Kernel alapú virtualizáció (KVM)

kliens- és szerveroldali megoldások

Írta: Cziva Richárd – UYD1S5
Tárgy neve: Virtualizációs technológiák és alkalmazásaik BMEVIMIAV89
Oktatók: Micskei Zoltán, Tóth Dániel

1. Bevezetés

1.1 A virtualizációs technológiák

A virtualizáció az utóbbi években egy elterjedt és sokrétű fogalommá vált. Egyik legfontosabb tulajdonsága az, hogy el tudjuk különíteni a fizikai hardvert a felhasznált erőforrásoktól, és fel tudjuk osztani azt. A főbb virtualizációs típusok a teljesség igénye nélkül:

- Platformvirtualizáció
  - Az egyik legáltalánosabb virtualizációs fajta amikor teljes gépet virtualizálunk egy már futó gépben. Ide sorolható a KVM technológia is.
  - Sokféle típusa lehet, akár igényelhet hardveres támogatást is, a célja a hardver erőforrásainak megosztása.
- Erőforrás-virtualizáció
  - Lemezkötetek, hálózati erőforrások összerendelése és menedzselése.
- Megjelenítés-virtualizáció
  - Vékonykliens technológiák, amik biztosítják, hogy egy központi szerveren fusson minden felhasználó virtuális gépe, és azokhoz távolról kapcsolódhassanak. Pl.: Novell Verde.
- Operációsrendszer-szintű virtualizáció
  - Ide sorolhatóak azok a szoftveres megoldások, amik a hoszt gépen belül egy operációs rendszer felett kis független környezetet (container, jail) hoznak létre.
- Alkalmazásszintű virtualizáció
  - Konkrét alkalmazást szeretnénk futtatni egy másik környezetben. A VMware vApp megoldást nyújt többet között erre is. Szoftveres szimulációs megoldások: Wine, DOSEMU, XP mode.
- Grid computing, fürtözés
  - Több számítógép összefogása nagyobb metaszámítógépekké.

A fent említett néhány technológia csupán töredéke annak, amit a virtualizáció adhat nekünk. A technológia viszonylag újnak mondható, mégis beépült rengeteg nagyvállalati struktúra elemei közé, pl. költséghatékonysága, hordozhatósága, könnyű menedzselhetősége miatt.

1.2 Platform típusú virtualizáció

A platformvirtualizációt két típusra oszthatjuk:

- Hosted típusú megoldások: Ilyenkor a hoszt OS-en belül fut a virtualizációs szoftver, és a felett futnak a különböző VM-ek. Ez a megoldás jellemzően kliensoldali. Ide sorolható be a VMware Workstation, MS VirtualPC, VirtualBox, és ide tartozik a KVM is, habár azt széles körben alkalmazzák szerveroldali virtualizációban is.

- Bare-metal megoldások: Nincsen klasszikus értelemben vett hoszt OS, hanem közvetlenül a hardver felett található a virtualizációs szoftver, amit legtöbbször speciálisan hypervisornak hívunk. Ilyenkor a hypervisor irányítja az összes felette elhelyezkedő guest OS-t. Ez a megoldás jellemzően szerveroldali (pl. ESX).

A KVM elhelyezkedése azért speciális ebben a felosztásban, mert bármilyen Linux alapú operációs rendszeren üzembe helyezhető, tehát hosted típusú, viszont sokan használják az így kiépített rendszert hypervisorként, tehát ez alapján akár a másik kategóriába is besorolhatnánk.

2. Telepítés

2.1 Mi az a KVM?

A KVM tehát kernel alapú virtualizációt jelent, és a Kernel-based Virtual Machine névből kapta a KVM rövidítést. Ez egy olyan alapvetően hosted típusú natív virtualizációs technológia, mely képes kihasználni a processzorokban rejlő hardveres virtualizációt megvalósító technológiát, működéséhez kernelmodulokra és egy QEMU emulátorra van szükség.

2.2 Rendszerigény

Az működéséhez csupán egy Linux operációs rendszerre van szükségünk, illetve amennyiben azt szeretnénk, hogy használható legyen a rendszer sebesség szempontjából, úgy ajánlatos egy hardveres virtualizációt támogató processzorral virtualizálni (Intel VT-X, AMD-V). A teszteléshez egy openSUSE 11.2 Linux operációs rendszert használok, a processzor a gépben pedig egy Intel P8400-as processzor, továbbá fizikailag 2 GB memória áll rendelkezésemre. Fontos, hogy alapból nem minden esetben van engedélyezve a BIOS-ban a hardveres virtualizáció. Ezt mindenféleképpen tegyük meg.

Azt, hogy a processzorunk támogat-e hardveres virtualizációt, a következőképpen tudjuk leellenőrizni:

```
lenovo:~ # egrep '^flags.*(vmx)' /proc/cpuinfo
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
pbe nx lm constant_tsc arch_perfmon pebs bts pni dtes64 monitor
ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm sse4_1 lahf_lm
tpr_shadow vnmi flexpriority
```

Mivel 2 magos processzor van a gépben, ezért kétszer fog megjelenni a flags, ugyanolyan tartalommal. Az AMD processzorok estén itt az svm flaget kell keresni.

2.3 Kernel

Mint a nevében is benne van, ez a technológia kernel alapú, vagyis kernelmodulokat kell betölteni a működéséhez, ahhoz, hogy valójában ki tudjuk használni a hardveres virtualizációt. Amennyiben nincs engedélyezve a BIOS-ban a VT-X, vagy nincsenek betöltve a szükséges modulok a kernelben, akkor a QEMU csak szimulált módban tud majd működni, ami lényegesen lassabb működést eredményez.

A működéshez 2 kernel modulra van szükségünk:

1. kvm.ko
2. kvm-intel.ko (AMD esetén ez kvm-amd.ko)

Ezt a két modul a hivatalos alapkernelben is megtalálható, a 2.6.20-as verziótól kezdve. Az éppen aktuális kernel, amit használunk:

```
lenovo:~ # uname -r
2.6.31.14-0.4-desktop
```

Mivel ez a kernel 2.6.20-as verzió fölött van, így nincs más dolgunk, mint ellenőrizni, hogy be van-e töltve a kvm és a kvm-intel modul, és amennyiben nincsen, akkor betöltsük azt. A betöltés, majd ellenőrzés a következőképpen történhet:

```
lenovo:~ # modprobe kvm kvm-intel
lenovo:~ # lsmod | grep kvm
kvm_intel              52392  0
kvm                   201052  1 kvm_intel
```

Mivel a kvm modult használja a kvm-intel modul, így amennyiben el szeretnénk távolítani a modulokat, úgy először a kvm-intel modult, majd a kvm modult kell eltávolítani.

Amennyiben a kernelünkben nincsenek benne a szükséges kernelmodulok, úgy azok forrását letölthetjük a linux-kvm.org oldalról, majd a következő módszerrel fordíthatjuk azt:

```
lenovo:~ # tar xjf kvm-kmod-release.tar.bz2
lenovo:~ # cd kvm-kmod-release
lenovo:~ # ./configure
lenovo:~ # make
lenovo:~ # make install
lenovo:~ # modprobe kvm-intel
```

2.4 KVM installálás

A KVM csomagot ma már az összes alapvető repositoryban megtalálhatjuk, így installálása nagyon leegyszerűsödött. A mi esetünkben a zypper csomagkezelőt használva csupán ennyi:

```
lenovo:~ # zypper install kvm
Loading repository data...
Reading installed packages...
Resolving package dependencies...

The following NEW packages are going to be installed:
  kvm python-curses

2 new packages to install.
```

A telepítés után a kvm csomag a következőket biztosítja:

- `kvm_stat`: A kernelmodul képes visszaadni információkat arról, hogy éppen milyen műveleteket végzünk. Amennyiben nem fut virtuális gép, úgy a kernelmodult sem használjuk, tehát csupa nullát fogunk látni a statisztikákban. Később azonban érdekes eredményeket mutathat.
- `qemu-img-kvm`: Alapvetően ez a QEMU egy része, ezzel tudunk majd KVM-es virtuális imageket létrehozni.
- `qemu-kvm`: Ez a legfontosabb komponens, ez indítja el a virtuális gépet, ez végzi magát a virtualizálást a kernelmodulok segítségével.

2.5 QEMU

A QEMU egy nyílt forráskódú, platformfüggetlen processzoremuláló és dinamikus fordítóprogram. Képes arra, hogy a hoszt architektúrájától és operációs rendszerétől függetlenül egy másik tetszőleges architektúrát vagy operációs rendszert emuláljon. Architektúrákból képes megvalósítani például x86-ot vagy x86_64-et, operációs rendszerből pedig bármely PC-re írt operációs rendszer futtatására képes.

A KVM csomag telepítése során mi is kapunk egy pár funkciót a QEMU programjából, méghozzá két KVM-re szabott komponenst. Ez annyit jelent, hogy például a qemu-kvm képes együttműködni a KVM kernelmoduljaival, ezzel megvalósítva a teljes virtualizációt. Lényegében tehát a KVM-es virtualizáció nem más, mint a QEMU kiegészítése hardveres virtualizációval.

2.6 KVM tesztelése

Ebben a részben a tesztelni fogjuk a rendszerünket egy hivatalos KVM-tesztelő szkript segítségével. A teszteléshez szükségünk van egy verziókezelő rendszerre, ugyanis a KVM fejlesztői az éppen aktuális tesztelő benchmarkot egy git repositoryban tették közzé. Git telepítése után letölthető a benchmark a következő paranccsal:

```
lenovo:~ # git clone git://github.com/ehabkost/autotest.git
```

A teszt több lépésen keresztül ellenőrzi, hogy megfelel-e a rendszerünk. Különböző ISO fájlokat lehet letölteni pluszban hozzá, hogy kiterjesszük a szimpla ellenőrzési metódust. A winutis.iso letöltésével például Windowsos gépeket tesztelhetünk. A tesztet a get_started.py szkript futtatásával kezdjük.

```
lenovo:~/virt/git/autotest/client/tests/kvm # ./get_started.py

16:31:22 INFO | KVM test config helper
16:31:29 INFO | 4 - Verifying winutils.iso (make sure we have the
utility ISO needed for Windows testing)
16:31:29 INFO | In order to run the KVM autotests in Windows
guests, we provide you an ISO that this script can download
16:31:29 INFO | Verifying iso winutils.iso
16:31:31 DEBUG| /tmp/kvm_autotest_root/isos/windows/winutils.iso
present, with proper checksum
16:31:31 INFO | 5 - Checking if qemu is installed (certify qemu
and qemu-kvm are in the place the default config expects)
16:31:31 DEBUG| /usr/bin/qemu-kvm present
16:31:31 DEBUG| /usr/bin/qemu-img present
16:31:31 INFO | 6 - Checking for the KVM module (make sure kvm is
loaded to accelerate qemu-kvm)
16:31:31 DEBUG| Running '/sbin/lsmod'
16:31:31 DEBUG| KVM module loaded
16:31:31 INFO | 7 - Verify needed packages to get started
16:31:31 INFO | When you are done fixing eventual warnings found,
you can run the kvm test using the command line AS ROOT:
16:31:31 INFO | /root/virt/git/autotest/client/bin/autotest –
verbose /root/virt/git/autotest/client/tests/kvm/control
16:31:31 INFO | You can also edit the test config files (see
output of step 2 for a list)
```

Ez a szkript előkészíti és leellenőrzi a rendszerünket a tényleges teszt futtatásához. A teszt futtatása a következő paranccsal zajlik. (A --verbose módot bekapcsolva még több infót kapunk.) A teszt több ezer sort ír ki a konzolunkba, ezért ezt most nem másolnám be, csak az első leglényegesebb sorát.

```
lenovo:~/virt/git/autotest/client/tests/kvm #
/root/virt/git/autotest/client/bin/autotest --verbose
/root/virt/git/autotest/client/tests/kvm/control

16:38:31 INFO | Writing results to
/root/virt/git/autotest/client/results/default
```

Amint a teszt véget ért, az eredményeket meg tudjuk tekinteni a fent említett könyvtárban. Ha parse-olva szeretnénk az eredményeket látni, úgy használhatjuk a scan_result.py programot, vagy megtekinthetjük a böngészővel a result.html fájl. Ha minden rendben van, akkor készen állunk a KVM használatára. Két módszert nézünk meg: parancssoros használat és Virt-manager használata.

3. Parancssoros használat

3.1 Image-fájl létrehozása

Először is azt kell eldöntenünk, hogy hol szeretnénk tárolni a virtuális image-eket. Én a /mnt/share mappát választottam magamnak, ide fogom pakolni rendre az egyes képfájlokat. A konzolos bemutatás alatt egy Windows telepítését fogjuk megnézni, ezért a /mnt/share alatt létrehoztam egy windows mappát, amibe az image fájl kerülhet.

Az image fájl létrehozásánál meg kell adni, hogy milyen formátumban szeretnénk az image-et tárolni. Ezek a következők lehetnek:

- raw: Teljesen nyers formátum, akkor válasszuk ezt, ha későbbiekben szeretnénk más formátumba exportálni a képfájlt. Ha a fájlrendszer kezeli a lyukakat (olyan lemezterület, ahol nincs érdemi adat), akkor csak az adatot tartalmazó részek fognak helyet foglalni.
- qcow2: A QEMU legáltalánosabb formátuma. Használata kifejezetten előnyös, ha a fájlrendszerünk nem támogatja a lyukakat (pl. Windows). Opcionálisan tartalmaz AES titkosítást, zlib alapú tömörítést és snapshotkészítést.
- qcow: A régi QEMU formátum. Kompatibilitási gondok miatt nem használjuk már.
- cow: Copy On Write képformátum. Régi verziók használták, Windows alatt nem is működik.
- vmdk: VMware képfájlformátuma
- cloop: Linux Compressed Loop Image

A másik fontos paraméter az image mérete, amit magunknak kell meghatározni. Most 14 GB-os image-et fogunk létrehozni qcow2 típusú képfájlformátummal. Ez így néz ki:

```
lenovo:/mnt/share/windows # qemu-img create windows.img -f qcow2
14G

Formatting 'windows.img', fmt=qcow2 size=15032385536
encryption=off cluster_size=0
```

A létrejövő fájl mindössze párszáz KB-os, vagyis nem foglalja le a 14 GB-ot, hanem csak tényleg annyit, amennyit az effektív adat foglal a fájlban.

3.2 Rendszer telepítése

A képfájl létrehozása után telepíthetjük a rendszerünket. A telepítéshez a qemu-kvm programot fogjuk használni. A telepítés előtt ismerkedjünk meg a qemu-kvm legfontosabb kapcsolóival:

- no-acpi: ACPI kikapcsolása. Az APCI egy energiagazdálkodási rendszer, mely átveszi az irányítást a BIOS-tól, és szoftveres szinten szabályozza a rendszerünket. Egyik lényeges funkciója a button, ami azt szabályozza, hogy a számítógép kikapcsológombjának megnyomásakor mi történjen (pl. szabályos leállítás vagy semmi). Ha a telepítésnél bekapcsoljuk, akkor annál a rendszernél mindig használva lesz. Windows telepítésénél érdemes kikapcsolni, ugyanis nagyon lassú lesz tőle a rendszer.
- m: Ezzel adhatjuk meg, hogy mennyi virtuális memóriát használjon a virtuális gépünk
- boot: Miről bootoljon a virtuális rendszer. Pl. az image-fájlról, CD-ről, hálózatról.
- snapshot: Pillanatképek készítésének engedélyezése.
- usb: elérhetővé teszi az USB eszközöket a guest OS számára.
- usbdevice tablet: Kiléphetünk anélkül a virtuális gép ablakából, hogy használnunk kellene a Ctrl+Alt billentyűkombinációt.
- std-vga: Wide-screen használata.
- hda: A rendszert tartalmazó képfájl helye.

Most egy Windows XP-t fogunk feltelepíteni CD-ROM-ról. A telepítés a következő paranccsal zajlik:

```
lenovo:/mnt/share/windows # qemu-kvm -no-acpi -m 1024 -cdrom
/dev/cdrom -boot d windows.img
```

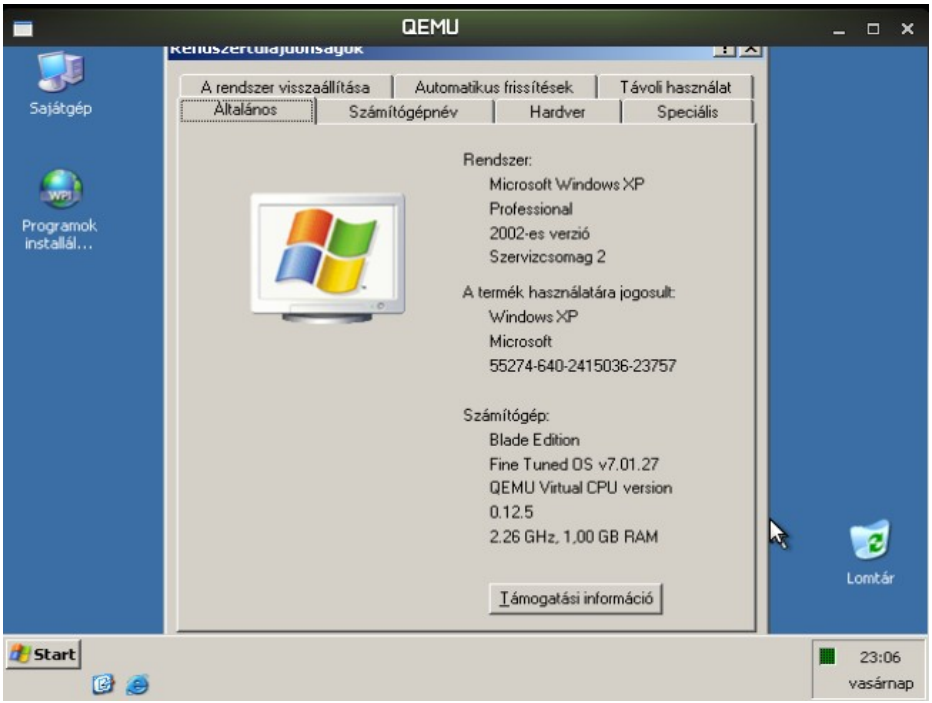
A parancs megnyitja a QEMU grafikus felületét. Ezek után a Windows telepítése pontosan olyan, mintha egy teljesen szokványos számítógépen telepítenénk. Jól látszik továbbá, hogy a Windows már csak a számára kijelölt 14 GB-os partíciót látja, és arra tud dolgozni:

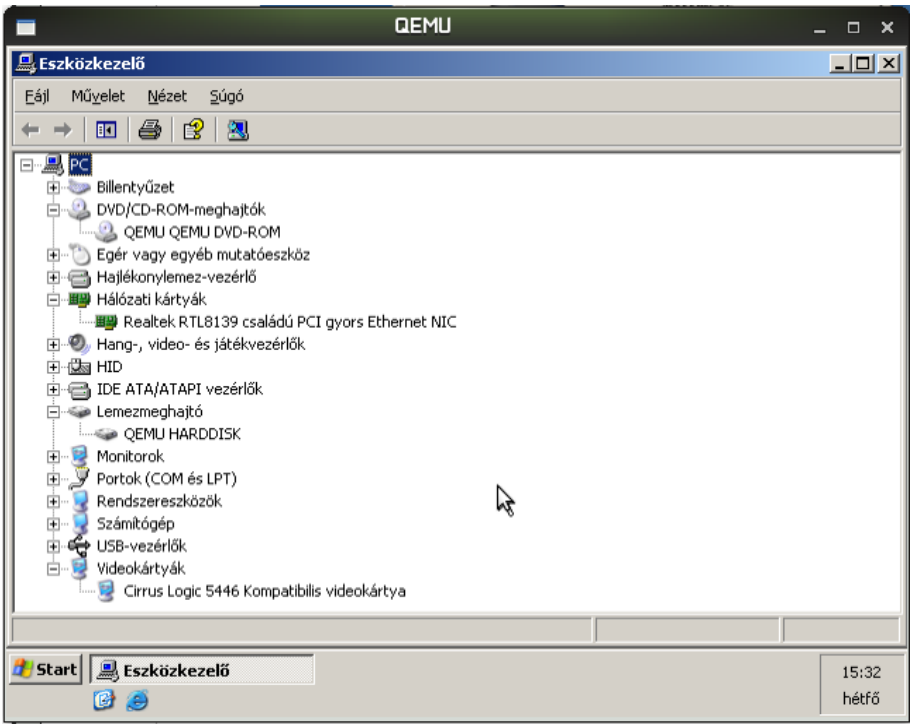
3.3 Rendszer elindítása, használata

A telepítés végeztével indítsuk el a rendszert a következő paranccsal:

```
lenovo:/mnt/share/windows # qemu-kvm -hda windows.img -m 1024
-cdrom /dev/cdrom -boot c -usb -usbdevice tablet -no-acpi
```

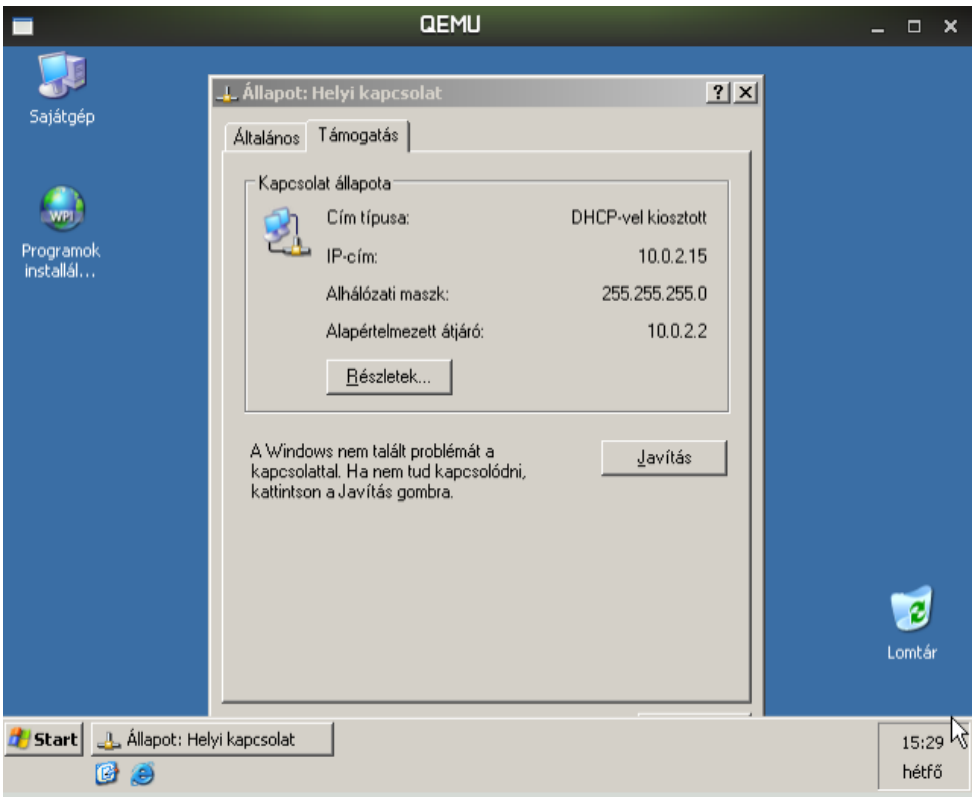
Indítás után tekintsük meg Sajátgép-Tulajdonságok, majd az Eszközkezelőben, hogy a virtuális gépünk milyen erőforrásokat lát. A processzornál látszik, hogy valójában egy QEMU által emulált processzorról van szó. A memória 1 GB, azaz pontosan annyi, mint amennyit megadtunk neki.



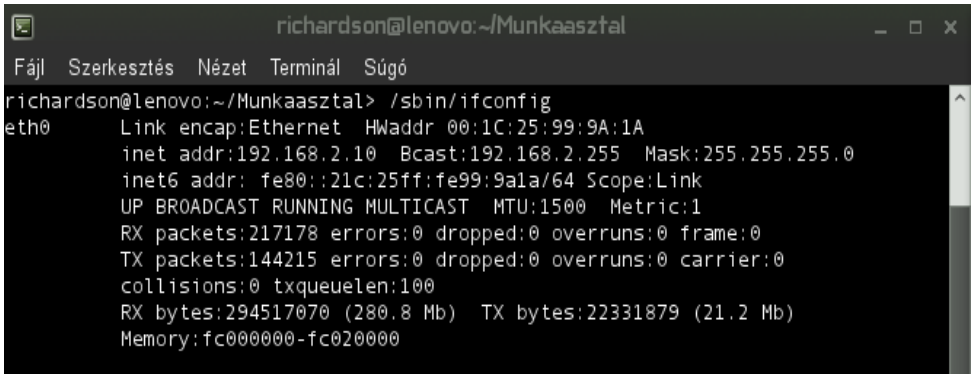


Az eszközkezelőben is pontosan azokat a hardvereket látjuk, amiket a QEMU biztosít számunka. Ilyen pl. a QEMU DVD-ROM, a QEMU-HARDDISK, és a Cirrus Logic videokártya.

Amennyiben telepítésnél nem választottunk mást (-net kapcsolóval), akkor alapértelmezetten a QEMU NAT-olni fogja a virtuális gép saját belső hálózatát. Ez azt jelenti, hogy a guest OS belső IP-címe teljesen független a valós hálózatunktól, kifelé tehát a hoszt OS és a vendég OS 1 IP-cím alatt fog látszani. Be lehet állítani, hogy a vendég rendszer külön IP-címet kérjen a belső hálózatunkon. A vendég OS saját IP tartománya a mi esetünkben: 10.0.2.0/24



A host OS IP-címe ezzel ellentétben teljesen más tartományból származik:



A NAT-olás következtében egy weboldal lekérése a következőképp jelenik meg hoszt OS-en:

Látható, hogy itt már mindenki a hoszt OS IP-címével (192.168.2.10) látszik.

3. Virtuális gépek menedzselése Virt-managerrel

3.1 Virt-manager bemutatása

A Virt-manager (teljes nevén: Virtual Machine Manager) egy olyan Red Hat által fejlesztett szoftver, mellyel grafikus felületen kezelhetjük a virtuális gépeinket, grafikonokat, statisztikákat tekinthetünk meg. A szoftver használatával pár kattintással tudjuk létrehozni, leállítani, elindítani, klónozni a virtuális gépeinket. Továbbá a Virt-manager tartalmaz egy beépített VNC klienst, amivel könnyedén hozzáférhetünk a guest OS konzoljához. A KVM-en kívül támogatja még a XEN technológiát is.

3.2 Telepítése, beszerzése

A Virt-manager egy ingyenes program, tehát letölthető a http://virt-manager.et.redhat.com/ címről, vagy megtalálható a program a hozzá tartozó libbel együtt a népszerűbb repositorykban. OpenSUSE rendszeren a telepítése a megszokott módon zajlik:

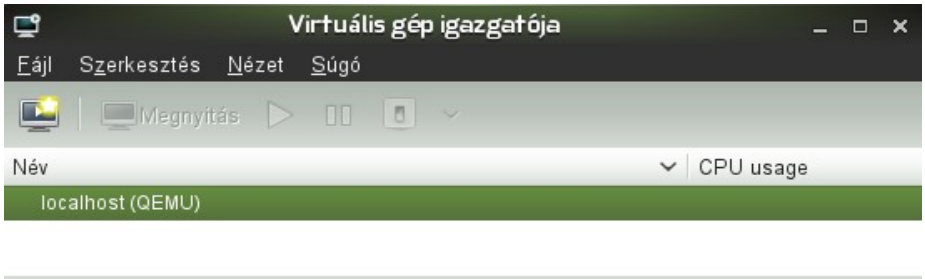
```
lenovo:~ # zypper install virt-manager
```

Telepítés után el kell indítanunk a libvirt daemonját (libvirtd), mert anélkül nem fog elindulni a Virt-manager sem. Ajánlatos ezt a daemont automatikus indításra állítanunk, amit a következőképp tehetünk meg:

```
lenovo:~ # /etc/init.d/libvirtd start
Starting libvirtd                                    done
lenovo:~ # chkconfig -a libvirtd
libvirtd                  0:off  1:off  2:off  3:on  4:off  5:on  6:off
```

3.3 Használata

Amennyiben megvagyunk a telepítéssel, úgy elindíthatjuk a Virt-managert egy terminálablakból. A Virt-manager indítása során leellenőrzi, hogy az adott gépen milyen virtualizációs szoftverek vannak feltelepítve, és ezek alapján fog a továbbiakban működni. Indítás után a következő kép fogad minket:

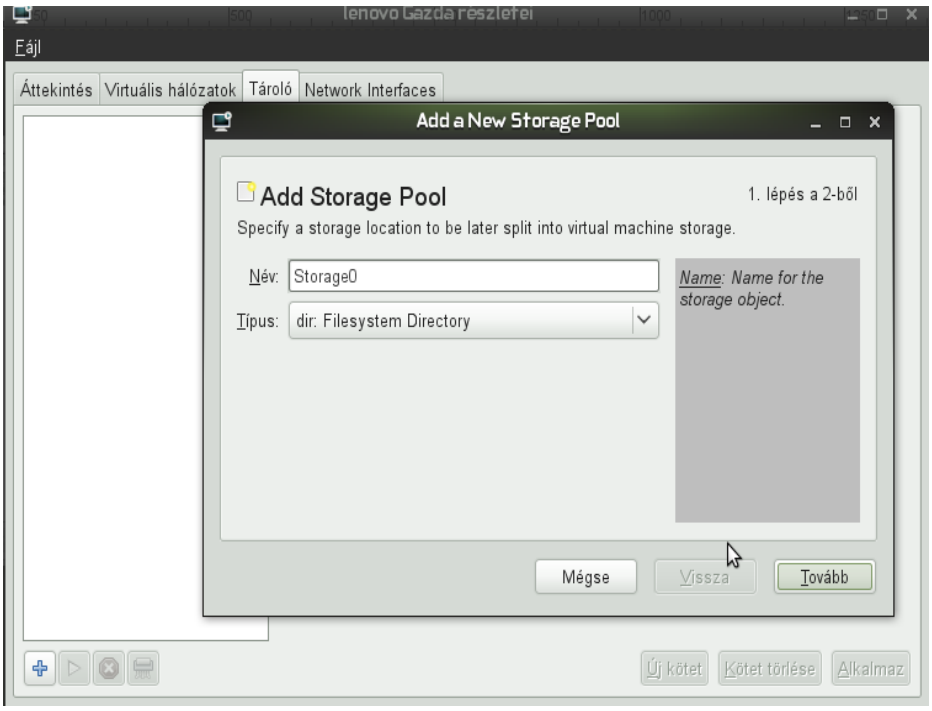


Első lépésként ajánlatos beállítani a storage-ünket, ahol az image-eket tárolni fogjuk. Ez lehet:

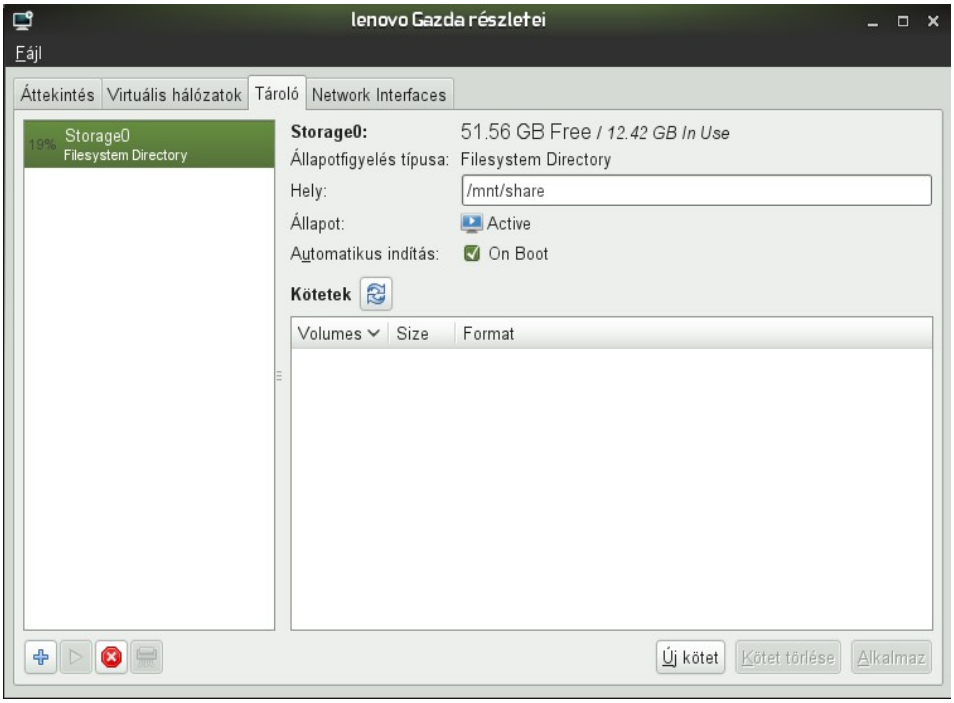
- egy mappa vagy fizikai diszk a gépen,
- NFS-en felcsatolt könyvtár,
- iSCSI-target,
- LVM partíció.

Mi most egy mappát fogunk kijelölni az image-ek tárolására (/mnt/share/).

Új Storage Pool létrehozásához Szerkesztés-Beállítások-Tároló menüpontot kell meghívni, majd a bal alsó sarokban lévő + jelre kell kattintani. Ekkor a következő ablak jön fel:

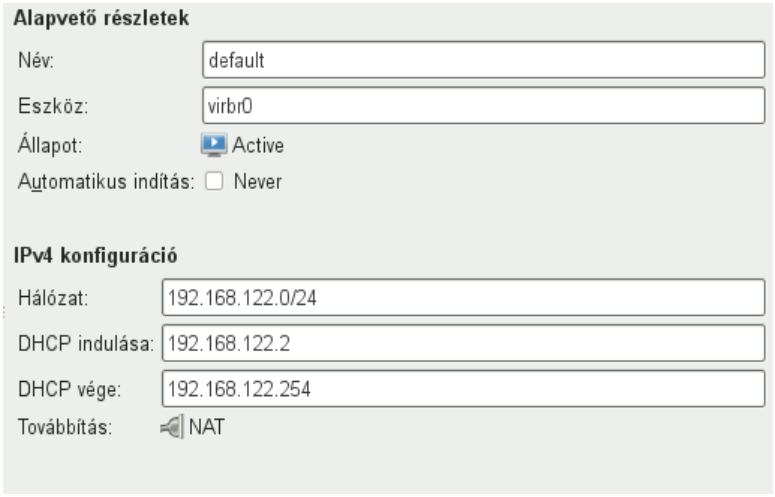


A Storage Pool nevének és típusának meghatározása után egy következő ablakban meg kell még adnunk, hogy a típusspecifikus beállításokat. Pl. iSCSI-target címe vagy a helyi könyvtár, amit használni fogunk. A sikeres konfigurálás után a Tárolók fül alatt láthatjuk a létrehozott Storage0 nevezetű poolt:



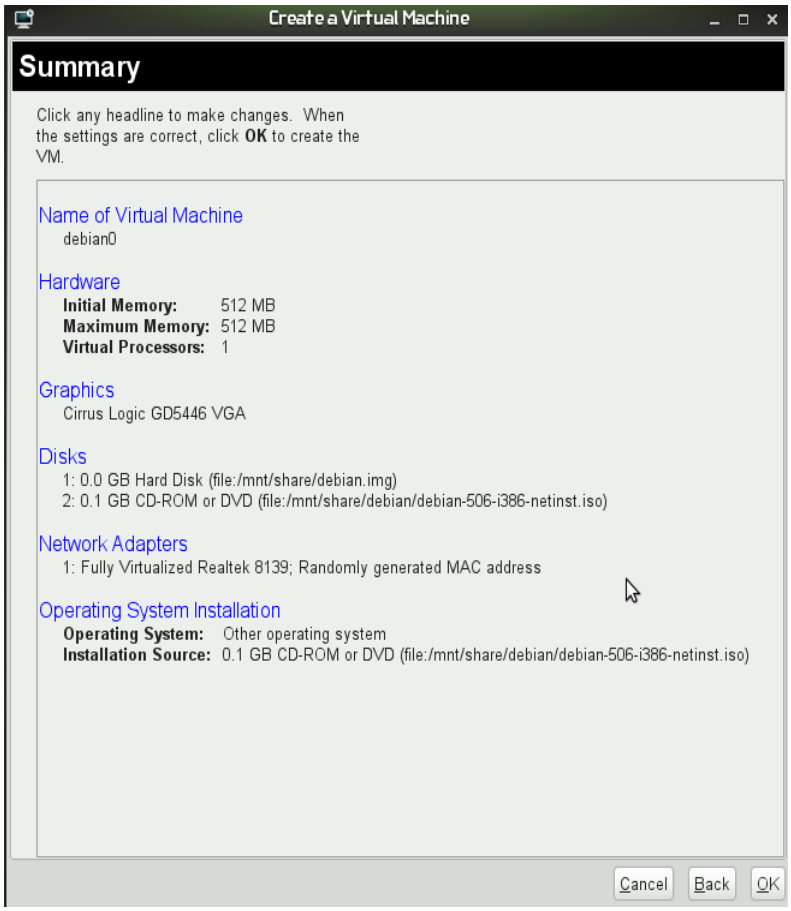
Az új kötet gombbal létrehozhatunk itt is üres képfájlokat, amibe rendszert telepíthetünk, de ezt a következőkben tárgyalt Virtuális gép létrehozása menüpont is elvégzi. Itt a Tárolók résznél tudunk több tárolót is kezelni, inaktívvá tenni vagy törölni.

Még egy fontos globális beállítást meg kell ejtenünk: ez a hálózat beállítása. Alapértelmezett módban egy NAT-olást megvalósító hálózat van felvéve a Virtuális hálózatok fül alatt. Amennyiben ezt szeretnénk használni, akkor indítsuk el, hogy a státusza aktív legyen. Az alábbi képen láthatjuk ennek a virtuális hálózatnak a beállításait:



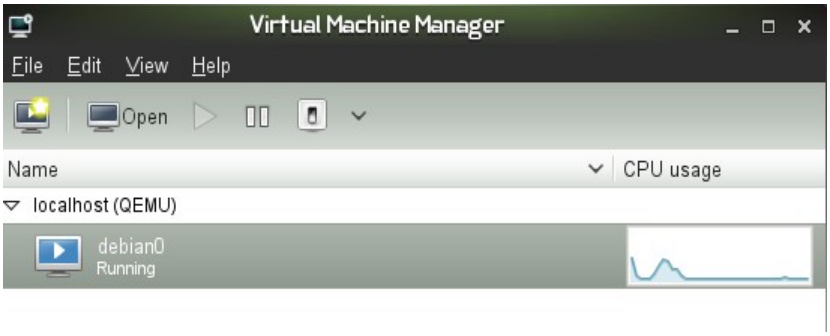
3.3 Virtuális gép létrehozása

Ha az alapvető beállításokkal végeztünk, hozzunk létre egy virtuális gépet. Ezt a Virt-manager főmenüjében tudunk a hosztunk nevére jobb egérgombbal kattintva, majd „New”-t nyomva. Ezek után egy varázsló vezet minket végig a virtuális gép létrehozásán. A varázslón végigmenve a következő dolgokat konfiguráltam be:

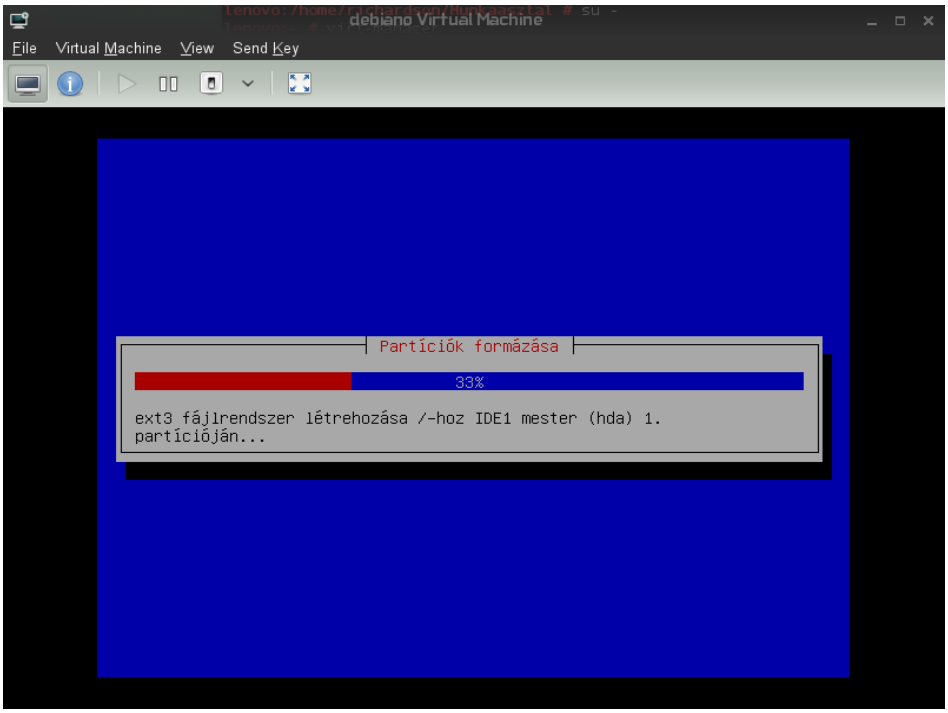


Fontos, hogy legalább 256 MB memóriát adjunk a gépnek, illetve, hogy a Virtuális CPU-k (VCPU) száma ne haladja meg a gépben lévő fizikai processzormagok számát. Vagyis egy 2 magos processzoros gép esetén ne legyen 3 vagy annál több VCPU-t használó virtuális gépünk, ugyanis az erősen a teljesítmény rovására fog menni. Amennyiben hálózatot is szeretnénk, ne felejtsük el nem csak itt, de a központi beállítások között sem beállítani.

A procedúra után Virt-manager főmenüjében megjelenik az új virtuális gépünk. A gépre jobb gombbal kattintva megtekinthetjük a műveleteket, amiket végre tudunk hajtani rajta. Pl.: leállítás, indítás, klónozás, migrálás vagy a konzol megtekintése. Hasznos funkció az egyes gépek CPU terhelésének kijelzése, ugyanis így könnyen észrevehető, ha valamelyik gépünk valami okból kifolyólag irreálisan nagy terhelést szenved el.



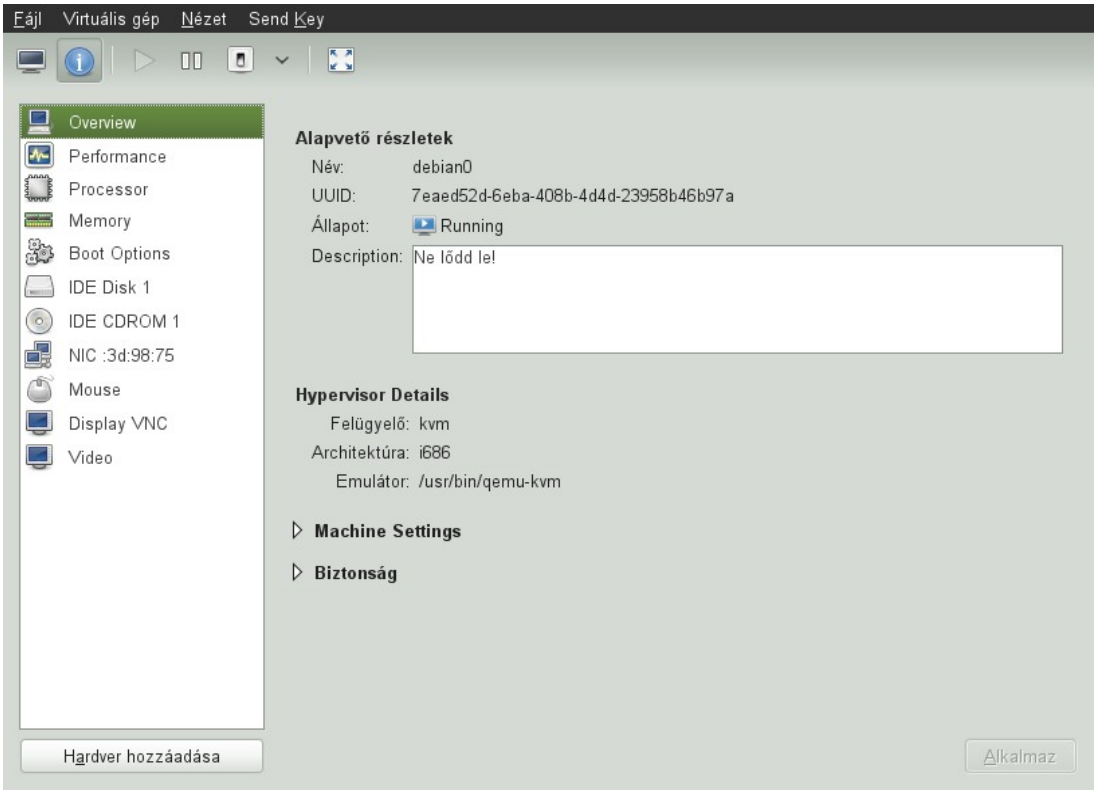
A rendszer konzolján éppen folyó telepítést pedig nyomon tudjuk követni a beépített VNC kliens használatával:



A telepítés után a Debian is egy hibátlan rendszer lett, működő hálózattal (a már beállított NAT-olással), USB/DVD használattal, ahogy azt elvárjuk. Amennyiben hangot is szeretnénk a virtuális gépben, akkor aktiválni kell a hangkártyát mint virtuális hardvert a virtuális gép számára. Erről a következő részben lesz szó.

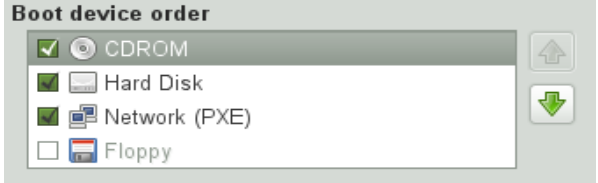
3.3 Virtuális gép módosítása

Egy nagyon kényelmesen használható funkciója a Virt-managernek a virtuális gépek hardverkonfigurációjának szerkesztése. Ennek szerkesztését a következőképp érjük el: a virtuális gépet nyissuk meg, és a konzoljának megtekintése helyett az Információ szimbólumba kattintsunk. Ekkor a következő kép tárul elénk:



Itt lényegében minden paramétert testreszabhatunk, rendre végigmenve az egyes menüpontok tartalmán:

- Overview: Erről látjuk a screenshotot is, itt a globális információk vannak, mint pl. a gép neve, UUID-je, állapota és leírása. A Hypervisor Details alatt pedig a legfontosabbak a Felügyelő és az architektúra információs mezők.
- Performance: 4 nem túl szép grafikon tárul elénk a processzor- és memória használatról, valamint a lemez és hálózat I/O terheltségéről. A CPU-használat grafikonja megegyezik azzal a grafikonnal, amit a kezdőlapon is látunk a virtuális gépünk mellett.
- Processor: Itt menet közben is állíthatjuk (ez nem jelenti azt, hogy menet közben is érvénybe lép a módosítás, ahhoz újra kell indítania virtuális gépet) a VCPU allokálások számát.
- Memory: A virtuális gép számára maximálisan lefoglalható memóriaterület mérete változatható itt meg.
- Boot options: Ez a menüpont tulajdonképpen a -boot kapcsolót váltja fel, és értelemszerűen a boot sorrend állítható be rajta, egy hasonló felületen:



- IDE Disk 1: Ez a jelenlegi HDD, amit a virtuális gépünk lát és használ. Beállításainál bekapcsolhatunk egy read-only módot, mellyel csak olvasásra kapja meg a rendszer az aktuális lemezt. Ezt rendszert tartalmazó HDD

esetén csak akkor tegyük meg, ha éppen nem arról a HDD-ről fogunk bootolni. Egy másik bepipálható opció a Shareable tulajdonság, amivel elérhető tesszük más virtuális gépek számára is az adott fizika lemezt (ami mint tudjuk csak a virtuálisan fizikai).

- IDE Cdrom 1: Ez pontosan ugyan olyan tulajdonságokkal bír, mint az IDE Disk 1, azzal a különbséggel, hogy itt a Read-only kapcsoló változtathatatlanul be van kapcsolva (érthető okokból).
- NIC: A virtuális hálókártya beállítása. A MAC-címet itt már nem lehet állítani, azt a telepítésnél a Virt-manager automatikusan generálta számunka. Ezen menüpont alatt egyedül a Device modelt váltogathatjuk.

- Mouse: Beállítási lehetőség nincs, viszont közli velünk, hogy automatikusan -usbdevice tablet kapcsolóval települt a gép, tehát bármikor kihúzhatjuk az egeret a virtuális gépből.
- Display VNC: A beépített VNC szerver beállítása.
- Video: Videokártya modelljét választhatjuk ki. Mi alapértelmezés szerinti, és a QEMU által megvalósított Cirrus videokártyát használjuk 9 MB RAM-mal.

A fent említett hardvereket eltávolíthatjuk, szerkeszthetjük és újakat adhatunk hozzá. Ha például szeretnénk még egy iSCSI storage szerver által kiszolgált 200 GB-os tárat a virtuális gépünknek dedikálni, akkor azt úgy tehetjük meg, hogy létrehozunk egy új storage-et, majd abban egy 200 GB-os image-et, és ezt már hozzá tudjuk adni a „Hardver hozzáadása” menüpont alatt mint újabb IDE vagy SCSI diszk (attól függ, melyiket választjuk).

Hangkártya hozzáadására is itt van lehetőségünk. Itt is többféle chipsetet választhatunk. Én a default beállítást próbáltam ki.

Egy érdekes hozzáadható hardverelem a Watchdog, amivel a bizonyos hardverelemek, vagy az éppen futó program által okozott nem szokványos viselkedésekre (pl. system reset, hiba a fő programban) meghívódik, és átvéve a vezérlést reagál. A triggerfeltétel és a reakció is változtatható a hardverelem hozzáadásánál.

4. Szerveroldali megoldások

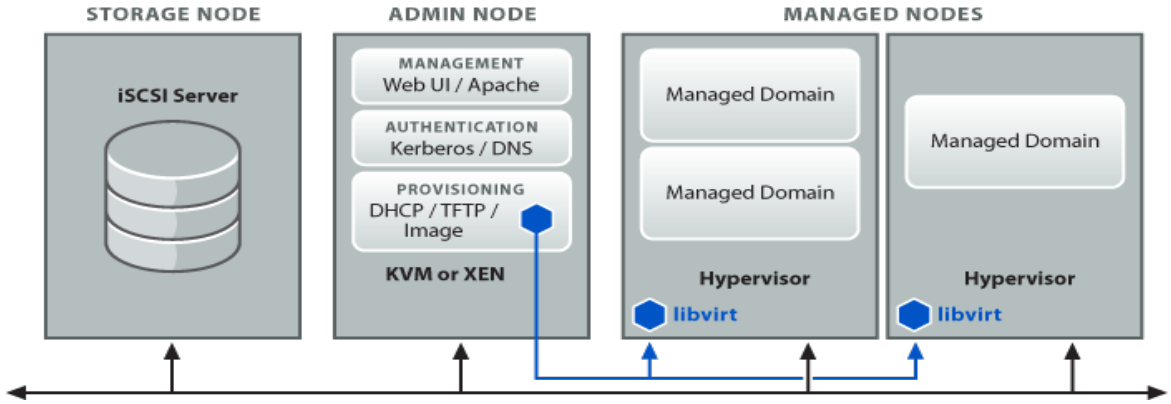
4.1 Motiváció

Mint azt az alcímben is említettem már, szeretnék kitérni a KVM technológiával megvalósított virtualizáció professzionális felhasználására. Ez jelentse azt, hogy a virtuális gépek kellő védelemmel vannak ellátva (nem fér hozzájuk közvetlenül senki), létezzen egy hypervisor operációs rendszer, amihez közvetlenül szintén ne lehessen hozzáférni, és menedzselhetőségét tekintve bármikor, bárhonnan, bármit el lehessen végezni, lehetőleg egyszerűen. Az egyszerűség mögött természetesen állhat egy teljesen testreszabottan konfigurált rendszer, viszont egy adott feladat megoldására létezzen egy menedzsment felületről kivitelezhető megoldás (pl. egy új virtuális gép indítása egy új projekthez 3-4 kattintás legyen). Lehetőleg kezelje a template-eket, képes legyen clusterekbe rendezni a gépeket, esetleg erőforrás-elosztást és live migrationt is valósítson meg.

Ezt azért tartom fontosnak, mert egy ilyen rendszer kiépítésével már akár kis vállalatok is egyrészt biztonságosabban (a szerver image-ek időközi lementése), költséghatékonyabban (kevesebb szerver = kevesebb áramfogyasztás vagy szerverbérleti díj, kevesebb hardver = kisebb amortizációs költség) tudnák megoldani szervereik üzemeltetését és dinamikus módosítását.

4.2 oVirt bemutatása

A feladat megoldására a libvirten alapuló Ovirt rendszert választottam. A rendszer vázlata alább látható:



A rendszert a Red Hat fejleszti, így érthető módon ajánlatos Fedora/CentOs rendszeren alkalmazni. A telepítése korántsem egyszerű feladat. Mind a szerver, mind a node csak forrásban jelenik meg, esetleg nagyon régi RPM-eket találunk. A forráskódot Git repositoryban teszik közzé, így mindig elérhetjük a legfrissebb változatot. A forráskódot először openSUSE alatt próbáltam lefordítani, ami sajnos sikertelenül végződött egy dbus-devel csomagtól való függés miatt. Akárhogyan próbáltam, nem sikerült a fordítást végrehajtanom. Ezek után egy Fedora 14-en próbáltam fordítani a programot, itt le is fordult a kód, és legenerálta a szükséges RPM-eket. Egy node RPM-et generált, amiből egy segédprogrammal könnyedén készíthetünk ISO fájlt, vagy esetleg megtehetjük azt is, hogy PXE-n keresztül bootoltatjuk be a menedzselt node-okat, így HDD-re sincs szükség a vezérelt gépekben. Sajnos azonban a Fedora 14 alatti tesztelésem is kudarcba fulladt: A Ruby verziója nem volt támogatott (deprecated hibák százai, aztán egy végzetes verzióhiba miatt leállt a telepítés), így nem tudtam a szerveroldali részt feltelepíteni. A Ruby amúgy a Ruby on Rails alapokon nyugvó webes menedzsment felülethez kell.

4.3 iSCSI target telepítése

Az oVirt csak és kizárólag iSCSI vagy NFS storage-et képes háttértárnak használni, tehát kapásból plusz egy szerverre szükségünk van, lehetőleg nagy tárkapacitással. Ezt a következő módon telepíthetjük (szintén Fedorára ajánlatos):

Logikai kötet létrehozása az iSCSI számára:

```
# lvcreate -n iSCSI1 -L +10G /dev/VolGroup00
```

Használhatunk fájlt is logikai kötet helyett:

```
dd if=/dev/zero of=/fs.iscsi.disk bs=1M count=10240
```

iSCSI target elindítása:

```
# service tgtd start
# tgtadm --lld iscsi --op new --mode target --tid 1 -T
host:storage
```

A fenti műveletekhez installálnunk kell a scsi-target-utils csomagot.

A target kiajánlása LUN-ként:

```
# tgtadm --lld iscsi --op new --mode logicalunit --tid 1 --lun 1
-b /dev/VolGroup00/iSCSI1
```

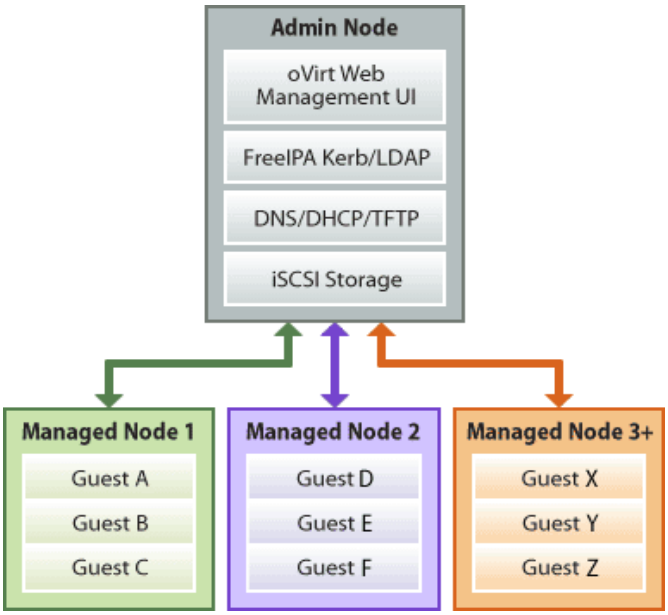
LUN elérhetővé tevése a következő paranccsal történhet:

```
# tgtadm --lld iscsi --op bind --mode target --tid 1 -I ALL
```

A telepítésnél kérni fogja az iSCSI target címét, ott értelemszerűen azt a szervert kell megadni, ahol ezeket a fenti műveleteket elvégeztük.

4.4 Admin node telepítése

Az admin node feladata a teljes rendszer vezérlése. Legfőbb funkciói az ábráról leolvashatóak:

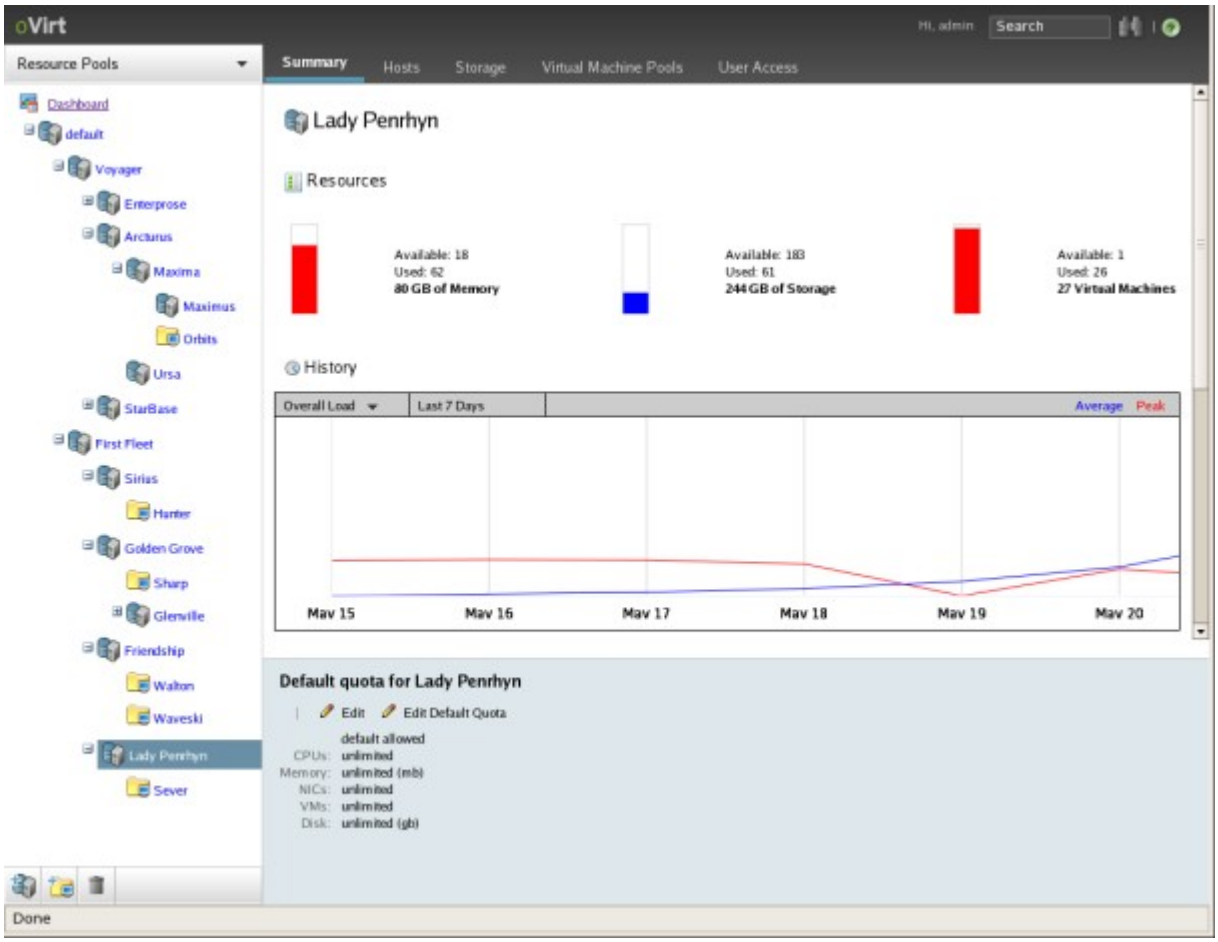


Az admin node azokból az RPM-ekből telepíthető, amiket a fordítás során kapunk. Nekem itt akadt el a kipróbálása a dolognak, mivel a Ruby verzióbeli eltérései miatt nem engedett tovább. Az oVirt oldalán sok funkcionális ábrát láthatunk az admin node pontos működéséről.

Két fontos funkcióját említeném meg az admin node-nak:

- Cobbler: PXE-n keresztül képes kiajánlani a menedzselt node-ok számára a bootolandó imaget.
- Webes felület: Innen irányíthatjuk a teljes rendszert.

A webes rendszer így néz ki:



4.5 Managed node telepítése

A letöltött forrás fordítása után kapunk egy node RPM-et, amiből aztán generálhatunk egy ISO-t, amit kiírhatunk CD-re, és telepíthetjük onnan a vezérelt gépeket, vagy PXE-n keresztül is használhatjuk a managed node-okat a Cobbler segítségével. A managed node tehát nem más, mint egy kicsi, robusztus operációs rendszer, mely folyamatosan kommunikál az admin node-dal. Belépni rá közvetlenül nem tudunk, és adatot sem tárolunk rajta, csak a számítási kapacitását használjuk. A rajta futó operációs rendszert nevezhetjük hypervisornak.

5. Összefoglalás

5.1 Összefoglalás

A KVM egy egyszerűen alkalmazható virtualizációs technológia, melyet előszeretettel használhatunk mind kliens-, mind pedig szerveroldalon. Óriási előnye abban rejlik, hogy bármilyen Linux típusú operációs rendszeren üzembe helyezhetjük, tehát magunknak állíthatjuk elő a teljes virtualizációs környezetet. Hardverigénye csupán annyi, hogy a processzorunk támogassa a VT-X vagy AMD-V technológiát.

Sok menedzsment felületet kapunk hozzá, ezek közül a legnépszerűbb a libvirten alapuló Virt-Manager, de szerveroldali megoldások között sem szenvedünk hiányt. Választhatjuk pl. az oVirtet, ConVirtet vagy a Ganetit.

5.2 Források

- Az oVirtről bővebben: https://fedorahosted.org/ovirt/
- oVirt screenshotok: http://ovirt.et.redhat.com/screenshots.html
- KVM-ről minden: http://ovirt.et.redhat.com/screenshots.html
- KVM-es menedzsment felületek: http://www.linux-kvm.org/page/Management_Tools

5.3 Wordle!