Politecnico di Torino

Computing Virtualization with KVM

Fulvio Risso

November 16, 2023

License

Acknowledgments

The author would like to thank all the people who contributed to this document.

Contents

1 Introduction

This lab aims at practicing with the creation and configuration of virtual machines (VMs) in a computing virtualization framework, namely **Linux KVM**[1]. This is the default hypervisor in Linux, oriented to production-grade virtualization, available only in the above OS, targeting preferably console-based guests such as Linux servers, although GUI-enabled Guest OSes can be supported as well.

As a more complex scenario, this lab additionally simulates a configuration in which two hosts that belong to different LANs can exchange traffic through a router, connected to the same switched network of the previous hosts.

[1] `https://www.linux-kvm.org/`

2 Practicing with virtual machines

While KVM supports the creation of a new VM installing the guest OS from scratch, this lab suggest to practice with another option that is much more common in cloud environment: setting up an already installed VM in KVM. In fact, often system administrators start from an already installed VM and then customize it according to their needs instead of installing a GuestOS from scratch, which requires more time.

For this reason, major Linux distributions, such as Ubuntu, already provide pre-installed VM targeting the cloud environment:

- **Ubuntu Cloud Images** (`https://cloud-images.ubuntu.com/`) are the official pre-installed disk images that have been customized to run on public clouds that provide Ubuntu Certified Images, Openstack, LXD, and more.
- **Ubuntu Minimized Cloud Images** (`https://cloud-images.ubuntu.com/minimal/`) are the official pre-installed disk images that have been customized to have a small runtime footprint in order to increase workload density in environments where humans are not expected to log in. More information at `https://ubuntu.com/blog/minimal-ubuntu-released`.

Note (1): this lab will use the *Ubuntu Cloud Images*.

Note (2): here we just assume that KVM (and the additional modules such as the `virt-manager` GUI) are already set up in your environment.

2.1 Physical and logical setup

This lab requires a physical setup such as depicted in Figure 2.1. Each student will be provided with two VMs running in the POLITO datacenter on the CrownLabs platform (`https://crownlabs.polito.it`), one acting as Client and one representing the student's server.

The Client VM represents our **management host**, which is used to configure our (remote) **server** environment. The student will connect to the graphical interface of the client VM through CrownLabs.

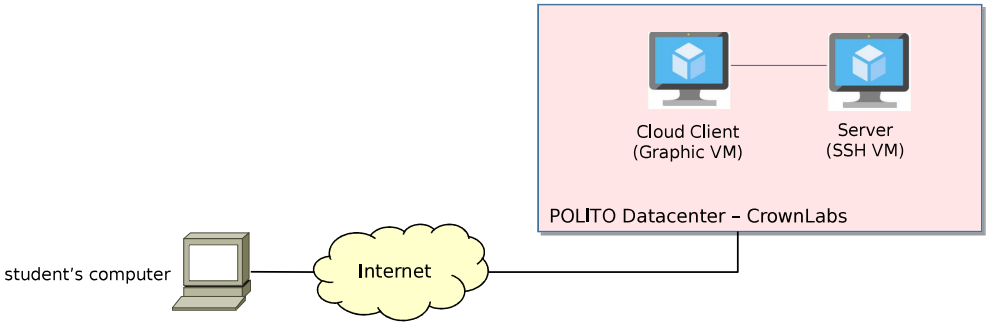


Figure 2.1: Physical setup: how to connect to your virtualized resources.

On the **server** side, this lab expects to create the configuration depicted in Figure 2.2: two VMs connected to a L2 virtual switch (Linux Bridge), which is then connected to the Linux stack and from there to the Internet. KVM will insert all the required NAT rules and routing table in order to allow VM1 and VM2 to connect to the Internet.

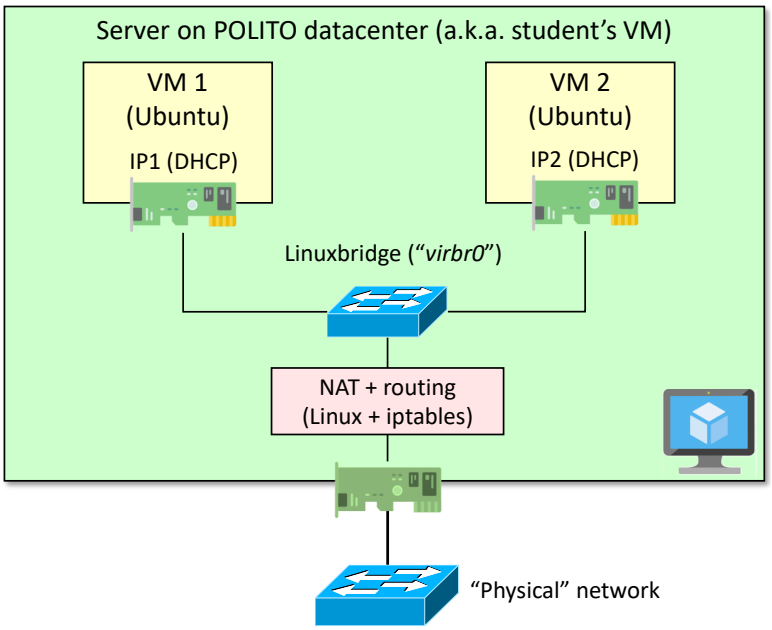


Figure 2.2: Logical setup: configuration that is expected in the target server at the end of the lab.

Please note that, being the student's server a virtual machine, this lab will use the concept of *nested virtualization*, as we will create a VM inside another VM.

2.2 Prepare the lab environment

2.2.1 Client VM (a.k.a. management host)

The **management host**, i.e., the Client VM, is used to control the remote hypervisor, running on the remote server.

This machine will use the SSH tool to connect to the remote server. It will also exploit the GUI-based Virtual Machine Manager (`virt-manager`), which makes easier the interactions with the remote hypervisor.

In fact, while the KVM hypervisor can be controlled by using command-line tools (e.g., `virsh`, to create VMs, and more), this is not that easy at the beginning. Instead, `virt-manager` provides a graphical interface (hence, cannot be directly executed on the server) and it can control a remote hypervisor (hence, the VMs running on it) by issuing the proper commands through an SSH connection. Hence, `virt-manager` facilitates the interactions with the remote hypervisor that works in console-mode only (no GUI).

2.2.2 Server VM (a.k.a. student's server)

The server comes with all the required tools preinstalled, hence nothing has to be done in order to prepare this environment. However, it is useful to configure a proper SSH key pair on Server VM for the authentication of Client VM (to avoid inserting the password multiple times).

Access to the server

First, you have to get the IP address of the server, the KVM Virtual Machine from the CrownLabs dashboard. The access to the server is possible with the following command (issued **on your management host**, i.e., the Client VM):

must go to "active" section and press the information (i) button

10.100.192.3

```
ssh crownlabs@IP_of_KVM_VM
```

The password for the login is `crownlabs`. If you are successfully authenticated, let's configure a proper SSH key pair.

Setting up SSH authentication

Using Key pairs relies on asymmetric authentication, and it is normally a preferable way to connect to an SSH server. This is particularly interesting because you can disable password authentication and the possibility of dictionary attacks, which are a major threat if your host is exposed on the Internet.

In addition, key-based authentication can be secure but also completely not-interactive. This opens the door to completely automated configuration of the host without the need to install a dedicated agent and is used by very popular tools such as Ansible.

First step is to generate the key pair to be used for the authentication. This is done issuing (**on the client VM**) the command `ssh-keygen` and accepting the default settings by pressing "enter" multiple times. This operation will produce a public key (`id_rsa.pub`) and a private key (`id_rsa`), which are both stored by default in the directory `~/.ssh`.

In order to use the generated key pair to authenticate on a remote system, we have to first copy the public key on the remote system. We can do this by issuing, on client VM, the following command:

```
ssh-copy-id crownlabs@IP_of_KVM_VM
```

Also in this case it will ask for the password in order to be authorized in copying the key on the remote server. The next time we try to ssh into the remote server, we get access to the system without typing any password since the aysmmetric keys will be used.

Configure permissions

Once you accessed the server through SSH, make sure that the `crownlabs` account has the permissions to interact with the `libvirt` daemon, adding it to the corresponding user group:

```
sudo adduser crownlabs libvirt
```

2.3 Configure proper storage folders on the server

IMPORTANT: unless explicitly specified, all the instructions below apply to the server. Hence, this lab assumes that the student is already logged in the server.

In order to create new VMs, we may need to store a set of ISO images (i.e., OS install disks, or configuration disks that will be used by `cloud-init`) and a set of VM images (i.e., the actual installed VM image) on the KVM server.

To avoid polluting your system, you should create two new directories, one to keep your ISO (install) disks, and one for your storage pools (installed VM images). The default storage pool in KVM is `/var/lib/libvirt/images`; we suggest to create instead two distinct folders under your home directory: `~/kvm-iso` and `~/kvm-pool`, as follows:

```
/home/crownlabs/
         |
         +----/kvm-iso
         |
         +----/kvm-pool
```

2.4 Download the Ubuntu Cloud image disk

Open a terminal and download the disk image of your choice, e.g., the latest LTS release, from the Ubuntu Minimized Cloud Images website (e.g., `wget https://cloud-images.ubuntu.com/focal/current/focal-server-cloudimg-amd64.img`).

- **Save** this image in your home folder;
- **Copy** the image into folder `~/kvm-pool`, while keeping the original in your home folder (you may need this file later).

2.5 Customize the disk image with your personal data

Cloud images are operating system templates and every instance starts out as an identical clone of every other instance. Before running the instance, images may need to be customized, e.g., with some additional software packages, by modifying some configuration parameter (e.g., network), and by creating the proper credentials for login[1].

In our lab you need simply to enable a user account in order to be able to login. We present here a possible procedure using `cloud-init`, but there exists also other alternatives to customize a disk image.

2.5.1 Cloud-init

`Cloud-init`[2] (`https://cloud-init.io/`) is one of the most widely used tool for passing some initialization parameters to a VM at boot time. `Cloud-init` relies on two text files that are read

[1] For security reasons, the default user account on the above image, i.e., user 'ubuntu', has the login disabled from both console and SSH.

[2] Latest documentation for `cloud-init` is available at `https://cloudinit.readthedocs.io/en/latest/`.

either from a predefined network location, or (such as in this lab) from a disk drive that is mounted to the VM at boot, as a CD-ROM device.

This procedure adds a new user to the VM and pushes the user's SSH public key in the VM itself, so that you will be able to login using your private SSH keys. In addition, it also adds a password for the default user (`ubuntu`), allowing that user to log-in only through the console. This procedure may be convenient in case, for any reason, you lose the connectivity with your server, hence logging in via console is the only way to recover your VM. In any case, be careful that the access via username/password is considered less secure compared to the use of a public/private key.

1. If you do not have them already, create your pair of public/private keys with `ssh-keygen` (as it is done for client VM in the previous section 2.2.2).
2. Once done, create a new text file named `user-data`. For who is not very familiar with console-mode text editors in Linux, we suggest `nano`, which allows a reasonable user-friendly interaction (compared to earlier software such as `vim`).
3. Write in the above file the text **exactly** such as in the following example, while the `ssh-rsa` section should be replaced with your public key (i.e., all the text in your public key file, usually stored in your home folder, i.e. `~/.ssh/id_rsa.pub`) in the `ssh_authorized_keys` section of the above file:

drag the file to the drive crownlabs and then use the shared file system from the server /media/MyDrive/cloud_config.txt

```
#cloud-config
users:
  - default
  - name: ubuntu
    ssh_authorized_keys:
      - ssh-rsa AAAA...DSHEx crownlabs@cloud-kvm
    sudo: ALL=(ALL) NOPASSWD:ALL
    groups: sudo
    shell: /bin/bash
# Password for the 'default' user (i.e., 'ubuntu' in Ubuntu cloud images)
password: ubuntu
# Password does not have to be changed at first login
chpasswd: { expire: false }
# Uncomment this line if you want to use username/password when
#  logging in also from network (ssh)
#ssh_pwauth: true
```

This code snipped is attached to the PDF file as "snippets/cloud-init.txt"

Warning: the `user-data` file relies on proper indentation to be correctly parsed. Pay attention to NOT use tabulations and ensure a consistent number of spaces across the file.

4. Create a second text file, called `meta-data`, with the following content:

   eco "local-hostname: test-vm1" > meta-data

   ```
   local-hostname: test-vm1
   ```

5. Use the tool `cloud-localds` to create a configuration disk, which is called *seed* in `cloud-init` terminology. Type the following command:

   ```
   cloud-localds seed-init.iso user-data meta-data
   ```

   to produce a file called `seed-init.iso`, which simply contains the above two files with a standard name, whatever name were assigned to your original files, i.e.:

```
root/
 |
 +----/user-data
 |
 +----/meta-data
```

6. Copy the produced ISO into your `~/kvm-iso` folder:

```
cp seed-init.iso ~/kvm-iso/.
```

Cloud-init and persistent disks

This lab uses KVM as virtualization framework, which makes persistent any change to your OS image. For instance, once you use your `cloud-init` file to configure your users, the above changes are written on the OS image on disk and are kept for the future runs of the above image. This is different from what happens in other systems (e.g., OpenStack), in which the OS image is never modified.

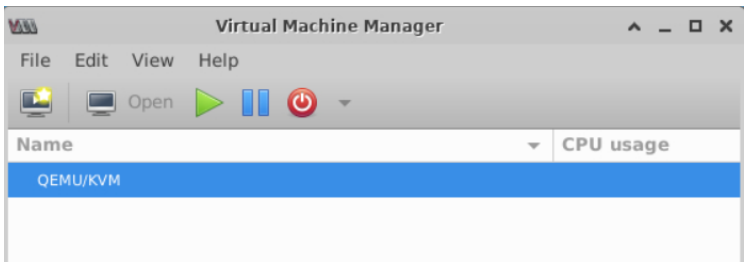
This brings us to two observations.

1. A first difference is that in our case the `cloud-init` disk may be required only in the first execution of the OS image, since in the following executions the added user will already be present; hence, we may modify the hardware profile of the VM by removing the `cloud-init` disk, without any effect. Instead, the `cloud-init` initialization disk must be always provided in other systems (e.g., OpenStack) as the OS image is always the original one.
2. A second difference is that, in our environment, replacing an already applied `cloud-init` image with another one, with a different configuration data, may not work. In fact, `cloud-init` may not be able to change the configuration of an already present user, while it should be able to add a new user with another key. As a consequence, if the student wants to change its login credentials and by creating another `cloud-init` disk, it would be safer to re-initialize the OS image (e.g., replace the current OS image with the original Ubuntu cloud image) in order to be sure that `cloud-init` will work again.

2.6 Create a new VM

In this step we operate on the management station, exploiting the GUI-based Virtual Machine Manager (`virt-manager`) that we presented in Section 2.2.1.

1. **Start Virtual Machine Manager** by typing `virt-manager` either in a console window, or in the Ubuntu launch manager. You will see something like in Figure 2.3.



not from the server!

Figure 2.3: Virtual machine manager: main window.

2. Add a new connection to the server from the graphical interface (`file -> add connection` and then set the username and IP of the server) as in Figure 2.4.

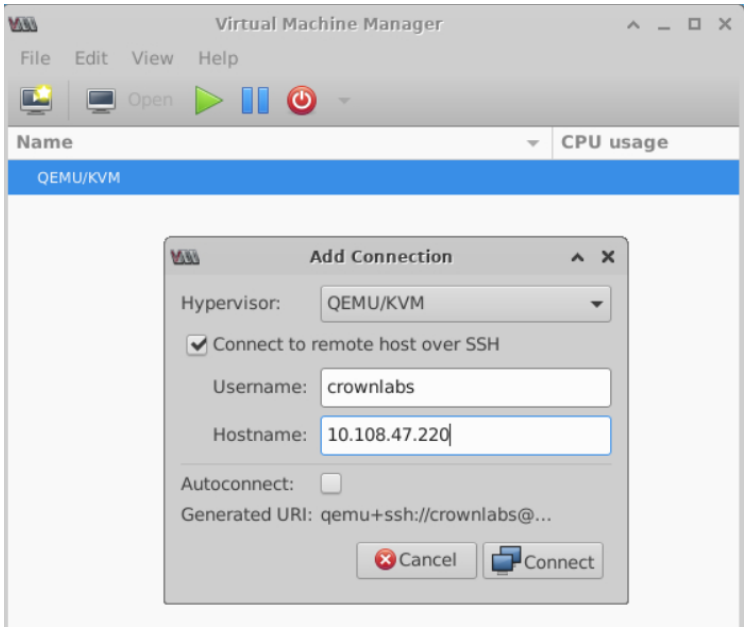


Figure 2.4: Virtual machine manager: add connection window.

Note (1): in case you use `virt-manager` to connect to a remote hypervisor through SSH, it is strongly encouraged to configure both server and client with SSH keys, as libvirt needs to open multiple SSH connections, one for each device (e.g., video card, keyboard, audio card, etc) that has to exchange data with the remote machine. Hence, relying on classical `user/password` mechanism would be rather annoying as the user would be requested to type his password several times. This should have already been done in section 2.2.2.

Note (2): in case you prefer to connect to the remote hypervisor through username/password, you should install the `ssh-askpass` package on your client machine (`sudo apt install ssh-askpass`), so that `virt-manager` can open a new window for you to type the password in it.

3. **Create a new VM** by going to *File >New Virtual Machine* (Figure 2.5). You can either:

   - Select **Import existing disk image**, then click Forward: select this option if you want to create a new VM by using a pre-installed disk. Note that the preinstalled image should be in one of the formats supported by KVM (e.g., QCOW/QCOW2, Vmware VMDK).

     Note: this is the preferred choice in this lab, as we have already a pre-installed disk, namely the Ubuntu Cloud Image retrieved in Section 2.4.

   - Select **Local Install Media** (ISO image or CDROM), then click Forward: select this option if you want to install a new VM using the installation disk. Note that the install disk should be in ISO format.

   This opens a window such as in Figure 2.6. In either cases, when you try to select the location of your image (either an ISO or a pre-installed image) by clicking on *Browse*, a new window *Choose Storage Volume* will appear (Figure 2.7), which contains only the `Default` storage pool, i.e., `/var/lib/libvirt/images`. Here you should configure the additional folders you created before:

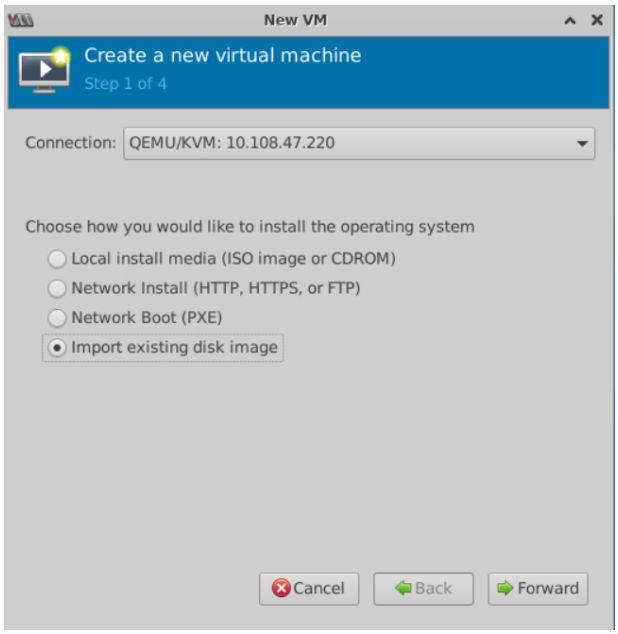


Figure 2.5: Create a new VM: step 1.

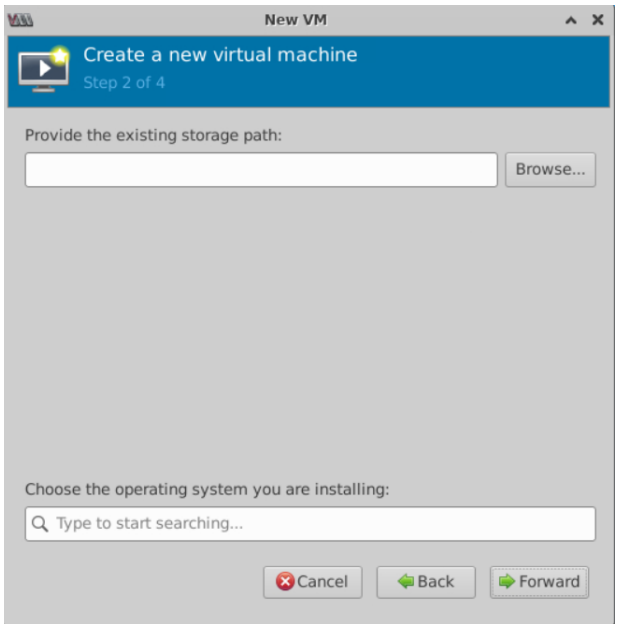


Figure 2.6: Create a new VM: step 2 (before choosing storage path).

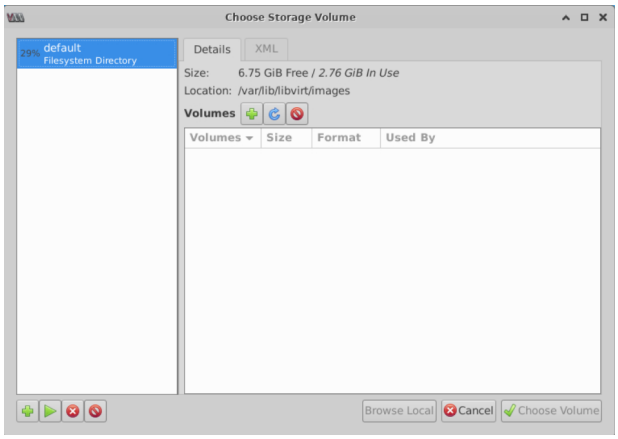


Figure 2.7: Available storage volumes (before adding the new storage pools)

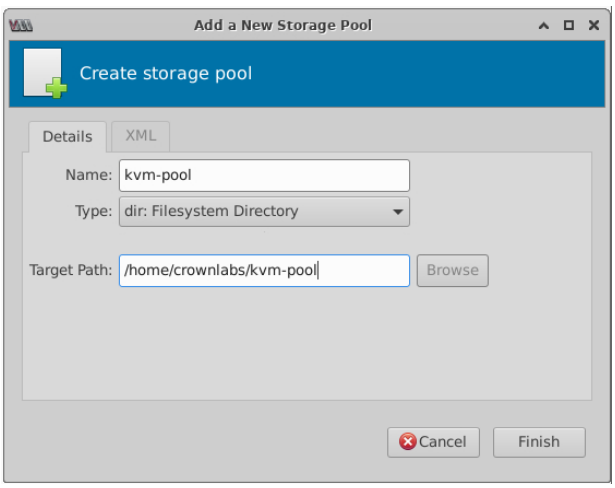


Figure 2.8: Create a new storage pool.

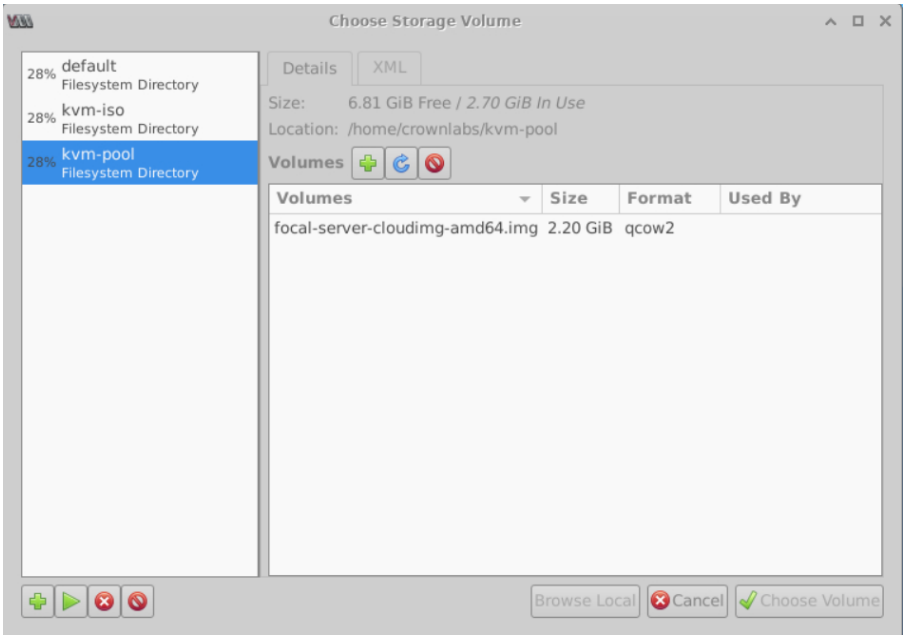


Figure 2.9: Available storage volumes (after adding the new storage pools).

- Select the '+' icon (**bottom left** in the previous window), to add a new storage pool.
- Choose a name for the new storage pool (e.g., `kvm-pool`), with type "`dir: Filesystem Directory`"; select the folder that has to be used for this storage pool (e.g., `/home/crownlabs/kvm-pool`), as in Figure 2.8, and press *Finish*.

  Note: if you are connected to a remote machine, the *Browse* button may not work; in this case you have to type the desired *Target Path* manually.
- Repeat the above steps to add also the `~/kvm-iso` folder as another storage pool.

4. Now that you are back at the *Choose Storage Volume* screen, you should see the default and your new storage pool in the left pane, with all the disk images/ISO that are available in that folder[3] (Figure 2.9). Select the disk image/ISO you want and click *Choose Volume*, you will return to a window such as in Figure 2.10. Here, select the type of the OS you are installing (*Linux*) and the version (*Ubuntu 20.04*). Once done select *Forward*.
5. Select the memory and the CPU to allocate to the above VM; given the constrained environment we are on, 512MB RAM and 1 CPU looks appropriate (Figure 2.11).
6. Once you reach last step (Figure 2.12), select "*Customize configuration before install*" in order to modify the virtual hardware of your VM and make the following modifications to the suggested profile:

   - Remove all unnecessary hardware (e.g., Display Spice, etc), leaving the minimal definition as shown in Figure 2.13;
   - Add a new **disk device** that is used to load the `cloud-init` configuration we created in Section 2.5.1.

     Detailed steps: Select *Add Hardware* button, then *Add a new storage device*, then select `Disk device` as device type, then "*Select or create custom image*" and choose the ISO you created in Section 2.5.1).

[3] In case no disk images/ISO appear, refresh the window by clicking on the circular arrow, near to the text **Volumes**.

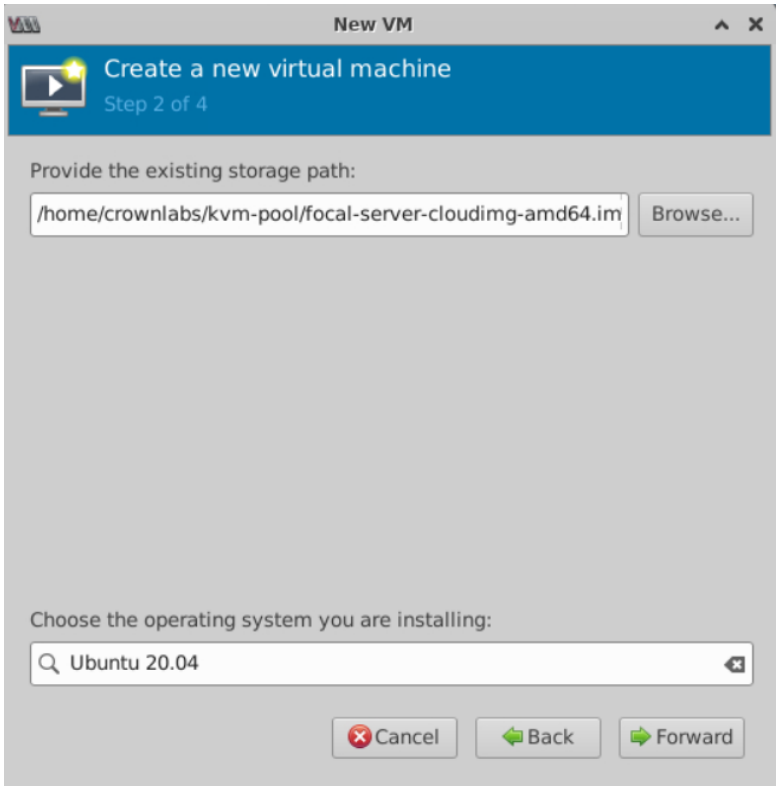


Figure 2.10: Create a new VM: step 2 (after choosing storage path).

Note: please note that the second disk should be a *disk device*; *CD-ROM devices* do not work.

- Expand the "*Network Selection*" section (expanding the details of the NIC hardware);
- Make sure the VM is connected to the "default" virtual network, with NAT. This enables your VM to connect to the Internet using the IP address of the hypervisor, while having a bridged connectivity to any other VMs connected to the same bridge.

You are now ready to start the VM with the button "`Begin Installation`", and log-in.

Start and customize the VM

or "login" command

You can now start the VM and login in the new machine, with the credentials you set before (user: `ubuntu`, password: `ubuntu`). It is preferred to login in the new machine using the SSH key configured in the cloud-init file. We can connect to the new VM by issuing, **on the remote server**, the following:

```
ssh ubuntu@IP_new_VM
```

Note: the IP address of the new VM could be found in different ways. One option could be to read it within the lines of logs printed when the VM is created, another one could be accessing to the VM via username/password to then found out its address (e.g., `ip addr`), or finally by issuing on the remote server the command `sudo virsh net-dhcp-leases default` to find out which address has been assigned to the created VM.

Figure 2.11: Create a new VM: step 3.

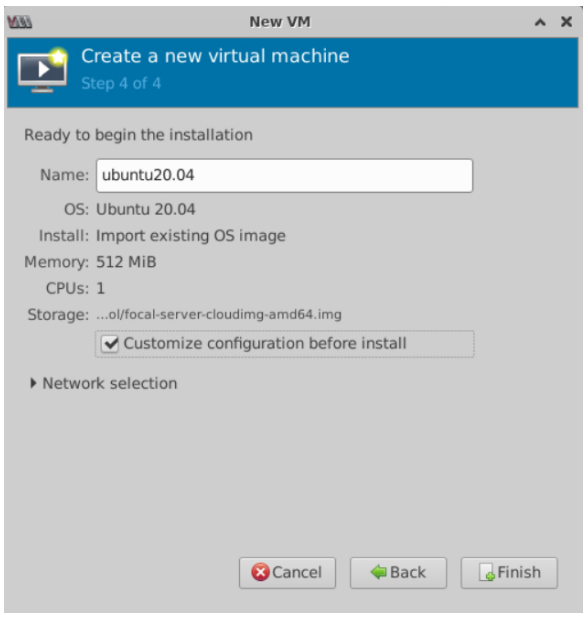


Figure 2.12: Create a new VM: step 4.

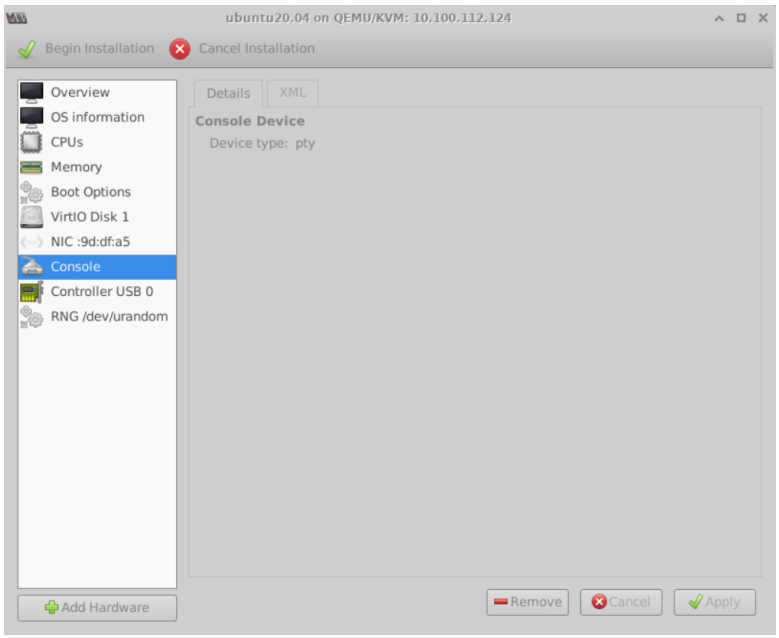


Figure 2.13: Virtual machine manager: minimal hardware.

Now we have to customize our environment, installing the `iperf3` tool, which will be used to generate some traffic and test the performance among VMs, and the `iputils-ping` package, which contains the `ping` utility:

```
sudo apt update
sudo apt install iputils-ping iperf3
```

create two different iso and img

2.6.1 Start a second VM

Repeat now the previous steps to start a second VM. Remember to copy the vanilla `img` file you downloaded previously, rather than copying the one already attached to the first VM, which has been modified. Create also a different initialization disk for `cloud-init`, in particular to show the name `test-vm2` as a prompt message.

Note: although it is also possible to proceed cloning the first VM, this process is more troublesome, since by default only certain cloud-init steps are re-executed after the first boot, as well as some identifiers are not reset. Specifically, such approach might lead to inconsistencies with the network configuration (e.g., both VMs sharing the same IP address).

2.6.2 Measure the network performance between the two VMs

To measure the network performance between the two VMs, you can use the the following commands:

- On VM1, acting as server:

  ```
  iperf3 -s
  ```

- On VM2, acting as client:

  ```
  iperf3 -c <VM1_IP_address>
  ```

  where the IP address of VM1 can be detected by typing the command `ip addr` within VM1.

The `iperf3` tool will start a set of tests using TCP traffic and, at the end, it will return the average speed measured between the two machines.

2.7 Network Configuration

In this section we create a new virtual network, using the XML definition and manipulating it to create the network via the CLI. This is mainly done because manipulating the XML allows the possibility to have complete control of virtual hardware specification, by setting up also configurations which are not possible using the GUI (e.g. leverage openvswitch as bridging technology).

As mentioned before, a subset of network configurations can be also set via the virt-manager GUI. We can create/delete/list virtual network of each hypervisor by selecting a connection, right-clicking on it and then open "Connection details". In this view, the virtual networks tab is available among "Overview" and "Storage".

Note: Unless when explicitly mentioned, **all the following instructions are intended to be executed on the server**.

2.7.1 Inspect the "default" network

After installing Libvirt on a Linux system,a "default virtual network" is provided out of the box. You can verify its configuration by typing:

use sudo prefix

```
# virsh net-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
```

This network is used now by the VMs just created in previous steps. To inspect more about the network configuration, it is possible to export every resource described in libvirt as an XML file. The virsh CLI can help inspect the configuration by "dumping" the XML of the network by typing:

```
sudo virsh net-dumpxml default
```

```
<network connections='2'>
  <name>default</name>
  <uuid>fe7073de-9253-4525-9529-9f0f9db345b9</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr0' stp='on' delay='0'/>
  <mac address='52:54:00:82:cd:11'/>
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'/>
    </dhcp>
  </ip>
</network>
```

192.168.123.0

netmask a 24

The important parameters we can observe are the following:

- The **forward mode** element describes how the traffic has to be routed outside the VMs to the Internet. In particular
  - **"nat"** forwarding (aka "virtual networks"): like the "default" network, a virtual network switch operates in NAT mode (using IP masquerading). This means that any guests connected through it use the hypervisor IP address to communicate with outside destinations. This configuration does not allow incoming traffic to be received by this vNIC from the outside world.
  - **"routed"** networking (aka "shared physical device"): the virtual switch is connected to the physical host LAN, passing guest network traffic back and forth without using NAT. A "bridged" vNIC can configure an IP in the network where the NIC of the hypervisor is bridged, and be contacted from the outside world.
  - If the forward section is absent, the network is isolated from the outside world.
- **bridge name** element indicates which software bridge has to be used to connect virtual NICs of VMs belonging to this network.
- The **IP** element: each virtual network switch can be given a range of IP addresses, to be provided to guests through DHCP. Libvirt uses a program, dnsmasq, for this. An instance of dnsmasq is automatically configured and started by libvirt for each virtual network switch needing it.

When added to a network, vNICs are attached to the virtual bridge indicated in the network specification. To verify this, we can simply inspect the virbr0 bridge, to check on which vNICs are connected:

```
sudo brctl show
```

```
bridge name bridge id          STP enabled    interfaces
```

```
virbr0      8000.52540082cd11    yes             virbr0-nic
                                                 vnet0
                                                 vnet1
```

In addition we can find which MAC addresses corresponds to which vNIC and detect which VM is actually attached:

```
sudo brctl showmacs virbr0
```

From virt-manager, clicking on a specific VM on the hardware view, we may obtain the NIC MAC address.

2.7.2 Create a new virtual network

Starting from the default network, extract the network configuration by dumping it into a file:

```
sudo virsh net-dumpxml default > br1.xml
```

Now, in order to create a new network modify the following fields in br1.xml:

- Change the network name, two different networks cannot have the same name.
- Delete the network UUID. It will be regenerated by the Libvirt daemon at creation time.
- Change the bridge name. If the bridge does not exists, it will created by the Libvirt daemon.
- Delete the MAC address configuration. It will be regenerated at creation time.
- Remove the forwarding section, to create an internal network.
- Assign another subnet in the IP element, appropriately adapting also the *dhcp* configuration.

Check if your network file is correct by typing:

```
sudo virsh net-define br1.xml
```

Then, start the new network:

```
sudo virsh net-start <name-of-the-network>
```

If the network is created with success, we can create a new NIC in the two VMs created before (you will need to shut off the VMs to apply this modification). New NICs can be created by the hardware information window as showed in figure 2.14 and attach the new NIC to the newly created network. We can test connectivity using iperf in this network as we did with the default network.

Note: remember to check the status of the newly created network interface (e.g., with `ip addr`), turn it up and request an IP address if necessary:

```
sudo ip link set <interface_name> up
sudo dhclient -v <interface_name>
```

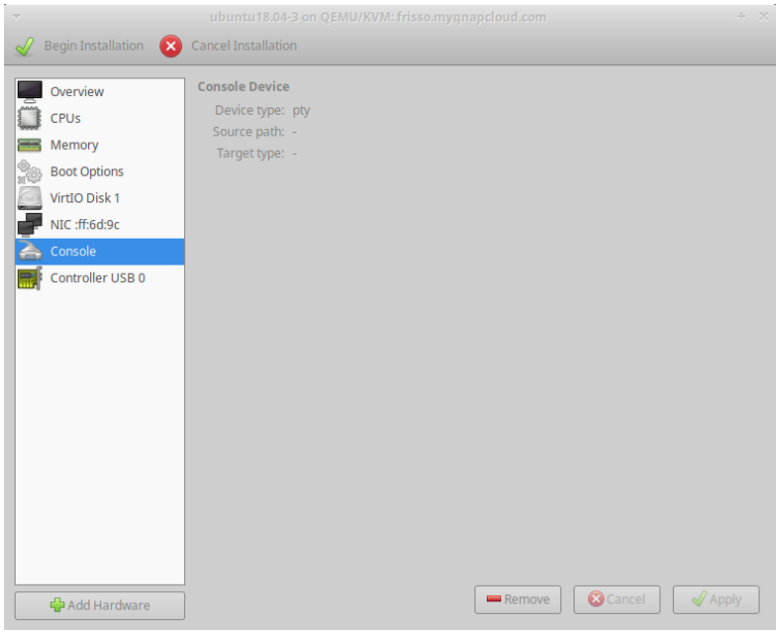


Figure 2.14: Virtual machine manager: add a new NIC from the hardware info section

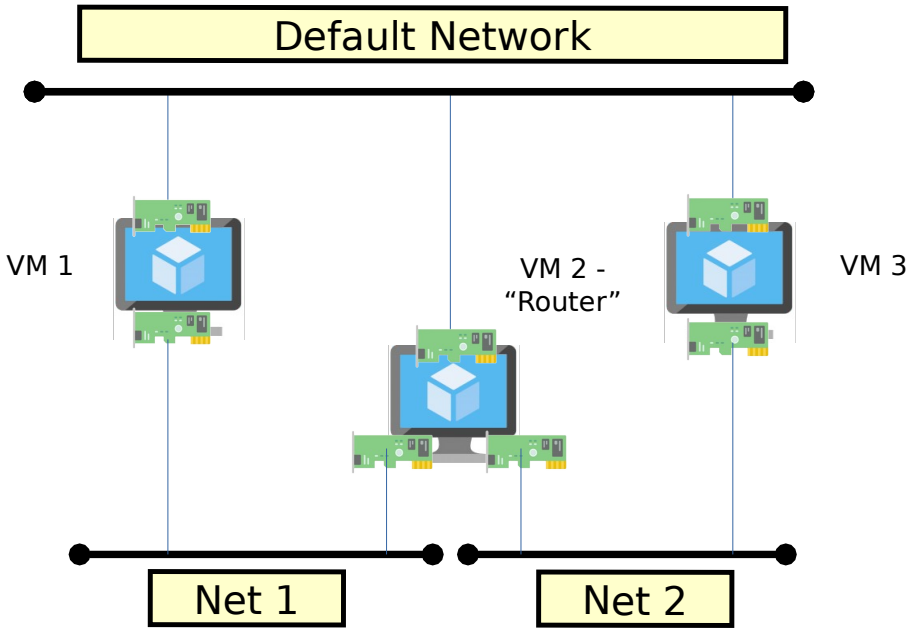


Figure 2.15: VM1 communicates with VM3 via VM2, used as a router. All VMs have a management NIC to support ssh connection from the outside world.

2.8 Use a router VM to exchange traffic between two VMs

In this section, it is proposed a more advanced exercise combining topics discussed in the previous sections. Therefore, instructions are less extensive, and it is up to the reader combine them together to satisfy the requirements.

As we discussed in the previous section, virtual networks can be combined to create multiple topologies and route traffic across VMs. In this exercise, we create the topology depicted in figure 2.15:

two VMs should communicate by using a dedicated secondary vNIC connected via a "router" VM.

- Create two internal networks without overlapping IPs.
  Suggestion (1): We suggest enabling DHCP in order to simplify IP assignation.
- Create two VMs with two NICs each with the **VirtIO driver**. The first NIC should be connected to the "default" virtual network or another virtual network accessible from the outside. The second one should be connected to one the two virtual networks respectively.
- Create the "Router" VM and attach three vNICs to it, one for the management network and the other two to be interconnected to the internal virtual networks.
- Configure NICs in the 3 VMs in order to enable routing. In particular, we should enable `ip_forwarding` on the "router" VM, and the appropriate routes on the other ones.
- Several tests that can be performed to check that the topology is implemented correctly:
  - Ping the IP of vNIC of VM3 to assess reachability from VM1.
  - Use traceroute to verify that your packets correctly traverse the router.
  - Test bandwidth using iperf as presented in Section 2.6.2 between VM1 and VM3.

2.8.1 Useful Commands

Here, you can find a list of commands useful to complete the exercise.

- Enable "IP forwarding" to let a Linux system act as a router:

```
sudo sysctl -w net.ipv4.ip_forward=1
```

- Create a dedicated route using `ip route`:

```
sudo ip route add <destination_network> via <gateway_ip>
```

- Request an IP using dhcp:

```
sudo dhclient -v <interface_name>
```