8
Configuring Linux Servers

In this chapter, you will learn how to configure different types of Linux servers, from **Domain Name System** (**DNS**) and **Domain Host Configuration Protocol** (**DHCP**) servers, to web servers, Samba file servers, **File Transfer Protocol** (**FTP**) servers, and **Network File System** (**NFS**) servers. All these servers, in one way or another, are powering the backbone of the **World Wide Web** (**WWW**). The reason your computer is showing the exact time is because of a well-implemented **Network Time Protocol** (**NTP**) server. You can shop online and transfer files between your friends and colleagues thanks to good working DHCP, web, and file servers. Configuring different types of Linux services that power all these servers represents the knowledge base for any Linux system administrator.

In this chapter, we're going to cover the following main topics:

- Introduction to Linux services
- Setting up a DNS server
- Setting up a DHCP server
- Setting up an NTP server
- Setting up an NFS server
- Setting up a Samba file server
- Setting up an FTP server
- Setting up a web server
- Setting up a printing server

Technical requirements

Basic knowledge of networking and Linux commands is required. No special technical requirements are needed—just a working installation of Linux on your system. We will use Ubuntu 20.04.1 **Long-Term Support** (**LTS**) as the distribution of choice for this chapter's exercises and examples. Nevertheless, any other major Linux distribution—such as CentOS, openSUSE, or Fedora—is equally suitable for the tasks detailed in this chapter.

Introduction to Linux services

Everything you have learned up to *Chapter 7*, *Networking with Linux* will easily apply to any workstation or desktop/laptop running Linux. Starting with *Chapter 7*, *Networking with Linux*, we have delved into advanced networking subjects that are meant to ease your learning path to becoming a seasoned Linux system administrator. Starting with this chapter, we will enter the server territory as a natural path to the cloud, which will be discussed in detail in the last four chapters of this book.

A Linux server, compared to a Linux workstation, is a system that serves content over a network. While doing so, a server provides its hardware and software resources to different clients that are accessing it. For example, every time you enter a website address into your browser, a server is accessed. That particular type of server is a web server. When you print over the network in your workplace, you access a print server, and when you read your email, you access a mail server. All these are specialized systems that run a specific piece of software (sometimes called a service) that provides you, the client, with the data you requested. Usually, servers are very powerful systems that have lots of resources available for a client's use.

In contrast, a workstation (which is yet another powerful piece of hardware) is generally used for personal work, not for client access over a network. A workstation is used for intensive work, similar to any regular desktop or laptop system. In the light of everything we have exposed up to now, the contents of this chapter and the following chapters are best suited for server use.

You've heard about setting up different Linux servers—such as a web server, a file server, or an email server—and have probably wondered why they are called that. They do not represent the hardware boxes that are the actual servers—they are basically services running inside Linux. What are Linux services? These are programs that run in the background. Inside the Linux world, those services are known as daemons. You were briefly introduced to daemons and the `init` process in *Chapter 5*, *Processes, Daemons, and Signals*, when we discussed what processes, daemons, and signals are and how to manage them in Linux. The mother of all processes is the `init` process, which is among the first processes when Linux boots up. Currently, the latest version of Ubuntu (and also CentOS, Fedora, openSUSE, and others) uses `systemd` as the default `init` process.

We will refresh your memory by using some of the basic commands for working with services in Linux. If you want more information, please refer to *Chapter 5*, *Processes, Daemons, and Signals*. The first command we will remind you of is the `ps` command. We will use it to show the `init` process running, as follows:

```
ps -ef | less
```

The output on our Ubuntu 20.04.1 system is shown in the following screenshot:

```
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 sep13 ?        00:01:41 /sbin/init splash
root           2       0  0 sep13 ?        00:00:03 [kthreadd]
root           3       2  0 sep13 ?        00:00:00 [rcu_gp]
root           4       2  0 sep13 ?        00:00:00 [rcu_par_gp]
```

Figure 8.1 – Showing the init process by using the ps command

In the process list shown, the first process is the `init` process, or `systemd`. It uses the `init` name for backward-compatibility issues, but to make sure that it really is `systemd`, you can use the manual pages of `init` for more details. When you type `man pages` in the command line, the manual page shown is for `systemd`. `systemd`, as the parent of all services, starts all the running processes in parallel as a way to make the boot process and service-time response more efficient. To see how efficient those processes are, you can run the `systemd-analyze` command, as shown in the following screenshot:

```
packt@neptune:~$ systemd-analyze
Startup finished in 44.504s (kernel) + 7.581s (userspace) = 52.086s
graphical.target reached after 7.556s in userspace
```

Figure 8.2 – The systemd-analyze command

The preceding command is a new command, not shown in *Chapter 5*, *Processes, Daemons, and Signals*, but a command you should already know is the `systemctl` command, which is the main command-line utility for working with `systemd` services (or daemons).

You may also know by now that `systemctl` invokes **units**, as system resources managed by `systemd`. Those units have several types, such as service, mount, socket, and others. To see those units listed by the time they use to start up, you can use the `systemd-analyze blame` command, as shown in the following screenshot:

```
packt@neptune:~$ systemd-analyze blame
6.113s NetworkManager-wait-online.service
3.308s fwupd.service
2.951s bolt.service
2.822s apt-daily-upgrade.service
2.275s plymouth-quit-wait.service
 996ms fstrim.service
 688ms dev-nvme0n1p2.device
 680ms snapd.service
 415ms plymouth-read-write.service
 368ms systemd-logind.service
 275ms man-db.service
 271ms snap-core18-1885.mount
 267ms snap-gtk\x2dcommon\x2dthemes-1506.mount
 180ms dev-loop1.device
 170ms networkd-dispatcher.service
 168ms snap-gnome\x2d3\x2d34\x2d1804-36.mount
 138ms udisks2.service
 111ms snap-snap\x2dstore-481.mount
 104ms logrotate.service
  87ms upower.service
  87ms systemd-resolved.service
  82ms avahi-daemon.service
  82ms NetworkManager.service
```

Figure 8.3 – The systemd-analyze blame command

The output shows different types of units, such as service, mount, and device. The preceding screenshot is only an excerpt of the running units. To learn more about the `systemctl` command, feel free to use the manual pages, or go back to *Chapter 5*, *Processes, Daemons, and Signals* to refresh your memory.

We believe that this short introduction to Linux services is only a refresher of the respective section from *Chapter 5*, *Processes, Daemons, and Signals*, just enough for you to start delving into setting up and configuring specific services (or servers) on Linux. We will show you how to manage some of the most important services in Linux, such as DNS, DHCP, NTP, Samba, NFS, web, FTP, and printing services. The theory on all those services was already provided in *Chapter 7*, *Networking with Linux*. Now, it's time to roll up your sleeves and configure them yourself.

Setting up a DNS server

One of the most widely used DNS services is **Berkeley Internet Name Domain 9** (**BIND 9**). You can visit its official website at the following address: https://www.isc.org/bind/. Before continuing, let's underline the system configuration and goals. For this section of the chapter, we will use an old test system, as we want to show you that this project can be run on older hardware too. We will configure BIND as a private-network DNS server using

Ubuntu 20.04.1 on a dual-core Intel i3 system, with 8 **gigabytes** (**GB**) of **random-access memory** (**RAM**) and 128 GB **solid-state drive** (**SSD**). On this system, we will create two types of servers, a caching name server and a primary name server, which you can use on your local network to manage hostnames and private **Internet Protocol** (**IP**) addresses.

There are other ways to do this, but for the purpose of showing you the basics of DNS setup, this configuration will suffice. If you would like a secondary server, you will need to have another spare system, or if you use **virtual private servers** (**VPSs**), they would have to be in the same data center and be using the same private network. In our case, however, we will use a local system on our small private network.

First, we will install the `bind9` package in Ubuntu by using the following command:

```
alexandru@asus:~$ sudo apt install bind9 bind9utils bind9-doc
[sudo] password for alexandru:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libfprint-2-tod1 linux-headers-5.4.0-26 linux-headers-5.4.0-26-generic
  linux-image-5.4.0-26-generic linux-modules-5.4.0-26-generic
  linux-modules-extra-5.4.0-26-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  bind9-utils python3-ply
Suggested packages:
  bind-doc resolvconf python-ply-doc
The following NEW packages will be installed:
  bind9 bind9-doc bind9-utils bind9utils python3-ply
0 upgraded, 5 newly installed, 0 to remove and 0 not upgraded.
Need to get 708 kB of archives.
After this operation, 3.828 kB of additional disk space will be used.
Do you want to continue? [Y/n] _
```

Figure 8.4 – Installing bind9 package on Ubuntu 20.04.1

The preceding command will install all the packages needed for Bind9 to run.

Important note

In the preceding output, you see that `apt` informs us about unnecessary packages that can be removed. Please take into consideration that this message might not appear on your system, as it is a result of a recent upgrade on our machine. Whenever it appears, run the `sudo apt autoremove` command to delete packages that aren't needed.

Once the packages are installed, you can test them to see that they work. For this, we will use the `nslookup` command, as shown in the following screenshot, by using the local address (or loopback address):

```
alexandru@asus:~$ nslookup google.com 127.0.0.1
Server:         127.0.0.1
Address:        127.0.0.1#53

Non-authoritative answer:
Name:   google.com
Address: 172.217.16.110
Name:   google.com
Address: 2a00:1450:400d:808::200e
```

Figure 8.5 – Checking to see if Bind is working

Now, you can start setting up the service. First, we will configure a caching DNS service. But before we start, we advise you to back up the following configuration files: `/etc/bind/named.conf`, `/etc/bind/named.conf.options`, `/etc/hosts`, and `/etc/resolv.conf`. Let's see how to create a caching server.

Caching a DNS service

The default behavior of Bind9 is as a caching server. This means that setting it up is quite straightforward. We will tweak the configuration file just a little, in order to make it work according to our requests. After seeing that the installed packages work as intended, you can also configure the firewall to allow Bind9, like this:

```
alexandru@asus:~$ sudo ufw allow Bind9
[sudo] password for alexandru:
Rules updated
Rules updated (v6)
```

Figure 8.6 – Allowing Bind9 access through the firewall

In Ubuntu, you will have to alter the `/etc/bind/named.conf.options` file to add or delete different options. We will do that by opening it with the Nano text editor. Inside the file, as shown in *Figure 8.7*, are a few settings that have been set up, and a lot of commented lines with details on how to use the file. The `//` double slashes indicate that the respective lines are commented out. All the modifications will be done inside the `options` directive, between curly brackets, as can be seen here:

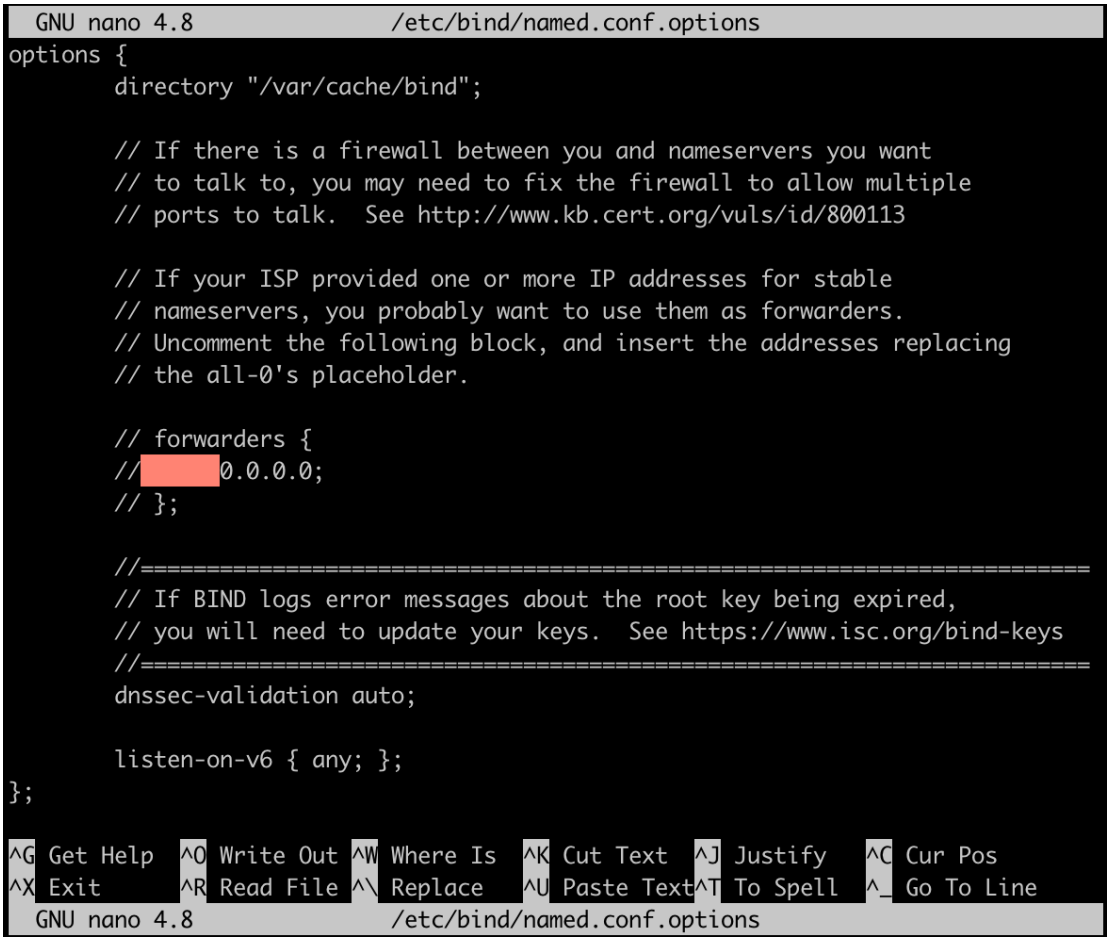

Figure 8.7 – The /etc/bind/named.conf.options file original contents

The first thing to do, as we will only use **IP version 4** (**IPv4**), is to comment out the following line by adding two slashes:

```
// listen-on-v6 { any; };
```

You will have to add a list of IP addresses inside the `forwarders` directive. This line tells the server where to look in order to find addresses not cached locally. For simplicity, we will add the Google public DNS servers, but feel free to use your **Internet Service Provider's** (**ISP's**) DNS servers. Now, edit the `forwarders` directive to look like this:

```
forwarders {
        8.8.8.8;
        8.8.4.4;
          };
```

You can also add a line defining the `allow-query` spectrum. This line is telling the server which networks can be accepted for DNS queries. You can add your local network address. In our case, it will be the following:

```
allow-query {
      localhost;
      192.168.0.0/24;
        };
```

There is also a `listen-on` directive, where you can specify the networks the DNS server will work for. This applies for IPv4 addresses and is shown in the following code snippet:

```
listen-on {
        192.168.0.0/24;
        }
```

After the added options, the file should look like this:

```
forwarders {
8.8.8.8; 8.8.4.4;
};

//========================================================================
// If BIND logs error messages about the root key being expired,
// you will need to update your keys.  See https://www.isc.org/bind-keys
//========================================================================
dnssec-validation auto;

// listen-on-v6 { any; };

allow-query {
localhost; 192.168.0.0/24;
};

listen-on {
192.168.0.0/24;
};
```

Figure 8.8 – The final form of /etc/bind/named.conf.options after adding new options

Save the file and exit the editor. You can check the Bind9 configuration with the `named-checkconf` command. If there is no output, it means that the configuration of the file is

correct. Restart the Bind9 service and optionally check its status, as demonstrated in the following screenshot:

```
alexandru@asus:~$ sudo named-checkconf
alexandru@asus:~$ sudo systemctl restart bind9
alexandru@asus:~$ sudo systemctl status bind9
● named.service - BIND Domain Name Server
     Loaded: loaded (/lib/systemd/system/named.service; enabled; vendor preset: enabl>
     Active: active (running) since Wed 2020-11-04 14:36:50 EET; 6s ago
       Docs: man:named(8)
   Main PID: 7886 (named)
      Tasks: 14 (limit: 9375)
     Memory: 27.0M
     CGroup: /system.slice/named.service
             └─7886 /usr/sbin/named -f -u bind

nov 04 14:36:50 asus named[7886]: network unreachable resolving './NS/IN': 2001:503:b>
nov 04 14:36:50 asus named[7886]: network unreachable resolving './NS/IN': 2001:500:1>
nov 04 14:36:50 asus named[7886]: network unreachable resolving './NS/IN': 2001:7fe::>
nov 04 14:36:50 asus named[7886]: network unreachable resolving './NS/IN': 2001:dc3::>
nov 04 14:36:50 asus named[7886]: network unreachable resolving './NS/IN': 2001:500:2>
nov 04 14:36:50 asus named[7886]: network unreachable resolving './NS/IN': 2001:500:2>
nov 04 14:36:50 asus named[7886]: network unreachable resolving './NS/IN': 2001:500:2>
nov 04 14:36:50 asus named[7886]: network unreachable resolving './NS/IN': 2001:500:9>
nov 04 14:36:50 asus named[7886]: managed-keys-zone: Key 20326 for zone . is now trus>
nov 04 14:36:50 asus named[7886]: resolver priming query complete
```

Figure 8.9 – Checking the file configuration and restarting the Bind9 service

Knowing that the Bind9 service is working fine, you can test the service from any other system on the network with the `nslookup` command, as follows:

```
~ ❯❯❯ nslookup google.com 192.168.0.205
Server:         192.168.0.205
Address:        192.168.0.205#53

Non-authoritative answer:
Name:   google.com
Address: 216.58.214.238
```

Figure 8.10 – Testing the Bind9 implementation

The preceding output shows that the DNS service on our Asus machine is working fine. The test was conducted from a MacBook Pro on the same network.

You now have a working caching DNS server on your private network. In the next section, we will show you how to create a primary DNS server.

Primary DNS server

In order to configure a primary DNS server, we will need a domain name that it will serve. For this section's purpose, we will use the `openlark.com` domain name. We will have to create new zones to the Bind9 configuration, and we will add information to the `/etc/bind/named.conf.local` file about the ones we create. Right now, we will create a new zone for our `openlark.com` domain. In the following screenshot, you can see the contents of our configuration file, followed by details on each of the lines:

```
alexandru@asus:/etc/bind$ cat named.conf.local
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

zone "openlark.com" {
        type master;
        file "/etc/bind/db.openlark.com";
        allow-transfer { 192.168.0.205; };
        also-notify { 192.168.0.205; };
};
```

Figure 8.11 – New zone for our domain in /etc/bind/named.conf.local file

Now, let's explain the contents of a zone directive. First, we had to give the name of the domain that the zone will serve. The type of the zone is set as `master`, but there are other types to use, such as `slave`, `forward`, or `hint`. The file represents the path to the actual zone file that will be created. In the `allow-transfer` list, the IPs of DNS servers that handle the zone are set. In the `also-notify` list, the IPs of servers that will be notified about zone changes are indicated. For more details about DNS zones, please refer to https://ns1.com/resources/dns-zones-explained. The following screenshot shows how we copied the `db.local` file under another name and used it for a new zone:

```
alexandru@asus:/etc/bind$ ls
bind.keys  db.255    named.conf               named.conf.options
db.0       db.empty  named.conf.default-zones  rndc.key
db.127     db.local  named.conf.local          zones.rfc1918
alexandru@asus:/etc/bind$ sudo cp db.local db.openlark.com
```

Figure 8.12 – Creating the zone file for our domain

The next step is to create the zone file, as indicated in the zone directive about `openlark`. The details are shown in the preceding screenshot. Once the file is created using `db.local` as a template, you can open it with your favorite text editor and add information about your server IP and domain name. In the following screenshot, you can see the zone file for `openlark` created on our Asus machine on the network:

```
alexandru@asus:/etc/bind$ cat db.openlark.com
;
; BIND data file for local loopback interface
;
$TTL    604800
@       IN      SOA     ns.openlark.com. admin.openlark.com. (
                              2         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      ns.openlark.com.
@       IN      A       192.168.0.205
ns      IN      A       192.168.0.205
```

Figure 8.13 – Zone file information

The DNS records are introduced at the end of the file. Here are some details about the contents of the file:

- The table has a specific format that contains details about hostname (first column), class (second column), DNS record type (third column), and value (the last column).
- As the hostname, we entered `@`, which means that the entry of the record refers to the zone name from the file.
- The class is `IN`, which indicates that the type of network is an internet one.
- DNS records types are `A`, `NS`, `MX`, `CNAME`, `TXT`, and `SOA`. `A` indicates the IP address of the domain name; `NS` indicates the IP address of the DNS server; `MX` is the address of the email server; `CNAME` is an alias (canonical name); and `TXT` has a custom entry, `SOA`, which indicates the authoritative name server for the zone, with details on the administrator, serial number, and refresh rates.
- The value in the last column most often compraises of the IP address or the hostname.

The next step is to restart the **Remote Name Daemon Control** (**RNDC**), which is a control utility inside BIND that controls the name server. The command to do that is shown here:

```
sudo rndc reload
```

Now, you can check to see if the primary DNS server works. Try the `nslookup` command from another system on the network, as illustrated in the following screenshot:

```
~ ❯❯❯ nslookup openlark.com 192.168.0.205                                    ✘ 1
Server:         192.168.0.205
Address:        192.168.0.205#53

Name:   openlark.com
Address: 192.168.0.205
```

Figure 8.14 – Testing the DNS server from another system on the network

The output of the preceding command shows that the local DNS server on the system with the indicated IP address has a working zone file. Your primary DNS server works as expected on the local network.

It is always a good idea to create a secondary DNS server in case the first one stops working, which is why we will show you how to set up a second one in the following section.

Secondary DNS server

It comes as no surprise that the secondary DNS server should be set up on a different hardware box from the primary one. If you do it inside a data center, use a VPS in the same network as the first one. If you plan to experiment with it on your home private network, make sure you have another system at your disposal.

Before setting up the secondary server, you will need to modify the configuration of the primary DNS server first by allowing it to send the zone details to the secondary server.

You will have to open the `/etc/bind/named.conf.local` configuration file and add some new lines to it. We add the second server's IP address inside the `allow-transfer` and `also-notify` directives, as follows:

```
zone "openlark.com" {
        type master;
        file "/etc/bind/db.openlark.com";
        allow-transfer { 192.168.0.205; 192.168.0.206; };
        also-notify { 192.168.0.205; 192.168.0.206; };
```

```
};
```

Save the file and restart the Bind9 service. You will also need to open the `/etc/bind/named.conf.options` configuration file and add two extra parameters with the `allow-transfer` and `also-notify` directives. We will also add an access list parameter (`acl "trusted"`) with all the accepted IP addresses on the network. In our case, the primary server has the address `192.168.0.205`, and the second server has the address `192.168.0.206`. The `acl` directive will have the following code:

```
acl "trusted" {
        192.168.0.205;
        192.168.0.206;
};
```

Inside the `options` directive, we also add the `allow-transfer` and `also-notify` directives, as follows:

```
recursion yes;
allow-recursion { trusted; };
listen-on { 192.168.0.206; };
allow-transfer { 192.168.0.206; };
also-notify { 192.168.0.206; };
```

The configuration is finished, and we will restart the Bind9 service using the `systemctl` command, as follows:

```
sudo systemctl restart bind9.service
```

At this point, you have two working local DNS servers—a primary and a secondary one—that will serve your private network.

Next on the list is a local DHCP server, and we will show you how to configure one in the next section.

Setting up a DHCP server

The DHCP is a network service that is used to assign settings to hosts on a network. The settings are enabled by the server, without any control from the host. Most commonly, the DHCP server provides the IP addresses and netmask for clients, the default gateway IP, and the DNS server's IP address.

To install the DHCP service on Ubuntu, use the following command:

```
sudo apt install isc-dhcp-server
```

As a test system, we will use the same Asus system on which we installed the DNS services in the previous section. After installation, you will configure two specific files. The default configuration will be changed inside the `/etc/dhcp/dhcpd.conf` file, while the interfaces will be configured inside the `/etc/default/isc-dhcp-server` file.

We will show you how to set up a basic local DHCP server. In this respect, we will alter the `/etc/dhcp/dhcpd.conf` file by adding the IP addresses pool. You can either uncomment one of the subnet directives from the file or you can add a new one, which is what we will do. Our existing subnet is `192.168.0.0/24`, and we will add a new one for this new DHCP server. The example is shown in the following code snippet:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
 range 192.168.1.100 192.168.1.200;
 option routers 192.168.1.1;
 option domain-name-servers 192.168.1.2, 192.168.1.3;
 option domain-name "homecomputer.local";
 option broadcast-address 192.168.1.255
}
```

Once the options are modified, you must specify the network interface in the `/etc/default/isc-dhcp-server` file. Open the file with your preferred editor and add the interface. If you don't remember your interface name, we will give you a short reminder next. Run the `ip addr show` command and select the appropriate one. In our case, the system we use has both Ethernet and wireless interfaces, and we will choose the Ethernet interface for the DHCP server, as illustrated in the following screenshot:

```
alexandru@asus:/etc/dhcp$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 100
0
    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: enp2s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group de
fault qlen 1000
    link/ether 2c:56:dc:2c:aa:fc brd ff:ff:ff:ff:ff:ff
3: wlp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
 qlen 1000
    link/ether 80:a5:89:71:08:d7 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.205/24 brd 192.168.0.255 scope global dynamic noprefixroute wlp3s0
       valid_lft 5943sec preferred_lft 5943sec
    inet6 fe80::8b4e:e6da:99bc:c259/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figure 8.15 – Showing the network interfaces with the ip command

In the preceding output, you may have seen that the system is assigned an IP from the currently existing subnet on the wireless interface. Nonetheless, it will be disconnected from the existing subnet and will serve the newly configured one after the service restart. Inside the `/etc/default/isc-dhcp-server` file, add the interface as follows:

```
INTERFACESv4="enp2s0"
```

Then, save the changes to the file and restart the DHCP service with the following command:

```
sudo systemctl restart isc-dhcp-server
```

Now, you have a working DHCP server on your system of choice. A DHCP server gives you some advantages in managing your local network. There are times when you might not need to create a new one, as all the routers provide a fully working DHCP service right out of the box. In the next section, we will show you how to set up an NTP server on your local network.

Setting up an NTP server

The NTP is a protocol used to synchronize time across networks. This protocol ensures that all computers are synchronized to a **millisecond** (**ms**) level. Across the world, there are several tiers of NTP servers, starting from ones connected to atomic clocks (tier one), on to ones that serve time requests over the internet. Accurate time management might be of paramount importance when running your servers, so you should therefore know how to configure NTP.

Inside Ubuntu, there used to be a package called `npt` that was installed and used, but ever since version 16.04, two `systemd` packages called `timedatectl` and `timesyncd` have been used to replace `ntp`. Inside Ubuntu, clients can also use `chrony` to send queries to NTP. By default, `chrony` is not installed, and the package used instead is `systemd-timesyncd`.

Nevertheless, default packages cannot serve NTP as they are only clients. Before installing NTP, let's see how the `systemd-timesyncd` package works by default, as follows:

1. First, check the status of the service using the `systemctl` command, as shown here:

```
packt@neptune:~$ systemctl status systemd-timesyncd
● systemd-timesyncd.service - Network Time Synchronization
     Loaded: loaded (/lib/systemd/system/systemd-timesyncd.service; enabled; ve>
     Active: active (running) since Mon 2020-11-09 12:36:46 EET; 8min ago
       Docs: man:systemd-timesyncd.service(8)
   Main PID: 334847 (systemd-timesyn)
     Status: "Initial synchronization to time server 91.189.89.199:123 (ntp.ubu>
      Tasks: 2 (limit: 9142)
     Memory: 1.6M
     CGroup: /system.slice/systemd-timesyncd.service
             └─334847 /lib/systemd/systemd-timesyncd

nov 09 12:36:46 neptune systemd[1]: Starting Network Time Synchronization...
nov 09 12:36:46 neptune systemd[1]: Started Network Time Synchronization.
nov 09 12:36:46 neptune systemd-timesyncd[334847]: Initial synchronization to t>
lines 1-14/14 (END)
```

Figure 8.16 – Checking the status of the systemd-timesyncd module

2. Then, check the time configuration of `timedatectl`, as follows:

```
packt@neptune:~$ timedatectl status
               Local time: Lu 2020-11-09 13:10:41 EET
           Universal time: Lu 2020-11-09 11:10:41 UTC
                 RTC time: Lu 2020-11-09 11:10:41
                Time zone: Europe/Bucharest (EET, +0200)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

Figure 8.17 – Status of the timedatectl module

Up to now, all we did was check the status of `timesyncd` and see how it works by default on Ubuntu. It shows the local, universal, and **real-time clock** (**RTC**) time; the time zone;

and the synchronization status. If you want to create an NTP server, the Ubuntu developers recommend you use `chrony`, but you can also use `ntpd` and `open-ntp`.

Installing the NTP server

In order to serve NTP information to other systems on the network or on the internet, you will need to install an NTP server. We will show you how to install and configure `chrony`, as it is the recommended solution. The command to install the `chrony` package is shown in the following screenshot:

```
packt@neptune:~$ sudo apt install chrony
[sudo] password for packt:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  systemd-timesyncd
The following NEW packages will be installed:
  chrony
0 upgraded, 1 newly installed, 1 to remove and 0 not upgraded.
Need to get 220 kB of archives.
After this operation, 292 kB of additional disk space will be used.
Do you want to continue? [Y/n] _
```

Figure 8.18 – Installing chrony in Ubuntu

Pay attention to the preceding message, as you are warned that the `systemd-timesyncd` package will be removed from your system. Furthermore, the installation process will show you several other warning messages, but the packages requested will be installed, and others no longer able to run will be removed. Such a warning message is shown in the following code snippet. The installation process shows which packages are to be removed and which will be installed:

```
systemd depends on systemd-timesyncd | time-daemon; however:

  Package systemd-timesyncd is to be removed.

  Package time-daemon is not installed.

  Package systemd-timesyncd which provides time-daemon is to
be removed.

  Package chrony which provides time-daemon is not installed.
```

By installing the package, two binaries will be provided to the user: `chronyd` and `chronyc`. The first one is a daemon that runs the NTP protocol, and the second is the

interface for the daemon. To configure the daemon, you will need to edit the `/etc/chrony/chrony.conf` file. By default, our configuration file looks like this:

```
# About using servers from the NTP Pool Project in general see (LP: #104525).
# Approved by Ubuntu Technical Board on 2011-02-08.
# See http://www.pool.ntp.org/join.html for more information.
pool ntp.ubuntu.com        iburst maxsources 4
pool 0.ubuntu.pool.ntp.org iburst maxsources 1
pool 1.ubuntu.pool.ntp.org iburst maxsources 1
pool 2.ubuntu.pool.ntp.org iburst maxsources 2

# This directive specify the location of the file containing ID/key pairs for
# NTP authentication.
keyfile /etc/chrony/chrony.keys

# This directive specify the file into which chronyd will store the rate
# information.
driftfile /var/lib/chrony/chrony.drift

# Uncomment the following line to turn logging on.
#log tracking measurements statistics

# Log files location.
logdir /var/log/chrony

# Stop bad estimates upsetting machine clock.
maxupdateskew 100.0

# This directive enables kernel synchronisation (every 11 minutes) of the
# real-time clock. Note that it can’t be used along with the 'rtcfile' directiv>
rtcsync

# Step the system clock instead of slewing it if the adjustment is larger than
# one second, but only in the first three clock updates.
makestep 1 3
```

Figure 8.19 – Default contents of the chrony.conf file on Ubuntu 20.04.1

If you want to stay on the safe side, you can leave the default settings unaltered. You can check the time sources that the default configuration uses, as well as a detailed stats list. Note that the blank line in the following screenshot is a private IP address that has been removed for privacy reasons:

```
packt@neptune:/etc/chrony$ chronyc sources
210 Number of sources = 8
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^+ golem.canonical.com           2   7   377    71   -382us[ -334us] +/-   55ms
^+ alphyn.canonical.com          2   8   377    72   +344us[ +392us] +/-  110ms
^+ chilipepper.canonical.com     2   8   373    73   +315us[ +363us] +/-   47ms
^+ pugot.canonical.com           2   8   377     8   -433us[ -433us] +/-   44ms
^+                               3   8   377    75  +1967us[+2015us] +/-   99ms
^+ ftp.upcnet.ro                 2   8   377    75  -4319us[-4271us] +/-   57ms
^* main-fe0.b.astral.ro          2   8   377    15   -388us[ -340us] +/-   51ms
^+ ntp0.chroot.ro                2   8   377    73  +2795us[+2843us] +/-   41ms
packt@neptune:/etc/chrony$ chronyc sourcestats
210 Number of sources = 8
Name/IP Address            NP  NR  Span  Frequency  Freq Skew  Offset  Std Dev
==============================================================================
golem.canonical.com        21  14   23m     +0.567      1.549   +575us   684us
alphyn.canonical.com       25  14   31m     +0.127      0.635   +635us   456us
chilipepper.canonical.com  24  15   31m     +0.085      0.970  +1194us   680us
pugot.canonical.com        25  14   32m     +0.077      0.538  -1364us   396us
                           25  11   31m     +0.289      0.364  +2308us   258us
ftp.upcnet.ro              25  14   31m     +0.305      0.175  -4285us   134us
main-fe0.b.astral.ro       24  13   32m     -0.014      0.157   -362us   106us
ntp0.chroot.ro             17   9   25m     -0.144      0.119  +2791us    46us
```

Figure 8.20 – The chronyc sources and stats for our NTP server

To further optimize the NTP server, you could add the NTP pool addresses in the `/etc/chrony/chrony.conf` file, based on where you live. To see all the available servers, check the following address: https://www.ntppool.org/en/. After changing the configuration file, restart the service by using the following command:

```
sudo systemctl restart chrony
```

In order to make the server available to all the systems on your network, you will have to add an `allow` directive inside your `/etc/chrony/chrony.conf` configuration file. For our network, the directive looks like this:

```
allow 192.168.0.0/24

allow 192.168.0.244

server 192.168.0.244 prefer iburst
```

The `prefer` option ensures that the indicated source is the one preferred, compared to other sources that could be also available. The `iburst` option ensures that the interval between the first four requests that are sent to the server is 2 seconds or less. Generally, these two options are used by default, so it is considered safe to use them as is.

Now, we will modify the configuration file according to the aforementioned settings. Here is the output of the new file:

```
pool    pool.ntp.org            iburst  maxsources      4
pool    0.europe.pool.ntp.org   iburst  maxsources      1
pool    1.europe.pool.ntp.org   iburst  maxsources      1
pool    2.europe.pool.ntp.org   iburst  maxsources      2


#network access
allow 192.168.0.0/24
allow 192.168.0.244
```

Figure 8.21 – Modified pool list and local network access

Also, here is the updated sources and stats list that changed once we changed the pool servers:

```
packt@neptune:/etc/chrony$ chronyc sources
210 Number of sources = 9
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* neptune                       2   6   377   626  +9563ns[+4038ns] +/-   24ms
^- alpha.rueckgr.at              2   6   377    48  +3321us[+3321us] +/-   47ms
^- srv02.spectre-net.de          2   6   377    52  +2091us[+2091us] +/-   41ms
^- static.9.86.99.88.client>     2   6   377    50  +2197us[+2197us] +/-   47ms
^- ntp2.wtnet.de                 2   6   377    50  +1167us[+1167us] +/-   53ms
^- 183-150-172-163.instance>     5   6   377    49  +3623us[+3623us] +/-   93ms
^- 85.199.214.102                1   6   377    52    +61us[  +61us] +/-   22ms
^- vsrv02141.customer.xenwa>     3   6   377    51   +962us[ +962us] +/-   47ms
^- ntp.cnh.at                    2   6   377    51   +325us[ +325us] +/-   57ms
packt@neptune:/etc/chrony$ chronyc sourcestats
210 Number of sources = 9
Name/IP Address            NP  NR  Span  Frequency  Freq Skew  Offset  Std Dev
==============================================================================
neptune                     4   3    72    -12.662    219.996  -8014us   488us
alpha.rueckgr.at           14   9   655     -0.610      3.738   +431us   657us
srv02.spectre-net.de       14   7   654     +0.557      2.783  +1521us   447us
static.9.86.99.88.client>  14   6   655     +1.237      4.238  +2017us   893us
ntp2.wtnet.de              14   8   653     +0.214      8.685  +1583us  1813us
183-150-172-163.instance>  14  10   658     +0.286      2.010  +2724us   430us
85.199.214.102             14   6   655     -1.340      4.190   -618us   816us
vsrv02141.customer.xenwa>  14   7   654     +0.200      5.549   +730us  1051us
ntp.cnh.at                 13   8   654     -1.072      5.278   +533us   947us
```

Figure 8.22 – Updated chrony sources and stats based on the new pool list

As you can see in the outputs shown in *Figure 8.22*, there is our local server (called `neptune`) listed, among others. Once the configuration is done on the server, you will need to configure a Linux client to accept your server, as follows:

```
pool ntp.ubuntu.com        iburst maxsources 4
pool 0.ubuntu.pool.ntp.org iburst maxsources 1
pool 1.ubuntu.pool.ntp.org iburst maxsources 1
pool 2.ubuntu.pool.ntp.org iburst maxsources 2

server 192.168.0.244 prefer iburst
```

Figure 8.23 – Updated chrony configuration for a Linux client

After the modifications are made, you can use the `chronyc sources` command on the client side. In the output, you will see the local NTP server's IP address, as follows:

```
alexandru@asus:~$ chronyc sources
210 Number of sources = 9
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* 192.168.0.244                 3   6    17     5   -183us[ -627us] +/-   33ms
^- pugot.canonical.com           2   6    17     4   -430us[ -430us] +/-   50ms
^- golem.canonical.com           2   6    35     3  +1135us[+1135us] +/-   45ms
^- chilipepper.canonical.com     2   6    17     4   +679us[ +679us] +/-   50ms
^- alphyn.canonical.com          2   6    17     4   +977us[ +977us] +/-  135ms
^- main-fe0.b.astral.ro          2   6    17     4   +324us[ +324us] +/-   55ms
^- cache.alsys.ro                2   6    17     3   +803us[ +803us] +/-   48ms
^- ntp0.chroot.ro                2   6    17     4  +3557us[+3557us] +/-   52ms
^- 92.86.106.228                 3   6    17     5  +2614us[+2170us] +/-  129ms
```

Figure 8.24 – The NTP IP address on the client side

As you can see, the system with the shown IP address is working as an NTP server on your local network. You can also synchronize the time from a Windows or a macOS system on your network, but this is out of the scope of this section. As a hint, on a macOS machine, you can run the `sntp` command, followed by the IP address of the local NTP server. In the next section, we will show you how to set up an NFS server.

Setting up an NFS server

The NFS is a distributed filesystem used to share files over a network. To show you how it works, we will set up the NFS server on one of our machines on the network. We now have three servers running on the network: two with Ubuntu and one with CentOS. We will use one of the two Ubuntu systems. For more in-depth theoretical information about NFS, please refer to *Chapter 7*, *Networking with Linux*.

The NFS filesystem type is supported by any Linux and/or Unix environment and also by Windows, but with some limitations. For mostly-Windows client environments, we recommend using the Samba/**Common Internet File System** (**CIFS**) protocol instead. Also, for those of you concerned about privacy and security, please keep in mind that the NFS protocol is not encrypted, thus any transfer of data is not protected by default.

Installing and configuring the NFS server

On our network, we will use an Ubuntu machine as a server and we will show you how to access the files from another Linux client. First, let's install and configure the server, as follows:

1. We will install the `nfs-kernel-server` package using the `apt` command, as shown in the following screenshot:

```
packt@neptune:~$ sudo apt install nfs-kernel-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  keyutils libnfsidmap2 libtirpc-common libtirpc3 nfs-common rpcbind
Suggested packages:
  open-iscsi watchdog
The following NEW packages will be installed:
  keyutils libnfsidmap2 libtirpc-common libtirpc3 nfs-common
  nfs-kernel-server rpcbind
0 upgraded, 7 newly installed, 0 to remove and 0 not upgraded.
Need to get 503 kB of archives.
After this operation, 1.936 kB of additional disk space will be used.
Do you want to continue? [Y/n] _
```

Figure 8.25 – Installing the nfs-kernel-server package on Ubuntu

2. Once the package is installed, you will have to start the service using the `systemctl` command, and then we will check its status. The command can be seen here:

```
packt@neptune:~$ sudo systemctl start nfs-kernel-server.service
packt@neptune:~$ sudo systemctl status nfs-kernel-server.service
● nfs-server.service - NFS server and services
     Loaded: loaded (/lib/systemd/system/nfs-server.service; enabled; vendor>
     Active: active (exited) since Sat 2020-11-14 10:04:46 EET; 3min 29s ago
   Main PID: 404014 (code=exited, status=0/SUCCESS)
      Tasks: 0 (limit: 9142)
     Memory: 0B
     CGroup: /system.slice/nfs-server.service
```

Figure 8.26 – Starting the NFS server and checking its status

For a more efficient workflow inside our network, we will add all the directories to be exported inside a single parent directory on the server. As a general rule, you have two options for creating shared directories. You could export the already existing `/home` directory for all clients on the network, or you could make a dedicated shared directory from the start. You could create a new directory starting from the root, but you could also create your shared directory inside specific directories such as `/var`, `/mnt`, or `/srv`—it's your choice.

We will create a new directory called `/home/export/shares` inside our `/home` directory using the commands shown in the following screenshot (make sure that you are already inside your `/home` directory if you want to use the command as is):

```
packt@neptune:/home$ sudo mkdir -p export/shares
packt@neptune:/home$ ls
export  packt
packt@neptune:/home$
```

Figure 8.27 – Creating the /home/export/shares directory

Set permissions to `777` as we will not use **Lightweight Directory Access Protocol** (**LDAP**) authentication on the following example:

```
packt@neptune:/home/export/shares$ sudo chmod 777 /home/export/shares/
packt@neptune:/home/export/shares$ ls -la
total 8
drwxrwxrwx 2 root root 4096 nov 14 23:14 .
drwxr-xr-x 3 root root 4096 nov 14 12:29 ..
-rwxrwxrwx 1 root root    0 nov 14 23:14 test-file
```

Figure 8.28 – Setting permissions for the shared directory

The directory is now ready to be exported. We will add configuration options to the `/etc/exports` file. There are three configuration files for NFS (`/etc/default/nfs-kernel-server`, `/etc/default/nfs-common`, and `/etc/exports`) but we will

only alter one of them. In the following screenshot, you will see the two files inside `/etc/default` and the default contents of the `/etc/exports` configuration file:

```
packt@neptune:/etc/default$ ls
acpi-support     chrony          intel-microcode      nss
acpid            console-setup   irqbalance           openvpn
alsa             crda            kerneloops           rpcbind
amd64-microcode  cron            keyboard             rsync
anacron          dbus            locale               saned
apport           grub            networkd-dispatcher  ssh
avahi-daemon     grub.d          nfs-common           ufw
bsdmainutils     im-config       nfs-kernel-server    useradd
packt@neptune:/etc/default$ cat /etc/exports
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_s
ubtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
```

Figure 8.29 – Contents of the /etc/exports configuration file

Open the `/etc/exports` file with your preferred text editor and edit it according to your configuration. Inside the file, we will add lines for each shared directory. Before doing that, you might have noticed that inside the file there are two types of directives: one for NFS versions 2 and 3, and one for version 4. For more details about the differences between those versions, we encourage you to consult the following document: https://archive.fosdem.org/2018/schedule/event/nfs3_to_nfs4/attachments/slides/2702/export/events/attachments/nfs3_to_nfs4/slides/2702/FOSDEM_Presentation_Final_pdf.pdf.

Now that you know the differences between those versions, let's start editing the configuration file. We will add a new line containing the directory, the IP address of the client, and configuration options. The general syntax is as follows:

```
/first/path/to/files  IP(options)[ IP(options) IP(options)]

/second/path/to/files IP(options)[ IP(options) IP(options)]
```

We share only one directory, hence the single line added. If you want all clients on the network to access the shares, you can add the subnet class (for example, `192.168.0.0/24`). Or, if you want only specific clients to access the shares, you should

add their IPs. More clients will be added on the same line, separated by spaces. There are many options that you can add, and for a full list, we advise that you consult https://linux.die.net/man/5/exports or the local manual file with the `man exports` command. In our file, we added the following options:

- `rw` for both read and write access
- `sync` to force write changes to disk (this reduces the speed, though)
- `no_subtree_check` to prevent subtree checking, which is mainly a check to see if the file is still available before the request

In a nutshell, here is the line we added to the file:

```
/home/export/shares   192.168.0.0/24(rw,sync,no_subtree_check)
```

After saving and closing the file, restart the service with the following command:

```
sudo systemctl restart nfs-kernel-server.service
```

Then, apply the configuration with the following command:

```
sudo exportfs -a
```

The `-a` options export all directories without specifying a path.

Once the service is restarted and running, you can set up a firewall to allow NFS access. For this, it is extremely useful to know that the port NFS is using by default is port `2049`. As we allow all the systems from our network to access the shares, we will add the following new rule to the firewall:

```
packt@neptune:/home/export$ sudo ufw app list
Available applications:
  CUPS
  OpenSSH
packt@neptune:/home/export$ sudo ufw allow nfs
Rule added
Rule added (v6)
packt@neptune:/home/export$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
OpenSSH                    ALLOW       Anywhere
2049                       ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
OpenSSH (v6)               ALLOW       Anywhere (v6)
2049 (v6)                  ALLOW       Anywhere (v6)
```

Figure 8.30 – Adding a firewall rule to allow NFS

In the preceding screenshot, you can see that after adding the new rule using the `sudo ufw allow nfs` command, port `2049` was added to the list of allowed rules. The basic configuration on the server is now complete. In order to access the files, you will need to configure the clients too.

First, you will have to install NFS on the client by using the following command:

```
sudo apt install nfs-common
```

Now that the needed packages are installed on the client too, we can create directories to mount the shares, as follows:

```
alexandru@asus:/home$ sudo mount 192.168.0.244:/home/export/shares /home/shares
alexandru@asus:/home$ df -h
Filesystem                          Size  Used Avail Use% Mounted on
udev                                3,9G     0  3,9G   0% /dev
tmpfs                               787M  1,8M  786M   1% /run
/dev/sda2                           110G   11G   93G  11% /
tmpfs                               3,9G     0  3,9G   0% /dev/shm
tmpfs                               5,0M  4,0K  5,0M   1% /run/lock
tmpfs                               3,9G     0  3,9G   0% /sys/fs/cgroup
/dev/loop0                           97M   97M     0 100% /snap/core/9665
/dev/loop2                           56M   56M     0 100% /snap/core18/1885
/dev/loop1                           98M   98M     0 100% /snap/core/10185
/dev/loop4                          256M  256M     0 100% /snap/gnome-3-34-1804/36
/dev/loop5                           56M   56M     0 100% /snap/core18/1932
/dev/loop6                          218M  218M     0 100% /snap/gnome-3-34-1804/60
/dev/loop8                           63M   63M     0 100% /snap/gtk-common-themes/1506
/dev/loop9                           50M   50M     0 100% /snap/snap-store/467
/dev/loop10                          51M   51M     0 100% /snap/snap-store/481
/dev/loop11                          31M   31M     0 100% /snap/snapd/9721
/dev/loop13                         150M  150M     0 100% /snap/zoom-client/108
/dev/loop14                          30M   30M     0 100% /snap/snapd/8542
/dev/sda1                           511M   12M  500M   3% /boot/efi
tmpfs                               787M   32K  787M   1% /run/user/125
/dev/loop15                         147M  147M     0 100% /snap/slack/31
/dev/loop12                         144M  144M     0 100% /snap/slack/32
/dev/loop3                          232M  232M     0 100% /snap/zoom-client/113
tmpfs                               787M  8,0K  787M   1% /run/user/1000
192.168.0.244:/home/export/shares   468G  9,0G  436G   3% /home/shares
```

Figure 8.31 – Mounting shares on the client and checking the status

In the preceding screenshot, we mounted the shares on the server and on the client using the `mount` command and checked to see if everything went well with the `df -h` command. The new mount is shown last in the `df` command's output.

At this point, we have finished the setup for the NFS shares. We now need to test the configuration to prove that it works.

Testing the NFS setup

Once the setup is finished on both server and client, you can test to see if everything works according to your expectations. In our test, we created a file named `test-file` using the root user on the server, and a file called `file` using the regular user on the client machine. Here is the output, showing the result on both the client and the server, running Ubuntu:

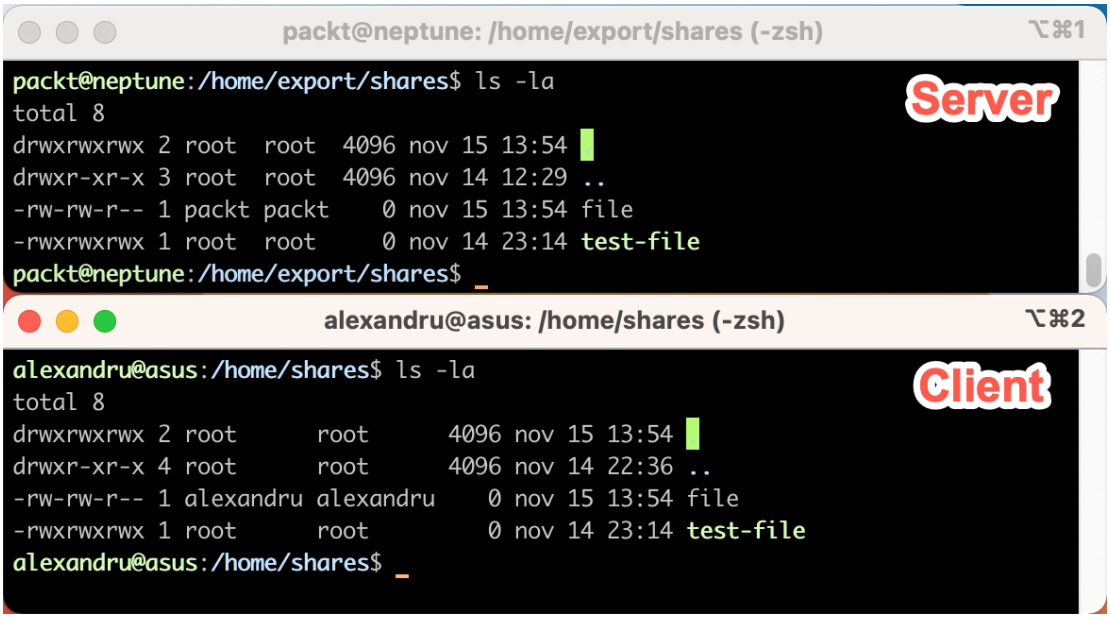

Figure 8.32 – Testing the NFS on Ubuntu server and client

There you go: the NFS server and client are working just fine. You can use the **graphical user interface** (**GUI**) on the client too, not just the **command-line interface** (**CLI**) as we did. We are connected through **Secure Shell** (**SSH**) to both the client and server from a local Mac machine, hence the interface from the preceding screenshot.

In the next section, we will show you how to configure a Samba/CIFS share, to be accessed by Windows clients on the network.

Setting up a Samba file server

The Samba server allows you to share files over a network where clients use different operating systems, such as Windows, macOS, and Linux. In this section, we will set up a Samba server on Ubuntu and access shares from different operating systems on the

network. The SMB/CIFS protocol is developed by Microsoft and, for more details, you can visit their developer pages at https://docs.microsoft.com/en-us/windows/win32/fileio/microsoft-smb-protocol-and-cifs-protocol-overview. Some information about the **Server Message Block** (**SMB**)/CIFS protocol can be found in *Chapter 7*, *Networking with Linux*, too.

Installing and configuring Samba

The installation procedure has the following steps:

1. First, we will install Samba on the system, using the `sudo apt install samba` command. The output should be similar to the one in the following screenshot:

```
packt@neptune:~$ sudo apt install samba
[sudo] password for packt:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  attr ibverbs-providers libcephfs2 libibverbs1 librados2 librdmacm1
  python3-crypto python3-dnspython python3-gpg python3-ldb python3-markdown
  python3-packaging python3-pygments python3-pyparsing python3-samba
  python3-tdb samba-common samba-common-bin samba-dsdb-modules
  samba-vfs-modules tdb-tools
Suggested packages:
  python-markdown-doc python-pygments-doc ttf-bitstream-vera
  python-pyparsing-doc bind9 bind9utils ctdb ldb-tools smbldap-tools winbind
  heimdal-clients
The following NEW packages will be installed:
  attr ibverbs-providers libcephfs2 libibverbs1 librados2 librdmacm1
  python3-crypto python3-dnspython python3-gpg python3-ldb python3-markdown
  python3-packaging python3-pygments python3-pyparsing python3-samba
  python3-tdb samba samba-common samba-common-bin samba-dsdb-modules
  samba-vfs-modules tdb-tools
0 upgraded, 22 newly installed, 0 to remove and 0 not upgraded.
Need to get 10,0 MB of archives.
After this operation, 66,5 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Figure 8.33 – Installing Samba on Ubuntu

Once Samba is installed, you need to create a directory to share on the network. Let's begin!

2. We will make a new directory inside our `home` directory using the `mkdir` command, as follows:

```
mkdir /home/packt/shares
```

3. After creating the new directory, we will edit the Samba configuration file. This is located under `/etc/samba/smb.conf`. The configuration file has two major sections: a `[global]` section with general configuration settings, and a `[shares]` section that configures the shares' behavior. We can back up the original `smb.conf` file and add the configuration settings inside a new file, like this:

```
sudo cp /etc/samba/smb.conf /etc/samba/smb.conf.original
```

4. Inside the configuration file, we will edit the `[global]` section with directives, as follows:

```
[global]
interfaces = 127.0.0.0/8 eno1
bind interfaces only = yes
usershare allow guests = yes
```

We will add details about the shared directory on our server. In our case, it is called *sambashares* and is located under `/sambashares`. We will add some directives, such as a short description inside the `comment` slot, and `read only` and `browsable` information, as shown here:

```
[shares]
        comment = Shares on Neptune
        path = /sambashares/
        read only = no
        browsable = yes
```

Figure 8.34 – Adding info about your shared directories inside smb.conf

5. Once the modifications are done, we will restart the service with the following command:

```
sudo systemctl restart smbd.service
```

6. Furthermore, we will adjust the firewall rules so that Samba will be allowed, using the following command:

```
sudo ufw allow samba
```

7. Once the firewall is set up, you can run the `testparm` command to test that the configuration has been done correctly, with no errors, as follows:

```
packt@neptune:~$ testparm
Load smb config files from /etc/samba/smb.conf
Loaded services file OK.
Server role: ROLE_STANDALONE

Press enter to see a dump of your service definitions
```

Figure 8.35 – Testing the Samba configuration

The output shows `Loaded services file OK`, which means that the configuration files have no syntax errors. After the system is restarted and the firewall configured, we can proceed to setting up a Samba password for users who can access the shares.

Creating Samba users

Each Samba server needs to have specific users that can access the shared directories and files. Those users need to be both Samba and system users, as this is necessary for users to be able to authenticate and to read and write system files. Let's assume that you need to create a local share for your small business or family group.

By creating local users specifically for using the Samba shares, you don't need them to act like actual users as they only need to be able to access the shares.

Nevertheless, we will create some more users that will be able to access the shares. We will give some generic names to those users, such as `mike`, `tom`, `jane`, and `anna`. You have two scenarios with the users. You can create regular users, which will create a dedicated home directory for each user, followed by their respective groups; or, in the second scenario, you can create different home directories for each, for better Samba management. Furthermore, you can add them to the `sambashare` group, a default group created by Samba when first installed, without giving them any login access. We will use the second scenario in our example, because having dedicated Samba home directories that are separate from the user's default home directories will facilitate better Samba resource management. We'll proceed as follows:

1. First, we will create home directories for each user, which will make things easier to manage, as everything will be in the same `/sambashares` directory. We will show

you how to create the user `jane`, as the process for creating other users is the same. Create a home directory for `jane` under the `/sambashares` directory, but first we will add a new group owner to the parent directory, as follows:

```
packt@neptune:~$ sudo chown :sambashare /sambashares/
packt@neptune:~$ sudo mkdir /sambashares/jane
```

Figure 8.36 – Creating a home directory for the new user

2. Once the password and the basic configuration are finished, we can try to connect to the shares from another system. Next, we will add the user `jane`, with the home directory set to `/sambashares/jane`, with no login shell or home directory created by default, and add the user to the `sambashare` group right from the start. Next, we are prompted to enter a password and more details about the user, as illustrated in the following screenshot:

```
packt@neptune:~$ sudo adduser --home /sambashares/jane --no-create-home --she
ll /usr/sbin/nologin --ingroup sambashare jane
Adding user `jane' ...
Adding new user `jane' (1001) with group `sambashare' ...
Not creating home directory `/sambashares/jane'.
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for jane
```

Figure 8.37 – Adding a new user, called jane

3. Next, we will set owner permissions on the home directory accordingly. We set the permissions to `2770`, in order for the new files and directories created inside to inherit the group ownership of the parent directory. The code for this can be seen here:

```
packt@neptune:~$ sudo chown jane:sambashare /sambashares/jane/
packt@neptune:~$ sudo chmod 2770 /sambashares/jane/
```

Figure 8.38 – Setting ownership and permissions for the new directory

Setting the ownership, as shown previously, ensures that if any other user creates a directory inside the shared directory, the owner will be able to read and write to it.

4. The next step is to add the new user to the Samba server. Samba has its own register where users are stored, thus it is able to authenticate to the server. After

adding the user to Samba (using the `-a` option), they must be enabled (using the `-e` option). The following two commands will be used:

```
packt@neptune:~$ sudo smbpasswd -a jane
New SMB password:
Retype new SMB password:
Added user jane.
packt@neptune:~$ sudo smbpasswd -e jane
Enabled user jane.
```

Figure 8.39 – Adding and enabling the new user with Samba

We will do the same for the other users, `mike`, `tom`, and `anna`, but we will not show this here. Do the same if you want more users on your system. You can also create an administrative account for all the shares. We will create one, named `adminsamba`, as follows:

```
packt@neptune:~$ sudo mkdir /sambashares/everyone
[sudo] password for packt:
packt@neptune:~$ sudo adduser --home /sambashares/everyone --no-create-home -
-shell /usr/sbin/nologin --ingroup sambashare adminsamba
Adding user `adminsamba' ...
Adding new user `adminsamba' (1005) with group `sambashare' ...
Not creating home directory `/sambashares/everyone'.
packt@neptune:~$ sudo chown adminsamba:sambashare /sambashares/everyone/
packt@neptune:~$ sudo chmod 2770 /sambashares/everyone/
packt@neptune:~$ sudo smbpasswd -a adminsamba
New SMB password:
Retype new SMB password:
Added user adminsamba.
packt@neptune:~$ sudo smbpasswd -e adminsamba
Enabled user adminsamba.
```

Figure 8.40 – Adding the administrative user for Samba shares

5. Once users have been added, we can modify the configuration file to add details about each user's personal shares. Here are the details for the shares of user `jane`:

```
[jane]
      comment = Jane's shares on Neptune
      force create mode = 0660
      force directory mode = 02770
```

```
        path = /sambashares/jane
        read only = No
        valid users = jane @adminsamba
```

Each user's section is similar to the preceding one. Let's explain what each line is, as follows:

- On the first line, in square brackets, is the *shares name*.
- `comment` is a short comment for the share, with minimal details.
- `force create mode` forces the permissions for each file created inside the shares.
- `force directory mode` forces the permissions for each directory created.
- `path` is the absolute path of the shared directory.
- `read only` determines if users can write to the shares or not.
- `valid users` is a list of users with share access.

After editing all the required sections, run the `testparm` command to check that everything is fine and that there are no syntax errors. If there are no errors, restart the service with the `systemctl` command.

In the next section, we will show you how to access the Samba shares from different systems on the network.

Accessing the Samba shares

On your network, you can access the shares from Linux, macOS, or Windows. We will show you how to access those shares from each of those operating systems. Let's start with Linux first.

Accessing Samba shares from Linux

On a Linux system, you need to install the Samba client first. We will assume that you will have an Ubuntu Linux client, but the steps are similar for other Linux distributions too. On Ubuntu, first install the Samba client with the following command, but make sure that your repositories are updated before you do this:

```
sudo apt install smbclient
```

If you are running a CentOS client, install the Samba client with the following command:

```
sudo yum install samba-client
```

To make everything a little bit challenging, worthy of a Linux master, we will show you how to access the Samba shares in Linux using the CLI only. We will let you find out for yourselves how to access them from the GUI. To access the shares from the CLI, use the `smbclient` tool. For example, let's access the shares of user `jane` from our Asus machine. Remember that the shares are on our Neptune Ubuntu server. We will use our local IP for the Neptune server and will run the following code:

```
alexandru@asus:~$ smbclient //192.168.0.244/jane -U jane
Enter WORKGROUP\jane's password:
Try "help" to get a list of possible commands.
smb: \> _
```

Figure 8.41 – Accessing Jane's shares from an Ubuntu client

In the preceding screenshot, you can see that the Samba access was successful, and the user `jane` managed to access the Samba shares on the server. We used the `–U` option followed by the username, to specify the name of the user we are connecting with.

Similar to the preceding process, we will now try to access the shares from another server on our network, as follows:

```
[packt@jupiter ~]$ smbclient //192.168.0.244/tom -U tom
Unable to initialize messaging context
Enter SAMBA\tom's password:
Try "help" to get a list of possible commands.
smb: \> _
```

Figure 8.42 – Accessing Tom's shares from a CentOS client

In the preceding screenshot, we accessed Tom's shares from a CentOS client (our Jupiter server) on the same network. You can see that the software tool is the same, even though the name of the package initially installed was different. The syntax is identical on both Linux distributions.

We will next show you how to access the shares from a macOS client, using the GUI.

Accessing Samba shares from macOS

On a macOS system (based on macOS 11.0.1 Big Sur), the procedure of accessing Samba shares is as follows:

1. Open a new *Finder* window first, and then select the **Go** menu entry and the **Connect to Server** option. In the new window, enter the server address shown as follows. We used the user `jane` as an example:

   ```
   smb://jane@192.168.0.244/jane
   ```

2. Then, save the server's address using the small plus (**+**) sign button in the bottom-left corner. The next step is to click the **Connect** button.
3. Once the details are added, click on the **Connect** button. Next, a pop-up window warns us that a connection to a server is in progress and asks us to click **Connect** once more to proceed.
4. After that, a window requesting the user's credentials appears. Enter the password for the user you want to connect with and click the **Connect** button.
5. After adding the credentials, the shares for user `jane` are available in your *Finder* window, inside a new tab.

Connecting to the shares in macOS is not a difficult task, as you can see from the preceding screenshots. Next, we will show you how to connect to the shares from a Microsoft Windows machine.

Accessing Samba shares from Windows

We will use a Windows 10 machine on our network to show you how to connect to the Samba shares, as follows:

1. First, open a new **File Explorer** window, and then right-click on **This PC**. In the menu that appears, select the **Add a network location** option by clicking on it.
2. Once the option is selected, the window that follows gives you the option to choose a custom network location. Select it and click **Next**. In the next window, you will have to add the location of your share. It must be in a Windows-specific syntax (Windows uses backslashes compared to the regular slash in Linux). We connected to the shares belonging to user `anna`, thus we used their credentials and shares address. An example is given, as follows:

```
\\192.168.0.244\anna
```

3. Once the address is given, select **Next** and move to the following window. It will ask you to add a name for the new location. Choose a name that you desire, or leave the default text given by the system. Click **Next**. The next window will ask you for the username and password of the user. Enter the credentials and click the **OK** button. Don't take into consideration the **Access is denied** warning as this will disappear once you enter the credentials.

4. Once you click **OK**, a window showing the contents of Anna's shares will open. You can now start adding files to your shares, as accessing them from every operating system on your network works as expected.

Simply setting up Samba is not enough—we need to think about securing the shares. This is why, in the following section, we will show you how to configure basic Samba security.

Securing Samba shares

As Samba is based on SMB/CIFS, it borrows from its legacy. Security-wise, there are two security levels inherited from it: a *user level* and a *share level*. Samba doubles down on security, by implementing four user-level modes and one shared-level mode.

Among the four user-level modes, we will emphasize one: `security = user`; the other modes are `security = domain`, `security = ADS` and `security = server`. The first mode will enable user authentication by default and is one of the most useful ones to implement, and not that complicated. For hardened security and better management, you can install the `libpam-winbind` package to synchronize the system users with the Samba user database, as follows:

```
sudo apt install libpam-winbind
```

Then, you can add a new line to the `[shares]` section inside the `smb.conf` file that would not allow guest access, as follows:

```
guest ok = no
```

Then, you restart the Samba service, as follows:

```
sudo systemctl restart smbd.service
```

The Samba suite provides the **Network Basic Input/Output System** (**NetBIOS**) name server, which can be useful if no Windows domain server is running. The following screenshot illustrates this:

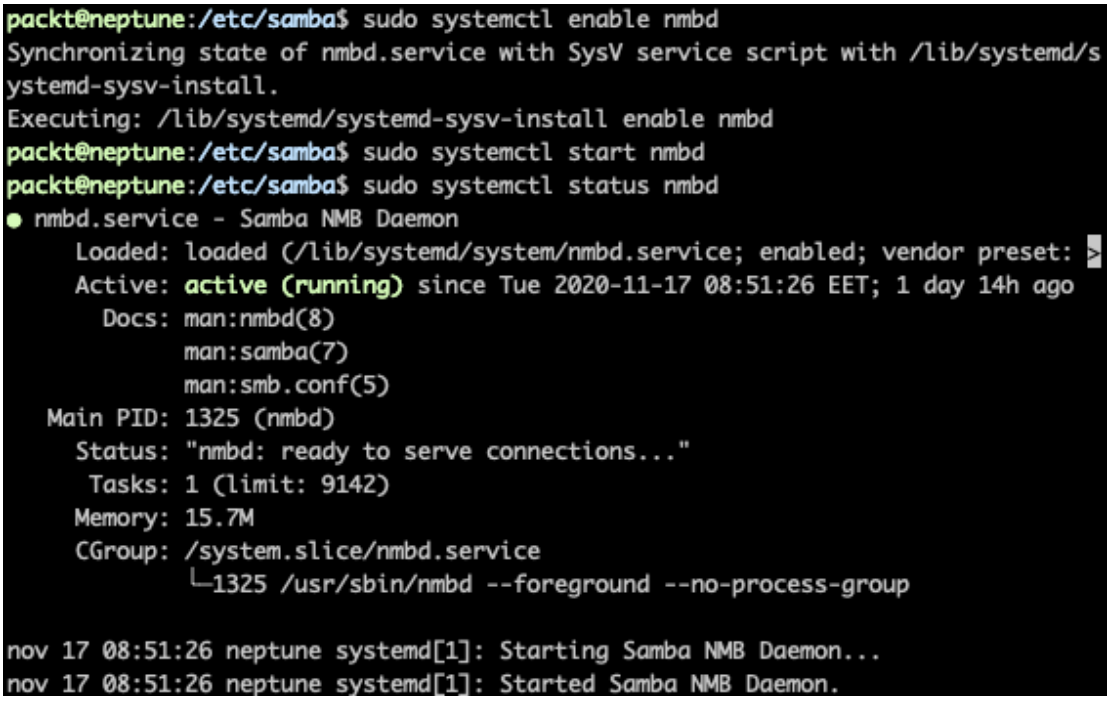

Figure 8.49 – Enabling and starting the NetBIOS name server

Furthermore, you can install an **AppArmor** profile for Samba on Ubuntu, but it must be adapted and configured according to your Samba setup. For more details on AppArmor, visit *Chapter 9*, *Securing Linux*. In short, at this point, you could install the AppArmor profiles (the package contains profiles for many binaries) and edit the specific file according to your Samba server configuration. To install the profiles, use the following command:

```
sudo apt install apparmor-profiles apparmor-utils
```

Once the profiles are installed, they are running in `complain` mode by default. To make it run in `enforce` mode, you will need to modify the configuration file in `/etc/apparmor.d/usr.sbin.smbd`. Modify the profile according to your Samba config after you go through *Chapter 9*, *Securing Linux*.

For more details about Samba and the vast options available, we advise you to visit the official wiki page at https://wiki.samba.org/index.php/Main_Page. Samba is a very useful protocol as you can configure it to work on every known operating system, giving you a versatile way to share files on a network.

In the following section, we will show you how to install and configure an FTP server.

Setting up an FTP server

The FTP is a network protocol used to transfer files across computers, using the **Transmission Control Protocol** (**TCP**) for downloading files. Details about the protocol were given to you in *Chapter 7*, *Networking with Linux*. It is now time to install and set up a **file transfer server** (**FTS**) on our Ubuntu machine. There are many FTP software tools, but the most used ones are `proFTPD` and `vsftpd`. We will show you how to install and configure `vsftpd` in this section.

FTP is not doing great security-wise. As a matter of fact, it is not providing any encryption at all, and this is mainly the reason why it stopped being used as a preferred file transfer protocol. It is mostly used to support legacy applications, but there are nevertheless ways to add some layers of extra security for file sharing, by using secure and encrypted tools based on SSH, such as `sftp`, `scp`, and `rsync`. FTP is still used for sharing public documents such as open source repositories, but not for private documents.

The FTP protocol works as a client-server model, where the server is called an FTP daemon and can be managed as **anonymous** or **authenticated**. Both ways are insecure, as the transfer between the client and the server is done in plain text, with no encryption. Keep this in mind when you want to transfer personal files.

Installing and configuring vsftpd

As we said before, we will install the `vsftpd` package on our system. This can be easily done by using the `apt` command, shown as follows:

```
sudo apt install vsftpd
```

Once the package is installed, you should allow FTP traffic through your firewall by adding new rules to it. Remember that the ports the FTP protocol is using are ports `20` and `21` by default. Furthermore, you can enable access for port `990` for TLS-enabled traffic, as illustrated in the following screenshot:

```
packt@neptune:~$ sudo ufw allow 20/tcp
Rule added
Rule added (v6)
packt@neptune:~$ sudo ufw allow 21/tcp
Rule added
Rule added (v6)
packt@neptune:~$ sudo ufw allow 990/tcp
Rule added
Rule added (v6)
packt@neptune:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
OpenSSH                    ALLOW       Anywhere
2049                       ALLOW       Anywhere
Samba                      ALLOW       Anywhere
20/tcp                     ALLOW       Anywhere
21/tcp                     ALLOW       Anywhere
990/tcp                    ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
OpenSSH (v6)               ALLOW       Anywhere (v6)
2049 (v6)                  ALLOW       Anywhere (v6)
Samba (v6)                 ALLOW       Anywhere (v6)
20/tcp (v6)                ALLOW       Anywhere (v6)
21/tcp (v6)                ALLOW       Anywhere (v6)
990/tcp (v6)               ALLOW       Anywhere (v6)
```

Figure 8.50 – Allowing traffic through the FTP ports

After setting up the firewall, you will need to create a backup of the primary configuration file, which is `/etc/vsftpd.conf`. We will copy this under another name, as follows:

```
sudo cp /etc/vsftpd.conf /etc/vsftpd.conf.original
```

You will now be able to modify the configuration file and have a backup version in case things go south. The next step is to start the `vsftpd` service using the `systemctl` command. First, we will start it, then check the status, and then enable it to automatically start at startup time (see *Figure 8.50*).

Before altering the configuration file, we will have to create the users that will access the FTP files, together with a dedicated directory for FTP data storage, as follows:

```
packt@neptune:~$ sudo systemctl start vsftpd
packt@neptune:~$ sudo systemctl status vsftpd
● vsftpd.service - vsftpd FTP server
     Loaded: loaded (/lib/systemd/system/vsftpd.service; enabled; vendor pre>
     Active: active (running) since Thu 2020-11-19 11:50:40 EET; 33min ago
   Main PID: 20690 (vsftpd)
      Tasks: 1 (limit: 9142)
     Memory: 664.0K
     CGroup: /system.slice/vsftpd.service
             └─20690 /usr/sbin/vsftpd /etc/vsftpd.conf

nov 19 11:50:40 neptune systemd[1]: Starting vsftpd FTP server...
nov 19 11:50:40 neptune systemd[1]: Started vsftpd FTP server.
packt@neptune:~$ sudo systemctl enable vsftpd
Synchronizing state of vsftpd.service with SysV service script with /lib/syst
emd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable vsftpd
```

Figure 8.51 – Starting and enabling the vsftpd service

Let's assume that you want a small FTP server to transfer public files such as general documentation for one of your open source projects, with your co-workers or a small group. For this, you will need to create system users for your team onto the server. As an example, we will create only one new user, but you can create as many as you need. Our new user will be called `dan`, as illustrated in the following screenshot:

```
packt@neptune:~$ sudo adduser dan
[sudo] password for packt:
Adding user `dan' ...
Adding new group `dan' (1002) ...
Adding new user `dan' (1006) with group `dan' ...
Creating home directory `/home/dan' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for dan
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n]
```

Figure 8.52 – Adding a new user called dan to use the FTP shares

Once the user is created, you can add a sub-directory inside its `home` directory that will be used only for FTP shares. This is a good practice security-wise, as FTP should be restricted to a specific directory only. We will call that directory `ftpshares`. The following screenshot shows the succession of commands that create the new directory, change the ownership to `nobody`/`nogroup`, remove the write permission, and then show the final result:

```
packt@neptune:~$ pwd
/home/packt
packt@neptune:~$ cd ..
packt@neptune:/home$ ls
dan  export  packt
packt@neptune:/home$ cd dan/
packt@neptune:/home/dan$ ls
packt@neptune:/home/dan$ sudo mkdir ftpshares
packt@neptune:/home/dan$ ls
ftpshares
packt@neptune:/home/dan$ sudo chown nobody:nogroup /home/dan/ftpshares/
packt@neptune:/home/dan$ sudo chmod a-w /home/dan/ftpshares/
packt@neptune:/home/dan$ sudo ls -la /home/dan/ftpshares/
total 8
dr-xr-xr-x 2 nobody nogroup 4096 nov 19 13:58 .
drwxr-xr-x 3 dan    dan     4096 nov 19 13:58 ..
```

Figure 8.53 – Creating a new directory, changing permission and ownership, and testing the result

Next, we will create a new directory for the file uploads, and we will call it `uploads`. Then, we will add a test file inside it, as follows:

```
packt@neptune:/home/dan$ sudo mkdir /home/dan/ftpshares/uploads
[sudo] password for packt:
packt@neptune:/home/dan$ sudo chown dan:dan /home/dan/ftpshares/uploads
packt@neptune:/home/dan$ sudo ls -la /home/dan/ftpshares/
total 12
dr-xr-xr-x 3 nobody nogroup 4096 nov 19 18:57 .
drwxr-xr-x 3 dan    dan     4096 nov 19 13:58 ..
drwxr-xr-x 2 dan    dan     4096 nov 19 18:57 uploads
packt@neptune:/home/dan$ echo "some text here..." | sudo tee /home/dan/ftpsha
res/uploads/test-file.txt
some text here...
```

Figure 8.54 – Creating the uploads directory

By now, most of the preparatory work is done. All you have to do is to start configuring the FTP server by altering the `/etc/vsftpd.conf` configuration file. Inside the file, you will alter some of the directives, such as the following:

- `anonymous_enable`: To allow or not anonymous login or not—we will set it to `NO`.
- `local_enables`: To allow local users to log in—we will set it to `YES`.
- `write_enables`: To let the user upload files to the shares—we will set to `YES`.
- `chroot_local_user`: To prevent a user accessing files outside the directory.
- `user_sub_token`: To add the username of any allowed user, present or future, to be able to use the shares.
- `local_root`: Similar to the preceding description
- We will limit the range of passive ports with the `pasv_min_port` and `pasv_max_port` directives, by setting minimum to `40000` and maximum to `50000`; for this to work, we will need to add a new rule, `sudo ufw allow 40000:50000/tcp`, to the firewall.
- You could optionally add a user list with the names of the users, but we will not do that in this example. The directives to use are `userlist_enable=YES`, `userlist_file=/etc/vsftpd.userlist`, and `userlist_deny=NO`.

The following is an excerpt of all the directives we added inside the configuration file:

```
listen=NO
listen_ipv6=YES
anonymous_enable=NO
local_enable=YES
write_enable=YES
local_umask=022
dirmessage_enable=YES
use_localtime=YES
xferlog_enable=YES
connect_from_port_20=YES
chroot_local_user=YES
secure_chroot_dir=/var/run/vsftpd/empty
pam_service_name=vsftpd
rsa_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem
rsa_private_key_file=/etc/ssl/private/ssl-cert-snakeoil.key
ssl_enable=NO
user_sub_token=$USER
local_root=/home/$USER/ftpshares
pasv_min_port=40000
pasv_max_port=50000
```

Figure 8.55 – Contents of our /etc/vsftpd.conf file

Simply installing and configuring the `vsftpd` service is not enough, and before declaring it as fit to use, run a couple of simple testing procedures, as shown in the following section.

Testing vsftpd

Now that the FTP server is already configured, we go on and test to see if everything works as expected. The server is set up on our Ubuntu Neptune machine, and we will try to connect from another system on the network.

In the following screenshot, we will use the `ftp` command, using the `-p` option. This option ensures access in passive mode, which allows the use of `ftp` even if a firewall prevents outside connections.

While trying to connect to the FTP server as anonymous, an error is shown. This is because our FTP configuration does not allow anonymous users to log in. The details are shown in the following screenshot:

```
alexandru@asus:~$ ftp -p 192.168.0.244
Connected to 192.168.0.244.
220 (vsFTPd 3.0.3)
Name (192.168.0.244:alexandru):
331 Please specify the password.
Password:
530 Login incorrect.
Login failed.
ftp> _
```

Figure 8.56 – Anonymous login to the FTP server is not allowed

Let's try with another user, one that is not on the `vsftpd` list. We will try to connect using our `packt` user. The details are shown in the following screenshot:

```
alexandru@asus:~$ ftp -p 192.168.0.244
Connected to 192.168.0.244.
220 (vsFTPd 3.0.3)
Name (192.168.0.244:alexandru): packt
331 Please specify the password.
Password:
500 OOPS: cannot change directory:/home/packt/ftpshares
Login failed.
421 Service not available, remote server has closed connection
ftp> _
```

Figure 8.57 – Trying to connect to the FTP server with a regular user

Now, let's try to connect with user `dan`. The output will show us that the user is accepted and can successfully connect to the FTP server. Remember that we created a directory called `uploads` inside the `ftpshares` directory of user `dan`. This directory has a `test-file.txt` inside it. Let's attempt to download the file to our local system. The file will be copied inside your present working directory. The details are shown in the following screenshot:

```
alexandru@asus:~$ ftp -p 192.168.0.244
Connected to 192.168.0.244.
220 (vsFTPd 3.0.3)
Name (192.168.0.244:alexandru): dan
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd uploads
250 Directory successfully changed.
ftp> ls
227 Entering Passive Mode (192,168,0,244,162,242).
150 Here comes the directory listing.
-rw-r--r--    1 0        0              18 Nov 19 18:58 test-file.txt
226 Directory send OK.
ftp> get test-file.txt
local: test-file.txt remote: test-file.txt
227 Entering Passive Mode (192,168,0,244,167,64).
150 Opening BINARY mode data connection for test-file.txt (18 bytes).
226 Transfer complete.
18 bytes received in 0.00 secs (106.5341 kB/s)
ftp> exit
221 Goodbye.
```

Figure 8.58 – Connecting with user dan and downloading the already existing file

To verify if the file has been downloaded, run the `ls -l` command inside your present working directory, as follows:

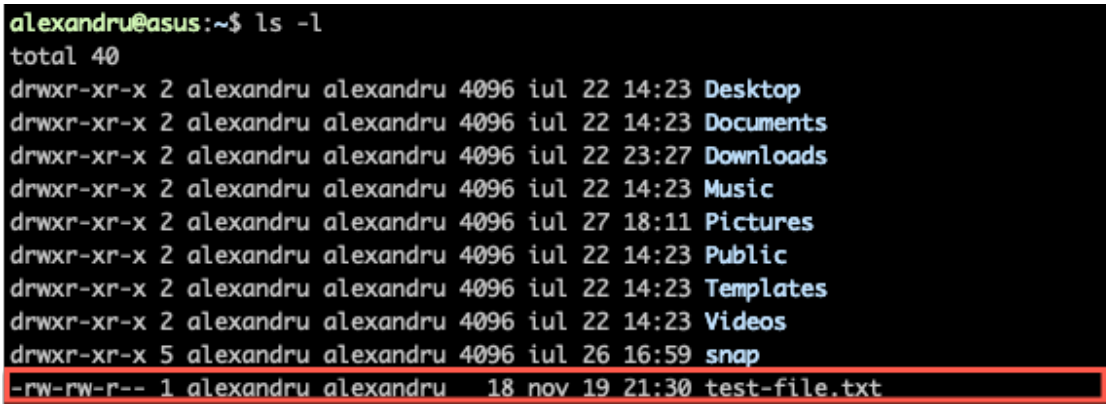

Figure 8.59 – Verifying if the download was successful on the client

The output from the preceding screenshot shows that the file has been successfully downloaded from the FTP server to the client. If you do not want to use the command line, there are plenty of GUI clients available for all known operating systems. One of the most known and used tools is FileZilla. Try using it on your own, by providing the details set up in this section.

Securing FTP

As we said at the beginning of this section, the FTP protocol does not use encryption and sends data in plain text, including the user credentials. This might be a potential danger, even if used only on your local network. A relatively easy solution is to use a private **Secure Sockets Layer** (**SSL**) certificate for TLS/SSL-enabled transactions. We will guide you through the process of creating a private certificate and modifying the main configuration file. We will use the `openssl` tool to create the certificate. The following screenshot shows the command that generates the new **Rivest-Shamir-Adleman** (**RSA**) certificate, which will be valid for 1 year (365 days), will use 2,048 bits, and will be saved under `/etc/ssl/private/private-ftp-cert.pem`:

```
packt@neptune:/etc/ssl$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2
048 -keyout /etc/ssl/private/private-ftp-cert.pem -out /etc/ssl/private/priva
te-ftp-cert.pem
Generating a RSA private key
..........+++++
.....................................................................................
+++++
writing new private key to '/etc/ssl/private/private-ftp-cert.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:WA
Locality Name (eg, city) []:Seattle
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Home Computer
Organizational Unit Name (eg, section) []:Sysadmin
Common Name (e.g. server FQDN or YOUR name) []:192.168.0.244
Email Address []:
```

Figure 8.60 – Generating a new private RSA certificate to secure the FTP connection

Once the certificate is created, you can open the configuration file and add details about the new certificate. For more details on encryption, refer to *Chapter 9*, *Securing Linux*.

Here are the new directives we added to the configuration file:

```
rsa_cert_file=/etc/ssl/private/private-ftp-cert.pem
rsa_private_key_file=/etc/ssl/private/private-ftp-cert.pem
ssl_enable=YES

allow_anon_ssl=NO
force_local_data_ssl=YES
force_local_logins_ssl=YES

ssl_tlsv1=YES
ssl_sslv2=NO
ssl_sslv3=NO

require_ssl_reuse=NO
ssl_ciphers=HIGH
```

Figure 8.61 – New SSL directives inside /etc/vsftpd.conf

Before continuing, let's explain what every new directive means, as follows:

- `rsa_cert_file`: Sets the destination of the certification file.
- `rsa_private_key_file`: Sets the destination of the private key file (in our case, it is the same as the certification file).
- `ssl_enable`: Setting this to `YES` will enable SSL transactions.
- `allow_anon_ssl`: Setting this to `NO` will block anonymous connections over SSL.
- `force_local_data_ssl`: Setting this to `YES` will automatically require the use of SSL for data transfers.
- `force_local_logins_ssl`: Setting this to `YES` requires the use of SSL for logins.
- `ssl_tlsv1`: Setting this to `YES` will default to the use of TLSv1.
- `ssl_sslv2(v3)`: Setting this to `NO` will block default SSL v2 and v3 connections (will only use TLS by default).
- `require_ssl_reuse`: Setting this to `NO` will deny SSL reuse.
- `ssl_ciphers`: Setting this to `HIGH` will require strong encryption keys, 128 bits or higher.

After saving the file, we will restart the `vsftpd` service and check its status to see whether everything is working fine. At this point, you will see that using an unsecure client such as the FTP CLI client is no longer possible. You will need to use a client that supports TLS—such as FileZilla, for example.

By now, we have covered some of the most important services, the only ones remaining being the web and printing servers. In the following section, we will show you how to install and set up a web server.

Setting up a web server

On Linux, a traditional web server is usually known by the **LAMP** or **LEMP** acronyms, which stand for **Linux, Apache, MySQL/MariaDB, PHP** and **Linux, Nginx MySQL/MariaDB, PHP**, respectively. Such types of web servers are still widely used, even though they are slowly being replaced by the use of containerized or serverless web applications. Nevertheless, knowing how to quickly and efficiently set up a web server is a must for any Linux sysadmin.

Choosing between the two web servers, Apache and nginx, is a 2-decades-old problem. We will not debate which one is better. They both serve the same purpose, and choosing one boils down to your needs or preferences. To learn more about each one, visit *Chapter 7*, *Networking with Linux*, or visit the following links:

- DigitalOcean (https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations)
- IONOS (https://www.ionos.com/community/server-cloud-infrastructure/nginx/nginx-vs-apache-advantages-and-disadvantages/)
- Hostinger (https://www.hostinger.com/tutorials/nginx-vs-apache-what-to-use/)

The one you choose will be up to you, but our aim in this chapter is to show you how to set up a web server, no matter the technology it uses. Thus, due to editorial space reasons, we will only show you how to set up and install LAMP on Ubuntu. The setup process is similar for both servers when installing the necessary packages. The differences will arise when it comes to setting each server to work and seamlessly integrating it with the other components.

Installing and configuring a LAMP stack on Ubuntu

For this tutorial, we will use the same Neptune Ubuntu machine on our network. This means that everywhere we have to use the IP address, we will use the machine's local address. If you have a VPS and want to configure it as a web server to host your personal or business websites, you will need to use that VPS's IP address. The server must have a regular (not root) user that has `sudo` privileges, SSH access, and a firewall enabled.

Installing Apache

The LAMP stack is using Apache as the web server. This is one of the most widely used web servers in the world, with an active user and developer base, and it is well supported by major Linux distributions.

The first step is to install the Apache package using the package manager, right after you update the repositories and install (if needed) all the upgrades.

Important note

When using Ubuntu, the name of the Apache package is `apache2`, but please take into consideration that it does not have the same name when using CentOS, for example. When using CentOS, the name of the package you need to install is **httpd**.

To install the package, run the following commands:

```
sudo apt update -y

sudo apt install apache2 -y
```

After installing the package, we need to configure the firewall to allow **HyperText Transfer Protocol** (**HTTP**) traffic. By default, the `apache2` package installs a specific firewall profile. To see all the profiles available for the firewall, run the `ufw app list` command. If you test on your local network, you can select only the normal, unencrypted `Apache` profile, which uses port `80`. If you plan to host your website on a VPS, we advise you to use the `Apache Full` profile for TLS/SSL-encrypted traffic, which uses port `443`. Review *Chapter 7*, *Networking with Linux*, to refresh your memory about ports. The following screenshot shows the available firewall profiles for Apache:

```
packt@neptune:~$ sudo ufw app list
[sudo] password for packt:
Available applications:
  Apache
  Apache Full
  Apache Secure
  CUPS
  OpenSSH
  Samba
```

Figure 8.62 – Available firewall profiles for Apache

For now, as we only use the local machine on our network, we will set the unencrypted profile. By the end of this section, we will show you how to use and configure an encrypted **HyperText Transfer Protocol Secure** (**HTTPS**) connection on a VPS. Add the new rule to the firewall, using the following command:

```
sudo ufw allow in "Apache"
```

Then, you can check to see a full list of the modules available, with the new Apache modules for IPv4 and IPv6 being among them, as shown in the following screenshot:

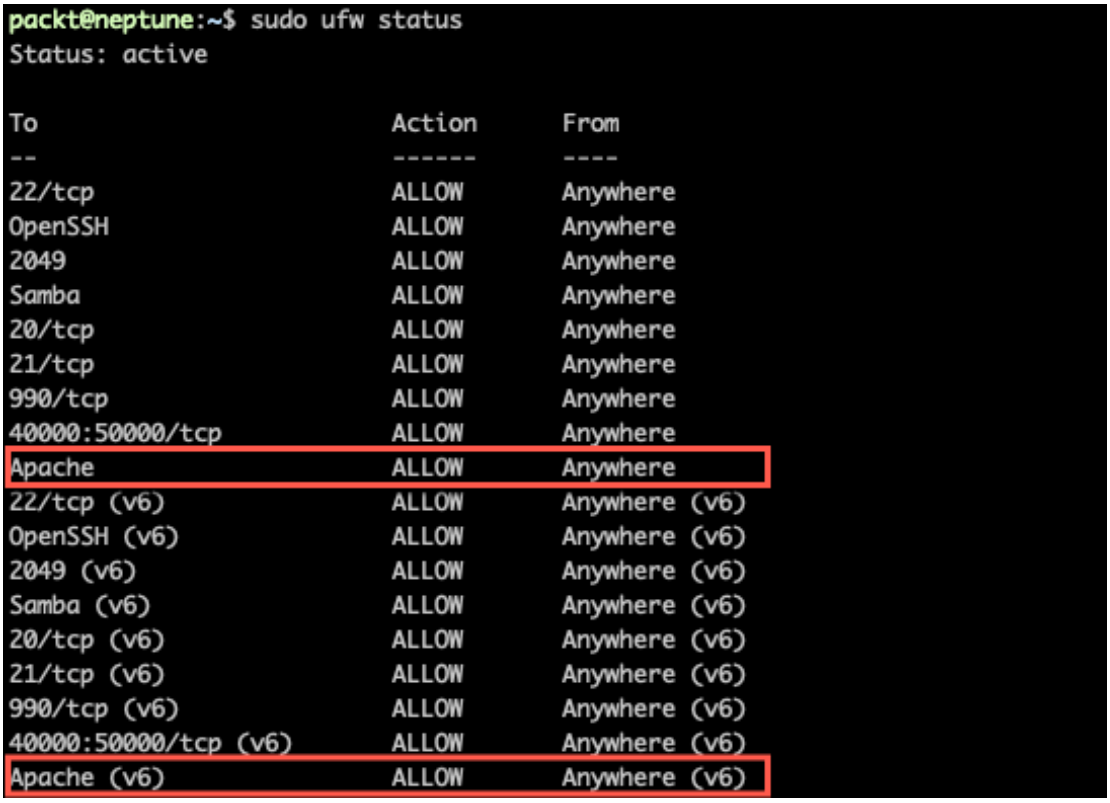

Figure 8.63 – Available firewall modules on our test system

Now that the Apache package is installed, the HTTP traffic is allowed through the firewall, and we can test to see if the web server is working properly. We will open a web browser on another system on the network and enter the server's IP address.

If you are using a VPS, you will have to use your server's public IP address, not like how we did it in the following screenshot example. The IP address was sent to you when you created the VPS, so you should check your email.

Accessing the IP address will render a welcome page, showing details about the Apache package we just installed. This means that the web server is working fine. The next step is to install MySQL or MariaDB.

Installing MySQL/MariaDB

It is now time to install the database management system. Most widely used in a LAMP stack are MySQL or MariaDB. MySQL is owned by Oracle and is not completely open sourced, while MariaDB is an open source alternative that emerged as a fork of MySQL. The database management system is needed to manage the data for the web server. We will use the MariaDB open source package, not Oracle's MySQL.

The installation process is pretty straightforward, with several additional packages being installed too. The installation script also suggests other packages, and if you feel that you might need some of them, proceed to their installation. Otherwise, the default installed packages are sufficient.

Important note

When choosing to install MySQL, the package you need to install in Ubuntu is called `mysql-server`.

Details about the packages that will be installed alongside MariaDB will be shown once you run the following command:

```
sudo apt install mariadb-server
```

Once the installation is finished, we can proceed to the configuration.

Configuring MariaDB

After installation, it is recommended to run a security hardening script called `mysql_secure_installation`. It has the same name for both MariaDB and MySQL. This script will simply run by using the following command:

```
sudo mysql_secure_installation
```

It will start asking you a series of questions—for example, about setting a root password, removing anonymous users, disallowing root login remotely, removing test databases, and reloading the privileges table. We advise you not to set any root password when asked to do so. The following note provides a short explanation about why not. For the rest of the questions, you can answer with the default `Yes`.

Important note

While using Ubuntu, the system's root account can access the database. The MariaDB root account is related to the system, as it uses socket authentication by default. Since version 10.4.3 of MariaDB, the `unix_socket` plugin is installed by default. This was done in order to have a more secure experience with the database management software. For more details about the plugin, visit the following link: https://mariadb.com/kb/en/authentication-plugin-unix-socket/.

You could either disable the plugin (but we do not advise you to do that) or you can create a new user with administrative rights that could access the database using the password. This

way, you will not be using the root user. The new user will be called `admin` and will have the same privileges as the root user. The following screenshot illustrates this:

```
packt@neptune:~$ sudo mariadb
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 42
Server version: 10.3.25-MariaDB-0ubuntu0.20.04.1 Ubuntu 20.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement
.

MariaDB [(none)]> GRANT ALL ON *.* TO 'admin'@'localhost' IDENTIFIED BY 'pass
word' WITH GRANT OPTION;
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]>
```

Figure 8.66 – Adding a new user with root privileges in MariaDB

The line that adds a new user to MariaDB is framed inside the red square. Please add a password of your choice that is secure enough for your needs. Once the new user is created, flush privileges and exit MariaDB.

The next step is to restart the MariaDB service and test its functionality. To restart the service, use the `systemctl` command. After restarting, you can check the status of the service, just to make sure that everything is working fine. Use the following commands:

```
sudo systemctl restart mariadb.service; sudo systemctl status
mariadb.service
```

MariaDB is now running on the system, and you can connect to it by using the new `admin` user. For testing purposes, let's connect with the new user to the database management software, as follows:

```
packt@neptune:~$ mysql -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 38
Server version: 10.3.25-MariaDB-0ubuntu0.20.04.1 Ubuntu 20.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement
.

MariaDB [(none)]>
```

Figure 8.68 – Connecting to MariaDB with the new admin user

The preceding output shows that connecting to MariaDB with the new `admin` user is working as expected and the configuration was successful.

The next step is to install the other components of the LAMP stack. By this point, we have installed Apache for the web server and MariaDB for the database to store and manage data. Next, we will install **PHP: Hypertext Preprocessor** (**PHP**) to manage dynamic content on a website. PHP is used by major **content management systems** (**CMSs**) such as **WordPress**, **Drupal**, or **Joomla**. It is also used by **learning management systems** (**LMSs**) such as **Moodle** and many others (such as **MediaWiki**, and so on). PHP is therefore an important component of the LAMP stack.

Installing PHP

Installing the PHP component is as straightforward as installing the other stack packages. All you have to do is to install the components shown below:

```
sudo apt install php libapache2-mod-php php-mysql
```

After installing PHP, the entire LAMP stack should be working on the system, and you can now begin hosting your websites on the server. Let's assume a theoretical situation, in which we would like to host our company's local documentation (or wiki) website in-house on the company's server and make it available through a LAMP stack.

Setting up a local wiki website

Apache uses virtual hosts to define websites. This way, a single machine can host more than one website, by defining different virtual hosts for each website it serves. There are two types of Apache virtual host: one is **name-based** and the other is **IP-based**. In order to host multiple websites on a single server that has one IP address, the name-based model will be used.

There are several files and locations that you will need to know in order to completely configure Apache. For starters, you should know that Apache, under Ubuntu, is configured to serve a website from within the `/var/www/html` location. When using multiple websites, it is a good practice to define separate directories to serve each website. This means that we will have to create one directory for each website we want to build. Another location is `/etc/apache2/sites-available`, where Apache keeps configuration files for each website we want to create. All the websites we build need to have a configuration file inside this directory. After websites are configured, they must be enabled and stored inside the `/etc/apache2/sites-enabled` directory. To enable a website, Apache has a special command called `a2ensite`. There is also a special `a2dissite` command to disable a website.

Here is the output when listing the contents of the `/etc/apache2/` directory, showing the `sites-available` and `sites-enabled` directories:

```
packt@neptune:~$ cd /etc/apache2/
conf-available/  mods-available/  sites-available/
conf-enabled/    mods-enabled/    sites-enabled/
```

Figure 8.70 – Contents of the /etc/apache2/ directory

Now, let's create a small wiki website for a small business, based on **MediaWiki**, and use it locally for documentation. As we already have the LAMP stack installed, we need to download MediaWiki. At the time of writing this section, the latest version of MediaWiki was version 1.35.0 (https://www.mediawiki.org/wiki/Download). At the same time, a package is already available in Ubuntu's repositories, but it only has version 1.31.7. For a simple and seamless integration, we will use the package available in the official repositories, as follows:

```
sudo apt install mediawiki
```

Once the package is installed, the configuration file is available inside `/etc/mediawiki`. All you need to do is to enable it with the following command:

```
sudo a2enconf mediawiki
```

After the website is enabled, open a new web browser window and go to your server's IP/mediawiki (in our case: `192.168.0.244/mediawiki`), as follows:

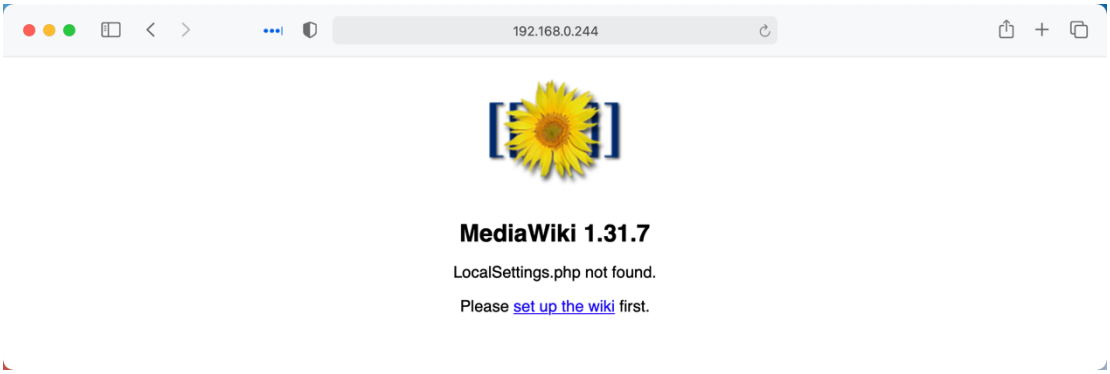

Figure 8.71 – Opening your wiki location in a web browser

Before running the first MediaWiki installation, we need to create a MariaDB user and database for the new website. Here are the steps you need to follow to do this:

1. Connect to MariaDB using the following command:

   ```
   sudo mysql -u admin -p
   ```

2. Create a new user called `wiki-user` with the following command:

   ```
   CREATE USER 'wiki-user'@'localhost' IDENTIFIED BY
   'password';
   ```

3. Create a new database called `wiki` by running the following command:

   ```
   CREATE DATABASE wiki;
   ```

4. Set the new database as default by running the following command:

   ```
   use wiki;
   ```

5. Grant access to the new user to the new database by running the following command:

   ```
   GRANT ALL ON wiki.* TO 'wiki-user'@'localhost';
   ```

After the user and database creation, we can go back to the web browser and click on the **set up the wiki** link. This will start the initial MediaWiki configuration. There will be steps where we will need to give information about the database we just created, the database username (the user), password, and the name of the new wiki. We will also have to choose a new username for the website administration. In the end, if everything goes well without any errors, you will be able to see a page similar to the following:

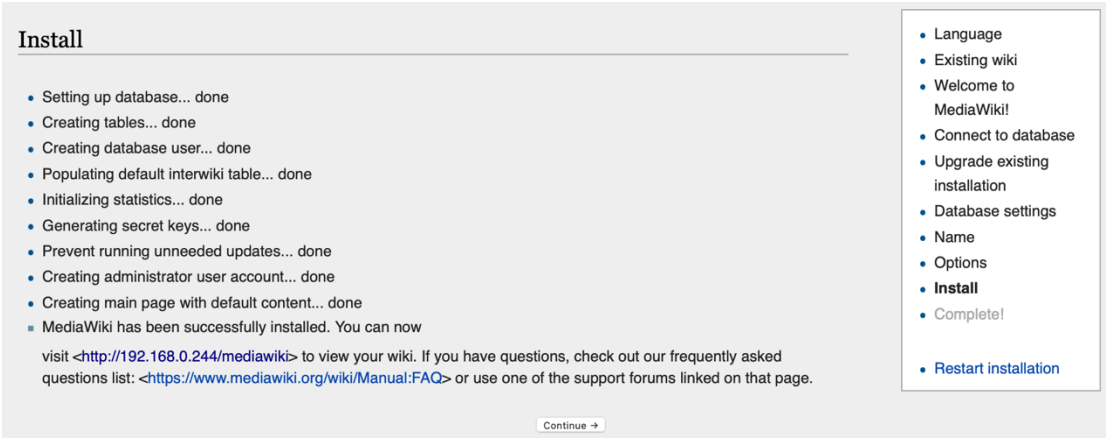

Figure 8.73 – The finished installation of MediaWiki

The setup process is not finished yet. As shown in the following screenshot, you are prompted to download a `LocalSettings.php` file and place it under `/etc/mediawiki`. If you are not accessing the wiki from your server but from a client, you will have to download it locally and then transfer it to the server:

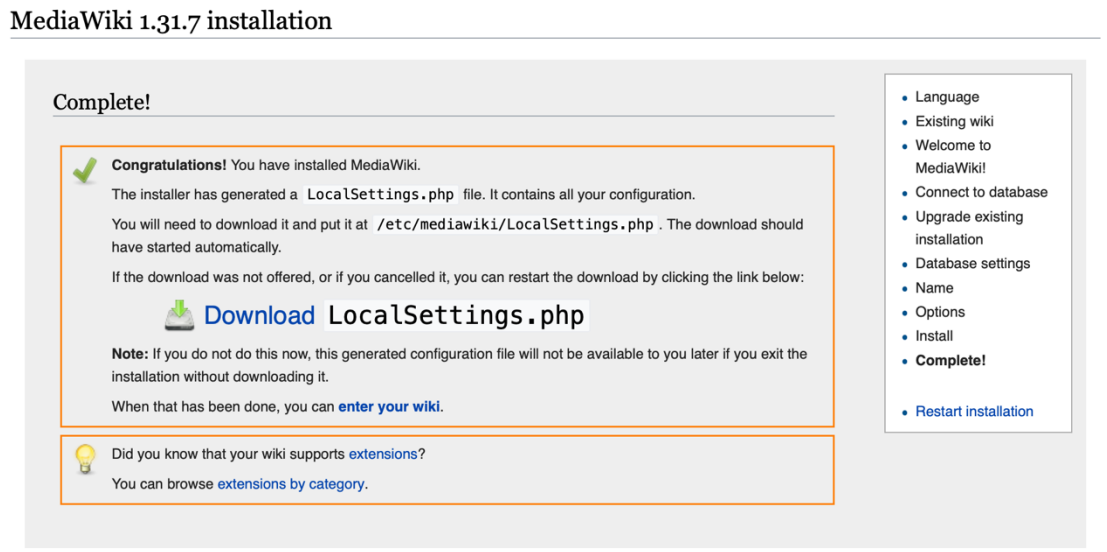

Figure 8.74 – Request to download the LocalSettings.php file

Save the file locally and then transfer it to the server. We used `scp` to transfer the file from one of our Mac systems to our Neptune Ubuntu server, as shown here:

```
~/Downloads ❯❯❯ scp LocalSettings.php packt@192.168.0.244:/home/packt/     ✘ 1
packt@192.168.0.244's password:
LocalSettings.php                             100% 4532     2.4MB/s   00:00
```

Figure 8.75 – Copying the file from the client to the server

After the file was transferred to the server inside the `packt` user's home directory, we copied it to the requested location, `/etc/mediawiki`, as shown here:

```
packt@neptune:/etc/mediawiki$ sudo cp /home/packt/LocalSettings.php .
[sudo] password for packt:
```

Figure 8.76 – Copying the file to the required location

You can now use your wiki by accessing http://server_ip/mediawiki, where the `server_ip` should be replaced with the actual IP of the machine.

The choice of a MediaWiki installation as our example in this chapter was not by chance. We chose it for the following two reasons:

- It is an example of using a package directly available from the Ubuntu repositories.
- It uses a slightly different default configuration, with different default files and file locations.

In our opinion, this could benefit from an equally valuable understanding of how Apache works when used with a preconfigured setup, such as in the case of the MediaWiki package available inside the **Universe** repository.

Setting up a local CMS website

When it comes to installing a CMS, most people would choose **WordPress**. Even though it is used by a large number of websites, we encourage users to try other solutions too, and **Drupal** is a mature and viable one. It has strong security, and a dedicated user and developer base. Drupal is an open source CMS solution that is written in PHP and has reached version 9 at the time of writing this section. For more details about it, please visit the official website at https://www.drupal.org/about/9.

As the LAMP stack is already installed, we need to create a new user and database for Drupal, and a new Apache virtual host for the CMS website. After those steps are complete, the setup process will be continued through a website.

First, let's create a new database user and Drupal-dedicated database. The process is similar to the one used when created the Wikipedia website.

Let us assume that the new user is called `drupal`. While creating it, please make sure to add a good password. The database we assume is called `drupal` too.

The next step is to download and install the latest version of Drupal. The best way is to download it from the official website, as there is the latest version available. Before

downloading the package, create a dedicated directory inside `/var/www` called `drupal`, (for example), as illustrated in the following screenshot:

```
packt@neptune:/var/www$ sudo mkdir drupal
[sudo] password for packt:
packt@neptune:/var/www$ ls
drupal  html  wiki
packt@neptune:/var/www$ cd drupal/
packt@neptune:/var/www/drupal$ ls
```

Figure 8.78 – Creating a new directory for the CMS website

After creating the new directory and setting it as your present working directory, follow these steps:

1. Download the latest Drupal package from the official website, with the following command:

   ```
   sudo wget https://www.drupal.org/download-latest/tar.gz -O drupal.tar.gz
   ```

2. Extract the archive, as follows:

   ```
   sudo tar -xf drupal.tar.gz
   ```

3. Copy the content to the parent directory, as follows:

   ```
   sudo cp -R drupal-9.0.8/* .
   ```

4. Change the file permissions and ownership for Drupal to be able to access the files, as follows:

```
packt@neptune:/var/www/drupal$ sudo chown -R www-data:www-data /var/www/drupal/
packt@neptune:/var/www/drupal$ sudo chmod -R 755 /var/www/drupal/
```

Figure 8.79 – Setting file permissions and ownership for Drupal access

We are now at the last part of the CMS configuration. We still need to create an Apache virtual host for the website. Do you remember where the configuration files for virtual hosts are? We told you at the beginning of this section. They are located inside the `/etc/apache2/sites-available/` directory, and can be seen here:

```
<VirtualHost *:80>

        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/drupal
        ServerName private-cms.local
        ServerAlias www.private-cms.local
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
        <Directory /var/www/drupal>
                Options FollowSymLinks
                AllowOverride All
                Require all granted
        </Directory>

</VirtualHost>
```

Figure 8.80 – The virtual host details from the configuration file

After the virtual host has been saved, all we have to do is to enable the website and go to the web browser to start the configuration. Once the website is enabled, we will enable the `rewrite` module too. The required code can be seen here:

```
packt@neptune:~$ sudo a2ensite drupal.conf
[sudo] password for packt:
Enabling site drupal.
To activate the new configuration, you need to run:
  systemctl reload apache2
packt@neptune:~$ sudo a2enmod rewrite
Enabling module rewrite.
To activate the new configuration, you need to run:
  systemctl restart apache2
packt@neptune:~$ sudo systemctl reload apache2
```

Figure 8.81 – Enabling the website and rewrite module and restarting the service

After the website is enabled and the service restarted, we can proceed with starting the actual Drupal configuration in the browser. Go to a web browser on any client on the network and add the IP address of your web server, which in our case is `192.168.0.244`, and the Drupal configuration starts. You should see the following screen:

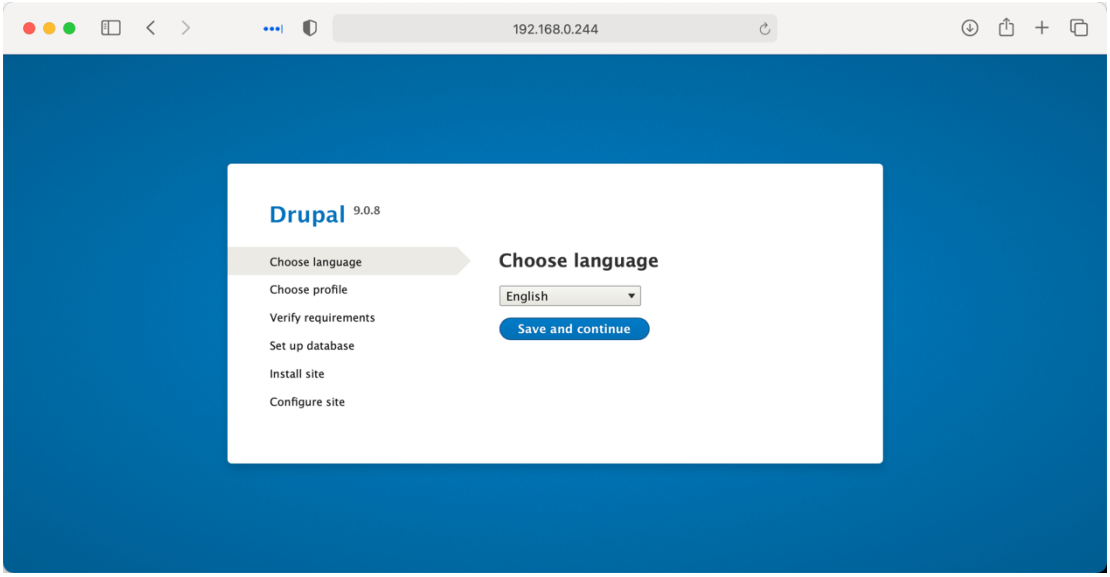

Figure 8.82 – Drupal configuration inside a web browser window

Drupal is now installed and ready to host your future website. With the current configuration, your website would be accessing the internet using only the HTTP protocol. For better security and privacy, we will show you how to use free SSL/TLS certificates for your domain.

Setting up HTTPS using Let's Encrypt

Let's Encrypt is one of the most widely used **Certificate Authorities** (**CAs**) that lets you use free SSL/TLS certificates. The setup process is quite simple and straightforward.

The CA has an installation software called **Certbot**. To install it in Ubuntu using the Apache web server, run the following command:

```
sudo apt install certbot python3-certbot-apache
```

Before running it, make sure that your Apache virtual host configuration file is created and has no errors, and that the firewall has the HTTPS profile activated. We will remind you of the command to run for these tasks. To allow HTTPS traffic through the firewall, run the following command:

```
sudo ufw allow 'Apache Full'
```

To verify the integrity of the Apache virtual host configuration file, run the following command:

```
sudo apache2ctl configtest
```

If the output shows no errors, this means that everything is running smoothly and is ready for Certbot configuration.

Important note

Using HTTPS-enabled traffic is advised on any public-facing website. The Certbot tool is useful when installed on a VPS, not on your local network, using a valid domain with a valid and public suffix (**top-level domain**, or **TLD**).

The Certbot application can be run using the following command:

```
sudo certbot --apache
```

Next, the application will ask you several questions in order to provide a valid certificate. You will have to interact with it and respond to the following points accordingly:

1. Provide your email address for renewal notices.
2. Agree with their terms of service.
3. Share your email with the **Electronic Frontier Foundation** (**EFF**).
4. Specify the names for your domain, by choosing from a list.
5. Choose to redirect (or not) all traffic through HTTPS.

The certificate is now issued for your domain. The last thing you can do is to test the auto-renewal facility. Certbot automatically installs a renewal script that runs twice a day and will automatically renew any certificate due to expire. To test the renewal script, run the following command:

```
sudo certbot renew - -dry-run
```

There should be no errors if the entire installation process runs smoothly.

You now have a valid certificate configured for your domain. This will enable automatic SSL-encrypted traffic for your website.

In this section, entitled *Setting up a web server*, we took you on a journey that showed you how to install and set up the LAMP stack, together with installing two different types of websites: a wiki and a CMS website. We consider that this is sufficient for basic web server training, and you should only go up from here.

In the next section, we will show you how to install and set up a printing server on Linux.

Setting up a printing server

Let's assume the same scenario as in the previous section: you have a small business and need to set up the necessary service for it to work according to your expectations. Now, it's time to install and configure a printing server.

We will use the same Neptune Ubuntu server we have used up to now. Ubuntu uses a **Common UNIX Printing System** (**CUPS**), which uses the **Internet Printing Protocol** (**IPP**) to manage printing. CUPS can use several other protocols besides IPP, such as Samba or AirPrint. For the printing server to work, you need a printer (or more than one) and printing drivers installed on the server. We have two printers on the network.

First, you will need to install the CUPS package with the following command:

```
sudo apt install cups -y
```

If the package is already installed, the shell will show you a message similar to the following:

```
packt@neptune:~$ sudo apt install cups -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
cups is already the newest version (2.3.1-9ubuntu1.1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

Figure 8.83 – Installing CUPS on Ubuntu

Once the CUPS package is installed, start and enable the service. If CUPS was already installed, there is a small chance that it is already running. In that case, you could at least enable it using the following command:

```
packt@neptune:~$ sudo systemctl enable cups
Synchronizing state of cups.service with SysV service script with /lib/system
d/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable cups
```

Figure 8.84 – Enabling CUPS service

Open the `/etc/cups/cupsd.conf` administrative file and edit some of its directives, as follows:

1. First of all, we want to make the printers that are available browsable, so look for the line that says `Browsing Off` and turn it to `Browsing On`.

2. Look for the line that shows the port on which the web interface is available (such as `Listen localhost:631`) and change it to port `631`. This will make CUPS available to all other computers on the network.

3. To further enable access from all systems on the network, find the directive that shows the following lines, and add a new line that says `Allow @LOCAL` to enable access from all computers on the same local network:

```
<Location />
Order allow, deny
Allow @LOCAL
</Location>
```

4. To allow remote administration, you can also find the line with the following directive, and add `Allow @LOCAL` to it:

```
<Location /admin>
Order allow, deny
Allow @LOCAL
</Location>
```

The next step after altering the configuration file is to allow traffic through the firewall for every client on the network to access port `631`. We will add the following rule to the **Uncomplicated Firewall** (**UFW**) firewall:

```
packt@neptune:~$ sudo ufw allow in from 192.168.0.0/24 to any port 631
[sudo] password for packt:
Rule added
packt@neptune:~$ sudo ufw status
Status: active

To                         Action      From
631                        ALLOW       192.168.0.0/24
```

Figure 8.85 – Allowing traffic through port 631 for the entire network

If we execute the `sudo ufw status` command, somewhere among all the rules already enabled there is the one we just added, and in the preceding screenshot we highlighted just that.

Now that CUPS is installed and configured and the firewall is set up, we need to install the local drivers on the server. In our case, they are already installed, but only partially. We have two **Hewlett Packard** (**HP**) printers on the network and we have a **HP Linux Imaging and Printing** (**HPLIP**) driver installed, but we will install an extra driver that offers support for HP, Canon, and Epson printers too, just in case we were to add some other printers at some point in the future. The package is called `printer-driver-gutenprint`. The following screenshot provides an example of how to install it:

```
packt@neptune:~$ sudo apt install hplip
Reading package lists... Done
Building dependency tree
Reading state information... Done
hplip is already the newest version (3.20.3+dfsg0-2).
hplip set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
packt@neptune:~$ sudo apt install printer-driver-gutenprint
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libgutenprint-common libgutenprint9
Suggested packages:
  gutenprint-locales gutenprint-doc
The following NEW packages will be installed:
  libgutenprint-common libgutenprint9 printer-driver-gutenprint
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 1.500 kB of archives.
After this operation, 9.988 kB of additional disk space will be used.
Do you want to continue? [Y/n] _
```

Figure 8.86 – Installing HPLIP and additional driver support

At this point, you will need to have the printer connected to the server. In our case, both printers are on the network, accessible only through the Ethernet cables, and none of them is directly connected to the server using a **Universal Serial Bus** (**USB**) cable (due to the fact that they are in a separate room on our premises, not in the same room as the servers). This renders them unusable with the `lp` command-line printing tool at this point, but we still want to make them available through the server and use at least one of them as a shared printer from the server.

As we do not have GUI access to the server, the usual tools to configure the two printers are unavailable to us. Nevertheless, we can still have access to the printer's configuration through the web browser. If you find yourselves in this situation, here is a viable solution to make your network-shared printers available for the server.

Open a web browser on any client on the network and go to the following address: `https://192.168.0.244:631/admin/`.

This is the printer's administrative console on our server. Use your own server's IP address, not the one we used. Once inside the administrative page, something similar to the details in the following screenshot should be available to you too:

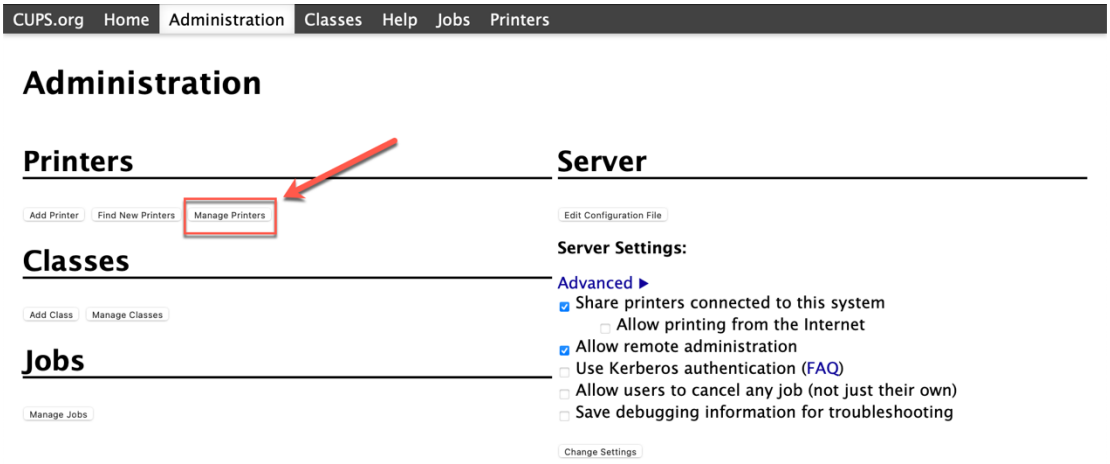

Figure 8.87 – Printing administrative web page

In the new page, click on the **Manage Printers** button, and this will redirect you to a new page where the available printers are listed. Select the one you want to configure. In our case, it will be the LaserJet model, as illustrated in the following screenshot:

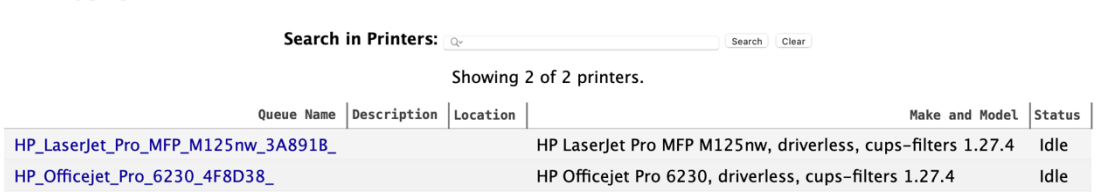

Figure 8.88 – Available printers listed

Once we select the desired model, the **LaserJet Pro MFP M125nw**, we can proceed to the next page and make it the default printer for the server, as illustrated in the following screenshot:

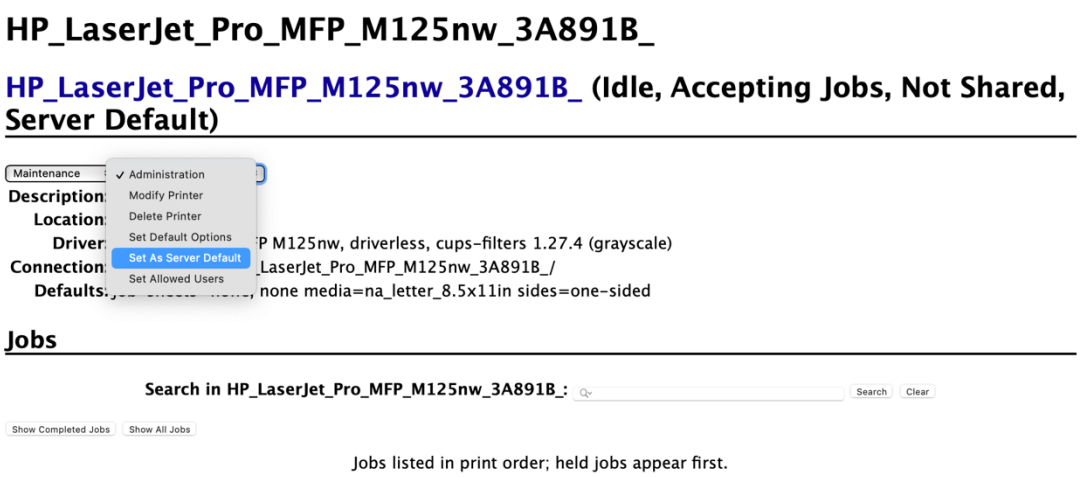

Figure 8.89 – Setting the default printer for the server

The printer is now directly available through the server and can be used by it. Being a network-available printer, having a print server to use it could be a bit of a stretch, but it is nevertheless a good exercise to carry out for your setup. If you can connect your printer directly to the server using a USB cable, please do that, and the setup process using the command line would be a little bit easier. All the steps described in this section are applicable.

Nevertheless, there still are some commands that you can use if you are in our situation. For example, you can use the `lpinfo` command to see if there is available information for your printer model, as follows:

```
packt@neptune:~$ lpinfo -m | grep "HP LaserJet Pro MFP M125nw"
driverless:ipp://HP%20LaserJet%20Pro%20MFP%20M125nw%5B3A891B%5D._ipp._tcp.loc
al/ Hewlett-Packard HP LaserJet Pro MFP M125nw, driverless, cups-filters 1.27
.4
```

Figure 8.90 – Finding information about a specific printer model

Used with the `-m` option it will list all the available printers, but used with `grep` you can search for a specific model, as follows:

```
packt@neptune:/etc/cups/ppd$ ls
HP_LaserJet_Pro_MFP_M125nw_3A891B_.ppd  HP_Officejet_Pro_6230_4F8D38_.ppd
```

Figure 8.91 – Available ppd files from CUPS

Furthermore, by going to the `/etc/cups/ppd` directory, you can see the available `ppd` files for the printers that CUPS recognizes as being on the network or directly connected, as illustrated in the following screenshot:

```
packt@neptune:~$ lpoptions
device-uri=ipp://HP%20LaserJet%20Pro%20MFP%20M125nw%5B3A891B%5D._ipp._tcp.loc
al/ printer-info='HP LaserJet Pro MFP M125nw[3A891B]' printer-location printe
r-make-and-model='Hewlett-Packard HP LaserJet Pro MFP M125nw' printer-type=83
886084
```

Figure 8.92 – lpoptions command output

At this point, the printers are only available through CUPS, thus being browsable from Linux and, but we still need to add IPP discoverability using the **multicast DNS** (**mDNS**) protocol. For this, we need to have the `avahi-daemon` package installed. If the package is already installed, you will have a message like the one in the next screenshot.

Important note

Keep in mind that the IPP/Bonjour delivers the ability of driverless printing, as it makes CUPS-supported printers available over the network without any driver installation. This is called driverless printing.

The next step is to start the service and check its status, using the following commands:

```
sudo apt install avahi-daemon; sudo systemctl start avahi-
daemon; sudo systemctl status avahi-daemon
```

One more package that would be needed is `cups-ipp-utils`. You could install this with the following command:

```
sudo apt install cups-ipp-utils
```

The `avahi-daemon` package listens by default on **User Datagram Portal** (**UDP**) port `5353`, and the next step is to allow it through the firewall by running the following command:

```
sudo ufw allow 5353/udp
```

At this point, you have a working printing server. This, together with all the other servers configured in this chapter, is a base for a strong server setup.

Summary

In this chapter, we covered the installation and configuration processes for the most known services available for Linux. Knowing how to configure all the servers described in this chapter—from DNS, DHCP, Apache, and a printing server—is a minimum request for any Linux administrator.

By going through this chapter, you learned how to provide essential services for any Linux server. You learned how to set up and configure a web server using the Apache package; how to provide networked printing services to a small office or home office; how to run an FTP server and share files over TCP; how to share files with Windows clients on your network using the Samba/CIFS protocol; how to share files over Unix and Linux systems using the NFS file-sharing protocol; how to set up NTP to show an accurate time; and how to configure DNS and local DHCP servers. In a nutshell, you learned a lot in this chapter, and yet we have barely scratched the surface of Linux server administration.

In the next chapter, we will cover extensive Linux security themes, from working with `iptables`/`nftables` firewalls to setting up **Security-Enhanced Linux** (**SELinux**), AppArmor, Netfilter, and VPNs.

Questions

Now that you have a clear view of how to manage some of the most widely used services in Linux, here are some exercises that will further contribute to your learning:

1. Try using a VPS for all the services detailed in this chapter, not on your local network.
2. Try setting up a LEMP stack on Ubuntu.
3. Exercise with all the services described in this chapter, using the CentOS 8 distribution.

Further reading

For more information about the topics covered in the chapter, you can check the following link:

- Ubuntu 20.04 official documentation: https://ubuntu.com/server/docs