**Step 4:**

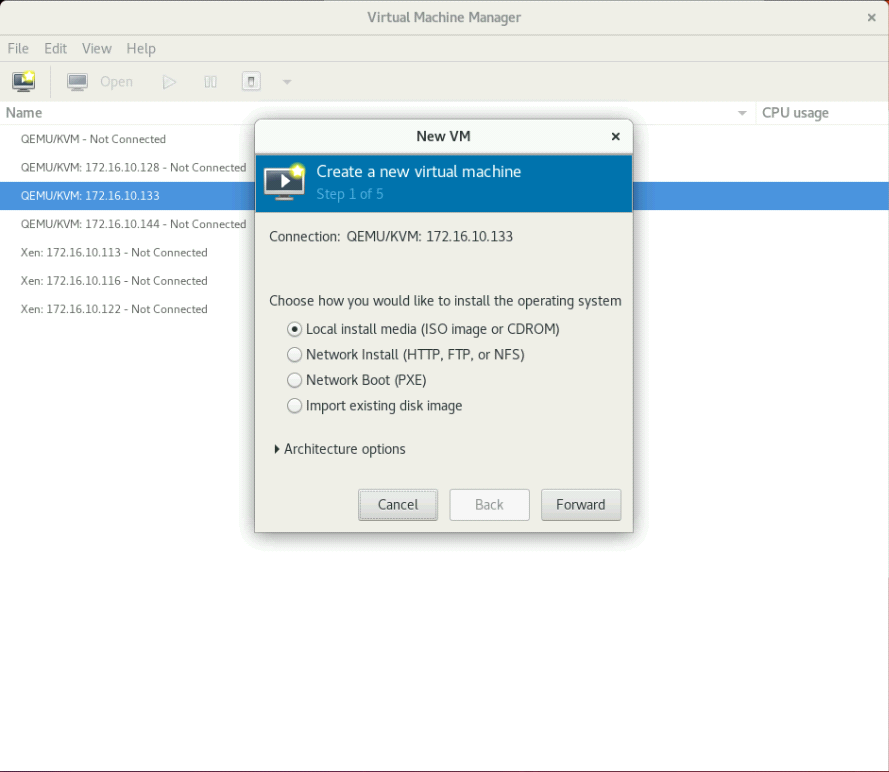
* If everything is working fine, then you will see the running virtual machines on the selected KVM hypervisor.
* As shown in Figure 6 below, there is no VM running on the selected KVM hypervisor as it is a recent install and VMs are not yet created.



***Figure 6: VMM – List of VMs on the Hypervisor***

**Step 5:**

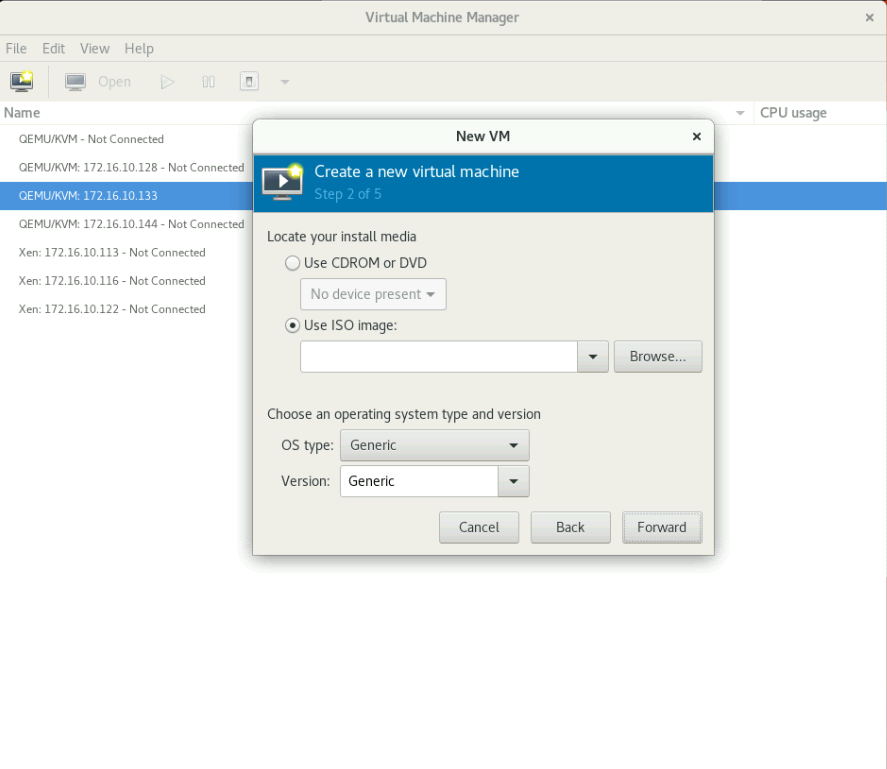
* Select the hypervisor (172.16.10.133) and right click on new.
* A window (New VM) will pop-up with all the different options, as shown in Figure 7.
* Select the suitable option for the installing the media (Local install media in this case).
* Click on forward button to proceed.



***Figure 7: VMM – Create New Virtual Machine***

**Step 6:**

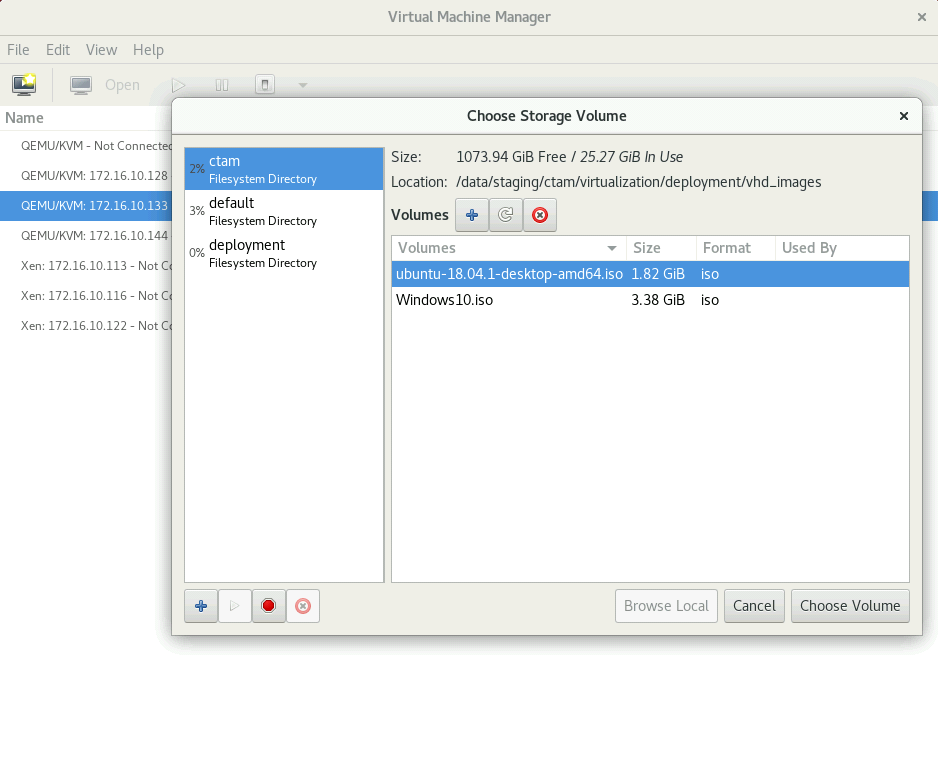
* Select the appropriate ISO image.
* Click on browse button to choose from the available options.
* Select OS type and Version from the drop-down list, as shown in Figure 8.



***Figure 8: VMM – Create New Virtual Machine – Select ISO***

**Step 7:**

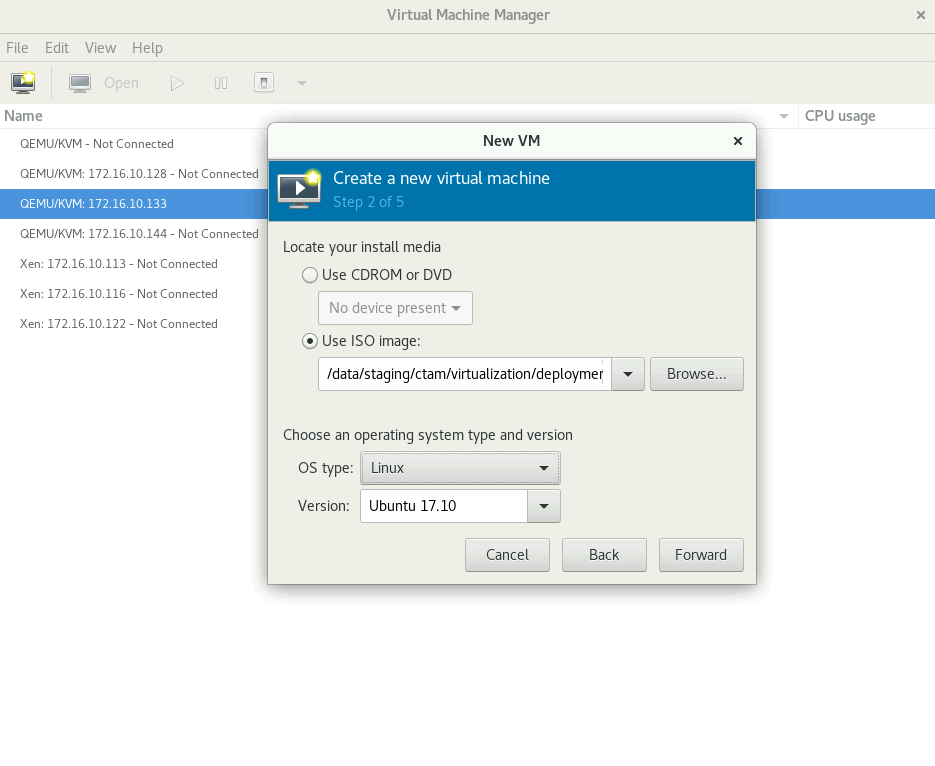
* Select the ISO file (Ubuntu 18.04 in this example).
* Click on choose volume button to select the ISO as shown in Figure 9.



***Figure 9: VMM – Select ISO File***

**Step 8:**

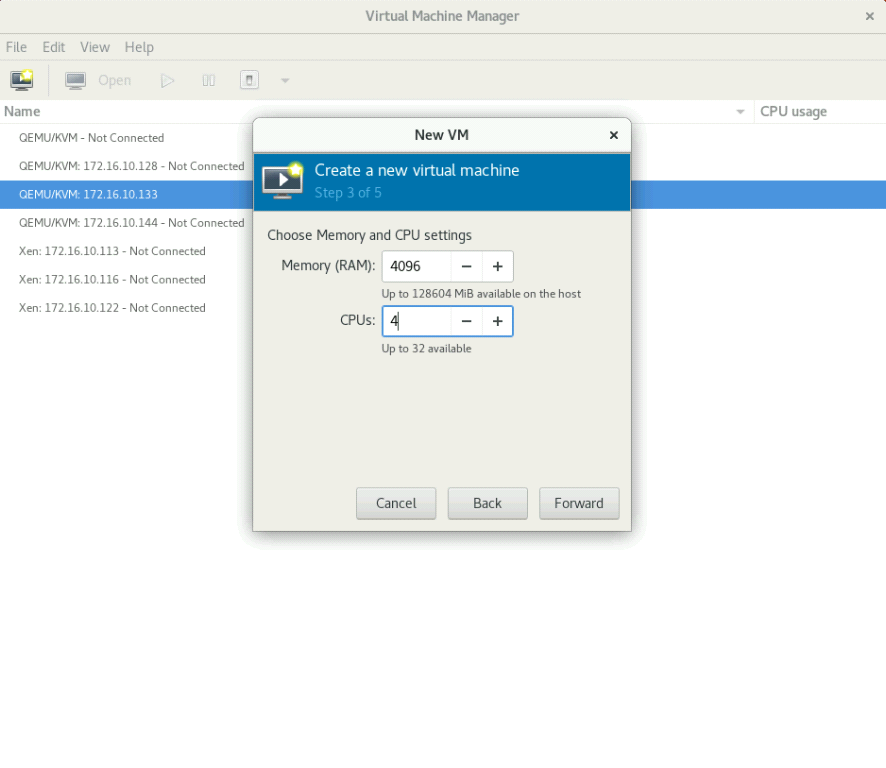
* Select OS type and versions from the drop-down menu.
* Click on forward button to proceed, as shown in Figure 10.



***Figure 10: VMM – Select OS Type and Version***

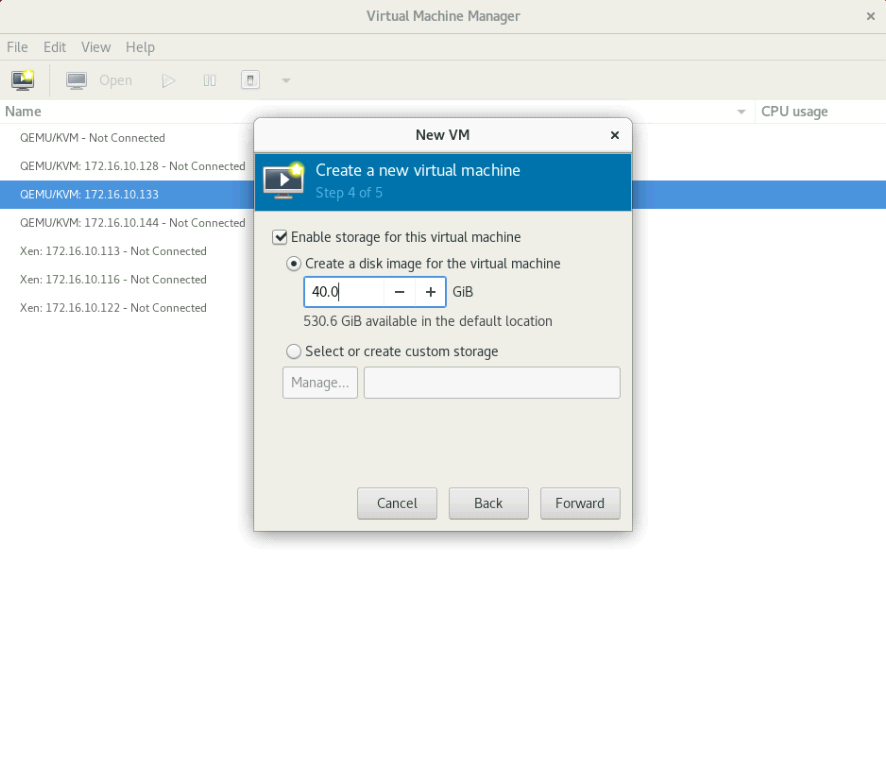
**Step 9:**

* The virtual machine image needs to have system configuration in terms of Memory (RAM) and CPUs.
* Select an optimal amount of the resources.
* Figure 11 below shows the available resources for the allocation.

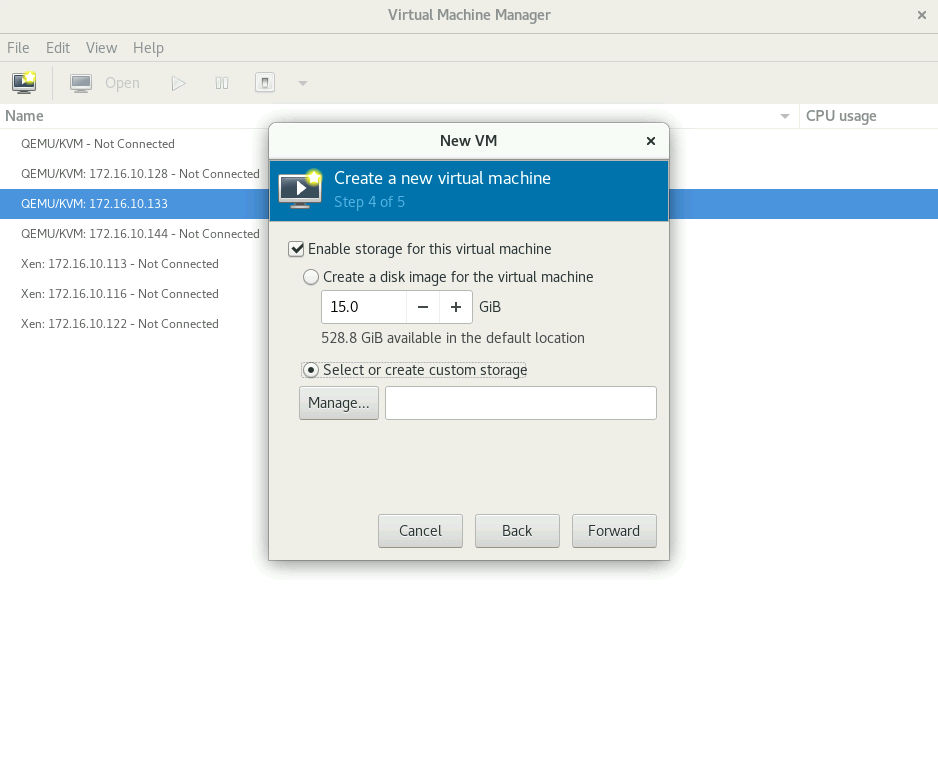


***Figure 11: VMM – Allocate Memory (RAM) and CPU***

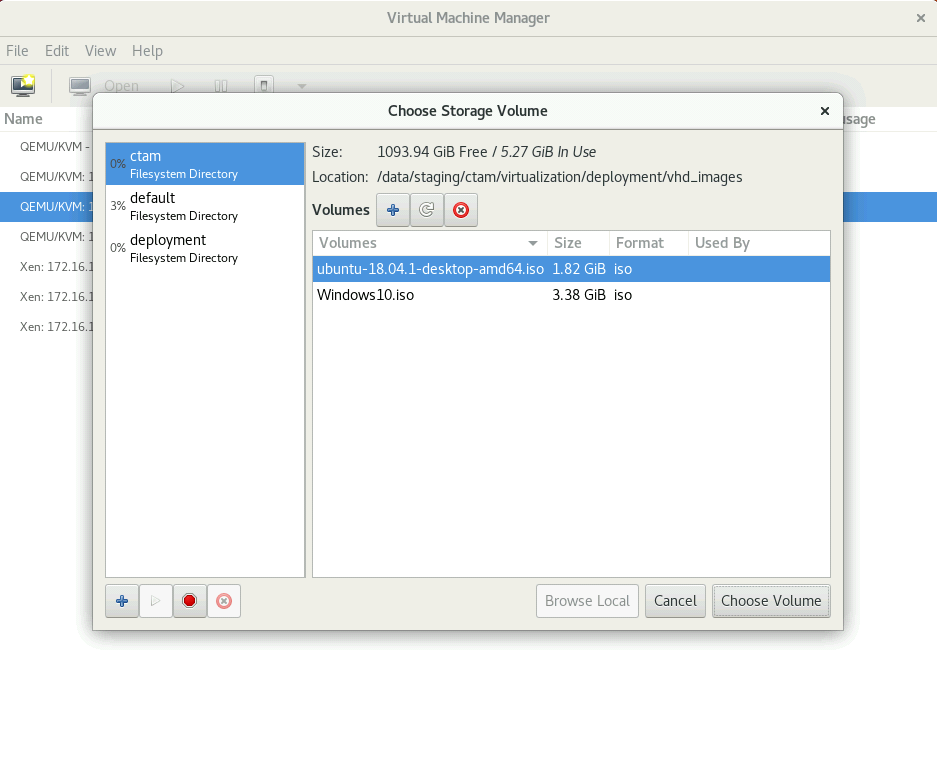
**Step 10:**

* In this step, we must create a custom storage for the image being created.
* As shown in Figure 12, it has two options:
* Create a disk image for the virtual machine
* Select or create custom storage
* Select the second option (a) as shown in Figure 13.
* Click on manage button to proceed.
* Figure 14 shows the screen that contains the options to create a custom volume.
* Click on the “+” sign next to the Volumes label.
* A window to add a storage volume will open as shown in Figure 15.
* We can change the VM image name, Max capacity and Allocation for the storage volume.
* We can see the changes in Figure 16.
* Click finish to button to apply changes.
* Select the newly created image, as shown in Figure 17.
* Click on choose volume button.
* Figure 18 shows the changes made by us for the new volume.
* Click on forward button to proceed.

***Figure 12: VMM – Allocate Disk Storage***



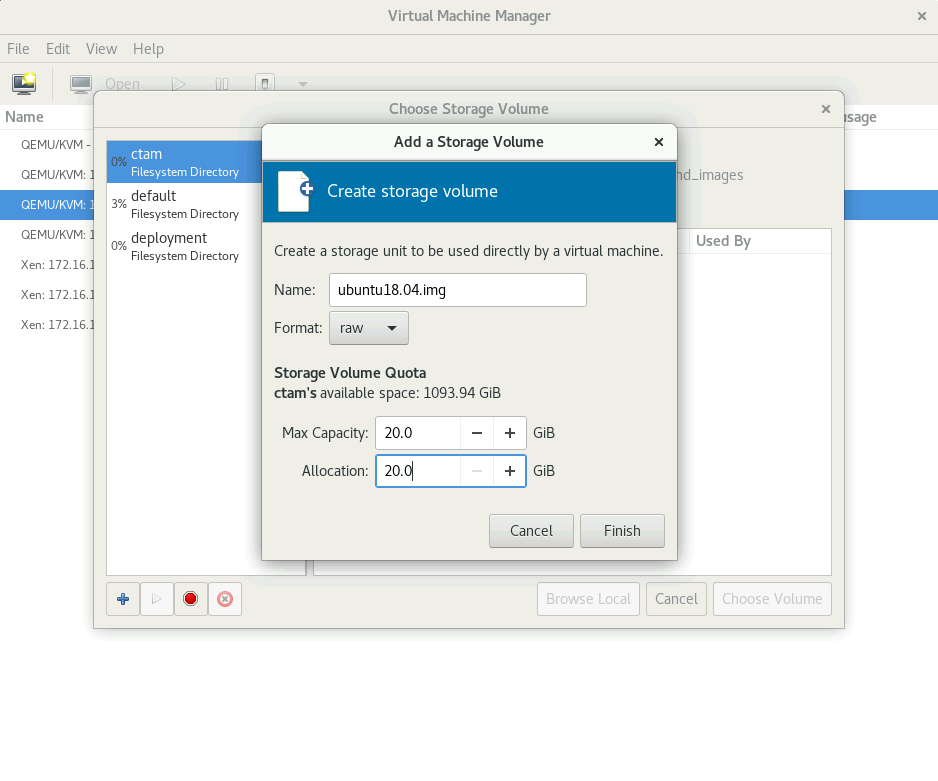
***Figure 13: VMM – Create Custom Volume Storage***



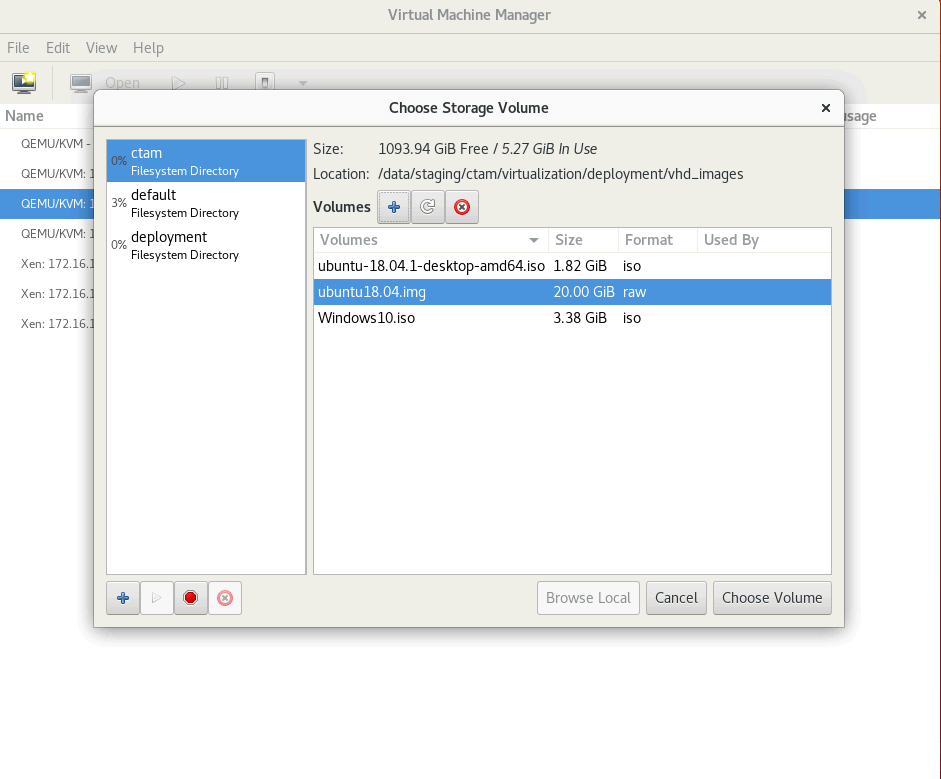
***Figure 14: VMM – Choose Storage Volume***



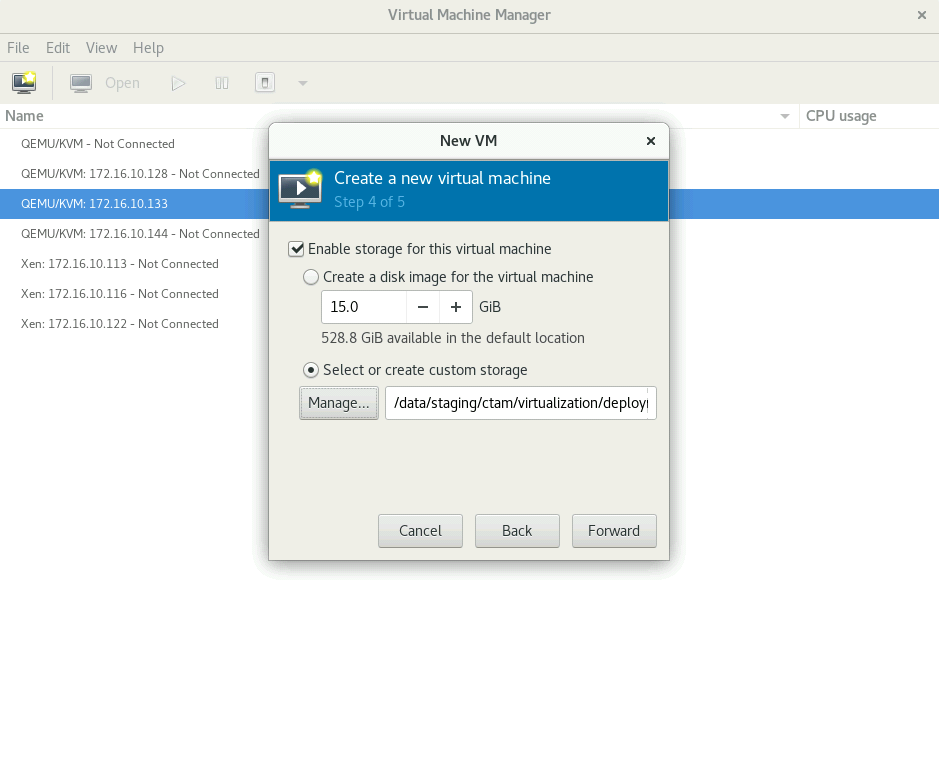
***Figure 15: VMM – Add Storage Volume***



***Figure 16: VMM – Allocate Storage Volume – Update Max Capacity***



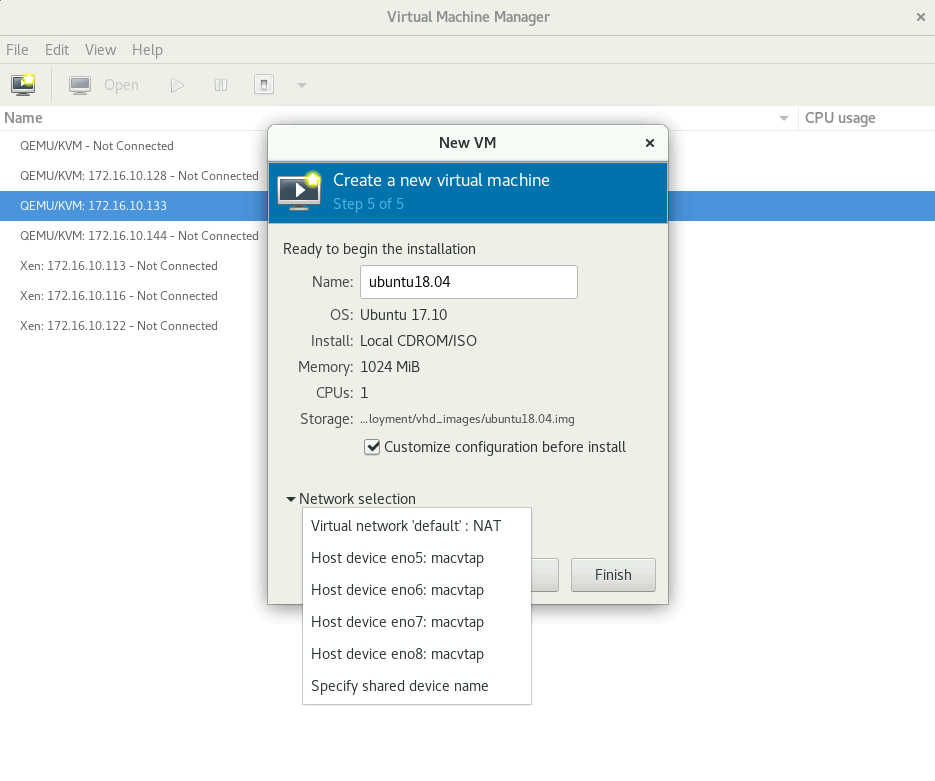
***Figure 17: VMM – Choose Disk Storage – New Storage Volume***



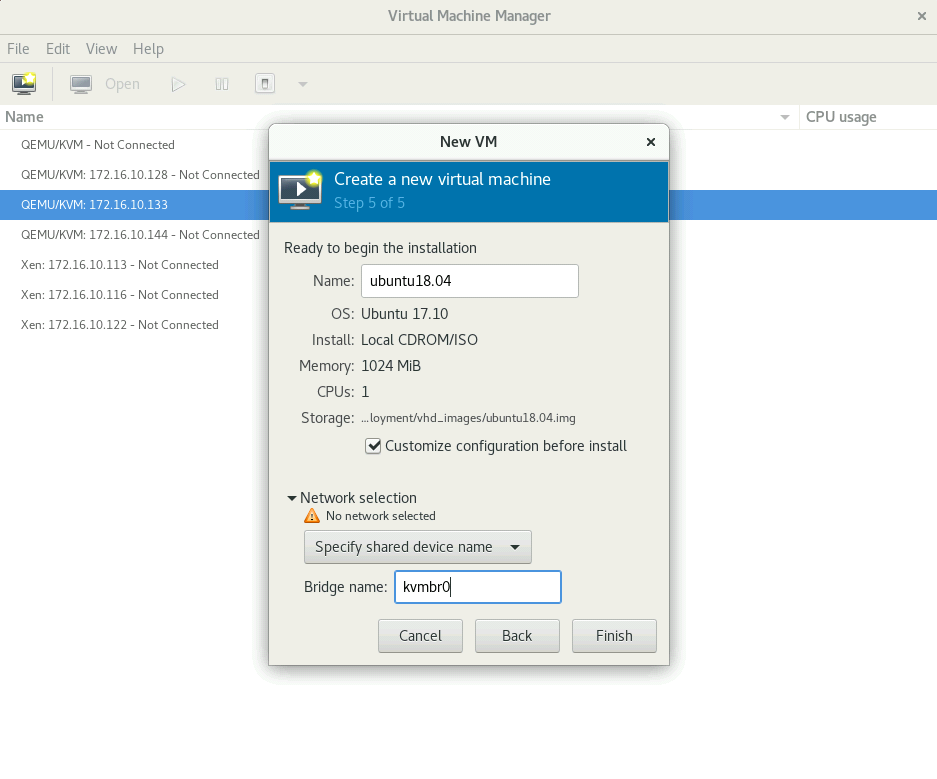
***Figure 18: VMM – Create New Virtual Machine***

**Step 11:**

* Name the VM as per your requirements.
* Select the check box for “Customize configuration before install”.
* This option allows to configure various settings of the VM before installing it.
* Select the network selection drop down and click on the last option “specify shared device name”, as shown in Figure 19.
* Enter the bridge name from the step 1, “kvmbr0” in the text field area, as shown in Figure 20.



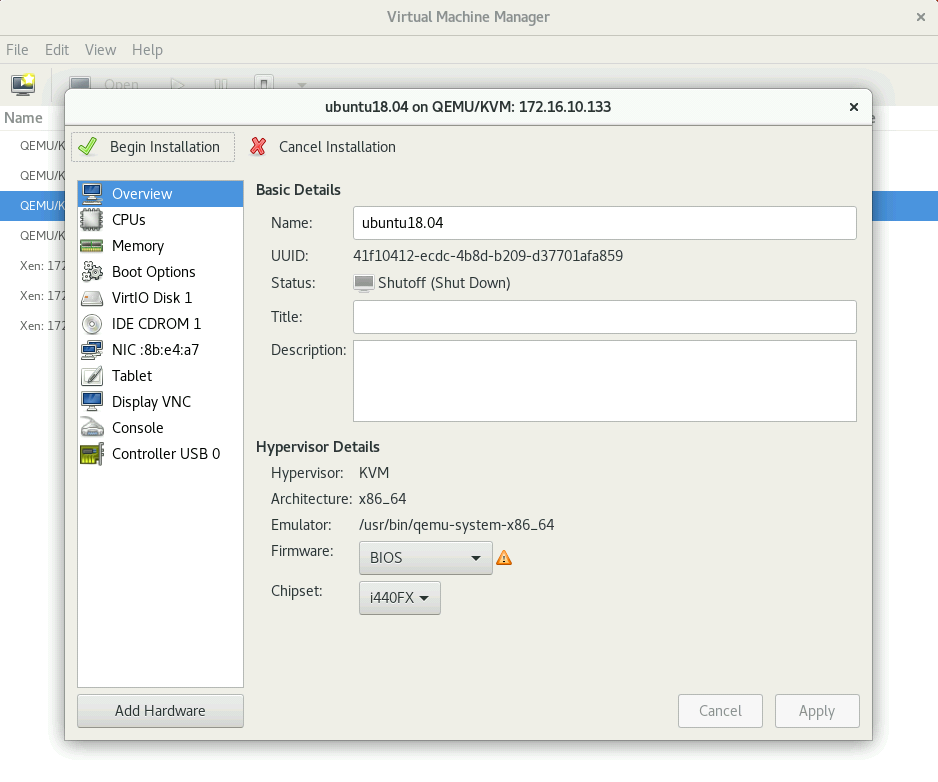
***Figure 19: VMM – Network Selection***



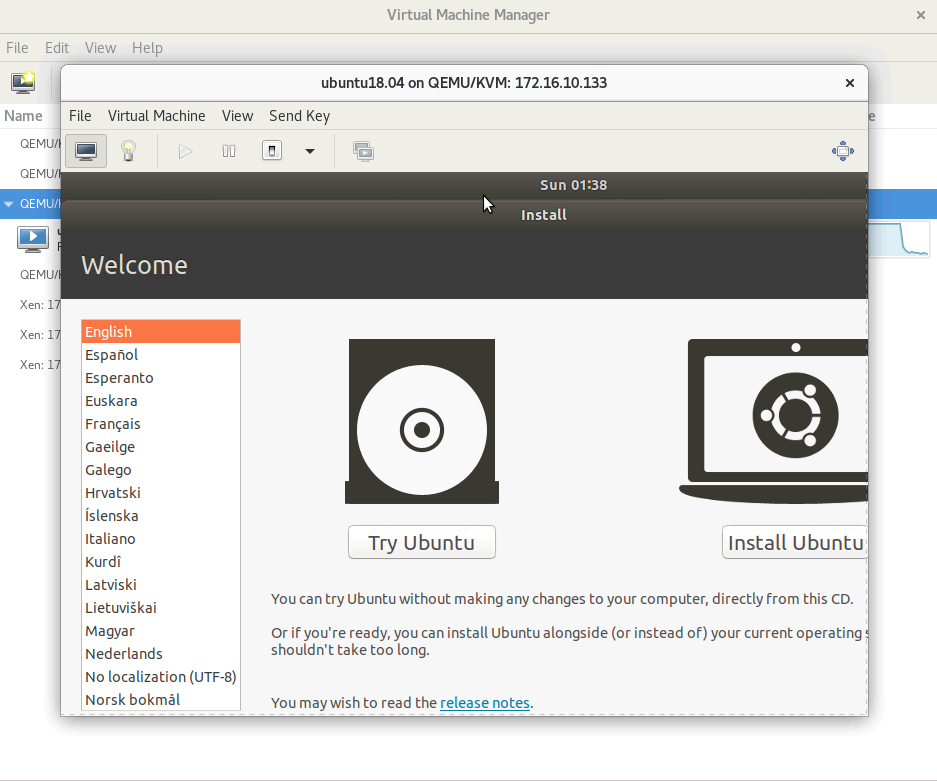
***Figure 20: VMM – Change VM Name and Bridge Name***

**Step 12:**

* Click on Begin Installation and follow the basic steps of OS installation and VM image will be created as shown in Figure 21.
* Figure 22 shows the initial phase of ubuntu installation on virtual machine.



***Figure 21: VMM – Linux OS Installation***

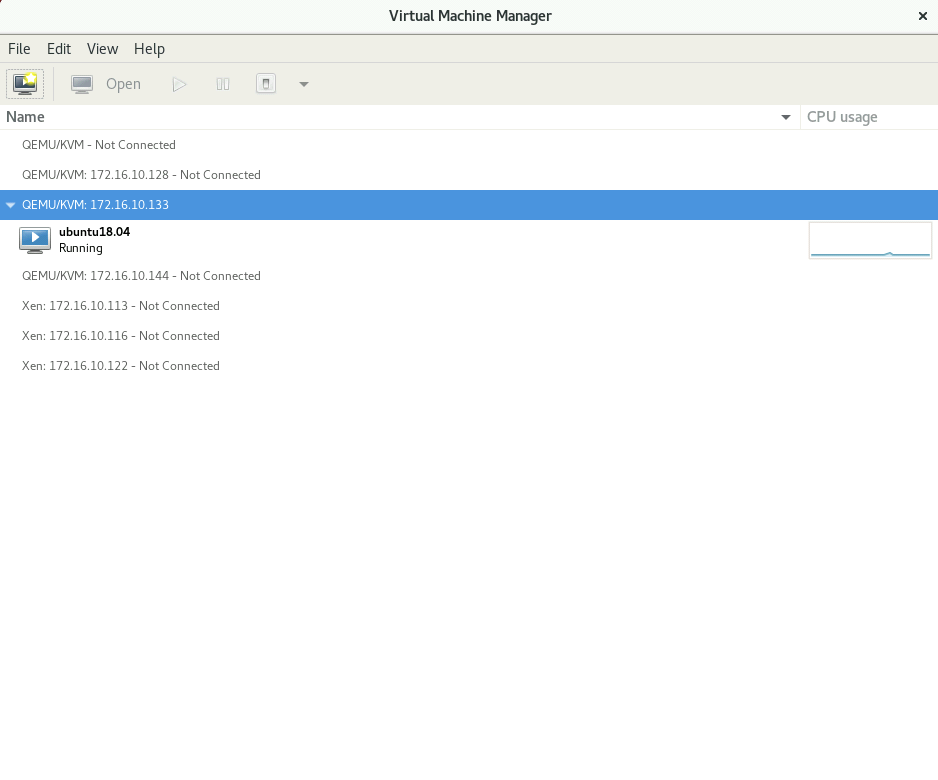


***Figure 22: VMM – Ubuntu Installation Page***

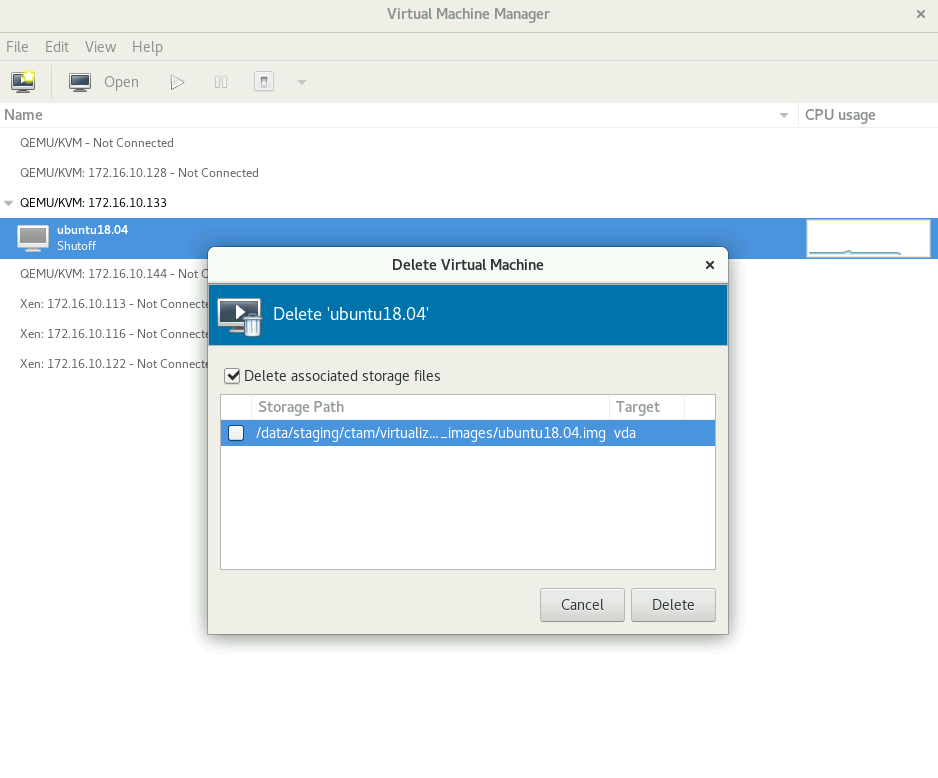
**Step 13:**

* Figure 23 below shows the status of the ubuntu is being installed on the VM. It will be available on the VMM home screen.

* We must delete this image from the VMM home screen, and we will keep the .img file in the repository as shown in Figure 24.



***Figure 23: VMM – Home screen***



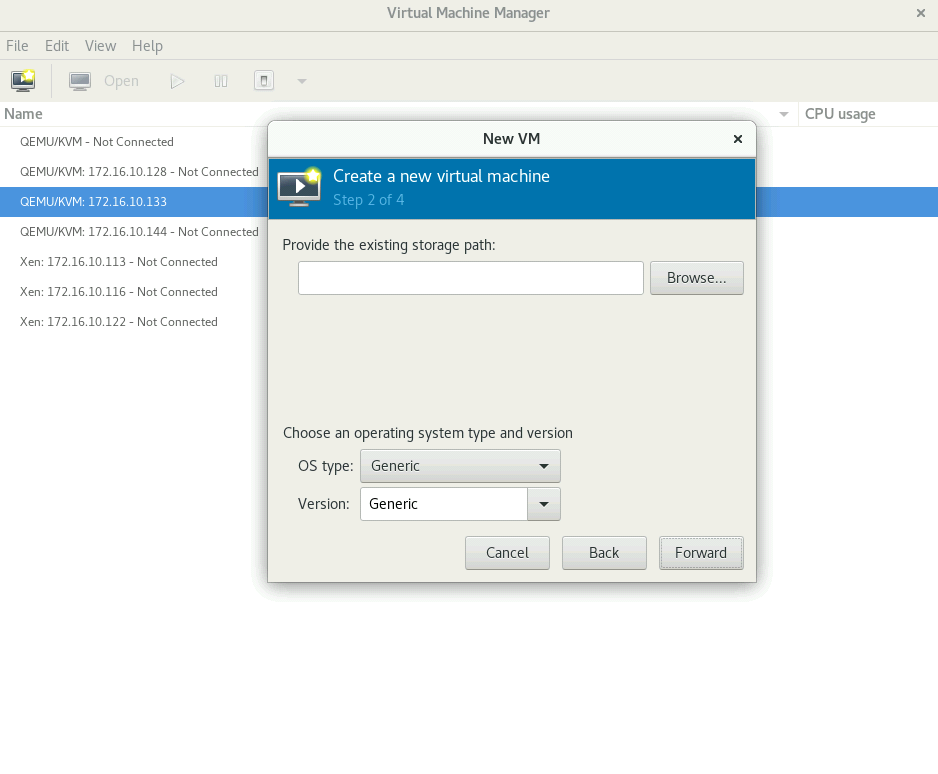
***Figure 24: VMM – Delete the VM Image Name from the VMM Home Screen***

**Step 14:**

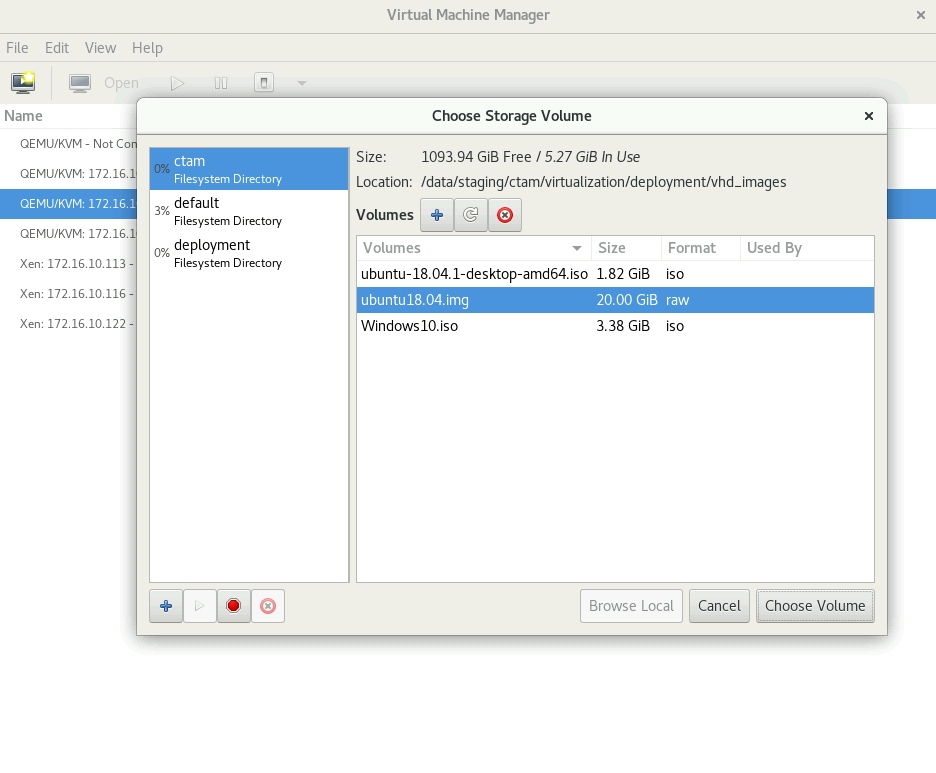
* Right click on the hypervisor and click on “Import existing disk image” to start the VM with the image created in the previous step (Figure 25).
* Click on forward button to proceed.
* Browse the .img file to build VM (Figure 26).
* Choose the ubuntu18.04.img image from the list as shown in Figure 27.
* Select the OS type and version from the drop down and click on forward button as shown in Figure 28.
* Configure the amount of RAM and CPU for the VM, as shown in Figure 29.
* Enter VM name and bridge name in the network selection area as shown in Figure 30.
* Click on the begin installation button as shown in Figure 31 to start the VM.



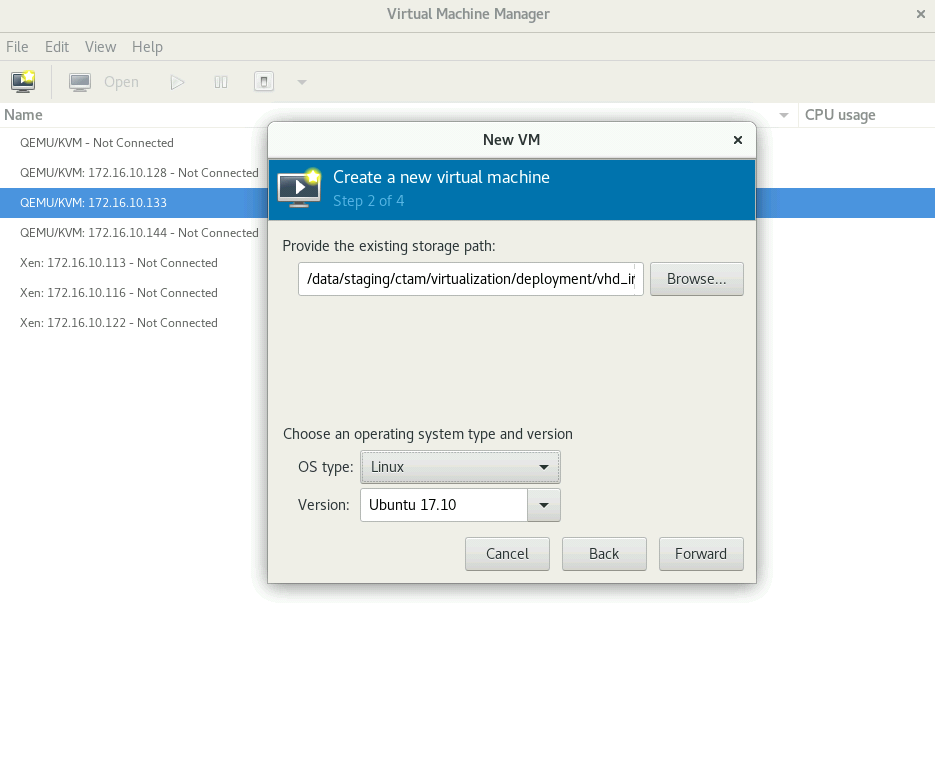
***Figure 25: VMM – Import Existing Disk Image***



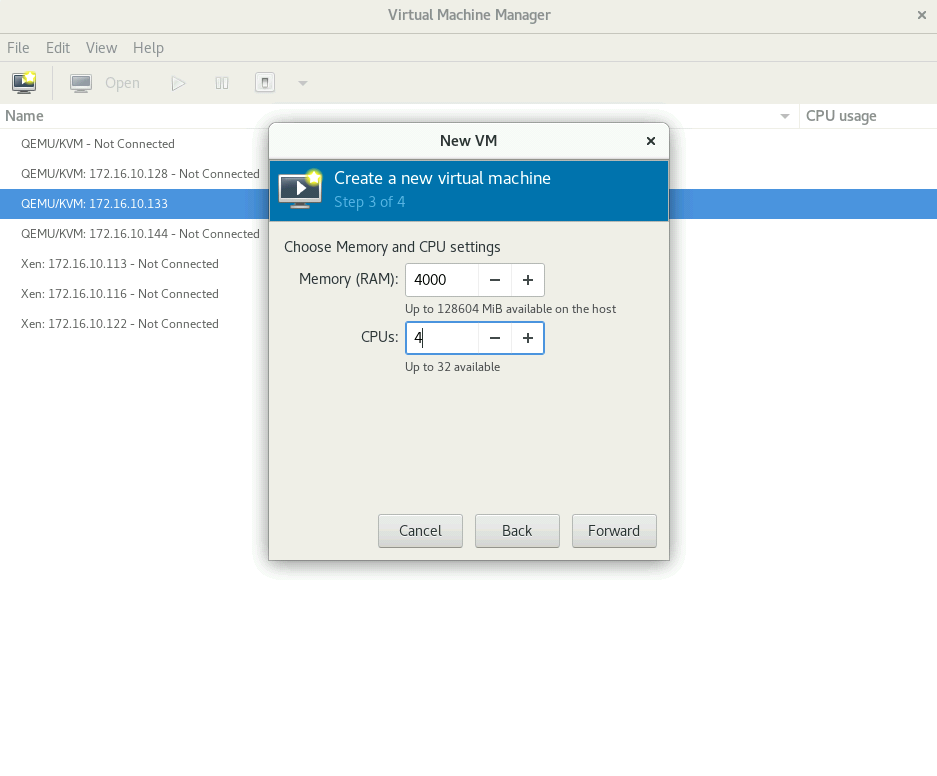
***Figure 26: VMM – Browse the .img File***



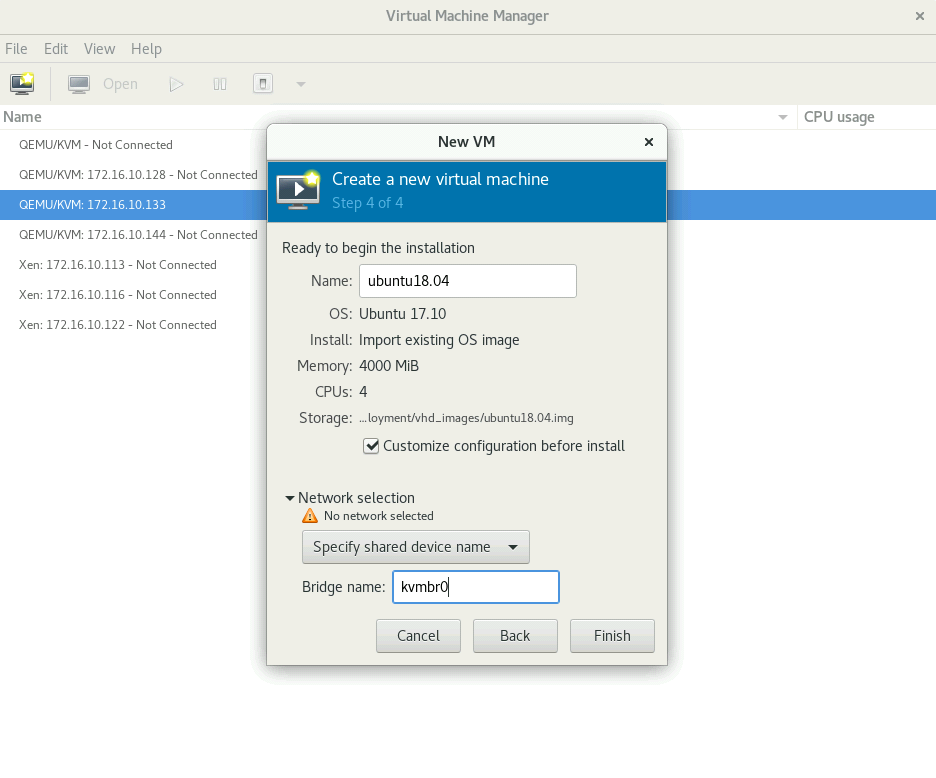
***Figure 27: VMM – Choose the .img File***



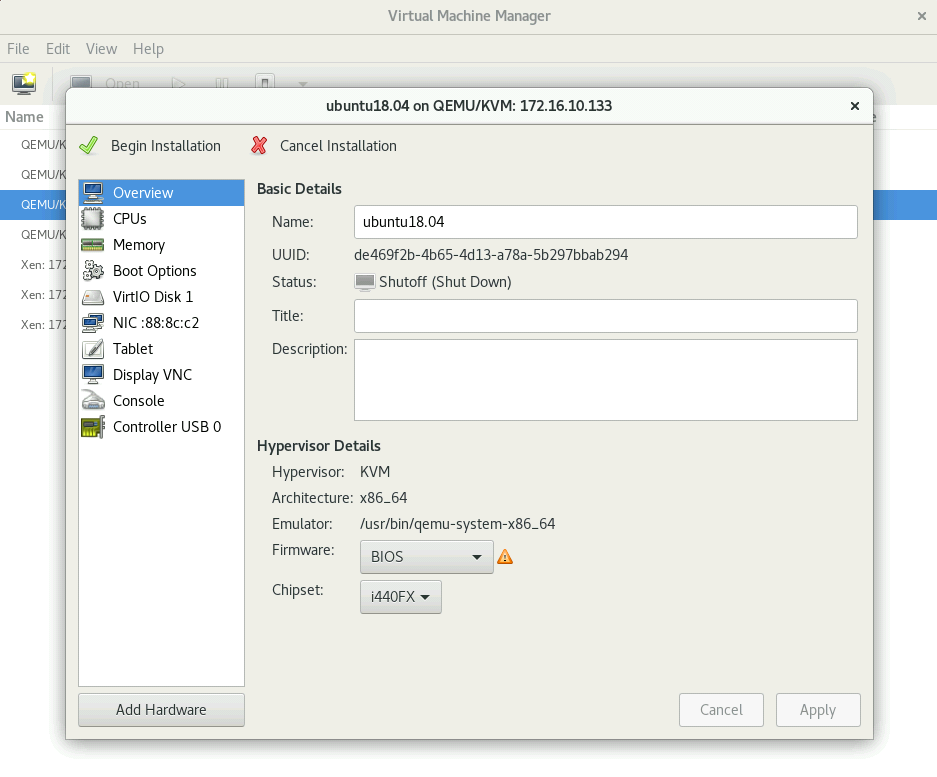
***Figure 28: VMM – Select OS type and Version***



***Figure 29: VMM – VM Configuration***



***Figure 30: VMM – Change VM Name and Bridge Name***



***Figure 31: VMM – Boot Device Order Setup***

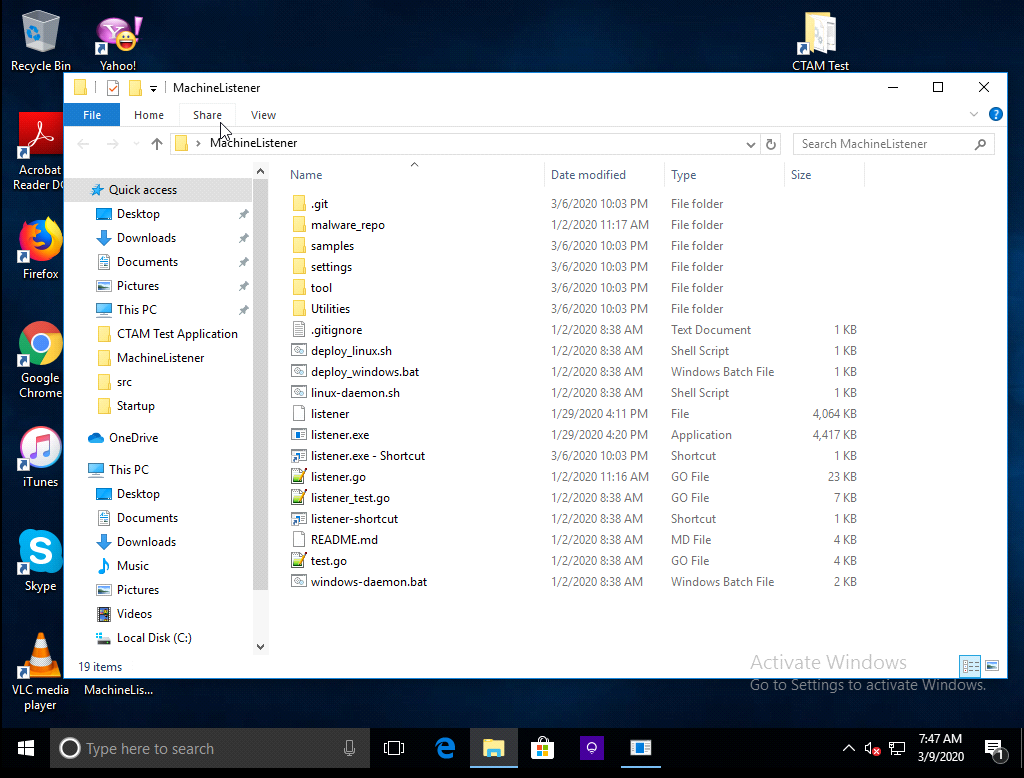
**Step 15:**

**Machine Listener Installation on VM:**

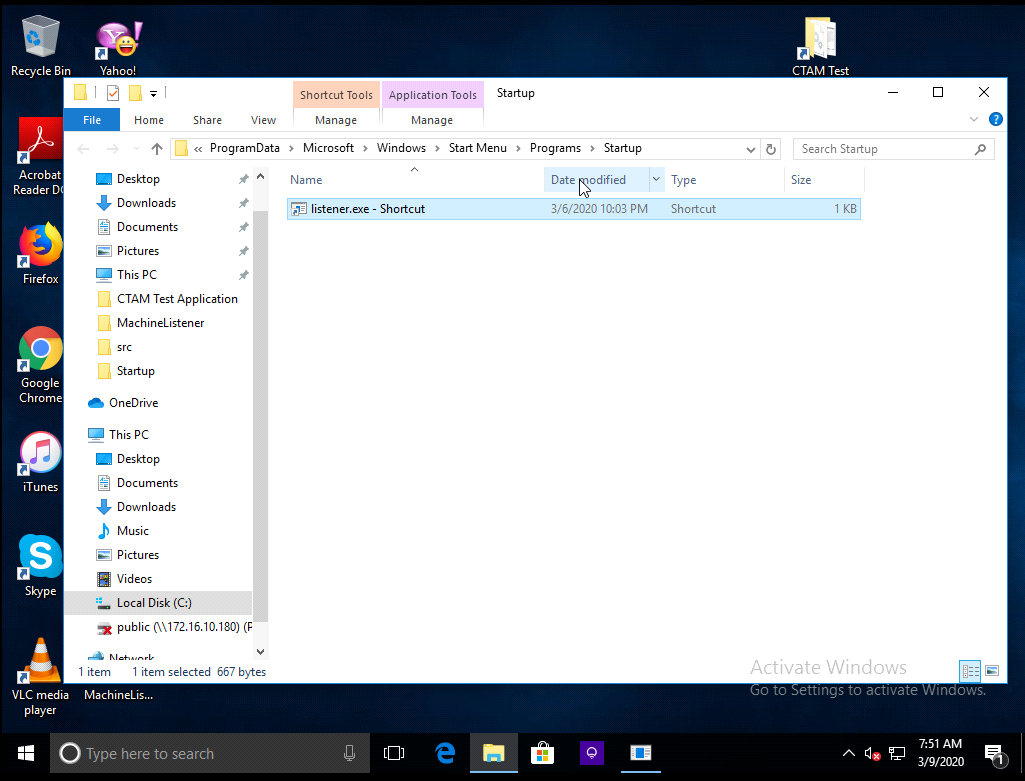
Machine Listener is an agent routine running on the Virtual Machine that listens to the call from the Test Control Center (TCC) and execute the tasks according the received commands. It extracts the system level information of the VM such as CPU utilization, memory utilization. The machine listener must be embedded into the virtual machine during the image creation process.

**Machine Listener setup for the Windows VM:**

* Open and start Windows Virtual Machine through Virtual Machine Manager.
* Copy the Machine Listener folder from the shared directory to the VM’s desktop.
* Create shortcut of the Listener.exe application (In case of Windows VM).
* The contents of the Machine Listener folder are as shown below in Figure 32.
* Run Win + R on the VM and run this command “shell:Common Startup” .
* A new window will open, copy the listener shortcut in this directory, as shown in Figure 33.
* Restart the VM, the Machine Listener will start by default at the startup.



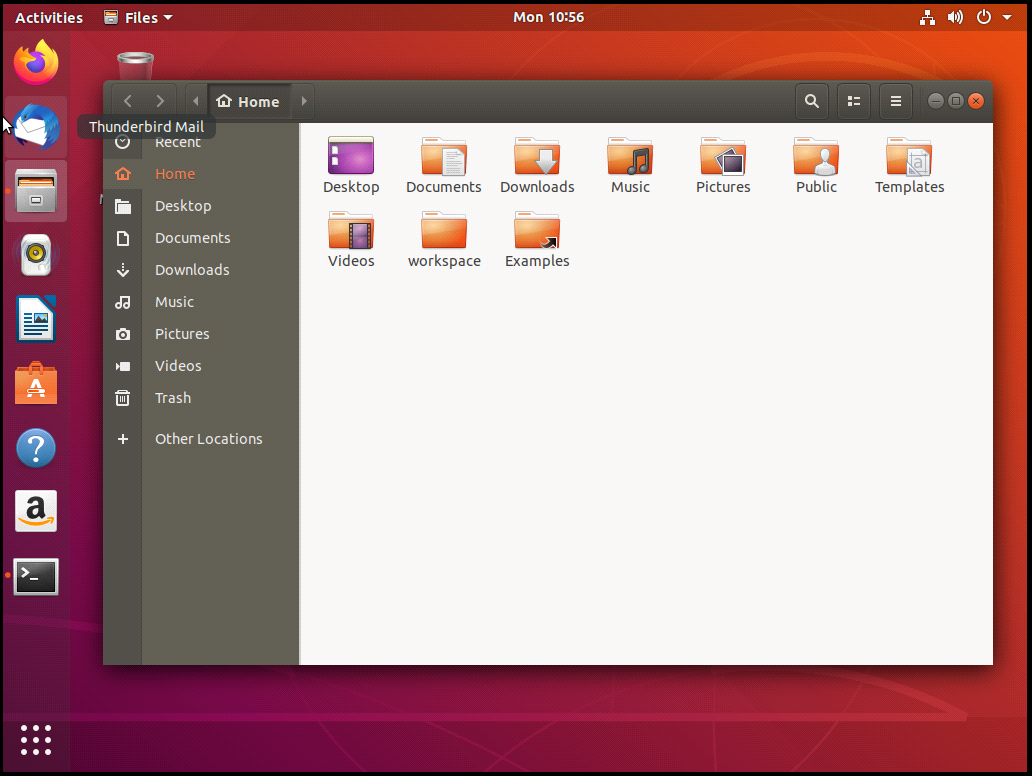
***Figure 32: Copy Machine Listener to VM Desktop from Share Drive***



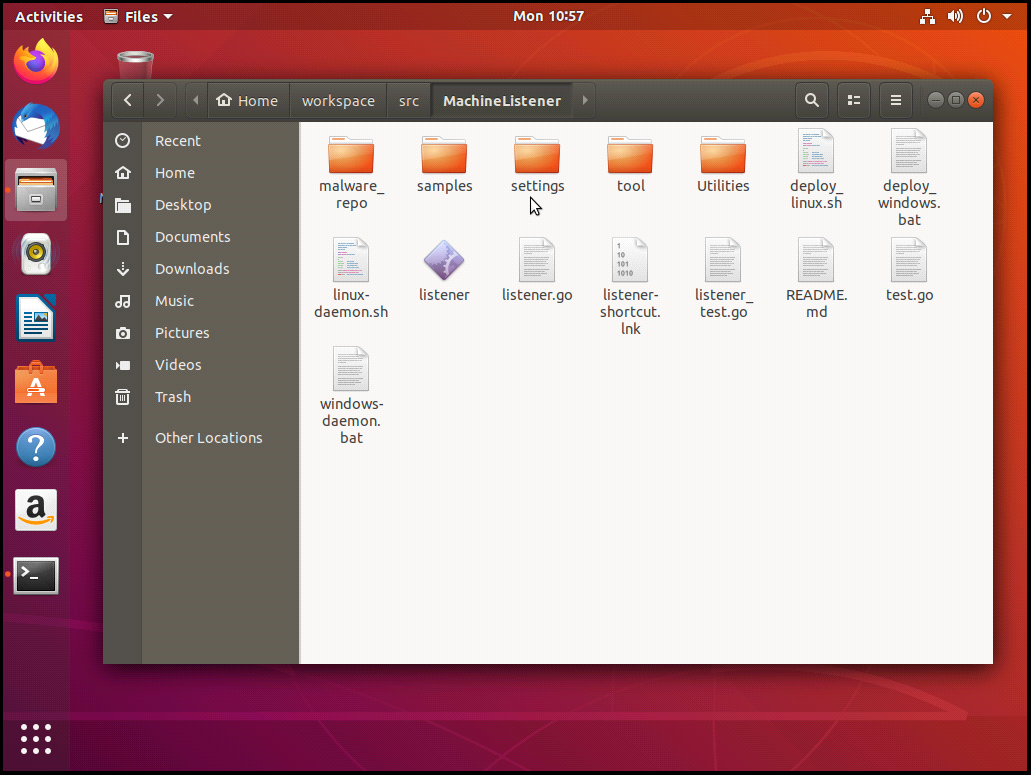
***Figure 33: Copy Machine Listener shortcut to the Startup Directory***

**Linux VM Machine Listener Setup:**

* Open and start Linux Virtual Machine through Virtual Machine Manager.
* Copy the **“workspace”** folder from the shared directory to the VM’s home directory.
* “workspace” directory contains the Machine listener folder with the same content as Windows (as shown in Figure 34).



***Figure 34: Copy Machine Listener to VM Desktop from Share Drive***



***Figure 35: Machine Listener in Workspace Directory***

* Click on the Activity tab on the top left corner of VM, and type and open startup application.
* Click on add button and fill the following information.

Name: Machine Listener

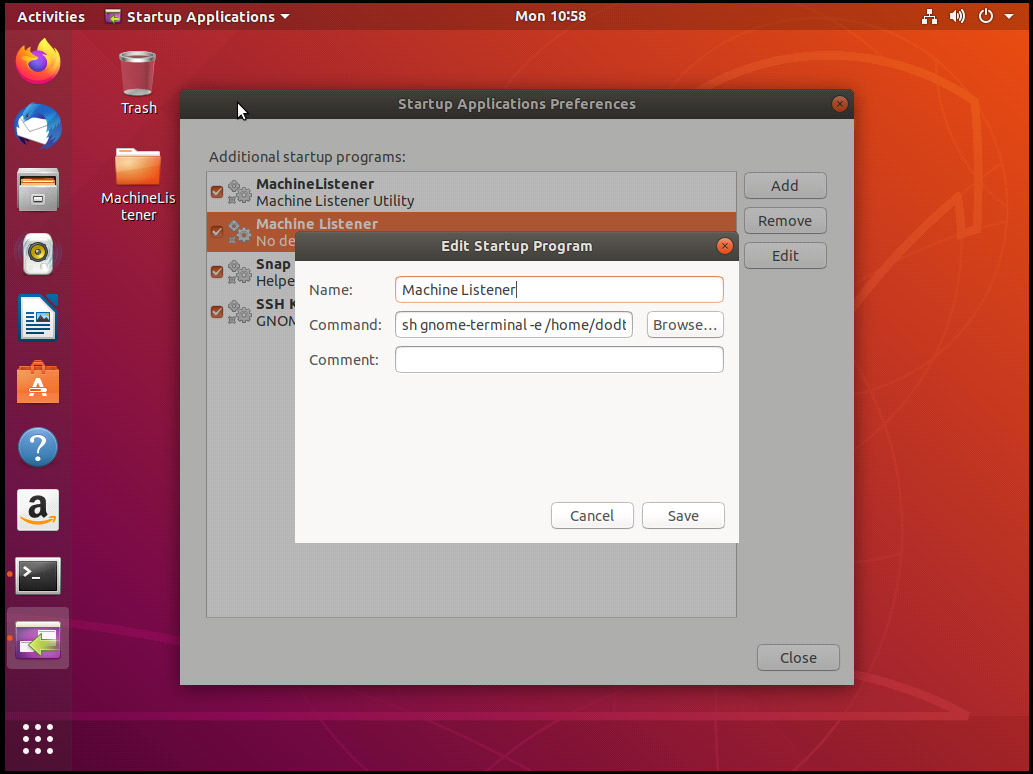
Command: sh gnome-terminal -e “path of linux-daemon.sh”

* Click on save.
* Click on the Activity tab on the top left corner of VM, and type and open startup application.
* Click on add button and fill the following information.

Name: Machine Listener

Command: sh gnome-terminal -e “path of linux-daemon.sh”

* Click on save.
* Additionally, run the following script from the Machine Listener directory in the terminal: ./deploy\_linux.sh
* Restart the VM, the Machine Listener will start by default at the startup.

***Figure 36: Copy Machine Listener shortcut to the Startup Directory***

**Step 16:**

* Newly created VM images will be available in the following location: /data/staging/ctam/virtualization/deployment/vhd\_images/ubuntu18.img, /data/staging/ctam/virtualization/deployment/vhd\_images/win10.img.
* Copy and rename the image to “Windows\_10\_Win\_10#Master.img” for windows and “Ubuntu\_1804\_LTS\_generic#Master.img” for Linux and store in the vhd\_images directory of the project.

5.0 Rekall Profile Creation:

Rekall is an advanced forensic and incident response framework. Rekall leverages debugging information provided by the operating system vendors to precisely locate significant kernel data structures. Rekall profiles contain the necessary memory offsets that are required to perform memory acquisition on the different operating systems. These offsets change based on the version of the kernel used on the specific operating system.

5.1 Rekall Profile for Windows VM:

Please follow the below mentioned steps to create Rekall profile for Windows VM.

* Create the JSON configuration file for the Windows domain.
* Get the debug information for the Windows kernel from the LibVMI vmi-win-guid tool.
* The important fields are:

PDB GUID: 684da42a30cc450f81c535b4d18944b12

Kernel filename: ntkrpamp.pdb

* Generate the JSON configuration file (make sure to adjust the kernel name and GUID as necessary):

|  |
| --- |
| cd /tmp  python3 ~/drakvuf/volatility3/volatility/framework/symbols/ windows/pdbconv.py --guid 684da42a30cc450f81c535b4d18944b12 -p ntkrpamp.pdb -o windows10-sp1.json  sudo mv windows10-sp1.json /root |

5.2 Rekall Profile for Linux VM:

Please follow the below mentioned steps to create Rekall profile for Windows VM.

* Install the target kernel's debug package to get the DWARF symbols for the kernel:

|  |
| --- |
| ssh [root@linu](mailto:root@linu)x  echo "Kernel version: "$(uname -r)  apt-get install linux-image-$(uname -r)-dbg  exit |

* Use the dwarf2json tool to convert the debug information into the required JSON configuration file. Make note of the Kernel version of the target and make sure to update the System.map and vmlinux paths to reflect the correct kernel version:

|  |
| --- |
| cd ~/drakvuf/dwarf2json  go build  sudo su  kpartx -a /dev/vg/linux  mount -o ro /dev/mapper/vg-linux1 /mnt  ./dwarf2json linux --system-map /mnt/boot/System.map-5.3.0- 0.bpo.2-amd64 --elf /mnt/usr/lib/debug/vmlinux-5.3.0-0.bpo.2- amd64 > /root/linux.json  umount /mnt  kpartx -d /dev/vg/linux  printf "linux {\n\tvolatility\_ist = \"/root/linux.json\";\n}" >> /etc/libvmi.conf |

6.0 CTAM Virtualization Configuration:

Please follow the below mentioned instructions to copy required virtualization modules into the hypervisor. All the required files will be provided on the external hard drive.

Please copy the CTAM directory, which contains all other required sub-directories to run the virtualization platform.

**Ctam:** This is the root directory, which consists of virtualization sub-directory.

**Virtualization:** This is the 2nd level directory, which contains of all other required sub-directories such as src, deployment and documentation.

**Sources Directory (src):** This directory contains the source code for the virtualization platform. It contains below mentioned sub-directories.

* **Introspector:** This directory contains the source code for the introspection module. It has two main files introspector.go and agent.go. In addition, it contains certain sub-directories that helps to perform data structure scans efficiently.
* **IntrospectorResources:** This directory contains the drakvuf introspection library, which is used to extract the system call traces from the virtual machines.
* **rekall-profiles**: This directory consists of the Rekall profiles for the Windows and Linux operating systems.
* **vhd\_images**: This directory contains the master image files for both the windows and Linux operating systems. These image files are used to create VM templates, which are used to create virtual machines.
* **vhd\_templates:** This directory consists of the templates of the images for both windows and Linux operating systems.

**Deployment Directory:** This directory contains the executables and required dependencies to run the introspector. Details of the sub-directories are mentioned below.

* **rekall-profiles**: This directory consists of the Rekall profiles for the Windows and Linux operating systems.
* **vhd\_images**: This directory contains the master image files for both the windows and Linux operating systems. These image files are used to create VM templates, which are used to create virtual machines.
* **vhd\_templates:** This directory consists of the templates of the images for both windows and Linux operating systems.
* **Settings:** This directory contains the introspector configuration file “introspector.conf”. This file contains all the configuration details required to run the introspector. It also contains "introspectorTccCommands" file, which contains the path required by the Test Control Center to start and stop introspector and to update the “introspector.conf” file.
* **Introspector Executable:** This is the main executable file, which needs to be run with admin privileges to start the introspector. (sudo ./introspector)
* **IntrospectorResources:** This directory contains the drakvuf introspection library, which is used to extract the system call traces from the virtual machines.

**Documentation Directory:** This directory contains the all the required documentation for the virtualization platform.

**Introspector TCC Commands File:**

The contents of the "introspectorTccCommands" file are as shown below.

*STOP\_INTROSPECTOR =*

*START\_INTROSPECTOR = cd /ctam/virtualization/deployment/Introspector/ && sudo ./Introspector*

*MODIFY\_INTROSPECTOR = cd  /ctam/virtualization/deployment/Introspector/settings && cat introspector.conf*

Administrator need to update this file, if path of introspector is changed.

**Directory path:** /ctam/virtualization/deployment/setting/introspectorTccCommands

**Introspector Configuration File:**

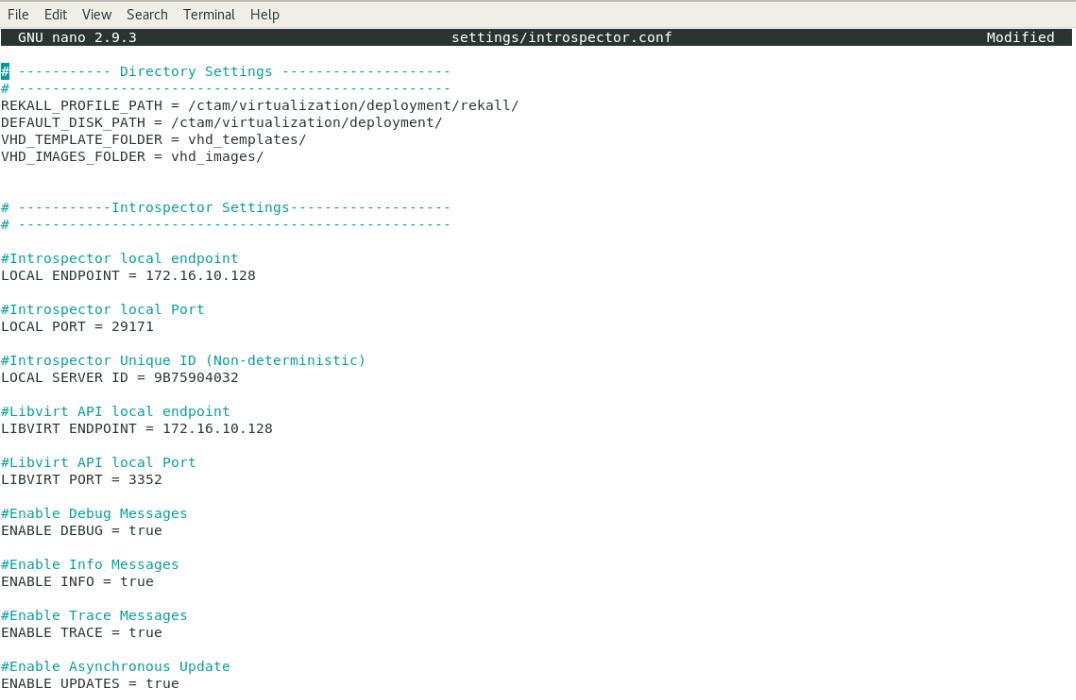
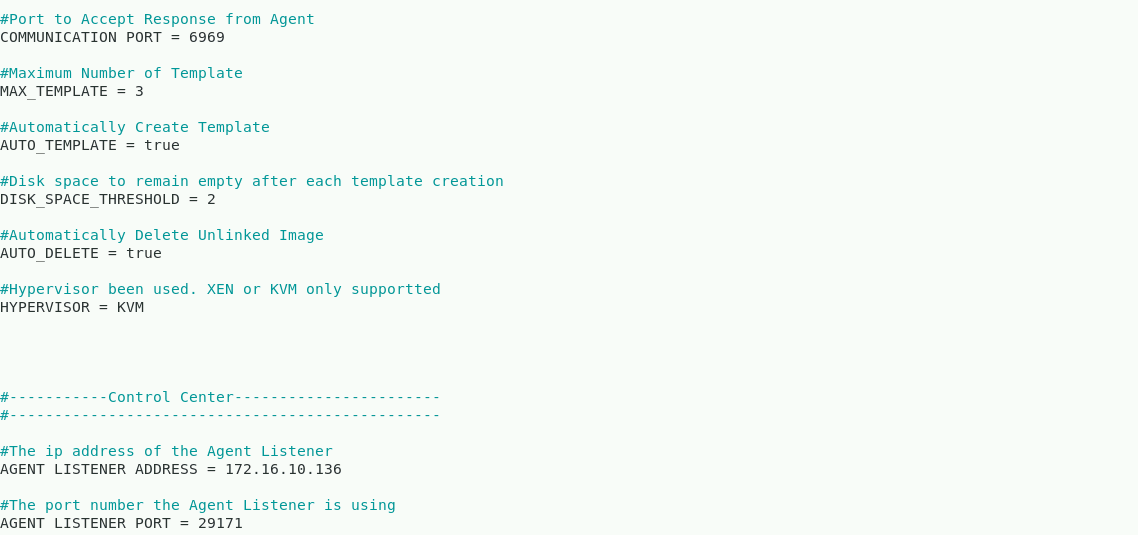
Administrator need to update the introspector configuration file in the following directory.

**Directory path:** /ctam/virtualization/deployment/setting/introspector.conf

**Introspector Configuration Details:**

|  |
| --- |
| INTROSPECTOR LOCAL ENDPOINT IP: This is the IP address of the hypervisor where the introspector is running.  INTROSPECTOR LOCAL PORT: This is the port number on which the introspector is communicating on the network.  LOCAL SERVER ID: This is the Unique ID for the introspector.  LIBVIRT ENDPOINT: This is the IP address of the Libvirt API local endpoint.  LIBVIRT PORT: This is the port number of the Libvirt API local endpoint.  ENABLE DEBUG: It is used to enable/disable the Informational messages from the introspector.  ENABLE ASYNCHRONOUS UPDATE:  COMMUNICATION PORT: This is the Port to accept response from the Agent.  MAX TEMPLATE: This is the maximum number of VM templates that the user can create.  AUTO TEMPLATE: Allowed values Yes/No, If yes templates created automatically. If No, then templates need to be created manually.  DISK SPACE THRESHOLD: This is the amount of disk space that remain empty after each template creation.  AUTO DELETE: This enables and disables the automatic deletion of the un-linked image.  AGENT LISTENER ADDRESS: This is the IP address of the agent listener to communicate and send data to database.  AGENT LISTENER PORT: This is the port of the agent listener.  SETUP COMPLETE: This is by default “TRUE”  LAST UPDATED: This is the timestamp for the last update on the configuration file.  ALLOW EDIT: This must be TRUE to enable editing on the configuration file.  REQUIRE ADMIN: It is by default FALSE, as it does not require admin privileges to the access configuration file. |

Introspector Configuration file is as shown below in Figure 37.



***Figure 37: Introspector Configuration File***

**Creating the Introspector Executables:**

Please execute the following commands in the Introspector directory to make the executables of the Introspector.go and agent.go program files.

|  |
| --- |
| export GOPATH=/data/staging/ctam/virtualization/  make |

7.0 Acronyms:

|  |  |
| --- | --- |
| KVM | Kernel-based Virtual Machine |
| CPU | Central Processing Unit |
| AMD-V | Advanced Micro Dynamics- Virtualization |
| HWE | Hardware Enablement |
| VCPU | Virtual Central Processing Unit |
| VMM | Virtual Machine Manager |
| JSON | JavaScript Object Notation |
| GUID | Globally Unique Identifier |
| VM | Virtual Machine |
| TCC | Test Control Center |
| OS | Operating system |
| RAM | Random Access Memory |