Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

Legal note

These slides form an outline of the curriculum and contain multiple references to the official Red Hat training materials (codes RH124, RH134 and RH254), since students of this subject are also eligible for the named RH learning materials. Some diagrams and pictures originating from the RH courses will show up in this outline as there is an agreement in place which allows for our Faculty to teach this subject based on the Red Hat curriculum.

All other pictures and diagrams sourced from third parties are explicitly marked with a reference to the origin.

Be aware that the mentioned parts and also this series of slides as a whole are property of their respective owners and are subject to copyright law.

Legal note

All students of PTE-MIK who have officially taken the course „Linux System Administration” are eligible to download these slides from the Faculty’s internal network, and are eligible for license keys for Red Hat System Administration online learning materials at rhlearn.gilmore.ca as part of their training program.

Do not share, redistribute, copy or otherwise offer these learning support materials, license keys or account information either for money or free of charge to anyone, as these materials contain **intellectual property.**

Unauthorized distribution is both against the law and university policy and will result in an in-campus inquiry, suing for damages and/or criminal prosecution by the University of Pécs, Red Hat Inc. and other parties.

NFS

Lecture 12

Chapter 1

NFS

- Network File System is an open standard and it is the native protocol for sharing directories over a network for Unix like operating systems.
- Directory sub-trees made available to network users (shared directories) are called **exports** in NFS termino-logy. A server exports directories, a client mounts them.
- NFS is an internet standard, Windows 7 and newer versions can connect to NFS exports without any third-party software.
- NFS versions 2 and 3 support both TCP and UDP as a transport layer, NFSv4 only supports TCP. NFSv4 also introduced advanced security and authentication features, efficient sparse file support and more.
- By default RH 7 uses version 4 of the NFS protocol but can fall back to previous version if needed.

Shared filesystems, locking

- Multiple processes can open the same file. The kernel has knowledge about all such operations (they happen via system calls) on a local filesystem and keeps track of what file/range is being read and/or written.
- Network shares on the other hand make it possible for multiple processes of different kernels (hosts) to open the same file. In this case there is no single kernel on the network that would be aware of all the open files involved in an export. This is a universal problem of networked filesystems (not just NFS).
- NFSv2 and 3 doesn't support file locking at all by itself, but has an ancillary protocol available called NLM (Network Lock Manager).
- File locking is supported as part of the NFSv4 protocol, all the locking information is available at the server and is propagated through the network to clients and back.

NFS security

- Version 2 and 3 of NFS authorizes clients based on their IP address, there is no authentication involved. If your computer can assume a specific IP address (for e.g. you have privileges to set your own IP address), than you have access to the relevant share.
- File and directory permissions are enforced on exports, but this is only effective if the users and groups are managed centrally on all the computers involved.
- If you can assume a UID or GID on your local computer (for e.g. you have root access to your own workstation, then you can locally su or sudo to anyone), you have the same permissions on the export as the given UID and GID would have on the server side filesystem.
- To mitigate the risks arising from that, the **squashing** mechanism was introduced.

NFS security

- Squashing means, that the a users permissions are remapped to the nfsnobody user (UID=65534) or another anonymous user (specified with the **anongid=** and **anonuid=** export options). The root user perms are intentionally squashed on the server side by default (use **no_root_squash** to defer), the **all_squash** export option squashes all user permissions to anonymous.
- NFSv4 introduced additional security methods:
  - none: no access check at all, with all user mapped to the anonymous user (nfsnobody).
  - sys: default, the same as NFS v2 and v3 security, standard Unix permissions and ACL checks work.
  - krb5: like sys (standard Unix permissions apply), but client must also prove identity with Kerberos.
  - krb5i: like krb5 with all request getting signed (no tampering with requests possible).
  - krb5p: in addition to Kerberos authentication all traffic is encrypted, has a performance impact.

Exporting shares

- The **exports** file format: It contains one line for each share, and at least two fields for each line: First column is the shared directory path, second column contains the list of allowed clients, and the export options in effect for the client, in braces. Further clients(options) pairs can optionally be specified in additional columns.
- The client specifier can be a single IPv4 or IPv6 address, a subnet, a resolvable FQDN or a domain name with wildcards.

/etc/exports (example)

```
/share/exported_directory 192.168.10.0/24(rw)
/another/share 10.0.0.1(rw) 10.0.0.2(ro,all_squash)
/some/export workstation[0-20].example.com(sync)
```

- Some important export options: sync, async, rw, ro, root_squash, all_squash, anongid, anonuid, no_acl, fsid, sec=krb5p, sec=sys, sec=none, ...

Exporting shares

- To use NFS, install the nfs-utils package (both on server and client computers): **yum install nfs-utils**.
- Edit the **/etc/exports** and/or the .exports files in the **/etc/exports.d** drop-in directory of the server.
- Open up port 2049/tcp on the server's firewall: **firewall-cmd --add-service=nfs --permanent; firewall-cmd --relaod;**
- Finally start and enable the service with systemd: **systemctl start nfs-server; systemctl enable nfs-server**
- Whenever modifying the exports file(s) either restart or notify the nfs-server about the changes with **systemctl restart nfs-server** or **exportfs -r**.
- Complete practice **RH254 CH 8.2. (NFS export)**

Mounting shares

- Once identified the correct share, you can simply mount it with e.g. **mount.nfs -o sync,nfsvers=3 serverN:/share /mnt/nfs_share**. Just like any other mount command, this also has options, a source, and a target mount point. Use the **nfsvers** option if you would like to defer from the default NFS version of 4. Use the **sec** mount option to set security (eg. sec=krb5i)
- You can create permanent nfs mounts in **/etc/fstab** with the method learned before. (what, where, type, opts)
- To identify an exported resource use the following command on the client for NFSv3 and v2: **showmount -e server_name_or_ip**. To identify available exports on NFSv4: mount the root directory of the server: **mount.nfs serverX:/ /mnt_point** then **ls /mnt_point**. Do this as root. You won't have access to Kerberos shares, but they will also be listed.

Secure exports

- Kerberos authentication requires at least the presence of a proper /etc/krb5.keytab file. You usually get this file from the admin responsible for security, when joining a Kerberos Realm. Creating such a realm is not covered in this class, but a ready made keytab will be provided for the purpose of the practices.
- Kerberos security options require the **nfs-secure-server** systemd unit to also be enabled and started besides normal **nfs-server** unit. On the client side the **nfs-secure** unit is the only one needed for this.
- Have an export based on the following example: **/secured_export *.example.com(sec=kerb5p,rw)**
- Mount it on the client with: **mount.nfs -o sec=krb5p serverx:/secured_export /mnt/secure_nfs_mount**
- Complete practices **RH134 CH 11.2.** (NFS mount) **and RH254 CH 8.4.** (Secure NFS)

NFS and SELinux

- By default the **nfs_export_all_ro** and **nfs_export_all_rw** booleans are enabled, allowing access to almost any file over an NFS export. To tighten this, turn these booleans off and label your shared files with either **nfs_t** or **public_content_t**. To allow write access use **nfs_t** or **public_content_rw_t** in conjuction with the **nfs_anon_write** boolean.
- SELinux context is not propagated to clients by default. All files have the **same** (by default: nfs_t) label on the client's mount point. The default can be overridden with the **-o context="*something*"** mount option: still all files will have the same context on the client.
- NFSv4.2 supports SELinux context propagation. To enable this, change the relevant line in **/etc/sysconfig/nfs** to RPCNFSDARGS="-V 4.2" and mount with **-o v4.2** on the client.

Autofs

Lecture 12

Chapter 2

Autofs

- Autofs service can be instructed to watch directories and mount filesystems on them as the directory is accessed. Autofs defaults to handling nfs mounts, but can be used with many other types of filesystems.
- Autofs provides a convenient way of letting users without root privilege to mount a restricted set of filesystems. It is also commonly used to define wildcard mounts (e.g. each user can have their respective home directories auto-mounted over nfs without creating a constant mount for all the users on all the hosts of the whole network.)
- Autofs is able to dismount and clean up after itself when a mount point is not used anymore, alleviating both server and client side resource load.

Autofs

- Autofs has two mapping modes: direct and indirect.
- In both cases autofs mounts directories under /tmp/mnt based on mapping rules described in configuration files. In both cases, when a path under autofs's supervision gets accessed (e.g. ls) it will immediately get mounted.
- Direct mounts can be used for absolute paths. Each mount point entry will have a separate mnttab entry, (it can grow quite large). Changes to configuration require autofs to be reloaded.
- In case of indirect maps, autofs creates a virtual file-system on a given directory, and works more like a symlink server, that manipulates links to the /tmp/mnt directory dynamically, based on current needs. This scheme is used for relative paths under the indirect directory and only the indirect directory itself gets into the mnttab.

Autofs master configuration

- Autofs has a master configuration, stored in the **/etc/auto.master** file and **/etc/.autofs.master.d** drop-in directory. The master configuration contains references to other configuration files that contain the direct or indirect mappings for specific directories. A reference to a direct map configuration file always has a path of **/-** while an indirect reference contains the relevant directory path.

/etc/auto.master [excerpt]

```
/-                /etc/config1.direct
/automnt_shares /etc/config2.indirect
```

- Config file names are freely selectable, but usually something informative is used.

Autofs configuration

- Indirect mapping files have relative paths. This example with the former auto.master file means the following: Automount provider.com:/shared/somethg to the local mountpoint of /automnt_shares/something whenever the local path gets accessed.

/etc/config2.indirect

```
something -fstype=nfs,rw provide.com:/share/somethg
```

- Direct mappig file's format is very similar, but it contains an absolute path.

/etc/config1.direct

```
/local/something -sync,rw provide.com:/share/someth
```

- The middle column contains the regular mount options in a comma separated list, without spaces. The fstype=nfs option can be omitted, since it is the default.

Autofs configuration

- Any type of filesystem can be automounted with a correct fstype option. An SMB automount example looks like this: **samba_mount -fstype=cifs,credentials=/etc/smb.cred ://smb_server/sharename**
- The indirect mount can be used very efficiently with wildcards. For e.g. when an NFS server exports the user home directories, they can be automounted on any workstation in the network with the following simple line.

```
/etc/homedirs.indirect
*    -rw,sync    server:/shares/home_dirs/&
```

- If accessing any directory in the indirect root, the * (asterisk) will match against it, and the & (ampersand) will substitute whatever the first match was.
  E.g.: ls /automnt/**bob** command will automatically mount server:/shares/home_dirs/**bob** to /automnt/**bob** directory.

Autofs setup

- Issue **yum install autofs** to install necessary packages.
- Edit the master autofs file or a file in the master drop-in directory, and name a direct or indirect mapping file.
- Create the named direct/indirect mapping file and set at least one line in it.
- Execute **systemctl enable autofs** and **systemctl start autofs** to start and enable the service.
- Whenever modifying direct mounts, remember to restart the service. When modifying an indirect mount no restart is necessary, changes take effect automatically the next time the directory is mounted.
- Complete practice **RH134 CH 11.4. (automount NFS)**

Backup, archiving

Lecture 12

Chapter 3

Copying files between systems

- In this chapter a few well-known ways will be shown on how to copy files between systems. Remember that making a copy of files/database dumps inside the same filesystem is not considered a backup, since in case of a hard disk failure all copies are lost at once.
- One common way for backing up whole directory structures is to collect all the FS object under a path to **tarball file**, compress it and copy it to another disk or another host. This can be done manually or by the means of a script that gets scheduled for. e.g. by cron.
- The other well known way of creating backups is via the **rsync** utility, which can differentialy replicate a whole directory tree to a local or even directly to a remote destination. Also consider cron for scheduling this.
- Use the regular **cp** command with the **-a** switch (archive) for preserving perms, owner, times,… (root may be rqd.)

Creating tar backups

- Let’s suppose you already have a database dumped to an sql file or you already have a directory ready for archival.
- First you should run the **tar** command, which will collect all files and their basic metadata (e.g. timestamps) to one single file. This was originally intended for tape backup drives, which are not made to retrieve individual files quickly anyway.
- The tar command can compress collected data on the fly with many different algorithms, directly producing a .tar.gz, .tar.bz2, .tar.xz, etc… file.
- Then you should copy the file with the regular **cp** command to another local disk/network share or using the **scp** command as discussed in Lecture 10.

Tar switches

- Tar is a BSD style command, does not need a - (dash) for arguments. The most common switches are:
  - t for testing a tar archive
  - c for creating a tar archive
  - x for extracting the files and dirs in the archive
  - v for verbose output
  - f for telling the archive filename (this switch has an argument: the filename)
  - z to also compress data while processing (gzip alg.)
  - j to compress with bzip2 algorithm
  - J for compressing with the xz algorithm
  - p for preserving permissions (while extracting)

Tar switches

- Remember that switches can be combined and reordered, but some switches have arguments, like the f switch. Therefore f is usually left last.
- Tar example for creating an archive of a specific folder in gz format: **tar czvf /backups/jane.tar.gz /home/jane** To extract it to a directory: **cd /extracted; tar xzvf /backups/jane.tar.gz**
- To transfer files securely to another host: **scp /backups/*.tar.gz backupserver:/data/workstation1** or **scp -r /backups bcksrv:/datastore** for transferring an entire directory.
- You may also use the **sftp** command. This will execute remote system commands through an SSH tunnel, but usage is the same as of the regular **ftp** CLI client.
- Complete practices **RH 124 CH 12.2.** (tar create, extract) and **CH 12.4.** (scp)

Rsync priciples

- Rsync can be used synchronize complete Filesystem sub-trees to another location. It can operate in two modes: EITHER source and destination are both local OR one of them is a remote computer.
- Remote transfer can be done via the rsync protocol or over SSH. Former requires that the server runs an rsyncd process. Latter only requires that both sides have the rsync command installed, it will be automatically called on the remote side via an SSH session.
- Rsync can efficiently update large data sets over narrow-band networks, because it only send the differences, that occurred since the last synchronization. Of course it takes about the regular amount of time to transfer a backupset for the first time. Besides the difference transfer capabi-lities rsync can also implement regular compression.

Rsync pricipkes

- Rsync’s simplified operating method is the following:
  - First it will compare source and dest. file metadata (size, timestamps, …) one-by-one for each element of the directory tree. In case of exact match, nothing further will be done, they are considered to be in sync, otherwise they are marked in memory for further examination.
  - Rsync will transfer a marked file as a whole file if both src. and dst. are on local filesystems (like cp). If one of them is remote, than the file will get sliced to logical blocks. Rsync will calculate a hash for each of the blocks on both sides and only transfer the hash over the network, for comparison.
  - In case the current blocks hash does not match, it will transfer the block only, not the whole file. This is repeated for all the blocks in all the marked files.

Rsync usage

- The general structure of the rsync command is the following:
  rsync -*switches source_dir user@remotehost:/dest_dir*
  where one of source or destination can be remote host OR both can be a local directory or mount point.
- The most important switches are:
  - -v for verbose
  - -a for archive, the same as -rlptgoD
  - -r for recurse into directories
  - -l for copy symlinks as symlink (do not dereference)
  - -p to preserve permissions
  - -t to preserve modification timestamp
  - -g to preserve group
  - -o to preserve owner

Rsync usage

- The most important switches are (continued):
  - -D to preserve device and special files
  - -H to preserve hard links
  - -A to preserve ACLs
  - -X to preserve extended attributes (SELinux labels)
  - -z to compress transferred data
  - -u for update-only (skip files newer on the dest.)
  - -h for human readable units
  - -P for show progress during transfer (pv)
  - -m for delete empty directories on destination
  - -E to preserve executability
  - --inplace for updating file blocks in palce (no tmp)
  - --sparse for treating blocks of zeroes efficiently

Rsync usage

- The most important switches are (continued):
  - --compress-level=N where N=[1..9]
  - --numeric-ids for skipping passwd file id<→name resolution (just copy the numeric value)
  - --del for deleting files on destination, that are not present at the source anymore
  - --log-file=*filename* for record output in a log file
  - --stats to show statistics at the end of transfer
  - -e 'ssh -p 12022' for telling the underlying remote execution method, for e.g. can hold arguments for keyfile, ssh port, or non-standard ssh command
  - --filter=*filterlist.file* for file filtering rules to be read from given filename
- Complete practices **RH 124 CH 12.6.** (rsync) and **12.7.**

Virtualization basics

Lecture 12

Chapter 4

KVM

- KVM stands for Kernel-based Virtual Machine. The Linux Kernel itself can act as a full-virtualization hypervisor for VMs. It competes with commercial products like Vmware.
- KVM was Originally developed by Avi Kivity at Qumranet. This company was bought by RedHat in 2008.
- KVM is a component of the mainline kernel. The kvm-intel or kvm-amd kernel module (and a proper CPU) is required for the kernel to act as a fully hardware assisted hypervisor. Use **modprobe** command to load a module.
- Note that KVM can be categorized as both Type I and Type II hypervisor. This is a matter of active debate. KVM turns the kernel to a bare-metal hypervisor (Type I) while the OS functionalities still work at full extent. KVM works in close cooperation with the qemu-system-x86 emulator. All Vms are processes from the OSs perspective (trait of Type II hypervisors).

qemu

- Qemu is an open-source, efficient emulator family. It can be compiled to run on a wide range of systems to emulate another range of systems. For e.g. qemu-system-x86, qemu-system-arm, qemu-system-mips64el, etc.
- In case of hardware assisted full virtualization (for e.g. running qemu-system-x86 on a modern x86 CPU) qemu doesn't need to trap, binary translate or emulate the x86 instructions. Instead its role is mainly to emulate hardware devices, such as virtual or para-virtual storage controllers, network cards, display adapters.
- The paravirtualized hardware devices emulated by qemu are natively supported by Linux and also supported by Windows guest when installing the certified, digitally signed, mature RedHat virtio-win drivers.

libvirt

- The qemu-KVM technology, with proper hardware, also allows for assigning physical hardware devices (like PCI cards) directly to VMs via IOMMU. This can offer uniform hardware (storage, network, etc... → easy backup and migration) for VMs with for e.g. physical GPUs attached for ultimate parallel processing performance. The hybrid physical-virtual systems backed by qemu-KVM are extremely efficient. If properly configured, they can provide up to 97-98% of the native performance.
- The libvirt library and API is an open source framework for managing virtulized systems. The qemu-KVM back-end is only one of libvtirts supported technologies. (with connectors it can manage VirtualBox hosts, Xen hosts, the OpenStack platform and more...)
- Libvirtd is the resident daemon process that manages hypervisors and receives connections from user frontend.

virsh

- Just like many other daemons, libvirtd has a CLI client as well as a GUI client. There are also several Web front-ends working with the libvirt API.
- Libvirtd stores virtual resource settings (for individual resources or complete virzual domains) in xml files under /etc/libvirtd. Look up the format at: https://libvirt.org/formatdomain.html
- The standard command line interface to libvirtd is **virsh**. Virsh can be used to manage domains (VMs) defined in libvirt. Take the following command examples: **virsh list** will list all active domains. The **--all** switch lists every domain, including the ones in shutdown state.
- To manage the running state of domains use **virsh start/shutdown/reboot/reset/destroy** ***VMname*** to start, gracefully shut down, gracefully reboot, force reset or force power-off a VM respectively.

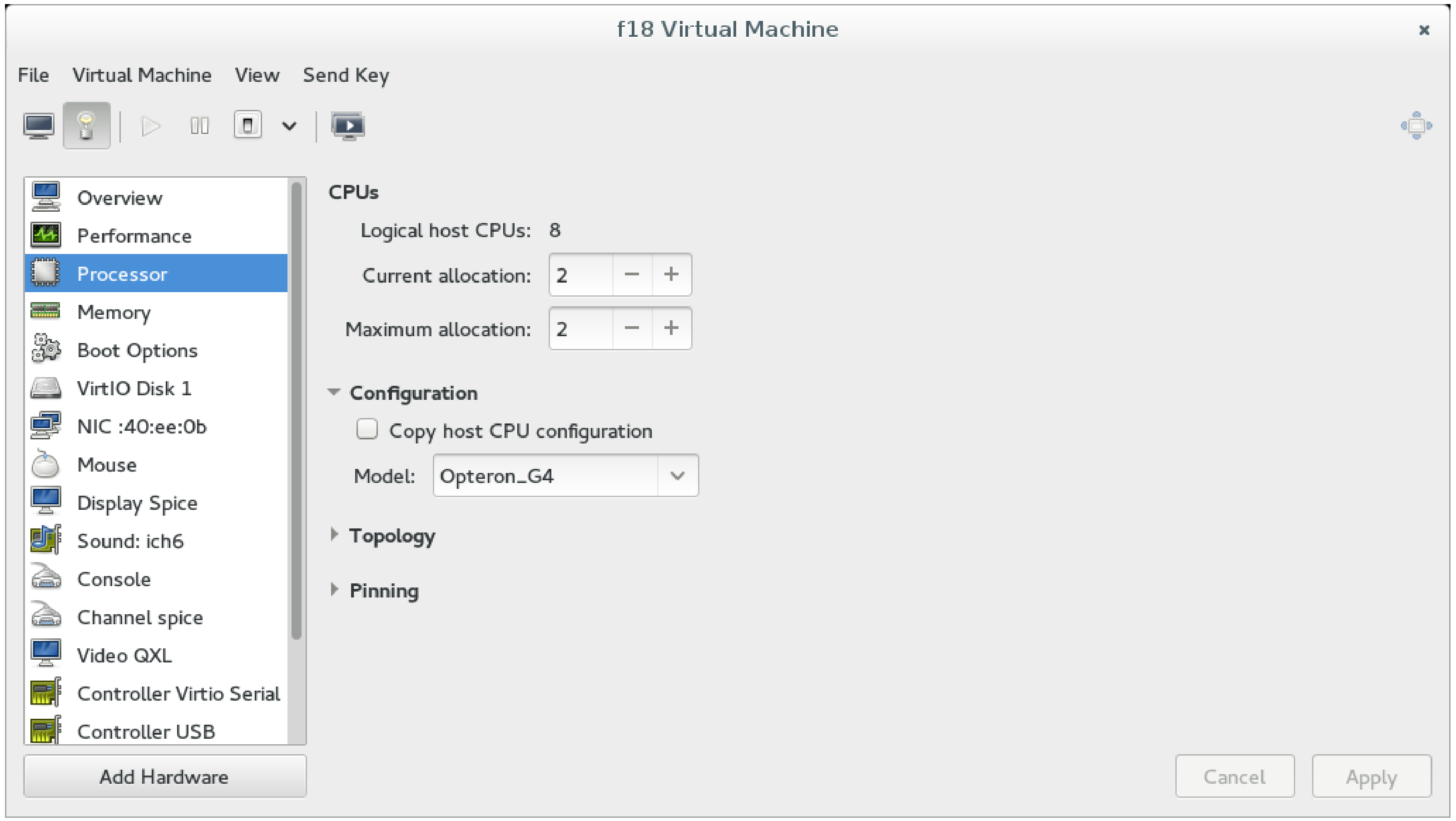
virsh

- Power management command examples:
  virsh dompmsuspend/dompmwakeup ***VMname***
- To edit an existing VM's xml configuration with the default editor use **virsh edit** ***VMname***.
- Use **virsh create vm.xml** to cretae a new VM from a domain xml file and start it. Use **virsh define vm.xml** to create/update the xml descipiron of a VM without starting it. Use **virsh undefine** ***VMname*** to remove a VM.
- Use **virsh console** ***VMname*** to connect current terminal to the serial console of the guest.
- Issue **virsh nodeinfo** ***VMname*** to get a domain's details.
- Look up **man virsh** for the snapshot-create-as, snapshot-delete, snapshot-info, snapshot-list, snapshot-revert sub-commands and further details.

virt-manager

- The virt-manager package installs a GUI client to libvirtd. The command name is also virt-manager.
- It has a very clear user interface and provides wizard-like interface for creating new VMs, managing storage pools, virtual networks. The user interface is quite comfortable (somewhat resembling to VirtualBox's GUI concepts).
- Virt-manager can connect to multiple virtualization host computers (multiple libvirt daemons) and manage whole server sets (like cloud infrastructure) at the same time.
- Virt-manager also has a built-in spice client. Spice is a VNC-like open source protocol, with additional features like sound and USB redirect. Spice is the default interface to VMs with emulated graphics and input devices.
- Follow **RH124 CH15.2.** for the GUI demonstration.

virt-manager



CPUs

Logical host CPUs: 8

Current allocation: 2

Maximum allocation: 2

Configuration

[ ] Copy host CPU configuration

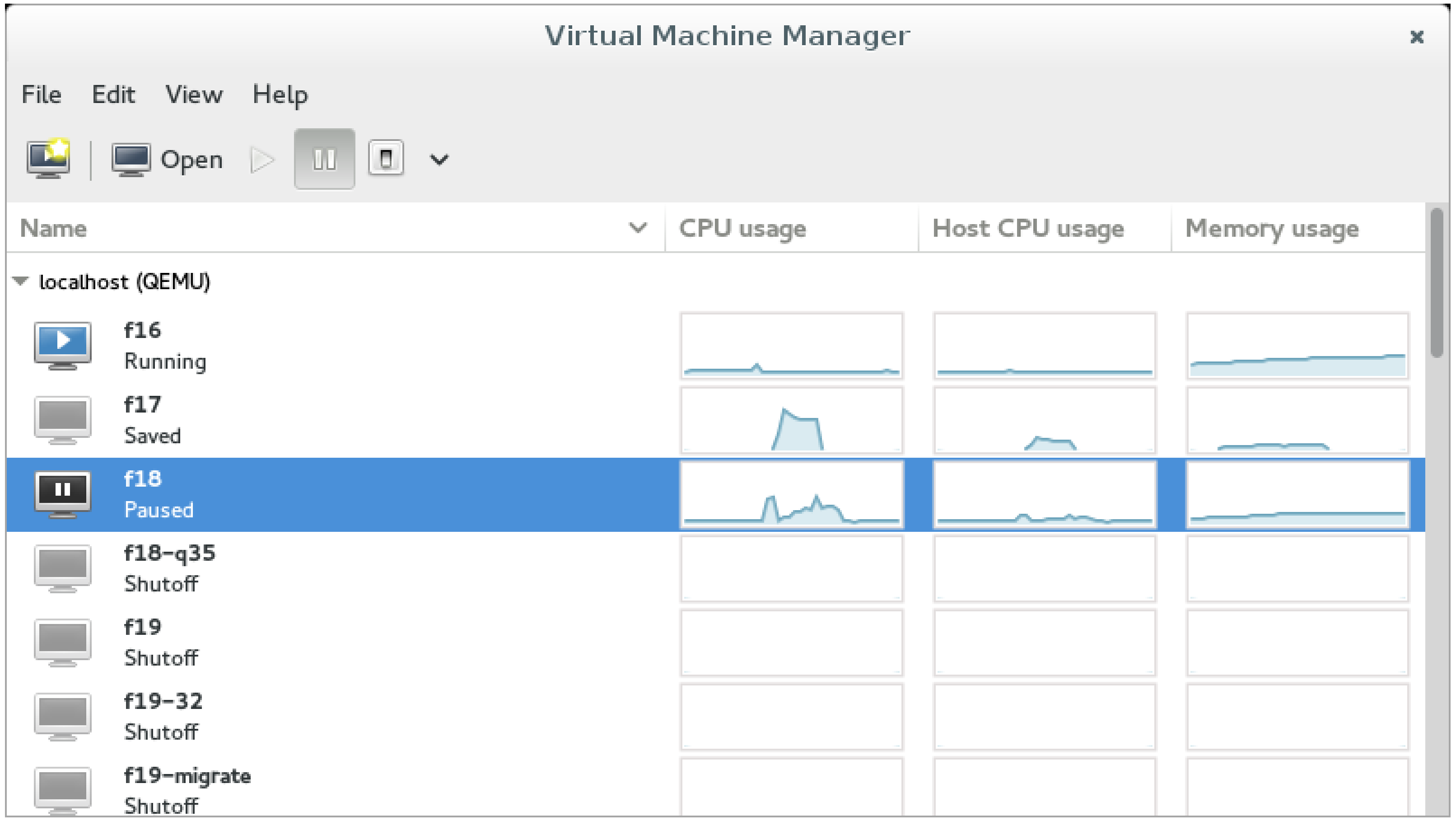
Model: Opteron_G4

Topology

Pinning

https://www.virt-manager.org/wp-content/uploads/2014/01/details.png

virt-manager



Virtual Machine Manager

Name	CPU usage	Host CPU usage	Memory usage
localhost (QEMU)			
f16 Running			
f17 Saved			
f18 Paused			
f18-q35 Shutoff			
f19 Shutoff			
f19-32 Shutoff			
f19-migrate Shutoff			

https://www.virt-manager.org/wp-content/uploads/2014/01/manager.png

Recommended reading

Lecture 12

Chapter 5

Relevant chapters in RH:

RH134 CH11	(NFS, autofs)
RH254 CH8	(File storage servers/NFS)
RH124 CH12	(Archiving and copying files)
RH124 CH15	(Virtualized systems)

Read by next lecture:

RH254 CH5	(E-mail transmission)
RH254 CH6	(DNS server)

Use your rhlearn.gilmore.ca login to access the learning materials.

Thank you for your attention!

Lecture 12, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK