RHCSA - EX200

Trainer

Ali Aydemir
CISCO, CCIE #47287 SP/RS, CCSI#35413
AVAYA, ACE-Fx #169
HUAWEI, HCDP

RHCSA Timetable

Day	AM	Lunch	PM
1	-Installing RHEL Server -Using Essential Tools	--	-Essential File Management -ToolsWorking with Text Files -Connecting to a RHEL Server
2	-User and Group Management -Permissions Management	--	-Configuring Networking -Process Management -Working with Virtual Machines
3	-Installing Software Packages -Scheduling Tasks	--	-Configuring Logging -Managing Partitions -Managing LVM Logical Volumes
4	-Basic Kernel Management -Configuring a Basic Apache Server	--	-Managing and Understanding the Boot Procedure -Essential Boot Procedure Troubleshooting
5	-Managing SELinux -Configuring a Firewall	--	-Configuring Remote Mounts and FTP -Configuring Time Services

Chapter 6:

User and Group Management

Chapter 6 Objectives

- The following topics are covered in this chapter:
  - Creating and Managing User Accounts
  - Creating and Managing Group Accounts
  - Logging in Through an External Authentication Server

- The following RHCSA exam objectives are covered in this chapter:
  - Create, delete, and modify local user accounts
  - Change passwords and adjust password aging for local user accounts
  - Create, delete, and modify local groups and group memberships
  - Configure a system to use an existing authentication service for user and group information

Different User Types

- In this chapter, you learn how to create and manage user accounts. Before diving into the details of user management, you learn how users are used in a Linux environment.

Users on Linux

- On Linux, there are two ways to look at system security. There are **privileged** users, and there are **unprivileged** users. The default privileged user is **root**. This user account has full access to everything on a Linux server and is allowed to work in system space without restrictions. The root user account is meant to perform system administration tasks and should be used for that only. For all other tasks, an unprivileged user account should be used.

Users on Linux

- To get information about a user account, you can use the **id** command. When using this command from the command line, you can see details about the current user. You can also use it on other user accounts to get details about those accounts.

Listing 6.1 Getting More Information About Users with **id**

```
[root@localhost ~]# id linda
uid=1001(linda) gid=1001(linda) groups=1001(linda)
```

Working as Root

- On all Linux systems, by default there is the user root, also known as the superuser. This account is used for managing Linux. Root, for instance, can create other user accounts on the system. For some tasks, root privileges are required. Some examples are installing software, managing users, and creating partitions on disk devices. Generically speaking, all tasks that involve direct access to devices need root permissions.

Working as Root

- When you log in as root in a graphical environment, all tasks that are executed are running as root as well, and that involves an unnecessary security risk. Therefore, you should instead use one of the following alternative methods. Table 6.1 provides an overview of these methods.

Table 6.2 Methods to Run Tasks with Elevated Permissions

su	Opens a subshell as a different user, with the advantage that only in the subshell commands are executed as root
sudo	Allows you to set up an environment where specific tasks are executed with administrative privileges
PolicyKit	Allows you to set up graphical utilities to run with administrative privileges

Using *su*

- The **su** command allows users to open a terminal window, and from that terminal start a sub shell in which the user has another identity. To perform administrative tasks, for instance, you can log in with a normal user account and type **su** to open a root shell. This brings the benefit that only in the root shell root privileges are used.
- If just the command **su** is typed, the username **root** is implied. But su can be used to run tasks as another user as well. Type **su linda** to open a subshell as the user *linda*, for example.

TIP Using **su -** is better than using **su**. When the **-** is used, a login shell is started, without the **-**, some variables may not be set correctly. So, you are better off using **su -** immediately.

Using *sudo*

- Instead of using the root user account, unprivileged users can be configured for using administrator permissions on specific tasks by using sudo. When sudo is configured, ordinary users have sudo privileges and to use these privileges, they will start the command using **sudo**.
- So, instead of using commands like **useradd** as the root user, you use an ordinary user account and type **sudo useradd**. This is definitely more secure because you will only be able to act as if you have administrator permissions while running this specific command.

Using *sudo*

- When creating Linux users during the installation process, you can select to grant administrator permissions to that specific user. If you select to do so, the user will be able to use all administrator commands using sudo. It is also possible to set up sudo privileges after installation. To do that in a very easy way, you have to accomplish a simple two-step procedure:
  - Make the administrative user account member of the group wheel by using **usermod -aG wheel user**.
  - Type **visudo** and make sure the line %wheel ALL=(ALL) ALL is included.

PolicyKit

- Most administration programs with a graphical user interface use PolicyKit to authenticate as the root user. If a normal user who is not a member of the group wheel accesses such an application, he will be prompted for authentication. If a user who is a member of the group wheel opens a PolicyKit application, he will have to enter his own password. For the RHCSA exam, you do not have to know PolicyKit. If you are interested, you can take a look at the man pages of the **pkexec** and **polkit** commands for more details.

Exercise 6.1 Switching User Accounts

1. Log in to your system as a non-privileged user and open a terminal.
2. Type **whoami** to see which user account you are currently using. Type **id** as well, and notice that you get more detail about your current credentials when using **id**.
3. Type **su**. When prompted for a password, enter the root password. Type **id** again. You see that you are currently root.
4. Type **visudo** and make sure that the line %wheel ALL=(ALL) ALL is included.
5. Type **useradd -G wheel lisa** to create a user lisa who is a member of the group wheel.
6. Type **id lisa** to verify that she has been added to the group wheel.
7. Set the password for lisa by typing **passwd lisa**. Enter the password **password** twice.
8. Log out and log in as lisa.
9. Type **sudo useradd lori**. Enter the password when asked. You notice that user lori will be created.

Managing User Accounts

- Now that you know how to perform tasks as administrative or nonadministrative user, it is time to learn how to manage user accounts on Linux. In this section, you learn what is involved.

System and Normal Accounts

- On a typical Linux environment, two kinds of user accounts exist. There are user accounts for the people who need to work on a server and who need limited access to the resources on that server. These user accounts typically have a password that is used for authenticating the user to the system. There are also system accounts that are used by the services the server is offering. Both user accounts share common properties, which are kept in the files /etc/passwd and /etc/shadow. Listing 6.2 shows the contents of the /etc/passwd file.

System and Normal Accounts

Listing 6.2 Partial Contents of the /etc/passwd user Configuration File

```
ntp:x:38:38::/etc/ntp:/sbin/nologin
chrony:x:994:993::/var/lib/chrony:/sbin/nologin
abrt:x:173:173::/etc/abrt:/sbin/nologin
pulse:x:171:171:PulseAudio System Daemon:/var/run/pulse:/sbin/nologin
gdm:x:42:42::/var/lib/gdm:/sbin/nologin
gnome-initial-setup:x:993:991::/run/gnome-initial-setup/:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
user:x:1000:1000:user:/home/user:/bin/bash
```

NOTE On many Linux servers, there are no user accounts that are used by people. Many Linux servers are installed to run a specific service, and if people interact with that service, they will authenticate within the service.

fields are used in */etc/passwd*

- **Username:** This is a unique name for the user. User names are important to match a user to his password, which is stored separately in /etc/shadow (see next). On Linux, there can be no spaces in the user name.
- **Password:** In the old days, the second field of /etc/passwd was used to store the hashes password of the user. Because the /etc/passwd file is readable by all users, this poses a security threat, and for that reason on current Linux systems the hashes passwords are stored in /etc/shadow (discussed in the next section).
- **UID:** Each user has a unique user ID (UID). This is a numeric ID. It is the UID that really determines what a user can do. When permissions are set for a user, the UID is stored in the file metadata (and not the user name). UID 0 is reserved for root, the unrestricted user account. The lower UIDs (typically up to 999) are used for system accounts, and the higher UIDs (from 1000 on by default), are reserved for people that need to connect directory to the server. The range of UIDs that are used to create regular user accounts is set in /etc/login.defs.

fields are used in */etc/passwd*

- **GID:** On Linux, each user is a member of at least one group. This group is referred to as the *primary group*, and this group plays a central role in permissions management, as discussed later in this chapter.
- **Comment field:** The Comment field, as you can guess, is used to add comments for user accounts. This field is optional, but it can be used to describe what a user account is created for. Some utilities, such as the obsolete finger utility, can be used to get information from this field. The field is also referred to as the GECOS field, which stands for General Electric Comprehensive Operating System and had a specific purpose for identifying jobs in the early 1970s when General Electric was still an important manufacturer of servers.
- **Directory:** This is the initial directory where the user is placed after logging in, also referred to as the *home directory*. If the user account is used by a person, this is where the person would store his personal files and programs. For a system user account, this is the environment where the service can store files it needs while operating.

fields are used in */etc/passwd*

- **Shell:** This is the program that is started after the user has successfully connected to a server. For most users this will be **/**bin/bash, the default Linux shell. For system user accounts, it will typically be a shell like /sbin/nologin. The **/sbin/nologin** command is a specific command that silently denies access to users (to ensure that if by accident an intruder logs in to the server he cannot get any shell access). You can create a file with the name /etc/nologin.txt that contains a message that will be displayed when a user who has /sbin/nologin as its shell tries to log in.

fields are used in */etc/shadow*

Listing 6.3 Sample Content from /etc/shadow

```
[root@localhost ~]# tail -n 10 /etc/shadow
ntp:!!:16420::::::
chrony:!!:16420::::::
abrt:!!:16420::::::
pulse:!!:16420::::::
gdm:!!:16420::::::
gnome-initial-setup:!!:16420::::::
postfix:!!:16420::::::
sshd:!!:16420::::::
tcpdump:!!:16420::::::
user:$6$3VZbGx1djo6FfyZo$/Trg7Q.3foIsIFYxBm6UnHuxxBrxQxHDnDuZxgS.We/
MAuHn8HboBZzpaMD8gfm.fmlB/ML9LnuaT7CbwVXx31:16420:0:99999:7:::
```

fields are used in */etc/shadow*

- **Login name:** Notice that /etc/shadow does not contain any UIDs, but user-names only. This opens a possibility for multiple users using the same UID but different passwords (which, by the way, is not really recommended).
- **Encrypted password:** This field contains all that is needed to store the password in a secure way.

fields are used in */etc/shadow*

- **Days since Jan 1, 1970, that the password was last changed:** Many things on Linux refer to this date, which on Linux is considered the beginning of days. It is also referred to as *epoch*.
- **Days before password may be changed:** This allows system administrators to use a more strict password policy, where it is not possible to change back to the original password immediately that a password has been changed. Typically this field is set to the value 0.
- **Days after which password must be changed:** This field contains the maximal validity period of passwords. Notice that by default it is set to 99,999 (about 273 years).

fields are used in */etc/shadow*

- **Days before password is to expire that user is warned:** This field is used to warn a user when a forced password change is upcoming. Notice that the default is set to 7 (even if the password validity is set to 99,999 days!).
- **Days after password expires that account is disabled:** Use this field to enforce a password change. After password expiry, users can log in no longer.
- **Days since Jan 1, 1970, that account is disabled:** An administrator can set this field to disable an account. This is typically a better approach than removing an account, as all associated properties and files of the account will be kept, but it can be used no longer to authenticate on your server.
- **A reserved field, which was once added "for future use":** That was a long time ago; it will probably never be used.

Most of the password properties can be managed with the **passwd** or **chage** command,
which are discussed later in this chapter.

Creating Users

- There are many solutions for creating users on a Linux server. To start, you can edit the contents of the /etc/passwd and /etc/shadow files directly (with the risk of making an error that could make logging in impossible to anyone; so better just do not). There is also **useradd**. useradd is the utility that you should use for creating users.
- To remove users, you can use the **userdel** command. Use **userdel -r** to remove a user, including the complete user environment.

Modifying the Configuration Files

- To add user accounts, it suffices that one line is added to **/etc/passwd** and another line is added to **/etc/shadow**, in which the user account and all of its properties are defined. It is not recommended, though. By making an error, you might mess up the consistency of the file and make logging in completely impossible to anyone.
- Also, you might encounter locking problems if one administrator is trying to modify the file contents directly while another administrator wants to write a modification with some tool.

Modifying the Configuration Files

- If you insist on modifying the configuration files directly, you should use **vipw**. This command opens an editor interface on your configuration files, and more important, it sets the appropriate locks on the configuration files to prevent corruption. It does *not* check syntax, however, so make sure that you know what you are doing because even by making a typo you might still severely mess up your server. If you want to use this tool to modify the /etc/shadow file, use **vipw -s**. To edit the contents of the /etc/group file where groups are defined, a similar command with the name **vigr** exists.

NOTE It is nice to know that vipw and vigr exist, but it is better not to use these utilities or anything else that opens the user and group configuration files directly. Instead, use tools like useradd and groupmod.

Using *useradd*

- The useradd utility is probably the most common tool on Linux for managing users. It allows you to add a user account from the command line by using many of its parameters.

- Use, for instance, the command
  useradd -m -u 1201 -G sales,ops linda
  to create a user linda who is a member of the groups sales and ops with UID 1201 and add a home directory to the user account as well.

Home Directories

- All normal users will have a home directory. For people, the home directory is the directory where personal files can be stored.
- For system accounts, the home directory often contains the working environment for the service account.
- If when creating user accounts you tell your server to add a home directory as well (for instance, by using **useradd -m**), the content of the “skeleton” directory is copied to the user home directory. The skeleton directory is ***/etc/skel***, and it contains files that are copied to the user home directory at the moment this directory is created. These files will also get the appropriate permissions to ensure that the new user can use and access them.

Managing User Properties

- For changing user properties, the same rules apply as for creating user accounts. You can either work directly in the configuration files using **vipw** or you can use command-line tools.
- The ultimate command-line utility for modifying user properties is usermod. It can be used to set all properties of users as stored in **/etc/passwd** and **/etc/shadow**, plus some additional tasks, such as managing group membership.
- There is just one task it does not do well: setting passwords. Although usermod has an option **-p** that tells you to “use encrypted password for the new password,” it expects you to do the password encryption before adding the user account. That does not make it particularly useful. If as root you want to change the user password, you’d better use the **passwd** command.

Configuration Files for User Management Defaults

- When working with tools as useradd, some default values are assumed. These default values are set in two configuration files: **/etc/login.defs** and **/etc/default/useradd**

Listing 6.4 Useradd Defaults in /etc/default/useradd

```
[root@localhost skel]# cat /etc/default/useradd
# useradd defaults file
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

Configuration Files for User Management Defaults

- In the file **/etc/login.defs**, different login-related variables are set. This file is used by different commands, and it relates to setting up the appropriate environment for new users. Here is a list of some of the most significant properties that can be set from /etc/login.defs:

Configuration Files for User Management Defaults

- **MOTD_FILE:** Defines the file that is used as "message of the day" file. In this file, you can include messages to be displayed after the user has successfully logged in to the server.
- **ENV_PATH:** Defines the $PATH variable, a list of directories that should be searched for executable files after logging in.
- **PASS_MAX_DAYS, PASS_MIN_DAYS**, and **PASS_WARN_AGE:** Define the default password expiration properties when creating new users.
- **UID_MIN:** The first UID to use when creating new users.
- **CREATE_HOME:** Indicates whether or not to create a home directory for new users.
- **USERGROUPS_ENAB:** Set to yes to create a private group for all new users. That means that a new user has a group with the same name as the user as its default group. If set to no, all users are made a member of the group users.

Managing Password Properties

- You learned about the password properties that can be set in /etc/shadow. You can use two commands to change these properties for users: **chage** and **passwd**.
- The commands are rather straightforward. For instance, the command **passwd -n 30 -w 3 -x 90 linda** sets the password for user linda to a minimal usage period of 30 days and an expiry after 90 days, where a warning is generated 3 days before expiry.
- Many of the tasks that can be accomplished with **passwd** can be done with **chage** also. For instance, use **chage -E 2015-12-31 bob** to have the account for user bob expire on December 31, 2015.
  To see current password management settings, use **chage –l**

Managing Password Properties

Listing 6.5 Showing Password Expiry Information with **chage -l**

```
linux:~ # chage -l linda
Last password change                                    : Apr 11, 2015
Password expires                                        : Jul 10, 2015
Password inactive                                       : never
Account expires                                         : never
Minimum number of days between password change          : 30
Maximum number of days between password change          : 90
Number of days of warning before password expir         : 3
```

Creating a User Environment

- When a user logs in, an environment is created. The environment consists of some variables that determine how the user environment is used. One such variable, for instance, is $PATH, which defines a list of directories that should be searched when a user types a command. To construct the user environment, a few files play a role:
- **/etc/profile:** Used for default settings for all users when starting a login shell
- **/etc/bashrc:** Used to define defaults for all users when starting a subshell
- **~/.profile:** Specific settings for one user applied when starting a login shell
- **~/.bashrc:** Specific settings for one user applied when starting a subshell
- When logging in, the files are read in this order, and variables and other settings that are defined in these files are applied. If a variable or setting occurs in more than one file, the last one wins.

Exercise 6.2 Creating User Accounts

In this exercise, you apply common solutions to create user accounts.

1. Type **vim /etc/login.defs** to open the configuration file /etc/login.defs and change a few parameters before you start creating logging. Look for the parameter **CREATE_HOME** and make sure it is set to "yes." Also set the parameter **USERGROUPS_ENAB** to "no," which makes that a new user is added to a group with the same name as the user and nothing else.
2. Use **cd /etc/skel** to go to the /etc/skell directory. Type **mkdir Pictures** and **mkdir Documents** to add two default directories to all user home directories. Also change the contents of the file .bashrc to include the line **export EDITOR=/usr/bin/vim**, which sets the default editor for tools that need to modify text files.
3. Type **useradd linda** to create an account for user linda. Then, type **id linda** to verify that linda is a member of a group with the name linda and nothing else. Also verify that the directories Pictures and Documents have been created in linda's home directory.

Exercise 6.2 Creating User Accounts

4. Use **passwd linda** to set a password for the user you have just created. Use the password **password**.
5. Type **passwd -n 30 -w 3 -x 90 linda** to change the password properties. This has the password expire after 90 days (**-x 90**). Three days before expiry, the user will get a warning (**-w 3**), and the password has to be used for at least 30 days before (**-n 30**) it can be changed.
6. Create a few more users: lisa, lori, and bob, using **for i in lisa lori bob; do useradd $i; done**.
7. Use **grep lori /etc/passwd /etc/shadow /etc/group**. This shows the user lori created in all three critical files and confirms they have been set up correctly.

Understanding Linux Groups

- Linux users can be a member of two different kinds of groups. First, there is the primary group. Every user must be a member of a primary group and there is only one primary group. When creating files, the primary group becomes group owner of these files.
- Users can also access all files their primary group has access to. The users primary group membership is defined in **/etc/passwd**; the group itself is stored in the **/etc/group** configuration file.

Creating Groups with *vigr*

- With the **vigr** command, you open an editor interface directly on the /etc/group configuration file.

Listing 6.6 Sample /etc/group Content

```
kvm:x:36:qemu
qemu:x:107:
libstoragemgmt:x:994:
rpc:x:32:
rpcuser:x:29:
"/etc/group.edit" 65L, 870C
```

fields are used in */etc/group*

The following fields are used in /etc/group:

- **Group name:** As is suggested by the name of the field, this contains the name of the group.
- **Group password:** A feature that is hardly used anymore. A group password can be used by users that want to join the group on a temporary basis, so that access to files the group has access to is allowed.
- **Group ID:** A unique numeric group identification number.
- **Members:** Here you find the names of users that are a member of this group as a secondary group. Note that it does not show users that are a member of this group as their primary group.

Using *groupadd* to Create Groups

- Another method to create new groups is by using the **groupadd** command. This command is easy to use. Just use **groupadd** followed by the name of the group you want to add. There are some advanced options, the only significant of them is **-g**, which allows you to specify a group ID when creating the group.

Managing Group Properties

- To manage group properties, **groupmod** is available. You can use this command to change the name or group ID of the group, but it does not allow you to add group members. To do this, you use **usermod**. As discussed before, **usermod -aG** will add users to new groups that will be used as their secondary group. Because a group does not have many properties, it is quite common that group properties are managed directly in the /etc/group file by using the **vigr** command.

TIP Because users group membership is defined in two different locations, it can be difficult to find out which groups exactly a user is a member of. A convenient command to check this is **groupmems**. Use, for example, the command **groupmems -g sales -l** to see which users are a member of the group sales. This shows users who are a member of this group as a secondary group assignment, but also users who are a member of this group as the primary group assignment.

Exercise 6.3 Working with Groups

In this exercise, you create two groups and add some users as members to these groups.

1. Type **groupadd sales** followed by **groupadd account** to add groups with the names sales and account.
2. Use **usermod** to add users linda and lisa to the group sales, and lori and bob to the group account:

```
usermod -aG sales linda
usermod -aG sales lisa
usermod -aG account lori
usermod -aG account bob
```

3. Type **id linda** to verify that user linda has correctly been added to the group sales. In the results of this command, you see that linda is assigned to the group with gid=100(users). This is her primary group. With the **groups** parameter, all groups she is a member of as secondary group are mentioned:

```
linux:~ # id linda
uid=1000(linda) gid=100(users) groups=1000(sales),100(users)
```

Logging In Through an External Authentication Service

- When a user enters his login name and password, these are normally checked on the local server. If in your environment many servers are used, this approach is not the most convenient, and you might benefit from a centralized service that helps you managing users and groups. To provide such centralized authentication services, **LDAP** is a common solution.

Understanding LDAP

- The Lightweight Directory Access Protocol (LDAP) was developed as a protocol to get information from an X.500 directory service. This service was originally developed as an address book. Currently, LDAP has developed further into a service that can be used as a centralized authentication service.
- LDAP is an open standard, and many directory services are available that are using LDAP as their access protocol. Some common LDAP solutions are OpenLDAP, or the LDAP server that is integrated in the Red Hat Identity Management solution, which is also known as FreeIPA. For the RHCSA exam, you do not need to know how to set up an LDAP server yourself, but you do need to be able to set up a client for authentication on LDAP.

Understanding LDAP

LDAP directory servers are organized in a hierarchical, distributed and replicated way:

- LDAP is hierarchical; it is organized like DNS, using domains (which in LDAP are called containers) to organize the leaf objects (such as users) in a way that makes sense.
- LDAP is distributed because the entire database does not have to be available on one single server. The different containers in the LDAP hierarchy can be spread over multiple servers to make the information available where it needs to be available. To distribute the information in the LDAP directory, the directory tree is partitioned into different parts.
- LDAP is replicated; multiple copies of one partition can be created.

To authenticate on an LDAP server, there are two options:

- Password authentication
- Kerberos authentication

Configuring RHEL 7 for LDAP Authentication

To set up RHEL7 for LDAP authentication, you need to create a configuration file that explains which LDAP server to use, which TLS certificate to use, and which container in LDAP should be used as the base LDAP URL. To specify all this, three different tools can be used:

- **authconfig:** A command-line utility in which you have to specify all you want to do by using command-line options
- **authconfig-tui:** A menu-driven text user interface that allows you to select options to be used from a list. Use of this utility is recommended
- **authconfig-gtk:** A utility with a GUI, which for that reason can be used from a GUI environment only

Depending on which tool you use, a different authentication backend is configured. The nslcd service is configured and started when using autconfig-tui. When authconfig-gtk is used, the sssd service is used as the backend.

TIP If you install the sshd package before you configure anything, you should be able to deal with authentication through the sssd service and not using the nslcd service. It is easy to find yourself in a situation where nslcd is used instead of sssd, which is why you learn about both of them in the following sections.

Managing nslcd

- When you use **authconfig-tui**, the **nslcd** service is configured on your server to connect to the LDAP service. This service ensures that your local system will look beyond the local user information and get to LDAP. The nscld service is using a configuration file with the name /etc/nslcd.conf. In this file, you find all relevant settings that are required to connect to LDAP. In Listing 6.7, you can see the contents of this configuration file.

Listing 6.7 LDAP Connection Parameters in nslcd.conf

```
[root@localhost ~]# cat /etc/nslcd.conf | grep -v ^# | grep ^[a-Z]
uid nslcd
gid ldap
uri ldap://ipa.example.com
base dc=example,dc=com
tls_reqcert never
ssl start_tls
tls_cacertdir /etc/openldap/cacerts
```

Managing nslcd

The important parameters that were added by authconfig-tui are as follows:

- **uri** This specifies the name of the LDAP server.
- **base** This is the base container where LDAP users should be looked for.
- **tls_reqcert** This option was added to allow TLS connections without a trusted certificate authority (CA).
- **ssl** This option specifies that TLS should be used for establishing trusted connections.

Managing nslcd

- After configuring your server for LDAP authentication, use **systemctl status nslcd** to verify it is running. If it is not, check whether the sssd service is used instead, as described in the following subsection.

- If the nslcd service is not running, and neither is sssd, you can start it using the **systemctl start nslcd** command. Once it is running, you can use the **systemctl status nslcd** command for troubleshooting also. This command tells you exactly what is wrong if you receive an error when connecting to the LDAP server.

Managing sssd

- If you have initialized the connection to the LDAP server using authconfig-gtk after making sure that the sssd service is installed, the configuration is written to sssd. The sssd service integrates with the local authentication procedure and redirects all authentication requests to LDAP in that case.
- When sssd is used, you should check whether the service is running by using **systemctl status sssd**. If it is, you can check the configuration in /etc/sss/sssd.conf
- Normally, there should not be a need to modify the configuration in /etc/sssd/sssd conf directly because it is written by authconfig-gtk, but for verification purposes you might want to take a look anyway. You find all LDAP-related configuration lines in this file.

Managing sssd

TIP On the exam, be prepared to work with both nslcd and sssd. Being prepared for both ensures that you will not get into trouble because you do not know what is happening. The best tool to focus on during the exam is authconfig-tui, because it does not require many resources and additional programs to work.

NOTE When you use authconfig-tui, the variable FORCELEGACY=yes is set in /etc/sysconfig/authconfig. This makes that nslcd is used instead of sssd.

Exercise 6.4 Connecting to an External LDAP Server

- This exercise assumes that you have installed an LDAP server as offered by FreeIPA. A complete lab environment is available for download at http://rhatcert.com. Make sure to register.
- The lab environment is available as a free download for registered users only. All tasks described here are performed on your test server:

Exercise 6.4 Connecting to an External LDAP Server

1. Set up hostname resolution on your server so that the LDAP server can be used by its name. In this exercise, the IP address 192.168.122.200 is used for the LDAP server. Change this according to the setup you are using and enter the following line in the /etc/hosts file:

   ```
   192.168.122.200          ipa.example.com
   ```

2. Type **yum groupinstall -y "Directory Client"**.
3. Type **scp ipa.example.com:/root/cacert.p12/etc/openldap/cacerts**.
4. As root, type **authconfig-tui**. In the text user interface that opens now, under User Information select Use LDAP, and under Authentication, select Use LDAP Authentication. Do not unselect any option that is selected by default (see Figure 6.2).

Exercise 6.4 Connecting to an External LDAP Server

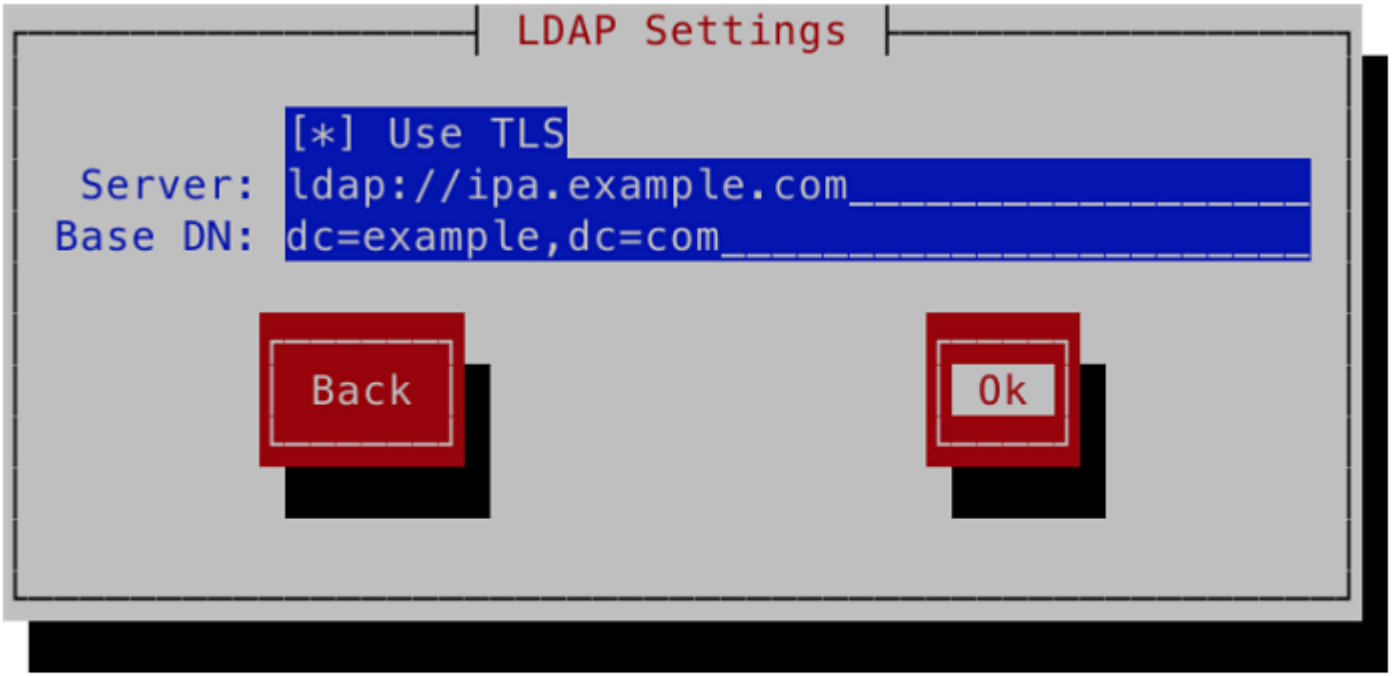

Figure 6.2 Configuring authentication from authconfig-tui.

5. In the next screen, you are prompted as to whether TLS should be used, see Figure 6.3. Select Use TLS, and then enter the server URL **ldap://ipa.example.com**. Make sure the base DN is set to dc=example,dc=com. Then click OK to continue.

Exercise 6.4 Connecting to an External LDAP Server

6. You now see a message indicating that you need to copy the certificate of the CA that has signed the public key certificate of the LDAP server to /etc/openldap/cacerts. You have already done this in step 3 of this exercise, so no action is required here.
7. Open the file /etc/nslcd.conf and make sure that it contains the line tls_reqcert never. Use **systemctl restart nslcd** to restart the nslcd service.

Exercise 6.4 Connecting to an External LDAP Server

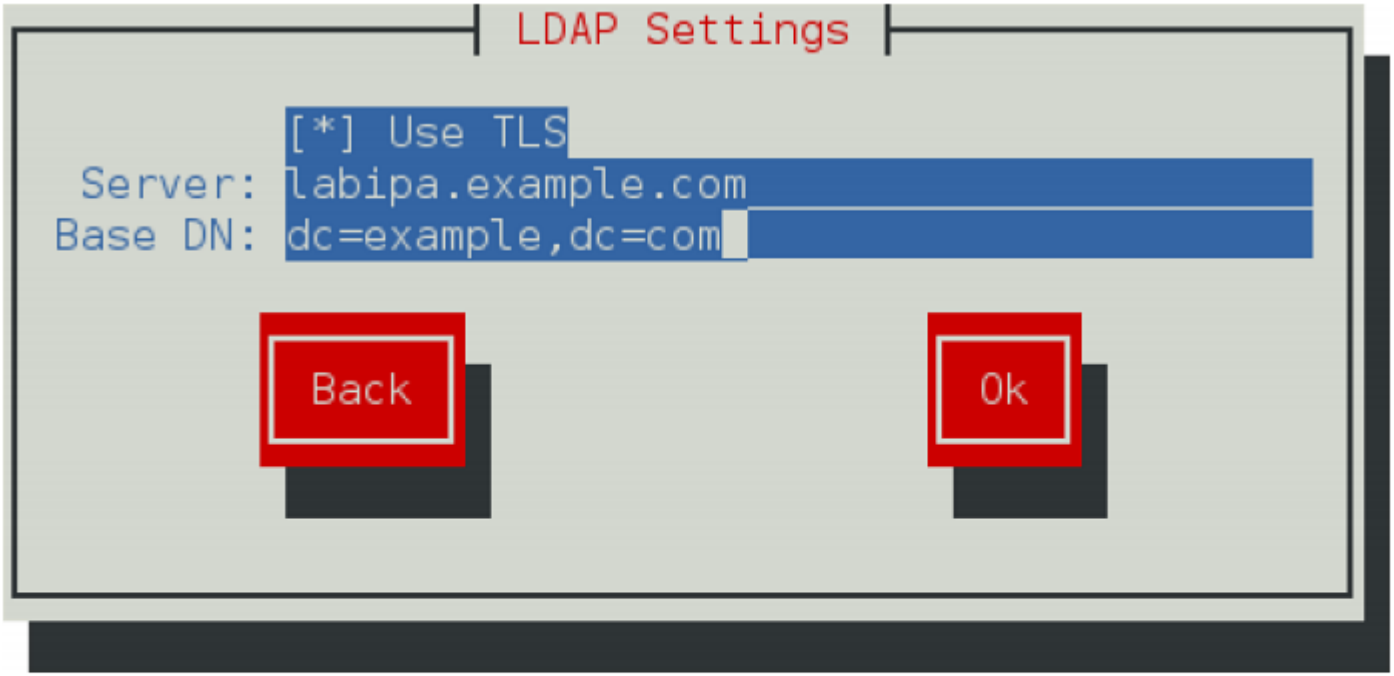

Figure 6.3 Configuring TLS settings.

8. Use **su - lara** to verify that you now have access to users on the IPA server. On the IPA server, a user with the name lara has been created. You notice that you are logged in as this user, but you also get a warning that the home directory could not be set. For the moment, you can ignore that warning.

Summary

- In this chapter, you learned how to create users and groups. You learned which configuration files are used to store users and groups, and you learned which properties are used in these files. You also learned which utilities are available to manage user and group accounts.

Define Key Terms

- Define the following key terms:

  - user,
  - password,
  - GECOS,
  - group,
  - primary group,
  - secondary group,
  - privileged user,
  - unprivileged user,
  - root,
  - LDAP

Lab 6.1

1. Set up a shared group environment that meets the following requirements:
   - Create two groups: sales and account.
   - Create users bob, betty, bill, and beatrix. Make sure they have their primary group set to a private group that has the name of the user.
   - Make bob and betty a member of the group sales, and bill and beatrix a member of the group account.
   - Set a password policy that requires users to change their password every 90 days.

Lab 6.2

1. Configure your server for authentication on an LDAP server. You can use the LDAP server that is available from the test environment at http://www.rhatcert.com, or set up an LDAP server as described in Appendix D of this book. After configuring LDAP login, try logging in as an LDAP user. The LDAP server on rhatcert.com has users linda, lisa, and lara preconfigured, which you can use for testing.

Chapter 7:

Configuring Permissions

Chapter 7 Objectives

- The following topics are covered in this chapter:
  - Managing File Ownership
  - Managing Basic Permissions
  - Managing Advanced Permissions
  - Managing ACLs
  - Setting Default Permissions with umask
  - Working with User Extended Attributes

- The following RHCSA exam objectives are covered in this chapter:
  - List, set, and change standard UGO/rwx permissions
  - Create and configure set-GID directories for collaboration
  - Create and manage access control lists
  - Diagnose and correct file permissions problems

Managing File Ownership

- Before discussing permissions, you must know about the role of file and directory ownership. File and directory ownership is vital for working with permissions. In this section, you first learn how you can see ownership. Then you learn how to change user and group ownership for files and directories.

Displaying Ownership

- On Linux, every file and every directory has two owners: a user and a group owner. These owners are set when a file or directory is created. On creation, the user who creates the file becomes the user owner, and the primary group of that user becomes the group owner.

Listing 7.1 Displaying Current File Ownership

```
[root@server1 home]# ls -l
total 8
drwx------.  3 bob      bob        74 Feb  6 10:13 bob
drwx------.  3 caroline caroline   74 Feb  6 10:13 caroline
drwx------.  3 fozia    fozia      74 Feb  6 10:13 fozia
drwx------.  3 lara     lara       74 Feb  6 10:13 lara
drwx------.  5 lisa     lisa     4096 Feb  6 10:12 lisa
drwx------. 14 user     user     4096 Feb  5 10:35 user
```

Displaying Ownership

Using the **ls** command, you can display ownership for files in a given directory. It may on occasion be useful to get a list of all files on the system that have a given user or group as owner. To do this, you may use **find**. The **find** argument **-user** can be used for this purpose. For instance, the following command shows all files that have user linda as their owner:

```
find / -user linda
```

You can also use **find** to search for files that have a specific group as their owner. For instance, the following command searches all files that are owned by the group users:

```
find / -group users
```

Changing User Ownership

To apply appropriate permissions, the first thing to consider is ownership. To do this, there is the **chown** command. The syntax of this command is not hard to understand:

```
chown who what
```

For instance, the following command changees ownership for the file account to user linda:

```
chown linda account
```

The **chown** command has a few options, of which one is particularly useful: **-R**. You might guess what it does, because this option is available for many other commands as well. It allows you to set ownership recursively, which allows you to set ownership of the current directory and everything below. The following command changes ownership for the directory /home and everything beneath it to user linda:

```
chown -R linda /home/linda
```

Changing Group Ownership

There are actually two ways to change group ownership. You can do it using **chown**, but there is also a specific command with the name **chgrp** that does the job. If you want to use the **chown** command, use a **.** or **:** in front of the group name. The following changes the group owner of directory /home/account to the group account:

```
chown .account /home/account
```

Changing Group Ownership

You can use **chown** to change user and/or group ownership in a number of ways, an overview of which follows:

- **chown lisa myfile** Sets user lisa as the owner of myfile
- **chown lisa.sales myfile** Sets user lisa as user owner and group sales as group owner of myfile
- **chown lisa:sales myfile** Sets user lisa as user owner and group sales as group owner of myfile
- **chown .sales myfile** Sets group sales as group owner of myfile without changing the user owner
- **chown :sales myfile** Sets group sales as group owner of myfile without changing the user owner

Changing Group Ownership

You can also use the **chgrp** command to change group ownership. Consider the following example, where you can use **chgrp** to set group ownership for the directory /home/account to the group account:

```
chgrp account /home/account
```

As is the case for **chown**, you can use the option **-R** with **chgrp** as well to change group ownership recursively.

Understanding Default Ownership

You might have noticed that when a user creates a file, default ownership is applied. The user who creates the file automatically becomes user owner, and the primary group of that user automatically becomes group owner. Normally, this is the group that is set in the **/etc/passwd** file as the user's primary group. If the user is a member of more groups, however, he can change the effective primary group.

To show the current effective primary group, a user can use the **groups** command:

```
[root@server1 ~]# groups lisa
lisa : lisa account sales
```

If the current user linda wants to change the effective primary group, she can use the **newgrp** command, followed by the name of the group she wants to set as the new effective primary group. This group will be continued to be used as effective primary group until the user uses the **exit** command or logs out. Listing 7.2 shows how user linda uses this command to make sales her effective primary group.

Understanding Default Ownership

Listing 7.2 Using **newgrp** to Change the Effective Primary Group

```
[lisa@server1 ~]$ groups
lisa account sales
[lisa@server1 ~]$ newgrp sales
[lisa@server1 ~]$ groups
sales lisa account
[lisa@server1 ~]$ touch file1
[lisa@server1 ~]$ ls -l
total 0
-rw-r--r--. 1 lisa sales 0 Feb  6 10:06 file1
```

After you change the effective primary group, all new files that the user creates will get this group as their group owner. To return to the original primary group setting, use **exit**.

To be able to use the **newgrp** command, a user has to be a member of that group. Alternatively, a group password can be set for the group using the **gpasswd** command. If a user uses the **newgrp** command but is not a member of the target group, the shell prompts for the group password. After you enter the correct group password, the new effective primary group is set.

Managing Basic Permissions

- The Linux permissions system was invented in the 1970s. Because computing needs were limited in those years, the basic permission system that was created was rather limited. This basic permission system uses three permissions that can be applied to files and directories. In this section, you learn how the system works and how to modify these permissions.

Understanding Read, Write, and Execute Permissions

Table 7.2 Use of Read, Write, and Execute Permissions

Permission	Applied to Files	Applied to Directories
Read	Open a file	List contents of directory
Write	Change contents of a file	Create and delete files and modify permissions on files
Execute	Run a program file	Change to the directory

Applying Read, Write, and Execute Permissions

To apply permissions, you use the **chmod** command. When using **chmod**, you can set permissions for user, group, and others. You can use this command in two modes: the relative mode and the absolute mode. In absolute mode, three digits are used to set the basic permissions. Table 7.3 provides an overview of the permissions and their numeric representation.

Table 7.3 Numeric Representation of Permissions

Permission	Numeric Representation
Read	4
Write	2
Execute	1

Applying Read, Write, and Execute Permissions

When setting permissions, calculate the value that you need. If you want to set read, write, and execute for the user, read and execute for the group, and read and execute for others on the file /somefile, for example, you use the following **chmod** command:

```
chmod 755 /somefile
```

When you use **chmod** in this way, all current permissions are replaced by the permissions you set. If you want to modify permissions relative to the current permissions, you can use **chmod** in relative mode. When using **chmod** in relative mode, you work with three indicators to specify what you want to do:

- First, you specify for whom you want to change permissions. To do this, you can choose between user (**u**), group (**g**), and others (**o**).
- Then, you use an operator to add or remove permissions from the current mode, or set them in an absolute way.
- At the end, you use **r**, **w**, and **x** to specify what permissions you want to set.

Applying Read, Write, and Execute Permissions

When changing permissions in relative mode, you may omit the "to whom" part to add or remove a permission for all entities. For instance, the following adds the execute permission for all users:

```
chmod +x somefile
```

When working in relative mode, you may use more complex commands as well. For instance, the following adds the write permission to the group and remove read for others:

```
chmod g+w,o-r somefile
```

TIP When using **chmod -R o+rx /data**, you set the execute permission on all directories as well as files in the /data directory. To set the execute permission to directories only, and not to files, use **chmod -R o+rX /data**. The uppercase X ensures that files will not get the execute permission unless the file has already set the execute permission for some of the entities. That makes X the more intelligent way of dealing with execute permissions; it will avoid setting that permission on files where it is not needed.

Exercise 7.1 Managing Basic Permissions

In this exercise, you create a directory structure for the groups that you have created earlier. You also assign the correct permissions to these directories.

1. From a root shell, type **mkdir -p /data/sales /data/account**.
2. Before setting the permissions, change the owners of these directories using **chown linda.sales /data/sales and chown linda.account /data/account**.
3. Set the permissions to enable the user and group owners to write files to these directories, and deny all access for all others: **chmod 770 /data/sales, and next chmod 770 /data/account**.
4. Use **su - lisa** to become lisa and change into the directory /data/account. Use **touch emptyfile** to create a file in this directory. Does this work?
5. Still as lisa, use **cd /data/sales** and use **touch emptyfile** to create a file in this directory. Does this work?

Setting Default Permissions with umask

You have probably noticed that when creating a new file, some default permissions are set. These permissions are determined by the umask setting. This shell setting is applied to all users when logging in to the system. In the umask setting, a numeric value is used that is subtracted from the maximum permissions that can be set automatically to a file; the maximum setting for files is 666, and for directories is 777. Some exceptions apply to this rule, however. You can find a complete overview of umask settings in Table 7.5.

Of the digits used in the umask, like with the numeric arguments for the **chmod** command, the first digit refers to user permissions, the second digit refers to the group permissions, and the last refers to default permissions set for others. The default umask setting of 022 gives 644 for all new files and 755 for all new directories that are created on your server. A complete overview of all umask numeric values and their result is shown in Table 7.5.

Setting Default Permissions with umask

Table 7.5 umask Values and Their Result

Value	Applied to Files	Applied to Directories
0	Read and write	Everything
1	Read and write	Read and write
2	Read	Read and execute
3	Read	Read
4	Write	Write and execute
5	Write	Write
6	Nothing	Execute
7	Nothing	Nothing

Setting Default Permissions with umask

An easy way to see how the umask setting works is as follows: Start with the default permissions for a file set to 666 and subtract the umask to get the effective permissions. Do the same for a directory and its default permissions 777.

There are two ways to change the umask setting: for all users and for individual users. If you want to set the umask for all users, you must make sure the umask setting is considered when starting the shell environment files as directed by /etc/profile. The right approach is to create a shell script with the name umask.sh in the /etc/profile.d directory and specify the umask you want to use in that shell script. If the umask is changed in this file, it applies to all users after logging in to your server.

An alternative to setting the umask through /etc/profile and related files where it is applied to all users logging in to the system is to change the umask settings in a file with the name .profile, which is created in the home directory of an individual user. Settings applied in this file are applied for the individual user only; therefore, this is a nice method if you need more granularity. I personally like this feature to change the default umask for user root to 027, whereas normal users work with the default umask 022.

Summary

- In this chapter, you learned how to work with permissions. You read about the three basic permissions, the advanced permissions, and how to apply on the file system.
- You also learned how to use the umask setting to apply default permissions.

Define Key Terms

- Define the following key terms:
  - ownership
  - permissions

Chapter 8:

Configuring Networking

Chapter 8 Objectives

- The following topics are covered in this chapter:
  - Networking Fundamentals
  - Managing Network Addresses and Interfaces
  - Validating Network Configuration
  - Configuring Network Configuration with nmtui and nmcli
  - Working on Network Configuration Files
  - Setting Up Hostname and Name Resolution

- The following RHCSA exam objectives are covered in this chapter:
  - Configure networking and hostname resolution statically or dynamically

Networking Fundamentals

To set up networking on a server, your server needs a unique address on the network. For this purpose, IP (Internet Protocol) addresses are used. Currently, two versions of IP addresses are relevant:

- **IPv4 addresses:** These are based on 32-bit addresses and look like 192.168.10.100.
- **IPv6 addresses:** These are based on 128-bit addresses and are written in eight hexadecimal written parts that are based on 16 bits each. An IPv6 address may look like fe80:badb:abe01:45bc:34ad:6723:8798.

Binary Notation

Because the number of IPv4 addresses is limited, in modern IPv4 networks variable network masks are used. These are network masks such as 212.209.113.33/27. In a variable subnet mask, only a part of the byte is used for addressing nodes, and another part is used for addressing the network. In the subnet mask /27, the first 3 bits of the last byte are used to address the network, and the last 5 bits are used for addressing nodes. This becomes a bit clearer if you write down the address in a binary notation:

IP address:

```
212.209.113.33 = 11010100.11010001.00001010.00100001
```

Subnet mask:

```
/27            = 11111111.11111111.11111111.11100000
```

MAC Addresses

IP addresses are the addresses that allow nodes to communicate to any other node on the Internet. They are not the only addresses in use though. Each network card also has an address, which is known as the MAC address. MAC addresses are for use on the local network (that is, the local cable or local WLAN, just up to the first router that is encountered); they cannot be used for communications between nodes that are on different networks. They are important, though, because MAC addresses help computers find the specific network card that an IP address belongs to.

Protocol and Ports

IP addresses are used to address nodes, and that is useful, but addressing nodes is not what it is all about. Nodes are useful because they offer specific services on the network, such as a web server or a mail server. To identify these services, port addresses are used. Every service has a specific port address, such as port 80 for Hypertext Transfer Protocol (HTTP) or port 22 for a Secure Shell (SSH) server, and in network communication, the sender and the receiver are using port addresses. So, there is a destination port address as well as a source port address involved in network communications.

Because not all services are addressed in a similar way, a specific protocol is used between the IP address and the port address, such as Transfer Control Protocol (TCP), User Datagram Protocol (UDP), or Internet Control Message Protocol (ICMP). Every protocol has specific properties: TCP is typically used when the network communication must be reliable and delivery must be guaranteed; UDP is used when it must be fast and guaranteed delivery does not count.

Managing Network Addresses and Interfaces

As a Linux server administrator, you need to manage network addresses and network interfaces. The network addresses can be assigned in two ways:

- **Fixed IP addresses:** Useful for servers that always need to be available at the same IP address.
- **Dynamically assigned IP addresses:** Useful for end-users devices, and for instances in a cloud environment. To dynamically assign IP addresses, a Dynamic Host Configuration Protocol (DHCP) server is usually used.

For a long time, network cards in Linux have had default names, such as eth0, eth1, and eth2. This naming was assigned based on the order of detection of the network card. So, eth0 was the first network card that got detected, eth1 the second, and so on. This worked well in an environment where a node has one or two network cards only. If a node has multiple network cards that need to be dynamically added and removed, however, this approach does not work so well anymore.

Managing Network Addresses and Interfaces

In RHEL 7, the default names for network cards are based on firmware, device topology, and device types. This leads to network card names that always consist of the following parts:

- Ethernet interfaces begin with en, WLAN interfaces begin with wl, and WWAN interfaces begin with ww.
- The next part of the name represents the type of adapter. An o is used for onboard, s is for a hotplug slot, and p is for a PCI location. Administrators can also use the x to create a device name that is based on the MAC address of the network card.
- Then follows a number, which is used to represent an index, ID, or port.
- If the fixed name cannot be determined, traditional names such as eth0 are used.

Based on this information, device names such as eno16777734 can be used, which stands for an onboard Ethernet device, with its unique index number.

Validating Network Configuration

Before you can learn how to set network information, you must know how to verify current network information. In this section, you learn how to do that, and you learn how to check the following networking items:

- IP address and subnet mask
- Routing
- Availability of ports and services

Validating Network Configuration

To verify the configuration of the network address, you need to use the **ip** utility. The **ip** utility is a modern utility that can consider advanced networking features that have been introduced recently. With the **ip** utility, many aspects of networking can be monitored:

- Use **ip addr** to configure and monitor network addresses
- Use **ip route** to configure and monitor routing information
- Use **ip link** to configure and monitor network link state

Apart from these items, the **ip** utility can manage many other aspects of networking, but you do not need to know about them for the RHCSA exam.

WARNING In earlier Linux versions, and some other UNIX-like operating systems, the **ifconfig** utility was and is used for validating network configuration. Do not use this utility on modern Linux distributions. Because Linux has become an important player in cloud computing, networking has evolved a lot to match cloud computing requirements, and many new features have been added to Linux networking. With the **ifconfig** utility, you cannot manage or validate these concepts.

Validating Network Configuration

To show current network settings, you can use the **ip addr show** command (which can be abbreviated as **ip a s** or even as **ip a**). The **ip** command is relatively smart and does not always require you to type the complete option.

The result of the **ip addr show** command looks as in Listing 8.1

Listing 8.1 Monitoring Current Network Configuration with **ip addr show**

```
[root@server2 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    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: eno16777736: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
  pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:b8:8c:eb brd ff:ff:ff:ff:ff:ff
```

Validating Network Configuration

If you are just interested in the link state of the network interfaces, you can use the **ip link show** command. This command (of which you can see the output in Listing 8.2) repeats the link state information of the **ip addr show** command.

Listing 8.2 ip link show Output

```
[root@server2 ~]# ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eno16777736: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_
fast state UP mode DEFAULT qlen 1000
    link/ether 00:0c:29:b8:8c:eb brd ff:ff:ff:ff:ff:ff
```

In case the **ip link show** command shows the current link state as down, you can temporarily bring it up again by using **ip link set**, which is followed by **dev devicename** and **up** (for example, **ip link set dev eno16777736 up**).

Validating Routing

One important aspect of networking is routing. On every network that needs to communicate to nodes on other networks, routing is a requirement. Every network has, at least, a default router (also called the default gateway) that is set, and you can see which router is used as the default router by using the command **ip route show** (see Listing 8.3). You should always perform one quick check to verify that your router is set correctly: the default router at all times must be on the same network as the local IP address that your network card is using.

Listing 8.3 ip route show Output

```
[root@server2 ~]# ip route show
default via 192.168.4.2 dev eno16777736  proto static  metric 1024
192.168.4.0/24 dev eno16777736  proto kernel  scope link  src
192.168.4.220
```

Validating the Availability of Ports and Services

Network problems can be related to the local IP and router settings but can also be related to network ports that are not available on your server or on a remote server. To verify availability of ports on your server, you can use the **netstat** command, or the newer **ss** command, which provides the same functionality. Example 8.2 shows how to verify network settings. By typing **ss -lt**, you'll see all listening TCP ports on the local system (see Listing 8.4).

Validating the Availability of Ports and Services

Listing 8.4 Use **ss -lt** to Display All Listening Ports on the Local System

```
[root@server2 ~]# ss -lt
State       Recv-Q Send-Q   Local Address:Port      Peer Address:Port
LISTEN      0      100      127.0.0.1:smtp          *:*
LISTEN      0      128      *:56601                 *:*
LISTEN      0      128      .0.0.1:x11-ssh-offse    *:*
LISTEN      0      128      *:sunrpc                *:*
LISTEN      0      128      *:ssh                   *:*
LISTEN      0      128      127.0.0.1:ipp           *:*
LISTEN      0      100      ::1:smtp                :::=*
LISTEN      0      128      :1:x11-ssh-offset       ::: *
LISTEN      0      128      :::sunrpc               :::*
LISTEN      0      128      :::34449                :::*
LISTEN      0      128      :::ssh                  :::*
LISTEN      0      128      ::1:ipp                 :::*
```

Exercise 8.2 Verifying Network Settings

1. Open a root shell to your server and type **ip addr show**. This shows the current network configuration. Note the IPv4 address that is used. Notice the network device names that are used; you need these later in this exercise.
2. Type **ip route show** to verify routing configuration.
3. If your computer is connected to the Internet, you can now use the **ping** command to verify the connection to the Internet is working properly. Type **ping -c 4 8.8.8.8**, for instance, to send four packets to IP address 8.8.8.8. If your Internet connection is up and running, you should get "echo reply" answers.
4. Type **ip addr add 10.0.0.10/24 dev <yourdevicename>**.
5. Type **ip addr show**. You'll see the newly set IP address, in addition to the IP address that was already in use.
6. Type **ifconfig**. Notice that you do not see the newly set IP address (and there are no options with the **ifconfig** command that allow you to see it). This is one example why you should not use the **ifconfig** command anymore.
7. Type **ss -tul**. You'll now see a list of all UDP and TCP ports that are listening on your server.

Configuring Network Configuration with nmtui and nmcli

As mentioned earlier in this chapter, networking on RHEL 7 is managed by the NetworkManager service. You can use the **systemctl status NetworkManager** command to verify its current status. When NetworkManager comes up, it reads the network card configuration scripts, which are in /etc/sysconfig/network-scripts and have a name that starts with ifcfg and is followed by the name of the network card.

When working with network configuration in RHEL 7, you should know the difference between a device and a connection:

- A device is a network interface card.
- A connection is the configuration that is used on a device.

In RHEL 7, you can create multiple connections for a device. This can make sense on mobile computers, to make a difference between settings that are used while connected to the home network and settings that are needed to the corporate network. Switching between connections on devices is something that is common on end-user computers, and not so common on servers. To manage the network connections that you want to assign to devices, you use the **nmtui** or the **nmcli** command.

Configuring the Network with nmcli

Earlier in this chapter, you learned how to use **ip** to verify network configuration. You have also applied the **ip addr add** command to temporarily set an IP address on a network interface. Everything you do with the **ip** command, though, is nonpersistent. If you want to make your configuration persistent, use **nmtui** or **nmcli**.

A good start is to use **nmcli** to show all connections. This shows active *and* inactive connections. You can easily see the difference because inactive connections are not currently assigned to a device (see Listing 8.5)

Listing 8.5 Showing Current Connection Status

```
[root@server2 ~]# nmcli con show
NAME  UUID                                  TYPE            DEVICE
eth0  3b2a6a84-ea82-45d1-a173-6674e770c096  802-3-ethernet  eno16777736
```

After finding the name of the connection, you can use **nmcli con show** followed by the name of the connection to see all properties of the connection. Notice that this command shows many properties.

Configuring the Network with nmtui

If you do not like the complicated syntax of the nmcli command line, you might like **nmtui**. This is a text user interface that allows you to create network connections easily. Figure 8.1 shows what the nmtui interface looks like.

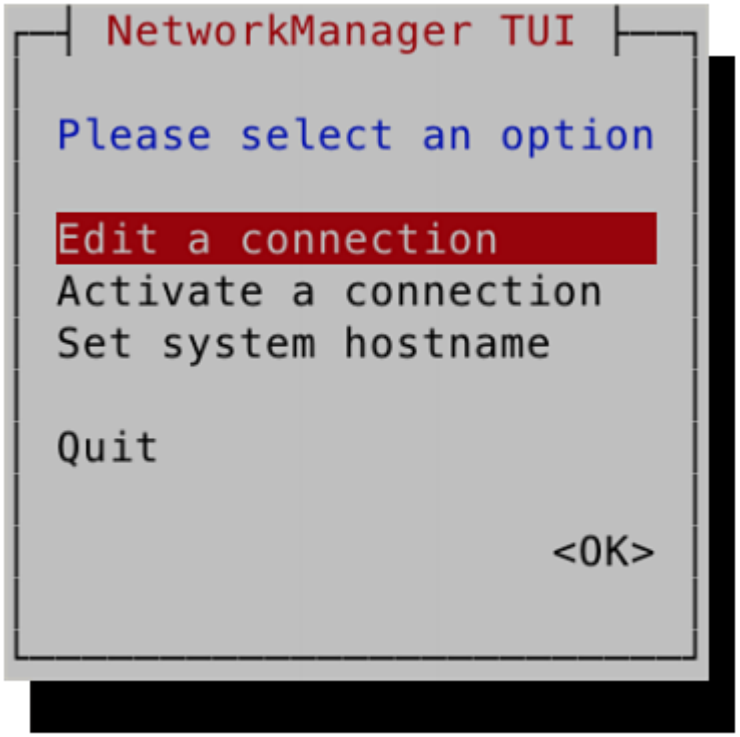

Figure 8.1 The nmtui interface.

Configuring the Network with nmtui

The nmtui interface consists of three menu options:

- **Edit a Connection:** Use this option to create new connections or edit existing connections.
- **Activate a Connection:** Use this to (re)activate a connection.
- **Set System Hostname:** Use this to set the hostname of your computer.

TIP If you like graphical user interface (GUI) tools, you are lucky. Use nm-connection-editor instead of nmtui, but be prepared that this interface offers a relatively restricted option set. It does not contain advanced options such as the options to create network team interfaces and manage network bridge interfaces. It does, however, offer all you need to manage address configuration on a network connection. Start it by using the **nm-connection-editor** command, or by using the applet in the GNOME graphical interface. Figure 8.2 shows what the default interface of this tool looks like.

Working on Network Configuration Files

Every connection that you create is stored as a configuration file in the directory /etc/sysconfig/network-scripts. The name of the configuration files starts with ifcfg- and is followed by the name of the network interface. In Listing 8.7, you can see what such a configuration file looks like.

Listing 8.7 Example of an ifcfg Configuration File

```
[root@server2 network-scripts]# cat ifcfg-eno16777736
TYPE="Ethernet"
BOOTPROTO=none
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
IPV6INIT="yes"
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="yes"
IPV6_FAILURE_FATAL="no"
NAME=eno16777736
UUID="3b2a6a84-ea82-45d1-a173-6674e770c096"
ONBOOT="yes"
IPADDR0=192.168.4.220
PREFIX0=24
```

Working on Network Configuration Files

```
GATEWAY0=192.168.4.2
DNS1=192.168.4.200
HWADDR=00:0C:29:B8:8C:EB
DNS2=8.8.8.8
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
```

Normally, there should be no need to modify these configuration files manually. If you want to, though, you can. After making changes to the configuration file, use the **nmcli con reload** command to activate the new configuration.

TIP You can set both a fixed IP address as a dynamic IP address in one network connection. To do that, set the BOOTPROTO option in the connection configuration file to dhcp, while you also specify an IP address and network prefix. You can do this also from the nmtui utility; just make sure that in nmtui the IPv4 configuration is set to automatic (and not to manual), and specify an IP address as well. I recommend that you to do this in the test configuration you are using with this book, because it allows you to use a static network address configuration for internal use, in addition to a dynamic configuration that allows you to access the Internet and install software from repositories.

Hostnames

Because hostnames are used to access servers and the services they're offering, it is important to know how to set the system hostname. A hostname typically consists of different parts. These are the name of the host and the DNS domain in which the host resides. These two parts together make up for the fully qualified domain name (FQDN), which looks like server1.example.com. It is good practice to always specify an FQDN, and not just the hostname. There are different ways to change the hostname:

- Use nmtui and select the option **Change Hostname**.
- Use **hostnamectl set-hostname**.
- Edit the contents of the configuration file /etc/hostname.

To configure the hostname with **hostnamectl**, you can use a command like **hostnamectl set-hostname myhost.example.com**. After setting the hostname, you can use **hostnamectl status** to show the current hostname. Listing 8.8 shows the output of this command.

Hostnames

Listing 8.8 Showing Current Hostname Configuration

```
[root@server2 ~]# hostnamectl status
   Static hostname: server2.example.com
         Icon name: computer
           Chassis: n/a
        Machine ID: 708ab34cdfca454d908224e0b37a8bf6
           Boot ID: 9fa3baf6fe46420aaa44c324a76f40b2
    Virtualization: vmware
  Operating System: CentOS Linux 7 (Core)
       CPE OS Name: cpe:/o:centos:centos:7
            Kernel: Linux 3.10.0-123.el7.x86_64
      Architecture: x86_64
```

The **hostnamectl** command is new in RHEL 7. When using **hostnamectl status**, you see not only information about the hostname but also information about the Linux kernel, virtualization type, and much more.

Hostnames

Alternatively, you can set the hostname using the nmtui interface. Figure 8.3 shows the screen from which this can be done.

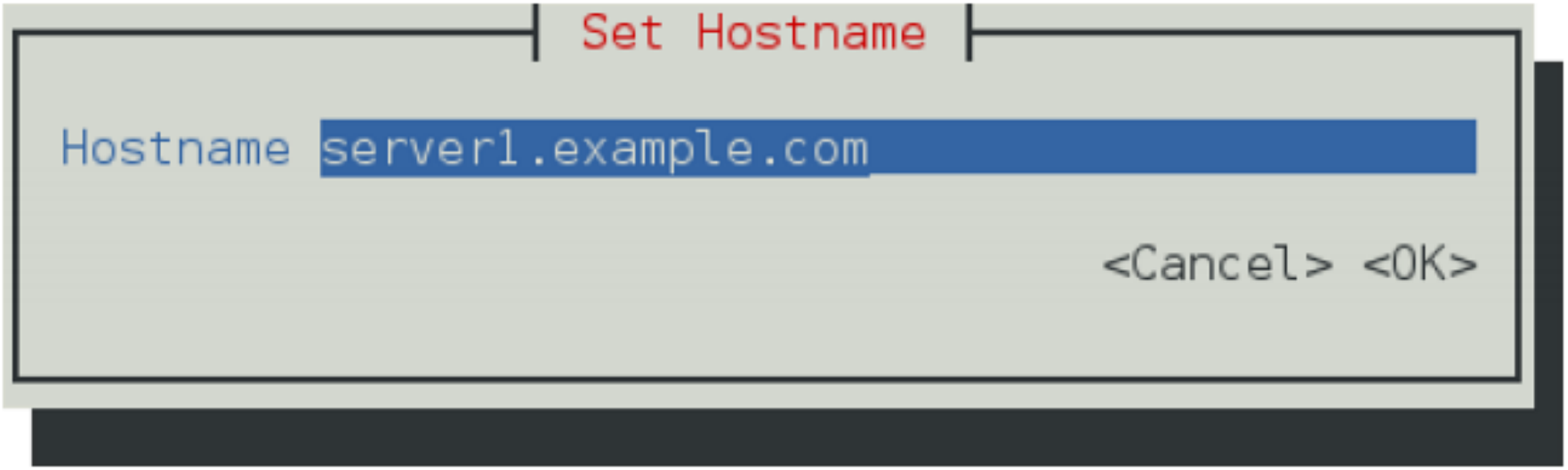

Figure 8.3 Changing the hostname using nmtui.

To set host name resolution, DNS is typically used. Configuring DNS is not an RHCSA objective. Apart from DNS, you can configure host name resolution in the /etc/hosts file. Listing 8.9 shows the contents of an /etc/hosts file.

Hostnames

Listing 8.9 /etc/hosts Sample Contents

```
[root@server1 ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

All hostname - IP address definitions as set in /etc/hosts will be applied before the hostname in DNS is used. This is configured as a default in the hosts line in /etc/nsswitch.conf, which by default looks like this:

```
hosts:    files dns
```

Setting up an /etc/hosts file is easy; just make sure that it contains at least two columns. The first column has the IP address of the specific host, and the second column specifies the hostname. The hostname can be provided as a short name (like server1), or an FQDN. In an FQDN, the hostname as well as the complete DNS name are included, as in server1.example.com.

Hostnames

If a host has more than one name, like a short name and a fully qualified DNS name, you can specify both of them in /etc/hosts. In that case, the second column must contain the FQDN, and the third column can contain the alias. Listing 8.10 shows a hostname configuration example.

Listing 8.10 /etc/hosts Configuration Example

```
[root@server2 ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.0.0.10     server1.example.com     server1
10.0.0.20     server2.example.com     server2
```

DNS Resolving

It is recommended to always set up at least two DNS name servers to be contacted. If the first name server does not answer, the second name server is contacted. So, this is why you want to use a second name server. If the first name server times out or cannot be reached, the second server is used. To specify which DNS name servers you want to use, you have a few different options:

- Use **nmtui** to set the DNS name servers. Figure 8.4 shows the interface from which you can do this.
- Set the DNS1 and DNS2 in the ifcfg network connection configuration file in /etc/sysconfig/network-scripts.
- Use a DHCP server that is configured to hand out the address of the DNS name server.
- Use **nmcli con mod <connection-id> [+]ipv4.dns <ip-of-dns>**.

DNS Resolving

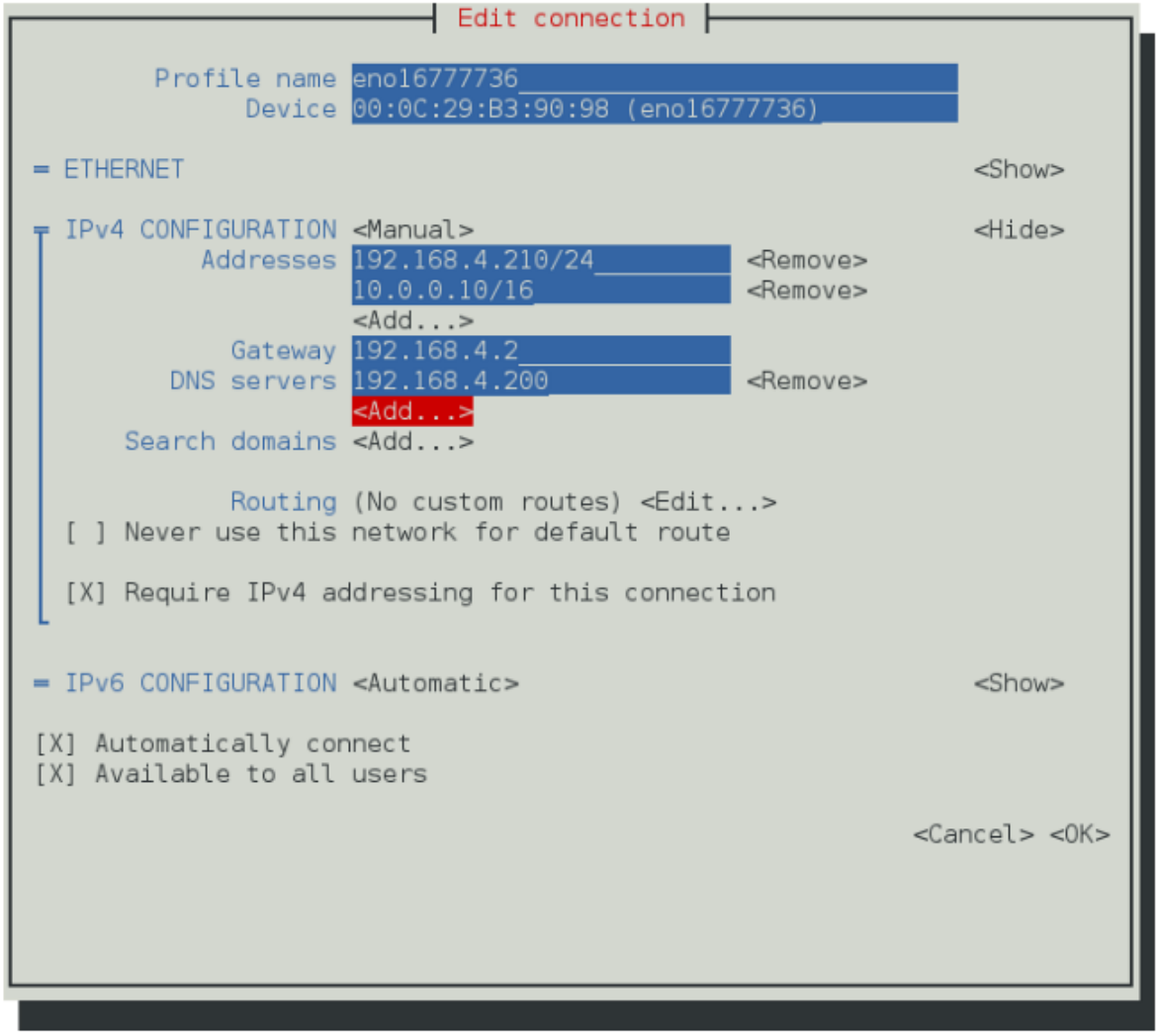

Figure 8.4 Setting DNS servers from the nmtui interface

DNS Resolving

Notice that if your computer is configured to get the network configuration from a DHCP server, the DNS server is also set via the DHCP server. If you do not want this to happen, you have two options:

- Edit the ifcfg configuration file to include the option **PEERDNS=no**.
- Use **nmcli con mod <con-name> ipv4.ignore-auto-dns yes**.

To verify host name resolution, you can use the **getent hosts <servername>** command. This command searches in both /etc/hosts and DNS to resolve the hostname that has been specified.

EXAM TIP Do *not* specify the DNS servers directly in /etc/resolv.conf. They will be overwritten by NetworkManager.

Summary

In this chapter, you learned how to configure networking in RHEL 7. First you read how the IP protocol is used to connect computers together, and then you read which techniques are used to make services between hosts accessible. Next you read how to verify the network configuration using the **ip** utility and some related utilities. In the last part of this chapter, you read how to set IP addresses and other host configuration in a permanent way by using either the nmcli or the nmtui utility.

Define Key Terms

- Define the following key terms:
  - ip,
  - ipv4,
  - ipv6,
  - protocol,
  - port,
  - subnet mask,
  - DNS,
  - DHCP,
  - connection,
  - interface,
  - FQDN

Lab 8.1

1. Set up the first server to use the FQDN server1.example.com. The second server should use server2.example.com.
2. On server1.example.com, use nmtui and configure your primary network card to automatically get an IP address through DHCP. Also set a fixed IP address to 192.168.122.100. On server2, set the fixed IP address to 192.168.122.110.
3. Make sure that from server1 you can ping server2, and vice versa.
4. To allow you to access servers on the Internet, make sure that your local DHCP server provides the default router and DNS servers.

Chapter 9:

Process Management

Chapter 9 Objectives

- The following topics are covered in this chapter:
  - Introduction to Process Management
  - Managing Shell Jobs
  - Using Common Command-Line Tools for Process Management
  - Using top to Manage Processes

- The following RHCSA exam objectives are covered in this chapter:
  - Identify CPU/memory-intensive processes, adjust process priority with renice, and kill processes

Introduction to Process Management

For everything that happens on a Linux server, a process is started. For that reason, process management is among the key skills that an administrator has to master. To do this efficiently, it is important to know which type of process you are dealing with. A major distinction can be made between two process types:

- Shell jobs are commands started from the command line. They are associated with the shell that was current when the process was started. Shell jobs are also referred to as interactive processes.
- Daemons are processes that provide services. They normally are started when a computer is booted and often (but certainly not in all cases) they are running with root privileges.

When a user types a command, a shell job is started. If no particular measures have been taken, the job is started as a foreground process, occupying the terminal it was started from until it has finished its work. As a Linux administrator, you need to know how to start shell jobs in the foreground or background and what can be done to manage shell jobs.

Running Jobs in the Foreground and Background

By default, any executed command is started as a foreground job. For many commands, that does not really matter because the command often takes a little while to complete, after which it returns access to the shell from which it was started. Sometimes it might prove useful to start commands in the background. This makes sense for processes that do not require user interaction. A process that does require user interaction will not be able to get that when running in the background, and for that reason will typically stall when moved to the background. You can take two different approaches to run a process in the background.

If you know that a job will take a long time to complete, you can start it with an **&** behind it. This immediately starts the job in the background to make room for other tasks to be started from the command line. To move the last job that was started in the background back as a foreground job, use the **fg** command. This command immediately, and with no further questions, brings the last job back to the foreground.

Running Jobs in the Foreground and Background

A job might sometimes have been started that takes (much) longer than predicted. If that happens, you can use **Ctrl+Z** to temporarily stop the job. This does not remove the job from memory; it just pauses the job so that it can be managed. Once paused, it can be continued as a background job using the **bg** command. An alternative key sequence that you can use to manage shell jobs is **Ctrl+C**. This stops the current job and removes it from memory.

A related keystroke combination is **Ctrl+D**, which sends the End Of File (EOF) character to the current job. The result is that the job stops waiting for further input so that it can complete what it was currently doing. The result of sending **Ctrl+D** is sometimes very similar to the result of sending **Ctrl+C**, but there is a difference. When **Ctrl+C** is used, the job is just canceled, and nothing is closed properly. When **Ctrl+D** is used, the job stops waiting for further input, which often is just what is needed to complete in a proper way.

Managing Shell Jobs

When moving jobs between the foreground and background, it may be useful to have an overview of all current jobs. To get such an overview, use the **jobs** command. As you can see in Table 9.2, this command gives an overview of all jobs currently running as a background job, including the job number assigned to the job when starting it in the background. These job numbers can be used as an argument to the **fg** and **bg** commands to perform job management tasks. In Exercise 9.1, you learn how to perform common job management tasks from the shell.

Table 9.2 Job Management Overview

Command	Use
& (used at the end of a command line)	Starts the command immediately in the background.
Ctrl+Z	Stops the job temporarily so that it can be managed. For instance, it can be moved to the background.
Ctrl+D	Send the End Of File (EOF) character to the current job to indicate that it should stop waiting for further input.

Managing Shell Jobs

Command	Use
Ctrl+C	Can be used to cancel the current interactive job
bg	Continues the job that has just been frozen using **Ctrl+Z** in the background.
fg	Brings the last job that was moved to background execution back to the foreground.
jobs	Shows which jobs are currently running from this shell. Displays job numbers that can be used as an argument to the commands **bg** and **fg**.

Exercise 9.1 Managing jobs

In this exercise, you apply the commands that you just learned about to manage jobs that have been started from the current shell.

1. Open a root shell and type the following commands:

```
sleep 3600 &
dd if=/dev/zero of=/dev/null &
sleep 7200
```

2. Because you started the last command with no & after the command, you have to wait 2 hours before you get control to the shell back. Type **Ctrl+Z** to stop it.
3. Type **jobs**. You will see the three jobs that you just started. The first two of them have the Running state, and the last job currently is in the Stopped state.

Exercise 9.1 Managing jobs

4. Type **bg 3** to continue running job 3 in the background. Notice that because it was started as the last job, you did not really have to add the number 3.
5. Type **fg 1** to move job 1 to the foreground.
6. Type **Ctrl+C** to cancel job number 1 and use **jobs** to confirm that it is now gone.
7. Use the same approach to cancel jobs 2 and 3 also.
8. Open a second terminal on your server.
9. From that second terminal, type **dd if=/dev/zero of=/dev/null &**.
10. Type **exit** to close the second terminal.
11. From the other terminal, start **top**. You will see that the **dd** job is still running. From top, use **k** to kill the dd job.

NOTE You read how to manage interactive shell jobs in this section. Notice that all of these jobs are processes as well. As the user that started the job, you can also manage it. In the next section, you learn how to use process management to manage jobs started by other users.

Managing Parent Child Relations

When a process is started from a shell, it becomes a child process of that shell. In process management, the parent-child relationship between processes is very important. The parent is needed to manage the child. For that reason, all processes started from a shell are terminated when that shell is stopped. This also offers an easy way to terminate processes no longer needed.

Processes started in the background will not be killed when the parent shell from which they were started is killed. To terminate these processes, you need to use the **kill** command, as described later in this chapter.

NOTE In earlier versions of the bash shell, background processes were also killed when the shell they were started from was terminated. To prevent that, the process could be started with the **nohup** command in front of it. Using **nohup** for this purpose is no longer needed in RHEL 7.

Common CLI Tools for Process Management

On a Linux server, many processes are usually running. On an average server or desktop computer, there are often more than a hundred active processes. With so many processes being active, things may go wrong. If that happens, it is good to know how noninteractive processes can be stopped, or how the priority of these processes can be adjusted to make more system resources available for other processes.

Understanding Processes and Threads

Tasks on Linux are typically started as processes. One process can start several worker threads. Working with threads makes sense, because if the process is very busy, the threads can be handled by different CPUs or CPU cores available in the machine. As a Linux administrator, you cannot manage individual threads; you can manage processes, though. It is the programmer of the multithreaded application that has to define how threads relate to one another.

Before talking about different ways to manage processes, it is good to know that there are two different types of background processes. To start, there are kernel threads. These are a part of the Linux kernel, and each of them is started with its own process identification number (PID). When managing processes, it is easy to recognize the kernel processes because they have a name that is between square brackets. Listing 9.1 shows a list of a few processes as output of the command **ps aux | head** (discussed later in this chapter), in which you can see a couple of kernel threads.

Understanding Processes and Threads

Listing 9.1 Showing Kernel Threads with **ps aux**

```
[root@server1 /]# ps aux | head
USER        PID %CPU %MEM    VSZ   RSS TTY   STAT START   TIME COMMAND
root          1  0.0  0.4  52984  4272 ?     Ss   Feb05   0:03 /usr/lib/
    systemd/systemd --switched-root --system --deserialize 23
root          2  0.0  0.0      0     0 ?     S    Feb05   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?     S    Feb05   0:00 [ksoftirqd/0]
root          5  0.0  0.0      0     0 ?     S<   Feb05   0:00 [kworker/0:0H]
root          7  0.0  0.0      0     0 ?     S    Feb05   0:00 [migration/0]
root          8  0.0  0.0      0     0 ?     S    Feb05   0:00 [rcu_bh]
root          9  0.0  0.0      0     0 ?     S    Feb05   0:00 [rcuob/0]
root         10  0.0  0.0      0     0 ?     S    Feb05   0:00 [rcuob/1]
root         11  0.0  0.0      0     0 ?     S    Feb05   0:00 [rcuob/2]
```

Using *ps* to Get Process Information

The most common command to get an overview of currently running processes is **ps**. If used without any arguments, the **ps** command shows only those processes that have been started by the current user. You can use many different options to display different process properties. If you are looking for a short summary of the active processes, use **ps aux** (as you saw in Listing 9.1). If you are not only looking for the name of the process but also for the exact command that was used to start the process, use **ps -ef**. Alternative ways to use **ps** exist as well, such as the command **ps fax**, which shows hierarchical relationships between parent and child processes (see Listings 9.2 and 9.3).

Using *ps* to Get Process Information

Listing 9.2 Use **ps -ef** to See the Exact Command Used to Start Processes

```
[root@server2 ~]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root       874     1  0 May14 ?        00:00:00 /sbin/auditd -n
root       885   874  0 May14 ?        00:00:00 /sbin/audispd
root       889   885  0 May14 ?        00:00:00 /usr/sbin/sedispatch
root       898     1  0 May14 ?        00:00:00 /usr/sbin/alsactl -s -n
19 -c -E ALSA_CONFIG_PATH=/etc/alsa/alsactl.conf --initfile=/lib/alsa/
root       899     1  0 May14 ?        00:00:00 /usr/sbin/bluetoothd -n
root       900     1  0 May14 ?        00:00:00 /usr/bin/python -Es /
usr/sbin/firewalld --nofork --nopid
avahi      902     1  0 May14 ?        00:00:00 avahi-daemon: running
[server2.local]
libstor+   903     1  0 May14 ?        00:00:00 /usr/bin/lsmd -d
root       905     1  0 May14 ?        00:02:14 /usr/bin/vmtoolsd
root       908     1  0 May14 ?        00:00:00 /usr/sbin/rsyslogd -n
root       910     1  0 May14 ?        00:00:00 /usr/sbin/abrtd -d -s
root       911     1  0 May14 ?        00:00:00 /usr/bin/abrt-watch-log
-F BUG: WARNING: at WARNING: CPU: INFO: possible recursive locking det
avahi      912   902  0 May14 ?        00:00:00 avahi-daemon: chroot
helper
```

Using *ps* to Get Process Information

NOTE For some commands, using a hyphen before options is optional. Some commands do not. The **ps** command is one of those latter commands. There is a historical reason why this is the case: The commands derive from the old BSD UNIX flavor, where it was common to specify command-line options without a - in front of them.

Listing 9.3 Use **ps fax** to Show Parent-Child Relationships Between Processes

```
[root@server2 ~]# ps fax
  PID TTY      STAT   TIME COMMAND
1603 ?        Ss     0:00 /usr/sbin/sshd -D
35395 ?       Ss     0:00  \_ sshd: root@pts/1
35417 pts/1   Ss     0:00   \_ -bash
35568 pts/1   R+     0:00   \_ ps fax
 1612 ?       Ss     0:00 /usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf
 1613 ?       Ssl    0:07 /usr/sbin/nslcd
 1652 ?       Ss     0:01 /usr/sbin/nmbd
 1689 ?       Ss     0:02 /usr/sbin/smbd
```

Using *ps* to Get Process Information

An important piece of information to get out of the **ps** command is the PID. Many tasks require the PID to operate, and that is why a command like **ps aux | grep dd**, which will show process details about dd, including its PID, is quite common. An alternative way to get the same result is to use the **pgrep** command. Use **pgrep dd** to get a list of all PIDs that have a name containing the string dd.

Adjusting Process Priority with *nice*

When Linux processes are started, they are started with a specific priority. By default, all regular processes are equal and are started with the same priority, which is the priority number 20. In some cases, it is useful to change the default priority that was assigned to the process when it was started. You can do that using the **nice** and **renice** commands. Use **nice** if you want to start a process with an adjusted priority. Use **renice** to change the priority for a currently active process. Alternatively, you can use the **r** command from the **top** utility to change the priority of a currently running process.

When using **nice** or **renice** to adjust process priority, you can select from values ranging from -20 to 19. The default niceness of a process is set to 0 (which results in the priority value of 20). By applying a negative niceness, you increase the priority. Use a positive niceness to decrease the priority. It is a good idea not to use the ultimate values immediately. Instead, use increments of 5 and see how it affects the application.

Adjusting Process Priority with *nice*

Let's take a look at examples of how to use **nice** and **renice**. The command **nice -n 5 dd if=/dev/zero of=/dev/null &** starts an infinite I/O-intensive job, but with an adjusted niceness so that some place remains for other processes as well. To adjust the niceness of a currently running process, you need the PID of that process. The following two commands show how **ps aux** is used to find the PID of the dd job from the previous example. Next, you see how the **renice** command is used to change the niceness of that command:

1. Use **ps aux | grep dd** to find the PID of the dd command that you just started. The PID is in the second column of the command output.
2. Use **renice -n 10 -p 1234** (assuming that 1234 is the PID you just found).

Note that regular users can only decrease the priority of a running process. You must be root to give processes increased priority.

Sending Signals to Processes with *kill*, *killall*, and *pkill*

Before starting to think about using the **kill** command or sending other signals to processes, it is good to know that Linux processes have a hierarchical relationship. Every process has a parent process, and as long as it lives, the parent process is responsible for the child processes it has created. This is particularly important when processes are terminated, because it is the parent that has to clean up the resources that were used by the children. When using kill on a parent process that still has active children, you will for that reason not just kill the parent process in question but also all of its currently active child processes.

Sending Signals to Processes with *kill*, *killall*, and *pkill*

The Linux kernel allows many signals to be sent to processes. Use **man 7 signals** for a complete overview of all the available signals. Three of these signals work for all processes:

- The signal SIGTERM (15) is used to ask a process to stop.
- The signal SIGKILL (9) is used to force a process to stop.
- The SIGHUP (1) signal is used to hang up a process. The effect is that the process will reread its configuration files, which makes this a useful signal to use after making modifications to a process configuration file.

To send a signal to a process, the **kill** command is used. The most common use is the need to stop a process, which you can do by using the **kill** command followed by the PID of the process. This sends the SIGTERM signal to the process, which normally causes the process to cease its activity.

Sending Signals to Processes with *kill*, *killall*, and *pkill*

Sometimes the **kill** command does not work because the process you want to kill is busy. In that case, you can use **kill -9** to send the SIGKILL signal to the process. Because the SIGKILL signal cannot be ignored, it forces the process to stop, but you also risk losing data while using this command. In general, it is a bad idea to use **kill -9**:

- You risk losing data.
- Your system may become unstable if other processes depend on the process you have just killed.

TIP Use **kill -l** to show a list of available signals that can be used with **kill**.

There are some commands that are related to kill: **killall** and **pkill**. The **pkill** command is a bit easier to use because it takes the name rather than the PID of the process as an argument. You can use the **killall** command if multiple processes using the same name need to be killed simultaneously.

Exercise 9.2 Managing Processes from the CLI

In this exercise, you learn how to work with ps, nice, kill, and related utilities to manage processes.

1. Open a root shell. From this shell, type **dd if=/dev/zero of=/dev/null &**. Repeat this command three times.
2. Type **ps aux | grep dd**. This shows all lines of output that have the letters dd in them; you will see more than just the dd processes, but that should not really matter. The processes you just started are listed last.
3. Use the PID of one of the dd processes to adjust the niceness, using **renice -n 5 <PID>**. Notice that in top you cannot easily get an overview of processes and their current priority.
4. Type **ps fax | grep -B5 dd**. The **-B5** option shows the matching lines, including the five lines before that. Because **ps fax** shows hierarchical relationships between processes, you should also find the shell and its PID from which all the dd processes were started.
5. Find the PID of the shell from which the dd processes were started and type **kill -9 <PID>**, replacing **<PID>** with the PID of the shell you just found. You will see that your root shell is closed, and with it, all of the dd processes. Killing a parent process is an easy and convenient way to kill all of its child processes also.

Using *top* to Manage Processes

root@server1:~

File Edit View Search Terminal Help

```
top - 16:02:38 up 16:05,  3 users,  load average: 2.13, 2.08, 1.92
Tasks: 392 total,   4 running, 387 sleeping,   1 stopped,   0 zombie
%Cpu(s): 10.4 us, 79.6 sy, 10.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   1010880 total,   934768 used,    76112 free,       40 buffers
KiB Swap:  1048572 total,   296580 used,   751992 free.   450752 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
17449 root      20   0  107920    624    532 R 100.0  0.1  92:52.18 dd
18666 root      25   5  107920    624    532 R  98.0  0.1  31:58.40 dd
 3300 user      20   0 1614116 151584  18028 S   1.0 15.0   1:34.06 gnome-shell
19150 root      20   0  123792   1836   1152 S   0.7  0.2   0:00.09 top
 1066 root      20   0  197196  17740   2036 S   0.3  1.8   0:05.29 Xorg
    1 root      20   0   52984   4272   2176 S   0.0  0.4   0:04.04 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.08 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.51 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      rt   0       0      0      0 S   0.0  0.0   0:00.18 migration/0
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/0
   10 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/1
   11 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/2
   12 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/3
   13 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/4
   14 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/5
   15 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/6
   16 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/7
   17 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/8
   18 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/9
   19 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/10
   20 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/11
   21 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/12
   22 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/13
   23 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/14
   24 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/15
   25 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/16
```

Figure 9.1 Using top makes process management easy.

Using *top* to Manage Processes

Table 9.3 Linux Process States Overview

State	Meaning
Running (R)	The process is currently active and using CPU time, or in the queue of runnable processes waiting to get services.
Sleeping (S)	The process is waiting for an event to complete.
Uninterruptable sleep (D)	The process is in a sleep state that cannot be stopped. This usually happens while a process is waiting for I/O.
Stopped (S)	The process has been stopped, which typically has happened to an interactive shell process, using the **Ctrl+Z** key sequence.
Zombie (Z)	The process has been stopped but could not be removed by its parent, which has put it in an unmanageable state.

Using *top* to Manage Processes

Now that you know how to use the **kill** and **nice** commands from the command line, using the same functionality from top is even easier. From top, type **k**. top will then prompt for the PID of the process you want to send a signal to. By default, the most active process is selected. After you enter the PID, top asks which signal you want to send. By default, signal 15 for SIGTERM is used. However, if you want to insist a bit more, you can type **9** for SIGKILL. Now press **Enter** to terminate the process.

To renice a running process from top, type **r**. You are first prompted for the PID of the process you want to renice. After entering the PID, you are prompted for the nice value you want to use. Enter a positive value to increase process priority or a negative value to decrease process priority.

Summary

Managing processes is a common task for a Linux system administrator. In this chapter, you learned how to look up specific processes and how to change their priority using nice and kill.

Define Key Terms

- Define the following key terms:

  - job,
  - process,
  - background,
  - foreground,
  - nice,
  - kill,
  - signal,
  - PID,
  - thread

Lab 9.1

1. Launch the command **dd if=/dev/zero of=/dev/null** three times as a background job.
2. Increase the priority of one of these commands using the nice value -5. Change the priority of the same process again, but use this time the value -15. Observe the difference.
3. Kill all the dd processes you just started.

Chapter 10:

Working with Virtual Machines

Chapter 10 Objectives

- The following topics are covered in this chapter:
  - Understanding RHEL 7 Virtualization
  - Making Your Server a KVM Host
  - Managing Virtual Machines

- The following RHCSA exam objectives are covered in this chapter:
  - Access a virtual machine’s console
  - Start and stop virtual machines
  - Configure a physical machine to host virtual guests
  - Install Red Hat Enterprise Linux systems as virtual guests
  - Configure systems to launch virtual machines at boot

Understanding RHEL 7 Virtualization

Red Hat Enterprise Linux is an important platform for virtualization. Since RHEL 6, Red Hat has been using KVM as the default virtualization solution. This section provides an overview of different Red Hat virtualization solutions as well as components that are used in a KVM virtualization environment.

Understanding KVM Virtualization

Different virtualization solutions are available on RHEL 7. The default virtualization solution, though, is KVM (Kernel Virtual Machine). KVM is included in the Linux kernel, and the solution offers hypervisor-based virtualization. That means that you do not have to run a specific program to host VMs; instead, virtualization support is inside the operating system kernel.

KVM virtualization is not supported by default on every RHEL 7 server; you'll have to install it separately on a server that meets the minimal requirements, as described later in this chapter. KVM virtualization can be used only on 64-bit computer architecture.

If you are used to using a desktop virtualization solution, such as VMware Workstation or Oracle Virtual Box, you need to be aware of one important difference between desktop-based virtualization and hypervisor-based virtualization. In desktop-based virtualization, the VMs are provided by the virtualization application. As a result, if you shut down the virtualization application, the VMs running within it shut down as well.

Understanding QEMU

While installing a Red Hat Enterprise Linux 7 server as a KVM hypervisor host, you also automatically install some QEMU components. QEMU (Quick Emulator) is open source software that was originally created to offer hardware virtualization through binary emulation. QEMU can be used by itself, but it is also used together with KVM to run VMs on near-native speed. Some important parts on KVM hypervisor hosts are not provided by KVM itself but are instead integrated from the QEMU project. An example of this is the disk format of image files in VMs, but many other parts come from QEMU as well.

Red Hat Beyond KVM

The Red Hat product offering goes way beyond just KVM on individual hypervisors. Red Hat also has the Red Hat Enterprise Virtualization (RHEV) product in its portfolio. This solution was developed to compete with VMware vSphere environments. It offers an infrastructural server that consists of multiple hypervisor nodes that are managed from a central RHEV Manager node, on which a web user interface makes managing a complex environment easy.

Red Hat also puts a lot of effort in showing itself as a major cloud provider. Red Hat is an important contributor to the OpenStack cloud project and offers its own cloud solution that is based on OpenStack. In that solution, KVM virtualization is also offered.

Understanding the Role of *Libvirtd*

To access VMs that are offered through KVM, you use libvirtd. Libvirtd is a process that sits between the virtualization layer and the application that an administrator is using to access the VMs. Without libvirtd, you cannot manage VMs. Virtual machine management options also are configured through the libvirtd configuration file /etc/libvirt/libvirtd.conf. In this file, you can set some advanced parameters. For instance, you can open a TCP port that allows you to connect to a libvirt process that is running on another host. This is convenient, because in a multihost environment it allows you to manage VMs that are not just running on your server but also VMs that are running on other hosts as well.

Understanding the Role of *Libvirtd*

Although the method to connect remotely directly to libvirtd works well and is convenient for hypervisors that have a minimal number of software packages installed, alternatively you can use Secure Shell (SSH) from virt-manager to connect to remote libvirtd processes.

If you are experiencing problems accessing VMs, libvirtd is the primary suspect, and you should at least ensure that it is running, by using the **systemctl status -l libvirtd** command. In Listing 10.1, you can see how this command shows that libvirtd is running; the command also shows information about recent activity on the process.

Understanding the Role of *Libvirtd*

Listing 10.1 Using **systemctl status -l libvirtd** to Get Details About the libvirtd Process Status

```
[root@lab ~]# systemctl status -l libvirtd
libvirtd.service - Virtualization daemon
   Loaded: loaded (/usr/lib/systemd/system/libvirtd.service; enabled)
   Active: active (running) since Mon 2015-03-09 16:39:45 EDT;
  2 weeks 4 days ago
 Main PID: 1538 (libvirtd)
   CGroup: /system.slice/libvirtd.service
           ├─1538 /usr/sbin/libvirtd
           └─2914 /sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/
  default.conf

Mar 28 10:23:12 lab.sandervanvugt.nl dnsmasq-dhcp[2914]:
  DHCPACK(virbr0) 192.168.122.13
52:54:00:41:45:35 ubuntu
Mar 28 10:37:30 lab.sandervanvugt.nl libvirtd[1538]: stream aborted at
  client request
Mar 28 10:38:49 lab.sandervanvugt.nl dnsmasq-dhcp[2914]:
DHCPREQUEST(virbr0) 192.168.122.30
52:54:00:f5:19:bf
```

Understanding the Role of *Libvirtd*

Several management utilities can be used on top of libvirtd. The Virtual Machine Manager (offered through the virt-manager binary) is a commonly used graphical user interface (GUI) to manage KVM. Alternatively, the **virsh** command is available as a shell interface to manage KVM VMs. In following sections in this chapter, you learn how to use both.

Making Your Server a KVM Host

In this section, you learn what is needed to make your server a KVM host. You first learn about the requirements for doing so. You then learn how to install the required KVM software packages. In the last part of this section, you learn how networking is enhanced to support for VM networking.

Checking Host Requirements

Before starting to configure a RHEL server as a hypervisor host, verify support. There really are just two requirements to start KVM hypervisor backend services:

- You must be using a 64-bit CPU architecture.
- Your CPU must support hardware virtualization.

To verify whether you are running 64-bit architecture is not difficult; just type **arch** to show the architecture that is currently used. If you are good, you'll see the architecture x86_64 being displayed. Alternatively, you can use the **uname -i** command, which also shows which type of kernel is used.

Checking the availability of virtualization support on your CPU can be a bit more complicated. To start, you need to make sure that virtualization support is switched on in the BIOS of your computer. After you have done that, you can use the command **cat /proc/cpuinfo**. In the output of this command (see Listing 10.2), you should see **vmx** on an intel CPU, and **svm** on an AMD CPU.

Checking Host Requirements

Listing 10.2 Partial Contents of /proc/cpuinfo

```
processor               : 1
vendor_id               : GenuineIntel
cpu family              : 6
model                   : 58
model name              : Intel(R) Core(TM) i7-3740QM CPU @ 2.70GHz
stepping                : 9
microcode               : 0x15
cpu MHz                 : 2693.276
```

Checking Host Requirements

```
cache size              : 6144 KB
physical id             : 2
siblings                : 1
core id                 : 0
cpu cores               : 1
apicid                  : 2
initial apicid          : 2
fpu                     : yes
fpu_exception           : yes
cpuid level             : 13
wp                      : yes
flags                   : fpu vme de pse tsc msr pae mce cx8 apic sep
  mtrr pge mca cmov pat pse36 clflush
dts mmx fxsr sse sse2 ss syscall nx rdtscp lm constant_tsc
  arch_perfmon pebs bts nopl xtopology
tsc_reliable nonstop_tsc aperfmperf eagerfpu pni pclmulqdq vmx ssse3
  cx16 pcid sse4_1 sse4_2
x2apic popcnt aes xsave avx f16c rdrand hypervisor lahf_lm ida arat
  epb xsaveopt pln pts dtherm
```

Installing the KVM Software

After verifying that you meet all hardware requirements, you can install the virtualization software. The most convenient way of doing so is by using **yum groupinstall "Virtualization Host"**. This command installs everything you need to set up a virtualization host environment.

Another important consideration is the availability of storage. When you are installing a VM, it needs to create a virtual disk. This virtual disk by default is stored in an image file in the directory /var/lib/libvirt/images. Make sure that you have enough available disk space in the partition where you want to install the VMs before starting the installation.

NOTE As an alternative to working with disk image files, you may prefer working with LVM logical volumes as the storage backend. This offers some advantages, such as the possibility to easily create backups by using the LVM snapshot feature. For this reason, LVM is often used as a storage backend in production KVM servers. For the RHCSA and RHCE exams, you do not need to know how to set this up.

Exercise 10.1 Setting Up Your Server as a KVM Hypervisor Host

In this exercise, you set up your server as a KVM hypervisor host. For optimal performance, it works best if you perform this exercise on a physical server. For study purposes, you can also set up a KVM or VMware VM as a hypervisor host. That has you using virtualization within virtualization, which is bad for performance, but at least allows you to see how it works. To do this, in your virtualization program check the CPU features and make sure that hypervisor options are enabled. Currently, KVM and VMware VMs support nested virtualization (a hypervisor host in a VM). VirtualBox does not support this kind of setup.

1. On the server that you want to use as hypervisor host, type **arch** to verify that you are using a 64-bit CPU architecture.
2. Next type **cat /proc/cpuinfo | egrep 'svm|vmx'** and read the CPU flags section to see whether either vmx or svm is listed as one of the flags.
3. Type **yum groupinstall "Virtualization Host"** to install everything that is needed to make your server a virtualization host.

Understanding KVM Host Networking

After installing the virtualization software on a host computer, networking also changes significantly. On the host, a virtual bridge is created. This virtual bridge works like an embedded switch, and it is used to connect one or more of the physical network interfaces in the host to the different VMs.

While communicating on the network, a VM sends out packets through its internal (virtual) network interface, which typically has the name eth0. At the hypervisor level, this network is represented by a vnet interface. The first VM that starts gets the interface vnet0, the second machine that starts gets vnet1, and so on.

This vnet interfaces on their turn connect to the virtual bridge. The virtual bridge itself is connected to the physical network interface on your host.

Understanding KVM Host Networking

Listing 10.3 Displaying Bridging Configuration with **brctl show**

```
[root@lab ~]# brctl show
    bridge name     bridge id               STP enabled     interfaces
    virbr0          8000.fe5400414535       yes             vnet0
                                                            vnet1
                                                            vnet2
                                                            vnet3
```

Managing Virtual Machines

After installing the required software packages, you can move on and start creating and managing VMs. In this section, you learn how.

Installing Virtual Machines

In Exercise 10.2, you learn how to install a VM using the Virtual Machine Manager. After the exercise, a small section explains where you can find the VM configuration that has been written to disk.

Exercise 10.2 Installing a Virtual Machine

In this exercise, you learn how to install a VM. Before starting this exercise, make sure that an installation disk containing RHEL or CentOS 7 is available. The easiest way to do so is by inserting a physical CD-ROM into the CD drive of the KVM host.

1. Open a root shell. From the root shell, type **lsmod | grep kvm**. You are looking for the **kvm** and **kvm_intel** module (**kvm_amd** in case you are using an AMD platform). If these modules are not currently loaded, type **modprobe -r kvm** to load them.
2. Type **systemctl status libvirtd**. This command checks to see whether the libvirtd service is currently loaded. If it is, you are good. If not, type **systemctl start libvirtd** to start the service.
3. Type **df -h** to verify the amount of available disk space.
4. Type **virt-manager &** to start the Virtual Machine Manager.
5. In Virtual Machine Manager, click **Create a New Virtual Machine**. This opens the step 1 of 5 window of the New VM Wizard (see Figure 10.1)

Exercise 10.2 Installing a Virtual Machine

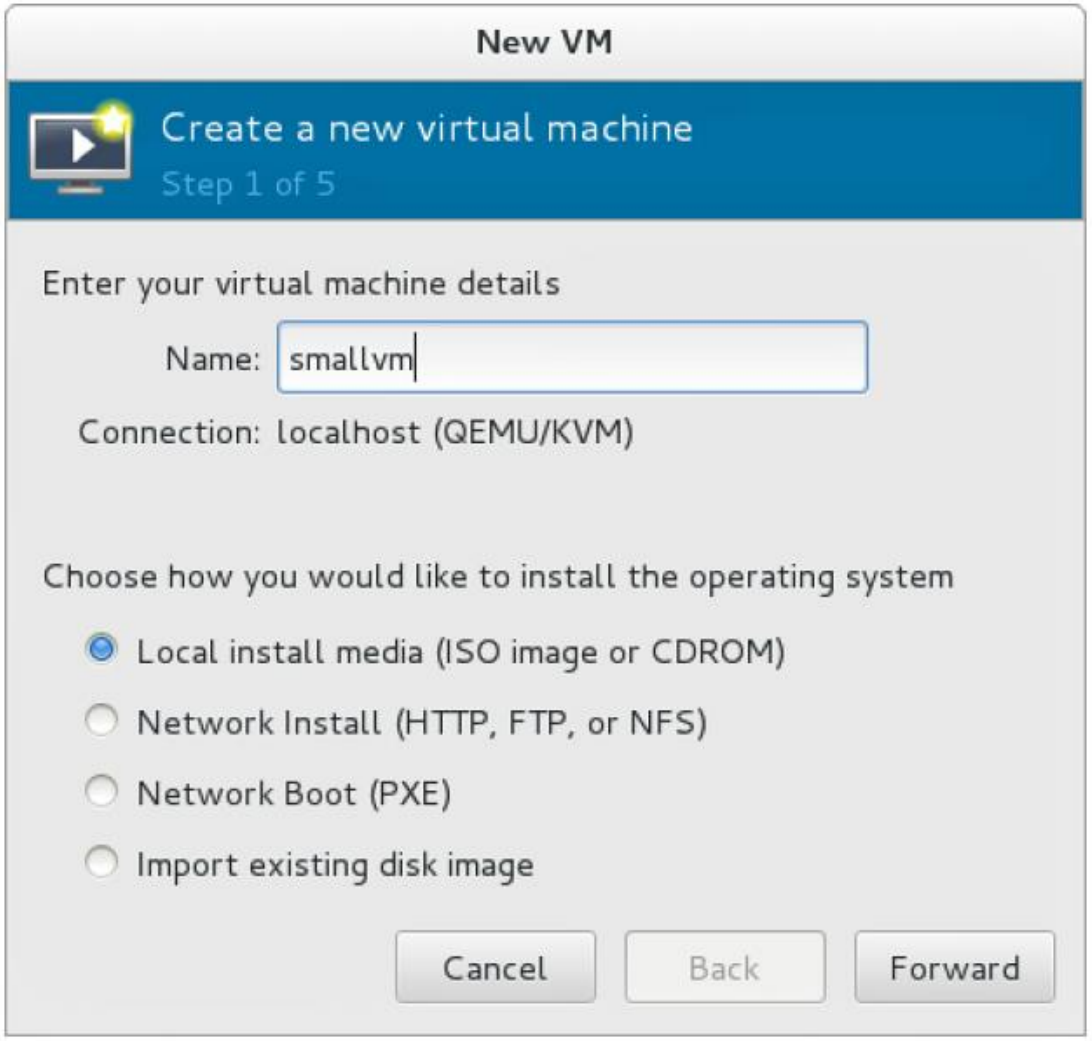

Figure 10.1 Step 1 of 5 of the Create a New Virtual Machine Wizard.

Exercise 10.2 Installing a Virtual Machine

6. In step 2 of 5 of the wizard, you need to specify the source where the installation disk can be found, as well as the operating system and operating system version you want to install. Make sure to use the same settings as in Figure 10.2.

Exercise 10.2 Installing a Virtual Machine

Figure 10.2 Specifying the installation source and OS settings.

Exercise 10.2 Installing a Virtual Machine

7. Next you need to specify the amount of RAM you want to dedicate to the VM, as well as the number of CPU cores. Of course, you need to have at least the resources you allocate in the host machine. For a minimal installation, use 512 MB RAM and 1 CPU (see Figure 10.3).

Exercise 10.2 Installing a Virtual Machine

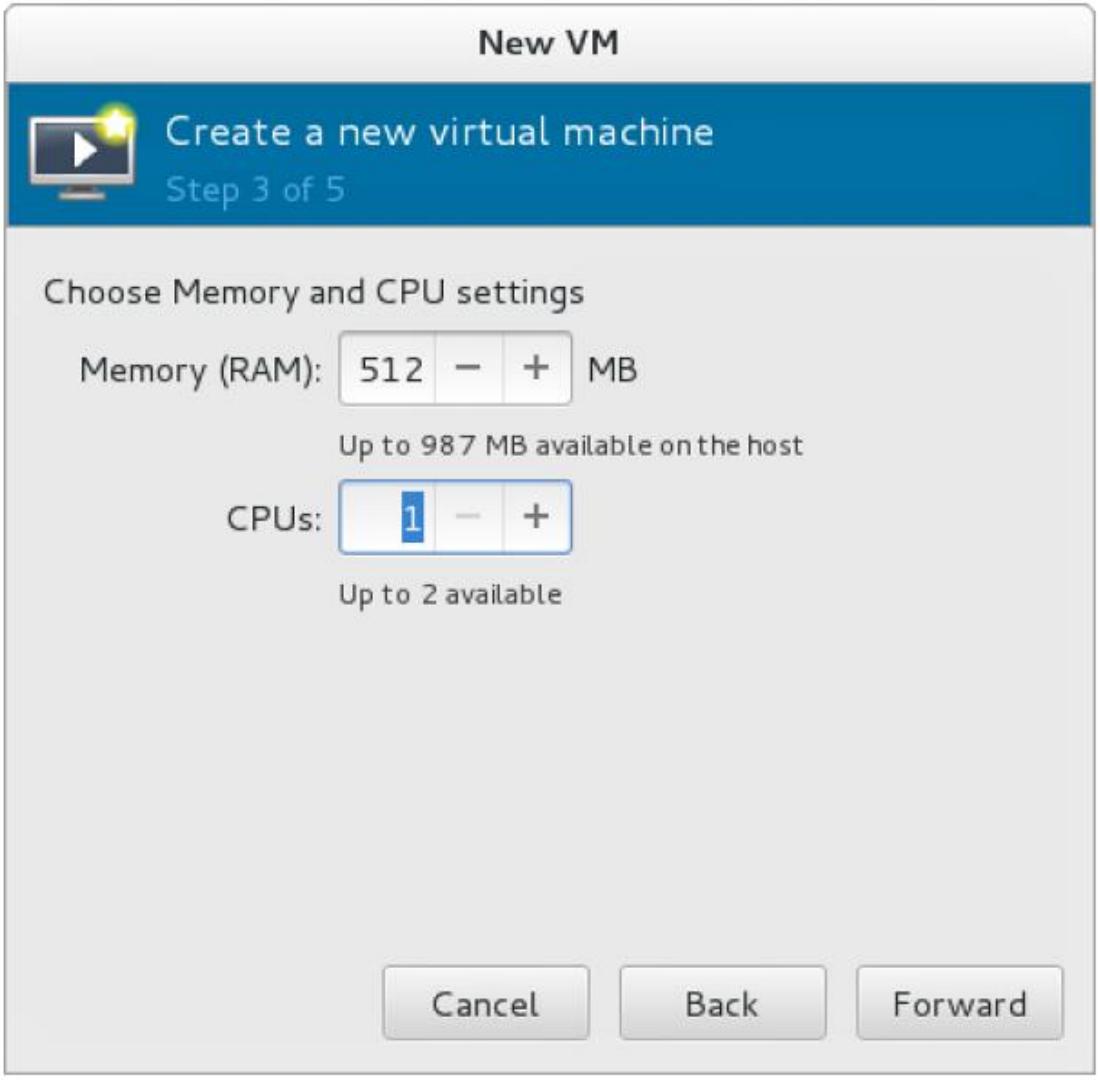

Figure 10.3 Entering VM properties.

Exercise 10.2 Installing a Virtual Machine

8. Now enter the properties of the virtual disk you want to create. For a basic installation, a minimum of 2GB is recommended (see Figure 10.4).

> **TIP** If you have a limited amount of disk space available, just create the VM with the disk space you have got available. You do not have to complete the entire installation procedure (so you do not need all the disk space that is normally needed to complete an installation). All that counts is that you know how to go through the steps in Virtual Machine Manager. (You already know how to install RHEL 7.)

9. At this point, you have entered all properties that are to be used for the VM. You can now click **Finish** to write the VM settings to disk and start the installation procedure. Notice that you do not have to complete the installation procedure. You do not have to use the VM that you are installing here for anything else anymore in later chapters in this book.

Exercise 10.2 Installing a Virtual Machine

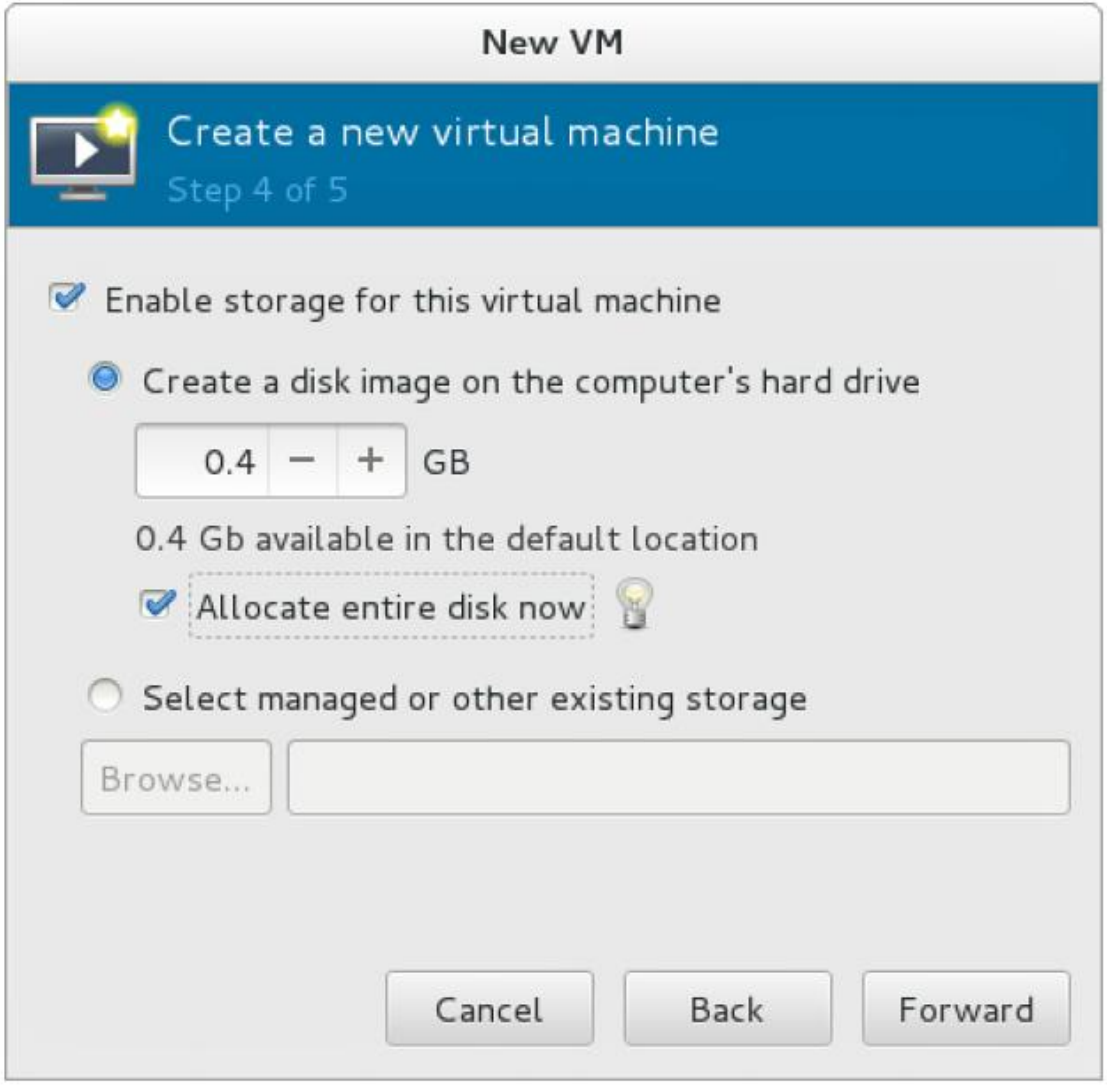

Figure 10.4 Entering the virtual disk size.

Exercise 10.2 Installing a Virtual Machine

In this procedure, you have defined the virtual hardware settings that your VM is going to use. The installation can now be started. The VM settings themselves have been written to an XML configuration file that is stored in the /etc/libvirtd/qemu directory. Listing 10.4 shows the partial contents of the configuration file that was just created.

Listing 10.4 Verifying the Virtual Machine Configuration File XML Code

```
[root@server1 qemu]# cat vmthin1.xml
<!--
WARNING: THIS IS AN AUTO-GENERATED FILE. CHANGES TO IT ARE LIKELY TO
  BE
OVERWRITTEN AND LOST. Changes to this xml configuration should be made
  using:
  virsh edit vmthin1
```

To change virtual hardware settings, you can use the VM properties in Virtual Machine Manager, or use the **virsh edit <vmname>** command. This is because access to the VM settings is streamlined through libvirtd, accessing the configuration directly will mess up your VM.

Using KVM Virtual Machines

After installing a KVM VM, there are multiple ways to access it, including the following:

- SSH into the VM
- Through Virtual Machine Manager
- Through GNOME Boxes
- Using virt-viewer
- Optionally, through third-party utilities.

Of these listed methods, accessing VMs through Virtual Machine Manager is the easiest way to use them. Virtual Machine Manager shows an overview of all the available VMs (see Figure 10.5), and you just have to access the VM window to use them.

Using KVM Virtual Machines

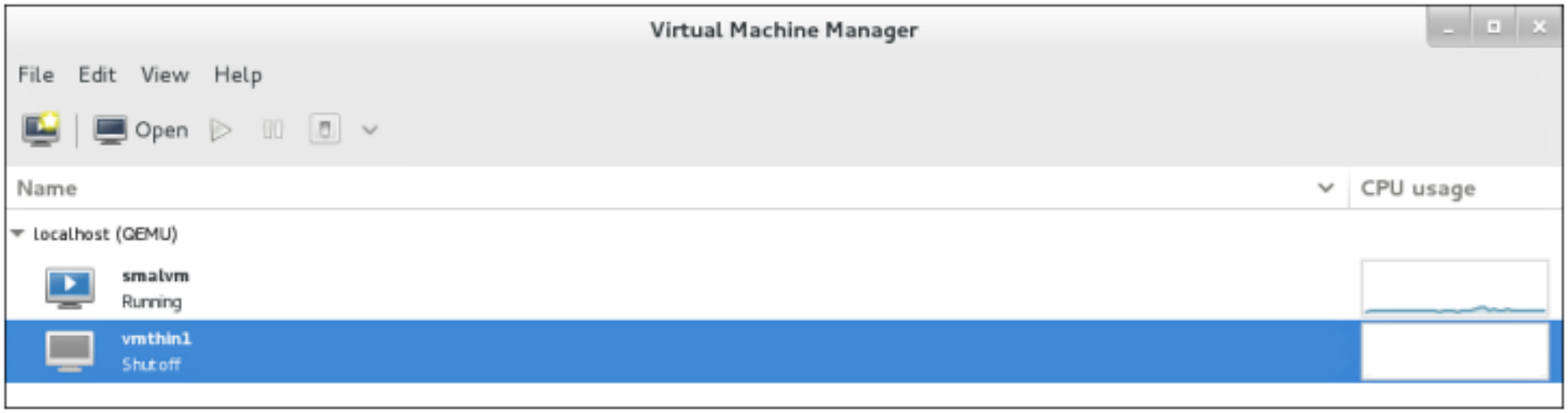

Figure 10.5 Accessing VMs through Virtual Machine Manager.

When accessing a VM in a mode that gives access to the full console environment, the mouse cursor is captured in the VM. To release the mouse cursor, press the left **Ctrl+Alt** keys on your keyboard simultaneously.

From Virtual Machine Manager, you also have access to different icons that enable you to perform specific tasks on the VM easily. For instance, you can use a play button to start a VM, and you can a use a pause button to pause it.

Accessing Virtual Machines from a Text-Only Console

1. Log in to the server1 VM and make sure that you have root privileges.
2. Type **grubby --update-kernel=ALL --args="console=ttyS0"**. Using the **grubby** command allows you to change the configuration of the GRUB2 boot loader without having to go through the GRUB2 configuration files. Alternatively, you can edit the /etc/default/grub file and add the argument **console=ttyS0** to the line that specifies the kernel arguments to be used. If you are modifying the grub.conf file, use **grub2-mkconfig -o /boot/grub2/grub.cfg** to write the changes to the boot loader main configuration file.
3. Restart your VM, using the **reboot** command.
4. From the KVM host, use the **virsh console server1.example.com** command to connect to the VM. You'll now get access to the VM console, as shown **in** Listing 10.5. Press **Ctrl+]** to get out of the virsh console session. Notice that the name of the VM you are connecting to has to match the VM name, as you can see it using the **virsh list** command.

Accessing Virtual Machines from a Text-Only Console

Listing 10.5 Using **virsh console** to Connect to a VM

```
[root@lab ~]# virsh console sander-vm1
Connected to domain sander-vm1
Escape character is ^]

CentOS Linux 7 (Core)
Kernel 3.10.0-123.el7.x86_64 on an x86_64

server2 login:
```

Managing Virtual Machine Properties

As an administrator, you'll occasionally have to change VM properties. The easiest way to do this is through Virtual Machine Manager. To access the properties in Virtual Machine Manager, you first must open the VM. It does not have to be started; it just needs to be open. After opening it, click the icon that looks like a lamp to open the interface shown in Figure 10.6.

Managing Virtual Machine Properties

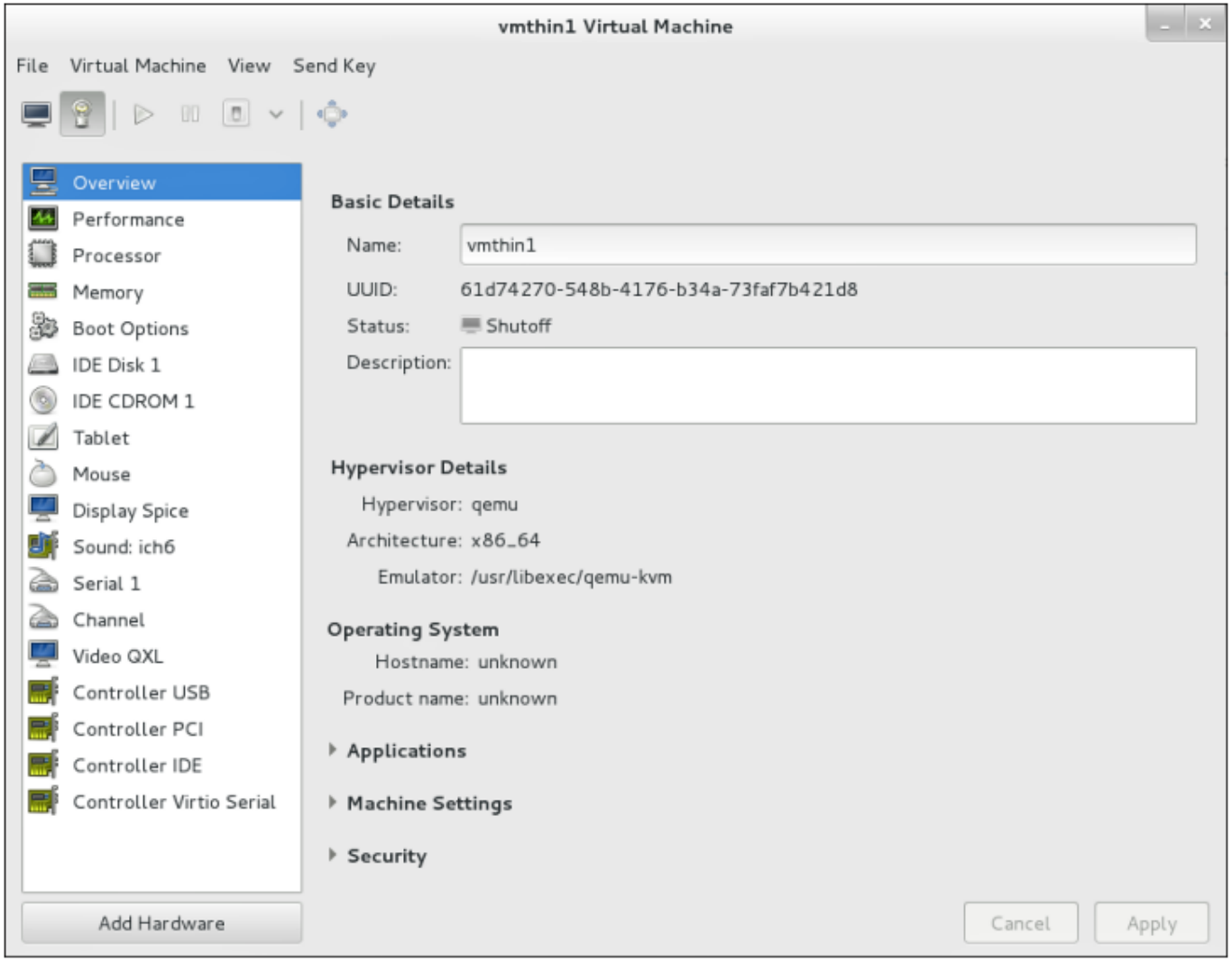

Figure 10.6 Accessing VM properties through Virtual Machine Manager.

Managing Virtual Machine Properties

As you can see, you have many options available from the Virtual Machine Manager properties interface. Many of them are self-explanatory. Some of the most common configuration tasks that you can access through this interface are as follows:

- To add new hardware, click the **Add Hardware** button in the lower-left part of the window. This opens an interface from which you can select the hardware device to be added, as well as its additional properties.
- Click the **Performance** option to show performance graphs about VM usage.
- Click the **Memory** option to grow or reduce the size of memory that is allocated to the VM.
- Click **Boot Options** to enable Autostart. This will start the VM upon host boot.
- Also from the Boot Options interface, you'll find the **Boot Device Order**. Select this to specify the order in which devices in your VM will be used for booting.

Managing Virtual Machines from the CLI

RHEL offers a versatile command-line interface to manage VMs directly from the command line. You can start the **virsh** command with many different arguments to perform specific tasks. You can also just type **virsh** to open a command-line interface from which you can type the specific commands immediately. Table 10.2 shows some of the most common **virsh** commands.

Managing Virtual Machines from the CLI

Table 10.2 virsh Command Interface

Command	Use
list	Shows all VMs that are currently active
list --all	Shows all VMs, including machines that are not currently active
help	Gives a list of all parameters that can be used with the virsh command
shutdown <vmname>	Shuts down the VM properly
destroy <vmname>	Halts a VM, similar to pulling the power plug on a real computer
edit <vmname>	Opens a vi interface that allows you to edit the XML configuration file belonging to a specific VM
console <vmname>	Connects to a VM directly from the console of a KVM host server
start <vmname>	Starts a VM
reboot <vmname>	Reboots a VM

Managing Virtual Machines from the CLI

When using **virsh** commands, you often have to specify the VM name. An alternative is to use the VM ID. To get an overview of VM IDs, use **virsh list**. You'll see IDs listed for all VMs that are currently active (see Listing 10.6).

Listing 10.6 Generating a List of Active Virtual Machines

```
[root@lab ~]# virsh list
 Id    Name                           State
----------------------------------------------------
 3     dan-vm1                        running
 4     dan-vm2                        running
 6     sander-vm1                     running
 7     sander-vm2                     running
 8     sander-ipa                     running
 10    vm1-rhel6-svv                  running
 12    vm2-rhel6-svv                  running
 16    sander-server1                 running
 18    dan-labipa                     running
```

Monitoring Virtual Machine Activity from Top

root@lab:~ | root@server1:/usr/lib/systemd/system

```
top - 10:37:47 up 18 days, 17:58,  2 users,  load average: 0.00, 0.01, 0.05
Tasks: 171 total,   1 running, 170 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.2 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  16196548 total,  4158596 used, 12037952 free,     1336 buffers
KiB Swap: 20479996 total,        0 used, 20479996 free.   247696 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
10821 qemu      20   0 3658648 1.156g   7312 S   2.3  7.5 523:09.19 qemu-kvm
 3028 qemu      20   0 1793264 1.160g   7460 S   2.0  7.5 525:42.93 qemu-kvm
10840 qemu      20   0 3127136 416508   7356 S   1.7  2.6 336:15.41 qemu-kvm
    1 root      20   0   52980   6896   3756 S   0.0  0.0   0:27.02 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.31 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:01.89 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      rt   0       0      0      0 S   0.0  0.0   0:00.67 migration/0
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/0
   10 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/1
   11 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/2
   12 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/3
   13 root      20   0       0      0      0 S   0.0  0.0   1:02.29 rcu_sched
   14 root      20   0       0      0      0 S   0.0  0.0   0:22.01 rcuos/0
   15 root      20   0       0      0      0 S   0.0  0.0   0:23.95 rcuos/1
   16 root      20   0       0      0      0 S   0.0  0.0   0:25.96 rcuos/2
   17 root      20   0       0      0      0 S   0.0  0.0   0:23.15 rcuos/3
   18 root      rt   0       0      0      0 S   0.0  0.0   0:07.93 watchdog/0
   19 root      rt   0       0      0      0 S   0.0  0.0   0:07.92 watchdog/1
   20 root      rt   0       0      0      0 S   0.0  0.0   0:00.13 migration/1
   21 root      20   0       0      0      0 S   0.0  0.0   0:00.89 ksoftirqd/1
   23 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:0H
   24 root      rt   0       0      0      0 S   0.0  0.0   0:06.73 watchdog/2
   25 root      rt   0       0      0      0 S   0.0  0.0   0:00.30 migration/2
```

Figure 10.7 Monitoring VM activity with **top**.

Summary

In this chapter, you learned how to work with VMs in Red Hat Enterprise Linux 7. You learned about virtualization in RHEL7 and how to set up a KVM host server. You also learned how to install a VM and how to manage it, using a graphical utility like Virtual Machine Manager or the **virsh** command-line interface.

Define Key Terms

- Define the following key terms:

  - hypervisor,
  - KVM,
  - libvirt,
  - cloud,
  - openstack,
  - virsh,
  - virbr,
  - virtual bridge,
  - qemu

Lab 10.1

1. Set up one server as a KVM hypervisor host. Notice that if you're doing these labs on a virtual machine, you'll need to enable the hypervisor extensions in the virtual machine software. Consult the documentation of your virtualization software to find out how to do this.
2. On this host, install a KVM virtual machine. You do not have to complete the entire installation procedure. Just make sure that the virtual machine configuration is created and written to disk.

Q&A