Preparing the environment

During this lab assignment, you might have to reinstall your server.

Task 1

Make sure that Ubuntu 22.04 64-bit is installed on your server. Use the PXE service to start the installation process if needed. Partition your disks such that you have at least 200 GB of unpartitioned space.

Task 2

Install the lvm2 package and create a physical volume using 100 GB of the 200 GB free space reserved before. On top of the physical volume create a volume group called VolumeGroupKVM. Here you will store the virtual machine images. We will create the logical volumes later. Hints: pvcreate, vgcreate, pvdisplay.

Install lvm2 package

```
sudo apt-get update
sudo apt-get install lvm2
```

Check disk partitions with paths

```
root@tallinn:/# lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
loop0    7:0    0  63.4M  1 loop /snap/core20/1974
loop1    7:1    0  53.3M  1 loop /snap/snapd/19457
loop2    7:2    0 111.9M  1 loop /snap/lxd/24322
loop3    7:3    0  73.9M  1 loop /snap/core22/1663
loop4    7:4    0   132M  1 loop /snap/docker/2932
sda      8:0    0 465.8G  0 disk
├─sda1   8:1    0     1G  0 part /boot/efi
└─sda2   8:2    0   200G  0 part /
sdb      8:16   0   1.8T  0 disk
sdc      8:32   0  19.3G  0 disk
```

Verify disk partitions with sectors

```
root@tallinn:/# fdisk -l
Disk /dev/loop0: 63.45 MiB, 66531328 bytes, 129944 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/loop1: 53.26 MiB, 55844864 bytes, 109072 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/loop2: 111.95 MiB, 117387264 bytes, 229272 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/loop3: 73.88 MiB, 77463552 bytes, 151296 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/loop4: 131.98 MiB, 138391552 bytes, 270296 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/sdc: 19.3 GiB, 20724056064 bytes, 40476672 sectors
Disk model: PERC H330 Adp
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
```

```
Disk identifier: 471224E5-8948-4AAF-BD1A-EA4129BCA723


Disk /dev/sda: 465.76 GiB, 500107862016 bytes, 976773168 sectors
Disk model: Samsung SSD 860
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 9AB60FA7-847E-4353-9E43-0A7290D0CB9F

Device       Start       End   Sectors  Size Type
/dev/sda1     2048   2203647   2201600    1G EFI System
/dev/sda2  2203648 421634047 419430400  200G Linux filesystem


Disk /dev/sdb: 1.8 TiB, 1979120091136 bytes, 3865468928 sectors
Disk model: PERC H330 Adp
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

Create new partition with 100G of space

```
root@tallinn:/# sudo fdisk /dev/sdb

Welcome to fdisk (util-linux 2.37.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0x8499dc1f.

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1):
First sector (2048-3865468927, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-3865468927,
        default 3865468927): +100G

Created a new partition 1 of type 'Linux' and of size 100 GiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

Create new Physical Volume and Volume Group

```
root@tallinn:/# sudo pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created.
root@tallinn:/# sudo vgcreate VolumeGroupKVM /dev/sdb1
  Volume group "VolumeGroupKVM" successfully created
```

Verify that both were created.

```
root@tallinn:/# sudo pvdisplay
  --- Physical volume ---
  PV Name               /dev/sdb1
  VG Name               VolumeGroupKVM
  PV Size               100.00 GiB / not usable 4.00 MiB
  Allocatable           yes
  PE Size               4.00 MiB
  Total PE              25599
  Free PE               25599
  Allocated PE          0
  PV UUID               HZA2Fw-2B1X-sg0F-U7O3-xb0X-Y22N-jOCIP6

root@tallinn:/# sudo vgdisplay
  --- Volume group ---
  VG Name               VolumeGroupKVM
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                0
```

```
  Open LV               0
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               <100.00 GiB
  PE Size               4.00 MiB
  Total PE              25599
  Alloc PE / Size       0 / 0
  Free  PE / Size       25599 / <100.00 GiB
  VG UUID               V57Eeu-hNUt-9TcZ-RYQA-vUDm-HBdy-XZC3Zp
```

Task 3

Verify if your server has virtualization and KVM support enabled on your system. Install the cpu-checker qemu-kvm, libvirt-daemon-system and libvirt-clients packages. Make sure the libvirt daemon is started on boot, start the daemon and ensure your current user is allowed to manage kvm and libvirt (add your user to the required groups).

Verify if we have required package cpu-checker

```
root@tallinn:/# sudo apt install cpu-checker
```

Check if the CPU supports virtualization (Intel vmx, AMD svm)

Outputs different than 0 means its supported.

```
root@tallinn:/# egrep -c '(vmx|svm)' /proc/cpuinfo
8
```

Check if KVM is enabled.

```
root@tallinn:/# sudo kvm-ok
INFO: /dev/kvm exists
KVM acceleration can be used
```

We got successful output.

Verify if we have required package qemu-kvm libvirt-daemon-system libvirt-clients

```
root@tallinn:/# sudo apt install qemu-kvm libvirt-daemon-system
        libvirt-clients
```

Start Libvirt daemon service and enable it on boot.

```
root@tallinn:/# sudo systemctl enable libvirtd
root@tallinn:/# sudo systemctl start libvirtd
root@tallinn:/# sudo systemctl status libvirtd
● libvirtd.service - Virtualization daemon
     Loaded: loaded (/lib/systemd/system/libvirtd.service; enabled;
        vendor preset: enabled)
     Active: active (running) since Mon 2024-10-28 13:42:14 UTC; 1h
        8min ago
TriggeredBy: ● libvirtd-ro.socket
             ● libvirtd.socket
             ● libvirtd-admin.socket
       Docs: man:libvirtd(8)
             https://libvirt.org
   Main PID: 5633 (libvirtd)
      Tasks: 21 (limit: 32768)
     Memory: 10.4M
        CPU: 288ms
     CGroup: /system.slice/libvirtd.service
             ├─5633 /usr/sbin/libvirtd
             ├─5757 /usr/sbin/dnsmasq --conf-
        file=/var/lib/libvirt/dnsmasq/default.conf --leasefile-ro --
        dhcp-script=/usr/lib/libvirt/libvirt_leaseshelper
             └─5758 /usr/sbin/dnsmasq --conf-
        file=/var/lib/libvirt/dnsmasq/default.conf --leasefile-ro --
        dhcp-script=/usr/lib/libvirt/libvirt_leaseshelper

Oct 28 13:42:14 tallinn systemd[1]: Started Virtualization daemon.
Oct 28 13:42:14 tallinn dnsmasq[5757]: started, version 2.90 cachesize
        150
Oct 28 13:42:14 tallinn dnsmasq[5757]: compile time options: IPv6 GNU-
        getopt DBus no-UBus i18n IDN2 DHCP DHCPv6 no-Lua TFTP
        conntrack ipset no-nftset auth cryptohash DNSSEC loop-detect
        inotify dumpfile
Oct 28 13:42:14 tallinn dnsmasq-dhcp[5757]: DHCP, IP range
        192.168.122.2 -- 192.168.122.254, lease time 1h
Oct 28 13:42:14 tallinn dnsmasq-dhcp[5757]: DHCP, sockets bound
        exclusively to interface virbr0
Oct 28 13:42:14 tallinn dnsmasq[5757]: reading /etc/resolv.conf
Oct 28 13:42:14 tallinn dnsmasq[5757]: using nameserver 127.0.0.53#53
```

```
Oct 28 13:42:14 tallinn dnsmasq[5757]: read /etc/hosts - 8 names
Oct 28 13:42:14 tallinn dnsmasq[5757]: read
        /var/lib/libvirt/dnsmasq/default.addnhosts - 0 names
Oct 28 13:42:14 tallinn dnsmasq-dhcp[5757]: read
        /var/lib/libvirt/dnsmasq/default.hostsfile
```

It started successfully.

Add current user to the required groups libvirt and kvm.

```
root@tallinn:/# sudo usermod -aG libvirt,kvm $USER
```

After that I closed the session and logged back in to update the groups of the session.

Check version of Libvirt and QEMU with virsh

```
dmarques@tallinn:~$ virsh version
Compiled against library: libvirt 8.0.0
Using library: libvirt 8.0.0
Using API: QEMU 8.0.0
Running hypervisor: QEMU 6.2.0
```

Verify that the virsh daemon is running

```
dmarques@tallinn:~$ virsh list --all
 Id   Name   State
--------------------
```

We have a successful output.

Difference Between KVM, Libvirt, and QEMU

KVM (Kernel-based Virtual Machine):

KVM is a virtualization module in the Linux kernel that allows the kernel to function as a hypervisor. It turns a Linux system into a bare-metal hypervisor, allowing multiple virtual machines (VMs) to run simultaneously with hardware-assisted virtualization (e.g., Intel VT or AMD-V). KVM itself is not a standalone application but rather a part of the Linux kernel.

QEMU (Quick Emulator):

QEMU is an open-source emulator that can run and emulate different operating systems and architectures. When paired with KVM, QEMU acts as the user-space component that provides the emulation, while KVM provides the acceleration and management capabilities through the kernel. QEMU can work without KVM for emulating different architectures, but it will be significantly slower.

Libvirt:

Libvirt is a toolkit and management layer for managing virtualization platforms, including KVM and QEMU. It provides a consistent API for managing VMs across different hypervisors (e.g., KVM, Xen, VMware). Libvirt allows for easier management of VM operations such as start, stop, pause, or migrate VMs and simplifies interactions with the underlying hypervisor (KVM) and emulator (QEMU). Libvirt abstracts much of the complexity of working directly with QEMU commands, providing a more user-friendly and consistent interface for managing VMs.

Networking

The goal of the next set of tasks is to create the networking environment for your virtual machines. First, we create an virtual Ethernet network to which the VMs will be connected. In Linux we can create such a network using a virtual Ethernet bridge.

Task 4

Since brctl is deprecated, we will use the iproute2 package to create a bridge. Using the ip command, manually create a bridge named kvmbr0. Do not add any interfaces to it; we will use routing instead of switching to connect the VMs to the Internet.

Create the Bridge: Use the ip command to create the bridge interface.

```
root@tallinn:~$ sudo ip link add name kvmbr0 type bridge
```

Bring the Bridge Interface Up: Now, activate the kvmbr0 interface.

```
root@tallinn:~$ sudo ip link set kvmbr0 up
```

Verify the Bridge Creation: Confirm that the bridge kvmbr0 was created and is active.

```
root@tallinn:~$ ip addr show kvmbr0
6: kvmbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
        state DOWN group default qlen 1000
    link/ether ee:d7:67:51:82:19 brd ff:ff:ff:ff:ff:ff
```

This setup creates an empty bridge named kvmbr0 without adding any interfaces. Using routing instead of switching allows virtual machines to communicate through IP forwarding, maintaining network isolation but enabling routing for external connectivity when needed.

Task 5

When creating the bridge, Linux will also create a network interface called kvmbr0 that connects your server to that bridge. The IPv4 addresses to use for your VMs are those in the /28 subnet which is routed to your server (see SNE students mailing list). Assign the first free IPv4 address from your /28 subnet to this kvmbr0 network interface using ip addr. The first free address is your subnet is the address at the start of the range plus 1. The starting address of the range is reserved to act as a network address (e.g. 145.100.104.0) in currrent Internet practice. The last address in the range is reserved to act as the broadcast address (see RFC3021). The address you assign will act as the IP gateway address for your virtual machines.

A /28 subnet provides 16 IP addresses: Network address: 145.100.106.192 (reserved) Usable range: 145.100.106.193 to 145.100.106.206 Broadcast address: 145.100.106.207 (reserved) My subnet is 145.100.106.192/28, the first usable ip is 145.100.106.193 that I'll use for kvmbr0 as the gateway for my virtual machines.

Use the ip addr command to set 145.100.106.193 as the IP address for the kvmbr0 bridge interface.

```
root@tallinn:/# sudo ip addr add 145.100.106.193/28 dev kvmbr0
```

Check if the IP address has been correctly assigned to kvmbr0

```
root@tallinn:/# ip addr show kvmbr0
6: kvmbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
        state DOWN group default qlen 1000
    link/ether ee:d7:67:51:82:19 brd ff:ff:ff:ff:ff:ff
    inet 145.100.106.193/28 scope global kvmbr0
       valid_lft forever preferred_lft forever
```

We can see the ip was set to 145.100.106.193 for the bridge kvmbr0

Task 6

Test whether you can reach the address on the bridge interface from outside your machine. You may have to enable IPv4 and IPv6 routing (Hint: sysctl.conf). Make sure that you test using ping -n from your workstation or any other machine connected to the Internet. Note that reverse DNS for your /28 subnet is also delegated to you. Make sure you don't have any firewall filters that prevent forwarding IP traffic.

Modify /etc/sysctl.conf to enable IPv4/IPv6 forwarding

```
root@tallinn:/# sudo nano /etc/sysctl.conf
```

Uncommented the following lines to enable IPv4 forwarding

net.ipv4.ip_forward = 1

Apply changes

```
root@tallinn:/# sudo sysctl -p
net.ipv4.ip_forward = 1
```

Modify firewall rules with ufw

```
root@tallinn:/# sudo ufw status
Status: inactive
```

Currently inactive.

Add new rules for the bridge interface (These are temporary)

```
root@tallinn:/# sudo iptables -A FORWARD -i kvmbr0 -j ACCEPT
root@tallinn:/# sudo iptables -A FORWARD -o kvmbr0 -j ACCEPT
```

Persist the iptables rules by saving them on rules.v4

```
root@tallinn:/# mkdir /etc/iptables
root@tallinn:/# sudo iptables-save > /etc/iptables/rules.v4
```

Verify the changes were saved.

```
root@tallinn:/# cat /etc/iptables/rules.v4
# Generated by iptables-save v1.8.7 on Thu Oct 31 12:23:24 2024
*mangle
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:LIBVIRT_PRT - [0:0]
-A POSTROUTING -j LIBVIRT_PRT
-A LIBVIRT_PRT -o virbr0 -p udp -m udp --dport 68 -j CHECKSUM --
        checksum-fill
COMMIT
# Completed on Thu Oct 31 12:23:24 2024
# Generated by iptables-save v1.8.7 on Thu Oct 31 12:23:24 2024
*filter
:INPUT ACCEPT [0:0]
:FORWARD DROP [408832:19903872]
:OUTPUT ACCEPT [0:0]
:DOCKER - [0:0]
:DOCKER-ISOLATION-STAGE-1 - [0:0]
:DOCKER-ISOLATION-STAGE-2 - [0:0]
:DOCKER-USER - [0:0]
:LIBVIRT_FWI - [0:0]
:LIBVIRT_FWO - [0:0]
:LIBVIRT_FWX - [0:0]
:LIBVIRT_INP - [0:0]
:LIBVIRT_OUT - [0:0]
-A INPUT -j LIBVIRT_INP
-A FORWARD -j LIBVIRT_FWX
-A FORWARD -j LIBVIRT_FWI
-A FORWARD -j LIBVIRT_FWO
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j
        ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A FORWARD -i kvmbr0 -j ACCEPT
-A FORWARD -o kvmbr0 -j ACCEPT
-A FORWARD -i kvmbr0 -j ACCEPT
-A FORWARD -o kvmbr0 -j ACCEPT
-A FORWARD -i kvmbr0 -j ACCEPT
-A FORWARD -o kvmbr0 -j ACCEPT
-A FORWARD -i kvmbr0 -j ACCEPT
-A FORWARD -o kvmbr0 -j ACCEPT
-A OUTPUT -j LIBVIRT_OUT
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-
        ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN
-A LIBVIRT_FWI -d 192.168.122.0/24 -o virbr0 -m conntrack --ctstate
        RELATED,ESTABLISHED -j ACCEPT
-A LIBVIRT_FWI -o virbr0 -j REJECT --reject-with icmp-port-unreachable
-A LIBVIRT_FWO -s 192.168.122.0/24 -i virbr0 -j ACCEPT
-A LIBVIRT_FWO -i virbr0 -j REJECT --reject-with icmp-port-unreachable
-A LIBVIRT_FWX -i virbr0 -o virbr0 -j ACCEPT
-A LIBVIRT_INP -i virbr0 -p udp -m udp --dport 53 -j ACCEPT
-A LIBVIRT_INP -i virbr0 -p tcp -m tcp --dport 53 -j ACCEPT
-A LIBVIRT_INP -i virbr0 -p udp -m udp --dport 67 -j ACCEPT
-A LIBVIRT_INP -i virbr0 -p tcp -m tcp --dport 67 -j ACCEPT
-A LIBVIRT_OUT -o virbr0 -p udp -m udp --dport 53 -j ACCEPT
-A LIBVIRT_OUT -o virbr0 -p tcp -m tcp --dport 53 -j ACCEPT
-A LIBVIRT_OUT -o virbr0 -p udp -m udp --dport 68 -j ACCEPT
-A LIBVIRT_OUT -o virbr0 -p tcp -m tcp --dport 68 -j ACCEPT
COMMIT
# Completed on Thu Oct 31 12:23:24 2024
# Generated by iptables-save v1.8.7 on Thu Oct 31 12:23:24 2024
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
```

```
:DOCKER - [0:0]
:LIBVIRT_PRT - [0:0]
-A PREROUTING -m addrtype --dst-type LOCAL -j DOCKER
-A OUTPUT ! -d 127.0.0.0/8 -m addrtype --dst-type LOCAL -j DOCKER
-A POSTROUTING -j LIBVIRT_PRT
-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE
-A DOCKER -i docker0 -j RETURN
-A LIBVIRT_PRT -s 192.168.122.0/24 -d 224.0.0.0/24 -j RETURN
-A LIBVIRT_PRT -s 192.168.122.0/24 -d 255.255.255.255/32 -j RETURN
-A LIBVIRT_PRT -s 192.168.122.0/24 ! -d 192.168.122.0/24 -p tcp -j
       MASQUERADE --to-ports 1024-65535
-A LIBVIRT_PRT -s 192.168.122.0/24 ! -d 192.168.122.0/24 -p udp -j
       MASQUERADE --to-ports 1024-65535
-A LIBVIRT_PRT -s 192.168.122.0/24 ! -d 192.168.122.0/24 -j MASQUERADE
COMMIT
# Completed on Thu Oct 31 12:23:24 2024
```

The rules were added but are not persistent.

Test IPv4 Connectivity from a different machine by pinging the ip 145.100.106.193

```
dmarques@Mac ~ % ping -n 145.100.106.193 -c 4
PING 145.100.106.193 (145.100.106.193): 56 data bytes
64 bytes from 145.100.106.193: icmp_seq=0 ttl=62 time=3.282 ms
64 bytes from 145.100.106.193: icmp_seq=1 ttl=62 time=2.755 ms
64 bytes from 145.100.106.193: icmp_seq=2 ttl=62 time=4.346 ms
64 bytes from 145.100.106.193: icmp_seq=3 ttl=62 time=4.289 ms

--- 145.100.106.193 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 2.755/3.668/4.346/0.676 ms
```

This confirms the bridge is reachable over IPv4, indicating that IP forwarding and firewall configurations are working correctly.

Task 7

Edit /etc/netplan/00-installer-config.yaml such that kvmbr0 will persist across system reboots.

To ensure that kvmbr0 persists across reboots using Netplan, we need to edit the /etc/netplan/00-installer-config.yaml file and define the bridge configuration.

Add kvmbr0 configuration to netplan yaml file.

```
root@tallinn:/# sudo nano /etc/netplan/00-installer-config.yaml
root@tallinn:/# cat /etc/netplan/00-installer-config.yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    eno1:
      critical: true
      dhcp-identifier: mac
      dhcp4: true
      nameservers:
        addresses:
        - 145.100.96.11
        - 145.100.96.22
        search:
        - studlab.os3.nl
    eno2:
      dhcp4: true
  bridges:
    kvmbr0:
      interfaces: []  # No physical interfaces added for routing-only
        mode
      dhcp4: false
      addresses:
        - 145.100.106.193/28  # IP for kvmbr0 bridge
      parameters:
        stp: false
        forward-delay: 0
  version: 2
```

interfaces: []: No physical interfaces are attached, allowing kvmbr0 to operate in routing-only mode. addresses: Assigns the first available IP in the /28 subnet to the bridge.

This configuration ensures that kvmbr0 persists across reboots with the configured IP address and bridge settings.

Apply changes on the configuration

```
root@tallinn:/# sudo netplan apply
```

Verify the configuration changes

```
root@tallinn:/# ip addr show kvmbr0
6: kvmbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
        state DOWN group default qlen 1000
    link/ether ee:d7:67:51:82:19 brd ff:ff:ff:ff:ff:ff
    inet 145.100.106.193/28 brd 145.100.106.207 scope global kvmbr0
       valid_lft forever preferred_lft forever
```

KVM Virtual Machines

Task 8

Define the earlier created VolumeGroupKVM as a storage pool so newly created VMs can use it. Verify the pool exists and make sure the pool is started, also after a reboot. Hint: You might find useful to inspect the man page of virsh, you need to define, build and start the storage pool before it can be used. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_and_managing_virtualization/managing-storage-for-virtual-machines_configuring-and-managing-virtualization#creating-lvm-based-storage-pools-using-the-cli_assembly_managing-virtual-machine-storage-pools-using-the-cli

Verify pool-capabilities

```
root@tallinn:/# virsh pool-capabilities | grep "'logical'
        supported='yes'"
  <pool type='logical' supported='yes'>
```

Create the Volume Group.

```
root@tallinn:/# sudo nano /etc/libvirt/storage/VolumeGroupKVM.xml
<pool type='logical'>
  <name>VolumeGroupKVM</name>
  <uuid>84792a18-21ac-4792-a73b-99eb39630c3e</uuid>
  <capacity unit='G'>100</capacity>
  <allocation unit='G'>0</allocation>
  <available unit='G'>100</available>
  <source>
    <device path='/dev/sdb2'/>
    <name>VolumeGroupKVM</name>
    <format type='lvm2'/>
  </source>
  <target>
    <path>/dev/VolumeGroupKVM</path>
  </target>
</pool>
root@tallinn:/# sudo virsh pool-define
        /etc/libvirt/storage/VolumeGroupKVM.xml
Pool VolumeGroupKVM defined from
        /etc/libvirt/storage/VolumeGroupKVM.xml
root@tallinn:/# sudo virsh pool-list --all
 Name             State      Autostart
-------------------------------------------
 VolumeGroupKVM   inactive   no
root@tallinn:/# sudo virsh pool-build  VolumeGroupKVM
root@tallinn:/# virsh pool-autostart VolumeGroupKVM
root@tallinn:/# sudo virsh pool-start VolumeGroupKVM
root@tallinn:/# sudo virsh pool-list --all
 Name             State    Autostart
-------------------------------------------
 VolumeGroupKVM   active   yes
```

The new VolumeGroupKVM was added.

Task 9

Create a new volume in the pool VolumeGroupKVM called guest01volume, download the latest ubuntu Yammy cloud-init image (https://cloud-images.ubuntu.com/jammy/current/), convert into to a raw image and upload the image to the created volume. Hint: wget the image with -kvm in the name. The qemu-img convert command can help you with converting the downloaded qcow2 image. e.g. qemu-img convert ¡source¿ -O raw ¡deationation¿.raw virsh provides a command that allows you to upload the raw disk image directly onto the LVM parition, or you could use the dd command line tool.

Download and prepare the image.

```
root@tallinn:/home/dmarques# wget https://cloud-
        images.ubuntu.com/jammy/current/jammy-server-cloudimg-amd64-
        disk-kvm.img^C
root@tallinn:/home/dmarques# ls
jammy-server-cloudimg-amd64-disk-kvm.img
root@tallinn:/home/dmarques# qemu-img convert jammy-server-cloudimg-
        amd64-disk-kvm.img -O raw jammy-server-cloudimg-amd64-disk-
        kvm.raw
root@tallinn:/home/dmarques# ls -lh jammy-server-cloudimg-amd64-disk-
        kvm.raw
-rw-r--r-- 1 root root 2.2G Oct 31 14:11 jammy-server-cloudimg-amd64-
        disk-kvm.raw
```

Create new Volume guest01volume in the VolumeGroupKVM pool

```
root@tallinn:/home/dmarques# lvcreate VolumeGroupKVM -n guest01volume
        -L 20G
  Logical volume "guest01volume" created.
root@tallinn:/home/dmarques# lvdisplay
  --- Logical volume ---
  LV Path                /dev/VolumeGroupKVM/guest01volume
  LV Name                guest01volume
  VG Name                VolumeGroupKVM
  LV UUID                3VZ8w4-CI1B-7WWd-smqm-qzST-lmWT-jYi4UA
  LV Write Access        read/write
  LV Creation host, time tallinn, 2024-10-31 15:36:31 +0000
  LV Status              available
  # open                 0
  LV Size                20.00 GiB
  Current LE             5120
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0
```

Task 10

Use virt-install to create a Ubuntu virtual machine that has the following characteristics:

• Hostname: Guest-01 • RAM: 2048MB • Disk size: 20GB • VolumeGroup: VolumeGroupKVM • VolumeName: guest01volume • Distribution: Ubuntu Jammy • Virtual CPUs: 2 • IP: 145.100.106.192/28

Install packager

```
root@tallinn:/home/dmarques# apt install virtinst cloud-image-utils -y
```

Copy the image to the volume

```
root@tallinn:/home/dmarques# sudo dd if=jammy-server-cloudimg-amd64-
        disk-kvm.raw of=/dev/VolumeGroupKVM/guest01volume bs=4M
        status=progress
2042626048 bytes (2.0 GB, 1.9 GiB) copied, 3 s, 681 MB/s
563+0 records in
563+0 records out
2361393152 bytes (2.4 GB, 2.2 GiB) copied, 14.3552 s, 164 MB/s
```

Create network profile

```
root@tallinn:/home/dmarques/configs# cat network-guest01.yaml
version: 2
ethernets:
  eth0:
    dhcp4: false
    addresses:
      - 145.100.106.192/28
```

Create userdata profile

Hash password

```
root@tallinn:/home/dmarques/configs# openssl passwd -6
Password:
Verifying - Password:
$6$d4cl.5VJD6DsUA4/$9CkAnmboKv8FF1OeoTAFBdE0wNt4pWaniZvwrBFOx4MEZaIzj8zNkb8LPRBbsIAGsmwTRmzRF4mXgPjSo
```

```
root@tallinn:/home/dmarques/configs# cat userdata-guest01.yaml
#cloud-config
hostname: Guest-01
users:
  - name: marques
    password:
        $6$d4cl.5VJD6DsUA4/$9CkAnmboKv8FF1OeoTAFBdE0wNt4pWaniZvwrBFOx4MEZaIzj8zNkb8LPRBbsIAGsmwTRmzRF
    sudo: ['ALL=(ALL) NOPASSWD:ALL']
    shell: /bin/bash
    lock_passwd: false
    ssh_authorized_keys:
      - ssh-rsa
        AAAAB3NzaC1yc2EAAAADAQABAAABgQDbD9LeF5ZHMi5pJAtH6XRC9gA3WuPo0flvhZqqDVBqH4R229NXgFXJqRR29g9eX
        root@tallinn
    groups:
      - sudo
chpasswd:
  expire: false
package_update: true
package_upgrade: true
```

Create virtual machine

sudo virt-install –name Guest-01 –memory 2048 –vcpus 2 –disk vol=VolumeGroupKVM/guest01volume,size=20,format=raw –cloud-init user-data=/home/dmarques/configs/userdata-guest01.yaml,network-config=/home/dmarques/configs/network-guest01.yaml,disable=off –network bridge=kvmbr0,model=virtio –os-variant ubuntujammy –graphics none –import

```
root@tallinn:/home/dmarques# sudo virt-install \
  --name Guest-01 \
  --memory 2048 \
  --vcpus 2 \
  --disk vol=VolumeGroupKVM/guest01volume \
  --cloud-init user-data=/home/dmarques/configs/userdata-
        guest01.yaml,network-config=/home/dmarques/configs/network-
        guest01.yaml,disable=off \
  --network bridge=kvmbr0,model=virtio \
  --os-variant ubuntujammy \
  --graphics none \
  --import

Starting install...
Creating domain...
        |    0 B  00:00:00
Running text console command: virsh --connect qemu:///system console
        Guest-01
Connected to domain 'Guest-01'
Escape character is ^] (Ctrl + ])
```

I came across several configuration problems, The model=virtio needs to be present otherwise our network configuration is not loaded. I tried to create an iso image with “cloud-localds cloud-init-guest01.iso userdata.yaml network.yaml” and used it on the installation command instead of the –cloud-init parameters. After successfully creating the image, the configurations were not set on the VM. During the attemps, I found out that when undefining the VM its mandatory to remove the storage, recreate the volume and copy the original jammy image otherwise the settings from the old attempts were present. NOTE: I’ve attempted to create my own cloud init iso to load on the cd-rom but the settings were not being applied on the vm installation.

Task 11

The MAC address starts with 52:54:00. Explain why this prefix is used.

The MAC address prefix 52:54:00 is commonly used in virtualized environments, particularly with QEMU and KVM, to designate network interfaces as virtual rather than physical. This prefix falls within the range of “locally administered addresses” (LAA), which are not globally unique and can be assigned by administrators for internal use. By utilizing this prefix, virtual machines can have unique MAC addresses without conflicting with physical hardware addresses assigned by manufacturers. This practice helps maintain network consistency and minimizes issues related to MAC address duplication in virtualized settings.

Task 12

Start the virtual machine and login to its console and test network connectivity Hint virsh console . Exit by hitting CTRL +]

Start the VM

```
root@tallinn:/home/dmarques# sudo virsh start Guest-01
Domain 'Guest-01' started
```

Access the VM

```
root@tallinn:/home/dmarques# sudo virsh console Guest-01
Guest-01 login: marques
Password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 5.15.0-1067-kvm x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Sun Nov  3 20:47:55 UTC 2024

  System load: 0.0               Memory usage: 7%   Processes:
        85
  Usage of /:  7.4% of 19.20GB   Swap usage:   0%   Users logged in: 0


Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
marques@Guest-01:~$
marques@Guest-01:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
         group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
         state UP group default qlen 1000
    link/ether 52:54:00:7b:ba:df brd ff:ff:ff:ff:ff:ff
    inet 145.100.106.194/28 brd 145.100.106.207 scope global enp1s0
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe7b:badf/64 scope link
       valid_lft forever preferred_lft forever
marques@Guest-01:~$ ping google.nl
PING google.nl (142.250.179.131) 56(84) bytes of data.
64 bytes from ams17s10-in-f3.1e100.net (142.250.179.131): icmp_seq=1
         ttl=118 time=0.751 ms
64 bytes from ams17s10-in-f3.1e100.net (142.250.179.131): icmp_seq=2
         ttl=118 time=0.569 ms
64 bytes from ams17s10-in-f3.1e100.net (142.250.179.131): icmp_seq=3
         ttl=118 time=0.636 ms
64 bytes from ams17s10-in-f3.1e100.net (142.250.179.131): icmp_seq=4
         ttl=118 time=0.628 ms
64 bytes from ams17s10-in-f3.1e100.net (142.250.179.131): icmp_seq=5
         ttl=118 time=0.659 ms
64 bytes from ams17s10-in-f3.1e100.net (142.250.179.131): icmp_seq=6
         ttl=118 time=0.650 ms
^C
--- google.nl ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5007ms
rtt min/avg/max/mdev = 0.569/0.648/0.751/0.054 ms
marques@Guest-01:~$
```

Task 13

Use virsh list to find information about the running VM and then stop it and start it again.

Additional information command output on the task 14 “virsh dominfo Guest-01”

List VM's running

```
root@tallinn:/home/dmarques# sudo virsh list --all
 Id   Name       State
---------------------------
 11    Guest-01   running
```

Stop VM Guest-01

```
root@tallinn:/home/dmarques# sudo virsh shutdown Guest-01
Domain 'Guest-01' is being shutdown
```

Start VM Guest-01

```
root@tallinn:/home/dmarques# sudo virsh start Guest-01
Domain 'Guest-01' started
```

Task 14

Configure your system such that Guest-01 is auto-started after a reboot.

```
root@tallinn:/home/dmarques/image# sudo virsh autostart Guest-01
Domain 'Guest-01' marked as autostarted

root@tallinn:/home/dmarques/image# virsh dominfo Guest-01
Id:             25
Name:           Guest-01
UUID:           e9fa00ca-0e9f-4bad-889f-72b22a62be8e
OS Type:        hvm
State:          running
CPU(s):         2
CPU time:       7.0s
Max memory:     2097152 KiB
Used memory:    2097152 KiB
Persistent:     yes
Autostart:      enable
Managed save:   no
Security model: apparmor
Security DOI:   0
Security label: libvirt-e9fa00ca-0e9f-4bad-889f-72b22a62be8e
        (enforcing)
```

Task 15

autoinstall, preseed, cloud-init and kickstart can be used to aid in bootstrapping of virtual machine images. In fact, with virt-install you can utilize kickstart or cloud-init directly. When would you use any of the aformentioned methods?

Cloud-Init, Kickstart, Preseed, and Autoinstall each support automated VM and server setup, tailored to specific OS environments and deployment needs.

Cloud-Init is ideal for post-deployment configuration in cloud and virtualized environments across multiple OSes like Ubuntu, Debian, and CentOS. It enables first-boot customization, such as setting up users and networks, and integrates easily with cloud providers and virt-install.

Kickstart automates full OS installations on Red Hat-based systems (e.g., RHEL, CentOS), making it a strong choice for consistent, large-scale deployments. It handles everything from partitioning to post-install scripts and works well with virt-install.

Preseed provides similar full-install automation for Debian-based systems, like Debian and Ubuntu. It's highly customizable and integrates smoothly with virt-install, making it suitable for both VM and bare-metal deployments.

Autoinstall, replacing Preseed on newer Ubuntu versions (20.04+), is YAML-based and pairs with cloud-init for modern, flexible setups. It's designed for large-scale Ubuntu deployments in cloud and VM environments and is readable and easy to maintain.

Each tool suits different stages of provisioning—from initial OS installation to post-deployment configuration—allowing administrators to select the best fit for their infrastructure and operating system.

Method	When to Use	Suitable OS	Direct virt-install Integration
Cloud-Init	For post-boot configuration in cloud/virtualized environments	Ubuntu, Debian, CentOS, others	Yes
Kickstart	For complete automated installation of RHEL-based OSes	RHEL, CentOS, Fedora	Yes
Preseed	For full automation of Debian-based OS installation	Debian, Ubuntu	Yes

Method	When to Use	Suitable OS	Direct virt-install Integration
Autoinstall	For modern, YAML-based installations on Ubuntu	Ubuntu 20.04+	Limited (cloud-init integration)

Task 16

How do you think that the virtual machine communicates with the outside network in your setup? Draw a simple network diagram showing at least the network cards, the bridges and any routers that might be present. Don't forget to label everything with IP addresses and names.

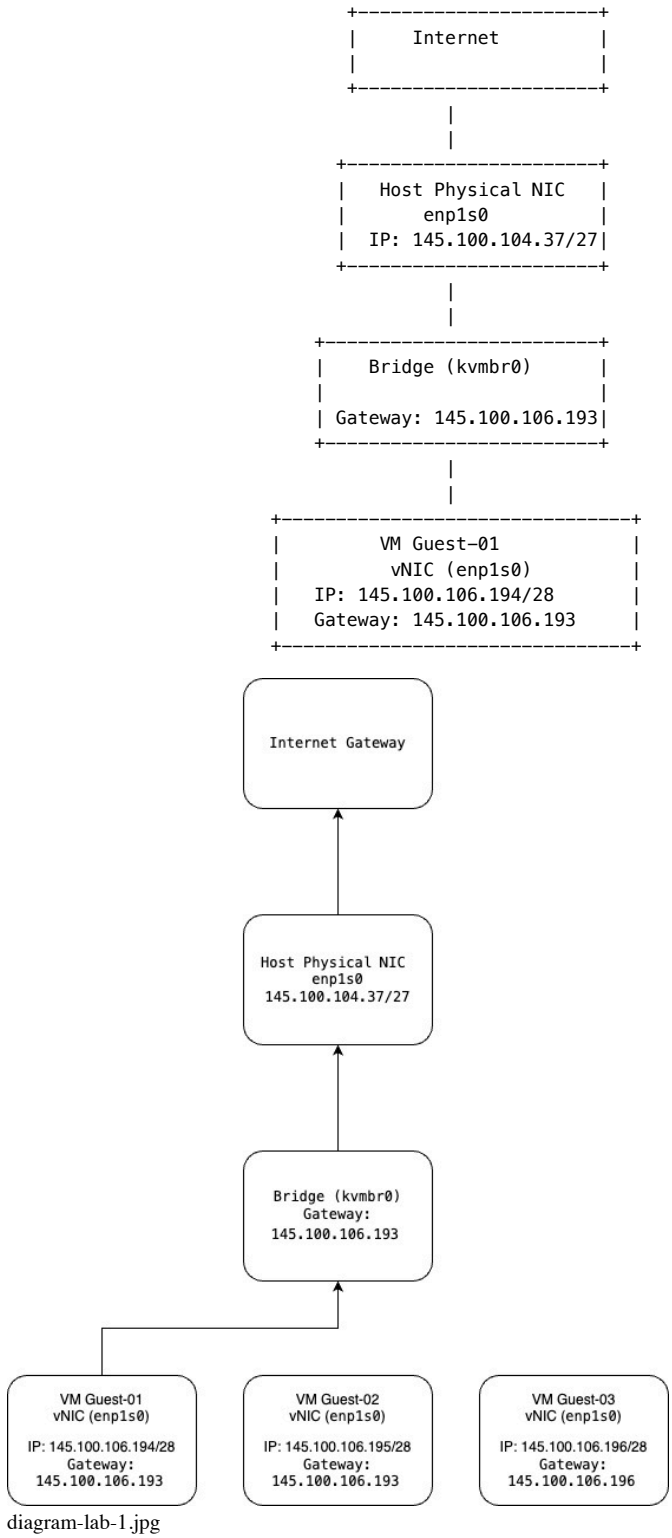

diagram-lab-1.jpg

In this setup, the host and VM are on separate subnets, requiring routing for the VM to access external networks via the host. The host's physical network interface (NIC) is assigned IP 145.100.104.37/27 and connects to the main network, providing internet access. Meanwhile, the bridge interface kvmbr0 on the host is configured with IP 145.100.106.193 on the VM's subnet, allowing it to act as a gateway for the VM.

The VM, Guest-01, has an IP of 145.100.106.194/28 on its virtual NIC (enp1s0) and uses kvmbr0 (145.100.106.193) as its gateway. When the VM sends outbound packets, they route through kvmbr0, which forwards them to the host's NIC, enp1s0, and onward to the internet.

For inbound communication, incoming traffic arrives at the host's NIC (enp1s0), and the host's routing rules direct the packets to kvmbr0, which forwards them to the VM at 145.100.106.194. This setup allows the VM on the 145.100.106.194/28 subnet to communicate externally via the host's NIC on the 145.100.104.37/27 subnet, with kvmbr0 acting as the essential bridge and gateway between the VM and the broader network.

If more VMs are added, they can share the same bridge to reach the internet or communicate within each other. I've added Guest-02 and Guest-03 to show where they would stand on the structure without connecting them to any interface.