Linux Server Cookbook

Get Hands-on Recipes to Install, Configure, and Administer a Linux Server Effectively

Alberto Gonzalez

bpb

Linux Server Cookbook

Get Hands-on Recipes to Install, Configure, and Administer a Linux Server Effectively

Alberto Gonzalez

Linux Server Cookbook

Get Hands-on Recipes to Install, Configure, and Administer a Linux Server Effectively

---

Alberto Gonzalez

www.bpbonline.com

First published: 2023

Published by BPB Online

WeWork

119 Marylebone Road

London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-5551-360-1

www.bpbonline.com

Dedicated to

To the best person I have met in my life:
Elizabeth Wangari.

About the Author

Alberto Gonzalez is a Principal Architect at Red Hat with more than 20 years' experience in the IT sector, with his main focus being Cloud and Infrastructure, and mainly open-source software. He is currently teaching and creating content on Hybrid Cloud technologies. Alberto has previously written two books in Spanish, one about Ansible and another about Docker.

About the Reviewer

Harshal Lakare is a Linux administrator with over 11 years of experience in the field. He has a strong background in system administration and has expertise in a variety of Linux distributions. Harshal is skilled in troubleshooting, problem-solving, and has a keen eye for detail. He is also a team player and enjoys collaborating with others to achieve common goals. In his spare time, Harshal enjoys reading and keeping up with the latest trends in the field. Overall, Harshal is a talented and dedicated Linux administrator who is committed to delivering high-quality work and helping his team succeed.

Acknowledgements

My gratitude goes to the team at BPB Publication for helping me during the writing process. Not being native in English is a challenge, but they helped during the whole process to create a book that I am proud to have written. They were supportive during the whole process, understanding the changes proposed and new topics added during the writing process.

I want to say thanks to my family and friends; they understood that writing a book requires to dedicate multiple hours which I should have dedicated to them. This book is the result of that effort.

Preface

This book covers how to administer a Linux server from basic tasks to advanced ones., starting with the installation of popular Linux distributions, going through administration users and software, to the installation and administration of advanced services such as databases and file sharing. This book will guide you through new technologies related to automation, containers and DevOps philosophy. Public and Private Clouds are covered, as well as why Hybrid Cloud is an important concept for enterprises.

Chapter 1 is the introduction to Linux. This chapter explains the history of the Operating System, the usage of Linux in most of the devices available and the IT sectors where Linux is the main operating system. This chapter also covers the latest features available and discusses the promising future of Linux.

Chapter 2 covers the Linux installation. It describes the different support levels available for the popular distributions. It also details the installation methods available and guides step-by-step how to install the popular Linux distributions available.

Chapter 3 introduces the Command Line Interface, a main component when a user is operating or administrating a Linux system. This chapter covers basic commands and concepts related to the input and output of the commands. Commands to identify the resources of the system are detailed with different examples.

Chapter 4 shows how to manage users and software on Linux. It describes how users and groups work on Linux. With multiple examples, this chapter describes how to install software on systems such as Ubuntu Server or Red Hat Enterprise Linux.

Chapter 5 covers how to manage files, directories and processes. It describes the directory structure on Linux, the permissions and how to access to the files. This chapter also covers the special characters and the regular expressions, as well as the important concepts working on the terminal. Popular editors and file managers are also described. The last part of the chapter focuses on the process management and the priorities.

Chapter 6 explains how to monitor the resources in the system. Commands to query the usage of CPU, Memory, Disk and Network are described with several examples. This chapter covers quotas and limits to limit the usage of the resources available.

Chapter 7 covers the basics of the networking in general and explains how to configure the networking on Linux system. It also discusses commands to manage the IP addresses, and the routes are shown with different examples. In addition, advanced network configuration, such as bonding, bridges and virtual switches are also covered.

Chapter 8 focuses on the security part on a Linux system. It discusses the different firewall solution available and how to ensure that the services are correctly protected against unwanted access. Different tools are described to monitor the traffic and the changes in the system.

Chapter 9 details the popular network services available on Linux and the software associated to offer the services. Services such as DHCP, DNS and SSH are detailed with examples and the commands associated.

Chapter 10 is about File Sharing solutions available on Linux. It describes NFS, SMB and FTP protocols and the software to provide the services and the client tools.

Chapter 11 covers the popular databases available on Linux system. This chapter describes the difference between relational databases and NoSQL databases. Different solutions such as MySQL and MariaDB are detailed for the relational databases and MongoDB is described for the NoSQL database.

Chapter 12 details about what is Automation and the importance in these days. This chapter explains how to perform automation with shell scripting and developing, with a programming language such as Python. The popular tool Ansible is detailed to perform simple and advanced automation tasks.

Chapter 13 details the containers and the modernization of applications. Software to run containers, such as Docker and podman are also explained with several examples of usage. A basic introduction to Kubernetes is explained. New practices such as Continuous Integration and Continuous Delivery (CI/CD) are explained and the software available for these practices are described with different examples.

Chapter 14 explains about the Backup and Restore and the importance in the enterprise level. Different open-source solutions are described. Installation, usage and examples of open-source software solutions called Bacula and Relax and Recover (ReaR) are covered.

Chapter 15 covers a new concept: Multi Cloud Management. This chapter describes the popular public cloud available, such as AWS, GCP and Azure, as well as the services offered. The concept Infrastructure as code and the software Terraform for managing multiple clouds is also detailed with multiple examples.

Chapter 16 is about Infrastructure as a Service. It details the components of a Private cloud and how a Hybrid Cloud operates. The popular IaaS solution called **OpenStack** is explained with detail and with several examples.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/372e55

The code bundle for the book is also hosted on GitHub at **https://github.com/bpbpublications/Linux-Server-Cookbook**. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

customer, you are entitled to a discount on the eBook copy. Get in touch with us at: **business@bpbonline.com** for more details.

At **www.bpbonline.com**, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Table of Contents

CHAPTER 1

Introduction to Linux

Introduction

On August 25, 2021, Linux celebrated 30 years since Linus Torvalds posted a message about the new operating system he had designed. The IT sector experienced enormous advances and changes during these three decades. Linux became and consolidated the most important operating system in the industry.

In this chapter, we will describe why Linux is essential for companies, its latest features, and the cases in which it is used.

Structure

In this chapter, we will discuss the following topics:

- The Magnitude of Linux
- Linux on key sectors of the IT industry
    - Software
    - Devices and Infrastructure
    - Information Technology and Business Services
    - Emerging Technologies
    - Telecommunications Services
- Latest Features in Linux
- Linux vs. Other Operating Systems
- Promising Future of Linux

The magnitude of Linux

Nowadays, it is impossible to understand our lives without a device connected to the internet: a phone, a tablet, or a personal computer,

accessing a Web server or using an application to read one's e-mails, look up something in a search engine (like Google), or shop for a product you need.

The most common operating system for phones and tablets is Android, a Linux variant. Popular services used by people on a daily basis, such as mail servers or Web servers, are all most probably running in systems with a Linux distribution. Other devices providing Internet access to those services are usually running Linux too.

Linux is also used by a large number of professional developers, IT professionals, and regular users as their main operating system. Linux is currently the third most popular operating system for the desktop.

Linux began as a personal project by Linus Torvalds in the year 1991. The development was done on *MINIX,* a *Unix-like* operating system. The year after, the *X Window System* was ported to Linux, helping to gain importance. The timeline for the more important releases are the following:

- The first stable version was released in the year 1994.
- The version 2.0 was released in the year 1996.
- The version 2.2 was released in the year 1999, 2.4 in the year 2001, and 2.6 in 2003.
- The version 3.0 was published the year 2011.
- The version 4.0 in the year 2015, 5.0 in the year 2019, and version 6.0 in 2022.

Some other important events related to Linux during history are as follows:

- Debian started in the year 1993.
- Red Hat Linux was released in the year 1994
- Ubuntu was released in the year 2004.
- Android version 1.0 was released in the year 2008.
- All the Top500 list of fastest supercomputers run Linux from the year 2017.

Linux on key sectors of the IT industry

Linux is a well-known platform for running applications and services in enterprise environments. But Linux is not limited to servers; it is a key part

of the Information Technology industry and its general areas, such as:

- Software
- Devices and Infrastructure
- Information Technology and Business Services
- Emerging Technologies
- Telecommunications Services.

Software

This area includes developing software for business or customer markets, such as internet applications, system software, databases, management, home entertainment, and so on. Historically speaking, there was a gap between developers and the system administrators, as well as between the developers and the environment where the applications are executed.

Modern development methodologies are based on the balance and relationship between the system administrators and the programmers. The concept of **DevOps** is a set of practices that combines software development (**Dev**) with IT Operations (**Ops**). It aims to improve the system development life cycle and provides continuous delivery with high software quality.

DevOps methodology is based on multiple tools, called *toolchains*, and is deployed by Linux in different stages. Some of these tools will be covered in this book. The following figure illustrates this:

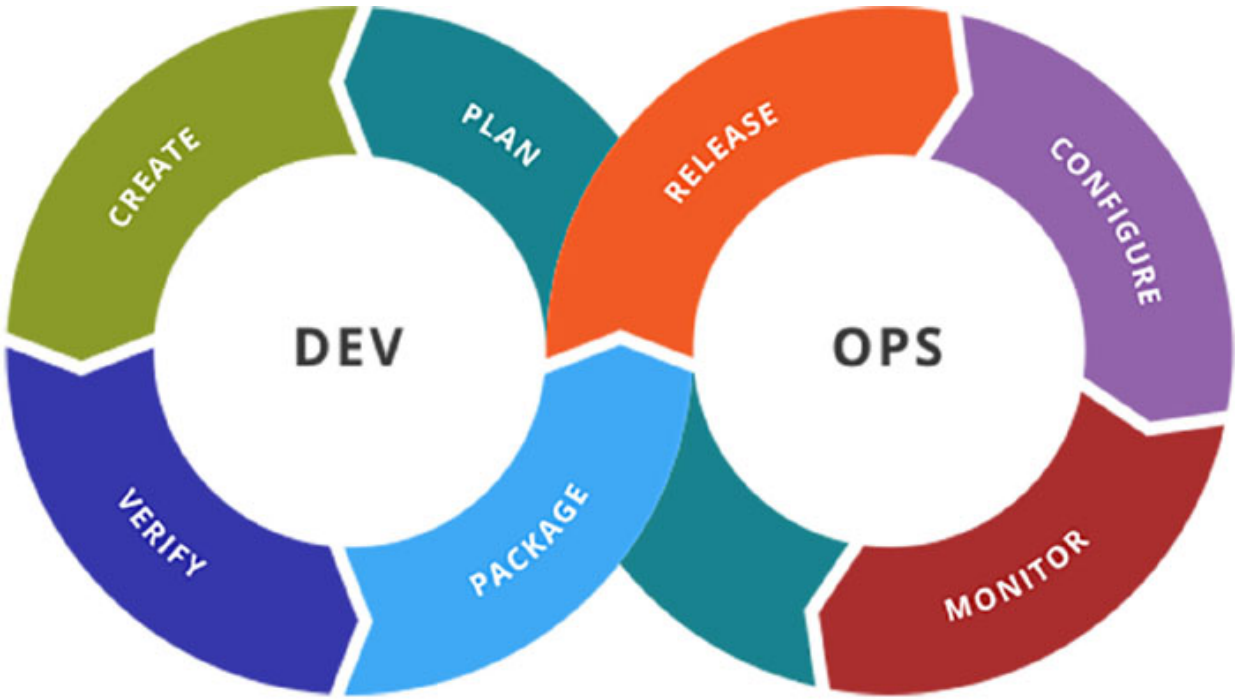


Figure 1.1: *DevOps stages. Source: Wikimedia*

- **Plan**: This stage is the definition of the activities required.
    - *Jira* is one of the most popular tools for this stage that can be executed on Linux. Some open-source alternatives are *Taiga* and *Kanboard*.
- **Create**: This stage includes the coding, building, and integration with the source repository as well as the continuous integration.
    - Some popular tools are as follows:
        - For coding, Integrated Development Environments (*IDEs*): *Vim*, *Emacs*, and *Visual Studio Code.*
        - For repository and packaging: JFrogArtifactory, Sonatype Nexus, Gitlab.
        - For building and continuous integration: *Maven*, *Jenkins*, and *Gitlab CI.*
- **Verify:** This stage ensures the quality of the software. Different tests are performed during this stage related to security, integration, and performance.
    - Some popular tools are as follows:

        - Testing tools: *Selenium, Appium, Cypress, and JMeter*.
        - Bug tracking: *Bugzilla, Redmine, and Mantis BT*.
        - Code review: *Gerrit, Gitlab, and ReviewBoard*.
        - Security: *OWASP ZAP*, *OSSEC*, and *SonarQube.*
- **Package**: After the verification stage, the release is ready to be deployed in a system with similar characteristics than production; this environment is called stage or preproduction.
    - Popular tools include *Docker* and other tools described in the Create stage.
- **Releasing:** This is one of the most important stages of *DevOps* methodology, and it includes the orchestration and the deployment of the software to production.
    - Container platforms: *Docker* and *Kubernetes.*
    - Popular continuous development tools: *goCD* and *ArgoCD.*
- **Configuring**: This stage includes infrastructure configuration and management and infrastructure as code tools.
    - Popular tools examples: *Ansible*, *Puppet*, and *Terraform*.
- **Monitoring**: This stage examines the performance monitoring and end-user experience for the applications.
    - Popular tools examples: Prometheus and Grafana, Elastic Search, Zabbix, and Nagios.

Companies are using Linux for developing and run software due to the reliability, customization, and performance offered by this system. Linux offers a secure environment to run critical and customer-facing applications.

Devices and infrastructures

Linux is not only limited to physical servers or virtual machines. Rather, it is present in every device type available, some examples of which are as follows:

- Tiny computers like the popular Raspberry PI

- Tablets, Chromebooks, and E-readers
- Televisions and streaming devices
- Routers and other network devices
- Supercomputers: the top 500 of which are using Linux

Enterprise infrastructure has undergone a great transformation during the last two decades. Devices for storage and networking, for example, were previously closed proprietary platforms in a monolithic implementation but have now moved to a modular, open implementation, virtualized, or containerized environment. The term *Software-defined Infrastructure* is the result of the transformation for compute, storage, and network resources. Linux played a key part in this transformation in deploying the services with the same, if not better, efficiency.

Software-defined compute (SDC): Also known as virtualization, it is when a compute function is virtualized and is abstracted from the hardware that is running it. A popular solution for virtualization is *KVM*.

- **Software-defined network (SDN):** A network architecture that makes the management of the network more flexible and easier, SDN centralizes the management by abstracting the control plane from the data plane. Controllers, the core element of SDN architecture, are run in a Linux system. Some popular solutions are *OpenvSwitch*, *OVN*, and *OpenDayLight*.
- **Software-defined storage (SDS)**: An architecture to offer dynamic storage to the endpoints, independent of the underlying hardware available. Some popular solutions are *Ceph* and *FreeNAS*.

Information technology and business services

IT services include providers offering services and integration, such as consulting and information management. This area also includes data processing and outsourcing services, including automation services.

Business services

Business services offered by IT companies have been transitioning from an implementation and maintenance stage to offering services and integration solutions. The concept of "*as-a-service*" is currently one of the most

important ones in business services, especially with the adoption of cloud computing. There are three popular types offered "*as-a-service*", where Linux is a key component:

- **Software as a Service (SaaS)**: This service offers applications where everything behind is managed by the provider. This includes everything related to storage, network, virtualization, server, operating system, and runtime. In this category, popular Web services for e-commerce, document sharing, and editing, or online editors are also included.
- **Platform as a Service (PaaS)**: This service offers the possibility of running your own applications on a platform fully managed by the provider. In this scenario, only the application and data is managed by the customer. The environment to build, test, and run applications are offered for that purpose. Some examples are *OpenShift*, *Heroku,* and *PaaS,* offered by cloud providers such as *AWS Elastic Beanstalk*, *Google App Engine,* or *WindowsAzure*.
- **Infrastructure as a Service (IaaS)**: This service offers the infrastructure required by the customer to provide solutions to end users. Networking, storage, servers, virtualization (if needed), operating system, and the rest of the elements needed to offer the solutions are managed by the administrator of the *IaaS* (the client who requested it). Popular private *IaaS* includes *OpenStack,* and popular public solutions include *DigitalOcean*, *Rackspace,* and services offered by public clouds such as *AWS*, *Google Cloud, IBM Cloud, and Microsoft Azure*, among others.

The following figure compares the three popular types, indicating the elements managed by the customer as well as the elements managed by the service provider:

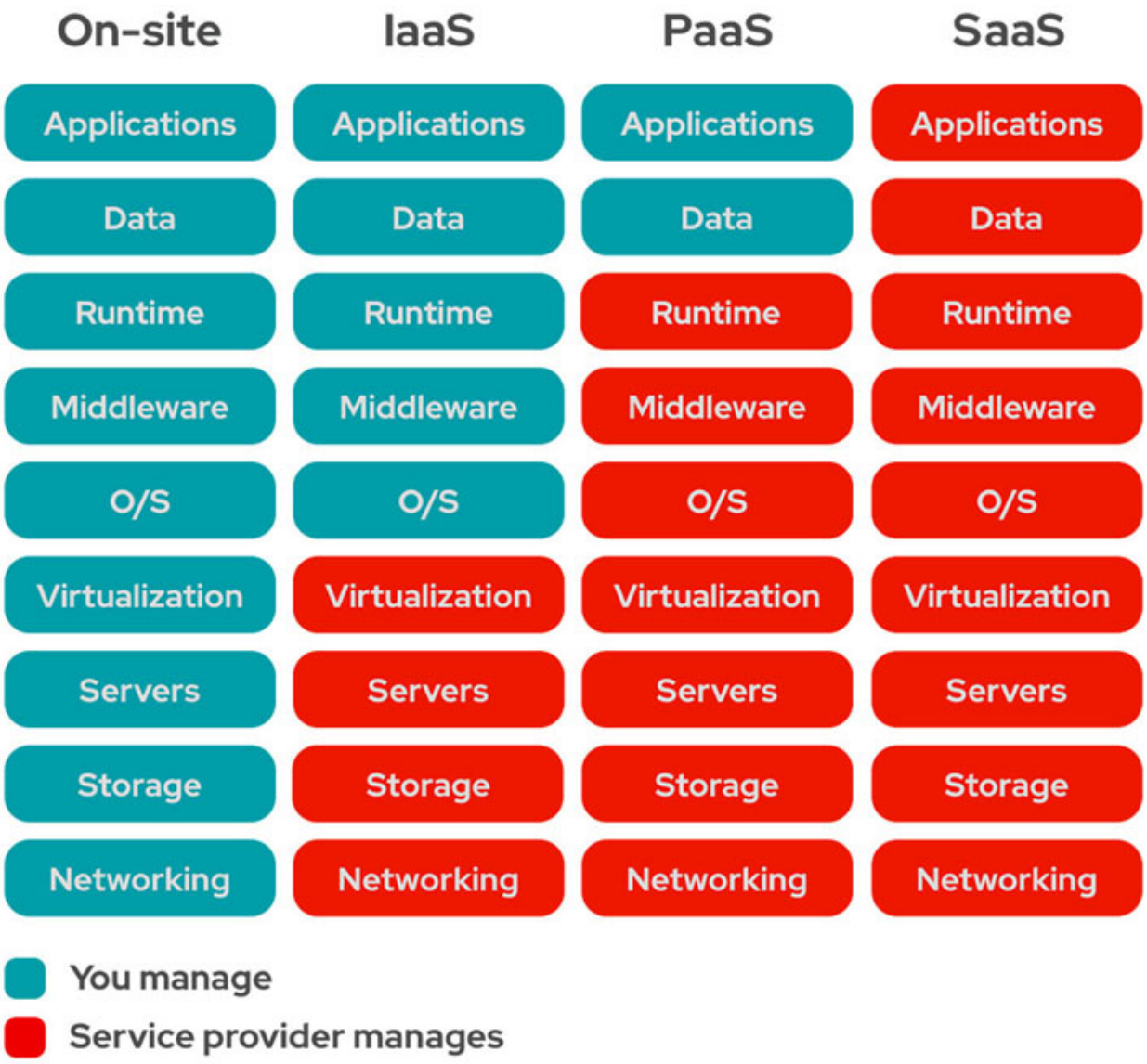


Figure 1.2: *Differences between On-Site, IaaS, PaaS, and SaaS. Source: Red Hat*

Automation services

Automation and integration services are key parts in the implementation of solutions. With automation solutions, companies can design and implement processes to reduce human interventions, including application release, providing new services, or scaling existing ones. Popular automation tools such as *Ansible* and *Terraform* are widely used to facilitate the orchestration and integration of new services.

Emerging technologies

Access to technology and digital services for most of the world's population was the biggest success of the *Digital Revolution*, also known as the *Third Industrial Revolution*. The IT sector is evolving and changing frequently with new technologies and innovations, and currently, we are part of the *Imagination Age* (*Fourth Industrial Revolution, Industry 4.0*), where the trend is automation and data exchange. Some of the top emerging technologies are as follows:

- **Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning**: Artificial Intelligence is a sub-field of computer science that explores how machines can imitate human intelligence. Machine Learning is the science to learn from seen data and then create models to make predictions from data. Deep Learning is a subset of machine learning where artificial neural networks learn from large amounts of data. Open-source tools and libraries for AI/ML are available; some examples are *TensorFlow*, *Keras*, *Scikit*, and *PyTorch*.
- **Big Data**: It is a term to describe large or complex volumes of data that can be structured or unstructured. Nowadays, the term technology is the software utilities designed for analyzing, processing, and extracting information from the data. *Apache Hadoop* and *Apache Spark* are two of the most popular solutions in this category.
- **Augmented reality (AR) and Virtual Reality (VR)**: Augmented reality is a real-time experience interacting with objects that reside in the real world but are virtually enhanced by computer-generated perceptual information. Virtual reality is an experience of an artificial environment provided by a computer. Some open-source options are *ARToolKit+*, *ARCore*, and *AR.js*.
- **Internet of the Things (IoT)**: This technology refers to the connection of physical objects embedded with sensors and software that transmit and receive data from the internet. That includes cars, home appliances (*smart homes*), cameras, and other common devices. Many Linux distributions are available for IoT: *Ubuntu Core*, *Raspberry Pi OS*, and *Yocto project* (to create custom embedded Linux-based systems).
- **Edge computing**: This technology is the distribution of computing topology closer to the source of the data. This is a key part of the Internet of Things and the 5G connections (described in the next section). For the *Infrastructure-as-a-Service* and Hybrid Cloud,

multiple solutions running on Linux to provide edge computing: *OpenStack* for virtualization and *Kubernetes* for containers, are the most popular.

Telecommunications services

This area includes communication equipment and services related to telecommunications. This industry experienced a big transformation during the last few decades. Evolution from 2G (second generation) mobile networks infrastructure based on **Global System for Mobile** (**GSM**), starting in 1991 and used for primarily for calls and SMS, to **third generation** (**3G**) based on network architecture named **Universal Mobile Telecommunications System** (**UMTS**), starting in 2001 and having internet access, has taken place. Most recently, in 2009, the **fourth generation** (**4G**) of mobile infrastructure came out, which puts forth a huge difference related to the data rate available for mobile devices today, allowing high bandwidth access to multimedia content, video and voice calls using technology **Long-Term Evolution** (**LTE**) standard broadband communication.

The first implementations of telecommunications services were on closed proprietary platforms using hardware-driven means. This was forcing companies to pay for license and hardware from one vendor while also having a monolithic infrastructure. Migration to a modular, using interchangeable systems and multi-vendor platforms, opened opportunities for Linux and Open Source. Recent migration of different telecommunication infrastructures to virtualized infrastructure, as well as the virtualization of *network functions*, positioned Linux as the main player as an operating system for that transformation.

In 2019, a new broadband technology standard generation, that is, 5G, has come forth that provides better speed transmissions and new features to users, and for emerging technologies described previously, like the Internet of Things (*IoT*) and Augmented Really (*AR*). These new features include low latency that is needed for the sensors and embedded devices, as well as better availability and coverage. The fifth generation implementation is based on cloud services, where a transformation takes place from virtualized network functions to containerized network functions. In this transformation, Linux and open-source technologies are a key part of the implementation.

Latest features in Linux

Linux has been evolving and implementing new features based on the needs from the new emerging technologies and the sector requirements. The Linux kernel is the core of the operating system, providing an interface between the applications and hardware, and it provides multiple functionalities. Linux distribution releases include recent versions of the kernel and updated versions of different tools. Some popular latest features in Linux include the following:

- **Live patching**: allows keeping the Linux server updated to the latest kernel version without the requirement of rebooting the system, which was previously needed. This feature has been around for many years, but the latest version of distributions includes mature tools for this. Some of the implementations for this feature are *Ksplice*, *Kpatch*, *Livepatch*, and *KernelCare*.
- **ExFAT support**: latest versions of Linux Kernel supports the popular Windows filesystem for flash memories.
- **Control Groups (v2)**: A mechanism to limit and isolate resources (CPU, memory, and disk I/O, for example) of a process or collection of processes. Software such as *Docker*, *systemd,* and *libvirt* uses this mechanism.
- **Nftables**: This low-level firewall solution has become the default in many distributions replacing the popular *iptables*. This software provides filtering and classification of network packets.
- **eBPF**: is a technology that can run sandboxed programs in the Linux kernel without changing the kernel source code or loading a kernel module.

Linux versus other operating systems

One of the main questions that customers using other operating systems, mainly Microsoft Windows, have is regarding the reason why Linux should be used and about the complexity related to its installation and maintenance.

The biggest advantages to using Linux compared to other operating systems for servers, desktops, or devices are as follows:

- **Open Source**: The source code for Linux kernel, libraries, and applications are available for everyone to review, collaborate, and create new content to share with others.
- **Free:** Download and installation of Linux is for free, with the possibility to have support from the most advanced companies in the IT sector.
- **Easy to install and maintain**: Installation of any popular Linux distribution is an easy task that can be performed by regular users or advanced administrators. Maintenance of a Linux system requires less effort than, for example, a Microsoft Windows Server due to less complexity to perform tasks such as updating software or updating the Linux distribution.
- **Software installation**: Installing software on Linux is easier than on other operating systems, thanks to the package manager and utilities around them. Repositories contain the software available, and the utilities to install packages resolve the required dependencies when we perform a software installation.
- **Mature, stable, and secure**: In more than 30 years of its existence, Linux has demonstrated that it is the most mature and secure operating system available. From small companies to the biggest corporations, all are strongly committed to using Linux to run critical applications on top of it.
- **Commodity hardware**: The requirements to run Linux compared to other operating systems do not require having the latest hardware. Linux can be run in old systems with less power resources, on new systems embedded with limited resources, or in virtual machines without the need to allocate a big quantity of memory or CPU cores.
- **Customisation**: Linux, compared to other operating systems, is the one with more options to customize to the desire of the regular user or to the system administration. Everything in Linux is possible to be configured to the requirements needed.

Promising future of Linux

As was described, Linux is currently a key factor in all the IT sectors and will continue being the operating system for most of the solutions in

emerging technologies, as companies rely on open source and in the features offered by Linux. In the near future, containers will go on being used to deploy new services; new architectures like ARM will be more available in the market, and the use of public and clouds will continue growing since companies are in an ongoing migration from on-premise infrastructure to a cloud one. Distributions that are generally available are as follows:

- **Architectures**: Linux Kernel is available for most of the architectures; some examples: *ARM/ARM64*, *IA-64*, *MIPS*, *PowerPC*, *RISC-V*, and *x86*.
- **Cloud image ready**: Most of the Linux distributions offers cloud image ready to be launched in public (*AWS*, *Azure*, and *Google Cloud*) and private clouds (*OpenStack*).
- **Container images**: Linux distributions are offering official images to run containers, using, for example, *Docker* or *Kubernetes*.

Conclusion

Linux has been the most important operating system in the IT sector for the last few decades, as well as the present. The future will bring new innovations in emerging technologies, and Linux will be an integral, if not the main, part of most of those implementations. Regular desktop users are getting more comfortable using Linux distributions as the main operating system, and developers have decided to move to Linux from other operating systems.

Key facts

- Linux is the most popular operating system in the world.
- Linux works in most of the architectures available.
- Linux is available on all the popular clouds.
- Most of the emerging technologies are using Linux.
- Containers are Linux.

Questions

1. All the Top 500 list of fastest supercomputers run Linux.

    a. True
    b. False

2. Android is a modified version of Linux.
    a. True
    b. False

3. What SaaS stands for?
    a. Service as a Service
    b. System as a Service
    c. Software as a Service

4. What PaaS stands for?
    a. Programming as a Service
    b. Platform as a Service
    c. Program as a Service

5. What IaaS stands for?
    a. Innovation as a Service
    b. Integration as a Service
    c. Infrastructure as a Service

Answers

1. a
2. a
3. c
4. b
5. c

CHAPTER 2

Linux Installation

Introduction

Installation of a Linux distribution is an easy task, requiring a little amount of time for the process. The number of actively maintained distributions is more than 300, each of them with the same main components (Linux kernel and general system and system libraries) but with different purposes and specific system tools. The difference between distributions is often related to package managers, desktop environments, and system services, like firewalls and network services.

Popular Linux distributions can be installed on different architectures and devices. Other distributions are available and optimized to be used in specific environments, for example, *Raspberry Pi Linux OS* (For Raspberry PI devices) or *OpenWrt* (for routers).

The popular website **distrowatch.com** contains a list of active distributions with useful information about the current version, the architecture supported, the default software included, and much more useful information.

This chapter will be focused on the installation of popular distributions, especially because of the support for enterprises. It covers the installation methods available, the steps of the installation, the advanced options, and the differences between those distributions.

Structure

In this chapter, we will discuss the following topics:

- Linux Support Levels
- Installation Methods
- Common Installation Steps
- Advanced Installation Steps
- Debian GNU/Linux

- Ubuntu Server
- Red Hat Enterprise Linux
- CentOS and CentOS Stream
- Rocky Linux and Alma Linux
- SUSE Linux Enterprise Server and openSUSE
- Other Distributions with Commercial Support

Linux support types

Knowing the purpose of the server is the most important factor in deciding which Linux distribution to use. Customers with the requirement of running high available services, mission-critical applications, or the need to run applications certified to run in specific distributions would require a commercial distribution where professional support with advanced engineering skills will provide the support. The main companies offering professional support and the distribution associated are the following:

- **Red Hat** offers *Red Hat Enterprise Linux* with three levels of support:
  - **Self-support**: Access to Red Hat Products and access to the knowledge base and tools from the Customer Portal.
  - **Standard**: Access to support of engineers during business hours.
  - **Premium**: Access to support engineers 24 × 7 for high-severity issues.
- **Canonical** offers commercial support for *Ubuntu Server* with two options:
  - **Ubuntu Advantage for Applications**: Security and support for open-source database, logging, monitoring, and aggregation services (LMA), server, and cloud-native applications.
  - **Ubuntu Advantage for Infrastructure**: Security and IaaS support for open-source infrastructure.
- **SUSE** offers two commercial supports for the *SUSE Linux Enterprise Server*:
  - **Standard**: Includes software upgrades and updates, unlimited support 12 × 5 using chat, phone, and Web.

    - **Priority**: Same support as the standard one but with 24 × 7 support.
- **Oracle** is offering two commercial support for *Oracle Linux:*
    - **Basic Support**: Includes 24 × 7 telephone and online support, including support for high availability with *Oracle Clusterware*, *Oracle Linux load balancer,* and *Oracle Container runtime for Docker*.
    - **Premium Support**: Adds up to the basic support applications like *Oracle Linux Virtualization Manager, Gluster Storage for Oracle Linux,* and *Oracle Linux software collections.*

Other popular distributions for servers have security teams for serious vulnerabilities, and bugs and issues are supported by the volunteers.

- **Debian**: The security team gives support to a stable distribution for about one year after the next stable distribution has been released.
- **Alma Linux**: CloudLinux is committed to supporting AlmaLinux for 10 years, including stable and thoroughly tested updates and security patches.
- **Rocky Linux**: Provides solid stability with regular updates and a 10-year support lifecycle, all at no cost.

Another important key point to choose the distribution and the version to be installed related to the support is the **Long-Term Support** (**LTS**) offering. This is crucial for companies running critical applications where the upgrade of the distribution is not always possible or recommended, and the requirement to have support during a long period is essential.

Red hat enterprise Linux long-term support

With the introduction of Red Hat Enterprise Linux version 8, Red Hat simplified the RHEL product phases from four to three: Full Support (five years), Maintenance Support (five years), and Extended Life Phase (two years). The following figure shows the Red Hat support lifecycle:

Figure 2.1: *Life cycle support for red hat enterprise Linux. Source: Red Hat*

Ubuntu server long-term support

For each *Ubuntu LTS* release, **Canonical** maintains the *Base Packages* and provides security updates for a period of 10 years. The lifecycle consists of an initial five-year maintenance period and five years of **Extended Security Maintenance** (**ESM**). The following figure shows the Ubuntu support lifecycle:

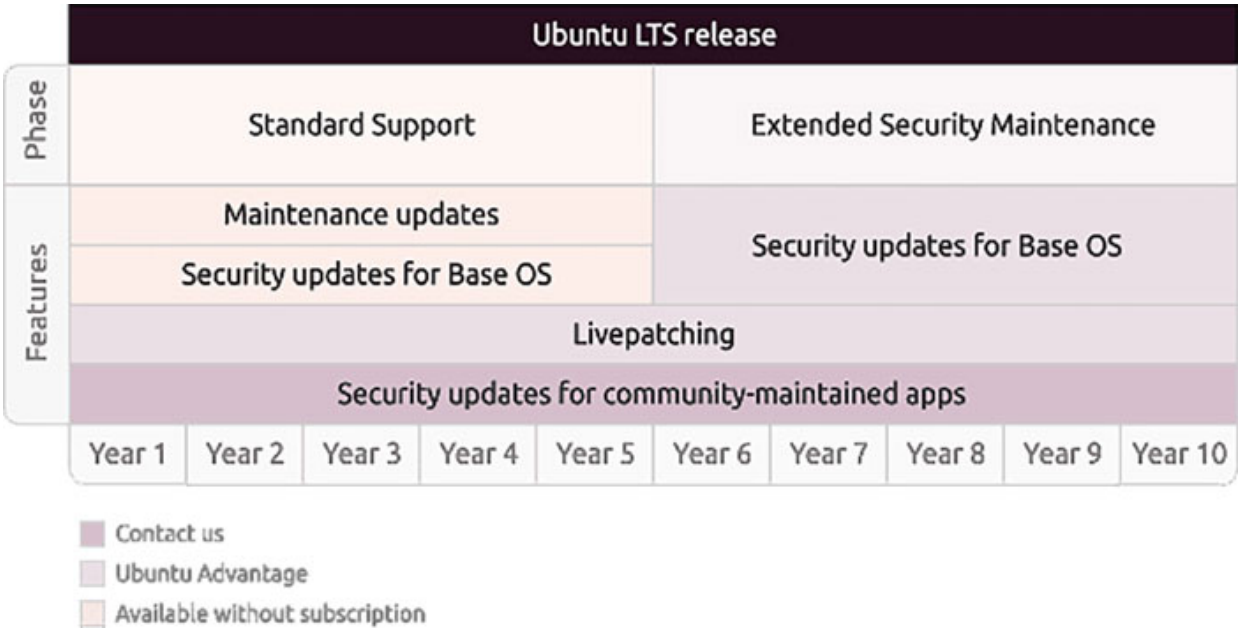


Figure 2.2: *Life cycle support for Ubuntu Server. Source: Ubuntu*

SUSE Linux enterprise server long-term support

Long-term Service Pack Support complements the existing SUSE Linux Enterprise Server subscription. LTS Pack Support offers the options:

- An additional 12 to 36 months of defect resolution and support as you postpone or defer migration to the latest service pack.

- An additional 12 to 36 months of technical support through the Extended Support phase. The following figure illustrates the SUSE support lifecycle:

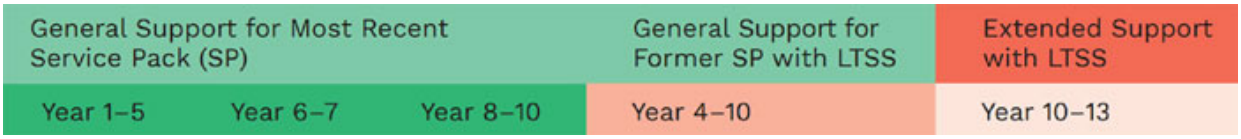


Figure 2.3: *Long-term service pack support for SUSE Linux enterprise server. Source: SUSE.*

Oracle Linux long-term support

Oracle Linux Premier Support for releases 5, 6, 7, and 8 is available for 10 years after their release date. After that, support can be extended for additional years with *Oracle Linux Extended Support*, followed by *Lifetime Sustaining Support*.

Installation methods

Linux distributions include several different installation methods depending on the target and the requirements. Some examples of those targets are as follows:

- A local server where access to the physical bare-metal node is possible.
- A remote server without physical access.
- A bunch of physical servers, local or remote, with or without physical access, where manual installation would not be possible because of the number of servers.
- A virtual machine.

The common installation methods depending on the needs, are as follows:

- **Full installation DVD or USB:** The image used for the installation contains all the requirements for a normal installation, and access to the network is optional.
- **Minimal installation using DVD, CD, or USB:** The image contains the minimum files necessary to start the installation, and the rest of the process requires access to the internet to download the required packages or access to a repository in the local network.

- **PXE server:** A *preboot execution environment* allows the installation of Linux during the boot process of the physical server or Virtual Machine. This process requires a server or device configured with a PXE Server, DHCP service, TFTP service with the installation files, and a syslinux bootloaders. Another alternative for modern servers is using iPXE, where the TFTP server can be replaced with a Web server containing the installation files.
- **Systemor cloud image-based installations**: These images have a preinstalled distribution and are ready to be used. These images are generally used on cloud platforms and virtualization platforms. These images can be reconfigured for either the user's needs or the system can be customized in the first boot.

Installation methods include the following methods when the installation is loaded:

- **Graphical User Interface (GUI)** based installation.
- Text-based installation.
- Advanced installations, including automatic installations without user interaction.

Common installation steps

The installation process is similar for all the Linux distributions except for specific configurations related to them. Common installation steps are the following:

1. Download the installation media and write to a CD/DVD/USB or boot it from the network.
2. Boot the system with the media.
3. Specify the interface type of the installation: **Graphical User Interface (GUI)** or **Text Based Interface (TUI)** installation.
4. Select a language for the installer. The language specified will be used during the installation and will be configured in some cases as the primary language after installation.
5. Clock and Time Zone configuration. Indicating the location of the clock and the time zone will be adjusted for the installation and for the

system after it.

6. Keyboard configuration. In this step, the layout for the keyboard to be used during and after the installation is configured.
7. Network configuration. This step includes the hostname and domain to be used for the system. Interface network configuration can be customized to be automatic (DHCP) or static IP configuration.
8. Storage configuration. in this step, the disk target is selected, and the partition layout is configured, indicating if a full disk will be used or a partition layout will be used.
9. Specify the root password and create a new user. After installation, it is highly recommended to use a non-administration user, and this step includes the creation of one user and setting a secure password.
10. Specify the repositories and indicate the software to be installed. At this point, the repositories (remote or local) are specified, and the software is to be installed depending on the purpose of the system, for example, Desktop environment, Web server, and SSH server software.
11. Start the installation. During this stage, the disk is partitioned, and the core software and the additional software are installed. Next, the networking is configured, and the user specified is created.
12. Complete the installation. The last step is used to complete the installation and configuring the boot loader. After the installation is completed, the system should be rebooted to boot from the installed distribution.

Three installations with the steps to perform a basic installation will be described in this chapter: Debian, Ubuntu Server, and Red Hat Enterprise Linux. Other popular distributions will be just described, and the available versions will be explained.

Advanced installation steps

During installation, there are three common advanced installation steps:

- **Link aggregation**: Allows to combine multiple Ethernet interfaces into a single logical link to work in parallel in active-passive mode or in active-active to sum the available throughput. In Linux, the terms used for link aggregation for the interfaces are bonding or teaming.

    - Redundancy for high availability can be configured as a round-robin or as an active backup.
    - To sum up the throughput, a Link Aggregation Control Protocol (LACP, 802.1AX, or 802.3ad) is required to be used and configured in the system and in the switches where the network interfaces are connected.

- **Volume Manager**: Linux distributions use the LVM tool for volume manager, which stands for Logical Volume Management. Using LVM provides flexibility and more advanced features than traditional partitioning. Installers usually offer to encrypt the data inside the disk using Linux Unified Key Setup (LUKS).
- **Data redundancy**: Technology to spread the data across several disks having multiple copies of the same information stored in more than one place at a time. RAID stands for Redundant Array of Independent Disks, which is usually implemented at the hardware level. However, Linux provides the possibility to reconfigure this using software. Depending on the required level of redundancy and performance, different RAID levels are available where popular ones are as follows:
    - **RAID-1 or mirror mode:** Two or more disks are combined into one volume, and all the blocks are copied to all the disks. All the disks except one can fail, and the data would still be accessible.
    - **RAID-5:** The most popular and useful RAID mode, this requires three or more disks. The data will be distributed between the disks, but they will not be mirrored. Even if one disk fails, the data will be available. The data of the failed disk can be recalculated using the parity method.
    - **RAID-6:** An extension of RAID-5, where many disks are used (requiring to have at least four). At this level, two disks can fail, and the data would still be accessible.

      Spare disks are an important part of storage redundancy, especially for RAID-5 and RAID6. These disks are not used to distribute the data, but there are on standby to be filled if one of the active disks has failed.

Debian GNU/Linux

Debian GNU/Linux is one of the most popular distributions for Linux servers due to the stability offered. The distribution is a result of a volunteer effort to create a free and high-quality distribution with a suite of applications.

Debian was the first Linux distribution to include package management, named `dpkg` (Debian Package), for easy installation, update, and removal. It was the first distribution as well to be able to be upgraded without requiring a re-installation.

To obtain the images for the installation, navigate to the following website where the different options are available **https://www.debian.org/distrib/**. Installation method options are the following:

- A small installation image: This installation requires an Internet connection to be completed. It is also called *Network Install*.
- A complete installation image: It contains more packages, and it makes the installation faster without the dependence on an Internet connection. There are options with DVD and CD images.
- **A live CD**: It is possible to boot a Debian system from a CD, DVD, or USB without installation. This image will boot in memory the Linux distribution, and it can be tested without performing any changes in the disk. After testing it, the live CD would allow us to perform the installation on the disk.
- **Cloud images**: Debian, like other Linux distributions, offers images with preinstalled versions to run in the private or public cloud. For example, QCOW2/RAW images for OpenStack, AMI images for Amazon EC2 or using AWS Marketplace, and Microsoft Azure images on the Azure Marketplace.

Installation menu

The first screen appearing when the server is booting from the Debian installer image, indicates which installation type is desired. The following figure shows the first screen of the installer:

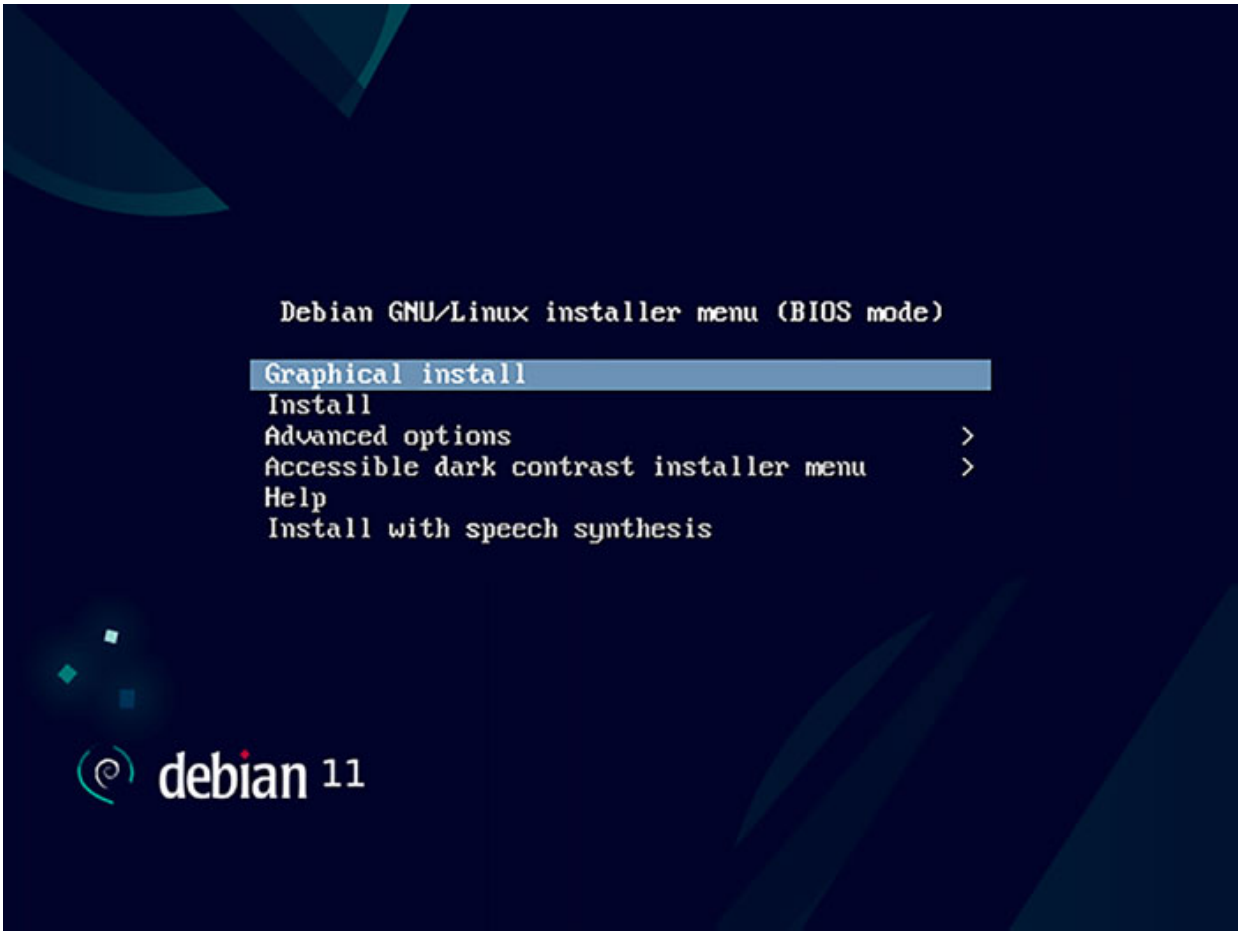


Figure 2.4: *Debian installer menu*

Selecting the Graphical install entry in the menu will move the installer to the next step as a continuation is shown.

Select a language

The first step after specifying which installation user interface would be used is to specify the language to be used during the installation process. This language, as indicated, will be used as the default language once the system is installed.

Select the location

After specifying the language to be used during the installation and to be used as default, the next step is to select the location to be used. This location will be used to set the time zone and the system locale. If the

location wanted is not in the list shown, it is possible to select “other” and afterward specify the continent/region and then the country.

Configure the keyboard

Once the correct location is indicated, the next configuration is the Keymap for the keyboard to be used.

Pressing Continue in this step will start to load the modules for the different components of the system (storage, network, and other devices), and it will try to configure automatically the network using DHCP.

Configure the network

This step includes two parts when DHCP is used: specify the hostname for the system and indicate the domain to be used. The following figures show the wizard steps to introduce the hostname and the domain:

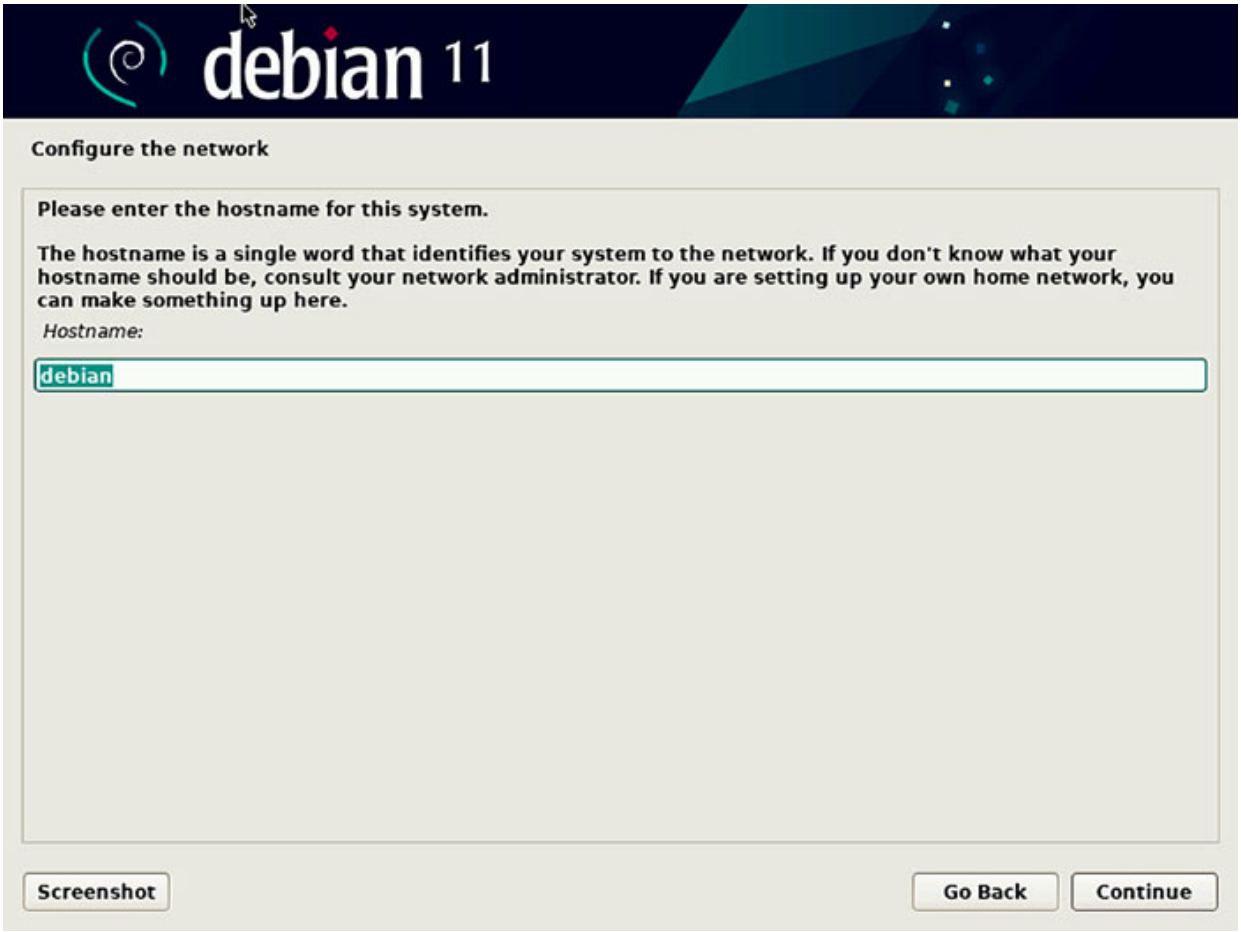


Figure 2.5/2.6: *Configure the network*

If the network is not configured automatically with DHCP, manual static IP should be specified after the hostname and domain are set. The following figure shows the step to configure a static IP:

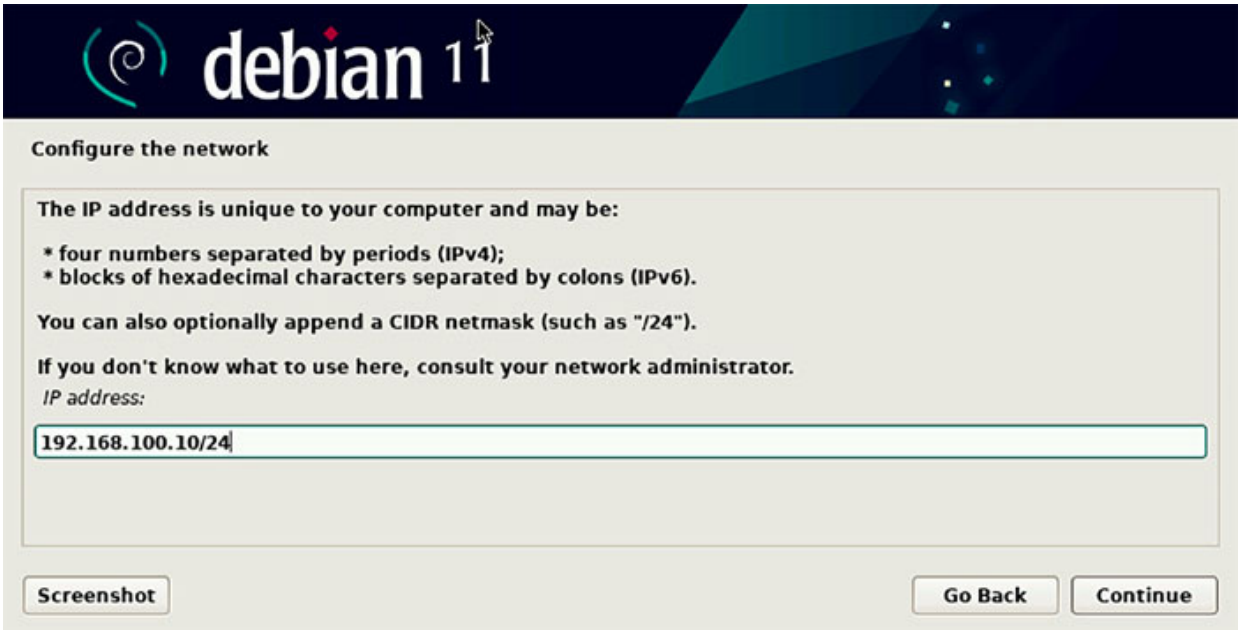


Figure 2.7: *Configure the network*

Set up users and passwords

After the network is configured, the following step is to set the password for the root user. The password has to be a secure one, or it will not be accepted. If the password in this step is not set, the root account will be disabled, and the regular user account to be created will have the possibility to run commands or switch to root using *sudo* (a command allowing delegating authority for administrative tasks).

After setting the password for the root user or keeping it empty to disable the user, the next step is to set the information for the new user to be created. The first dialog is to specify the full name of the new user. The second information required is the username for the new user. This username is going to be used to login into the system. And after deciding on the username for the account, the password would be specified.

Configure the clock

The next step is configuring the clock for the system. Keeping the time synchronized with the correct time is really important for some systems and for the system administrators when they are troubleshooting.

Partition disks

The installer allows to do manual partitioning disk or uses the guided option. Using the LVM is highly recommended to have dynamic partitions and more flexibility in managing disks and partitions. The following figure shows the partition disk step:

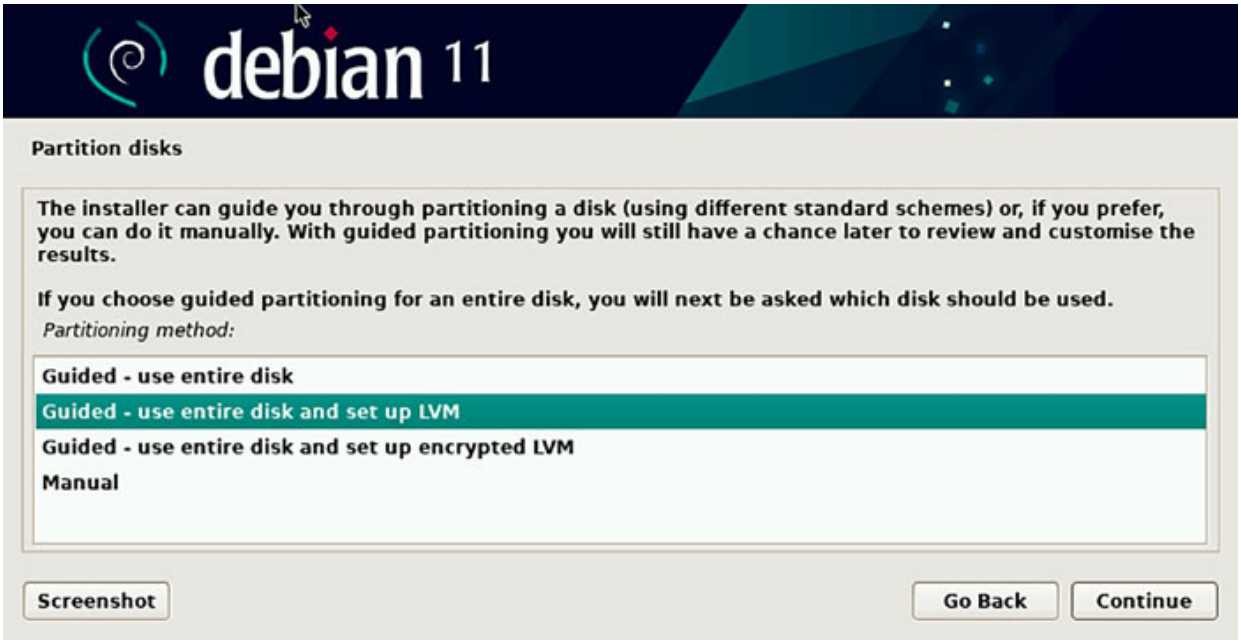


Figure 2.8: *Partition disks*

After indicating the partition method, a list of the disks available in the system is listed to specify in which disk the partitions will be created. Once the disk where the installation will be performed is selected, the installer gives some partition options (if the guided option was selected). The following figure shows the partitioning options:

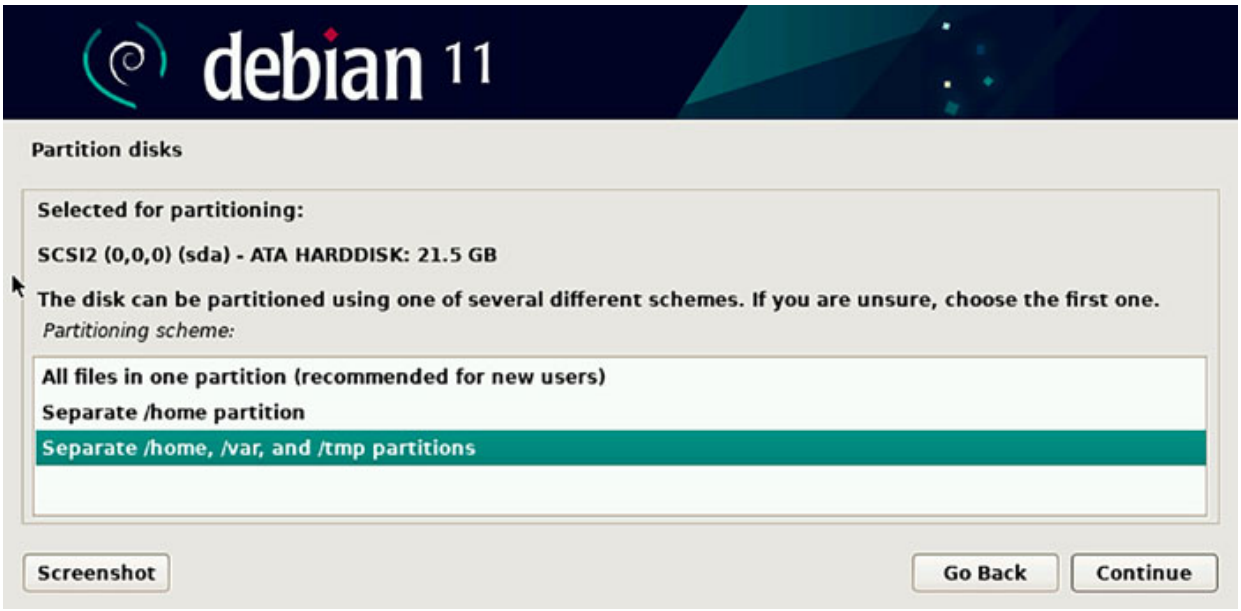


Figure 2.9: *Partition disks*

The next step related to storage is to confirm the changes to be performed on the disk. It is important to ensure that the correct disk is selected before the next steps are taken. The following figure shows the confirmation steps:

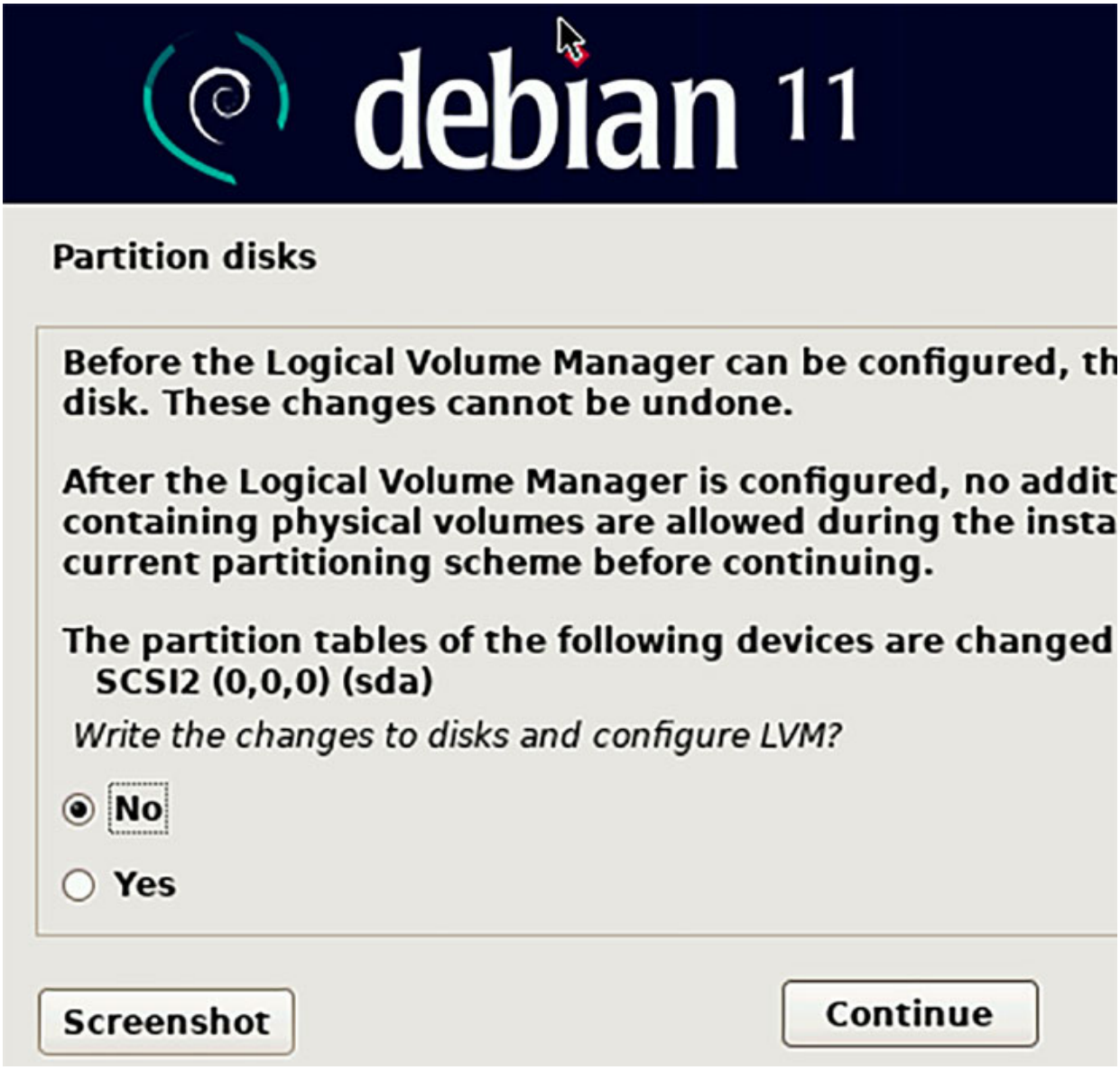

debian 11
Partition disks
Before the Logical Volume Manager can be configured, th
disk. These changes cannot be undone.
After the Logical Volume Manager is configured, no addit
containing physical volumes are allowed during the insta
current partitioning scheme before continuing.
The partition tables of the following devices are changed
SCSI2 (0,0,0) (sda)
Write the changes to disks and configure LVM?
No
Yes
Screenshot
Continue

Figure 2.10/2.11: *Partition disks*

Install the base system and install software

After the partition process, the installer will install the base system in the disk, and partitions will be configured. In this step, the Linux kernel, the system tools, libraries, and services will be installed and configured. When the installation of the minimal required packages for the base system is completed, it is possible to scan extra installation media or skip that process. Then it is needed to configure the package manager specifying the mirror country for Debian archive files and the server to be used. The following figures show the steps to configure it:

Figure 2.12/2.13: *Configure the package manager*

If a proxy is needed to access the internet repository, the provides the option to specify an HTTP proxy and, if needed, the username and password to use it. The installer will retrieve the package list from the server repository, and it will continue with the installation of the base system. After the core of the system is selected, the installer gives the option to specify the software to be installed during the process, as is shown in the following figure:

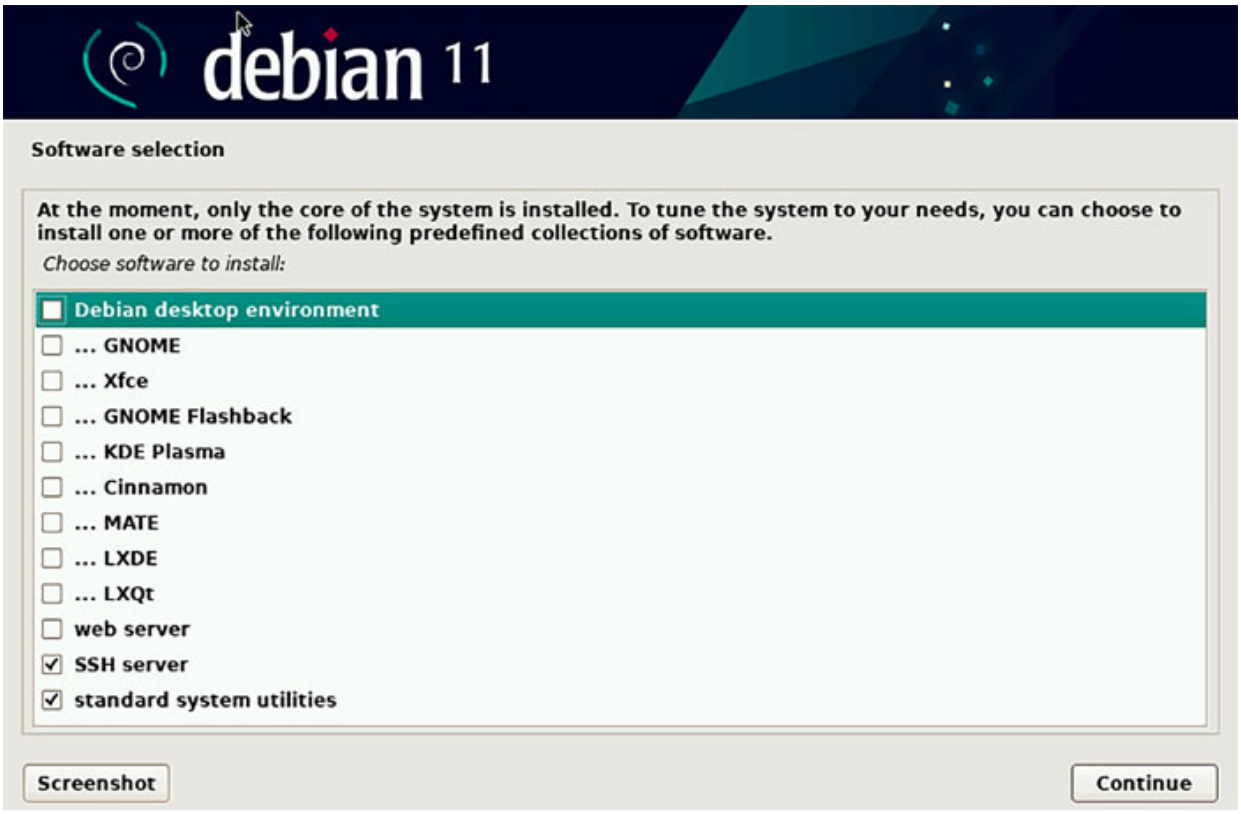


Figure 2.14: *Software selection*

Install the GRUB boot loader

After the base system and the software selected are installed, the last step of the installation is to install the boot loader on the disk. **Grand Unified Bootloader** (**GRUB**) is the boot loader predominant in Linux systems. The following figure shows the confirmation step:

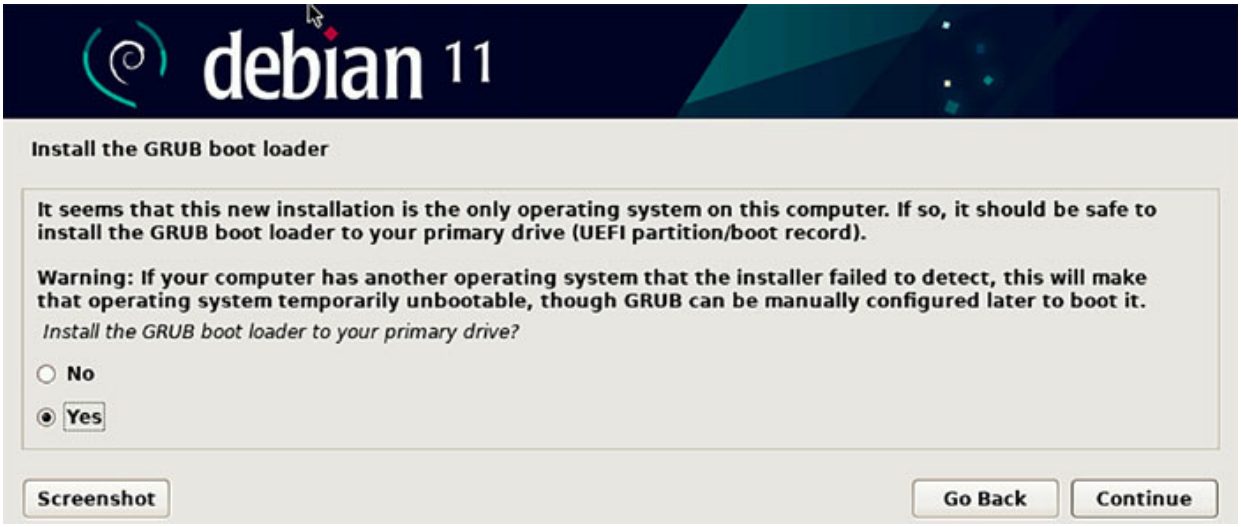


Figure 2.15: *Install the GRUB boot loader*

After the acceptance of the installation of GRUB, a list of devices and partitions will be listed to confirm where the bootloader will be installed. The installation will be completed, and the system can be rebooted to the new system, ensuring the media installation is removed to boot from the local disk.

Booting the system will show the GRUB menu, where it is possible to boot to Debian, to different distributions (if they are installed in different partitions), or start Debian in a recovery mode or with different kernel versions (Advanced Options).

Ubuntu server

Ubuntu is the most popular distribution for Desktop. It uses the same package manager (dpkg) as Debian. Ubuntu offers a faster release lifecycle compared with Debian, having the latest version of the software available. The canonical company, the publisher of Ubuntu, offers commercial support as described previously.

Ubuntu offers distribution variants through the official URL **https://ubuntu.com/download/,** including the following options:

- **Ubuntu Desktop:** This variant is the most popular for PCs and laptops, offering guaranteed support for five years with free security and maintenance updates.

- **Ubuntu Pro:** This variant is optimized and certified for cloud providers such as Amazon AWS, Microsoft Azure, Google Cloud Platform, IBM Cloud, and Oracle.
- **Ubuntu Core:** This option is optimized for Internet of Things (IoT) and the Edge.
- **Ubuntu Server:** Supported as described previously for 10 years. Used to be installed in private or public data centers and supporting multiple architectures.

Installation menu

The first screen appearing when the server is booting from the Ubuntu installer image starts the installation or tests the memory of the system. Ubuntu Server does not include a graphical user interface for the installation. The following figure shows the first screen of the installer:

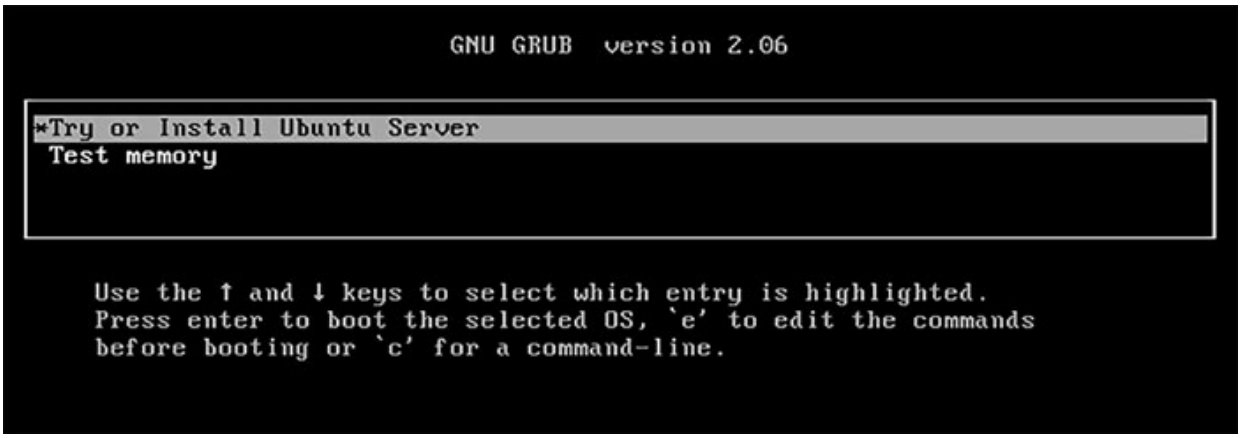


Figure 2.16: *Ubuntu installation menu*

Select installation language

After specifying the option to *Install Ubuntu Server,* the installer will move to the next step to specify the language for the installation.

Keyboard configuration

The next step in the installation process is to specify the keyboard configuration: the layout and the variant.

The base for the installation

After specifying the layout and the variant, the installer wizard shows the types of possible installation options for Ubuntu Server: default installation or minimized (used where users are not expected to log in to the system). The following screenshots show the options and the descriptions:

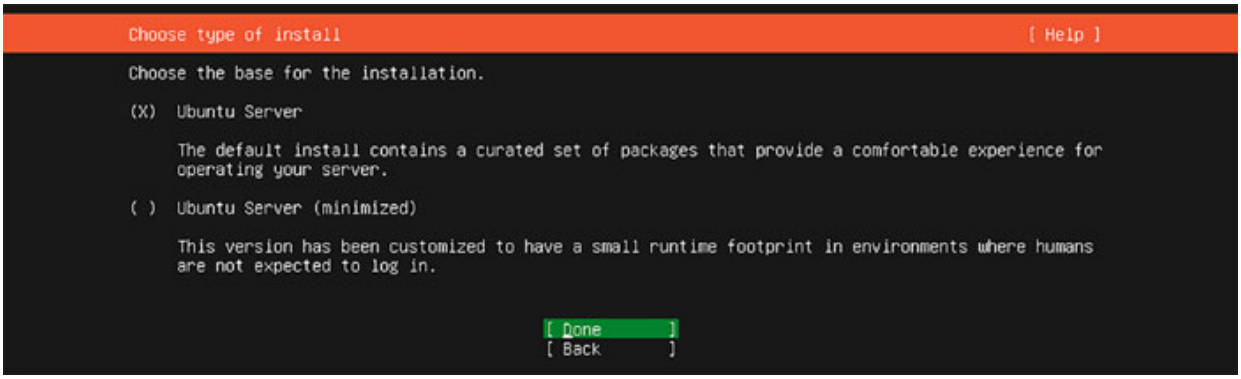


Figure 2.17: *Choose the type of install*

Network connections

The following step is the network configuration. If DHCP is used, the IP assigned will be shown. In case of no DHCP, a manual configuration of the IP is required to access the repositories. In this step, it is possible to configure a bonding interface. The following figure shows an example of a network automatically configured with DHCP:

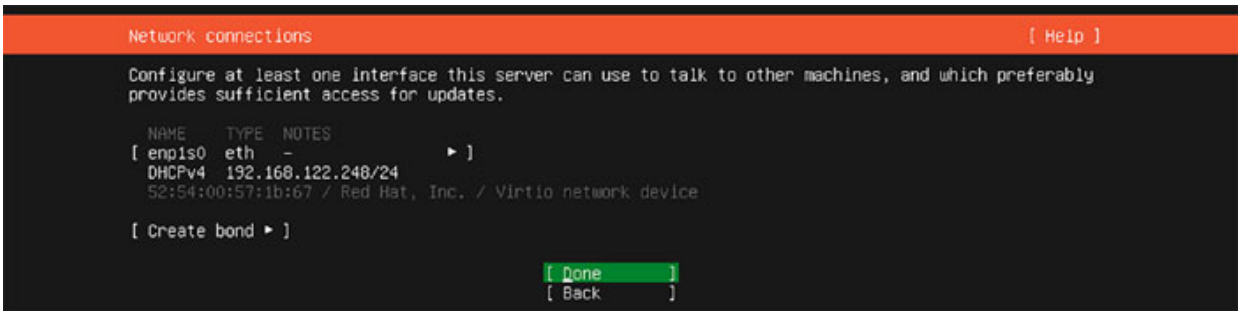


Figure 2.18: *Network connections*

After configuring the network interfaces, it is possible to configure the proxy to access the internet if it is needed.

Configure the Ubuntu archive mirror

An archive mirror is a Web server containing the packages to be used for the installation and to install software after the system is installed. The installer will detect automatically, based on the system location, the closest archive mirror URL, allowing the user to specify an alternative one.

Guided storage configuration

Ubuntu installer permits specifying how the storage would be configured if using a guided storage layout or creating a custom one. It is also possible to specify if the system uses Logical Volume Manager (LVM), what is selected by default, and if LVM would be encrypted with Linux Unified Key Setup (LUKS). The following figure shows an example of this step:

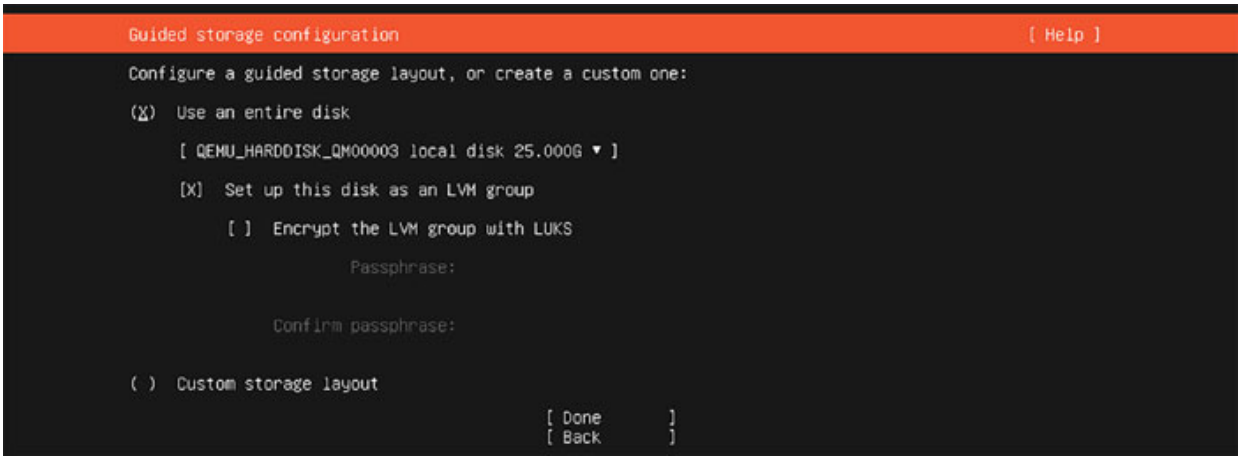


Figure 2.19: *Guided storage configuration*

After selecting the guided option and LVM, the next step will show the partition layout that will be applied to the disk. It is also possible to personalize the partitions to be created, the mount point, the sizes, and the names for logical volumes. The following figure shows a default partitioning:

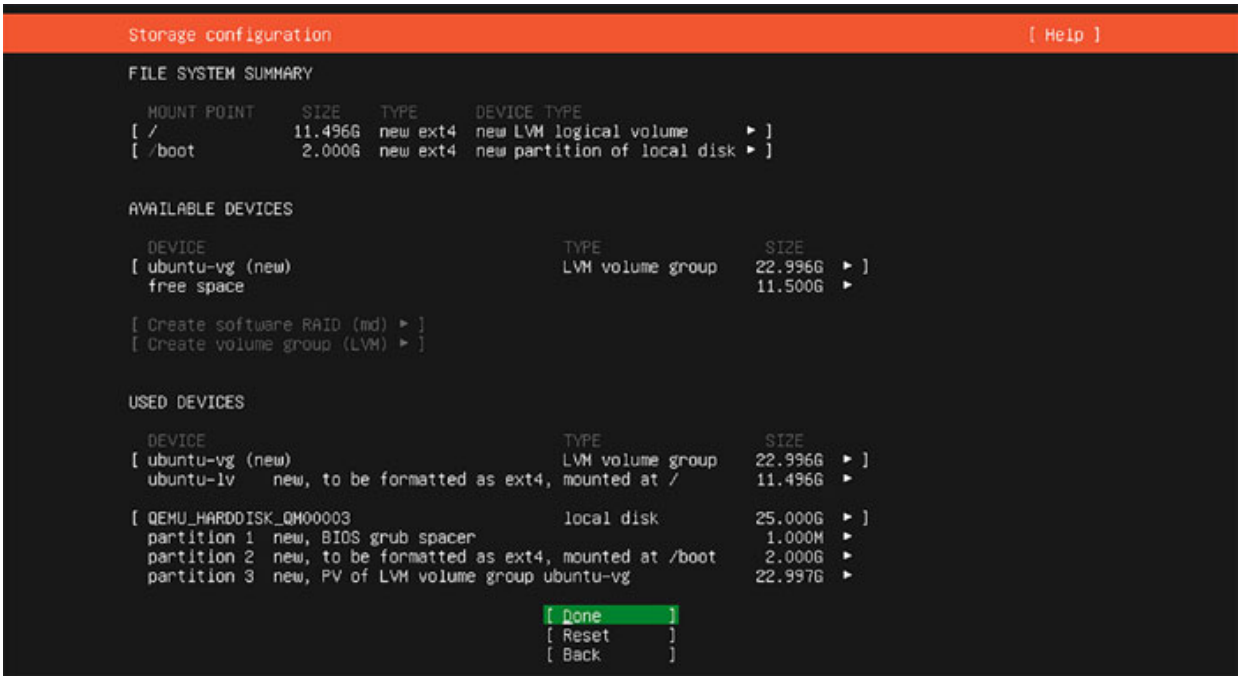


Figure 2.20: *Storage configuration*

Profile setup

After the disk layout is confirmed, the next step is related to the creation of a new user specifying the name, the username, and the password. The name of the server will also be specified in this step. The following figure shows the fields required to fill:

Figure 2.21: *Profile setup*

SSH setup

The installer will ask if the OpenSSH server will be installed and enabled to be administrated remotely. If the system is not a local system, it is recommended to enable this so that it can be administrated remotely.

Featured server snaps

Snap is a software packaging and deployment system developed by Canonical. The snaps are self-contained applications, including the needed libraries to run the service or application. Working with snaps, there are four important concepts: **`snap`** is the application package format and the command to execute the applications; **`snapd`** is the background service to manage and maintain; **`snapcraft`** is used to create customized applications and a marketplace for the snaps; and finally, **Snap Store** (**https://snapcraft.io/store**). The following figure shows some of the available snaps:

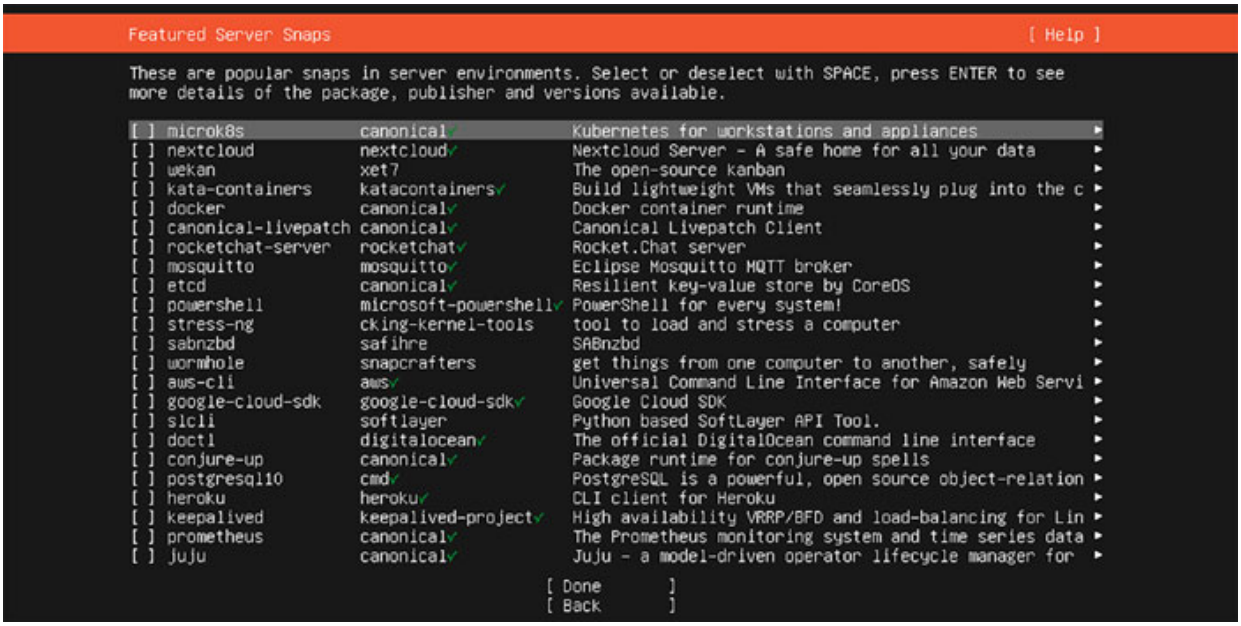


Figure 2.22: *Featured server snaps*

After selecting, if some of the Snaps are needed, the installation will start downloading and installing security updates. When the installation has finished, it is possible to review the full log of the installation process or reboot the system to boot the Ubuntu server installed on the disk.

Ubuntu Server installation, as observed, does not set the root password; it creates a user and sets the permissions in the system for that user to use *sudo*

for administrative tasks.

After the system is rebooted, it is possible to login with the user created during the installation.

Red Hat Enterprise Linux

Red Hat Enterprise Linux is the world's leading enterprise Linux platform, widely used in data center infrastructures, virtualization, and containers. As described previously, RHEL is a commercial open-source Linux distribution developed and supported by Red Hat for the commercial market.

RHEL uses **RPM Package Manager** (**RPM**) for package management, and this system is used in many distributions, such as CentOS, Fedora, Oracle Linux, AlmaLinux, or Rocky Linux.

Installation images for Red Hat Enterprise Linux are available on the website: **https://access.redhat.com/downloads/**. A subscription is required to access the repositories, with the option to request a no-cost Red Hat Developer subscription if the purpose is for development or a 60-day trial is available to try RHEL before buying the subscription.

RHEL offers three images for the installation:

- **Binary DVD:** It includes all required packages without the need for any additional repository. Useful when there is no internet access.
- **Boot ISO:** This image has the basic packages for the installation but requires an active network connection to access additional package repositories.
- **KVM Guest Image:** Used for KVM/QEMU hypervisor, such as Red Hat OpenStack (or community OpenStack) or Red Hat Virtualization (or oVirt project).

Installation menu

The first screen that appears when the server is booting from the Red Hat Enterprise Linux installer image allows us to install and test the media or perform troubleshooting. The following figure shows the first screen of the installer:

Figure 2.23: *RHEL installation menu*

Selecting the Graphical install entry in the menu will move the installer to the next step as a continuation is shown.

Select a language

The first step after specifying which installation user interface would be used is to specify the language to be used during the installation process. This language will be used as well as the default language once the system is installed.

Installation summary

After choosing the language, the installer shows a dashboard with the configuration requirements before starting the installation. Some configurations are detected automatically (that is, the network configuration and the installation source), and another configuration is required to specify manually (registration, disk partition, and user passwords). An example of the dashboard is shown in the following figure:

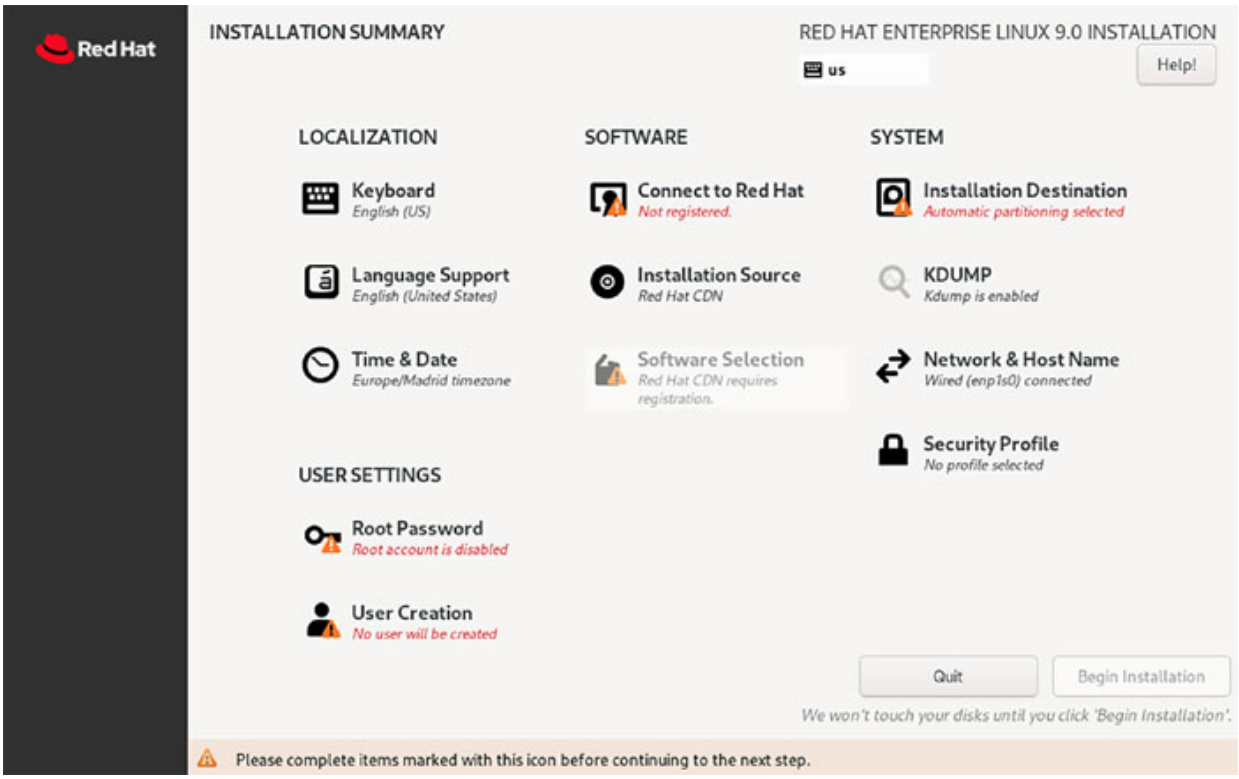


Figure 2.24: *Installation dashboard*

The steps required in this step are as follows:

1. Connect to Red Hat with the username/password for the subscription required.
2. Confirm the disk to be used for the installation and the partitioning desired.
3. **Software selection**: specify which software will be installed in the system.
4. Set the password for the root user.
5. Create a regular user.

Other options possible to configure are as follows:

- Network configuration.
- Configure the Keyboard layout.
- Configure the time zone, time, and date.
- Other options such as kernel dump (kdump) and security profile.

Connect to Red Hat

To subscribe to the system Red Hat, it is possible to login with a Red Hat account or use an activation key. Users can specify the purpose of the system, an optional configuration but recommended during the installation to ensure the correct subscription is attached. Purpose attached. The purpose of the system can be specified during installation or after installation using the command **`syspurpose`**.

Installation destination

Red Hat Enterprise Linux installer would automatically choose one of the disks available for the installation. It is required to confirm on which disk the installation will be performed, and it is possible to configure the partition layout automatically or manually. It is possible to encrypt the data during the installation process. LVM will be used when the automatic layout is selected. An example about an installation destination is shown in the following figure:

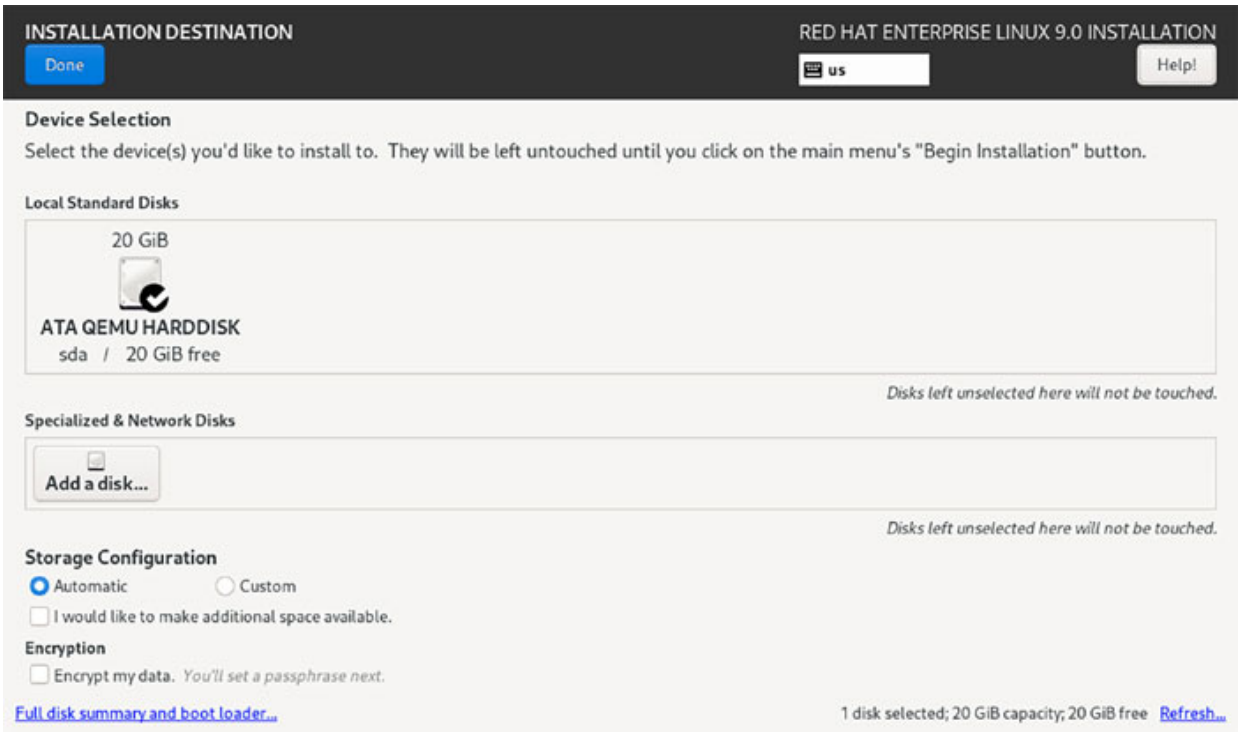


Figure 2.25: *Installation destination*

Software selection

In this step, it is possible to indicate the base environment and the additional software to be installed in the system. The following figure shows the available options for the base environment and for the additional software:

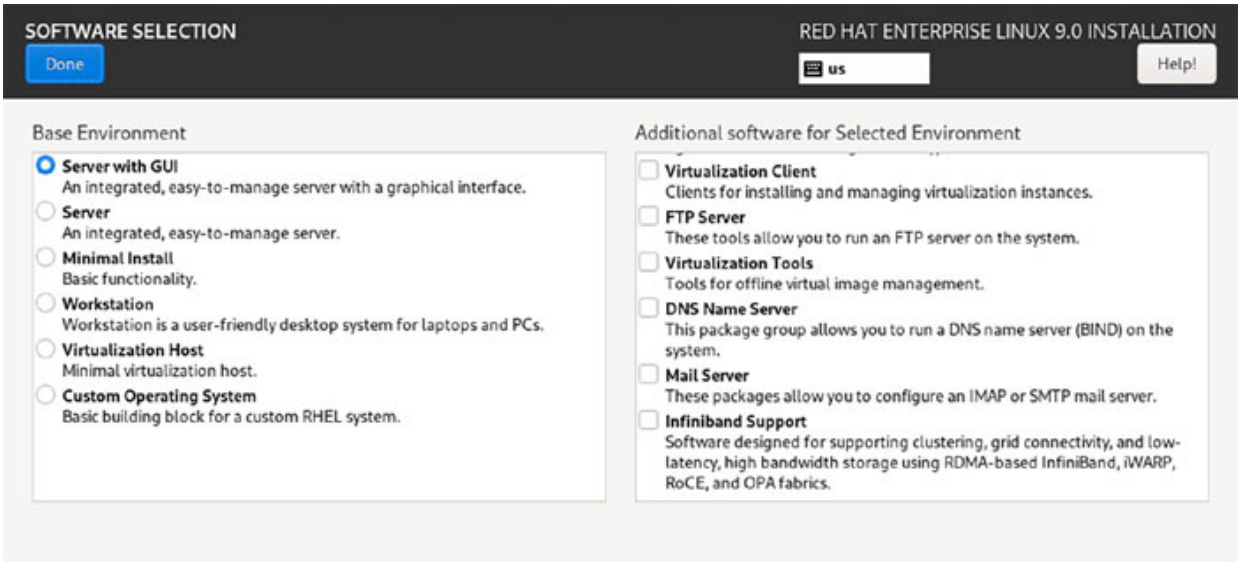


Figure 2.26: *Software selection*

Root password

Setting a root password is a required step during installation. The default option is to lock the root account to not be used, and the SSH is not connected to the system using the root.

User creation

If the root password is set, then user creation is not mandatory but highly recommended. After all the requirements are completed, the button “Begin Installation” is enabled, and the installation can be started.

Installation progress

The installer will download the required packages (if Boot ISO is used) or will use the packages in the Binary DVD for the system installation. The server is ready to be rebooted and be able to login with the root user or the user created.

CentOS and CentOS stream

Community Enterprise Operating System (CentOS) Linux is a popular production-ready distribution of Linux compatible with its upstream source, Red Hat Enterprise Linux. Between 2014 and 2020, CentOS stayed independent from RHEL, and versions of CentOS Linux 7 and 8 were released.

In December 2020, Red Hat decided to finish CentOS development and focus on the development of CentOS Stream, a midstream distribution between the upstream distribution Fedora and the downstream development for RHEL. Support of CentOS Linux 7 continued till mid-2024, whereas CentOS Linux 8 is not supported anymore.

CentOS Stream support is aligned to the RHEL support lifecycle, providing early access to the development stream of the next release of Red Hat Enterprise Linux. Installation steps and images are available as described before. Installation media can be downloaded from the website: **https://www.centos.org/centos-stream/**.

After Red Hat decision, two main distributions appeared to cover the CentOS space: Rocky Linux, from the original co-founder of CentOS, and AlmaLinux, from the company CloudLinux. Both descriptions will be covered next.

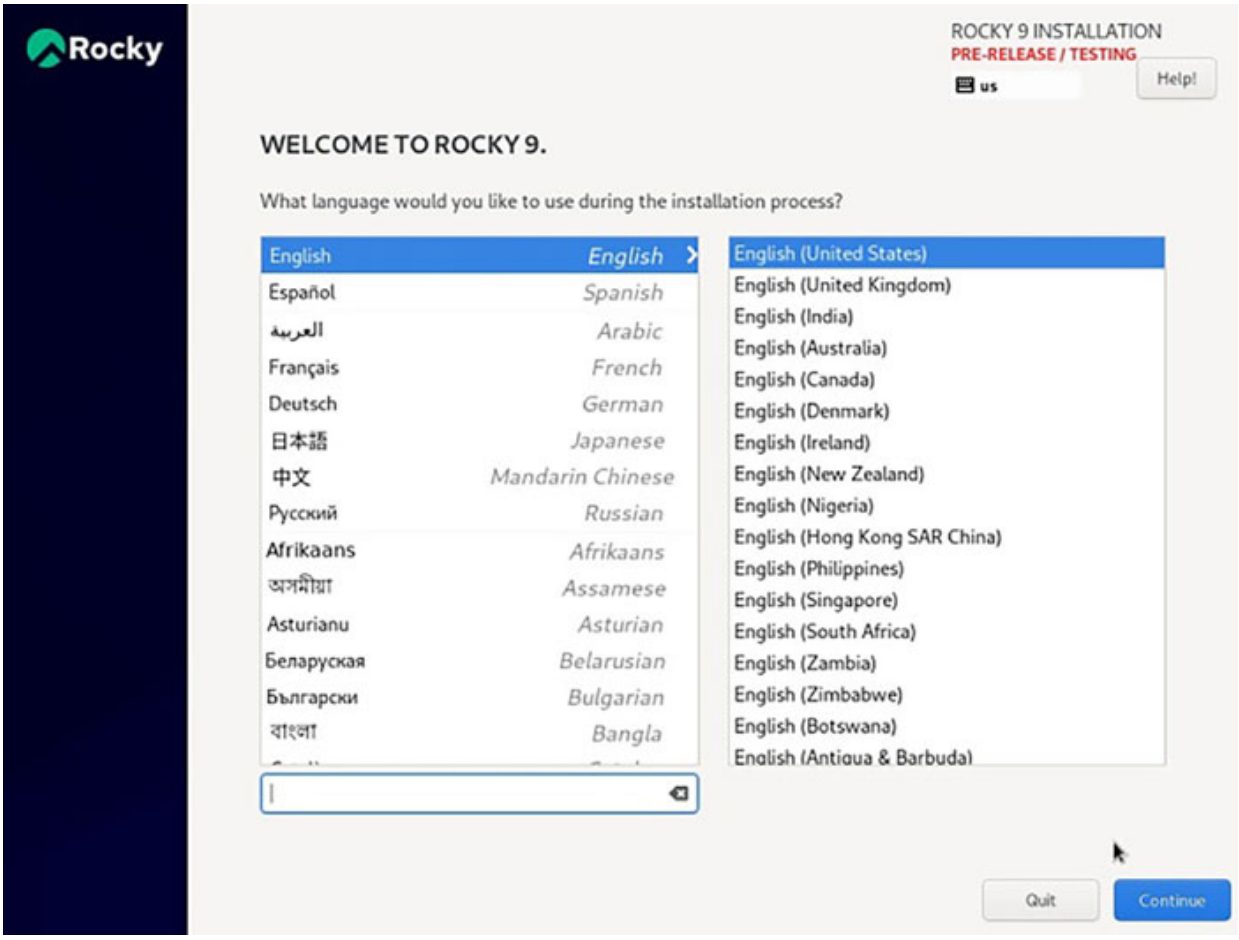
Rocky Linux and Alma Linux

Rocky Linux is an open-source enterprise operating system designed to be 100% compatible with RHEL, and it is developed by the community. Gregory Kurtzer, the original founder of CentOS, decided to create Rocky Linux after the Red Hat decision about CentOS. Official Rocky Linux includes the installation images on the website: **https://rockylinux.org/download**.

CloudLinux, the creator of CloudLinux OS, decided to create a separate, totally free distribution fully compatible with RHEL 8 and future versions. It is possible to download the ISO from the following website: **https://mirrors.almalinux.org/isos.html**.

Both distributions include a tool and commands to migrate from CentOS to Rocky Linux or AlmaLinux. Installation is with the same wizard as shown

for Red Hat Enterprise Linux. The following figures show how it looks the installer of AlmaLinux and Rocky Linux:



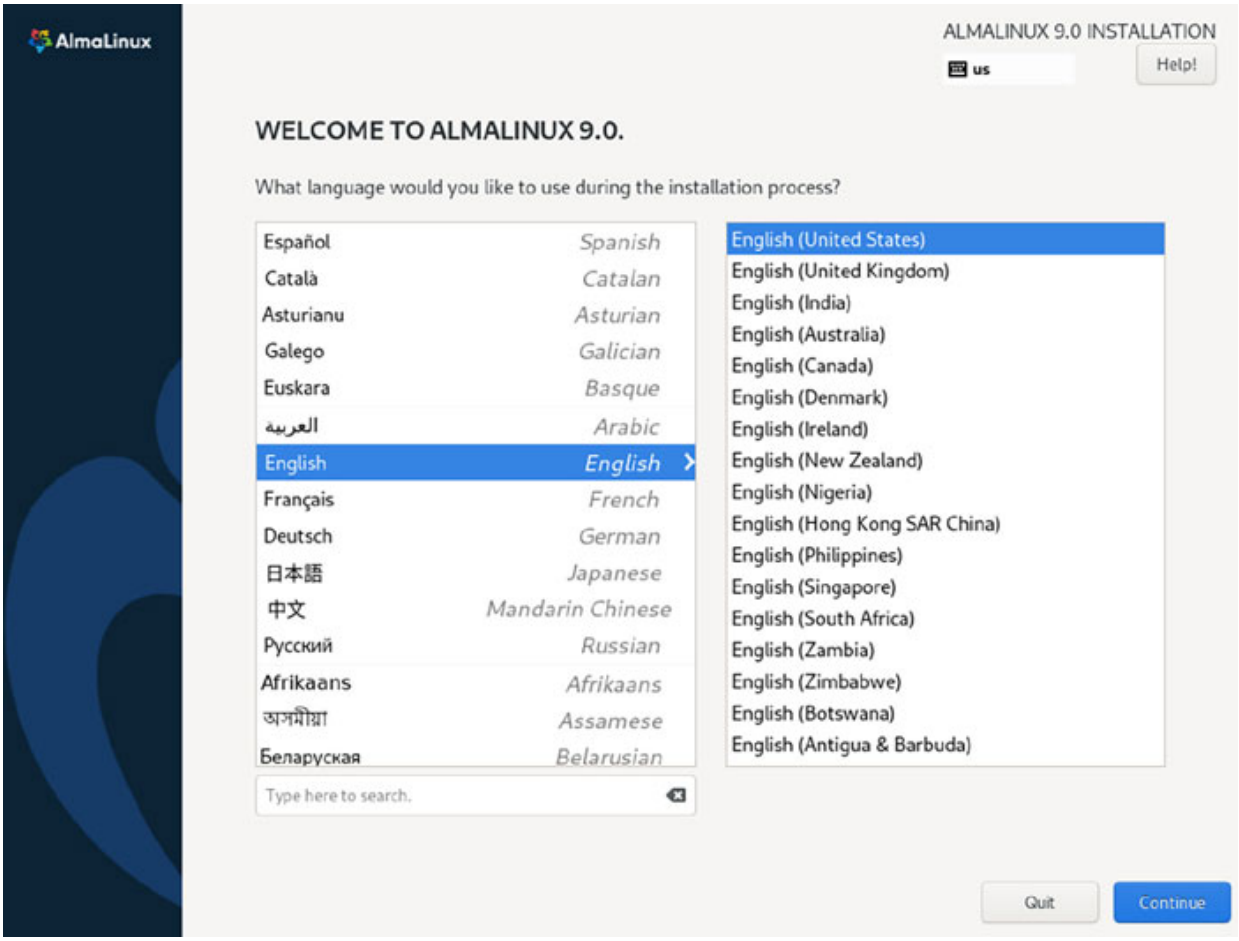


Figure 2.27/2.28: *AlmaLinux and RockyLinux installer wizard*

SUSE Linux enterprise server and openSUSE

SLES distribution is developed by SUSE, a German-based multinational open-source software company. It uses RPM as the package management system. Installation images can be found on the following website: **https://www.suse.com/download/sles/**

SUSE offers two non-commercial versions of SLES called Leap and Tumbleweed, managed by the project openSUSE. The second one, Tumbleweed, is a rolling release with more updated software. It can be downloaded from the website: **https://get.opensuse.org/**. The following figure shows the differences between the two versions:

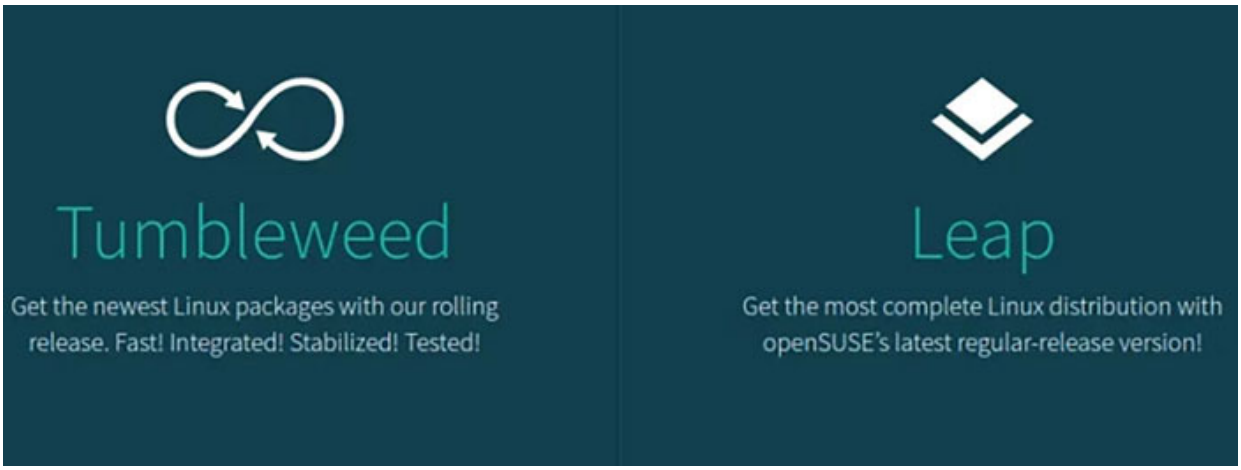


Figure 2.29: *OpenSUSE Tumbleweed and Leap differences. Source: SUSE*

Other popular distributions

Other popular distributions for servers are the following:

- **Oracle Linux (OL):** A derivative version of Red Hat Enterprise Linux, commercially supported and maintained by Oracle Company.
- **Gentoo:** A popular distribution due to the software being compiled for the system to be used and is not based on pre-compiled binaries software as other distributions.
- **Arch Linux:** A rolling-release distribution without major releases, all the new versions of the software or kernel will be available, and the installer images are updated and released every two months.

Conclusion

Linux is one of the most customizable operating systems to be installed and provides a big number of distributions from the community and from companies with professional support.

Installation does not require expert knowledge allowing beginners to have a secure and efficient system. Expert users can use advanced features during the installation to have a production-ready server after the installation.

The decision of Linux distribution includes different considerations to be taken, from the years of support, the lifecycle release for new versions, or the package manager to be used.

Key facts

- Huge number of Linux distributions are available.
- Some Linux distributions have a commercial support.
- Linux installation is a simple process.
- Linux distributions uses a package manager for software management.

Questions

1. What popular Linux distributions use the package manager called RPM?
    a. Red Hat Enterprise Linux
    b. Alma Linux
    c. Rocky Linux
2. What is the package manager is used by Debian and Ubuntu?
    a. Pacman
    b. YaST
    c. DPKG
3. Is it possible to install a Linux distribution without a DHCP server?
    a. True
    b. False
4. Is it possible to install Linux without an Internet connection?
    a. True
    b. False

Answers

1. a, b and c
2. c
3. a
4. a

CHAPTER 3

Using the Command Line Interface

Introduction

To administrate a Linux server either locally or remotely, it is fundamental to have knowledge about the **Command Line Interface** (**CLI**). This text-based interface is used to perform different tasks in a Linux distribution, from simple tasks, such as to create a directory, to advanced ones, such as configuring network and storage. A user is connecting to a Linux console, indicating the command to perform the task desired with the options and arguments required for the action.

In this chapter, the different terminals available to run commands will be described, as well as the possible configurations and commands to identify the system components and system information. This chapter will also explore how the commands will show the output to the display, how to redirect that output, and also the method to use it as input for another application.

Structure

In this chapter, we will discuss the following topics:

- Linux console and prompt
- Use of basic first CLI commands
- CLI commands to identify resources
- CLI commands to list elements
- Explanation of standard streams and pipes
- CLI commands for data stream

Linux console and the prompt

The Linux command line is known as well as the *terminal*, *console*, or *shell*. There are ways to access the command line as follows:

- Using the internal Linux console provided by the Linux kernel, accessible locally in the system. The name for the software providing the login is called *getty* and modern distributions include a default alternative named *agetty*.
- Using a terminal provided by the graphical environment of the system, where the commands will be executed in the system where it is running. Some examples are *GNOME Terminal, Konsole*, or *Terminator*.
- A remote terminal connecting from another console, usually using the protocol SSH.

When the system is booted and not installed with a graphical interface, a Linux terminal will appear to introduce the username and the password. Traditionally, there were six available consoles to be used, with the keyboard combinations Ctrl+Alt+F1 through Ctrl+Alt+F6. This behavior is configurable, and some distributions use Ctrl+Alt+F2 through Ctrl+Alt+F7.

When the system is installed, and a graphical interface enabled, the server will show that interface as default. Using the combination Ctrl+Alt+F1 through Ctrl+Alt+F6, as described previously, will show the native Linux console, whereas Ctrl+Alt+F7 will return to the graphical interface. Some distributions use Ctrl+Alt+F1 for the same purpose.

On Linux, there are the following two main software terminals:

- **TeleTYpewriter** (**tty**)**:** enables direct interaction with the operating system, handling the input from the keyboard and the output using the screen.
- **Pseudo-tty** (**pty**): A pseudoterminal behaves like a regular *TTY* using two pseudo-device endpoints. Using the graphical interface or connecting to a remote system, this pseudoterminal is used.

When the login is completed, or a terminal from a graphical environment is used, an input field to write commands appears, indicating the system is ready to accept tasks. This input is named known as *Linux prompt* and is showing every time the username, the name of the system, and the current directory. The following illustration shows the *Linux prompt* format:

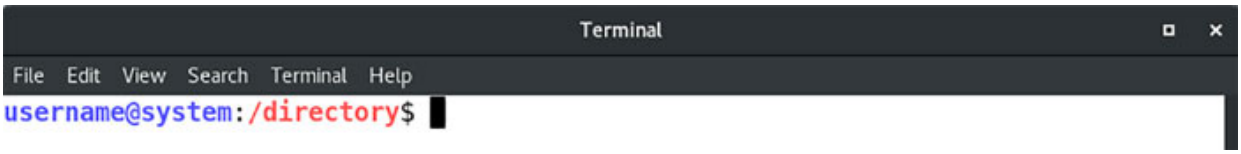


Figure 3.1: *Prompt format example*

In this example, before typing the command to perform the shell, ensure that the following information is provided:

- The user running the command is indicated before the @.
- The command will be executed in the system indicated after the @.
- The current directory is indicated after the username and the system.
- The command will be executed by a user non-administrator ($). For administrator users, the symbol sharp (#) will replace the dollar ($).

In the following figure (*figure 3.2*), a user with username *agonzalez* is working in the system *ubuntu* (this system can be local or remote) and currently is the directory */tmp*. This user is a regular user, indicated by the dollar before the input command. Please refer to the following figure:

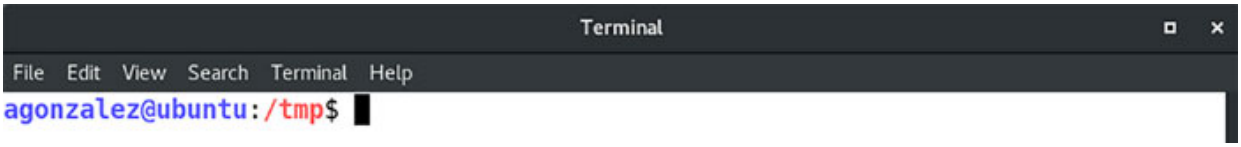


Figure 3.2: *Prompt example for a regular user*

The administrator user in Linux is commonly known as *root*. In the following example, it is possible to observe in the same system and directory how the prompt differs from a regular user. The following figure shows an example of the Linux prompt for *root* user:

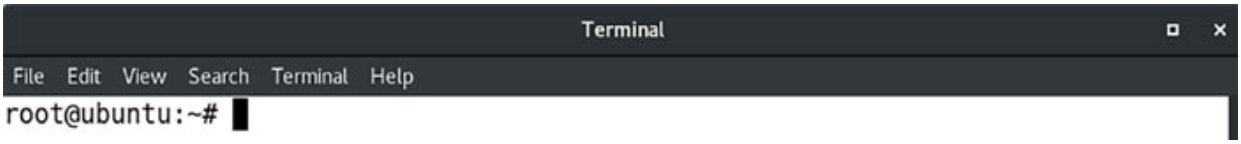


Figure 3.3: *Prompt example for the root user*

The interpreter who reads the commands from the user is called *shell*; the most popular and default in many distributions is called *bash*. Running a command in a Linux system has five important elements to have into consideration:

- **The command to be executed:** This can be a system command (for example, *mkdir* to create a directory), an internal function from the Linux terminal (for example, *alias* to create command shortcut) or can be a user application (such as *firefox*). The command can be specified in three different options:
    - Using the name of the command without indicating where it is located. The system has configured some directories where to find the command specified, and introducing the command will go in order through the commands to check if the command is available.
    - Using the full path where the command is situated, no matter the directory where currently the user is located, for example: */usr/bin/mkdir*.
    - Using a relative path from the directory where the user is located, for example: *.../bin/firefox*. In this example, the command *firefox* is located in a subdirectory called *bin* in the top-level directory where the user is located:
        - If the user is located in */home/agonzalez/Documents/* running the command previously indicated, it will execute /*home/agonzalez/bin/firefox*. The two dots *".."* indicates the top-level directory.
        - The special character "~" indicates the home directory for the user. If the home directory is, for example, */home/agonzalez/,* then indicating *~/bin/firefox* will be expanded to */home/agonzalez/bin/firefox.*
- **The options for the command**: This can be one or more options if it is needed to change the command behavior. Some options can be required. The command *ls* to list the content of the current directory will show the name of files and directories, using the option *-l* (*ls -l*) shows more information such as permissions, size, and owner of the file.
    - Some options can be indicated in short format (*-a*) or in long format (*--all*), taking into consideration that the long format uses two hyphens instead of one. With this option, the command *ls* shows hidden directories (directories starting with a dot).

    - One important option for commands is usually *-h* or *--help* to obtain information about the syntax of the command, including possible options and arguments.

- **The arguments for the command:** Some commands require arguments to perform a task, whereas for other commands, the arguments are optional or they do not accept arguments. The command *mkdir* requires an argument indicating which directory needs to be created; the command *ls* lists the content of the current directory, but by specifying as an argument, a directory will show its content of it, and the command *uptime* will show how long the system is running but does not accept any argument.

Order for options and arguments in some commands are interchangeable, but the recommendation will specify first the options and then the arguments. Some examples of how to run commands using options and arguments are shown in *figure 3.4*:

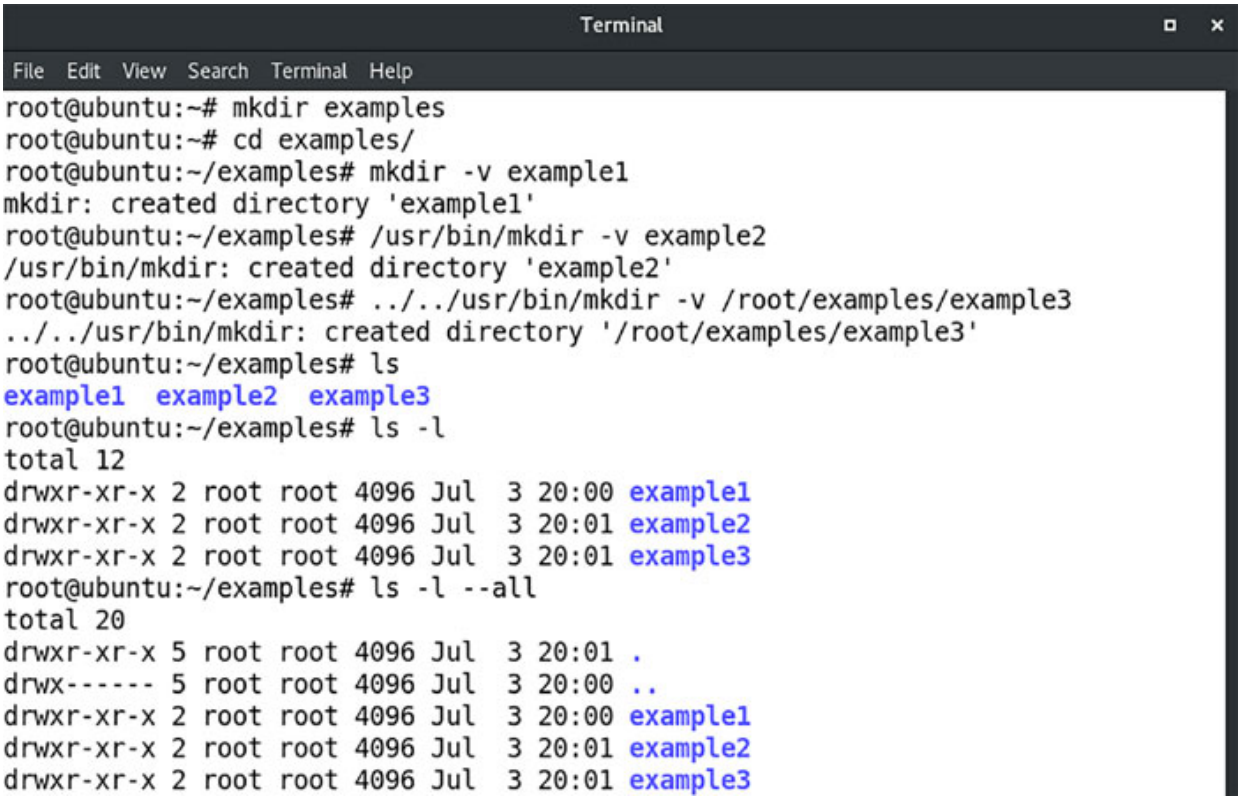


Figure 3.4: *Command execution examples*

Use of basic first CLI commands

As observed previously, the Linux console and the prompt are showing every moment—in which directory and the system the user is working on. Linux distributions offer some commands to provide more information, such as the full directory where the user is located or obtain the current time and date for the system.

Command *pwd*

The command *pwd* (*p*rint *w*orking *d*irectory) shows the absolute current directory where the user is located. The prompt usually shows the relative directory. This command, even having some options, is generally used without any option or argument. Working with several directories with the same name can be problematic and cause unexpected errors. The command *pwd* helps to ensure that the task will be executed in the desired directory. In the following example, the directory "*examples*" exists inside the home directory for the user *root* and the user *examples*. The output of the command *pwd* will provide the full path of the directory to avoid confusion. The following figure shows the use of this command:

```
root@ubuntu:~# cd examples/
root@ubuntu:~/examples# pwd
/root/examples
root@ubuntu:~/examples#
logout
agonzalez@ubuntu:~/examples$ pwd
/home/agonzalez/examples
```

Figure 3.5: *Command pwd output example*

Command *whoami*

The command *whoami* provides a simple output: the name of the current user. This command does not accept options or arguments. This command is usually used inside scripts, where a typical use case is to ensure that an

administrative user (*root*) is not running a script for security reasons. The following figure is an example of the output for this command:

Figure 3.6: *Command whoami output example*

Command *hostname*

The command *hostname* gives information about the name and domain of the system where the commands will be executed. The command *hostname* accepts some of the following options, and *figure 3.7* shows some example outputs for the different options:

Short option	Long option	Information
-d	--domain	Display the name of the DNS domain.
-f	--fqdn --long	Display the Fully Qualified Domain Name (FQDN).
-i	--ip-address	Display the network address of the host name.
-I	--all-ip-addresses	Display all the network addresses of the host.
-s	--short	Display the short host name without the domain.

Table 3.1: *Common options for the command hostname*

```
root@ubuntu:~# hostname
ubuntu.example.com
root@ubuntu:~# hostname --short
ubuntu
root@ubuntu:~# hostname --long
ubuntu.example.com
root@ubuntu:~# hostname -i
192.168.122.101
root@ubuntu:~# hostname -I
192.168.122.101 10.0.0.240
```

Figure 3.7: *Command hostname output examples*

Command *man*

This command is one of the most useful commands when a user wants to obtain information about one topic, such as a command or a system file. The simplest call of *man* is indicating one argument to get the documentation desired; for example: *man mkdir* will show the documentation for the command *mkdir*. The following figure illustrates this:

```
MKDIR(1)                          User Commands                          MKDIR(1)

NAME
       mkdir - make directories

SYNOPSIS
       mkdir [OPTION]... DIRECTORY...

DESCRIPTION
       Create the DIRECTORY(ies), if they do not already exist.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -m, --mode=MODE
              set file mode (as in chmod), not a=rwx - umask

       -p, --parents
              no error if existing, make parent directories as needed

       -v, --verbose
              print a message for each created directory

       -Z     set SELinux security context of each created  directory  to  the
              default type

 Manual page mkdir(1) line 1 (press h for help or q to quit)
```

Figure 3.8: *Example running command man mkdir*

The manual pages are organized into nine numbered sections:

Section number	Section description
1	Executable programs or shell commands
2	System calls (functions provided by the kernel)
3	Library calls (functions within program libraries)
4	Special files (usually found in /dev)
5	File formats and conventions, e.g. /etc/passwd
6	Games
7	Miscellaneous (including macro packages and conventions)
8	System administration commands (usually only for root)
9	Kernel routines [Non standard]

Table 3.2: *Manual pages section numbers*

topic name can be part of different sections; for example, *mkdir* can be an executed program as previously described, and it will, by default, show the information in the first section number. However, *mkdir* is also a system call, having information in the second section. To indicate the section number, the options are as follows:

- *man 2 mkdir*
- *man mkdir.2*
- *man "mkdir(2)"*

Command *cd*

The command *cd* (*c*hange *d*irectory) is one of the mostly used tools in the terminal and is used to navigate between directories. This tool is not a binary application; rather, it is a *built-in* functionality from the *shell* used. There common uses of the command *cd* are as follows:

- **Without arguments:** the directory will be changed to the home directory of the user.
- **With an absolute path:** the directory will be changed to the full path indicated.
- **With an relative path:** the relative path will be extended to an absolute path.
- **With the hyphen argument (-):** it will move to the previous directory the user was located.

In the following figure, the common uses are shown as follows:

```
root@ubuntu:/tmp# pwd
/tmp
root@ubuntu:/tmp# cd
root@ubuntu:~# pwd
/root
root@ubuntu:~# cd examples/
root@ubuntu:~/examples# cd /tmp/
root@ubuntu:/tmp# cd -
/root/examples
```

Figure 3.9: *Example moving between directories*

Command *history*

Working in the *shell* leads to write multiple numbers of commands to perform different tasks during the life of the system. As an administrator or a regular user, it is interesting to review the commands executed or repeated one command after the other. The tool *history* lists the commands executed previously by the user invoking it. By default, it saves the previous 1000 commands that were executed. The popular options are *-c* to clear the history list and *-i* to write the changes to the file (by default named *.bash_history*). The following screenshot shows some of the commands executed in this chapter:

```
root@ubuntu:~/examples# history
    1  mkdir examples
    2  cd examples/
    3  mkdir -v example1
    4  /usr/bin/mkdir -v example2
    5  ../../usr/bin/mkdir -v /root/examples/example3
    6  ls
    7  ls -l
    8  ls -l --all
    9  pwd
```

Figure 3.10: *Output example for command history*

It is possible to repeat the last command writing to exclamation (*!!*) or repeat a specific command using exclamation and the number from the history output (for example !9 to run the command *pwd* from the output of *figure 3.10*)

Command *uptime*

This simple tool shows valuable information about the current time, how long the system has been running, how many users are currently logged on, and the system load average for the last minute, 5 minutes, and 15 minutes. This command does not accept any argument, and the useful option is *-p(-pretty)* to show only the uptime information. The following figure shows the output example:

```
root@ubuntu:~# uptime
 20:16:03 up  4:36,  5 users,  load average: 0.00, 0.00, 0.00
root@ubuntu:~# uptime -p
up 4 hours, 36 minutes
```

Figure 3.11: *Output example for command uptime*

CLI commands to identify resources

Linux distributions include some system commands to identify system resources. These utilities are usually available after the installation and do not require any package installation to run the following commands. These

commands provide information about the CPU, RAM, and devices available in the system.

Command *lscpu*

This command shows information about the number of CPUS, threads, cores, sockets, and advanced information present in the system. This command does not require any argument, and the options are only needed for advanced information (for example, the option *--json (-J)* will show the output in *JSON* formation). The following figure shows the output example for a physical server:

```
root@ubuntu:~# lscpu
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
CPU(s):                  8
On-line CPU(s) list:     0-7
Thread(s) per core:      2
Core(s) per socket:      4
Socket(s):               1
NUMA node(s):            1
Vendor ID:               GenuineIntel
BIOS Vendor ID:          Intel(R) Corporation
CPU family:              6
Model:                   142
Model name:              Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz
BIOS Model name:         Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz
Stepping:                10
CPU MHz:                 2100.000
CPU max MHz:             4200,0000
CPU min MHz:             400,0000
BogoMIPS:                4224.00
Virtualization:          VT-x
L1d cache:               32K
L1i cache:               32K
L2 cache:                256K
L3 cache:                8192K
NUMA node0 CPU(s):       0-7
```

Figure 3.12: *Example output of command lscpu*

The previous image shows an example of the output of the commands *lscpu*, where the CPU architecture, number of CPUs, thread per core, cores per socket, and socket number are shown. Other useful information, such as model, model name, and the speed of the CPU, is detailed.

Command *lshw*

This command lists hardware present in the system, with the possibility to filter the category to be shown. By default, command *lshw* is showing all the hardware available in the system, such as memory configuration, firmware versions, mainboard configuration, CPU, and PCI devices.

This tool is showing in text format the output, but using the options, it is possible to show the outfit in different formats: *HTML* (*-html*), *XML* (*-xml*), and *JSON* (*-json*). It is possible to filter the category (class) of the devices using the option *-class* (*-c);* some popular classes are network, disk, processor, and system. The following figure shows the output example of a personal computer:

```
root@ubuntu:~# lshw -c network -short
H/W path                 Device          Class          Description
=====================================================================
/0/100/1c.6/0            wlp4s0          network        Wireless 8265 / 8275
/0/100/1f.6              enp0s31f6       network        Ethernet Connection (4)
/b                       enp14s0u1       network        Ethernet interface
```

Figure 3.13: *Example listing network devices in short mode using command lshw*

Command *free*

Command *free* displays the amount of free and used memory in the system. This command doesn't accept any argument and the popular option is to display the memory in different units (for example: *--giga* for gigabytes and *--tera* for terabytes) or in human-readable format (*-h/--human)* instead of in megabytes, which is generally the default format. The output will show information as well related to the swap (*virtual memory*) available. The displayed columns are as follows:

Column	Column description
total	Total installed memory
used	Used memory (used = total -- free -- buffers - cache)

free	Unused memory
shared	Memory used by *tmpfs* (a virtual memory filesystem)
buffers	Memory used by kernel buffers
cache	Memory used by the page cache
buff/cache	Sum of buffers and cache
available	An estimation about how much memory is available for new applications.

Table 3.3: *Columns for the command free*

Memory in operating systems (not only in Linux) is a complicated topic requiring advanced knowledge about how memory is reserved and shared between different processes. The columns *free* and *available* offer a good overview if the memory in the system is exhausted. The following figure shows the example output for a physical server:

```
root@ubuntu:~# free -h
              total        used        free      shared  buff/cache   available
Mem:           31Gi        11Gi       5,8Gi       2,7Gi        13Gi        16Gi
Swap:          31Gi          0B        31Gi
```

Figure 3.14: *Output example for the free command with a human-readable format*

Command *df*

Command *df* (*d*isk *f*ree) reports file system disk space usage. This command, without arguments, will show the information for all filesystems and mount points, and indicating an argument will limit the output to that filesystem or mount point. Popular options to filter the result are as follows:

Option	Option description
-h / --human-readable	Shows information in *human readable* format instead in kilobytes.
-i / --inodes	Shows the number of files in the partition and the available quantity.
-l / --local	Excludes remote partitions mounted.
-t / --type=TYPE	Filters the list of filesystems to match with the TYPE specified.
-T / --print-type	Shows the file system type.
-x / --exclude-type=TYPE	Filters the list of filesystems to exclude the TYPE specified.

Table 3.4: *Common options for the command df*

The following figure shows the example output showing the human-readable format and the filesystem type information. The following figure is the example output for a Virtual Machine:

```
root@ubuntu:~# df -hT
Filesystem                         Type   Size  Used Avail Use% Mounted on
tmpfs                              tmpfs  393M  1.2M  392M   1% /run
/dev/mapper/ubuntu--vg-ubuntu--lv ext4    12G  5.0G  5.7G  47% /
tmpfs                              tmpfs  2.0G     0  2.0G   0% /dev/shm
tmpfs                              tmpfs  5.0M     0  5.0M   0% /run/lock
/dev/vda2                          ext4   2.0G  126M  1.7G   7% /boot
tmpfs                              tmpfs  393M  4.0K  393M   1% /run/user/1000
```

Figure 3.15: *Output example for command df*

Commands *lspci*, *lsusb*, and *lsblk*

These three popular commands are used to display the following information:

- **`lspci`**: is a utility for displaying information about the PCI buses in the system and devices connected to them. It is possible to obtain more information using the option *-v*, *-vv,* or *-vvv (more* v *indicates more verbosity).*
- **`lsusb`**: is a utility for displaying information about USB buses in the system and devices connected to them. This command accepts the option *-v* to show more advanced information.
- **`lsblk`**: is a utility to list information about all available or specified block devices, showing partitions for devices, label, and ids for the partitions. It is possible to specify arguments to filter the output to the indicated device (for example, */dev/sda)*.

CLI commands to list elements

There are two popular commands to list and find elements such as files, directories, links, or block devices in the system: *ls* and *find*.

Command *ls*

The command *ls* lists information about files and directories, listing the current directory if no argument is specified. This command has many

options; some of the most useful are described in the following table:

Option	Option description
-a / --all	Shows hidden files and directories (starting with dot)
-c / --color=WHEN	Indicates when colorize the output, can be *always*, *auto* or *never*.
-d / --directory	List the directory itself and not the content.
-h/ --human-readable	Shows the sizes in *human readable* with the more readable unit.
-l	Use a long listing format including owner, size and permissions.
-r / --reverse	Reverse order while sorting
-R / --recursive	List subdirectories recursively.
-S	Sort by file size, largest files will be shown first.
-t	Sort by time, newest files/directories will be shown first.
-1	List one file per line instead in columns (when option *-l* is not in use)

Table 3.5: *Common options for the command ls*

Figure 3.16 shows the command *ls* sorting by size (*-S)* and in reverse mode (*-r*), using the long format (*-l*) and the human-readable (*-h*).

```
root@ubuntu:~/examples# ls -lhSr
total 12K
drwxr-xr-x 2 root root 4.0K Jul  3 20:01 example3
drwxr-xr-x 2 root root 4.0K Jul  3 20:27 example2
drwxr-xr-x 2 root root 4.0K Jul  3 20:00 example1
-rw-r--r-- 1 root root 1.0M Jul  3 20:27 file1M.bin
-rw-r--r-- 1 root root 2.0M Jul  3 20:58 file2M.bin
-rw-r--r-- 1 root root  10M Jul  3 20:29 file10M.bin
```

Figure 3.16: *Output example for command ls*

Command *find*

The command *find* searches for files, directories, or other objects in the system. This command, without options and arguments, will list all the elements recursively under the current directory. It forces to specify first the

path and then the expressions to filter. There are multiple options and combinations, and the following table shows some of the most used:

Option	Option description
-amin n	File was accessed less than, more than or exactly *n* minutes ago.
-atime n	File was last accessed less than, more than or exactly *n**24 hours ago.
-cmin n	File was changed less than, more than or exactly *n* minutes ago.
-ctime n	File was last changed less than, more than or exactly *n**24 hours ago.
-mmin n	File was modified less than, more than or exactly *n* minutes ago.
-mtime n	File was last modified less than, more than or exactly *n**24 hours ago.
-name pattern	Filter the name of the file or directory with the pattern
-perm mode	Filter by the permission, useful to find vulnerable files
-size n[kMG]	File uses less than, more than or exactly *n* units.
-type c	Filter the type of the file, some of the options are: • *b*: block special • *c:* character special • *d*: directory • *f*: file • *l*: link
-user username/userid	File is owned by the user or user id specified.
-xdev	Don't descend directories on other filesystems.

Table 3.6: *Common options for the command find*

The numeric *n* can be used with *+n* for greater than *n*, *-n* for less than *n,* and using *n* only is exactly that value. It is possible to perform different actions with the results using the options *-delete*, *-exec* commands as an example or formatting the output with options like *-ls* or *-print0*. The following figure shows how to filter files bigger than 1 megabyte:

```
root@ubuntu:~# find examples/ -size +1M -ls
   406107      0 -rw-r--r--   1 root     root      2097152 Jul  3 20:28 examples/file2M.bin
   406122      0 -rw-r--r--   1 root     root     10485760 Jul  3 20:29 examples/file10M.bin
```

Figure 3.17: *Output example for command find*

Explanation of standard streams

In a Linux system, when a command is executed, there are three streams associated: one for the data input, one for the data output, and another one for diagnostic or error output. These streams are associated with one name and one integer associated, as shown in the following table:

Integer	Name	Stream description
0	*stdin*	Standard input stream, the application receives data using this stream.
1	*stdout*	Standard output stream, the data will show the information using it.
2	*stderr*	Standard error stream, diagnostic, or errors will show using it.

Table 3.7: *Streams information and integer associated*

When a command requires the user to enter data, the application will use the *stdin* to read what the user will type to introduce the data requested. When the application returns information to the user, it will send it to the standard output (*stdout*), and if there is any error or diagnostic information to be shown, it will use the standard error (*stderr*). The following illustration shows the standard streams information:

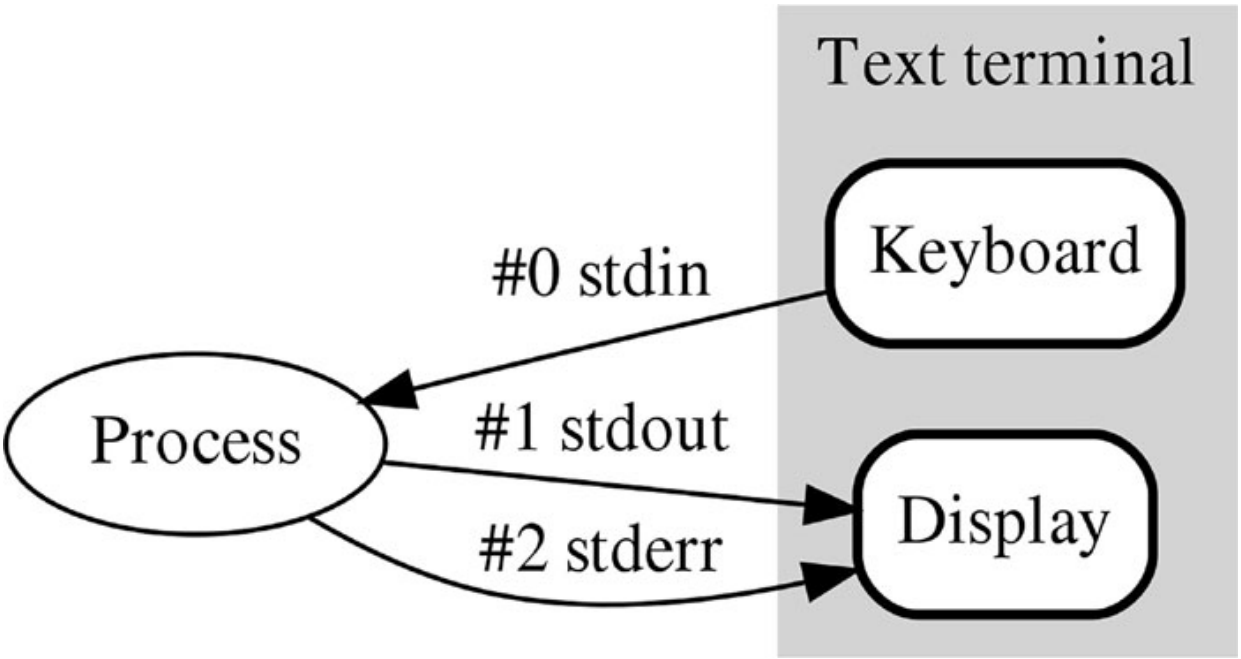


Figure 3.18: *Standard streams diagram example. Source: Wikimedia*

The default behavior, keyboard for input and display for outputs, can be modified to use a different input source or redirect the output. To modify them, the following table describes the different options:

Stream	Modifier	Command examples
stdin (0)	<	*read HOSTNAME </etc/hostname*
sdout(1)	*1>* *>*	*pwd 1>hostname* *pwd >hostname*
sderr(2)	*2>*	*ls noexists* 2>error.log *ls noexists* 2>&1
stdout(1) and stderr(2)	*&>*	*ls /etc/services noexists*&>output.log

Table 3.8: *Modify streams options.*

The command (shell function) *read* asks to the user to introduce a value and is saved in a variable indicated as an argument; in the example, a variable named **HOSTNAME**, changing the default behavior using the symbol less than (<) does not ask the user and read the data from a file. The following figure shows how to change the default behavior for the standard input:

```
root@ubuntu:~# read HOSTNAME
user input here
root@ubuntu:~# echo $HOSTNAME
user input here
root@ubuntu:~# read HOSTNAME </etc/hostname
root@ubuntu:~# echo $HOSTNAME
ubuntu.example.com
```

Figure 3.19: *The command read asks the user for input, but it is possible to read the input from a file instead*

As shown before, the command *pwd* is displaying on the screen the current directory where the user is located. Using a symbol greater than (*> or 1>*) will redirect the output to a file, creating the file or overwriting the content. In the following example, the *cat* command is used to see the content of the file. Using two greater than symbols (>>), it will append the content to the existing file or will create the file if it does not exist. The following figure shows how to redirect the standard output to a file:

```
root@ubuntu:~/examples# pwd
/root/examples
root@ubuntu:~/examples# pwd >pwd1.txt
root@ubuntu:~/examples# cat pwd1.txt
/root/examples
root@ubuntu:~/examples# pwd 1>pwd2.txt
root@ubuntu:~/examples# cat pwd2.txt
/root/examples
```

Figure 3.20: *The output of one command can be redirected to one file using > or 1>*

When a file shows an error on the screen, it means that a different output stream is being used rather than the normal output information. In the following example, the default behavior is shown and also how to redirect the output for one stream (using *2*>), for both (using 2>and >), or how to combine them (using *&>)*. The following figure shows how to redirect the standard error stream to a file or to the standard output stream.

```
root@ubuntu:~/examples# ls pwd1.txt pwd3.txt
ls: cannot access 'pwd3.txt': No such file or directory
pwd1.txt
root@ubuntu:~/examples# ls pwd1.txt pwd3.txt 2>error1.txt
pwd1.txt
root@ubuntu:~/examples# cat error1.txt
ls: cannot access 'pwd3.txt': No such file or directory
root@ubuntu:~/examples# ls pwd1.txt pwd3.txt 2>error2.txt >good.txt
root@ubuntu:~/examples# cat error2.txt
ls: cannot access 'pwd3.txt': No such file or directory
root@ubuntu:~/examples# ls pwd1.txt pwd3.txt &>everything.txt
root@ubuntu:~/examples# cat everything.txt
ls: cannot access 'pwd3.txt': No such file or directory
pwd1.txt
```

Figure 3.21: *The standard error can be redirected to one file or can be combined with the standard output*

An output of one command can be the input for another command, and this is called a *pipeline*. This means the *standard output* from one command can

be redirected to be the *standard input* of another command and this command to the next one, and so on. The two commands are executed in parallel; the second command will wait till the first command finishes sending the standard output. The character used for *pipelines* is the vertical bar (known as a *pipe* in Linux): *command1* | *command2* or more than two, *command1* | *command2* | *command3* | *commandN.* The following illustration shows how the standard input and output streams communicate between processes using the pipeline. Please refer to the following figure:

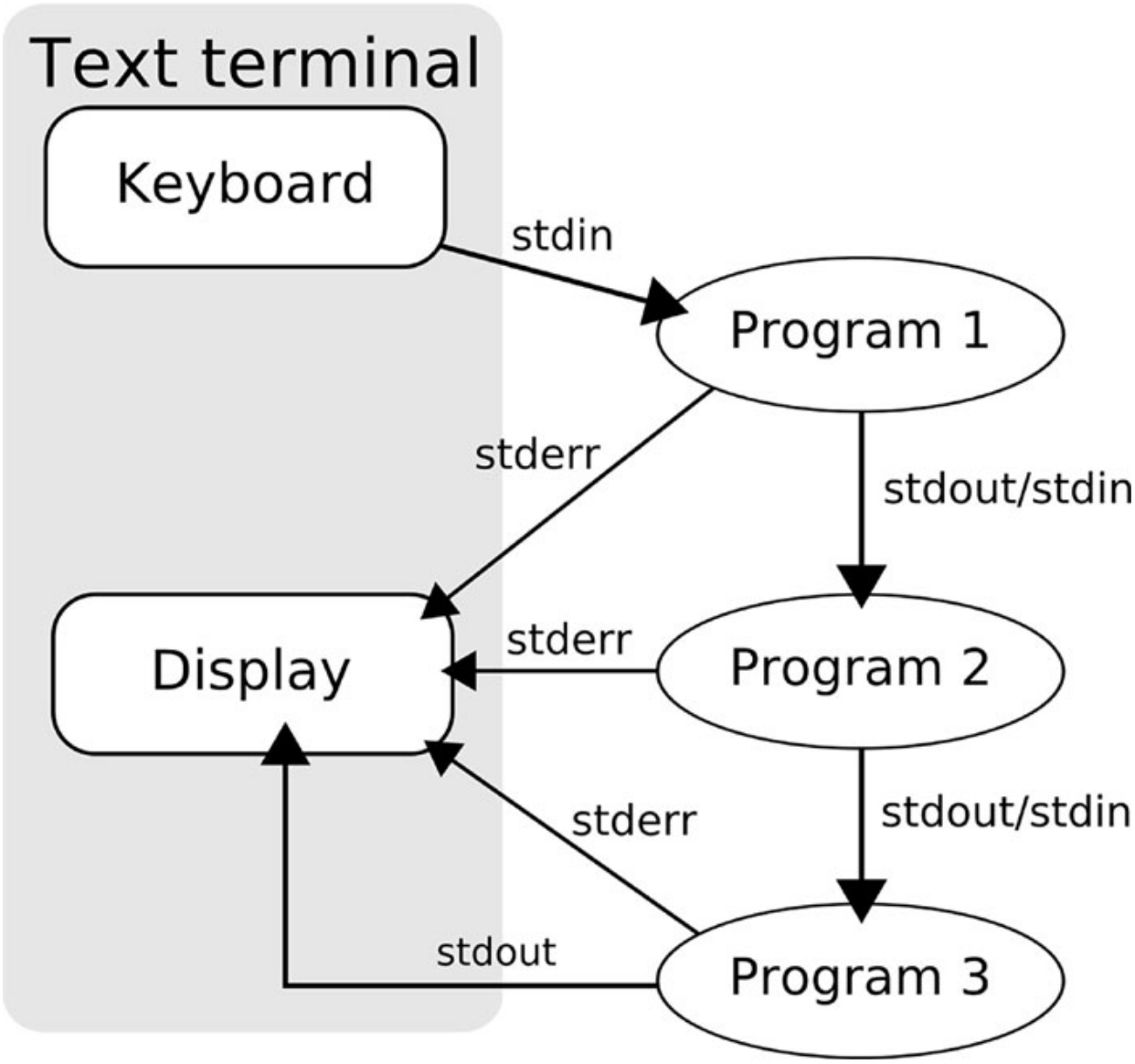


Figure 3.22: *Pipeline diagram example. Source: Wikimedia*

In the following figure, the output of one command (*cat*) will be used as input for another command (*head*), and another command (*nl*) will receive in

the *standard input* the data from *standard output* of the second command. This combination will show the first 10 lines of the file (*head* command) and will number it (*nl command*) from the output of the file */etc/services* (*cat* command*)*.

```
root@ubuntu:~# cat /etc/services | head | nl
     1  # Network services, Internet style
     2  #
     3  # Updated from https://www.iana.org/assignments/service-names-port-numbe
rs/service-names-port-numbers.xhtml .
     4  #
     5  # New ports will be added on request if they have been officially assign
ed
     6  # by IANA and used in the real-world or are needed by a debian package.
     7  # If you need a huge list of used numbers please install the nmap packag
e.

     8  tcpmux          1/tcp                           # TCP port service multi
plexer
     9  echo            7/tcp
```

Figure 3.23: *Pipeline example using three commands*

CLI commands for data stream

There are three simple important standard commands to operate with the standard streams: *echo*, *read*, and *tee*.

Command *echo*

This simple command displays a line of text. It is useful to show the value of variables, create files with content, or append data to an existing file. The most common option is *-n* not to add a newline at the end of the line. In the following figure, the different uses of the command echo are shown:

```
agonzalez@ubuntu:~$ echo "Show some text and value of variable $HOME"
Show some text and value of variable /home/agonzalez
agonzalez@ubuntu:~$ echo "Create a file" > example.txt
agonzalez@ubuntu:~$ echo "Append content to file" >> example.txt
agonzalez@ubuntu:~$ cat example.txt
Create a file
Append content to file
```

Figure 3.24: *Command echo examples and show file content with command cat*

Command *read*

This command, as shown before, is requesting the user to introduce a value from input standard and is saved in a variable. Some common options are *-p* to add a prompt text and *-t* to specify a timeout where the user should provide the input. In the following screenshot, the use of the command read is shown:

```
agonzalez@ubuntu:~$ read -p "Introduce username: " USERNAME
Introduce username: testuser
agonzalez@ubuntu:~$ echo $USERNAME
testuser
```

Figure 3.25: *Command read allows to ask the user for input data*

Command *tee*

One of the limitations to redirect the standard output is the impossibility to specify two destinations, a display, and a file. This can be achieved using the command *tee*, where the standard output will be displayed, and a file with the output will be created. Using the option *-a* will append the content to an existing file. In the following figure, the usage of command tee is shown:

```
agonzalez@ubuntu:~$ lsusb | tee lsusb.txt
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 0627:0001 Adomax Technology Co., Ltd QEMU USB Tablet
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
agonzalez@ubuntu:~$ cat lsusb.txt
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 0627:0001 Adomax Technology Co., Ltd QEMU USB Tablet
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Figure 3.26: *Command tee is redirecting the output to display and to one file*

Conclusion

Working with a Linux console is crucial for system administration. It is important to understand how to run commands and specify the options and the arguments required and desired. Accessing the documentation with the *man* command or using the options *-h* or *–help* will be of great help to know the options and arguments available. Linux distributions are offering some commands to identify the resources available in the system.

The concept of standard streams and how to change the default behavior for the input and the output is crucial to pass data between commands as in the following chapters the importance of this will be shown during other chapters of this book.

Key facts

- Command line is the best way to administrate a Linux system.
- Linux brings many commands to identify resources.
- There are three standard streams: input, output and error.
- It is possible to redirect the outputs.
- An output of one command can be used as input for another command.

Questions

1. Which command is used to display the current directory?
    a. whoami
    b. path
    c. pwd
2. Which command is used to obtain information about the memory?
    a. df
    b. free
    c. du
3. Which command is used to obtain about the total disk usage?
    a. df
    b. free
    c. du
4. Which command is used to request user data and save in a variable?
    a. read
    b. prompt
    c. input

5. Which command is used to show in standard output and save it in a file?
    a. pipe
    b. tee
    c. export

Answers

1. c
2. b
3. a
4. a
5. b

CHAPTER 4

User Administration and Software Management

Introduction

After learning how to use the *Linux terminal* to perform different tasks, this chapter will discuss how to administrate users. The different commands used to create, modify, and delete users will also be explored. Other important topics described in this chapter will be how to operate with groups, how to assign it to users, and the difference between primary and secondary groups.

The second part of this chapter is focused on software management. As described in the ***Linux Installation*** chapter, the two popular package managers are *RPM and DEB. RPM is* used by Red Hat Enterprise Linux and derivatives distributions such as SUSE or CentOS, whereas *DEB is* used by Debian and derivatives distributions such as Ubuntu Server. This chapter will delve deep into these package managers, including the commands to operate with them.

The last part is related to services and an overview of how they are working on Linux, as well as how to operate with them, such as to enable/disable, start, stop, or restart them.

Structure

In this chapter, we will discuss the following topics:

- Introduction to users and groups
- Best practices for user accounts
- Commands to administrate users
- Commands to administrate groups
- Introduction to RPM and DEB package formats
- Commands to operate with RPM packages
- Commands to operate with DEB packages
- Introduction to services

Introduction to users and groups

During the installation of a Linux distribution, an administrative user named *root* is created. A new user is also recommended to be created using *sudo* for security reasons to login and to perform administrative tasks.

Linux-based cloud images for different distributions also include a default user. This user has no password set, and the only possibility to connect with it is using a private SSH key. This will be covered in the network and security chapter. The typical login for cloud images are **`cloud-user`** (for most of the RPM-based distributions), **`ec2-user`** (for Linux systems in *Amazon Web Services*), or specific for distribution (*centos* for CentOS system, or *ubuntu* for Ubuntu Server).

Users in Linux are not only reserved for human logins but there are also system users associated usually with services or applications. Linux users are identified by a numerical ID (*uid*): predefined users created during the installation will use a range between 0 and 99, whereas system users created using installing services and applications will use a range between 100 and 999. Regular users will have a bigger range, from 1,000 to 6,000. The *uid* 0 is reserved for the *root* user.

Linux also has a list of groups, and the users are a member of them. A user is always associated with a primary group and, optionally, with multiple secondary groups. These groups help to associate permissions to files and applications with several users. Same as with the users, there exist system groups created during the installation of the Linux distribution or when a new service or application is installed. Groups are also identified by a numerical ID (*gid*), with the same range as for the users: 0–99 for predefined groups created during installation, 100–999 for the system groups created during the installation of services and applications, and 1,000–6,000 for the regular ones. The root user has their own group called *root* with the *guid* 0.

Administrators are able to create, administrate, and also delete users and groups with available system tools. Linux stores the information for users, passwords, groups, and passwords for groups in files. The four main files for keeping the information are as follows:

- **/etc/passwd**: Contains the list of the users locally available in the system. This file contains separated by colons (":") the following fields:
    - **username**: The user login with a maximum length of 32 characters.
    - **password:** It will contain the letter "x" indicating that it is stored in /etc/shadow. It is also possible to store the password encrypted in this file, although it is highly not recommended.
    - **user id (*uid*):** A unique numerical id associated with the user.
    - **group id(*gid*):** A numerical id, as primary group referencing a group in the file */etc/group*.
    - **User information**: Nowadays, this field contains the full name of the user. Historically, these fields contained the name, location, phone number, and

office number, and they were separated by commas.

- **Home directory**: Full path where the user will have the main directory for his data.
- **Shell**: Full path to the shell to be used. System users will use `/sbin/nologin`, `/usr/sbin/nologin` or `/bin/false` to avoid them from logging in as regular users. Normal users usually use `/bin/bash`, `/bin/sh`, or other popular alternatives such as `/bin/zsh`.

- **/etc/shadow**: This file contains the password information and the expiration data for the local users in the system. This file should be protected and made unable to be read by non-administrative users. It contains nine fields separated by colons (":"):
  - **username**: A user existing in /etc/passwd
  - **encrypted password**: A password that is usually automatically filled by command *passwd*.
  - **date of last password change**: This field has three possible values:
    - Empty means the user has no expiring date.
    - A 0 value means the user would have to change the password after she logins.
    - An integer expressed as the number of days since January 1, 1970, 00:00 UTC, indicates when the password was changed for the last time.
  - **minimum password age:** The minimum number of days the user will have to wait before they are allowed to change the password again. An empty field and value 0 mean that there is a minimum password changing time required.
  - **maximum password age:** The maximum number of days after which the user will have to change their password. The user should be asked to change the password the next time they will log in. An empty field means that there is no maximum password age, no password warning period, and no password inactivity period.
  - **password warning period**: The number of days before a password is going to expire, during which the user should be warned. An empty field and value of 0 mean that there is no password warning period.
  - **password inactivity period:** The number of days after a password has expired during which the password should still be accepted.
  - **account expiration date**: The date of expiration of the account, expressed as the number of days since January 1, 1970, 00:00 UTC.

    - **reserved field:** This field is not used currently.

- **/etc/group:** This file defines the groups of the system. It contains the following fields separated by colon (":"):
    - **group name**: The name of the group
    - **password**: An "x" indicating the password is stored in file **`/etc/gshadow`** or empty if password is not needed.
    - **group id (*gid*):** A numerical id.
    - **user list**: a list of users separated by comma belonging to this group as one of the secondary groups.
- **/etc/gshadow**: This file contains the password for the groups, as well as the administrators for the group and the members. This file should be protected and only readable by administrators. The fields separated by colon (":") are as follows:
    - **group name**: The name of the group from **`/etc/shadow`**
    - **encrypted password**: A password usually automatically filled by the command **`gpasswd`**
    - **administrators**: A comma-separated list of administrator users. These users can change the password and the members of the group.
    - **members**: A comma-separated list of member users. It should match the list from **`/etc/group`.**

The file **`/etc/login.defs`** defines the default values for different parameters related to users and groups, such as user id ranges, group ranges, password expiration defaults, and default home directory permissions. An example of the file content is shown in *table 4.1*:

Option	Value
MAIL_DIR	/var/spool/mail
UMASK	022
HOME_MODE	0700
PASS_MAX_DAYS	99999
PASS_MIN_DAYS	0
PASS_MIN_LEN	5
PASS_WARN_AGE	7
UID_MIN	1000
UID_MAX	60000
SYS_UID_MIN	201
SYS_UID_MAX	999

GID_MIN	1000
GID_MAX	60000
SYS_GID_MIN	201
SYS_GID_MAX	999
CREATE_HOME	yes
USERGROUPS_ENAB	yes
ENCRYPT_METHOD	SHA512

Table 4.1: *Options in /etc/login.defs*

During the installation of a Linux distribution, a **Linux Standard Base** (**LSB**) is configured. This base contains a predefined number of users and groups, where most of which have a historical meaning but are not in use in modern systems. *Table 4.2* shows the users configured in a fresh installed Ubuntu Server:

Username	UID	Primary GID	User information	Home dir.	Shell
root	0	0	root	/root	/bin/bash
daemon	1	1	daemon	/usr/sbin	/usr/sbin/nologin
bin	2	2	bin	/bin	/usr/sbin/nologin
sys	3	3	sys	/dev	/usr/sbin/nologin
sync	4	65534	sync	/bin	/bin/sync
games	5	60	games	/usr/games	/usr/sbin/nologin
man	6	12	man	/var/cache/man	/usr/sbin/nologin
lp	7	7	lp	/var/spool/lpd	/usr/sbin/nologin
mail	8	8	mail	/var/mail	/usr/sbin/nologin
news	9	9	news	/var/spool/news	/usr/sbin/nologin
uucp	10	10	uucp	/var/spool/uucp	/usr/sbin/nologin
proxy	13	13	proxy	/bin	/usr/sbin/nologin
www-data	33	33	www-data	/var/www	/usr/sbin/nologin
backup	34	34	backup	/var/backups	/usr/sbin/nologin
list	38	38	Mailing List Manager	/var/list	/usr/sbin/nologin
irc	39	39	ircd	/run/ircd	/usr/sbin/nologin
gnats	41	41	Gnats Bug-Reporting System (admin)	/var/lib/gnats	/usr/sbin/nologin
nobody	65534	65534	nobody	/nonexistent	/usr/sbin/nologin

Table 4.2: *Default users created in a Ubuntu Server*

The user *nobody* is a special user, used by some services when they do not want to use a dedicated user to run an application. A group named *nobody* is also created during the installation, as observed in *table 4.3* of default groups created in an Ubuntu Server:

Group	GID	Group	GID	Group	GID	Group	GID
root	0	news	9	floppy	25	irc	39
daemon	1	uucp	10	tape	26	src	40
bin	2	man	12	sudo	27	gnats	41
sys	3	proxy	13	audio	29	shadow	42
adm	4	kmem	15	dip	30	utmp	43
tty	5	dialout	20	www-data	33	video	44
disk	6	fax	21	backup	34	plugdev	46
lp	7	voice	22	operator	37	staff	50
mail	8	cdrom	24	list	38	games	60

Table 4.3: *Default groups created in Ubuntu Server*

Best practices for user accounts

For human user accounts, it is recommended that the account is protected with strong passwords and expiration rules are applied. When possible, it is also recommended to disable the password login (for example, in *SSH*) and instead use only SSH private/public keys for login.

It is important to keep a list of the users required for each system and remove the non-required users for the system to avoid password breaches or vulnerabilities, which can affect our infrastructure. Another important thing to keep track of is the user and group belonging to mapping, thus, ensuring the user does not belong to a group of which he should not be a part of.

Automating the creation and maintenance of the users with tools like *Ansible* will help administrators to keep the systems secure and easy to maintain.

Commands to administrate users

Command *id*

This simple command shows information about the user and the groups to which the user belongs. This command can be executed without any argument to get information for the user who is executing it. Moreover, by specifying this command as an

argument, a user will be able to see the user's group information. The common options for the command this command are shown in *table 4.4*:

Option	Description
-g, --group	print only the effective group ID
-G, --groups	print all group IDs
-u, --user	print only the effective user ID

Table 4.4: *Common options for the command id*

In *figure 4.1*, an example of an output without options and arguments but with the argument of another user and using the options listed previously is illustrated as follows:

```
agonzalez@ubuntu:~$ id
uid=1000(agonzalez) gid=1000(agonzalez) groups=1000(agonzalez),4(adm),24(cdrom),
27(sudo),30(dip),46(plugdev),110(lxd)
agonzalez@ubuntu:~$ id root
uid=0(root) gid=0(root) groups=0(root)
agonzalez@ubuntu:~$ id -u
1000
agonzalez@ubuntu:~$ id -g
1000
agonzalez@ubuntu:~$ id -G
1000 4 24 27 30 46 110
```

Figure 4.1: *Output examples using command id*

Commands useradd and adduser

The command **useradd** is a command available in all Linux distributions, with the aim of creating users. Some distributions, for example Ubuntu Server, offers a tool **adduser** to help to introduce interactively the user information. In some other distributions, for example, Red Hat Enterprise Linux, the command **adduser** invokes the command **useradd** when it is executed.

The default values (specified in the file **/etc/default/useradd**) *can* be listed with the command **useradd** -D, as shown in *figure 4.2*:

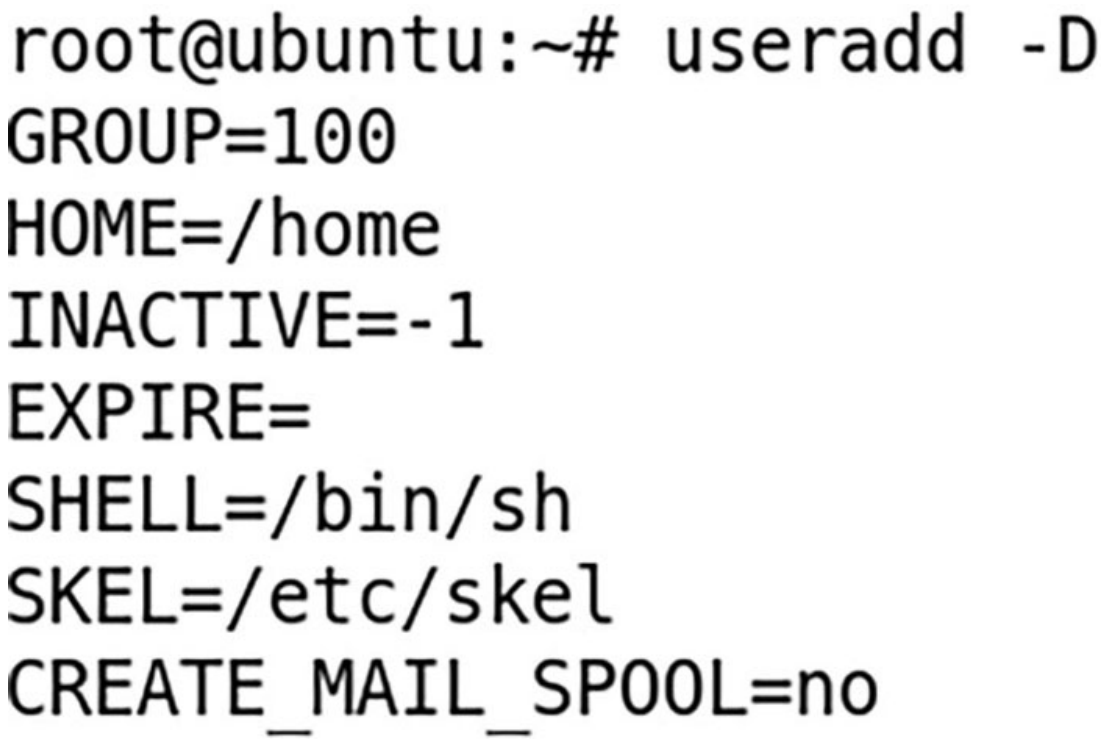


Figure 4.2: *Example of default options for command useradd*

The only argument required for this command is for the username to be created. The most common options for the command **`useradd`** are shown in *table 4.5*:

Option	Description
`-c, --comment COMMENT`	Usually used to specify the user's full name.
`-d,--home-dir HOME_DIR`	The home directory for the user instead use /home/<username>
`-g, --gid GROUP`	The main group name or group id for the user.
`-G,--groups GROUP1,GRP2`	A list, command separated, of secondary groups for the user.
`-k, --skel SKEL_DIR`	The skeleton directory is to be used instead */etc/skel*.
`-m, --create-home`	Create the user's home directory if it does not exist.
`-r, --system`	Create a system account instead of a regular user. GID will be different.
`-s, --shell SHELL`	The path of the user's login shell.
`-u, --uid UID`	The numerical value of the user's ID. Useful for migrating users.
`-U, --user-group`	Create a group with the same name as the user and set it as user's primary group.

Table 4.5: *Common options for the command useradd*

In *figure 4.3*, a user named **`testuser`** is created, specifying some of the options from the previous table, as well as the options with command *id*:

```
root@ubuntu:~# useradd -c "Test User" -G syslog -m testuser
root@ubuntu:~# id testuser
uid=1001(testuser) gid=1001(testuser) groups=1001(testuser),113(syslog)
```

Figure 4.3: *Example creating a user testuser using useradd.*

The command **`adduser`**, when available as a separate tool, usually is used without any options. However, the popular ones are described in *table 4.6*:

Option	Description
--home DIR	The home directory instead of the default.
--shell SHELL	The default shell instead defaults one.
--no-create-home	Do not create the home directory.
--gecos GECOS	Specify the user information
--gid ID	Specify the primary group ID or name.

Table 4.6: *Common options for the command adduser*

Figure 4.4 shows the output and the interaction with the command **`adduser`**:

```
root@ubuntu:~# adduser testuser2
Adding user `testuser2' ...
Adding new group `testuser2' (1002) ...
Adding new user `testuser2' (1002) with group `testuser2' ...
Creating home directory `/home/testuser2' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for testuser2
Enter the new value, or press ENTER for the default
        Full Name []: Test User2
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n]
```

Figure 4.4: *Example creating a user testuser2 using adduser*

Command *usermod*

The command **`usermod`** allows to modify most of the parameters for the user after it is created. The only argument required is for the username to be modified. The most common options, similar to **`useradd`**, are described in *table 4.7*:

Option	Description

-a, --append	Add the user to the secondary group(s). Use only with the -G option
-c, --comment COMMENT	Usually used to specify user's full name.
-d, --home HOME_DIR	The user's new login directory.
-g, --gid GROUP	The new main group name or group id for the user.
-G,--groups GROUP1,GRP2	A list, command separated, of secondary groups for the user.
-L, --lock	Lock a user's password.
-m, --move-home	Move the content of the user's home directory to the new location.
-s, --shell SHELL	The path of the user's new login shell.
-U, --unlock	Unlock a user's password.

Table 4.7: *Common options for the command usermod*

In *figure 4.5*, a secondary group named **`syslog`** is added to the user **`testuser2`** created previously:

```
root@ubuntu:~# usermod -G syslog testuser2
root@ubuntu:~# id testuser2
uid=1002(testuser2) gid=1002(testuser2) groups=1002(testuser2),113(syslog)
```

Figure 4.5: *Example adding a secondary group to user testuser2 using usermod*

Command *lslogins*

The command **`lslogins`** is a modern command to display information about available users in the system. Not specifying any argument or options will list all the users and information about the last login and the number of processes running by the user if the password is locked. This can be shown in *figure 4.6*:

```
root@ubuntu:~# lslogins
  UID USER              PROC PWD-LOCK PWD-DENY LAST-LOGIN GECOS
    0 root               108        0        0            root
    1 daemon               0        0        1            daemon
    2 bin                  0        0        1            bin
  112 usbmux               0        0        1            usbmux daemon,,,
 1000 agonzalez            5        0        0      12:21 agonzalez,12,1234,12345
678
 1001 testuser             0        0        1            Test User
 1002 testuser2            0        0        0            Test User2,,,
```

Figure 4.6: *Output example (truncated) of command lslogins*

Specifying one of the existing users as argument, detailed information about the account is shown, as illustrated in *figure 4.7*:

```
root@ubuntu:~# lslogins agonzalez
KZAK>>> alloc '0x5654e7531780' for agonzalez
Username:                            agonzalez
UID:                                 1000
Gecos field:                         agonzalez,12,1234,12345678
Home directory:                      /home/agonzalez
Shell:                               /bin/bash
No login:                            no
Password is locked:                  no
Password not required:               no
Login by password disabled:          no
Password encryption method:          SHA-512
Primary group:                       agonzalez
GID:                                 1000
Supplementary groups:                lxd,adm,cdrom,sudo,dip,plugdev
Supplementary group IDs:             110,4,24,27,30,46
Last login:                          12:21
Last terminal:                       pts/0
Last hostname:                       192.168.122.1
Failed login:                        Jul01/22:31
Failed login terminal:               ssh:notty
Hushed:                              no
Password expiration warn interval:   7
Password changed:                    Jun26/00:00
Maximum change time:                 99999
Running processes:                   5

Last logs:
12:59 sudo[1476]: pam_unix(sudo-i:session): session closed for user root
12:59 su[1492]: (to root) agonzalez on pts/0
12:59 su[1492]: pam_unix(su-l:session): session opened for user root(uid=0) by a
gonzalez(uid=1000)
```

Figure 4.7: *Output example for command lslogins specifying a user.*

Some of the common options for the command *lslogins* are listed in *table 4.8*:

Option	Description
-a,--acc-expiration	Shows the date of the last password change and the account expiration date.
-f, --failed	Display data about the users' last failed login attempts.
-g, --groups=groups	Only show data of users belonging to groups.
-l, --logins=logins	Only show data of users with a login specified in logins. Comma separated.
-p, --pwd	Display information related to login by a password.
-u, --user-accs	Show user accounts and hidden system users.

Table 4.8: *Common options for the command lslogins*

Commands *who* and *w*

The command *who* shows information about users logged into the system. An example is shown in *figure 4.8*:

```
root@ubuntu:~# who
agonzalez tty1         2022-07-23 12:21
agonzalez pts/0        2022-07-23 12:21 (192.168.122.1)
```

Figure 4.8: *Output example of command who.*

The command *w* shows information about users logged into the system and also the main command they are executing, as can be seen in *figure 4.9*:

```
root@ubuntu:~# w
 13:15:56 up 58 min,  2 users,  load average: 0.01, 0.11, 0.07
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
agonzale tty1     -                12:21   54:57   0.05s  0.03s -bash
agonzale pts/0    192.168.122.1    12:21    0.00s  0.10s  0.02s sshd: agonzalez
```

Figure 4.9: *Output example of command w.*

Command *userdel*

This command is used to remove local users from the system. It requires an argument to specify the user who is to be deleted. The most common options for this command are shown in *table 4.9*:

Option	Description
-f, --force	This option forces the removal of the user account, even if the user is still logged in.
-r, --remove	Files in the user's home directory will be removed.

Table 4.9: *Common options for the command userdel*

In *figure 4.10*, the user **`testuser2`** and his files are deleted:

```
root@ubuntu:~# userdel -r testuser2
userdel: testuser2 mail spool (/var/mail/testuser2) not found
```

Figure 4.10: *Example of deleting the user testuser2 and his home directory.*

Command *passwd*

This command is used to operate with the password and the expiration password date of the local users. Creating a user with the command **`useradd`** requires this command to be run afterward to set the user's password. An administrator or regular user can run it without an argument to change their own password. The most common options are shown in *table 4.10*:

Option	Description
`-l, --lock`	Lock the password of the user, and it is available to root only.

`-u, --unlock`	Unlock the password for the user.
`-e, --expire`	The user will be forced to change the password in the next login attempt.
`-n, --minimum DAYS`	Minimum password lifetime, in days.
`-x, --maximum DAYS`	Maximum password lifetime.
`-w, --warning DAYS`	Days in advance, the user will receive warnings her password will expire.
`-i, --inactive DAYS`	Number of days that will pass before an expired password will be taken means that the account is inactive and should be disabled.
`-S, --status`	Output short information about the status of the password.

Table 4.10: *Common options for the command passwd*

In *figure 4.11*, it is shown how to set the password for one user and how to lock the account (the command *passwd -S* is showing the letter *L* indicating is locked):

```
root@ubuntu:~# passwd testuser
New password:
Retype new password:
passwd: password updated successfully
root@ubuntu:~# passwd -l testuser
passwd: password expiry information changed.
root@ubuntu:~# passwd -S testuser
testuser L 07/23/2022 0 99999 7 -1
```

Figure 4.11: *Output examples of the usage of the command passwd.*

Command *chage*

This command is used to change the user password expiry information. The popular options are shown in *table 4.11*:

Option	Description
`-d, --lastday LAST_DAY`	Set the date when the password was last changed. I
`-E, --expiredate EXPIRE_DATE`	Set the date when the user's account will no longer be accessible.
`-I, --inactive INACTIVE`	Set the number of days of inactivity after a password has expired before the account is locked.
`-l, --list`	Show account aging information.
`-m, --mindays MIN_DAYS`	Set the minimum number of days between password changes to MIN_DAYS.
`-M, --maxdays MAX_DAYS`	Set the maximum number of days during which a password is valid.

`-W, --warndays WARN_DAYS`	Set the number of days of warning before a password change is required.

Table 4.11: *Common options for the command chage*

In *figure 4.12*, the user **`testuser`** is forced to change the password in the next login, and information about the account is shown:

```
root@ubuntu:~# chage -d 0 testuser
root@ubuntu:~# chage -l testuser
Last password change                                    : password must be changed
Password expires                                        : password must be changed
Password inactive                                       : password must be changed
Account expires                                         : never
Minimum number of days between password change          : 0
Maximum number of days between password change          : 99999
Number of days of warning before password expires       : 7
```

Figure 4.12: *Examples using command chage.*

Command *last*

This command will show a listing of users who logged into the system. It is possible to filter the output with some of the options shown in *table 4.12*:

Option	Description
`-n, --limit number`	Tell last how many lines to show.
`-p, --present time`	Display the users who were present at the specified time.
`-R, --nohostname`	Suppresses the display of the hostname field.
`-s, --since time`	Display the state of logins since the specified time.
`-t, --until time`	Display the state of logins until the specified time.

Table 4.12: *Common options for the command last*

In *figure 4.13*, we are shown only the last three entries of the log of the users who were logged:

```
root@ubuntu:~# last  -n 3
agonzale pts/0        192.168.122.1    Sat Jul 23 12:21   still logged in
agonzale tty1                          Sat Jul 23 12:21   still logged in
reboot   system boot  5.15.0-40-generi Sat Jul 23 12:17   still running

wtmp begins Sun Jun 26 20:41:10 2022
```

Figure 4.13: *Example using command last*

Commands to manipulate groups

Command *groupadd*

This command will add a local group to the system. The two common options are shown in *table 4.13*:

Option	Description
`-g, --gid GID`	The numerical value of the group's ID. Useful for migrations.
`-r, --system`	Create a system group.

Table 4.13: *Common options for the command groupadd*

In *figure 4.14*, a new group called *example* is created:

```
root@ubuntu:~# groupadd example
```

Figure 4.14: *Example using command groupadd.*

Command *groups*

This simple command shows the groups the user invoking this command belongs to. Specifying an argument will show the groups that the user is part of. In *figure 4.15*, two scenarios are shown:

```
root@ubuntu:~# groups
root
root@ubuntu:~# groups agonzalez
agonzalez : agonzalez adm cdrom sudo dip plugdev lxd
```

Figure 4.15: *Example using command groups.*

Command *groupmod*

The command **`groupmod`** allows to modify some of the parameters of the specified group. The common options are shown in *table 4.14*:

Option	Description
`-g, --gid GID`	The new group id for the group.
`-n, --new-name NAME`	The new name for the group.

Table 4.14: *Common options for the command groupmod*

In *figure 4.16*, the group **`example`** previously created is renamed to *writers*:

```
root@ubuntu:~# groupmod -n writes example
```

Figure 4.16: *Example using command groupmod.*

Command *groupdel*

This command is used to delete groups for the system, and the only required argument is for the group to be deleted. In *figure 4.17*, the previously renamed group is deleted:

```
root@ubuntu:~# groupdel writes
```

Figure 4.17: *Example using command groupdel.*

Command *gpasswd*

This command administrates **`/etc/group`** and **`/etc/gshadow`** files. Every group can have administrators, members, and a password. The required argument is the group to operate it, and *table 4.15* shows the most common options:

Option	Description
`-a, --add user`	Add the user to the named group.
`-d, --delete user`	Remove the user from the named group.
`-r, --remove-password`	Remove the password from the named group.
`-R, --restrict`	Restrict the access to the named group.
`-A, --administrators user,…`	Set the list of administrative users.
`-M, --members user,…`	Set the list of group members.

Table 4.15: *Common options for the command gpasswd*

In *figure 4.18*, a group **`writers`** is created, where the user **`agonzalez`** is set as an administrator of the group, and a password is also set for the group:

```
root@ubuntu:~# groupadd writers
root@ubuntu:~# gpasswd -A agonzalez writers
root@ubuntu:~# gpasswd writers
Changing the password for group writers
New Password:
Re-enter new password:
```

Figure 4.18: *Example using command gpasswd*

Command *newgrp*

This command allows users to be part of a group either if they know the password of the group or if they are part of the group, but when they logged in to the system, they

were not. The only argument required is the group to join; if the password is needed, it will be prompted. In *figure 4.19*, the user `testuser` is joining the group `writers` using the password:

```
testuser@ubuntu:~$ groups
testuser syslog
testuser@ubuntu:~$ newgrp writers
Password:
testuser@ubuntu:~$ groups
writers syslog testuser
```

Figure 4.19: *Example using command newgrp.*

Introduction to RPM and DEB package formats

Linux distributions are working with packaging systems to make the process of installing new software, updating the software, or removing it easier. These packaging systems are responsible for solving the dependencies needed and offering the possibility of verifying the status.

A package consists of an archive of files and meta-data that are used to install and erase the archive files. The meta-data includes helper scripts, file attributes, and descriptive information about the package. Most of the distributions offer two types of packages:

- **Binary packages**: used to encapsulate the software to be installed.
- **Source packages**: containing the source code and everything else that is needed to produce binary packages.

There are two popular package managers and formats used by the most used distributions: *RPM Package Manager* and *Debian package*.

The **RPM Package Manager** *(***RPM***)* is an open packaging system that runs on *Red Hat Enterprise Linux* as well as other Linux distributions such as *Fedora, CentOS, Alma Linux,* and *Rocky Linux*. There are different tools to operate with the *RPM Package Manager (RPM):*

- **rpm**: The main tool to query, install, upgrade or remove packages from the system. This command will use the local *RPM* database when querying or uninstalling, and it will use local files with the extension *.rpm* to install or upgrade packages.

- **yum** (*Yellowdog Updater Modified*)**:** The primary tool for getting, installing, deleting, and querying packages from software repositories. This tool will be responsible for downloading the package and the dependencies needed to install the software requested. It allows the update or upgrade of the full system.
- **dnf** (*Dandified YUM*): The newest version of YUM, having compatibility with the YUM command and options. It is faster than the original *yum*, is more extensible, and resolves the dependencies in a better way.

The *Debian package* is the package format for Debian, Ubuntu, and derivatives. The extension for the packages is *.deb*. The main tools to operate with the *Debian packages* are as follows:

- **dpkg:** The main tool to query, install, build and remove *Debian packages*. It acts as a frontend for the low-level commands *dpkg-query* (for listing) and *dpkg-deb (*to install software).
- **apt-get**: One of the popular tools for handling packages from software repositories. It allows to install, update, reinstall, and remove software in the system.
- **apt-cache**: This command is used to search packages in the software repositories. The system will keep a local copy of the list of the available packages, and this tool will query this local database to search for packages, search files inside packages or obtain information about a package.
- **apt**: Provides a high-level interface for package management. Using this command simplifies the use of only one command for the installation (*apt-get* tasks) or manipulation of packages and the search or obtaining package information (*apt-cache* tasks).

Commands to operate with RPM packages

Command rpm

This command as described previously is the main command to operate with RPM packages. It contains multiple options depending on the action required. The main actions and syntax are shown in *table 4.16*:

Action	Syntax
Querying and verifying	rpm -q/--query [select options] [query options] [package]
Installing packages	rpm -i/--install [install-options] package.rpm
Upgrading packages	rpm -U/--upgrade [install-options] package.rpm *rpm -F/--freshen [install-options] package.rpm*
Reinstall packages	rpm --reinstall [install-options] package.rpm

Removing packages	*rpm -e/--erase packagename*

Table 4.16: *Actions and syntax for the command rpm*

The most common global option is *-v (--verbose*) for all the action. For *select options* and *query options* and their descriptions, refer to *table 4.17*:

Select options	Description
`-a, --all [SELECTOR]`	Query all installed packages or filter by SELECTOR
`-f, --file FILE`	Query the package owning the installed FILE specified.
`-p, --package package.rpm`	Query a package file and not an installed package.
`--whatrequires PACKAGE`	Query the packages that require the PACKAGE specified.
`--whatconflicts PACKAGE`	Query the packages that conflict with the PACKAGE specified.
`--changelog`	Display change information for the package.
`--conflicts`	List the packages conflicting with the one queried.
`-i / --info`	Obtains information about the package.
`-l / --list`	List the files inside the package
`-R / --requires`	List the packages on which this package depends.

Table 4.17: *Select options and query options*

Some examples of querying packages are shown in the following figures:

- *Figure 4.20* is a Query of all the packages in the system starting with the word *kernel:*

```
[root@rhel ~]# rpm -qa kernel*
kernel-tools-libs-5.14.0-70.13.1.el9_0.x86_64
kernel-core-5.14.0-70.13.1.el9_0.x86_64
kernel-modules-5.14.0-70.13.1.el9_0.x86_64
kernel-5.14.0-70.13.1.el9_0.x86_64
kernel-tools-5.14.0-70.13.1.el9_0.x86_64
```

Figure 4.20: *Output example query installed packages with a pattern.*

- *Figure 4.21* features obtaining information about one specified package (`jq`):

```
[root@rhel ~]# rpm -qi jq
Name        : jq
Version     : 1.6
Release     : 12.el9
Architecture: x86_64
Install Date: Sun 26 Jun 2022 10:50:42 PM CEST
Group       : Unspecified
Size        : 419885
```

```
License     : MIT and ASL 2.0 and CC-BY and GPLv3
Signature   : RSA/SHA256, Thu 25 Nov 2021 07:16:22 AM CET, Key ID 199e2f91fd431d51
Source RPM  : jq-1.6-12.el9.src.rpm
Build Date  : Mon 15 Nov 2021 04:02:54 PM CET
Build Host  : x86-vm-56.build.eng.bos.redhat.com
Packager    : Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
Vendor      : Red Hat, Inc.
URL         : http://stedolan.github.io/jq/
Summary     : Command-line JSON processor
Description :
lightweight and flexible command-line JSON processor

jq is like sed for JSON data – you can use it to slice
and filter and map and transform structured data with
the same ease that sed, awk, grep and friends let you
play with text.
```

Figure 4.21: *Output example getting information of an installed package*

- *Figure 4.22* shows checking the owner package of one installed file in the system:

```
[root@rhel ~]# rpm -qf /etc/services
setup-2.13.7-6.el9.noarch
```

Figure 4.22: *Output example querying who is the owner of a file.*

- *Figure 4.23* shows query a .rpm file listing the dependencies:

```
[root@rhel ~]# rpm -qp python3-netaddr-0.8.0-5.el9.noarch.rpm  --requires
warning: python3-netaddr-0.8.0-5.el9.noarch.rpm: Header V3 RSA/SHA256 Signature, key
 ID fd431d51: NOKEY
/usr/bin/python3
python(abi) = 3.9
python3.9dist(setuptools)
rpmlib(CompressedFileNames) <= 3.0.4-1
rpmlib(FileDigests) <= 4.6.0-1
rpmlib(PartialHardlinkSets) <= 4.0.4-1
rpmlib(PayloadFilesHavePrefix) <= 4.0-1
rpmlib(PayloadIsZstd) <= 5.4.18-1
```

Figure 4.23: *Output example listing the dependencies of a package.*

- *Figure 4.24* lists the files inside a package:

```
[root@rhel ~]# rpm -ql grub2-pc
/boot/grub2/grub.cfg
/boot/loader/entries
/etc/dnf/protected.d/grub2-pc.conf
/etc/grub2.cfg
```

Figure 4.24: *Output example listing the files created by an installed package.*

For installing, upgrading, and deleting, the following options shown in *table 4.18* are the most common:

Option	Description
`--force`	Same as using --replacepkgs, --replacefiles, and --oldpackage.
`-h, --hash`	Shows the progress as the package archive is unpackaged.
`--nodeps`	Do not do a dependency check before installing or upgrading a package.
`--oldpackage`	Allow an upgrade to replace a newer package with an older one.
`--replacefiles`	Install the packages even if they replace files from other installed packages.
`--replacepkgs`	Install the packages even if some of them are already installed on this system.
`--test`	Do not install the package; simply check for and report potential conflicts.

Table 4.18: *Options for installing, upgrading, and deleting*

Some examples of installing, updating, reinstalling, and uninstalling packages are shown in the following figures:

- *Figure 4.25* features installing a package from a local file.

```
[root@rhel ~]# rpm -ihv lftp-4.9.2-4.el9.x86_64.rpm
warning: lftp-4.9.2-4.el9.x86_64.rpm: Header V3 RSA/SHA256 Signature, key ID fd431d5
1: NOKEY
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:lftp-4.9.2-4.el9                 ################################# [100%]
```

Figure 4.25: *Output example installing a package from a local file.*

- Upgrade a package from a local file, as shown in *figure 4.26*:

```
[root@rhel ~]# rpm -Uhv vim-minimal-8.2.2637-16.el9_0.2.x86_64.rpm
warning: vim-minimal-8.2.2637-16.el9_0.2.x86_64.rpm: Header V3 RSA/SHA256 Signature,
 key ID fd431d51: NOKEY
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:vim-minimal-2:8.2.2637-16.el9_0.2################################# [ 50%]
Cleaning up / removing...
   2:vim-minimal-2:8.2.2637-15.el9    ################################# [100%]
```

Figure 4.26: *Output example upgrading a package from a local file.*

The difference between *-U (`--upgrade`)* and *-F* (`--freshen`) is that the first option will install the package if it previously was not installed; the second one will not perform any action if the previous version was not in the system. Both of them will uninstall the previous version and then install a new version for upgrading an existing package.

- *Figure 4.27* features reinstalling an existing package:

```
[root@rhel ~]# rpm --reinstall -hv vim-minimal-8.2.2637-16.el9_0.2.x86_64.rpm
warning: vim-minimal-8.2.2637-16.el9_0.2.x86_64.rpm: Header V3 RSA/SHA256 Signature,
 key ID fd431d51: NOKEY
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:vim-minimal-2:8.2.2637-16.el9_0.2################################# [ 50%]
Cleaning up / removing...
   2:vim-minimal-2:8.2.2637-16.el9_0.2################################# [100%]
```

Figure 4.27: *Output example reinstalling a package using a local file.*

- *Figure 4.28* features removing an installed package from the system.

```
[root@rhel ~]# rpm -q lftp
lftp-4.9.2-4.el9.x86_64
[root@rhel ~]# rpm -ehv lftp
Preparing...                          ################################# [100%]
Cleaning up / removing...
   1:lftp-4.9.2-4.el9                 ################################# [100%]
[root@rhel ~]# rpm -q lftp
package lftp is not installed
```

Figure 4.28: *Output example removing a package from the system.*

Commands *yum* and *dnf*

Both command work with software repositories having the option to work with local files. The directory containing the repositories for the system is */etc/yum.repos.d/*. Inside it, there are files with the extension *.repo* containing the information of the repository. An example of a repository definition for a Red Hat Enterprise Linux 9 is shown in the following snippet:

```
[rhel-9-for-x86_64-baseos-rpms]
name = Red Hat Enterprise Linux 9 for x86_64 - BaseOS (RPMs)
```

```
baseurl =
https://cdn.redhat.com/content/dist/rhel9/$releasever/x86_64/baseos/os
enabled = 1
gpgcheck = 1
gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
sslverify = 1
sslcacert = /etc/rhsm/ca/redhat-uep.pem
sslclientkey = /etc/pki/entitlement/3068237762125458331-key.pem
sslclientcert = /etc/pki/entitlement/3068237762125458331.pem
metadata_expire = 86400
enabled_metadata = 1
```

Running the command *dnf repolist* will show the enabled repositories in the system, as shown in *figure 4.29*:

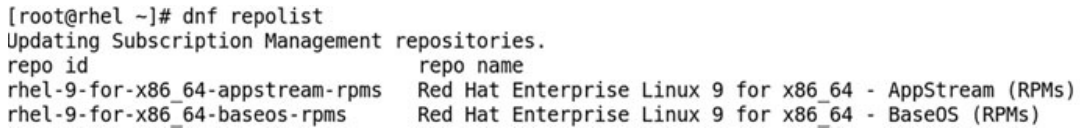


Figure 4.29: *Output example listing the enabled repositories in the system*

The syntax for the commands *dnf* and *yum* is: *yum [options] action [args]*. The popular actions are shown in *table 4.19*:

Action	Description
autoremove	Remove all the packages installed as dependencies but are not needed anymore.
check-update	Check if there is an available update for the installed packages in the system.
distro-sync	Perform the operations needed to update the distribution to the latest version available.
download	Download the package(s) specified in the current directory.
downgrade	Downgrades the specified packages to the previously highest version available.
info	Shows information about the package indicated.
install	Installs the package(s) indicated and all the dependencies needed. It is possible as well to specify a *.rpm* file located in the local system or in a Web server. Groups can be installed by specifying the character at (*@*) as a prefix.
list	Shows all the packages installed and the available in software repositories enabled in the system.
makecache	Downloads and caches metadata for the enabled repositories in the system
provides	Finds the package providing the file or directory indicated as an argument.
reinstall	Reinstall the specified package(s).
remove	Removes the specified package(s).
search	Searches for a specific package(s).

upgrade	Upgrades the system or the package(s) indicated to the latest version available.

Table 4.19: *Actions for the commands dnf and yum*

Some of the popular global options are shown in *table 4.20*:

Option	Description
--downloadonly	Download package(s) and dependencies only without performing the installation.
--show-duplicates	If software repositories offer multiple versions for the same package, all of them will be listed. Default *dnf/yum* shows only the latest version.
-v, --verbose	Shows debug information.
-y, --assumeyes	Does not ask the users if they want to confirm the operation.

Table 4.20: *Popular global options for the commands dnf and yum*

In the following figures, the different tasks performed using command **`dnf`** are shown:

- *Figure 4.30* checks if there is any available package update available:

```
[root@rhel ~]# dnf check-update
Updating Subscription Management repositories.
Last metadata expiration check: 10:37:29 ago on Sat 23 Jul 2022 09:27:36 PM CEST.

curl.x86_64                   7.76.1-14.el9_0.4               rhel-9-for-x86_64-baseos-rpms
libcurl.x86_64                7.76.1-14.el9_0.4               rhel-9-for-x86_64-baseos-rpms
```

Figure 4.30: *Output example checking the packages that can be updated.*

- Upgrade the Linux distribution to the latest version available.
  - *Figure 4.31* shows that the first part of the output is the transaction summary information:

```
[root@rhel ~]# dnf distro-sync
Updating Subscription Management repositories.
Last metadata expiration check: 10:38:04 ago on Sat 23 Jul 2022 09:27:36 PM CEST.
Dependencies resolved.
================================================================================
 Package      Arch        Version                  Repository                  Siz
================================================================================
Upgrading:
 curl         x86_64      7.76.1-14.el9_0.4        rhel-9-for-x86_64-baseos-rpms  300
 libcurl      x86_64      7.76.1-14.el9_0.4        rhel-9-for-x86_64-baseos-rpms  288

Transaction Summary
================================================================================
Upgrade  2 Packages

Total download size: 588 k
Is this ok [y/N]: y
```

Figure 4.31: *First part of example output for command dnf distro-sync.*

  - After confirming the operation, typing *y* or using the option *-y/--assume-yes,* the part is the downloading packages progress, as is shown in *figure 4.32*:

```
Downloading Packages:
(1/2): libcurl-7.76.1-14.el9_0.4.x86_64.rpm                  86 kB/s | 288 kB     00:03
(2/2): curl-7.76.1-14.el9_0.4.x86_64.rpm                     90 kB/s | 300 kB     00:03
--------------------------------------------------------------------------------------
Total                                                       176 kB/s | 588 kB     00:03
Running transaction check
Transaction check succeeded.
```

Figure 4.32: *Second part of example output for command dnf distro-sync.*

- The last part of the output is the information about the transaction, which includes the installation, cleaning, and verification of the process for each of the packages. This is illustrated in *figure 4.33*:

```
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                              1,
  Upgrading        : libcurl-7.76.1-14.el9_0.4.x86_64                             1,
  Upgrading        : curl-7.76.1-14.el9_0.4.x86_64                                2,
  Cleanup          : curl-7.76.1-14.el9.x86_64                                    3,
  Cleanup          : libcurl-7.76.1-14.el9.x86_64                                 4,
  Running scriptlet: libcurl-7.76.1-14.el9.x86_64                                 4,
  Verifying        : curl-7.76.1-14.el9_0.4.x86_64                                1,
  Verifying        : curl-7.76.1-14.el9.x86_64                                    2,
  Verifying        : libcurl-7.76.1-14.el9_0.4.x86_64                             3,
  Verifying        : libcurl-7.76.1-14.el9.x86_64                                 4,
Installed products updated.

Upgraded:
  curl-7.76.1-14.el9_0.4.x86_64              libcurl-7.76.1-14.el9_0.4.x86_64

Complete!
```

Figure 4.33: *Last part of example output for command dnf distro-sync.*

- Download a package named **`dmidecode`** from a software repository to the current directory without installing it, as can be seen in *figure 4.34*:

```
[root@rhel ~]# dnf download dmidecode
Updating Subscription Management repositories.
Last metadata expiration check: 23:27:22 ago on Sat 23 Jul 2022 09:27:36 PM CEST.
dmidecode-3.3-7.el9.x86_64.rpm                               28 kB/s |  93 kB     00:03
[root@rhel ~]# ls -l dmidecode-3.3-7.el9.x86_64.rpm
-rw-r--r--. 1 root root 95279 Jul 24 20:55 dmidecode-3.3-7.el9.x86_64.rpm
```

Figure 4.34: *Output example downloading a package.*

- Install in the system a package named **`traceroute`** from a software repository. This is shown in *figure 4.35*:

```
[root@rhel ~]# dnf install -y traceroute
Updating Subscription Management repositories.
Last metadata expiration check: 10:49:53 ago on Sat 23 Jul 2022 09:27:36 PM CEST.
Dependencies resolved.
================================================================================
 Package         Arch        Version                Repository                Size
================================================================================
Installing:
 traceroute      x86_64      3:2.1.0-16.el9         rhel-9-for-x86_64-baseos-rpms       61 k

Transaction Summary
================================================================================
Install  1 Package
```

```
(omitted)
Installed:
  traceroute-3:2.1.0-16.el9.x86_64

Complete!
```

Figure 4.35: *Package installation output example*

- Search for a package named **`firefox`** in the software repository, as illustrated in *figure 4.36*:

```
[root@rhel ~]# dnf search firefox
Updating Subscription Management repositories.
Last metadata expiration check: 10:57:35 ago on Sat 23 Jul 2022 09:27:36 PM CEST.
============================== Name & Summary Matched: firefox ==============================
firefox.x86_64 : Mozilla Firefox Web browser
```

Figure 4.36: *Output example searching a package using dnf.*

- *Figure 4.37* illustrates checking what package provides the file **`/etc/sensors3.conf`**:

```
[root@rhel ~]# dnf provides /etc/sensors3.conf
Updating Subscription Management repositories.
Last metadata expiration check: 10:58:09 ago on Sat 23 Jul 2022 09:27:36 PM CEST.
lm_sensors-3.6.0-10.el9.x86_64 : Hardware monitoring tools
Repo        : rhel-9-for-x86_64-appstream-rpms
Matched from:
Filename    : /etc/sensors3.conf
```

Figure 4.37: *Output example searching who provides a file.*

- Remove a package previously installed in the system named *traceroute*, as can be seen in *figure 4.38*:

```
[root@rhel ~]# dnf remove -y traceroute
Updating Subscription Management repositories.
Dependencies resolved.
================================================================================================
 Package         Arch        Version               Repository                            Size
================================================================================================
Removing:
 traceroute      x86_64      3:2.1.0-16.el9        @rhel-9-for-x86_64-baseos-rpms       108 k

Transaction Summary
================================================================================================
Remove  1 Package

Freed space: 108 k
Running transaction check
```

```
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Erasing          : traceroute-3:2.1.0-16.el9.x86_64                       1/1
  Running scriptlet: traceroute-3:2.1.0-16.el9.x86_64                       1/1
  Verifying        : traceroute-3:2.1.0-16.el9.x86_64                       1/1
Installed products updated.

Removed:
  traceroute-3:2.1.0-16.el9.x86_64

Complete!
```

Figure 4.38: *Output example removing a package*

Commands to operate with DEB packages

Command *dpkg*

This command is the main tool to perform operations with *Debian package* files, including installation, getting information, or removing them from the system. *Table 4.21* shows the most common options to perform different tasks for the command **dpkg**:

Option	Description
`-i, --install file.deb|-a|--pending`	Install the package.
`-r, --remove package|-a|--pending`	Remove an installed package keeping the configuration files.
`-P, --purge package…|-a|--pending`	Uninstall (if it is installed) and remove configuration files from the system.
`--configure package…|-a|--pending`	Configure a package that has not yet been configured.
`-V, --verify [package-name…]`	Verifies the integrity of the package
`-C, --audit [package-name…]`	Performs database sanity and consistency checks
`-I, --info archive [control-file…]`	Show information about a package.
`-l, --list package-name-pattern`	List packages matching the given pattern.

`-s, --status package-name`	Report the status of the specified package.
`-L, --listfiles package-name`	List files installed to your system from package-name.
`-S, --search filename-search-pattern`	Search for a filename from installed packages.

Table 4.21: *Most common options for the command dpkg*

For the first four options, it is possible to specify the options **`-a`** or **`-pending`** (instead of the package name) to operate with packages that are not fully installed or configured.

In the following figures, different operations with *Debian* packages are illustrated:

- Obtain information about one **`.deb`** file for the *lftp* software, as can be seen in *figure 4.39*:

```
root@ubuntu:~# dpkg -I lftp_4.9.2-1build1_amd64.deb
 new Debian package, version 2.0.
 size 720478 bytes: control archive=1818 bytes.
      15 bytes,     1 lines      conffiles
    1732 bytes,    32 lines      control
    1279 bytes,    20 lines      md5sums
 Package: lftp
 Version: 4.9.2-1build1
 Architecture: amd64
 Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
 Installed-Size: 1912
 Depends: libc6 (>= 2.34), libgnutls30 (>= 3.7.2), libidn2-0 (>= 2.0.0), libreadline8 (>= 6.0
), libstdc++6 (>= 5), libtinfo6 (>= 6), zlib1g (>= 1:1.1.4), netbase
 Recommends: ssh-client | openssh-client
 Section: net
 Priority: optional
 Homepage: https://lftp.tech
 Description: Sophisticated command-line FTP/HTTP/BitTorrent client programs
  Lftp is a file retrieving tool that supports FTP, HTTP, FISH, SFTP, HTTPS,
  FTPS and BitTorrent protocols under both IPv4 and IPv6. Lftp has an amazing
  set of features, while preserving its interface as simple and easy as possible.
```

Figure 4.39: *Output example getting information of a local .deb file.*

```
root@ubuntu:~# dpkg -i lftp_4.9.2-1build1_amd64.deb
Selecting previously unselected package lftp.
(Reading database ... 108789 files and directories currently installed.)
Preparing to unpack lftp_4.9.2-1build1_amd64.deb ...
Unpacking lftp (4.9.2-1build1) ...
Setting up lftp (4.9.2-1build1) ...
Processing triggers for man-db (2.10.2-1) ...
```

Figure 4.40: *Output example installing a local .deb file to the system.*

1. Install a software named *lftp* from a local file, as shown in *figure 4.40*:
2. List the files created by the package named **`pci.ids`**, as shown in *figure 4.41*:

```
root@ubuntu:~# dpkg -L pci.ids
/.
/usr
/usr/share
/usr/share/doc
/usr/share/doc/pci.ids
/usr/share/doc/pci.ids/changelog.Debian.gz
/usr/share/doc/pci.ids/copyright
/usr/share/misc
/usr/share/misc/pci.ids
```

Figure 4.41: *Output example listing the files created by a specific package.*

3. List from all packages installed in the system, as shown in *figure 4.42*:

```
root@ubuntu:~# dpkg -l
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                                  Version                                   Architectu
+++-=====================================-=========================================-==========
ii  adduser                               3.118ubuntu5                              all
ii  amd64-microcode                       3.20191218.1ubuntu2                       amd64
ii  apparmor                              3.0.4-2ubuntu2                            amd64
ii  apport                                2.20.11-0ubuntu82.1                       all
ii  apport-symptoms                       0.24                                      all
ii  apt                                   2.4.5                                     amd64
ii  apt-utils                             2.4.5                                     amd64
ii  base-files                            12ubuntu4                                 amd64
ii  base-passwd                           3.5.52build1                              amd64
ii  bash                                  5.1-6ubuntu1                              amd64
ii  bash-completion                       1:2.11-5ubuntu1                           all
```

Figure 4.42: *Output example listing the packages installed in the system.*

4. Filter the list of the packages installed using a pattern, as shown in *figure 4.43*:

```
root@ubuntu:~# dpkg -l "distro-info*"
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name           Version       Architecture Description
+++-==============-=============-============-=================================================
ii  distro-info     1.1build1     amd64        provides information about the distributions
ii  distro-info-data 0.52ubuntu0.1 all          information about the distributions' release
lines 1-7/7 (END)
```

Figure 4.43: *Output example filtering the list of packages installed in the system.*

5. Search what installed packages created the file **`/etc/sudo.conf`**, as shown in *figure 4.44*:

```
root@ubuntu:~# dpkg -S /etc/sudo.conf
sudo: /etc/sudo.conf
```

Figure 4.44: *Output example querying which package owns a file in the system.*

6. Remove the software previously installed named *lftp*. Configuration files will be kept in the system, as illustrated in *figure 4.45*:

```
root@ubuntu:~# dpkg --remove lftp
(Reading database ... 108812 files and directories currently installed.)
Removing lftp (4.9.2-1build1) ...
Processing triggers for man-db (2.10.2-1) ...
```

Figure 4.45: *Output example removing a package from the system.*

7. Remove the configurations kept previously uninstalling the software *lftp*, as shown in *figure 4.46*:

```
root@ubuntu:~# dpkg --purge lftp
(Reading database ... 108790 files and directories currently installed.)
Purging configuration files for lftp (4.9.2-1build1) ...
```

Figure 4.46: *Output example removing remaining configuration files*

Command apt-get, apt-cache, and apt-file

These three commands work with the repositories configured in the file **`/etc/apt/sources.list`** or in the files configured inside **`/etc/apt/sources.list.d/`**. A repository file contains a list of repositories, and each repository specifies if it is for binary packages (**`deb`**) or for the source (**`deb-src`**), as well as the address of the repository, the distribution version, and the type of the software in this repository; (**`main`** for free software, **`restricted`** for non-free software. The context of the word *free* where is related to open source and not about cost). An example of repositories entries are shown for a Ubuntu Server with the codename for the distribution called **`jammy`**:

```
deb http://es.archive.ubuntu.com/ubuntu jammy main restricted
deb http://es.archive.ubuntu.com/ubuntu jammy-updates main restricted
```

The command **`apt-get`** is used to install, update or remove software using a software repository. The common actions are shown in *table 4.22*:

Action	Description
update	Resynchronize the package index files from their sources.
upgrade	Install the newest versions of all packages currently installed in the system.
dist-upgrade	Upgrades the packages related to the core of the distribution.
install	Install the package(s) specified.

reinstall	Reinstalls the package specified. It is an alias to use *install --reinstall.*
remove	Remove the package(s) specified.
purge	Remove the package(s) specified and the configuration belonging to them.
source	Fetch the source instead the binary packages.
download	Download the package(s) to the current directory.
clean	Cleans the local files related to the information for the software repositories.
autoremove	Remove the non-needed anymore packages automatically installed as dependences.

Table 4.22: *Actions for the command apt-get*

Some popular options for the command **`apt-get`** are shown in *table 4.23*:

Option	Description
`-q, --quiet`	Reduces the information in the output.
`-s, --dry-run`	Do not perform any change in the system; instead, simulates the action to perform.
`-y, --yes`	Do not ask for confirmation for the requested task.

Table 4.23: *Popular options for the command apt-get*

The following figures provide examples of the use of **`apt-get`** with the actions shown previously:

- Update the local copy of the package index from repositories, as illustrated in *figure 4.47*:

```
root@ubuntu:~# apt-get update
Hit:1 http://es.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://es.archive.ubuntu.com/ubuntu jammy-security InRelease
```

Figure 4.47: *Output example of updating local package index*

- Install a new package called **`vsftpd`** from the software repository, as shown in *figure 4.48*:

```
root@ubuntu:~# apt-get install -y vsftpd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ssl-cert
The following NEW packages will be installed:
  ssl-cert vsftpd
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 140 kB of archives.
After this operation, 391 kB of additional disk space will be used.
Get:1 http://es.archive.ubuntu.com/ubuntu jammy/main amd64 ssl-cert all 1.1.2 [17.4 kB]
Get:2 http://es.archive.ubuntu.com/ubuntu jammy/main amd64 vsftpd amd64 3.0.5-0ubuntu1 [123 k
```

```
B]
Fetched 140 kB in 1s (98.1 kB/s)
Preconfiguring packages ...
Selecting previously unselected package ssl-cert.
(Reading database ... 108814 files and directories currently installed.)
Preparing to unpack .../ssl-cert_1.1.2_all.deb ...
Unpacking ssl-cert (1.1.2) ...
Selecting previously unselected package vsftpd.
Preparing to unpack .../vsftpd_3.0.5-0ubuntu1_amd64.deb ...
Unpacking vsftpd (3.0.5-0ubuntu1) ...
Setting up ssl-cert (1.1.2) ...
Setting up vsftpd (3.0.5-0ubuntu1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/vsftpd.service → /lib/systemd/sys
tem/vsftpd.service.
```

Figure 4.48: *Output example of a package installation*

- Remove the package **`vsftpd`** previously installed, as illustrated in *figure 4.49*:

```
root@ubuntu:~# apt-get remove -y vsftpd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  ssl-cert
Use 'apt autoremove' to remove it.
The following packages will be REMOVED:
  vsftpd
0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.
After this operation, 326 kB disk space will be freed.
(Reading database ... 108880 files and directories currently installed.)
Removing vsftpd (3.0.5-0ubuntu1) ...
Processing triggers for man-db (2.10.2-1) ...
```

Figure 4.49: *Output example of a package removal*

- Auto-remove the packages what at not required anymore as a dependency, as shown in *figure 4.50*:

```
root@ubuntu:~# apt-get autoremove
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages will be REMOVED:
  ssl-cert
0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.
After this operation, 65.5 kB disk space will be freed.
Do you want to continue? [Y/n]
(Reading database ... 108828 files and directories currently installed.)
```

```
Removing ssl-cert (1.1.2) ...
Processing triggers for man-db (2.10.2-1) ...
```

Figure 4.50: *Output example auto removing not anymore needed packages.*

The command **`apt-cache`** is used to query the local index of packages retrieved from the software repositories. The common actions are shown in *table 4.24*:

Action	Description
show	Show information about one specific package.
search	Search in the index for the specific pattern.
depends	Lists the dependencies needed for the specified package.

Table 4.24: *Actions for the command apt-cache*

The following figures and examples of the common uses for **`apt-cache`** are provided:

- Get information about the package **`firefox`**, as shown in *figure 4.51*:

```
root@ubuntu:~# apt-cache show firefox
Package: firefox
Architecture: amd64
Version: 1:1snap1-0ubuntu2
Priority: optional
Section: web
Origin: Ubuntu
Maintainer: Ubuntu Mozilla Team <ubuntu-mozillateam@lists.ubuntu.com>
Bugs: https://bugs.launchpad.net/ubuntu/+filebug
Installed-Size: 255
Provides: gnome-www-browser, iceweasel, www-browser, x-www-browser
Pre-Depends: debconf, snapd
Depends: debconf (>= 0.5) | debconf-2.0
Breaks: firefox-dbg (<< 1:1snap1), firefox-dev (<< 1:1snap1), firefox-geckodriver (<< 1:1snap
1), firefox-mozsymbols (<< 1:1snap1)
Replaces: firefox-dbg (<< 1:1snap1), firefox-dev (<< 1:1snap1), firefox-geckodriver (<< 1:1sn
ap1), firefox-mozsymbols (<< 1:1snap1)
Filename: pool/main/f/firefox/firefox_1snap1-0ubuntu2_amd64.deb
Size: 72292
MD5sum: 4cfb8522595425213edbd3440dfaab6c
SHA1: ec27ebd8fca2d918add69c94f1163b593fd4fb66
SHA256: ddccd2ef5cc7291364ab4998e57d684aac993488ae561bb67ed31ddb617659cc
SHA512: 35cb18af47666668b77357132f7786946fe2f90abba2fc5abd55f73124de672645a07ac97a6e12ca66b77
d892f07356f3a69d8c321b5b657b39419b146681f2a
Description-en: Transitional package - firefox -> firefox snap
 This is a transitional dummy package. It can safely be removed.
 .
 firefox is now replaced by the firefox snap.
Description-md5: 28593f0f24b1284477e30b88c6717ba4
Task: xubuntu-live, ubuntukylin-desktop
```

Figure 4.51: *Output example obtaining information about a package.*

- Search for packages with pattern **`^ansible`** (name or description starts with word *ansible*), as shown in *figure 4.52*:

```
root@ubuntu:~# apt-cache search ^ansible
ansible - Configuration management, deployment, and task execution system
ansible-core - Configuration management, deployment, and task execution system
ansible-lint - lint tool for Ansible playbooks
ansible-mitogen - Fast connection strategy for Ansible
shade-inventory - Ansible inventory script for OpenStack clouds
```

Figure 4.52: *Output example searching for a pattern.*

- Show the dependencies for the package **`python3-urllib`**, as shown in *figure 4.53*:

```
root@ubuntu:~# apt-cache depends python3-urllib3
python3-urllib3
  Depends: <python3:any>
    python3
  Depends: python3-six
  Recommends: ca-certificates
  Suggests: python3-cryptography
  Suggests: python3-idna
  Suggests: python3-openssl
  Suggests: python3-socks
```

Figure 4.53: *Output example for dependencies of a package.*

The command **`apt`** is a frontend for the commands **`apt-get`** and **`apt-cache`**. The actions available for the command are shown in *table 4.25*:

Action	Equivalent	Action	Equivalent
update	*apt-get update*	reinstall	*apt-get install*
upgrade	*apt-get upgrade*	remove	*apt-get remove*
full-upgrade	*apt-get dist-upgrade*	purge	*apt-get purge*
install	*apt-get install*	autoremove	*apt-get autoremove*
search	*apt-cache search*	show	*apt-cache show*

Table 4.25: *Actions and Syntax for rpm*

Introduction to services

A Linux service (or sometimes called **daemon**) is an application or set of applications that are running in the background performing some tasks or waiting for some event. Some services can execute some applications, and when it finishes, the service is stopped. An example is a service to configure the network in the system. Other services are running constantly, waiting to offer some resource; for example, an *SSH* service is waiting for users to connect and offer the possibility to login to the server.

Most Linux distributions use the service manager called **systemd**, which is a replacement for the historical and popular **SysV init** daemon. When a Linux distribution is booted, the first process executed is **systemd**, and it will be responsible for starting the rest of the services enabled in the system to be started on boot.

Systemd is a full suite of applications aiming to unify the service configuration and the behavior across different *Linux distributions*. The main command to manage services is **systemctl**, and the popular actions are described in *table 4.26*:

Action	Description
list-units	List the *units* (services, mount points, devices, and sockets) available in the system.
status	Check the status of the specified *unit(s)*.
show	Show the properties of the specified *unit(s)*.
start	Start the *unit(s)* specified.
stop	Stop the *unit(s)* specified.
reload	The *unit(s)* specified will reload the configuration.
restart	Stop and start the *unit(s)* specified.
enable	Enable the *unit(s)* specified to be started when the system boots.
disable	Disable the *unit(s)* specified to do not be started when the system boots.

Table 4.26: *Actions for system*

In the following figures, some examples of the actions for the command *systemctl* listed are shown:

- Check the status of the service named **cron**, as shown in *figure 4.54*:

```
root@ubuntu:~# systemctl status cron
● cron.service - Regular background program processing daemon
     Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor preset: enabled)
     Active: active (running) since Sun 2022-07-24 13:29:20 UTC; 58min ago
       Docs: man:cron(8)
   Main PID: 659 (cron)
      Tasks: 1 (limit: 4579)
     Memory: 424.0K
        CPU: 14ms
     CGroup: /system.slice/cron.service
             └─659 /usr/sbin/cron -f -P

Jul 24 13:29:20 ubuntu.example.com systemd[1]: Started Regular background program processing
Jul 24 13:29:20 ubuntu.example.com cron[659]: (CRON) INFO (pidfile fd = 3)
Jul 24 13:29:20 ubuntu.example.com cron[659]: (CRON) INFO (Running @reboot jobs)
```

```
Jul 24 14:17:01 ubuntu.example.com CRON[2056]: pam_unix(cron:session): session opened for us
Jul 24 14:17:01 ubuntu.example.com CRON[2056]: pam_unix(cron:session): session closed for us
```

Figure 4.54: *Output example about the status of one service.*

- Disable a service what is currently enabled, as shown in *figure 4.55*:

```
root@ubuntu:~# systemctl disable cron
Synchronizing state of cron.service with SysV service script with /lib/systemd/systemd-sysv-i
nstall.
Executing: /lib/systemd/systemd-sysv-install disable cron
Removed /etc/systemd/system/multi-user.target.wants/cron.service.
```

Figure 4.55: *Output example disabling a service.*

- Enable a disabled service, as illustrated in *figure 4.56*:

```
root@ubuntu:~# systemctl enable cron
Synchronizing state of cron.service with SysV service script with /lib/systemd/systemd-sysv-i
nstall.
Executing: /lib/systemd/systemd-sysv-install enable cron
Created symlink /etc/systemd/system/multi-user.target.wants/cron.service → /lib/systemd/syste
m/cron.service.
```

Figure 4.56: *Output example enabling a service.*

Conclusion

Administrating users and groups is one of the main tasks for Linux administrators and regular users administrating a system. Linux provides easy-to-use tools to perform all the user and group-related tasks, such as create, query, modify, or delete.

Other important knowledge needed when working with a Linux distribution is to know how to install and administrate software in the system. Understanding the package format and software repositories and the tools available are needed skills to configure a system, install new software and then install new services on it. Services are applications usually running in the background, performing some tasks or offering some service to regular users.

Key facts

- Users and groups information are stored in plain files.

- Passwords are encrypted and saved in files only accessible by administrators.
- Linux Distributions provide several tools to operate with users and groups.
- Package Format for distributions helps in the installation process of new software and updates the system.
- Software repositories keep software and libraries to be accessible from the system and the system tools to install the software.
- Services are processes running in the background performing some operating or waiting to offer functionality.

Questions

1. What option for command useradd is creating the directory for the user?
    a. -h, --create-home
    b. -m, --create-home
    c. -d, --create-directory
2. Which of the following commands are not used to operate with RPM packages?
    a. rpm
    b. dpkg
    c. dnf
3. With the command rpm, which option ensures to install or update the local file specified?
    a. -F, --freshen
    b. -U, --upgrade
    c. -i, --install
4. What command(s) can be used to search packages in Software Repositories for DEB packages?
    a. apt
    b. apt-get
    c. apt-cache
5. What action for command systemctl is used to ensure a service is starting on boot?
    a. start
    b. enable

c. activate

Answers

1. b
2. b
3. b
4. a and c
5. b

CHAPTER 5

Managing Files, Directories, and Processes

Introduction

This chapter will cover how to work with files, directories, and processes. Everything in Linux is reflected with files, such as regular files and documents, or devices, such as a hard disk, a USB device, or inter-process and network communications. Process information is also going to be accessible through files.

The first part of the chapter will describe the directory structure in Linux distributions to better explain how the hierarchy of the directories and the functionality of the most important ones.

The second part of the chapter aims to describe how permissions work in Linux for files, directories, and applications. Commands to work with files and directories and an operation such as create, remove, or filter the content are going to be shown with different examples. Commands to operate with permissions are going to be described as well.

The third part of the chapter will describe how to work with popular editors, which is required knowledge to operate a Linux server. This includes some examples and tricks on how to use a basic editor and how to use a more advanced editor. In this part, file managers are going to be described as being able to navigate between directories in an interactive way.

The last part of the chapter will show how the processes in a Linux distribution work and how it is possible to set permissions to allow or not to allow execution, depending on the user or group who is trying to run it. Commands to change the priority of the processes are going to be described with different examples.

Structure

In this chapter, we will discuss the following topics:

- Linux directory structure
- Permissions

- Access to files and understanding files on Linux
- Special characters
- Regular expressions
- File editors and file managers
- Processes management
- Operate with processes priorities

Linux directory structure

Linux distributions have a common layout directory structure, where each directory has a specific purpose. The main directory is called *root* and is referred to with a slash (/). The rest of the directories appear under this directory. *Figure 5.1* shows the standard **Filesystem Hierarchy Standard** (**FHS**):

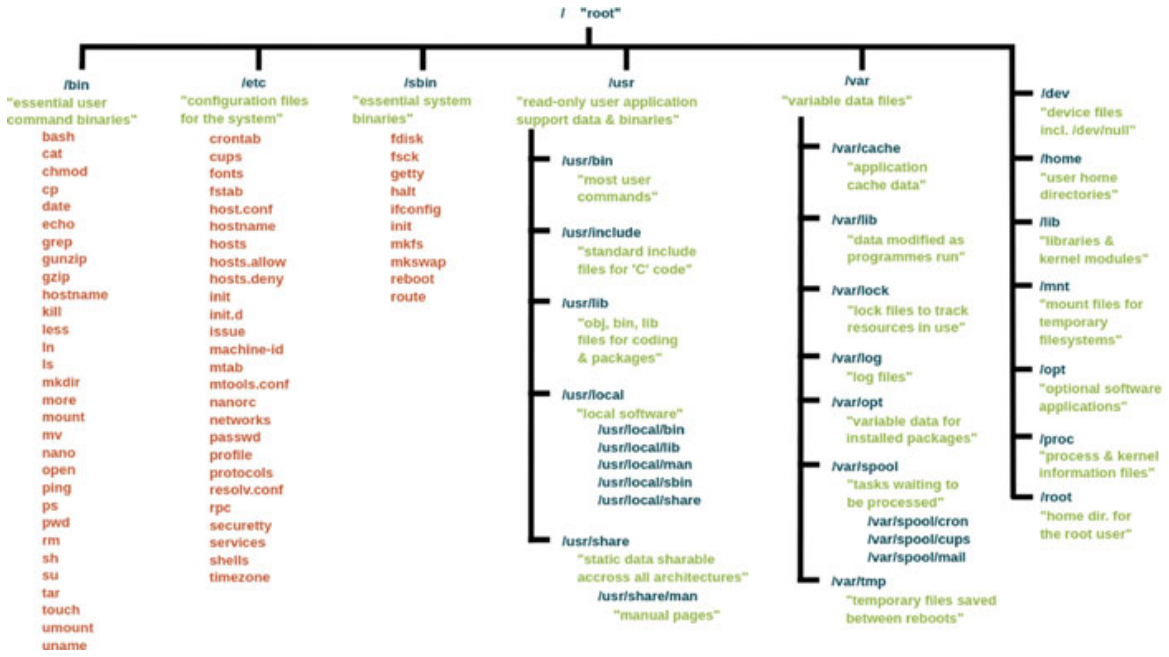


Figure 5.1: *Standard Linux Filesystem hierarchy. Source: Wikipedia.*

As is possible to observe in the previous image, the directories start with a slash (/) and, afterward, the name of the directory, for example,*/bin/*. If a directory is part of another directory, the separation is using a slash (/) again. For example, the directory **`/usr/share/`** indicates the directory **`share/`** is under the directory **`usr/`**, which is also under the *root* directory (/). Directories are represented by ending with a slash (/). It is possible to refer to the directory without specifying the last slash, but it is usually a good practice so that there is no confusion between file names.

Modern distributions are applying some changes to the standard *FHS* to ease the directory structure and also adding new directories for new features:

- Directory **`/bin/`** is a link to directory **`/usr/bin/`** and does not contain different content anymore.
- Directory **`/lib/`** is a link to directory **`/usr/lib/`**.
- Directories **`/lib32/`** (for 32-bit libraries) and **`/lib64/`** (for 64-bit libraries) are linked to directories **`/usr/lib32/`** and **`/usr/lib64/`**.
- Directory **`/sbin/`** is a link to directory **`/usr/sbin/`**
- A new directory called **`/sys/`** contains information about devices, drivers, and kernel features.
- A new directory called **`/run/`** contains a temporary filesystem associated with the memory available where run-time available data are stored. This directory is cleared during the boot process.

Directories storing applications

There are two main directories where applications are stored when we install software using a package (*deb* or *rpm*). These directories are as follows:

- **`/usr/bin/`** (linked to **`/bin/`**): It contains regular software which any user in the system can execute them. Some commands were reviewed previously, such as **`mkdir`** or **`groups`** are stored here.
- **`/usr/sbin/`** (linked to **`/sbin/`**): It contains software used by administrators in the system. Some commands were reviewed previously, such as **`useradd`** or **`groupadd`**.

Another directory that can contain software installed without using a Linux distribution package is **`/usr/local/bin/`**.

Directories storing user files

Regular users would have their own directory inside the directory */home/*. It is accessible only by the user or by an administrator and is isolated from the rest of the users. For example, the user *agonzalez* will have as default home directory **`/home/agonzalez/`**. An exception to the **`/home/`** directory is for the home directory for the *root* user. This administrator user has as home directory labeled **`/root/`**. It is important not to confuse between the **`root directory (/)`** and the root's home directory *(***`/root/`**).

Directories storing configurations

The main directory to store configuration globally in the system is */etc/,* which contains configuration for system tools, services, and stores as described earlier. This directory contains configuration files and subdirectories containing configurations. Some file configuration examples are shown in table 5.1:

File(s)	Description
passwd, group	Contains a user list and group information
resolv.conf	Contains DNS resolution configuration
hosts	Contains a map list between DNS names and IPs
fstab	Contains the filesystems and mounpoint configuration
sudoers	Contain the definition of administrator commands, which regular users can execute.

Table 5.1: *Files examples inside /etc/ directory*

Some directories configuration examples are shown in table 5.2:

Directory	Contains
default/	Some default configurations for applications
systemd/	Definition and configuration for *systemd*
ssl/	Public and private SSL certificates and configuration.
security/	Security configuration related to system and users.

Table 5.2: *Directories examples inside /etc/ directory*

Some applications executed by regular users use a directory named `.config` (directories starting with a `dot [.]` in Linux are hidden) in their home directory to store configuration for the different applications. Other applications use their own directory to store the configuration.

- Chromium browser will store the configuration and temporary files under `/home/username/.config/chromium/`
- SSH tools will store the configuration under `/home/username/.ssh/` directory.

Directories storing libraries

Linux applications use libraries to be able to operate and perform different operations. These libraries can be available already in the system or can be

installed libraries, which are automatically installed as dependencies when new software is installed in the system. Some libraries are a virtual library provided by the Linux Kernel. The directories keeping libraries are as follows:

- **`/usr/lib/`**, **`/usr/lib64/`**, and **`/usr/lib32/`** (directories linked to **`/lib/`**, **`/lib64/`**, and **`/lib32/`**: It is globally accessible for all the applications in the system.
- **`/usr/local/lib/`**, **`/usr/local/lib64/`**, **and** **`/usr/local/lib32/`**: Contains libraries for the software installed that does not use Linux distribution packages.

It is important to mention that the purpose of the directory */var/lib* is not to contain libraries but some variable state information for applications.

Directories storing variable data

The top directory containing data, which is not static, is **`/var/`**. The standard directories inside are shown in *table 5.3*:

Directory	Contains
backups/	The copies of Backup applications are stored here
cache/	Temporary files for different applications.
crash/	If one application crash generates a file in this directory.
lib/	Variable state information for applications.
local/	Variable data for applications installed outside of the regular Linux distribution packages.
lock/	Lock files indicating the application is using some resource.
log/	Log files and directories inside.
mail/	Mail local files for users
opt/	Optional files
run/	A link to */run/* directory
spool/	Contain files which should be treated later, such as print jobs.
tmp/	Temporary directory

Table 5.3: *Directories inside directory /var/*

Directories storing data for users

As observed in *figure 5.1*, the directory **`/usr/`** contains data and applications oriented to users. Directories **`/usr/lib/`**, **`/usr/lib32/`**, **`/usr/lib64/`**, **`/usr/bin/`**, and **`/usr/sbin/`** were described previously. Other important directories under **`/usr/`** are shown in *table 5.4*:

Directory	Contains
include/	Files required for software compilation from source code.
local/	The same structure as */usr* but for software not installed using Linux distribution packages.
libexec/	Internal applications are executable by other programs but not for regular users.
src/	Source code for applications or for the Linux kernel.
share/	Data for applications that are independent of system architecture. Data such as documentation, fonts, public global certificates, icons, applications plugins, and so on.

Table 5.4: *Other directories inside directory /usr/*

Directories storing system data information and boot files

As everything in Linux is represented as a file, Linux distributions have some special directories containing information about devices, hardware, and process information. The kernel core and the boot file containing drivers are stored in a special directory too. These directories are shown in *table 5.5*:

Directory	Contains
/boot/	Kernel and booting files, including configuration files for the boot loader *(*for example, *grub*).
/dev/	List of files referencing available devices in the system, such as disk devices (*/dev/sda* for a primary disk) or virtual devices like */dev/random* to generate random data. Standard input, output, and error are reflected here as */dev/stdin*, */dev/stdout,* and */dev/stderr*.
/proc/	Process and kernel information. Each process has an associated directory inside with information about the execution. Other files contain information about the system, such as */proc/cpuinfo,* having details about the system CPUs.
/sys/	Devices, drivers, and kernel features. For example, the subdirectory */sys/class/net/* contains information about the network devices.

Table 5.5: *Directories storing system data information and boot files*

When a process is running in the system, a subdirectory is created inside the directory `/proc/` with the process id assigned. For example, the process with id 687 will have assigned a dedicated directory `/proc/687/`. Inside the directory, there are files and directories with useful information about the process. *Table 5.6* shows the most useful ones:

Directory	Description
/proc/PID/cmdline	Command line arguments.
/proc/PID/cpu	Current and last CPU in which it was executed.
/proc/PID/cwd	Link to the current working directory.
/proc/PID/environ	Values of environment variables.
/proc/PID/exe	Link to the executable of this process.
/proc/PID/fd	Directory, which contains all file descriptors.
/proc/PID/maps	Memory maps to executables and library files.
/proc/PID/mem	Memory is held by this process.
/proc/PID/root	Link to the root directory of this process.
/proc/PID/stat	Process status.
/proc/PID/statm	Process memory status information.
/proc/PID/status	Process status in human-readable form.

Table 5.6: *Directories and files related to the process inside /proc/ directory.*

Permissions

On a Linux system, each file and directory defines access rights for the user owner, for the members part of the group owner, and for the rest of the users not included in the group. The rights available are: *read*, *write,* and *execution.* The execution permission for directories has that meaning if the user is able to read the content inside.

The rights can be represented with a single character or with a numeric digit. *Table 5.7* shows the available rights and the representations:

Permission	Symbol representation	Octal representation
read	r	4
write	w	2
execution	x	1

Table 5.7: *Permission presentation in symbol and octal*

When assigning permissions in symbol representation, the owner is represented with the character *o*, the group owner with the character *g,* and other users with the letter *o*. The permission *u=rwx, g=rw, o=r* indicates:

- The user owner is able to read (r), write (w), and execute (x) the file.
- The group members associated to the file would be able to read and write the file but not execute it.
- Other users can read the file but not write or execute it.

The permission u=rwx, g=rx, o= indicates:

- The user owner is able to access the directory (x), able to read the content (r), and create new files (w).
- The group members associated with the directory would be able to access to a directory (x) and able to read the content of the directory (r) but do not write new files.
- Other members are not going to be able to access the directory as they don't have execution permission (x).

The octal representation is the sum of the permissions desired to assign. For example, for permission, *read* and *write* is indicated with the number 6 (4+2). The first digit is for the user owner, the second for the group owner, and the last one for others. The representations shown before as character representations and other examples are illustrated in *table 5.8*:

Symbol representation	Conversion	Octal representation
u=rwx,g=rw,o=r	u=4+2+1,g=4+2,o=4	764
u=rwx,g=rx,o=	u=4+2+1,g=4+1,o=0	750
u=rwx,g=rwx,o=rx	u=4+2+1,g=4+2+1,o=4+1	775
u=rw,g=rw,o=r	u=4+2,g=4+2,o=4	664

Table 5.8: *Permission conversion examples*

The command *ls* with the option *-l* will show the permission for files and directories. The first character would indicate the type of file that it is:

- Character—indicates that it is a regular file.
- Letter *d* indicates that it is a directory.
- Letter *l* indicates that it is a symbolic link.

- Letter *c* indicates that it is a character device
- Letter *p* indicates that it is a pipe
- Letter *s* indicates that it is a socket
- Letter *d* indicates that it is a block device.

After the file type, the next three characters indicate the permissions for the user owner, then the next three for the group owner, and the last three for other users. *Figure 5.2* shows some examples of different permissions and file types:

```
root@ubuntu:~# ls -ld /etc/{,passwd,shadow} /dev/{vda,console} /run/snapd.socket
crw--w----  1 root tty      5, 1 Jul 25 17:40 /dev/console
brw-rw----  1 root disk   252, 0 Jul 25 17:40 /dev/vda
drwxr-xr-x 97 root root     4096 Jul 24 15:31 /etc/
-rw-r--r--  1 root root     1838 Jul 24 13:32 /etc/passwd
-rw-r-----  1 root shadow   1198 Jul 24 13:32 /etc/shadow
srw-rw-rw-  1 root root        0 Jul 25 17:40 /run/snapd.socket
```

Figure 5.2: *Different types of files and permissions associated.*

The third column indicates the user owner, and the fourth indicates the group owner. For example, for the file **`/etc/shadow`** in the figure, the user *root* has read and write access to the file, and the members in the group *shadow* have read-only access. The rest of the users have no access to the file for security reasons, as this file contains encrypted passwords.

Another way to see the permissions for a file or directory is by using the command *stat*, which will show more related information, such as when the file was modified, changed (permissions or owners), created, or accessed. *Figure 5.3* shows an example using the file **`/etc/passwd`**:

```
root@ubuntu:~# stat /etc/passwd
  File: /etc/passwd
  Size: 1838            Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 397042      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2022-07-30 09:55:03.691999361 +0000
Modify: 2022-07-24 13:32:07.456812114 +0000
Change: 2022-07-24 13:32:07.456812114 +0000
 Birth: 2022-07-24 13:32:07.456812114 +0000
```

Figure 5.3: *Output example for command stat.*

Default permission when a file or directory depends on the system or user *file-creation mask* value. The default value is defined in the file **`/etc/login.defs`** and usually is defined with value 002. This value would be rested on the full permission for files (666) and for directories (777), causing a regular file by default would be created with permission 664 and directories 775. The *bash*

utility called *umask* allows us to see and modify the *file-creation mask* value. *Figure 5.4* shows the default *umask* value and the permissions when a file and a directory are created:

```
agonzalez@ubuntu:~$ umask
0002
agonzalez@ubuntu:~$ touch newfile
agonzalez@ubuntu:~$ mkdir newdir
agonzalez@ubuntu:~$ ls -ld newfile newdir
drwxrwxr-x 2 agonzalez agonzalez 4096 Aug  7 13:36 newdir
-rw-rw-r-- 1 agonzalez agonzalez    0 Aug  7 13:36 newfile
```

Figure 5.4: *Create a file and directory with default file-masking value*

Figure 5.5 shows how to modify the default *file-masking value* using **`umask`** and how it affects permissions for new files and directories:

```
agonzalez@ubuntu:~$ umask 0077
agonzalez@ubuntu:~$ touch newfile2
agonzalez@ubuntu:~$ mkdir newdir2
agonzalez@ubuntu:~$ ls -ld newfile2 newdir2
drwx------ 2 agonzalez agonzalez 4096 Aug  7 13:40 newdir2
-rw------- 1 agonzalez agonzalez    0 Aug  7 13:40 newfile2
```

Figure 5.5: *Modify file-masking value and create directory and file.*

In *figures 5.3* and *5.4*, four digits appear, as Linux also provides some special permissions named *set user identity* (*setuid*), *set group identity* (*setgid),* and *sticky bit*. *Table 5.9* shows the representation and the meaning of these permissions:

Permission	Symbol	Numeric	Description
setuid	*s (in user execution field)*	4	The application will be executed always as the user who owns the application.
setgid	*s (in group execution field)*	2	If it is an application, it will be executed as the group who owns the application. If it is a directory, files created inside will have the group associated with the directory.
sticky bit	t *(in other execution field)*	1	This permission restricts file deletion on directories.

Table 5.9: *Permission presentation in symbol and octal*

The command *passwd* is an example of an application using the special permission *setuid*. The command can be executed by any user to modify their own password, but the execution will be performed in the name of the *root* user to

be able to modify **`/etc/passwd`** and **`/etc/shadow`** files. *Figure 5.6* shows how to use command *stat* to show the octal and the character representation for a file:

```
root@ubuntu:~# stat -c "%a %A" /usr/bin/passwd
4755 -rwsr-xr-x
```

Figure 5.6: *Output example for command stat*

On Debian and derivate distributions, the command change has the special permission **`setgid`**. A regular user running this command will use the group owner associated with the file **`/usr/bin/chage`**, *and with this group, user would be able to access the file* **`/etc/shadow`**. *Figure 5.7* shows the permissions for the *chage* command:

```
root@ubuntu:~# stat -c "%a %A" /usr/bin/chage
2755 -rwxr-sr-x
```

Figure 5.7: *Output example for command stat*

Figure 5.8 shows the difference between creating a file inside a directory with the *setgid* special permission associated and in a regular directory. The file created in the directory with the *setgid* permission will keep the group of the directory where it is created. Default behavior without permission means using the primary group for the user.

```
agonzalez@ubuntu:~$ ls -ld specialdir/ regulardir/
drwxr-xr-x 2 agonzalez adm 4096 Jul 30 19:48 regulardir/
drwxr-sr-x 2 agonzalez adm 4096 Jul 30 19:48 specialdir/
agonzalez@ubuntu:~$ touch regulardir/file
agonzalez@ubuntu:~$ ls -l regulardir/file
-rw-rw-r-- 1 agonzalez agonzalez 0 Jul 30 19:53 regulardir/file
agonzalez@ubuntu:~$ touch specialdir/file
agonzalez@ubuntu:~$ ls -l specialdir/file
-rw-rw-r-- 1 agonzalez adm 0 Jul 30 19:53 specialdir/file
```

Figure 5.8: *Output example creating files in the regular directory and in a directory with special permission*

The most common example of the *sticky bit* is the temporary directory **`/tmp/`**. All the users are able to write in this directory, but users can remove their own files and not other user's files (except administrators, who can remove any file). *Figure 5.9* shows how the **`/tmp/`** directory has the *sticky permission* (*t*), and even the file is owned by the user's group. Here, it is not possible to delete because the user owner is different:

```
agonzalez@ubuntu:~$ ls -ld /tmp/
drwxrwxrwt 12 root root 4096 Jul 30 19:58 /tmp/
agonzalez@ubuntu:~$ ls -l /tmp/deleteme
-rw-rw-r-- 1 testuser agonzalez 0 Jul 30 19:57 /tmp/deleteme
agonzalez@ubuntu:~$ rm /tmp/deleteme
rm: cannot remove '/tmp/deleteme': Operation not permitted
```

Figure 5.9: *Output example trying to remove the file in a directory with special permission*

Figure 5.10 demonstrates how the same file can be deleted if it is a directory without the *sticky bit* set:

```
agonzalez@ubuntu:~$ ls -l deleteme
-rw-rw-r-- 1 testuser agonzalez 0 Jul 30 20:03 deleteme
agonzalez@ubuntu:~$ rm -v deleteme
removed 'deleteme'
```

Figure 5.10: *Deleting a file in a directory without special permission*

Due to security reasons, the files in the system owned by the user *root* or group *root* using special permissions *setuid* or *setgid* need to be limited to specific system tools. Any other software using those permissions is a potential risk to be used to escalate permissions from a regular user. Using the command *find,* it is possible to find the files with the *setguid* permissions (4xyz) and the *setgid* permissions (2xyz). *Figure 5.11* shows the output of searching for files with special permissions assigned in the system:

```
root@ubuntu:~# find /usr/bin/ -perm /4000
/usr/bin/su
/usr/bin/newgrp
/usr/bin/sudo
/usr/bin/chfn
/usr/bin/umount
/usr/bin/chsh
/usr/bin/gpasswd
/usr/bin/mount
/usr/bin/pkexec
/usr/bin/passwd
/usr/bin/fusermount3
root@ubuntu:~# find /usr/bin/ -perm /2000
/usr/bin/write.ul
/usr/bin/wall
/usr/bin/crontab
/usr/bin/expiry
/usr/bin/ssh-agent
/usr/bin/chage
```

Figure 5.11: *Output example listing files with special permissions.*

Access to files and understanding files on Linux

As described previously, in a Linux system, everything is a file. To understand how to access files to read content, manipulate content, and remove files, it is required to administrate and operate the Linux server. This section will focus on how to assign owner, user, and group to files and directories, how to set permissions, and how to access the files and filter or format the content.

Commands *chown* and *chgrp*

The main command to change the user owner and the group owner is *chown*. Administrator users can use this command to modify both the owners for files and

directories. In contrast, a regular user can only modify the group of files and directories only when the target group is one of the groups they belong to. *Table 5.10* shows all the possible syntax options:

Syntax	Description
chown user:group file(s)/dir(s)	Change user and group owners for the specified files and/or directories.
chown user file(s)/dir(s)	Change the user owner for the specified files and/or directories.
chown user: file(s)/dir(s)	Same as the previous syntax.
chown :group file(s)/dir(s)	Change group owner for the specified files and/or directories

Table 5.10: *Command chown possible syntax*

The user and group can be specified using the name or the **`group id`**. Old versions of command *chown* were using dot (.) as a separator instead of the colon (:).It is still a valid separator but not recommended to use anymore. The Argument can be a file, directory, or a list of files and/or directories separated by space. The popular options for command *chown* are shown in *table 5.11*:

Option	Description
-c, --changes	Shows the actions done but only the changes.
-v, --verbose	Shows all the actions done as well as the skipped ones.
--reference=SOURCE	Copy the owner and group owners from a specified source
-R, --recursive	Change all the permissions of all files and subdirectories for the specified directory.

Table 5.11: *Popular options for command chown*

Figures 5.12, *5.13*, and *5.14* feature examples using the command **`chown`**:

- Try to change the owner of a file being a regular user:

```
agonzalez@ubuntu:~$ ls -l example.txt
-rw-rw-r-- 1 agonzalez agonzalez 37 Jul  3 19:27 example.txt
agonzalez@ubuntu:~$ chown testuser example.txt
chown: changing ownership of 'example.txt': Operation not permitted
```

Figure 5.12: *Output example using command chown*

- Change the group owner for a file being a regular user:

```
agonzalez@ubuntu:~$ ls -l example.txt
-rw-rw-r-- 1 agonzalez agonzalez 37 Jul  3 19:27 example.txt
agonzalez@ubuntu:~$ groups
agonzalez adm cdrom sudo dip plugdev lxd
agonzalez@ubuntu:~$ chown :root example.txt
chown: changing group of 'example.txt': Operation not permitted
agonzalez@ubuntu:~$ chown -v :sudo example.txt
changed ownership of 'example.txt' from agonzalez:agonzalez to :sudo
```

Figure 5.13: *Output examples using command chown*

- Change recursively the user and group owner for a directory as administrator.

```
root@ubuntu:/tmp# chown -R -v agonzalez:agonzalez snap/
changed ownership of 'snap/lxd/22911' from root:root to agonzalez:agonzalez
changed ownership of 'snap/lxd/22923' from root:root to agonzalez:agonzalez
changed ownership of 'snap/lxd/common' from root:root to agonzalez:agonzalez
changed ownership of 'snap/lxd/current' from root:root to agonzalez:agonzalez
changed ownership of 'snap/lxd' from root:root to agonzalez:agonzalez
changed ownership of 'snap/' from root:root to agonzalez:agonzalez
```

Figure 5.14: *Output examples using command chown*

The command **`chgrp`** is used to change only the group owner for files and directories. The syntax is **`chgrp [options]group file(s)/dir(s)`**. Options are the same as for **`chown`** command.

Command *chmod*

This command is used to change the permission for files and directories. It allows us to specify the permissions using the symbolic or the octal representation. As described previously about permissions, the symbolic representation has the following syntax: **`u=permission,g=permission,o=permission`**, and the octal representation can contain three digits or four digits in case there is a need to specify special permission.

The command *chmod* can be used to replace current permissions using the character equal (**=**) or manipulate current permission by adding more rights using the character (**+**) or removing permissions using the character minus (**–**). The command **`chmod`** allows to add permission using the character plus (*+)* or delete permissions (–) instead of the symbol equal (=). As well, **`chmod`** allows to specify only the permission to be modified and not all the components. And it is possible to use the letter "*a"* to set, add or remove permission for all the components.

The options for this command are the same as for command **`chown`** (refer to *table 5.11*). *Figures 5.15*, *5.16*, and *5.17* show different examples using the command

chmod:

- Specify permission for the user to read and write, the group only to read, and no permission: to others.

```
agonzalez@ubuntu:~$ chmod -v u=rw,g=r,o= example.txt
mode of 'example.txt' changed from 0664 (rw-rw-r--) to 0640 (rw-r-----)
```

Figure 5.15: *Output examples using the command chmod*

- Remove read permission for group owner:

```
agonzalez@ubuntu:~$ chmod -v g-r example.txt
mode of 'example.txt' changed from 0640 (rw-r-----) to 0600 (rw-------)
```

Figure 5.16: *Output examples using the command chmod*

- Specify the permission to user and group to read and write, and read for others using octal representation:

```
agonzalez@ubuntu:~$ chmod -v 644 example.txt
mode of 'example.txt' changed from 0600 (rw-------) to 0644 (rw-r--r--)
```

Figure 5.17: *Output examples using the command chmod*

To understand the *execution* permission for files and directories, *figures 5.18* and *5.19* feature different examples:

- Add execution permission to a file to be able to be executed:

```
agonzalez@ubuntu:~$ stat -c "%a %A" app1
664 -rw-rw-r--
agonzalez@ubuntu:~$ ./app1
-bash: ./app1: Permission denied
agonzalez@ubuntu:~$ chmod -v u+x app1
mode of 'app1' changed from 0664 (rw-rw-r--) to 0764 (rwxrw-r--)
agonzalez@ubuntu:~$ ./app1
Example application
```

Figure 5.18: *Example using chmod to set execution permission.*

- Remove execution permission to a directory to not allow access to files inside:

```
agonzalez@ubuntu:~$ ls -ld regulardir/ regulardir/*
drwxr-xr-x 2 agonzalez adm       4096 Jul 30 19:53 regulardir/
-rw-rw-r-- 1 agonzalez agonzalez    0 Jul 30 19:53 regulardir/file
agonzalez@ubuntu:~$ chmod -v a-x regulardir/
mode of 'regulardir/' changed from 0755 (rwxr-xr-x) to 0644 (rw-r--r--)
agonzalez@ubuntu:~$ ls -ld regulardir/ regulardir/*
ls: cannot access 'regulardir/file': Permission denied
drw-r--r-- 2 agonzalez adm 4096 Jul 30 19:53 regulardir/
```

Figure 5.19: *Example using chmod to remove execution permission.*

Command *cat*

This command is used to visualize the content of one of several files. That content is visualized in the standard output, with the possibility, as described previously, to redirect to another file or to use *pipes* to another command. The popular options for this command are shown in *table 5.12*:

Option	Description
-b, --number-nonblank	Number non-blank lines
-n, --number	Number non-blank and blank lines.

Table 5.12: *Popular options for command cat*

Figure 5.20 shows how to visualize the content of one file and how to number the lines:

```
agonzalez@ubuntu:~$ cat /etc/legal

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

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

agonzalez@ubuntu:~$ cat -n /etc/legal
     1
     2  The programs included with the Ubuntu system are free software;
     3  the exact distribution terms for each program are described in the
     4  individual files in /usr/share/doc/*/copyright.
     5
     6  Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
     7  applicable law.
     8
```

Figure 5.20: *Example using the command cat*

Commands *head* and *tail*

The command *head* is used to visualize the beginning of files, and the command *tail* is used to visualize the last part of files. By default, it shows the first 10 or the last 10 lines. It is possible to specify a different number using *–X* (for example, –20) or *–n X* (for example *–n 20*).

The command *last* has a popular and useful option to continue tracking the file for new appended data to the file tracked. This option is used to track log files to check the status in *real-time*.

Figures 5.21 and *5.22* feature different examples:

- Visualize the first five lines of the file **`/etc/passwd`**:

```
root@ubuntu:~# head -n5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
```

Figure 5.21: *Example using command head*

- Visualize the last five lines of the file **`/var/log/syslog`** and wait for appended data:

```
root@ubuntu:~# tail -n5  -f /var/log/syslog
Jul 31 07:57:53 ubuntu systemd[1]: Finished Daily apt upgrade and clean activities
.
Jul 31 07:57:53 ubuntu systemd[1]: apt-daily-upgrade.service: Consumed 1.408s CPU
time.
Jul 31 08:06:11 ubuntu rsyslogd: [origin software="rsyslogd" swVersion="8.2112.0"
x-pid="666" x-info="https://www.rsyslog.com"] rsyslogd was HUPed
Jul 31 08:17:01 ubuntu CRON[6072]: (root) CMD (   cd / && run-parts --report /etc
cron.hourly)
Jul 31 09:17:01 ubuntu CRON[6321]: (root) CMD (   cd / && run-parts --report /etc
cron.hourly)
```

Figure 5.22: *Example using command head*

Special characters

Working in a Linux console and with commands involves the use of some special characters which have particular meanings for the *shell*. Some of these special characters were described previously, such as tilde (~) for the home directory, greater than (>) and less than (<) for *standard input and output,* and bar (|) for the data *pipeline*. Other special characters indirectly shown are hyphen (-) used to specify options, slash (/) to specify or separate directories, or space symbol to

separate commands from options and arguments. *Table 5.13* shows the rest of the special characters when working with a *shell*:

Special character	Description
\	Escape a special character
$ (dollar)	Used to access variable date
" (double quote)	Protects everything between double quotes to be treated as a special character and with the possibility to evaluate variables.
' (single quote)	Same as a double quote but not allowing to evaluate variables.
# (hash)	Line starting in the shell with hash is marked as a comment.
& (and)	Used to execute a process in the background. Described later in this chapter.
? (question mark)	It matches one character only
* *(asterisk)*	It matches one or more characters.
[ ] (square brackets)	Range of characters
{ } (curly brackets)	Specify different options
() (parentheses)	Start a *subshell*
; (semicolon)	Separates several commands in one line.

Table 5.13: *Special character list and description*

Figures 5.23 to *5.24* show the use of these special characters with real examples:

- Using double quotes and escape character (\) to access a file with a space in the name:

```
agonzalez@ubuntu:~$ ls filename with spaces
ls: cannot access 'filename': No such file or directory
ls: cannot access 'with': No such file or directory
ls: cannot access 'spaces': No such file or directory
agonzalez@ubuntu:~$ ls -l "filename with spaces"
-rw-rw-r-- 1 agonzalez agonzalez 0 Jul 31 10:29 'filename with spaces
agonzalez@ubuntu:~$ ls -l filename\ with\ spaces
-rw-rw-r-- 1 agonzalez agonzalez 0 Jul 31 10:29 'filename with spaces
```

Figure 5.23: *Example using special characters.*

- The difference between using double quotes and single quotes with variables:

```
agonzalez@ubuntu:~$ echo "My shell is $SHELL'
My shell is /bin/bash
agonzalez@ubuntu:~$ echo 'My shell is $SHELL'
My shell is $SHELL
```

Figure 5.24: *Example using special characters.*

- For example, using hash (#) to not execute a command useful to save in the history without executing it. Refer to *figure 5.25*:

```
agonzalez@ubuntu:~$ #this command is not executed
agonzalez@ubuntu:~$
```

Figure 5.25: *Example using special characters.*

- Using question marks and asterisks to match files and directories can be seen in *figure 5.26*:

```
agonzalez@ubuntu:~$ ls example.???
example.bin  example.doc  example.txt
agonzalez@ubuntu:~$ ls example.*
example.bin  example.doc  example.docx  example.txt
```

Figure 5.26: *Example using special characters.*

- Using square brackets to specify a range of characters to match. A hyphen can be used to specify a range, as shown in *figure 5.27*:

```
agonzalez@ubuntu:~$ ls example[1234].txt
example1.txt  example2.txt  example3.txt  example4.txt
agonzalez@ubuntu:~$ ls example[5-9].txt
example5.txt  example6.txt  example7.txt  example8.txt  example9.txt
```

Figure 5.27: *Example using special characters.*

- Using curly brackets is possible two specify different options; these two commands are evaluated as the same, as seen in *figure 5.28*:

```
agonzalez@ubuntu:~$ ls /etc/passwd /etc/shadow /etc/group
/etc/group  /etc/passwd  /etc/shadow
agonzalez@ubuntu:~$ ls /etc/{passwd,shadow,group}
/etc/group  /etc/passwd  /etc/shadow
```

Figure 5.28: *Example using special characters.*

- Using parenthesis allows to open a **`subshell`** to run commands, as seen in *figure 5.29*:

```
agonzalez@ubuntu:~$ echo "Current directory is $(pwd)"
Current directory is /home/agonzalez
```

Figure 5.29: *Example using special characters*

- Using semicolon is possible to run several commands in one line, as seen in *figure 5.30*:

```
agonzalez@ubuntu:~$ echo "Command 1" ; echo "Command 2"
Command 1
Command 2
```

Figure 5.30: *Example using special characters.*

There are special characters that can be combined to have a special functionality, as we already discussed. For example, "double greater than" (>>) appends data or symbol, and "and greater than" (&>) redirects *standard output and error*. Other three popular options are described in *table 5.14*:

Special character	Description
|| (double bar)	Indicates if the first command fails, the second command indicated will be executed
&& (double and)	Indicates if the first command is executed correctly, the second command will be executed.
-- *(double hyphen)*	Indicates the end of the options in the command line.

Table 5.14: *Special characters combination.*

Figure 5.31 shows the use of these special characters combination:

```
agonzalez@ubuntu:~$ failcommand || echo "First command failed"
failcommand: command not found
First command failed
agonzalez@ubuntu:~$ echo "Good command" || echo "This is never executed"
Good command
agonzalez@ubuntu:~$ failcommand && echo "This is never executed"
failcommand: command not found
agonzalez@ubuntu:~$ echo "Good command" && echo "First command succeeded"
Good command
First command succeeded
agonzalez@ubuntu:~$ ls -l -b.txt
ls: invalid option -- '.'
Try 'ls --help' for more information.
agonzalez@ubuntu:~$ ls -l -- -b.txt
-rw-rw-r-- 1 agonzalez agonzalez 0 Jul 31 13:30 -b.txt
```

Figure 5.31: *Example using special characters combination*

Regular expressions

Regular expressions are patterns used to match character combinations in strings. These expressions can be simple ones or advanced ones using a complex patterns. This section will focus on simple expressions to filter content inside files. There are three types of regular expressions: basic (*BRE)*, extended (*ERE),* and *Perl* (*PCRE).* Only the first two are going to be covered as an example. Some of the special characters for the *basic* type are shown in *table 5.15*:

Special character	Description
. (dot)	Matches any element
^ (circumflex)	Matches the beginning of the sequence.
$ (dollar)	Matches the end of the sequence.

Table 5.15: *Special characters combination*

For the *advanced* type, it is possible to match repetitions of characters and conditions. These types of special characters are shown in *table 5.16*:

Special character	Description
* *(asterisk)*	Zero or more repetitions
+ *(plus)*	One or more repetitions
? (question mark)	Zero or one repetition
{m}	Exactly *m* repetitions
{m,n}	Between *m* and *n* repetitions
| *(bar)*	Matches the left or the right expression.

Table 5.16: *Simple regular expressions examples.*

Commands *grep*

Command *grep* is a powerful tool with different possibilities to match text using regular expressions. *GREP* stands for *Global Search, a Regular Expression and Print*. This command, by default, uses basic regular expressions (*BRE),* but by using option *-E* is possible to use the advanced expressions (*ERE*) and the option *-P* for the Perl expressions (*PRE),* which is not covered in this book.

The syntax for the command *grep* is the following: `grep [options] pattern file(s)`. *Table 5.17* shows the popular options available:

Option	Description

-F, --fixed-strings	Interpret the pattern as a fixed string and not as a regular expression.
-e pattern, --regexp=pattern	Option to specify pattern and repeat this option to match several patterns.
-f file, --file=file	Obtain patterns from the file specified.
-i, --ignore-case	Ignore case distinctions in patterns.
-v, --invert-match	Invert the sense of matching, showing the non-matching lines.
-w, --word-regexp	Ensure the pattern matches with whole words.
-c, --count	Does not show the output and instead shows how many coincidences were found for each file.
-r, --recursive	Read all files under the directories specified.
-R, --dereference-recursive	Same as *-r, --recursive,* but if directories are symbolic, links will follow them too.

Table 5.17: *Popular options for the command grep*

Figures 5.32 to *5.36* show different examples using basic patterns and different *grep* options:

- Search for the text *daemon* in any position in the line inside of the file **`/etc/passwd`**:

```
agonzalez@ubuntu:~$ grep daemon /etc/passwd
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
usbmux:x:112:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologir
```

Figure 5.32: *Example using the command grep*

- Search for the text **`daemon`** in starting the line inside of the file **`/etc/passwd`**:

```
agonzalez@ubuntu:~$ grep ^daemon /etc/passwd
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
```

Figure 5.33: *Example using the command grep*

- Count how many lines in **`/etc/passwd`** contain the word *bash:*

```
agonzalez@ubuntu:~$ grep -c bash /etc/passwd
3
```

Figure 5.34: *Example using the command grep*

- Filter users, which are five characters length long. Five any character following by "*:x:*":

```
agonzalez@ubuntu:~$ grep ^.....:x: /etc/passwd
games:x:5:60:games:/usr/games:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologir
```

```
uuidd:x:108:114::/run/uuidd:/usr/sbin/nologin
```

Figure 5.35: *Example using the command grep*

- Count how many lines are empty in **`/etc/`**legal and show content ignoring empty lines using the option **`-v`** (**`--invert-match`**); pattern **`^$`** means the line starts and ends without any content on it:

```
agonzalez@ubuntu:~$ grep -c ^$ /etc/legal
3
agonzalez@ubuntu:~$ grep -v ^$ /etc/legal
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

Figure 5.36: *Example using the command grep*

Figures 5.37 to *5.41* show different examples using advanced patterns and different *grep* options:

- Check the content of the file and filter starting with the letter *r*, followed by a letter *o* (optionally) or multiple ones, and ending with the letter *t:*

```
agonzalez@ubuntu:~$ cat text
rt
rot
root
pen
peen
agonzalez@ubuntu:~$ grep -E ro*t text
rt
rot
root
```

Figure 5.37: *Example using the command grep*

- Check the content of the file and filter starting with the letter *r*, followed by a letter *o* or several, and ending with a letter *t:*

```
agonzalez@ubuntu:~$ grep -E ro+t text
rot
root
```

Figure 5.38: *Example using the command grep*

- Check the content of the file and filter starting with the letter *r*, followed by an *o* (optional) and ending with a letter *t:*

```
agonzalez@ubuntu:~$ grep -E ro?t text
rt
rot
```

Figure 5.39: *Example using the command grep*

- Check the content of the file and filter starting with the letter *r*, followed by two letters *o*, and ending with the letter *t*. Filter to starting with letter *p*, followed by 1 or 2 letters *e* and ending with letter *n*:

```
agonzalez@ubuntu:~$ grep -E -e "ro{2}t" -e "pe{1,2}n" text
root
pen
peen
```

Figure 5.40: *Example using the command grep*

- Check for the word *agonzalez* and *root* in two different ways:

```
agonzalez@ubuntu:~$ grep -e root -e agonzalez /etc/passwd
root:x:0:0:root:/root:/bin/bash
agonzalez:x:1000:1000:agonzalez,12,1234,12345678:/home/agonzalez:/bin/bash
agonzalez@ubuntu:~$ grep -E "root|agonzalez" /etc/passwd
root:x:0:0:root:/root:/bin/bash
agonzalez:x:1000:1000:agonzalez,12,1234,12345678:/home/agonzalez:/bin/bash
```

Figure 5.41: *Example using the command gre*

Commands *awk*

AWK is a pattern scanning and processing language. The tool *awk* uses regular expressions to match lines, and it allows to print some parts. This command has

many options and possibilities for complex operations; only some real use cases are shown in *figures 5.42* and *5.43*:

- Print the username of the users using shell **bash**:

```
agonzalez@ubuntu:~$ awk -F: '/bash$/{print $1}' /etc/passwc
root
agonzalez
testuser
```

Figure 5.42: *Example using the command awk*

- Search the string **/home** and print the columns for the username, the home, and the *shell:*

```
agonzalez@ubuntu:~$ cat /etc/passwd | awk -F":" '/:\/home/{print  $1":"$6":"$7 }
syslog:/home/syslog:/usr/sbin/nologin
agonzalez:/home/agonzalez:/bin/bash
testuser:/home/testuser:/bin/bash
```

Figure 5.43: *Example using the command awk with the output of the command cat*

Formatting the output

Linux provides several tools to format the content of a file or the output of another command. These commands are part of the package *coreutils*, which is installed by default on the Linux distributions. *Table 5.18* shows some of the popular commands and their description:

Command	Description
nl	Number lines of files
sort	Sort lines of a text file
tac	Shows the files specified in reverse
tr	Translate or delete characters
sed	Filters and transforms text
rev	Reverse lines character wise
wc	Word and line count
fmt	Simple optimal text formatter
uniq	Omit or report repeated lines
join	Join lines of two files on a common field
paste	Merge lines of files
shuf	Generate random permutations

Table 5.18: *Popular commands for formatting output*

Figures 5.44 to *5.55* show one example for each of the commands:

- Using *nl* to numerate the lines of one file:

```
agonzalez@ubuntu:~$ nl /etc/legal

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

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

Figure 5.44: *Output example for the command nl*

- Using *sort* for alphabetical sort:

```
agonzalez@ubuntu:~$ awk -F":" '/bash$/{print $1}' /etc/passwd | sort
agonzalez
root
testuser
```

Figure 5.45: *Output example for the command sort.*

- Using *tac* to visualize a file in a reverse way:

```
agonzalez@ubuntu:~$ tac /etc/legal

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

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

Figure 5.46: *Output example for the command tac*

- Using *tr* to convert characters from the text:

```
agonzalez@ubuntu:~$ echo "UXXER CASE LINE" | tr '[A-Z]' '[a-z]' | tr 'x' 'p
upper case line
```

Figure 5.47: *Output example for the command tr*

- Using *sed* to convert expressions from the text:

```
agonzalez@ubuntu:~$ cat /etc/hostname
ubuntu.example.com
agonzalez@ubuntu:~$ cat /etc/hostname | sed 's/example.com/localdomain/
ubuntu.localdomain
```

Figure 5.48: *Output example for the command sed*

- Using *rev* to reverse line character-wise:

```
agonzalez@ubuntu:~$ echo "Hello world" | rev
dlrow olleH
```

Figure 5.49: *Output example for the command rev*

- Using *wc* to count lines, words, and characters of one file:

```
agonzalez@ubuntu:~$ wc /etc/legal
  8  38 267 /etc/legal
```

Figure 5.50: *Output example for the command wc*

- Using *fmt* to format the output reformatting paragraphs to use 60 characters width:

```
agonzalez@ubuntu:~$ fmt -w 60 /etc/legal

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

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

Figure 5.51: *Output example for the command fmt*

- Using *uniq* to remove from the output the duplicated lines. Duplicated lines need to be adjacent:

```
agonzalez@ubuntu:~$ cat uniq.txt
one
two
three
two
one
agonzalez@ubuntu:~$ cat uniq.txt  | sort | uniq
one
three
two
```

Figure 5.52: *Output example for the command uniq*

- Using *join* to combine two files with a common field:

```
agonzalez@ubuntu:~$ cat names.txt
1 Alberto
2 John
3 Linus
agonzalez@ubuntu:~$ cat lastname.txt
1 Gonzalez
3 Torvalds
agonzalez@ubuntu:~$ join names.txt lastname.txt
1 Alberto Gonzalez
3 Linus Torvalds
```

Figure 5.53: *Output example for the command join*

- Using *paste* to merge two files line by line:

```
agonzalez@ubuntu:~$ cat names1.txt
Alberto
Linus
John
agonzalez@ubuntu:~$ cat lastname1.txt
Gonzalez
Torvalds
agonzalez@ubuntu:~$ paste names1.txt lastname1.txt
Alberto Gonzalez
Linus   Torvalds
John
```

Figure 5.54: *Output example for the command paste*

- Using *shuf* to randomly sort the lines:

```
agonzalez@ubuntu:~$ shuf /etc/passwd | head -1
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
agonzalez@ubuntu:~$ shuf /etc/passwd | head -1
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
```

Figure 5.55: *Output example for the command shuf*

File editors and file managers

Editing files is one of the common tasks when we operate with a Linux system. Edit configuration, developing scripts or applications, creating content such as documentation, or any other task related to files are usual tasks. For this purpose, there are available visual editors (*GEdit*, *Visual Studio, and Kate* as some examples) and text-based editors, which are more popular due to are usually already on the systems to administrate, either local or remote servers. The most popular editors are as follows:

- **nano:** a small and friendly editor. It allows editing a file in a quick way, with the possibility to open several files, have syntax highlight, and line numbering, among other features.
- **vi and vim: vi** is one of the most popular editors in Linux. Despite of its complexity, it is used by most Linux users, which requires advanced features during editing files. **vim** (VI Improved) enhances the traditional **vi** adding features such as multiple windows, syntax highlighting, and visual selection, among other features.
- **emacs:** This editor, like VI, is a complex one but used by many users. The biggest advance of this editor is the extensibility, not used only as a file editor but as a suite of tools such integrated browser, file manager, e-mail read, *git* integration, and much more functionalities.

This section will cover the basics for each of them. The editor **nano** is shown in *figure 5.56* after running it with the argument `names.txt`:

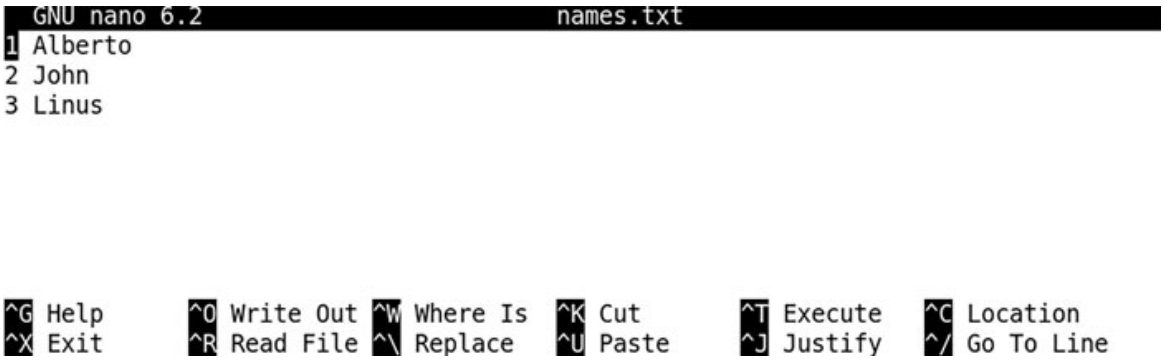


Figure 5.56: *nano editor*

In the bottom part of the screen, it indicates the actions available, and these actions are accessible using *Ctrl+Key*. For example, to write the changes is need to press *Ctrl+O*. For exiting the editing, the combination is *Ctrl+X*.

Figure 5.57 shows the editor **vim.** The editor VIM has three modes:

- *Normal mode*: in this mode is possible to write the changes, exit the application, copy lines, paste lines, and perform other operations related to the editor.
- *Insert mode*: in this mode is possible to write to the file.
- V*isual mode*: in this mode is possible to select several lines or columns.

Figure 5.57: *vim editor*

To start to manipulate the file and enter in the *Insert mode* is needed to press some of the following keys: *i* (for inserting where is the cursor), *a (*for appending after the cursor), *I* (for appending at the beginning of the line), or *A* (for appending at the ending of the line).

After the modifications are done, to write the changes or discard them is needed to go back to *Normal mode* by pressing the *Esc* key. The common operations inside are as follows:

- Type *:wq* or *:x* to write and quit the editor
- Type *:q* if no changes were made to quit the editor
- Type *:q!* to discard the changes

Figure 5.58 shows the editor **emacs**:

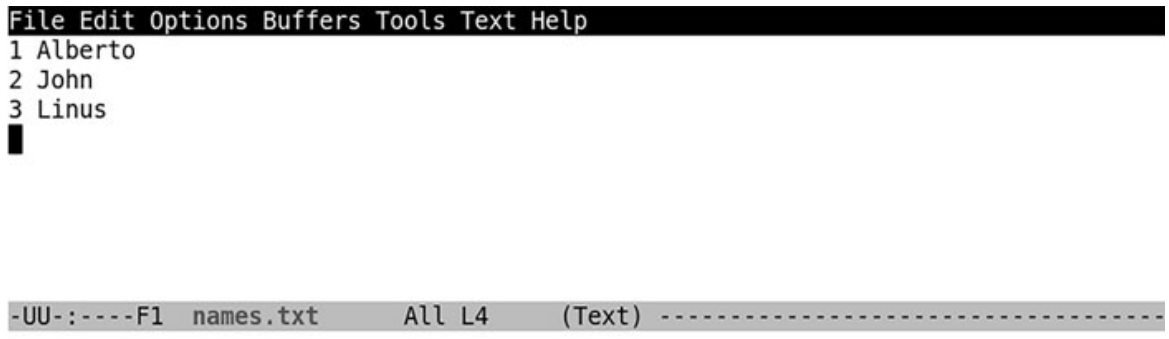


Figure 5.58: *emacs editor*

Some basic operations and the key combinations are shown in *Table 5.19*:

Key Combination	Description
CTRL-x CTRL-c	Exit emacs.
CTRL-x CTRL-s	Save the file without exiting.
CTRL-x u	Undo changes

Table 5.19: *Key combination for emacs*

A file manager is a tool to navigate between different directories and operate inside, such as create files or remove files. Most of the Desktop interfaces include a visual file manager for it. In a Linux console, navigation between directories and creation of directories or files is done using system-available commands. A console file manager allows ease to navigate and perform basic operations.

The file manager **mc** (*GNU Midnight Commanders)* is the most popular one, and *figure 5.59* shows an example of this file manager:

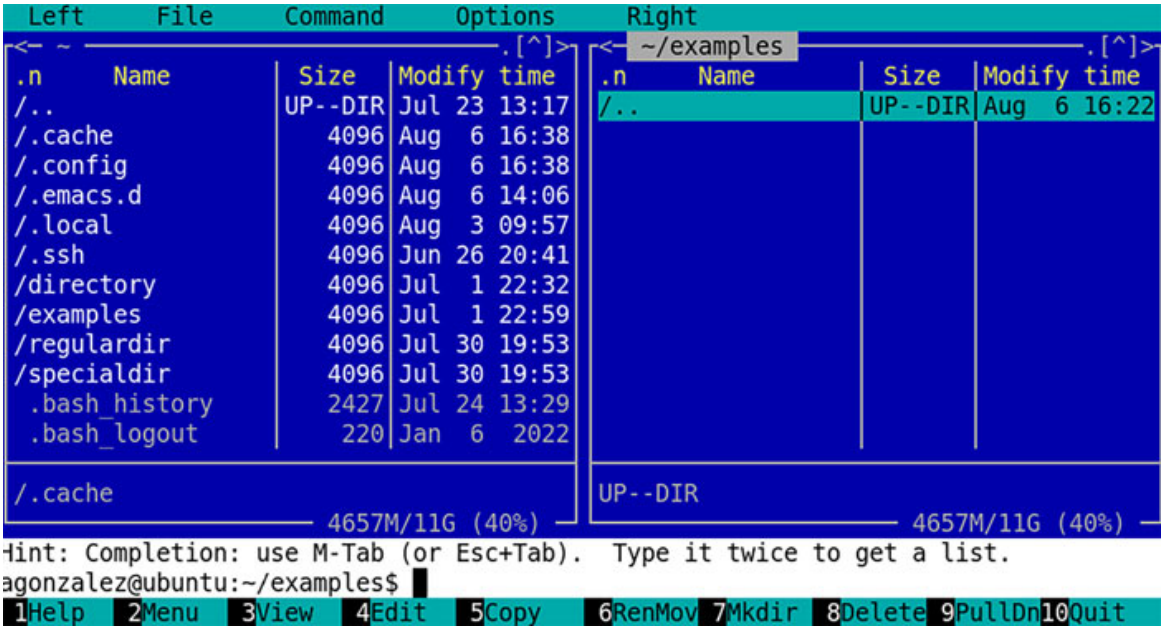


Figure 5.59: *mc file manager*

Using this file manager is possible to operate between two directories, such as moving or copying files between the left directory (left pane) and the right directory (right pane).

Processes management

A process is a program that is in execution. Each process has assigned a *process ID* (*pid*) and, as described previously, would have associated a directory inside

/proc. This directory will contain information about the process, such as how the application was called (file *cmdline*), a symbolic link to the directory from where it was called (file *cwd*), the environment variables for the process (file *environ*), a directory with links to files used by the application (directory *fd)*.

To list the processes running in the system, the command **`ps`** is used; this command is usually called with some of these options:

Option	Description
ps a	List all the processes running associated with a terminal
ps au	Uses user-oriented format
ps aux	Shows processes without a terminal associated
ps -e	Shows all the processes
ps -ef	Shows all the processes with a full-format listing.

Table 5.20: *Linux interruption signals*

Figure 5.60 shows an output example for the command **`ps`**:

```
agonzalez@ubuntu:~$ ps u
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
agonzal+   20949  0.0  0.1   9388  6048 pts/0    Ss   Aug02   0:00 -bash
agonzal+   26628  0.0  0.1   9144  5700 pts/1    Ss   14:53   0:00 -bash
agonzal+   26699  0.0  0.0  10892  3856 pts/1    S+   14:57   0:00 top
agonzal+   26707  0.0  0.0  10460  3248 pts/0    R+   14:57   0:00 ps u
```

Figure 5.60: *Output example for the command ps*

An interactive command to list the process is named **`top`**, which include system information resources which would be described in the next chapter. *Figure 5.61* shows the information shown by the command:

```
top - 15:33:12 up 11 days, 21:52,  2 users,  load average: 0.00, 0.00, 0.00
Tasks: 119 total,   1 running, 118 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3925.7 total,   1526.7 free,    261.0 used,   2138.0 buff/cache
MiB Swap:   2244.0 total,   2244.0 free,      0.0 used.   3371.7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  25197 root      20   0       0      0      0 I   0.3   0.0   0:01.06 kworker/0+
      1 root      20   0  175876  12884   8112 S   0.0   0.3   0:11.01 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.16 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
```

Figure 5.61: *Output example for the command top*

The command **`pidof`** is useful to list the **`process id`** for a command specified. *Figure 5.62* shows the **`pid`** for the command top running and using the option **`-q`**

of command ps to filter for that id:

```
agonzalez@ubuntu:~$ pidof top
26699
agonzalez@ubuntu:~$ ps -ef -q $(pidof top)
UID          PID    PPID  C STIME TTY          TIME CMD
agonzal+   26699   26628  0 14:57 pts/1    00:00:02 top
```

Figure 5.62: *Output example command pidof*

There are two types of processes on Linux:

- *Foreground processes:* this process interacts with the user who invoked it. This process would interact using the *standard input*, *standard output,* or *standard error*. Meantime the process running the *shell* used would not accept other commands.
- *Background processes*: this process does not interact with the user and usually are executed by the system. As the user is possible to run a background process, the *shell* will allow the user to execute other commands.

A process running is accepting signal interruptions. The common ones are shown in *Table 5.21*:

Signal	Signal ID	Key combination	Description
SIGHUP	*1*		Hangup detected, used to reload application configuration.
SIGINT	*2*	Ctrl-C	Interrupt from keyboard
SIGQUIT	*3*	Ctrl-D	Quit from keyboard
SIGKILL	*9*		Kill signal
SIGSTOP	*17,19,23*	Ctrl-Z	Stop process

Table 5.21: *Linux interruption signals*

A *foreground process* running can be *killed* using *Ctrl-C*; if it is waiting for *input* is possible to use Ctrl-D to finish the *standard input* wait, and to use Ctrl-Z would be stopped. It is possible to send signal interruptions to *background process*es using the command **`kill`**, its syntax is as follows: **`kill [-signal | -s signal] pid`**.

Some examples are as follows:

- Kill a process: **`kill -9 1234`**
- Stop a process: **`kill -s SIGTOP cat`**

To execute a process in *background* the symbol (&) is included in the command. It is possible to use the **`builtin`** *command* called **`jobs`** to list the processes for the user running in the *background,* as *figure 5.63* shows:

```
agonzalez@ubuntu:~$ sleep 30 &
[1] 26717
agonzalez@ubuntu:~$ sleep 60 &
[2] 26718
agonzalez@ubuntu:~$ jobs
[1]-  Running                 sleep 30 &
[2]+  Running                 sleep 60 &
```

Figure 5.63: *Example running commands in background and output example for the command jobs.*

A process running in the *foreground* can be sent to the *background* first stopping it (*Ctrl-Z*) and then running the command **`bg`** as is shown in *figure 5.64*:

```
agonzalez@ubuntu:~$ sleep 120
^Z
[1]+  Stopped                 sleep 120
agonzalez@ubuntu:~$ jobs
[1]+  Stopped                 sleep 120
agonzalez@ubuntu:~$ bg
[1]+ sleep 120 &
agonzalez@ubuntu:~$ jobs
[1]+  Running                 sleep 120 &
```

Figure 5.64: *Example of background and foreground processes.*

A process can be moved from *background* to *foreground* using the command **`fg`**. It is possible to specify the id adding as argument %N (same for command **`bg`**); *figure 5.65* shows an example:

```
agonzalez@ubuntu:~$ sleep 30 &
[1] 26721
agonzalez@ubuntu:~$ sleep 60 &
[2] 26722
agonzalez@ubuntu:~$ fg %1
sleep 30
```

Figure 5.65: *Example using the command fg.*

When the user session is closed, the background processes are *killed*. To avoid this behavior and keep running the processes Linux offers two options:

- Run the *builtin command* named **`disown`** to dissociate *background* running processes to the current user.
- Execute the *background process* using the application **`nohup`**, for example: **`nohup myapplication &.`**

Operate with processes priorities

Linux is using a *scheduler* to assign resources to perform tasks in the system. Some process requires to have more priority than others, especially the processes which are accessing to system resources or to the Linux core. There are two types of processes as follows:

- *Real-time processes*: These processes have the capability to run in time without being interrupted by any other processes. These processes have a system priority between the range 1 (low) and 99 (high).
- *Regular processes*: These processes can be interrupted by other processes with more priority, which require access to the system resources. These processes have a *nice* value which indicates the priority of getting the resources. These values are from –20 (high priority) to +19 (low priority). A regular process has as a default *nice* value 0. Negative values (higher priority) can be only specified by administrators.

The option *-l* for the command **ps** and the command **top** shows the priority and the *nice* value for the processes running. This section will focus on how to change the *nice* values for a process.

Figure 5.66 shows the example output for command **ps** with option *-l* to show the *priority* and the *nice* values:

```
agonzalez@ubuntu:~$ ps -l -q $(pidof top)
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   26798   26628  0  80   0 -  2723 do_sel pts/1    00:00:00 top
```

Figure 5.66: *Output example listing priority and nice value.*

The command **top** with the option *-p* and the **pid** will filter to that process and it would show the following example as shown in *figure 5.67*:

```
top - 16:19:09 up 11 days, 22:38,  2 users,  load average: 0.00, 0.00, 0.00
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3925.7 total,   1534.6 free,    253.0 used,   2138.1 buff/cache
MiB Swap:   2244.0 total,   2244.0 free,      0.0 used.   3379.7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  26798 agonzal+  20   0   10892   3900   3212 S   0.3   0.1   0:00.41 top
```

Figure 5.67: *Output example for the command top showing priority and nice value.*

The command **ps** would show the priority as a 100-*nice value* and the command *top* as a 20+*nice* value.

The command *nice* allows to execute commands with different *nice value,* and the command *renice* allows to change of the *nice value* for a running process. *Figures 5.68* to *5.69* show some examples using both commands:

- Run the command **sleep** with a different *nice value (*lower priority value)*:*

```
agonzalez@ubuntu:~$ nice -n 15 sleep 600 &
[1] 26907
agonzalez@ubuntu:~$ ps -l -q $(pidof sleep)
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   26907   20949  0  95  15 -  1547 hrtime pts/0    00:00:00 sleep
```

Figure 5.68: *Example using command nice*

- Change *nice value* for a running process. As regular users only decreased *nice value* is allowed:

```
agonzalez@ubuntu:~$ renice -n 16 $(pidof sleep)
26907 (process ID) old priority 15, new priority 16
agonzalez@ubuntu:~$ ps -l -q $(pidof sleep)
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   26907   20949  0  96  16 -  1547 hrtime pts/0    00:00:00 sleep
```

Figure 5.69: *Example using command renice*

Conclusion

Managing files, directories, and processes are required knowledge of administrating and using Linux systems. This chapter covered all the basics and more advanced options related to create and manipulate files and directories using system tools.

File and directory permissions are an important fact to protect files and directories to unwanted access. Different tools were described to assign user and group owners and permissions to files and directories.

Popular file editors and file managers were described to demonstrate the use of different tools to edit files, create directories, and navigate through different directories in the system.

Processes in Linux were described, and the importance of the priority when a process is executed and commands to modify the default priority were shown.

Key facts

- Linux have a default layout directory structure.
- Each file and directory have three permissions: read, write and execute.
- Permissions for files and directory can be set to the user owner, group owner or to others.
- Multiple commands are available for formatting the output.
- There are available multiple useful file editors and file managers.

Questions

1. What system directory keeps the system logs?
    a. /var/log/
    b. /tmp/
    c. /sys/log
2. What octal representation corresponds to *u=rwx,g=rwx,o=rx?*
    a. 664
    b. 775
    c. 777
3. What command is used to count lines, words, and characters?

a. wc
b. count
c. sed

4. Using *normal mode* in VI(M) how to exit discarding the changes?

a. :x
b. :wq
c. :q!

5. What range is valid for *nice value*?

a. -20 to 19
b. -19 to 20
c. 0 to 100

Answers

1. a
2. b
3. a
4. c
5. a

CHAPTER 6

Monitoring System Resources

Introduction

This chapter covers how to monitor the system resources in a Linux system. It will introduce how the CPU and memory work in modern systems and the common tools available to check the status with the objective to do troubleshooting in case of issues or overload.

Other important resources to monitor to avoid downtimes in applications is the disk usage and the available space. Different tools will be described to check the current status and to detect the physical state of the disks.

Monitoring the networking became one of the most important tasks in Linux systems, especially with the introduction of complex networking in the systems to provide connectivity for virtual machines and containers.

The last part of the chapter focuses on disk quotas and resource limits and helping to protect the system when it is accessible by regular users to avoid the resources getting exhausted.

Structure

In this chapter, we will discuss the following topics:

- Monitoring CPU resources
- Monitoring memory resources
- Monitoring disk usage and available space
- Monitoring network resources
- Quotas and limits

Monitoring CPU resources

A **Central Processing Unit** (**CPU**), a processor, executes instructions received from the system. In modern systems, there are three important concepts that are needed to understand, to be able to monitor and interpret the data received from the tools:

- **CPU Socket (CPU slot)**: is a mount on a computer *motherboard* that accepts a CPU chip.
- **CPU Core**: a separate processing unit that reads and executes instructions. In modern systems, a *CPU* contains several cores.
- **CPU Thread**: A *thread* is a queue for an operating system instruction. Modern systems enable *hyper-threading* enabling two *threads*.

For example, if a system has two sockets, with four cores for each CPU, and has enabled the *hyper-threading*, it would have available eight cores and 16 threads. It is capable of running eight operating system instructions per cycle but would be able to have a *queue* of up to 16 tasks per cycle.

An advanced concept for physical servers is named **Non-Uniform Memory Access** (**NUMA**). This architecture divides the memory into local and remote memory relative to the processors. CPU accessing the local memory assigned has better performance than accessing the remote access using an *interconnect.* *Figure 6.1* shows a representation of a node with two NUMA with one CPU socket and four cores each:

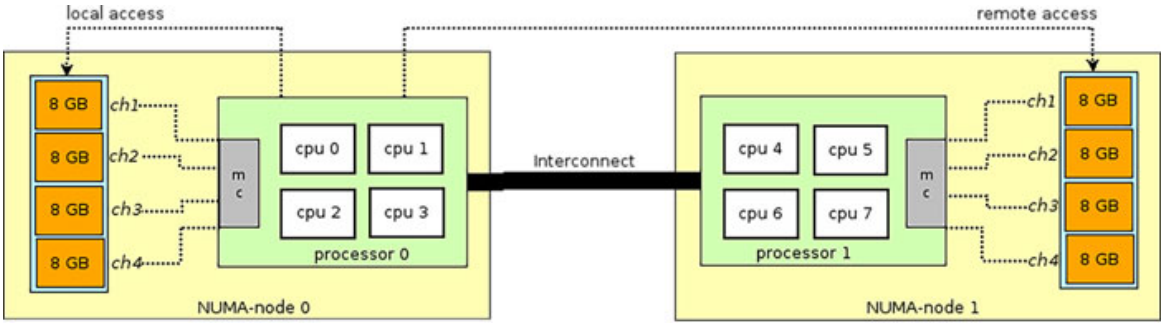


Figure 6.1: *Standard Linux filesystem hierarchy. Source: boost.org*

Obtaining CPU(s) information

The command `lscpu` described in *Chapter 3, Using the Command Line Interface*, shows the important information related to the CPU socket, core, threads, and NUMA. *Figure 6.2* shows a node with two CPU sockets and eight cores per socket.

```
[root@server ~]# lscpu|grep --color=never -E "^(CPU\(|Thread|Core|Socket|NUMA)"
CPU(s):              32
Thread(s) per core:  2
Core(s) per socket:  8
Socket(s):           2
NUMA node(s):        2
NUMA node0 CPU(s):   0-7,16-23
NUMA node1 CPU(s):   8-15,24-31
```

Figure 6.2: *Using command lscpu to obtain CPU information.*

The command **`lstopo-no-graphics`** part of the package **`hwloc`** provides a great visualization of the NUMA architecture and the association between CPUs and local memory. *Figure 6.3* illustrates an output example of this command for the same server as *figure 6.2*:

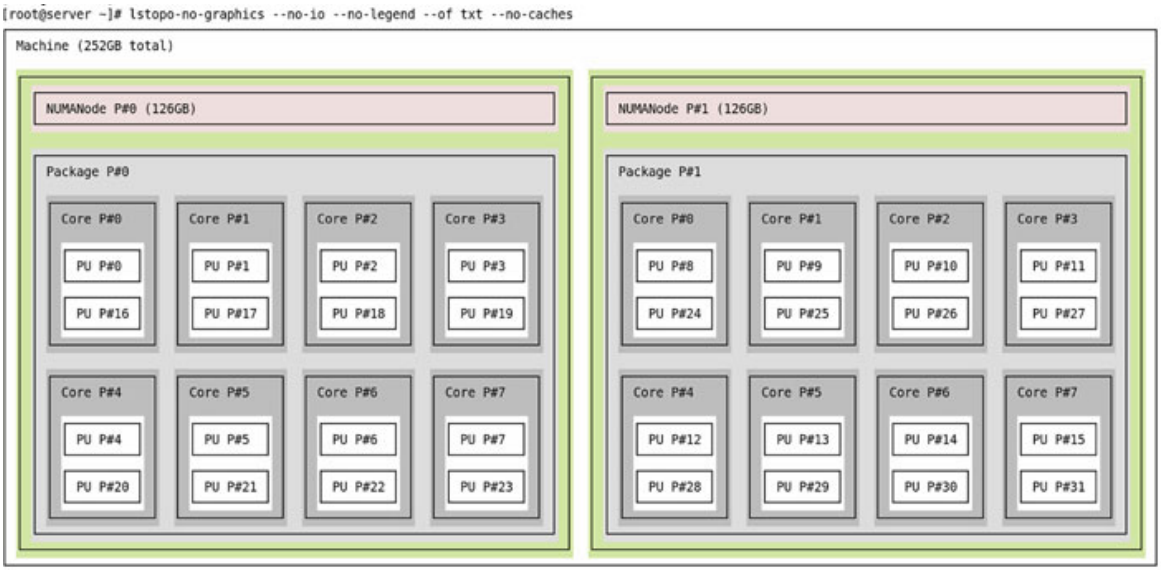


Figure 6.3: *Output example for the command lstopo-no-graphics.*

The file **`/proc/cpuinfo`** contains the list of all processors (*core/thread)* with a detailed information for each one. *Figure 6.4* shows an example using **`grep`**, **`sort`**, and **`uniq`** to create a report of how many CPUs are in each socket:

```
[root@server ~]# grep "physical id" /proc/cpuinfo |sort|uniq -c
     16 physical id     : 0
     16 physical id     : 1
```

Figure 6.4: *Getting information from file /proc/cpuinfo*

Other example commands to get processor information are as follows:

- *lshw -class processor (lshw -class processor -short)*
- *dmidecode -t Processor*
- *nproc* (shows only the number of processors)
- *hwinfo --short –cpu*

Understanding the system load and load average

System load is a measure of the amount of computational work that a computer system performs. The load average represents the system load over a period of time. A processor with a load of 140% indicates it is running overload by 40%, just as a processor with a load of 30% is idling for 70% of the time for the period observed. The CPU load is represented by a decimal number, such as 1.40 to indicate 140% and 0.30 to indicate 30%.

A common analogy to understand the CPU Load in Linux is related to the traffic. A single-core CPU is like a single lane in a road. Having more cores and CPUs would be like having more lanes, such that the load would be distributed among them comfortably. This is illustrated in *figure 6.5*:

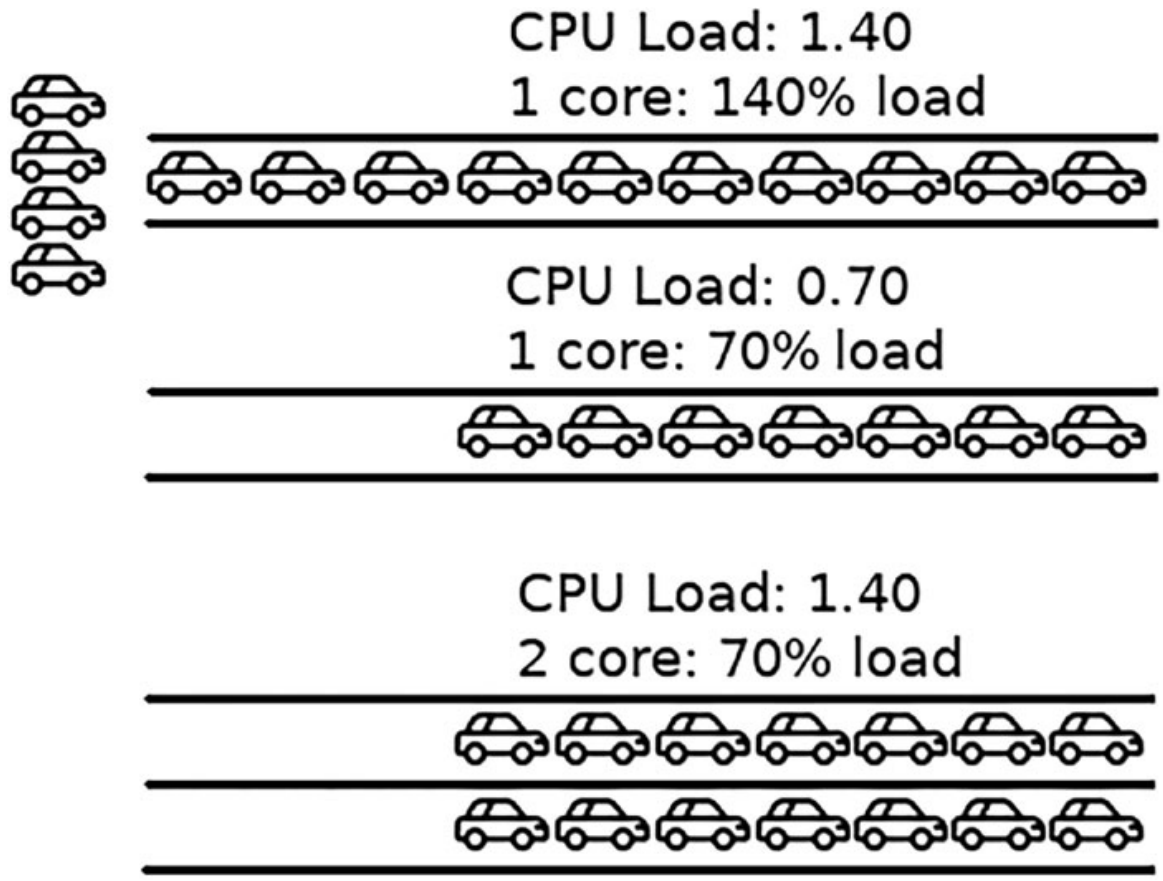


Figure 6.5: *Analogy with processors, load, and traffic*

By having multiple CPUs available, the tasks are going to be distributed between different CPUs, and the load would be related to the number of CPUs. A system load of 3.00 having four processors indicates that it is using 66.67% of the available CPU resources.

Command uptime and file /etc/loadavg

This command, described in *Chapter 3, Using the Command Line Interface*, to see how long the system was running, is used to have a simple overview. Command uptime shows the CPU load average in the system. The load average on Linux systems is represented for the periods of 1 minute, 5 minutes, and 15 minutes. *Figure 6.6* illustrates the average in the last minute was 0.13, for the last 5 minutes it was 0.07, and for the last 15 minutes it was 0.01:

```
[root@server ~]# uptime
 16:44:42 up 73 days, 10:00,  2 users,  load average: 0,13, 0,07, 0,01
```

Figure 6.6: *Output example for command uptime*

Most of the commands open the file **`/proc/loadavg`** to obtain information about the system load. This file contains the information for the periods described previously in the first three columns; the fourth column indicates, separated by a slash (/), how many processes are accessing to the CPU and how many processes are being executed in the system, and the last column indicates the last *process id* assigned. *Figure 6.7* features a content example for the file**`/prod/loadavg`**:

```
[root@server ~]# cat /proc/loadavg
0.04 0.09 0.05 2/640 2147326
```

Figure 6.7: *Content example for file /proc/loadavg*

Command top

The command *top* provides a dynamic real-time information about the system. In *Chapter 5, Managing Files, Directories, and Processes*, this command was introduced to list the processes running in the system. The head part of the information shown is related to the system resources usage, as is illustrated in *figure 6.8*:

```
top - 13:33:43 up 74 days,  6:49,  2 users,  load average: 0,08, 0,05, 0,05
Tasks: 466 total,   2 running, 463 sleeping,   1 stopped,   0 zombie
%Cpu(s):  0,0 us,  0,3 sy,  0,0 ni, 99,7 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem : 257568,8 total,  23417,0 free, 184388,5 used,  49763,3 buff/cache
MiB Swap:      0,0 total,      0,0 free,      0,0 used.  67273,6 avail Mem
```

Figure 6.8: *Top command output header example*

The information shown in the first three lines of the *top*'s command head are as follows:

- Uptime, the number of users logged in, and load average in the system. Same as *uptime*'s command output.

- List of the number of processes running, indicating how many are using the CPU and how many are in sleeping, stopped, and *zombie* (meaning is not accepting interruptions and usually require to reboot of the system to recover them) states.
- Percent of usage for all CPUs in the system, with the following usage:
    - *us (user)*: time running (without *nice value*) user processes.
    - *sy (system)*: time running *kernel* processes.
    - *ni (nice)*: time running (with *nice value*) user processes.
    - *id (idle)*: time spent idling.
    - *wa* (*input/output wait)*: time waiting for the Input/Output completion.

The command **`top`** has different visualization options (press key *h* to obtain more options); for example, pressing key *1* will display information for each CPU, and pressing key *2* will show for each CPU Socket (*NUMA Node*). *Figures 6.9* and *6.10* show both examples:

- Display pressing key *1* (truncated output):

```
top - 13:45:20 up 74 days,  7:00,  2 users,  load average: 0,00, 0,03, 0,04
Tasks: 467 total,   1 running, 465 sleeping,   1 stopped,   0 zombie
%Cpu0  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu1  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu2  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu3  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu4  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu5  :  0,3 us,  0,3 sy,  0,0 ni, 99,3 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu6  :  0,3 us,  0,0 sy,  0,0 ni, 99,7 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu7  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu8  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu9  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu10 :  0,3 us,  0,0 sy,  0,0 ni, 99,7 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
```

Figure 6.9: *Top command output header example*

- Display pressing key *2*:

```
top - 13:45:45 up 74 days,  7:01,  2 users,  load average: 0,00, 0,02, 0,04
Tasks: 467 total,   1 running, 465 sleeping,   1 stopped,   0 zombie
%Cpu(s):  0,0 us,  0,2 sy,  0,0 ni, 99,8 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Node0 :  0,0 us,  0,0 sy,  0,0 ni, 99,9 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Node1 :  0,0 us,  0,5 sy,  0,0 ni, 99,5 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
```

Figure 6.10: *Top command output header example*

Commands atop and htop

These commands are a modern alternative to the traditional *top*. Figure 6.11 shows the aspect of the application *atop*, which includes more information related to the system:

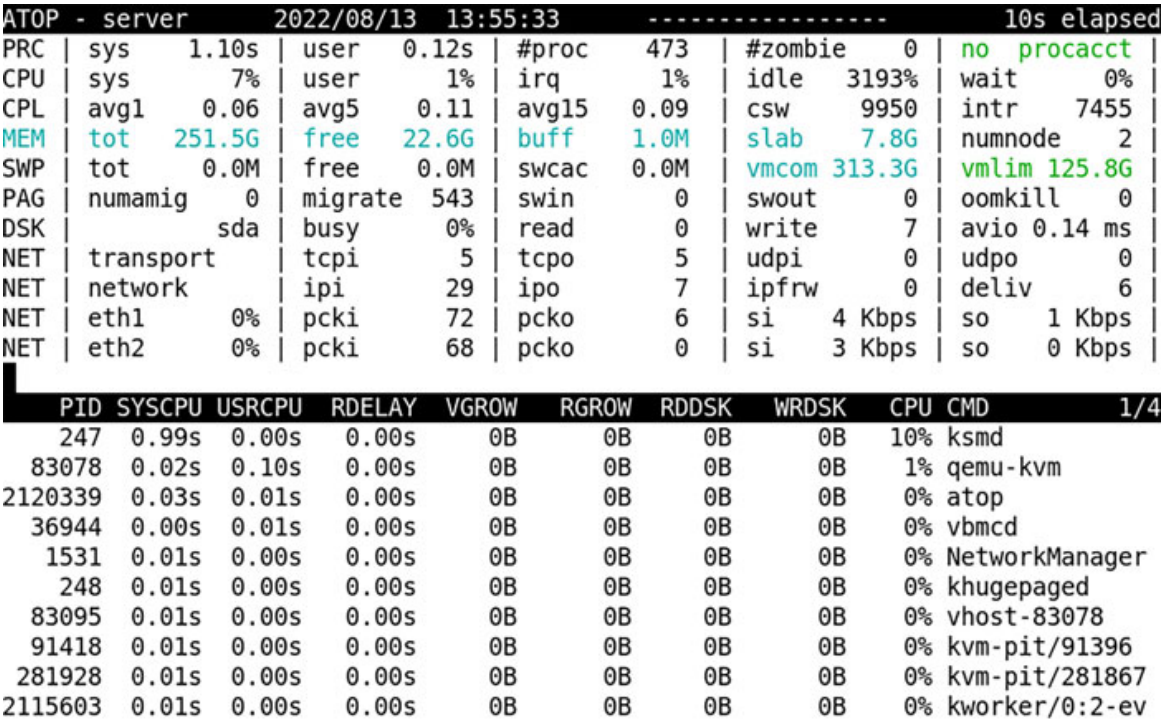


Figure 6.11: *Example aspect of command atop*

Figure 6.12 shows the aspect of the **`htop`** application, which have between overview of the *CPU* usage:

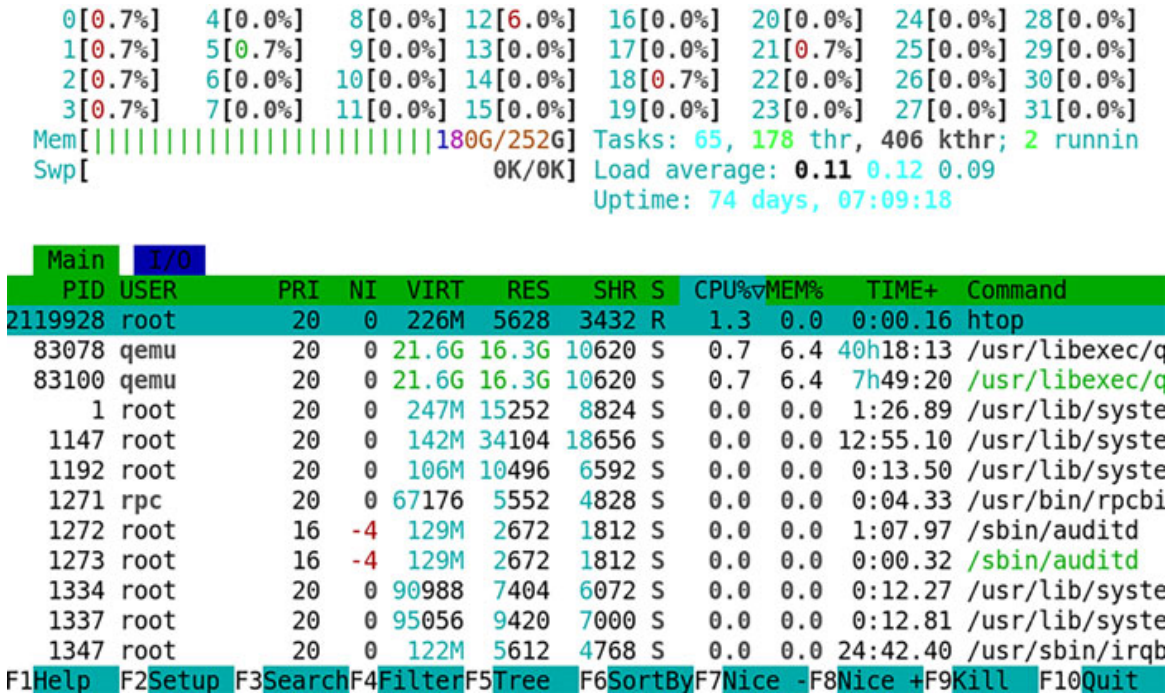


Figure 6.12: *Example aspect of command htop*

Command mpstat

This command is part of the *sysstat* package, which requires to be installed to be used. This application reports processors-related statistics. *Figure 6.13* shows an example output without any option or argument specified:

```
[root@server ~]# mpstat
Linux 4.18.0-305.30.1.el8_4.x86_64 (server)     13/08/22        _x86_64_        (32 CPU)

14:00:47     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
14:00:47     all    0,05    0,00    1,86    0,27    0,19    0,17    0,00    8,16    0,00   89,32
```

Figure 6.13: *Output example for the command mpstat*

The command *mpstat* allows the options *-N* to specify the *NUMA nodes* and the option *-P* to indicate the processors to display the statistics. The arguments available for the application are *interval* and *count*, which will create a *count* number of reports every *interval* seconds. *Figure 6.14* shows an example to generate five reports each 6 seconds:

```
[root@server ~]# mpstat 6 5
Linux 4.18.0-305.30.1.el8_4.x86_64 (server)     13/08/22        _x86_64_        (32 CPU)

14:06:46     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
14:06:52     all    0,00    0,00    0,26    0,01    0,02    0,00    0,00    0,02    0,00   99,70
14:06:58     all    0,02    0,00    0,25    0,00    0,01    0,01    0,00    0,00    0,00   99,72
14:07:04     all    0,01    0,00    0,27    0,01    0,01    0,01    0,00    0,01    0,00   99,70
14:07:10     all    0,01    0,00    0,24    0,00    0,01    0,00    0,00    0,02    0,00   99,72
14:07:16     all    0,00    0,00    0,22    0,01    0,02    0,02    0,00    0,01    0,00   99,73
Average:     all    0,00    0,00    0,25    0,00    0,01    0,01    0,00    0,01    0,00   99,71
```

Figure 6.14: *Output example for the command mpstat*

The columns displayed are shown in *table 6.1*:

Column	**Description**
CPU	Processor number.
%usr	CPU utilization that occurred at the user level (application).
%nice	CPU utilization that occurred at the user level with nice priority.
%sys	CPU utilization that occurred at the system level (kernel).
%iowait	Time that the CPU or CPUs were idle during which the system had an outstanding disk I/O request.
%irq	Time spent by the CPU or CPUs to service hardware interrupts.
%soft	Time spent by the CPU or CPUs to service software interrupts.
%steal	Time spent in involuntary wait by the virtual CPU or CPUs while the hypervisor was servicing another virtual processor.
%guest	Time spent by the CPU or CPUs to run a virtual processor.
%gnice	Time spent by the CPU or CPUs to run a nice guest.
%idle	Time that the CPU or CPUs were idle and the system did not have an outstanding disk I/O request.

Table 6.1: *Column description for command mpstat*

Command sar

System Activity Reporter (**SAR**) is part of *sysstat* package too. This command can collect, report or save system activity information. To see *CPU* information, use the option *-u* and specify how often and how many reports are needed, similar to *mpstat*. *Figure 6.15* illustrates an example:

```
[root@server ~]# sar -u 2 5
Linux 4.18.0-305.30.1.el8_4.x86_64 (server)     13/08/22        _x86_64_        (32 CPU)

14:14:37        CPU     %user     %nice   %system   %iowait    %steal     %idle
14:14:39        all      0,03      0,00      0,28      0,00      0,00     99,69
14:14:41        all      0,00      0,00      0,34      0,00      0,00     99,64
14:14:43        all      0,05      0,00      0,30      0,00      0,00     99,66
14:14:45        all      0,00      0,00      0,30      0,02      0,00     99,69
14:14:47        all      0,03      0,00      0,25      0,00      0,00     99,70
Average:        all      0,02      0,00      0,29      0,00      0,00     99,68
```

Figure 6.15: *Output example for the command sar*

With option *-q,* the load average of the system is also is showing, as in *figure 6.16*:

```
[root@server ~]# sar -q 2 5
Linux 4.18.0-305.30.1.el8_4.x86_64 (server)     13/08/22        _x86_64_        (32 CPU)

14:25:11      runq-sz  plist-sz   ldavg-1   ldavg-5  ldavg-15   blocked
14:25:13            0       645      0,10      0,09      0,09         0
14:25:15            0       645      0,17      0,10      0,09         0
14:25:17            0       645      0,17      0,10      0,09         0
14:25:19            0       645      0,17      0,10      0,09         0
14:25:21            0       645      0,24      0,12      0,10         0
Average:            0       645      0,17      0,10      0,09         0
```

Figure 6.16: *Output example for the command sar*

Command iostat

This is another command part of *systat* package used to show information about the *CPU* and disk. With option *-c*, it is limiting the output to the CPU information, as is shown in *figure 6.17*:

```
[root@server ~]# iostat -c
Linux 4.18.0-305.30.1.el8_4.x86_64 (server)     13/08/22        _x86_64_        (32 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           8,20    0,00    2,21    0,27    0,00   89,32
```

Figure 6.17: *Output example for the command iostat*

Monitoring memory resources

Random-access memory (RAM), is a limited and usually expensive resource in physical systems. Understanding how system and applications are using memory is a required knowledge, to be able to administrate systems and avoid issues running critical applications.

Linux systems include memory management which is responsible to provide dynamic allocated portions of the memory to processes and to release that portion when the application finishes. Memory management provides the following:

- Option to offer more memory than the physical availability. For example, using *swap*, which uses the disk space to increase the memory space available.
- Each process has assigned a virtual address space and is isolated from other processes.
- Memory mapping, where the content of a file is linked to a virtual address of the process.
- Fair allocation of the memory for processes.
- Shared virtual memory between processes, for example, for libraries in use.

The command *free* explained in *Chapter 3, Using the Command Line Interface*, is the main tool to see the current status of the system. The file used to obtain memory information from the *kernel* is */proc/meminfo,* and *figure 6.18* (output truncated) shows an example of the complex content inside:

```
[root@server ~]# cat /proc/meminfo
MemTotal:       263750488 kB    Writeback:             0 kB     VmallocChunk:          0 kB
MemFree:        23652996 kB     AnonPages:      180445696 kB    Percpu:            69504 kB
MemAvailable:   68818260 kB     Mapped:            112424 kB    HardwareCorrupted:     0 kB
Buffers:            1072 kB     Shmem:            4196320 kB    AnonHugePages:  13256704 kB
Cached:         48141820 kB     KReclaimable:     3070832 kB    ShmemHugePages:        0 kB
SwapCached:            0 kB     Slab:             8222324 kB    ShmemPmdMapped:        0 kB
Active:         162465792 kB    SReclaimable:     3070832 kB    FileHugePages:         0 kB
Inactive:       66132548 kB     SUnreclaim:       5151492 kB    FilePmdMapped:         0 kB
Active(anon):   159888276 kB    KernelStack:        10864 kB    HugePages_Total:       0
Inactive(anon): 24763492 kB     PageTables:         545028 kB    HugePages_Free:        0
Active(file):    2577516 kB     NFS_Unstable:          0 kB    HugePages_Rsvd:        0
Inactive(file): 41369056 kB     Bounce:                0 kB    HugePages_Surp:        0
Unevictable:           0 kB     WritebackTmp:          0 kB    Hugepagesize:       2048 kB
Mlocked:               0 kB     CommitLimit:    131875244 kB    Hugetlb:               0 kB
SwapTotal:             0 kB     Committed_AS:   328596212 kB    DirectMap4k:     4670164 kB
SwapFree:              0 kB     VmallocTotal:   34359738367 kB  DirectMap2M:    242677760 kB
Dirty:               108 kB     VmallocUsed:           0 kB    DirectMap1G:    23068672 kB
```

Figure 6.18: *Example content of file /proc/meminfo*

It is possible to obtain information about physical memory using, for example, the command **`lshw`** as is illustrated in *figure 6.19* (truncated):

```
[root@server ~]# lshw -c memory -short |grep -v empty
H/W path         Device     Class          Description
=====================================================
/0/0                        memory         64KiB BIOS
/0/2a                       memory         256GiB System Memory
/0/2a/0                     memory         32GiB DIMM DDR4 Synchronous 2667 MHz (0,4 ns)
/0/2a/3                     memory         32GiB DIMM DDR4 Synchronous 2667 MHz (0,4 ns)
/0/2a/6                     memory         32GiB DIMM DDR4 Synchronous 2667 MHz (0,4 ns)
/0/2a/9                     memory         32GiB DIMM DDR4 Synchronous 2667 MHz (0,4 ns)
/0/2a/c                     memory         32GiB DIMM DDR4 Synchronous 2667 MHz (0,4 ns)
/0/2a/f                     memory         32GiB DIMM DDR4 Synchronous 2667 MHz (0,4 ns)
/0/2a/12                    memory         32GiB DIMM DDR4 Synchronous 2667 MHz (0,4 ns)
/0/2a/15                    memory         32GiB DIMM DDR4 Synchronous 2667 MHz (0,4 ns)
```

Figure 6.19: *Output example for the command lshw*

Command *vmstat*

This command reports information about virtual memory statistics. The first report produced gives averages since the last reboot. *Figure 6.20* shows an example output without specifying any option or argument:

```
[root@server ~]# vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 23668764   1072 51212792    0    0     4    52    0    0  8  2 89  0  0
```

Figure 6.20: *Output example for the command vmstat*

The fields related to the memory available are shown in *table 6.2*:

Column	Field	Description
Procs	r	The number of runnable processes
	b	The number of processes blocked waiting for I/O to complete.
Memory	swpd	the amount of virtual memory used.
	free	the amount of idle memory.
	buff	the amount of memory used as buffers.
	cache	the amount of memory used as a cache.
	inact	the amount of inactive memory. (-a option)
	active	the amount of active memory. (-a option)
Swap	si	Amount of memory swapped in from disk (/s).
	so	Amount of memory swapped to disk (/s).

Table 6.2: *Field description for the command vmstat*

With the option *-w (--wide*), the data is visualized in a wide output, and option *-s (--stats)* shows event counter statistics. *Figure 6.21* shows the output using the option *-a,* active/inactive memory, and options *-w and -S (display unit)*:

```
[root@server ~]# vmstat -a -w -S M
procs ----------------------memory---------------------- ---swap-- -----io---- -system-- --------cpu--------
 r  b         swpd         free        inact       active   si   so    bi    bo   in   cs  us  sy  id  wa  st
 0  0            0        23098        64604       158636    0    0     4    52    0    0   8   2  89   0   0
```

Figure 6.21: *Output example for the command vmstat*

Commands top, htop, and atop

The example information by these commands are shown in *figures 6.22–6.24*:

- **`top`** command memory information:

```
MiB Mem :   3925.7 total,   1969.5 free,    237.7 used,   1718.4 buff/cache
MiB Swap:   2244.0 total,   2244.0 free,      0.0 used.   3410.0 avail Mem
```

Figure 6.22: *Example memory info in the command top*

It is possible to press the key *m* to toggle the visualization of the memory information.

- **`htop`** command memory information:

Figure 6.23: *Example memory info in the command htop*

- **`atop`** command memory information:

```
MEM | tot     3.8G | free    1.9G | buff   96.4M  | slab  183.5M | numnode    1
SWP | tot     2.2G | free    2.2G | swcac   0.0M  | vmcom 502.9M | vmlim   4.1G
PSI | cpusome   0% | memsome   0% | memfull   0%  | iosome    0% | iofull    0%
```

Figure 6.24: *Example memory info in the command atop*

PSI stands for **Pressure Stall Information**, and it contains percentages of resource pressure related to CPU, memory, and Input/Output. The values **`memsome`** and **`memfull`** are pressure percentages during the entire interval. The values **`ms`** and **`mf`** show three percentages over the last 10, 60, and 300 seconds (only shown when the window size is bigger).

Command sar

The command **`sar`** is useful to create a report in a period of time to see the usage of the memory, as shown in *figure 6.25*:

```
root@ubuntu:~# sar -r 5 2
Linux 5.15.0-43-generic (ubuntu.example.com)    08/13/22        _x86_64_        (2 CPU)

19:43:10    kbmemfree   kbavail kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit
kbactive   kbinact   kbdirty
19:43:15      1997052   3476112    211724      5.27     98884   1523420    511164      8.09
  787008    905576         0
19:43:20      1997052   3476112    211724      5.27     98884   1523420    511164      8.09
  787008    905576         0
Average:      1997052   3476112    211724      5.27     98884   1523420    511164      8.09
  787008    905576         0
```

Figure 6.25: *Output example for the command sar*

Memory usage for each process

When a process is running, it has three memory values assigned:

- Virtual memory: It includes all code, data, and shared libraries.
- Reserved memory: Resident size, the non-swapped physical memory used.
- Shared memory: The memory shared with other processes

Figure 6.26 shows an excerpt of the *top*'s output command after pressing *M* (*shift-m*) to sort by memory usage:

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
282441 qemu      20   0   70,6g  64,2g  12980 S   0,0  25,5  91484:59 qemu-kvm
281431 qemu      20   0   70,6g  53,4g  13144 S   0,0  21,2  53283:06 qemu-kvm
281867 qemu      20   0   70,6g  43,1g  13268 S   0,0  17,1  40395:58 qemu-kvm
 91344 qemu      20   0   37,2g  28,6g  13552 S   0,0  11,4  41616:33 qemu-kvm
 89631 qemu      20   0   37,2g  25,3g  13072 S   0,0  10,0  52131:50 qemu-kvm
 91396 qemu      20   0   37,2g  23,3g  13320 S   0,0   9,3  39959:53 qemu-kvm
 83078 qemu      20   0   21,2g  16,0g  10620 S   1,0   6,4   2419:43 qemu-kvm
```

Figure 6.26: *Output example for the command sar*

The command **`ps`** using the long format (*aux*) shows the usage of the memory of the process. *Figure 6.27* shows an example output filtering for a process where VSZ (*virtual size*) and RSS (*resident set size,* reserved memory*)* are displayed in kilobytes:

```
[root@server ~]# ps aux -q 282441
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
qemu      282441 89.1 25.5 74079692 67366984 ?   Sl   jun03 91484:59 /usr/libexec/qemu-kvm -name
```

Figure 6.27: *Output example for the command ps*

The command **`pmap`**, with the argument *process id,* shows an advanced output about the reserved and shared memory.

Out of memory management

When an application is executed in Linux, it reserves memory space but usually is not using all of it. That allows Linux to be able to execute more applications than the memory available, expecting that the memory used for the application is going to be less than the available. Linux includes a manager to detect when the system is using more memory than it should and automatically kills non-critical processes which are using memory.

Each process has a *score* associated; a higher score indicates it is more likely to be killed. The score for a process is stored on **`/proc/<pid>/oom_score`** file.

Monitoring disk usage and available space

Storage is one of the Input/Output elements which can cause a bottleneck in the system due to many operations performed in the same storage device or related to physical issues. Linux provides tools to check usage and available space, the disk statistics for input/output operations, and to check the physical status of the storage devices

Command **`lshw`** can be used to list the physical storage components in the system, such as *RAID* controllers or *SATA* controllers. *Figure 6.28* shows an example of physical server information:

```
[root@server ~]# lshw -c storage -short
H/W path         Device     Class          Description
=====================================================
/0/100/11.4                 storage        C610/X99 series chipset sSATA Contro
/0/100/14/0/e/2             storage        Mass storage device
/0/100/1f.2                 storage        C610/X99 series chipset 4-port SATA
/0/100/1f.5                 storage        C610/X99 series chipset 2-port SATA
/0/2/0           scsi1      storage        MegaRAID SAS-3 3108 [Invader]
/0/8f            scsi0      storage
```

Figure 6.28: *Output example for the command lshw*

The same command can be used to list the storage attached to the server, such as physical disks (can be virtual disks configured as *RAID*) or virtual disks (*emulated disks,* for example, for remote CD/Floppy). *Figure 6.29* shows an example of the same server illustrated before:

```
[root@server ~]# lshw -c disk -short
H/W path         Device     Class          Description
=====================================================
/0/2/0/2.0.0     /dev/sda   disk           799GB MR9361-8i
/0/8f/0.0.0      /dev/sdb   disk           1474KB Virtual Floppy
```

Figure 6.29: *Output example for the command lshw*

Some new modern system uses the new interface protocol called **Non-volatile Memory Express** (**NVME**), which improves the previous protocol named **Advanced Host Controller Interface** (**AHCI**), providing reduced latency and improvement in parallel tasks. *Figure 6.30* shows the output of the command **`lshw`** in a system using *NVMe*:

```
[root@server2 ~]# lshw -c disk -short
H/W path             Device          Class          Description
=====================================================================
/0/100/1d.2/0/0      /dev/ng0n1      disk           NVMe disk
/0/100/1d.2/0/1      /dev/nvme0n1    disk           512GB NVMe disk
/0/0/0.0.0           /dev/sda        disk           SD/MMC
/0/0/0.0.0/0         /dev/sda        disk
```

Figure 6.30: *Output example for the command lshw*

The command **`lsblk`** shows information about the disks and the partitions. *Figure 6.31* shows an example of the output for the physical server used before:

```
[root@server ~]# lsblk -a
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 744,7G  0 disk
├─sda1   8:1    0 744,6G  0 part /
└─sda2   8:2    0    64M  0 part
sdb      8:16   1   1,4M  0 disk
```

Figure 6.31: *Output example for the command lsblk*

In a system using **Logical Volume Manager** (**LVM**) or encrypted disks using **Linux Unified Key Setup** (**LUKS**), the command **`lsblk`** will give useful information about the configuration, as shown in *figure 6.32*:

```
[root@server2 ~]# lsblk
NAME                                    MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
nvme0n1                                 259:0    0   477G  0 disk
├─nvme0n1p1                             259:1    0   200M  0 part
├─nvme0n1p2                             259:2    0     3G  0 part
├─nvme0n1p3                             259:3    0   250G  0 part
│ └─luks-8768059b-73c3-4ab0-a3eb-6ab69f97380e
│                                       253:0    0   250G  0 crypt
│   ├─RHELCSB-Root                      253:1    0    50G  0 lvm   /
│   ├─RHELCSB-Home                      253:2    0   150G  0 lvm   /home
│   └─RHELCSB-Swap                      253:3    0    32G  0 lvm   [SWAP]
├─nvme0n1p4                             259:4    0 223,1G  0 part
└─nvme0n1p5                             259:5    0   680M  0 part
```

Figure 6.32: *Output example for the command lsblk*

Command *iostat*

This command part of the *sysstat* package, described previously for the *CPU* usage, is usually used for the input/output statistics. The statistics show the

physical storage devices, the number of transfers (*tps*) per second, the data read and write data per second, and the data read and written in the period observed. *Figure 6.33* shows the output example; option *-y* excludes the first entry related to the average from the system that was booted:

```
[root@server ~]# iostat -d 10 3 -y
Linux 4.18.0-305.30.1.el8_4.x86_64 (server)     14/08/22        _x86_64_        (32 CPU)

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda             871,80        15,60     77581,15        156     775811
sdb               0,00         0,00         0,00          0          0

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda             491,80        30,80      6878,05        308      68780
sdb               0,00         0,00         0,00          0          0

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda             332,90        19,20      4045,10        192      40451
sdb               0,00         0,00         0,00          0          0
```

Figure 6.33: *Output example for the command iostat*

Command *iotop*

This utility is similar to the *top* command but related to the Input/Output only. This tool should be installed (*iotop* package). *Figure 6.34* shows the aspect of the application:

```
Total DISK READ :      22.31 K/s | Total DISK WRITE :      11.15 M/s
Actual DISK READ:      22.31 K/s | Actual DISK WRITE:      11.15 M/s
    TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
2137334 be/4 qemu        0.00 B/s   52.05 K/s  0.00 %  1.50 % qemu-kvm -na~p=on [worker]
2137341 be/4 qemu        0.00 B/s   11.15 K/s  0.00 %  0.84 % qemu-kvm -na~outp [worker]
2137325 be/4 qemu        0.00 B/s   26.02 K/s  0.00 %  0.73 % qemu-kvm -na~p=on [worker]
2137313 be/4 qemu        0.00 B/s  156.14 K/s  0.00 %  0.41 % qemu-kvm -na~p=on [worker]
2137359 be/4 qemu        0.00 B/s    0.00 B/s  0.00 %  0.14 % qemu-kvm -na~outp [worker]
```

Figure 6.34: *Output example for the command iotop*

The information shown is important to detect which application is performing more input/output operations and the speed of the read/write tasks.

Command *atop*

This command, described previously for *CPU* usage, includes a great overview about the local and remote storage usage in the system. *Figure 6.35* illustrates the information shown as follows:

```
SWP |  tot    32.0G  | free   32.0G  |  swcac   0.0M  |  vmcom  17.5G |  vmlim  47.5G  |
LVM |  4ab0-a3eb-6a  | busy      6%  |  read       0  |  write    232 |  avio 2.70 ms  |
LVM |  RHELCSB-Home  | busy      6%  |  read       0  |  write    175 |  avio 3.19 ms  |
LVM |  RHELCSB-Root  | busy      1%  |  read       0  |  write      3 |  avio 22.7 ms  |
DSK |       nvme0n1  | busy      6%  |  read       0  |  write    220 |  avio 2.85 ms  |
```

Figure 6.35: *Example disk information in command atop*

The following fields are shown in *table 6.3* (the bigger size terminal shows all of them):

Field	Description
busy	the portion of time that the unit was busy handling requests
read	the number of read requests issued
write	the number of write requests issued
discrd	the number of discard requests issued
KiB/r	the number of Kibibytes per read
KiB/w	the number of Kibibytes per write
KiB/d	the number of Kibibytes per discard
MBr/s	the number of Mebibytes per second throughput for reads
MBw/s	the number of Mebibytes per second throughput for writes
avq	the average queue depth
avio	the average number of milliseconds needed by a request for seek, latency, and data transfer.

Table 6.3: *Field description for the command atop*

Command *smartctl*

Modern disks include a monitoring system called **Self-Monitoring, Analysis and Reporting Technology** (**S.M.A.R.T**) containing different indicators to anticipate hardware failures. The main command is *smartctl*, where specifying the disk would show extended information related to the disk. *Figures 6.36* and *6.37* is using different options to filter the information shown, to reduce the output's lines:

- *Figure 6.36* shows disk information, including model/serial numbers and firmware version:

```
[root@server2 ~]# smartctl -i /dev/nvme0n1
smartctl 7.1 2020-04-05 r5049 [x86_64-linux-4.18.0-372.9.1.el8.x86_64] (local build)
Copyright (C) 2002-19, Bruce Allen, Christian Franke, www.smartmontools.org

=== START OF INFORMATION SECTION ===
Model Number:                       INTEL SSDPEKKF512G8L
Serial Number:                      PHHH84800BY5512H
Firmware Version:                   L15P
PCI Vendor/Subsystem ID:            0x8086
IEEE OUI Identifier:                0x5cd2e4
Controller ID:                      1
Number of Namespaces:               1
Namespace 1 Size/Capacity:          512.110.190.592 [512 GB]
Namespace 1 Formatted LBA Size:     512
Namespace 1 IEEE EUI-64:            5cd2e4 259140262f
Local Time is:                      Sun Aug 14 12:11:25 2022 CEST
```

Figure 6.36: *Output example for command smartctl*

- Showing the *SMART* information, which contains the data written and read, the temperature, and other information related to the space used, is shown in *figure 6.37*:

```
[root@server2 ~]# smartctl -A  /dev/nvme0n1
smartctl 7.1 2020-04-05 r5049 [x86_64-linux-4.18.0-372.9.1.el8.x86_64] (local build)
Copyright (C) 2002-19, Bruce Allen, Christian Franke, www.smartmontools.org

=== START OF SMART DATA SECTION ===
SMART/Health Information (NVMe Log 0x02)
Critical Warning:                   0x00
Temperature:                        33 Celsius
Available Spare:                    100%
Available Spare Threshold:          12%
Percentage Used:                    8%
Data Units Read:                    66.386.661 [33,9 TB]
Data Units Written:                 58.888.607 [30,1 TB]
Host Read Commands:                 567.421.451
Host Write Commands:                1.875.052.632
Controller Busy Time:               68.306
Power Cycles:                       2.921
Power On Hours:                     11.370
Unsafe Shutdowns:                   161
Media and Data Integrity Errors:    0
Error Information Log Entries:      0
Warning  Comp. Temperature Time:    0
Critical Comp. Temperature Time:    0
```

Figure 6.37: *Output example for command smartctl*

By using the option *-l error* will show the errors detected by the *SMART* monitor system.

Commands *fio* and *hdparm*

These commands can be used to query the read/writing speed for a storage device, being useful tools to detect problems with disks. The command **`hdparm`** with the options *-tT* output is shown in *figure 6.38*:

```
[root@server2 ~]# hdparm -tT /dev/nvme0n1

/dev/nvme0n1:
 Timing cached reads:   10064 MB in  1.99 seconds = 5047.98 MB/sec
 Timing buffered disk reads: 3074 MB in  3.00 seconds = 1024.60 MB/sec
```

Figure 6.38: *Output example for command hdparm*

The command **`fio`** is a flexible Input/Output test*er*. It is more advanced than **`hdparm`** because it allows us to specify the number of threads and processes to perform the I/O actions. *Figure 6.39* (output truncated) shows the command in action to test random read and write using four jobs:

```
[root@server2 ~]# fio --randrepeat=1 --ioengine=libaio --direct=1 --gtod_reduce=1 --name=test --filename=test.fio
 --bs=4k --iodepth=64 --size=4G --readwrite=randrw --numjobs=4
test: (g=0): rw=randrw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=64
...
fio-3.19
Starting 4 processes
test: Laying out IO file (1 file / 4096MiB)
Jobs: 4 (f=4): [m(4)][96.2%][r=384MiB/s,w=383MiB/s][r=98.2k,w=98.1k IOPS][eta 00m:02s]

Run status group 0 (all jobs):
   READ: bw=164MiB/s (171MB/s), 40.8MiB/s-41.0MiB/s (42.8MB/s-42.0MB/s), io=8192MiB (8590MB), run=49990-50089msec
  WRITE: bw=164MiB/s (171MB/s), 40.9MiB/s-40.9MiB/s (42.9MB/s-42.9MB/s), io=8192MiB (8590MB), run=49990-50089msec

Disk stats (read/write):
    dm-1: ios=2076608/2076846, merge=0/0, ticks=7030954/4020718, in_queue=11051672, util=99.53%, aggrios=2097185/
2100015, aggrmerge=0/0, aggrticks=7047507/4054918, aggrin_queue=11102425, aggrutil=98.54%
    dm-0: ios=2097185/2100015, merge=0/0, ticks=7047507/4054918, in_queue=11102425, util=98.54%, aggrios=2097185/
2098583, aggrmerge=0/1459, aggrticks=6444743/3063238, aggrin_queue=9507982, aggrutil=98.31%
  nvme0n1: ios=2097185/2098583, merge=0/1459, ticks=6444743/3063238, in_queue=9507982, util=98.31%
```

Figure 6.39: *Output example for command fio*

Commands *df* and LVM commands

Command **`df`** described in *Chapter 3, Using the Command Line Interface*, shows the disk free and usage. If the system is using *LVM* (*Logical Volume Manager*), the commands to check the status are as follows:

- **`vgdisplay`**: Display volume group information as shown in *figure 6.40*:

```
[root@server2 ~]# vgdisplay  2>/dev/null
  --- Volume group ---
  VG Name               RHELCSB
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  11
  VG Access             read/write
```

```
VG Status              resizable
MAX LV                 0
Cur LV                 3
Open LV                3
Max PV                 0
Cur PV                 1
Act PV                 1
VG Size                499,98 TiB
PE Size                4,00 MiB
Total PE               131067903
Alloc PE / Size        59392 / 232,00 GiB
Free  PE / Size        131008511 / <499,76 TiB
VG UUID                wNqZcz-C2xE-xtfV-fFJn-Oe6n-QA5W-3Q13oz
```

Figure 6.40: *Output example for command vgdisplay*

- **`pvdisplay`**: Displays various attributes of the physical volume(s), as shown in *figure 6.41*:

```
[root@server2 ~]# pvdisplay  2>/dev/null
  --- Physical volume ---
  PV Name               /dev/mapper/luks-8768059b-73c3-4ab0-a3eb-6ab69f97380e
  VG Name               RHELCSB
  PV Size               499,98 TiB / not usable 3,00 MiB
  Allocatable           yes
  PE Size               4,00 MiB
  Total PE              131067903
  Free PE               131008511
  Allocated PE          59392
  PV UUID               WfAU4u-qd07-cyEW-L8cV-y4KA-xnRe-psIRNm
```

Figure 6.41: *Output example for command pvdisplay*

- **`lvdisplay`**: Display information about a logical volume, as shown in *figure 6.42*:

```
[root@server2 ~]# lvdisplay 2>/dev/null
  --- Logical volume ---
  LV Path                /dev/RHELCSB/Home
  LV Name                Home
  VG Name                RHELCSB
```

```
LV UUID                XkTEn3-dReS-F7TZ-4WTt-YjxN-Ffmt-OJA6ZN
LV Write Access        read/write
LV Creation host, time localhost, 2019-05-30 08:57:28 +0200
LV Status              available
# open                 1
LV Size                150,00 GiB
Current LE             38400
Segments               3
Allocation             inherit
Read ahead sectors     auto
- currently set to     256
Block device           253:2
```

Figure 6.42: *Output example for command lvdisplay*

Monitoring network resources

Networking is one of the core components to monitor to avoid bottlenecks. This section will focus on the statistics of the network devices to check the bandwidth usage and detect packet losses. This section is focused on general statistics, and the upcoming chapter related to networking configuration will include tools to check the status of connections end to end.

Command **`lshw`** gives information about the ethernet devices in the system, as *figure 6.43* illustrate for a physical system:

```
[root@server ~]# lshw -c network -short
H/W path         Device     Class          Description
=====================================================
/0/100/1/0       eth0       network        Ethernet Controller 10-Gigabit X540-AT
/0/100/1/0.1     eth1       network        Ethernet Controller 10-Gigabit X540-AT
/0/100/2/0       eth2       network        Ethernet Controller 10-Gigabit X540-AT
/0/100/2/0.1     eth3       network        Ethernet Controller 10-Gigabit X540-AT
```

Figure 6.43: *Output example for command lshw*

Directory **`/sys/class/net/`** contains a list of links for each of the physical and virtual interfaces. Each link points to a directory that contains information about the interface, such as configurations and statistics. *Figure 6.44* shows the files inside of the directory, which **`/sys/class/net/eth0`** points it (**`/sys/devices/pci0000:00/0000:00:01.0/0000:01:00.0/net/eth0`***):*

```
[root@server eth0]# ls
addr_assign_type    dev_id                  link_mode         proto_down
address             dev_port                master            queues
addr_len            dormant                 mtu               speed
broadcast           duplex                  name_assign_type  statistics
brport              flags                   netdev_group      subsystem
carrier             gro_flush_timeout       operstate         testing
carrier_changes     ifalias                 phys_port_id      tx_queue_len
carrier_down_count  ifindex                 phys_port_name    type
carrier_up_count    iflink                  phys_switch_id    uevent
device              ixgbe-mdio-0000:01:00.0 power             upper_baremetal
```

Figure 6.44: *Example content for a network device in /sys/class/net/*

Directory **`statistics/`** contains information about the received and traffic information for the device.

Command *ethtool*

This command allows one to query or control network and hardware settings. With this command, it is possible to enable some features for Ethernet devices, such as **`autonegotation`** or the default speed. It requires as an argument the interface, and by default, without specifying any option, it will display useful information related to the network interface, as shown in *figure 6.45*:

```
[root@server ~]# ethtool eth0
Settings for eth0:
        Supported ports: [ TP ]
        Supported link modes:   100baseT/Full
                                1000baseT/Full
                                10000baseT/Full
        Supported pause frame use: Symmetric
        Supports auto-negotiation: Yes
        Supported FEC modes: Not reported
        Advertised link modes:  100baseT/Full
                                1000baseT/Full
                                10000baseT/Full
        Advertised pause frame use: Symmetric
        Advertised auto-negotiation: Yes
        Advertised FEC modes: Not reported
        Speed: 1000Mb/s
        Duplex: Full
        Auto-negotiation: on
        Port: Twisted Pair
        PHYAD: 0
        Transceiver: internal
        MDI-X: Unknown
        Supports Wake-on: umbg
        Wake-on: g
        Current message level: 0x00000007 (7)
                               drv probe link
        Link detected: yes
```

Figure 6.45: *Output example for command ethtool*

The output includes some important information:

- The current speed for the interfaces.
- The support link modes for the interface.

- The advertised link modes are received from the switch where the interface is connected.
- The *Duplex* configuration and if the *Auto-negotiation* is enabled.
- If the interface has *Link detected*.

Command *nmon*

Command **`nmon`** is a popular tool to obtain general statistics about resources in the system and is specially used for networking. After executing the application, pressing the key *n* will show the networking statistics, as shown in *figure 6.46*:

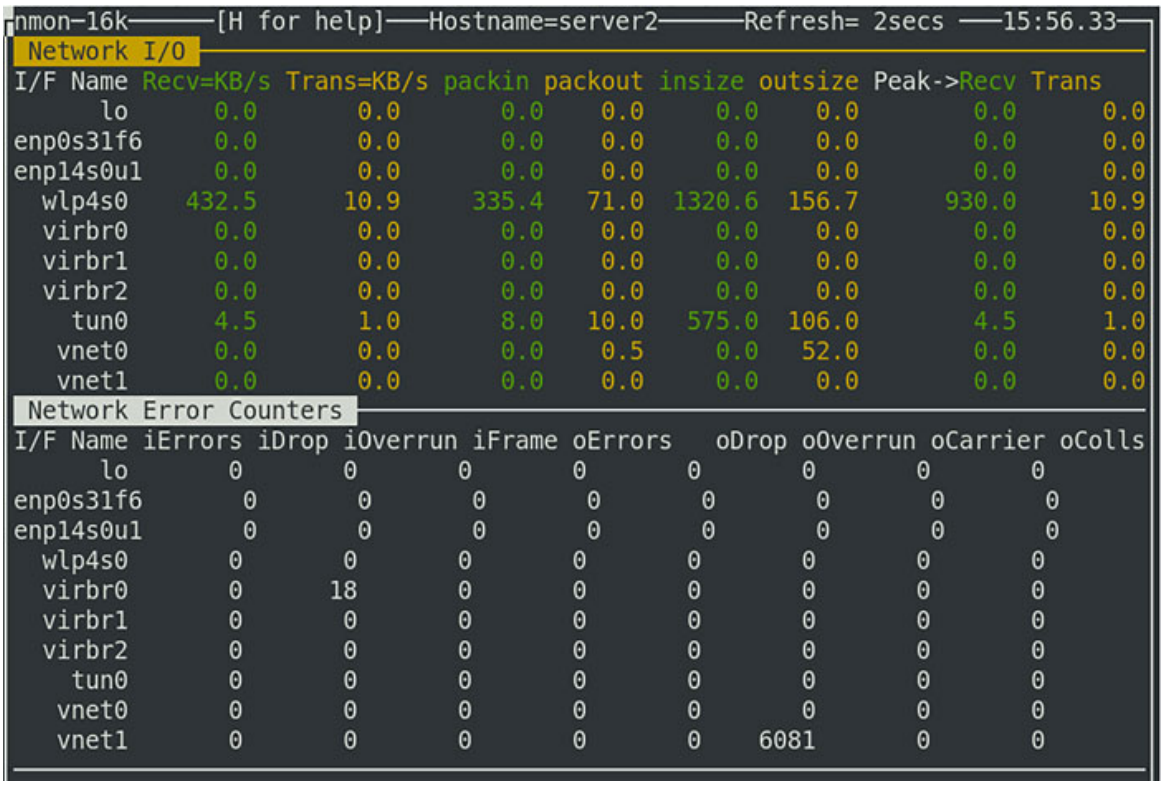


Figure 6.46: *Aspect example for command nmon*

Commands traceroute, tracepath, and mtr

Usually, troubleshooting network issues requires networking knowledge related to traffic flows. These three tools accept as an argument the destination IP to ensure all the intermediate gateways are working properly. *Figures 6.47*– *6.49* show the usage trying to reach the IP 8.8.8.8 (Google's DNS):

- Command **`traceroute`** (part of the package **`traceroute`**) prints the route packets trace to the network using ICMP protocol. Command **`traceroute6`**

can be used for *IPv6*, as shown in *figure 6.47*:

```
[root@server ~]# traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  169.254.163.26 (169.254.163.26)  0.555 ms  0.669 ms  0.862 ms
 2  ae107.ppr01.wdc04.networklayer.com (169.55.118.184)  0.851 ms  1.221 ms  0.73
5 ms
 3  86.76.3fa9.ip4.static.sl-reverse.com (169.63.118.134)  0.520 ms 80.76.3fa9.ip
4.static.sl-reverse.com (169.63.118.128)  0.314 ms 86.76.3fa9.ip4.static.sl-rever
se.com (169.63.118.134)  0.479 ms
 4  ae17.cbs02.eq01.wdc02.networklayer.com (169.45.18.232)  1.407 ms ae16.cbs02.e
q01.wdc02.networklayer.com (169.45.18.230)  1.346 ms ae17.cbs02.eq01.wdc02.networ
klayer.com (169.45.18.232)  1.351 ms
 5  ae29.bbr02.eq01.wdc02.networklayer.com (50.97.17.147)  0.738 ms  0.734 ms ae3
0.bbr01.eq01.wdc02.networklayer.com (50.97.17.149)  0.721 ms
 6  23.10.6132.ip4.static.sl-reverse.com (50.97.16.35)  0.745 ms 72.14.221.28 (72
.14.221.28)  0.502 ms 23.10.6132.ip4.static.sl-reverse.com (50.97.16.35)  0.775 m
s
 7  108.170.246.1 (108.170.246.1)  0.822 ms * *
 8  dns.google (8.8.8.8)  0.709 ms  0.738 ms  0.454 ms
```

Figure 6.47: *Output example for command traceroute*

- Commands **`tracepath`** and **`tracepath6`** (part of the package **`iputils`**) trace a path to a network, discovering the **Maximum Transmission Unit** (**MTU**) along this path. Messages like *no reply* indicates that the router or switch through the path is not providing information. Indicating a random *UDP*, the information will be shown similar as in *figure 6.48*:

```
[root@server2 ~]# tracepath -n 8.8.8.8 -p 33434
 1?: [LOCALHOST]                      pmtu 1492
 1:  192.168.1.1                                           1.029ms
 1:  192.168.1.1                                           0.677ms
 2:  no reply
 3:  no reply
 4:  212.166.147.22                                       13.179ms asymm  8
 5:  no reply
 6:  8.8.8.8                                              13.160ms reached
     Resume: pmtu 1492 hops 6 back 10
```

Figure 6.48: *Output example for command tracepath*

- Command **`mtr`** (part of package **`mtr`**) is a network diagnostic tool combining the functionality of *traceroute* and *ping* to investigate the network connection between the host and the destination host. *Figure 6.49* shows the example aspect indicating as argument *8.8.8.8*:

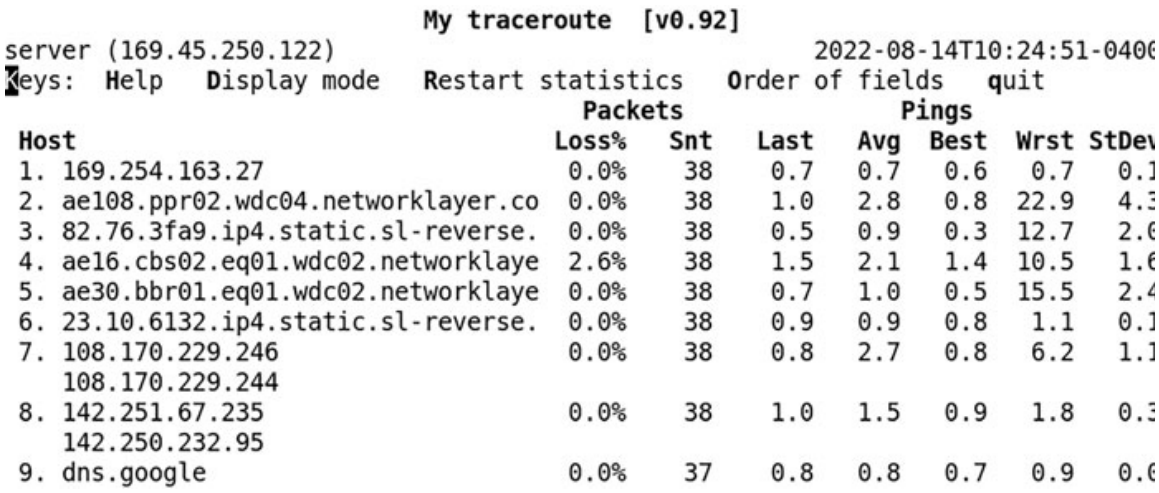


Figure 6.49: *Output example for command mtr*

- The big difference between *mtr* and *traceroute* is that *mtr* is interactive and is testing the connection constantly to detect if there is any package loss or any peak on the time to arrive at any of the path components.

Quotas and limits

One of the goals of a Linux system administrator is to protect the system, and that requires to set limits and quotas. So the system resources are not exhausted. Linux has two general ways to limit resources consumption by users:

- **Quotas:** Limits the disk space for users and groups. This can be used to limit the space a user can consume or limit the total number of files he can own.
- **Limits:** Possibility to limit different resources to the user, such as the number of open files, number of processes, or the memory used.

Quotas

Disk quotas are applied to a filesystem and to users and/or groups. A filesystem needs to be configured to track the files and the size that each user and group is consuming. It is possible to limit the total size and the number of files for a user or group, having two limits:

1. **Soft limit**: Indicates the limit when a grace period would be in place. For example, if the *soft limit* is 1 GB and the *grace period* is two days, the user would be able to be over that limit for two days. After the *grace period*, the

user will not be able to add more files till they delete the files to be below the limit.

2. **Hard limit**: The user will not be able, in any case, to have more than the limit specified.

Before quotas are used in a system, the user or the group quotas have to be enabled in the filesystem, which is required to be configured for quotas. This is not enabled by default. For that, it is needed to open the file **`/etc/fstab`** and, in the fourth column, add the words **`usrquota`** and/or **`grpquota`**, as is shown in *figure 6.50*:

```
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
# / was on /dev/ubuntu-vg/ubuntu-lv during curtin installation
/dev/disk/by-id/dm-uuid-LVM-BZrwBMRT0NAcX1Qc2onWgQRHUKMZmipJGLbA2z6NSilzWCGpNy88g
55A5GSEq1fi / ext4 defaults,usrquota,grpquota  0 1
```

Figure 6.50: *File /etc/fstab content example*

After this, the filesystem is required to be remounted to apply the new configuration. In this case, it is configured in the filesystem mounted in **`root (/): mount -o remount,rw /`**. It is possible to run the command **`mount`** without any option to ensure the quotas are enabled, as shown in *figure 6.51*:

```
root@ubuntu:~# mount -o remount,rw /
root@ubuntu:~# mount |grep quota
/dev/mapper/ubuntu--vg-ubuntu--lv on / type ext4 (rw,relatime,quota,usrquota,grpquota)
```

Figure 6.51: *Using command mount to remount and check status*

After enabling the quotas to attribute to the filesystem, it is required to create the files **`aquota.user`** and/or **`aquota.group`** to keep track of configuration and usage for users and groups. The way to create these files is through the command **`quotacheck`** with the options *-u (for users), -g (for groups)*, and *-m* (do not remount as read-only for this task). After installing the *quota* package, the command *quotacheck* would be available, as shown in *figure 6.52*:

```
root@ubuntu:~# quotacheck -ugm /
root@ubuntu:~# ls -l /aquota*
-rw------- 1 root root 9216 Aug 14 14:46 /aquota.group
-rw------- 1 root root 8192 Aug 14 14:46 /aquota.user
```

Figure 6.52: *Run command quotacheck and ensure needed files are created*

After the files are created, the quotas need to be enabled using the command **`quotaon`**, as shown in *figure 6.53*:

```
root@ubuntu:~# quotaon -v /
/dev/mapper/ubuntu--vg-ubuntu--lv [/]: group quotas turned on
/dev/mapper/ubuntu--vg-ubuntu--lv [/]: user quotas turned on
```

Figure 6.53: *Enable quotas for a filesystem*

After the quotas are enabled in the system, there are two ways to set quotas for one user or group:

1. Command **`edquota`**: it will open an editor to set the quotas for the user (*-u user*) or the group *(-g group*).
2. Command **`setquota`**: it will update the quotas for a user or group.

Figure 6.54 shows how to use *setquota* to set a quota for a regular user (200MB as soft limit, 200M as hard limit for space usage and without the number of file limits) and using the command **`repquota`** to list the current status:

```
root@ubuntu:~# setquota -u agonzalez 200M 220M 0 0 /
root@ubuntu:~# repquota -s /
*** Report for user quotas on device /dev/mapper/ubuntu--vg-ubuntu--lv
Block grace time: 7days; Inode grace time: 7days
                        Space limits                File limits
User            used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --   6352M      0K      0K           138k     0     0
man       --   1440K      0K      0K            146     0     0
_apt      --     24K      0K      0K              4     0     0
systemd-network --    276K      0K      0K              5     0     0
systemd-timesync --      4K      0K      0K              2     0     0
pollinate --      4K      0K      0K              2     0     0
syslog    --   1676K      0K      0K             17     0     0
tss       --      4K      0K      0K              1     0     0
landscape --      8K      0K      0K              3     0     0
agonzalez --    160K    200M    220M             61     0     0
```

Figure 6.54: *Set quota and report active quotas*

If the user tries to have more than 220MB (*hard* limit), the system will not allow to create more files. *Figure 6.55* uses a command called *dd* to generate a file of size 300MB:

```
agonzalez@ubuntu:~$ dd if=/dev/random of=file300M bs=4M count=60
dd: error writing 'file300M': Disk quota exceeded
55+0 records in
54+0 records out
230522880 bytes (231 MB, 220 MiB) copied, 9.70586 s, 23.8 MB/s
```

Figure 6.55: *Using command dd to test disk quota*

Limits

Linux allows limiting the resources a user can use during its session. The main file defining those limits is **`/etc/security/limits.conf`**, which contains the following fields:

- *domain*: it can be a user name, a group name with a prefix at (@), an asterisk (*) for all users, or a *uid* or *gid* range separated by colon (:).
- *type*: Values *soft* and *hard* are accepted types.
- *item*: one element to limit, from *table 6.4*

Limit	Description
core	limits the core file size (KB)
data	max data size (KB)
fsize	maximum filesize (KB)
memlock	max locked-in-memory address space (KB)
nofile	max number of open files
rss	max resident set size (KB)
stack	max stack size (KB)
cpu	max CPU time (MIN)
nproc	max number of processes (see note given as follows)
as	address space limit (KB)
maxlogins	max number of logins for this user
maxsyslogins	max number of logins on the system
priority	the priority to run user process with
locks	max number of file locks the user can hold
sigpending	max number of pending signals
msgqueue	max memory used by POSIX message queues (bytes)

nice	max nice priority allowed to raise to values: [–20, 19]
rtprio	max realtime priority

Table 6.4: *Limit fields and description information*

A user can check the limits applied using the command **`ulimit`**, as is shown in *figure 6.56*:

```
agonzalez@ubuntu:~$ ulimit -a
real-time non-blocking time  (microseconds, -R) unlimited
core file size              (blocks, -c) 0
data seg size               (kbytes, -d) unlimited
scheduling priority                 (-e) 0
file size                   (blocks, -f) unlimited
pending signals                     (-i) 15251
max locked memory           (kbytes, -l) 502268
max memory size             (kbytes, -m) unlimited
open files                          (-n) 1024
pipe size                (512 bytes, -p) 8
POSIX message queues         (bytes, -q) 819200
real-time priority                  (-r) 0
stack size                  (kbytes, -s) 8192
cpu time                   (seconds, -t) unlimited
max user processes                  (-u) 15251
virtual memory              (kbytes, -v) unlimited
file locks                          (-x) unlimited
```

Figure 6.56: *Output example for command ulimit*

In *figure 6.57*, an administrator limits the number of user processes can have. Users can set their own limits with the built-in shell command called **`ulimit`**, but always set a value lower than the default or the specific one for that user:

```
root@ubuntu:~# echo "agonzalez        hard    nproc           10" >> /etc/security/limits.conf
root@ubuntu:~# su - agonzalez
agonzalez@ubuntu:~$ ulimit -u
10
agonzalez@ubuntu:~$
agonzalez@ubuntu:~$ ulimit -u 5
agonzalez@ubuntu:~$ ulimit -u
5
```

Figure 6.57: *Setting limit for a user and setting the limit itself*

If the user, after setting its own limit to five processes, tries to run more than that number, they will receive the error, as shown in *figure 6.58*:

```
agonzalez@ubuntu:~$ sleep 50&
-bash: fork: retry: Resource temporarily unavailable
```

Figure 6.58: *Testing user limits*

Conclusion

In this chapter, we understood how to review the resources. It is important to avoid disruptions in the system. Linux provides both high- and low-level tools to obtain information about the resources to be able to predict future issues. Limiting the resources that regular users can consume is a required task to avoid full disk or overload of CPU or memory.

Key facts

- Multiple commands are available to obtain information about the resource usage.
- CPU load is represented in three average intervals: 1, 5 and 15 minutes.
- It is possible to set quotas on Linux for user and groups.
- Linux offers the possibility to limit the resources that a user or group can consume.

Questions

1. What command gives the best information related to *CPU* information?

    a. lscpu

    b. cpuinfo

    c. cpuid

2. Having only one CPU with one core and without threads, what means load 0.55?

    a. 155% usage

    b. 5.5% usage

    c. 55% usage

3. What file contains memory information usage and statistics?

    a. /dev/meminfo

b. /sys/meminfo
c. /proc/meminfo

4. What command is used to obtain information from *S.M.A.R.T*?

    a. smartinfo
    b. smartctl
    c. smartdump

5. What command is useful to have real-time statistics information about the path to reach a destination?

    a. mtr
    b. tracepath
    c. traceroute

6. What command is used to create a report about quotas?

    a. repquota
    b. showquota
    c. infoquota

7. A user can use *ulimit* to display and set their own limits.

    a. True
    b. False

Answers

1. a
2. c
3. c
4. b
5. a
6. a
7. a

CHAPTER 7

Network Configuration

Introduction

Networking is one of the core parts of enterprises. It involves multiple elements, such as servers, switches, routers, and cables. Understanding how the traffic flows and the possible configurations in Linux is important to provide a service in an efficient way to reduce costs, bottlenecks, and downtimes.

Linux servers allow simple configurations and advanced ones. This chapter will cover how to configure. Networking in different Linux distributions and advanced features such as aggregation, bridges, and Virtual LANs.

Structure

In this chapter, the following topics will be discussed:

- Network introduction
- Basic network configuration
- Routing
- Advanced network configuration

Network introduction

Modern systems depend on networks for most of the tasks, such as installing new software, updating the system, or offering services locally or remotely. Linux, as the most secure operating system, has different abstraction layers between the hardware, Kernel space, and user space. *Figure 7.1* features the various Linux operating system layers:

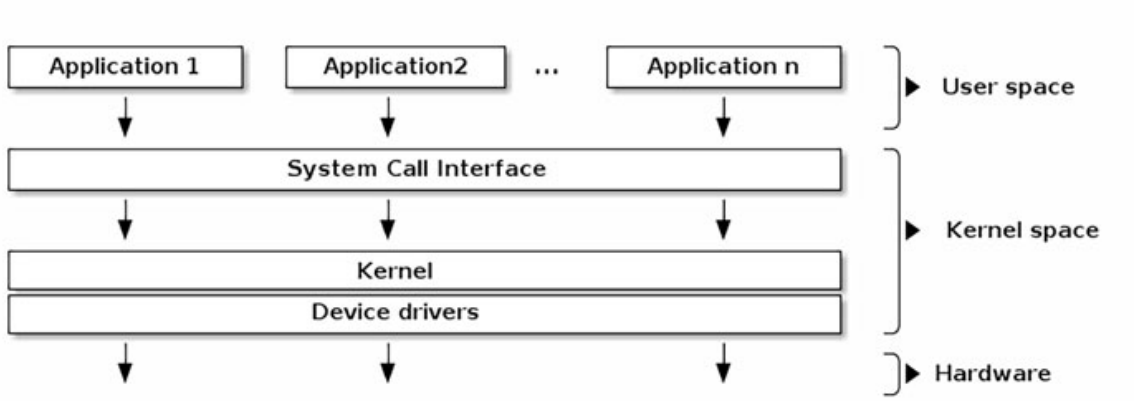


Figure 7.1: *Linux operating system layers. Source: Linux Kernel*

As a Linux administrator, it is important to understand the basic concepts of Networking, from how the traffic is managed by the physical devices to how the data is presented to the end user. The **Open Systems Interconnection** *Model* (*O***pen Systems Interconnection** *Model*) describes the universal standard communication. This model defines the following seven abstraction layers:

- **Physical layer (Layer 1):** Responsible for the transmission and reception of the data in a device, such as an Ethernet device in a system, a cable, or a port in a Router.
- **Data link layer (Layer 2):** Provides data transfer between different nodes. For example, a communication between two servers in the same local network.
- **Network layer (Layer 3):** Transfers data from (or to) one node from a network to a different network. This requires a node to route the traffic between different Layer 2 (data link).
- **Transport layer (Layer 4):** Data transmission using protocols, such as UDP or TCP.
- **Session layer (Layer 5):** Maintains and synchronizes connection between two nodes.
- **Presentation layer (Layer 6):** Formats and translates the data to be used, such as data encryption/decryption or compression/decompression.
- **Application layer (Layer 7):** The application where the user interacts.

Figure 7.2 illustrates the traffic flow through the different layers in the OSI Model:

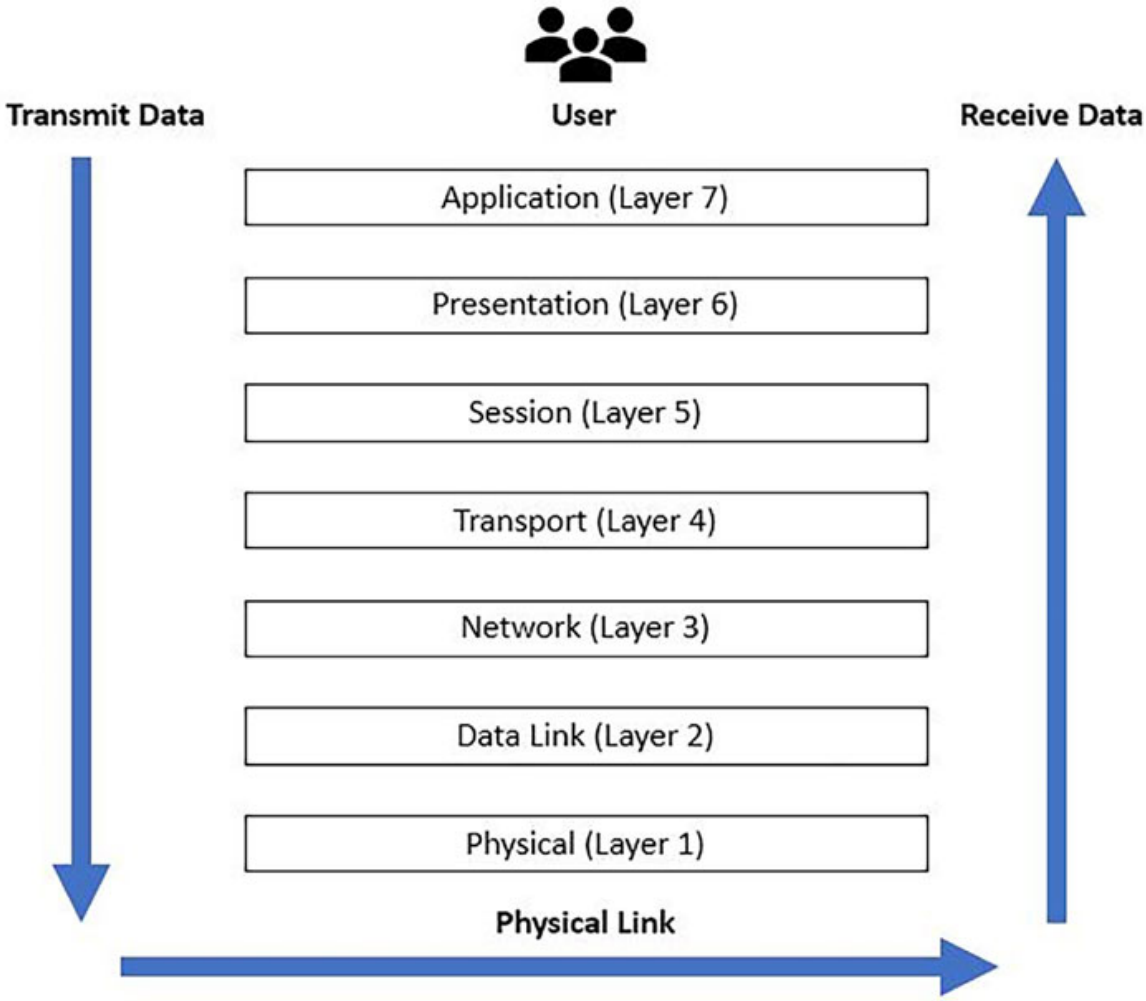


Figure 7.2: *OSI Model layer traffic flow. Source: Wikipedia*

Networking evolved in the last decades, from communication to the adoption of new technologies. First communications were mainly between physical servers, through switches and routers. The traffic dominant was a *north-south* data flow, where the systems were communicating with devices outside the local network.

Migration of the services to virtual machines, with the popularity of virtualization, introduced new challenges related to communication, thus, increasing the traffic *east-west*, which indicates the communication between different services inside of the same network.

Adoption of the containers increased the traffic between the systems, and advanced network configuration and advanced network implementations, such as S*oftware Defined Networks* and *tunnel protocols,* are knowledge required nowadays when a server is configured.

Another important transformation is part of large enterprises moving to a new network architecture called *Spine Leaf*, which increases the efficiency of the traffic and reduces the possible bottlenecks caused by the traditional architecture. *Figure 7.3* shows the difference between the traditional architecture, which has three layers (access, aggregation, and core) with limitations of the parallel traffic, and the new architecture, with a mesh connection with high bandwidth and low-latency features:

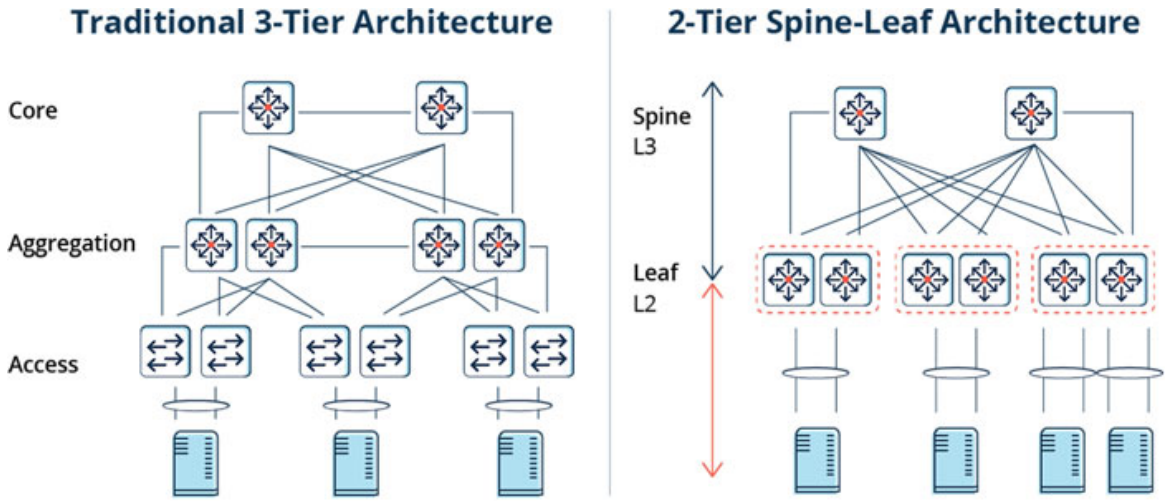


Figure 7.3: *Traditional and spine leaf architectures. Source: Aruba Networks*

Another important change in the last few years was the introduction of IPv6, the most recent version of the *Internet Protocol*. This new version uses 128-bit addresses instead of the 32-bit ones on IPv4. The use of IPv6 is still limited in local networks, but the adoption of ISP and large enterprises are a reality.

Physical layer (Layer 1)

Chapter 6, Monitoring System Resources, described using the command `lshw` to list the available Ethernet devices on the system. The command `ethtool` was introduced in the same chapter to obtain information about the status, such as if the link was detected and the possible speeds available.

The command `ip` replaces the operations to the traditional, and the deprecated command `ifconfig` in Linux replaces the operations to list information about the interfaces. Specifying as an argument, the word *link* lists the network interface's physical information as illustrated in *figure 7.4*:

```
root@ubuntu:~# ip --color=auto link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT gr
oup default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode
DEFAULT group default qlen 1000
    link/ether 52:54:00:ce:8b:b8 brd ff:ff:ff:ff:ff:ff
3: enp7s0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
    link/ether 52:54:00:07:3a:f8 brd ff:ff:ff:ff:ff:ff
```

Figure 7.4: *Output example for the command ip link*

The output for the command **`ip link`** contains important information. Linux distributions contain a local interface called *lo*, which is used for internal communications within the system, such as TCP/UDP connections. Physical interfaces have the format *en (Ethernet device) + p[PCI bus number] + s[Slot number]* in modern systems instead ethX (where X was a number starting in 0) in the past.

The output will indicate if the interface is UP (link is detected and the interface is active in the system) or DOWN (link undetected or the interface is not active in the system), the MTU (*maximum transmission unit*), which usually defaults to 1,500 and the *MAC address,* which identifies an individual network device.

The *MAC address* is an important value to configure the interface in the network, such as assigning a static IP when *Dynamic Host Configuration Protocol* (*DHCP*) is used or for the network installation using *Preboot Execution Environment* (*PXE*).

Communication inside a local network between nodes requires to know the MAC of the destination node. To know the *MAC* address of one node, Linux sends a packet to the network asking who is the owner of the destination IP address. When it obtains the information, it stores the relation in a table that contains the mapping between the IP address and the *MAC* address. The argument *neigh* (which replaces the historical command ***arp***) shows the table information as *figure 7.5* illustrates:

```
root@ubuntu:~# ip --color neigh
192.168.122.1 dev enp1s0 lladdr 52:54:00:81:84:22 REACHABLE
192.168.122.145 dev enp1s0 lladdr 52:54:00:a2:cb:2a REACHABLE
```

Figure 7.5: *Output example for the command ip neigh*

Data link layer (Layer 2)

Command **`ip`** can be used to show the Layer 2 information, the IP address, and the network mask. Adding the argument *address* (or *a*) shows the assigned addresses

for the interface, as *figure 7.6* illustrates:

```
root@ubuntu:~# ip --color=auto address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:ce:8b:b8 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.101/24 metric 100 brd 192.168.122.255 scope global dynamic enp1s0
       valid_lft 3414sec preferred_lft 3414sec
    inet6 fe80::5054:ff:fece:8bb8/64 scope link
       valid_lft forever preferred_lft forever
3: enp7s0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 52:54:00:07:3a:f8 brd ff:ff:ff:ff:ff:ff
```

Figure 7.6: *Output example for the command ip address*

The interface *lo* has always assigned and configured the IP 127.0.0.1 with the network mask 255.0.0.0 (/8). The preceding *figure 7.6*, shows an interface named *enp1s0* configured with the IP *192.168.122.101* with the netmask *255.255.255.0* (*/24*), and the *broadcast IP* is *192.168.122.255*.

The interface named *enp1s0* has an assigned IPv6 local address (starting with *fe80*) with a subnet mask */64*. *Figure 7.7* shows an example of an interface with an IPv6 address configured:

```
root@lab:~# ip --color -6 address show  dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 state UP qlen 1000
    inet6 2001:41d0:1:ee26::1/128 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::225:90ff:fe22:424a/64 scope link
       valid_lft forever preferred_lft forever
```

Figure 7.7: *Output example for the command ip address with IPv6 address*

Network layer (Layer 3)

Layer 3 requires to have information on how to access networks that are not the local ones. This includes a default gateway to access outside of the network, and specific gateways to access to other networks if needed. The command **`ip`** with the argument **`route`** lists the current table of destinations and gateways. *Figure 7.8* shows an example:

```
root@ubuntu:~# ip --color route
default via 192.168.122.1 dev enp1s0 proto dhcp src 192.168.122.101 metric 100
192.168.122.0/24 dev enp1s0 proto kernel scope link src 192.168.122.101 metric 100
```

Figure 7.8: *Output example for the command ip route*

In the example shown in *figure 7.8*, the default gateway is *192.168.122.1*. It would be used as a hop for all communications whose destination is not *192.168.22.0/24*. In case the communication is to an IP in the local network, the gateway is not involved, and the communication is direct. *Figure 7.9* shows an example of IPv6:

```
root@lab:~# ip --color -6 route
2001:41d0:1:ee26::1 dev eth0 proto kernel metric 256 pref medium
2001:41d0:1:eeff:ff:ff:ff:ff dev eth0 metric 1024 pref medium
fe80::/64 dev eth0 proto kernel metric 256 pref medium
default via 2001:41d0:1:eeff:ff:ff:ff:ff dev eth0 metric 1024 pref medium
```

Figure 7.9: *Output example for the command ip route with IPv6*

The command **`ip route`** allows a useful extra argument called *get* and the IP to obtain the information on how to access it. *Figure 7.10* shows two examples, the first one about an IP outside of the network and the second one about an IP in the local network:

```
root@ubuntu:~# ip --color route get 8.8.8.8
8.8.8.8 via 192.168.122.1 dev enp1s0 src 192.168.122.101 uid 0
    cache
root@ubuntu:~# ip --color route get 192.168.122.241
192.168.122.241 dev enp1s0 src 192.168.122.101 uid 0
    cache
```

Figure 7.10: *Output example for the command ip route get*

As described in *Chapter 6, Monitoring System Resources*, there are several tools to check the traffic hops from the system to a destination IP, such as *traceroute*, *tracepath,* or *mtr*. The protocol **Internet Control Message Protocol** (**ICMP**) is located in Layer 3; this protocol is used by the commands mentioned previously and by the popular command **`ping`**. This command is useful to check connectivity to local and remote IPs.

Transport layer (Layer 4)

This layer depends on protocols for the transmission of data. The transport protocols can be stateful, for example, **Transmission Control Protocol** (**TCP**), or stateless such as **User Datagram Protocol** (**UDP***)*. The stateful connection ensures that the data is sent correctly and waits for the transmission to be completed. An example of a stateful connection is an HTTP request. A stateless connection sends the data and waits for an answer, but without ensuring communication. An example of a stateless connection is a DNS request.

The Transport layer relies on IP addresses and ports to initiate a connection, for which specifying the destination IP and the destination port is needed. A random port in the local system will be assigned for the communication. A service will be listening for new connections in an IP and in a defined port. The port ranges in Linux (different than the ones suggested by the **Internet Assigned Numbers Authority***)* are defined in *table 7.1*:

Range	Description
0-1023	*System ports* (well-known ports). Only services/applications executed by system users can listen to these ports.
1024-32767	Unprivileged port for the rest of the services
32768-60999	Dynamic and/or Private ports. Also known as ephemeral ports.

Table 7.1: *Port ranges description*

The file **`/etc/services`** contain a list of the known services and the port assigned. Linux provides a tool named **`sysctl`** to be able to query or modify the system parameters, including those port ranges. *Figure 7.11* shows the two parameters and the values:

```
root@ubuntu:~# sysctl -a |grep --color=never -e local_port -e ip_unprivileged
net.ipv4.ip_local_port_range = 32768    60999
net.ipv4.ip_unprivileged_port_start = 1024
```

Figure 7.11: *Output example for the command sysctl*

It is possible to list the currently active connections in the system using the command **`socket statistics`** (**`ss`**), which is replacing the traditional command **`netstat`**. *Figure 7.12* shows an example using the options *-t (--tcp* to filter TCP connections*)* and *-n* (*--numeric* to show numeric ports and IPs):

```
root@ubuntu:~# ss -tn
State  Recv-Q  Send-Q     Local Address:Port        Peer Address:Port   Process
ESTAB  0       0        192.168.122.101:22         192.168.122.1:50086
ESTAB  0       0        192.168.122.101:22       192.168.122.145:38134
ESTAB  0       0        192.168.122.101:22         192.168.122.1:46006
ESTAB  0       0        192.168.122.101:53530    192.168.122.145:22
```

Figure 7.12: *Output example for the command ss*

The preceding example in *figure 7.12* shows the following connections:

- Two connections from the IP *192.168.122.1* to the local system (which has IP *192.168.122.101)* to Port 22, which is the port of *SSH*. As *figure 7.11* indicates, each connection uses a random port from the range port set in the system.

- One connection from the IP *192.168.122.145* to the local system to Port 22.
- One connection originated from the local system (*192.168.122.101*), which the random port generated *53530*), to IP *192.168.122.145* and port 22.

Session, presentation, and application layers (Layers 5, 6, and 7)

These three layers are part of the software layers that initiate and service the data to the end user. An HTTPS connection, for example, using the command **`curl`** initiates a session to a Web server (Layer 5), handles the SSL connection and the data compression/decompression (Layer 6), and performs the visualization (Layer 7). *Figure 7.13* is an example of the command **`curl`** output:

```
root@ubuntu:~# curl -v https://ifconfig.me 2>&1| grep -e "onnect" -e "5.22"
* Connected to ifconfig.me (34.160.111.145) port 443 (#0)
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=0
* Connection #0 to host ifconfig.me left intact
5.224.10.27
```

Figure 7.13: *Output example for the command curl*

Basic network configuration

Network configuration in a Linux system, as other configurations are defined in plain text files. Historically, configuring those files was not a simple task and led to misconfiguration, thus causing downtimes. Recent versions of the distributions include high-level commands to manage the networking, helping system administrators to configure basic or advanced configurations. These solutions ensure that the syntax and the configuration are correct.

Red Hat distribution and derivatives, such as CentOS Stream or Rocky Linux, are using *NetworkManager* system daemon to manage the network devices and connections. Ubuntu distribution uses *Netplan* and *Systemd-networkd* to perform configuration based on *YAML* files.

Network configuration on Red Hat-based systems

On previous versions of Red Hat-based systems, the network configuration files were under the directory **`/etc/sysconfig/network-scripts/`** when manual configuration was used. The file with the prefix **`ifcfg-`** followed by the name of the interface, contains the configuration for that interface. The following code

shows an example of the content of the file **ifcfg-ens3** to configure the interface **ens3** using *DHCP*:

```
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=dhcp
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
NAME=ens3
UUID=02cc87e6-dc7c-4c9e-a530-f908611c7f5e
DEVICE=ens3
ONBOOT=yes
```

On newer versions, *NetworkManager* is using the directory **/etc/NetworkManager/system-connections** to store the configuration. The following code shows an example of a file named **enp1s0.nmconnection** to configure the interface **enp1s0** using *DHCP*:

```
[connection]
id=enp1s0
uuid=320f6572-672b-33c8-9014-b0fb087ae1a2
type=ethernet
autoconnect-priority=-999
interface-name=enp1s0
timestamp=1656276248
[ethernet]
[ipv4]
method=auto
[ipv6]
addr-gen-mode=eui64
method=auto
[proxy]
```

NetworkManager includes several commands to query, manage devices, and connections configurations. The command **nmcli** is used to perform actions in the command line, and the command **nmtui** is used to perform operations in a user interface in the console.

Command nmcli

This command-line tool is used for controlling *NetworkManager* and query the status. The command allows a first argument indicating the component to manage, where the common options to configure networking are described in *table 7.2*:

Argument	Description
general	General status and set hostname
connection	Manage connections
device	Manage devices managed by NetworkManager
monitor	Monitors NetworkManager changes

Table 7.2: *First argument options for command nmcli*

The argument *general* shows the general status of the system, as is shown in the following *figure 7.14*:

```
[root@rhel ~]# nmcli general
STATE      CONNECTIVITY  WIFI-HW  WIFI     WWAN-HW  WWAN
connected  full          enabled  enabled  enabled  enabled
```

Figure 7.14: *Output example for the command nmcli.*

The argument *connection,* without other arguments, lists the connections configured in the system and managed by *NetworkManager*. It is possible to use the arguments *connection show* followed by the name of the connection to list all the configured options, as is shown in *figure 7.15*:

```
[root@rhel ~]# nmcli connection
NAME    UUID                                  TYPE      DEVICE
enp1s0  320f6572-672b-33c8-9014-b0fb087ae1a2  ethernet  enp1s0
[root@rhel ~]# nmcli connection  show enp1s0 |grep IP4.
IP4.ADDRESS[1]:                         192.168.122.145/24
IP4.GATEWAY:                            192.168.122.1
IP4.ROUTE[1]:                           dst = 192.168.122.0/24, nh = 0.0.0.0, mt = 100
IP4.ROUTE[2]:                           dst = 0.0.0.0/0, nh = 192.168.122.1, mt = 100
IP4.DNS[1]:                             192.168.122.1
```

Figure 7.15: *Output example for the command nmcli.*

If possible, specify arguments to add connections, modify them or delete them, among other operations. *Figure 7.16* shows some of the common operations:

```
[root@rhel ~]# nmcli c add type ether con-name "test" ifname enp7s0 ip4 "10.10.0.22/24"
Connection 'test' (f67e84a5-a754-4268-92ff-0d501a70d9ac) successfully added.
[root@rhel ~]# nmcli connection
NAME    UUID                                  TYPE      DEVICE
enp1s0  320f6572-672b-33c8-9014-b0fb087ae1a2  ethernet  enp1s0
test    f67e84a5-a754-4268-92ff-0d501a70d9ac  ethernet  enp7s0
[root@rhel ~]# nmcli c modify test ip4 "10.10.10.23/24"
[root@rhel ~]# nmcli connection delete test
Connection 'test' (f67e84a5-a754-4268-92ff-0d501a70d9ac) successfully deleted.
```

Figure 7.16: *Output example for the command nmcli.*

Table 7.3 shows some examples of common operations configuring and operating with basic networking:

Action	Command
Add a new connection with static IP, gateway, and DNS	nmcli c a type ether con-name “test” ipv4.method manual ip4 “10.10.10.23/24” gw4 “10.10.10.1” ipv4.dns “10.10.10.2”
Deactivate a connection	nmcli connection down “test”
Activate a connection	nmcli connection up “test”
Appends a DNS server to an existing connection	nmcli con modify “test” +ipv4.dns 8.8.8.8
Removes an IP	nmcli con modify “test” -ipv4.addresses “10.10.10.4/24”

Table 7.3: *Example commands for nmcli*

Command nmtui

This utility helps the configuration and the query to the connections and devices in the system in a visual way. Running the command **`nmtui`** shows the menu illustrated in *figure 7.17*:

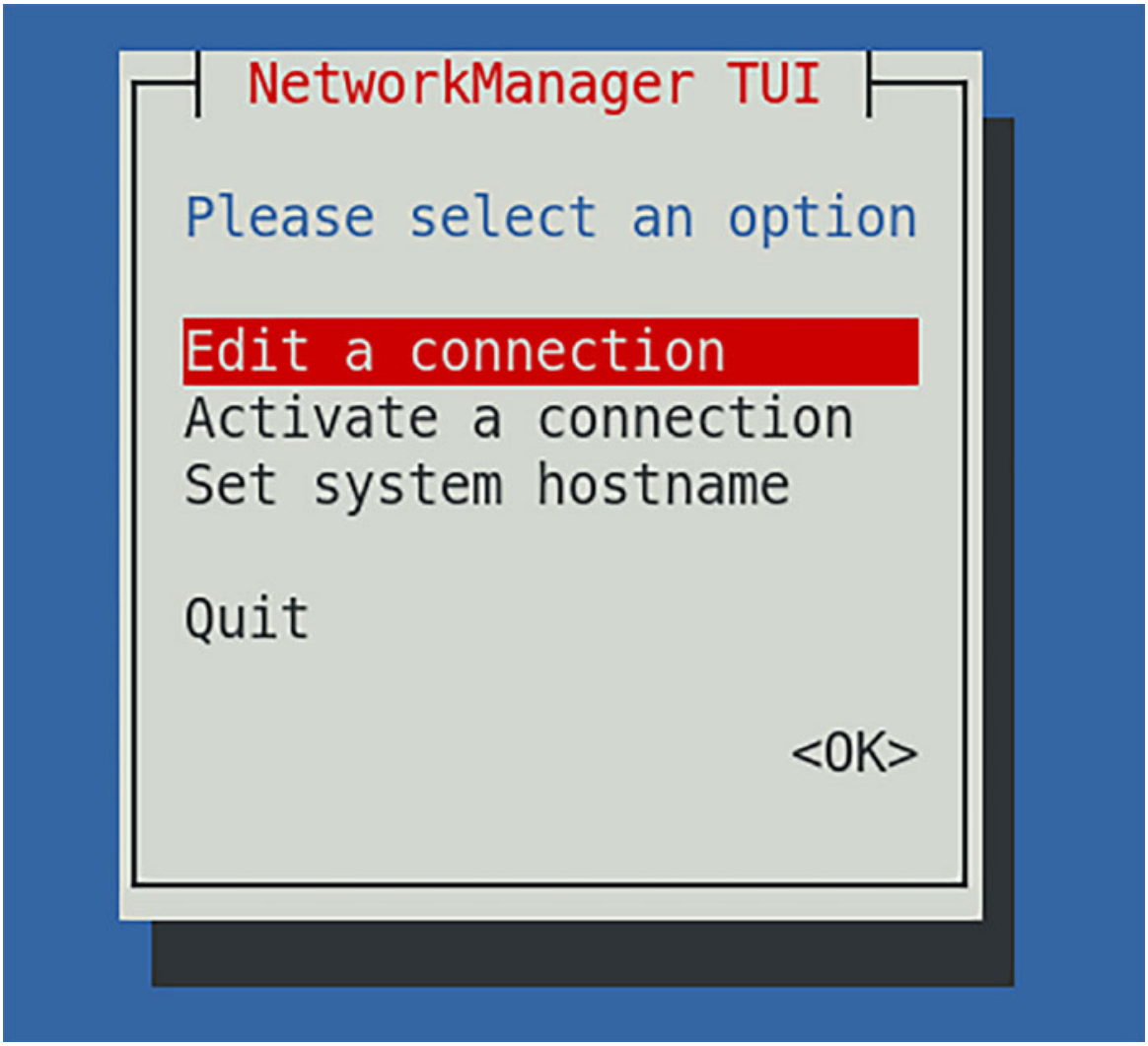


Figure 7.17: *Aspect of the command nmtui.*

The first option, *Edit a connection*, will list and allow the possibility to create new connections or edit existing ones. *Figure 7.18* shows the list and the actions available:

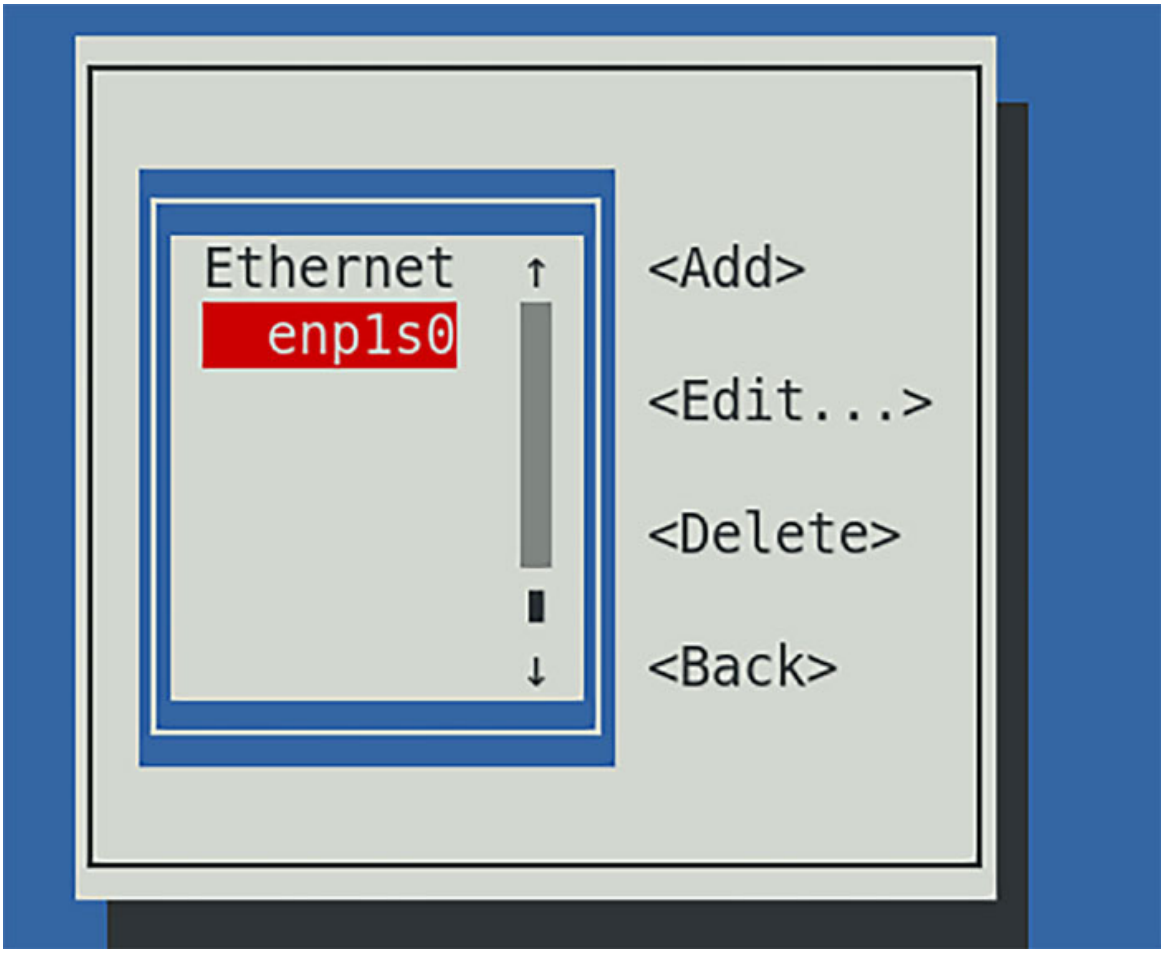


Figure 7.18: *Edit a connection menu on nmtui.*

Using the *Tab* key, it is possible to navigate between the available connections and the right menu. Pressing it in *<Edit...>* will show the configuration for the selected connection, as seen in *figure 7.19*:

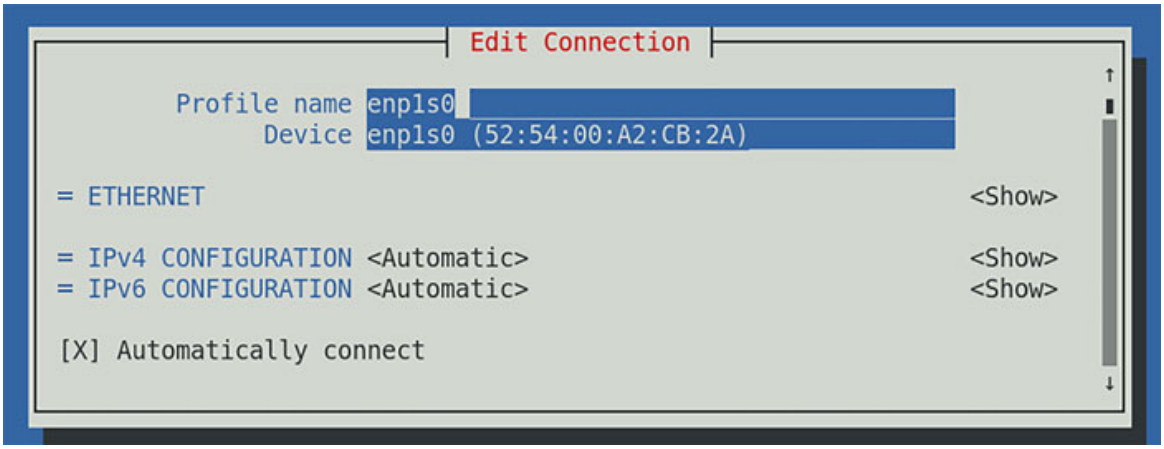


Figure 7.19: *Edit a connection dialog on nmtui.*

Adding a new connection would allow indicating if the interface will use *DHCP* or *static IP* configuration, DNS, and gateway configuration. It is also possible to disable IPv4 or IPv6 depending on the needs of the system and the interface.

Network configuration on Debian and Ubuntu

Debian is using a file-based configuration for networking. The main file defining the network configuration is **`/etc/network/interfaces`**, and the following code block shows an example of a static IP configuration:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
   address 37.139.7.55
   netmask 255.255.255.0
   gateway 37.139.7.1
   dns-nameservers 8.8.4.4 8.8.8.8 209.244.0.3
```

It is possible to execute the command **`man interfaces`** to obtain the possible syntax and different configurations possible inside of the file. After configuring the file, it is required to restart the service *networking* with the command **`systemctl restart networking`**.

Ubuntu distribution uses *netplan* to define the network configuration. The files are defined in YAML and are inside the directory **`/etc/netplan/`**. *Figure 7.20* shows how the configuration in *netplan* is used by **`systemd-networkd`** (default) or *NetworkManager*:

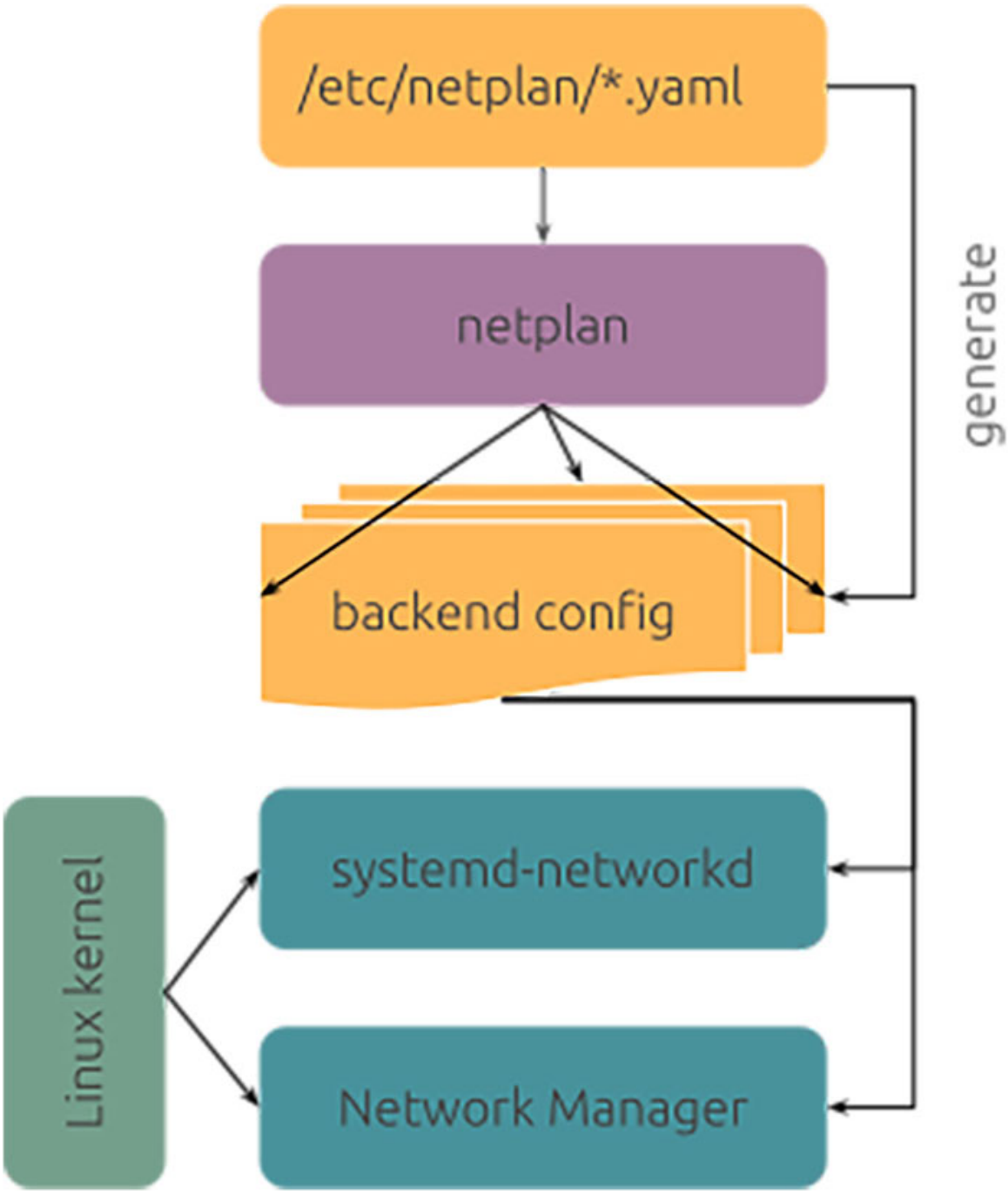


Figure 7.20: *Netplan diagram. Source: Canonical.*

During the installation, a file named **`00-installer-config.yaml`** is created, with the configuration specified during the process. The following code block shows an example:

```
# This is the network config written by 'subiquity'
network:
  ethernets:
```

```
    enp1s0:
      dhcp4: true
  version: 2
```

The following code block shows an example of the configuration of static IP:

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp7s0:
      addresses:
    - 10.10.10.2/24
      nameservers:
    addresses: [10.10.10.1, 1.1.1.1]
      routes:
    - to: default
      via: 10.10.10.1
```

The command **`netplan`** has two main arguments:

1. **`try`**: It tries the configuration defined on files inside **`/etc/netplan/`** and asks the user to confirm the new configuration. If the user does not accept the new settings or does not answer in 120, the configuration will be reverted.
2. **`apply`**: It applies the configuration defined on files inside **`/etc/plan/`** to the system.

Figure 7.21 shows the output of running the command **`netplan try`**:

```
root@ubuntu:/etc/netplan# netplan try
Do you want to keep these settings?


Press ENTER before the timeout to accept the new configuration


Changes will revert in 100 seconds
Configuration accepted.
```

Figure 7.21: *Output example for the command netplan*

Routing

Another important networking concept is routing, which is the process of selecting the path to reach a network. A Linux system can define the rules to

access networks using different gateways. Moreover, a Linux system can be configured as a router for other systems in the same network.

As described previously, using the command **`ip`** with the argument **`route`** lists the routes configured in the system. It is possible to define static routers using network files, *netplan* or *NetworkManager*. *Table 7.4* shows the content configuration or the commands to define the routes:

Element	Content / Command
Debian's interfaces file	allow-hotplug enp1s0 iface enp1s0 inet dhcp up ip route add 192.168.10.0/24 via 192.168.122.10 dev enp1s0 up ip route add 192.168.11.0/24 via 192.168.122.11 dev enp1s0
Ubuntu's netplan configuration	network: ethernets: enp1s0: dhcp4: true routes: - to: 192.168.10.0/24 via: 192.168.122.10 - to: 192.168.11.0/24 via: 192.168.122.11 version: 2
Red Hat's route-eth0 example	192.168.10.0/24 via 192.168.122.1 dev eth0 192.168.11.0/24 via 192.168.122.1 dev eth0
Using nmcli for NetworkManager	nmcli con mod test +ipv4.routes "192.168.10.0/24 192.168.122.10,192.168.11.0/24 192.168.122.11"

Table 7.4: *Routing configuration*

The output example after configuring static routes for the command **`ip route`** is shown in *figure 7.22*:

```
root@ubuntu:~# ip r
default via 192.168.122.1 dev enp1s0 proto dhcp src 192.168.122.101 metric 100
192.168.10.0/24 via 192.168.122.10 dev enp1s0 proto static onlink
192.168.11.0/24 via 192.168.122.11 dev enp1s0 proto static onlink
192.168.122.0/24 dev enp1s0 proto kernel scope link src 192.168.122.101 metric 100
```

Figure 7.22: *Output example for the command ip.*

Configuring a Linux system to act as a *router* does not require advanced configuration always when the same interface is used to redirect the traffic from the source clients to the destination. Using different interfaces requires knowledge related to security and firewall, which is described in the next chapter.

The parameter in the system to know if the traffic is allowed to be forwarded is defined in the **`sysctl`** configuration with the key **`net.ipv4.ip_forward`**. By

default, for security reasons, it is disabled (value *0*), and it is possible to enable it temporarily with the command **`sysctl`** or permanently using the file **`/etc/sysctl.conf`**.

Figures 7.23 to *7.26* show how to check and configure the ip forwarding and ensure that it is working with the Linux system acting as a router:

- Check the value for the system parameter **`ip_forward`** in a server named ubuntu:

```
root@ubuntu:~# sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 0
```

Figure 7.23: *Output example for the command sysctl.*

- Configure a system named **`rhel`** to use the IP of the ubuntu system (*192.168.122.101)* as a gateway and try to reach an external IP (Google's DNS IP *8.8.8.8*):

```
[root@rhel ~]# ip route del default
[root@rhel ~]# ip route add default via 192.168.122.101
[root@rhel ~]# ping -c1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

Figure 7.24: *Operate with command ip and output example for the command ping.*

- Enable **`ip forwarding`** in the system named *ubuntu*:

```
root@ubuntu:~# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

Figure 7.25: *Output example for the command sysctl*

- Try the **`ping`** command from the node **`rhel`** to ensure the communication is possible:

```
[root@rhel ~]# ping -c1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=34.9 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 34.929/34.929/34.929/0.000 ms
```

Figure 7.26: *Output example for the command ping*

Another important topic related to routing, especially for enterprises, is dynamic routing. This type of routing adjusts in real time; the path used for transmitting IP packets depends on different conditions, such as the shortest path or best availability. Protocols such as **Border Gateway Protocol** (**BGP**) and **Open Shortest Path First** (**OSPF**) are some popular examples of dynamic routing.

Advanced network configuration

There are three important advanced network configurations required nowadays for managing Linux systems:

- Systems offering services to end users require special network configuration to avoid bottlenecks and reduce the possibility to have network downtimes. This involves knowledge of configuring Link Aggregation (bonding).
- Systems running Virtual Machines using virtualization technology require to configure network bridges to provide connectivity and traffic isolation. Linux provides tools to create denominated *Linux bridges* or the use of *virtual switches* using some tools such as *Open vSwitch.*
- Managing multiple networks inside of a network requires knowledge related to **Virtual LANs** (**VLANs**) and configuring the Linux system to access to those networks.

Link aggregation (bonding)

Bonding refers to aggregate multiple network connections in one. The purpose is to have a high-availability connection, allowing one of the connections to fail without causing any downtime. Using a protocol named *LACP* (*802.3ad*) allows not only to have a high availability aggregation but also makes it possible to aggregate the traffic bandwidth available. This protocol requires having switches supporting this protocol and configuring the ports connected to the system.

Figure 7.27 shows an example using `nmtui` to define a new bonding combining two interfaces (`enp7s0` and `enp8s0`), thus creating a new virtual bonding interface called *bond0*:

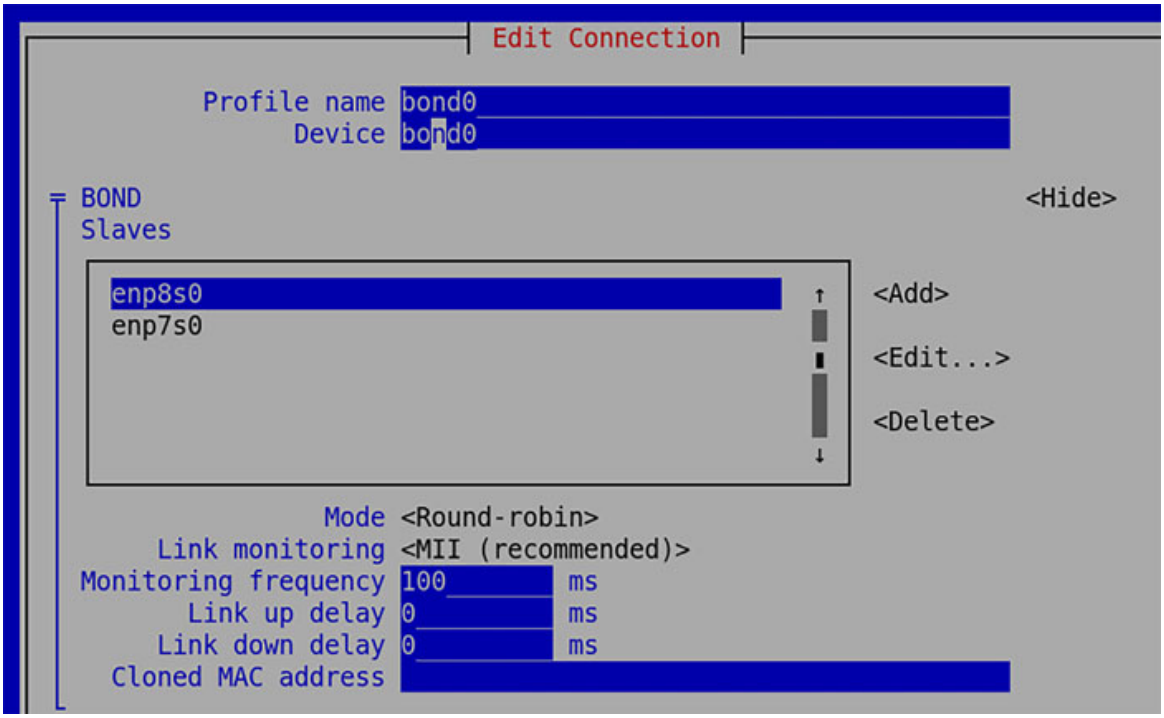


Figure 7.27: *Dialog menu in command nmtui*

Running the command **`ip`** to list the interfaces will show the physical interfaces and the new *bonding* interface. The physical interface will show the name of the *bonding* it belongs. Refer to *figure 7.28*:

```
[root@rhel ~]# ip -c a | tail -10
3: enp7s0: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
bond0 state UP group default qlen 1000
    link/ether 52:54:00:10:65:8d brd ff:ff:ff:ff:ff:ff
4: enp8s0: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
bond0 state UP group default qlen 1000
    link/ether 52:54:00:10:65:8d brd ff:ff:ff:ff:ff:ff permaddr 52:54:00:c8:55:25
5: bond0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
 group default qlen 1000
    link/ether 52:54:00:10:65:8d brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.218/24 brd 10.0.0.255 scope global dynamic noprefixroute bond0
       valid_lft 3474sec preferred_lft 3474sec
    inet6 fe80::6f77:ae7a:a30a:bf68/64 scope link noprefixroute
       valid lft forever preferred lft forever
```

Figure 7.28: *Output example for the command ip*

A file inside of the special directory */proc/net/bonding/*, with the name of the *bonding* interface, is created with the information related to the configuration. The following *code block* shows an example of the previous bonding shown in *figure 7.28*, which is a configuration with the mode *load balancing*:

```
Ethernet Channel Bonding Driver: v5.14.0-70.17.1.el9_0.x86_64
```

```
Bonding Mode: load balancing (round-robin)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0

Slave Interface: enp7s0
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 52:54:00:10:65:8d

Slave queue ID: 0
Slave Interface: enp8s0
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 52:54:00:c8:55:25
Slave queue ID: 0
```

Network bridges

A network bridge allows to aggregate multiple networks or network segments in a single device. A network bridge is different for routing; this aggregation happens in Layer 2 (*data link layer*). Network bridges are popular in virtualization, when multiple VMs are connected to the same device, but can be part of different networks. *Figure 7.29* illustrates how several VMs can be connected to a Linux bridge, which has as a member the network device, *enp7s0*:

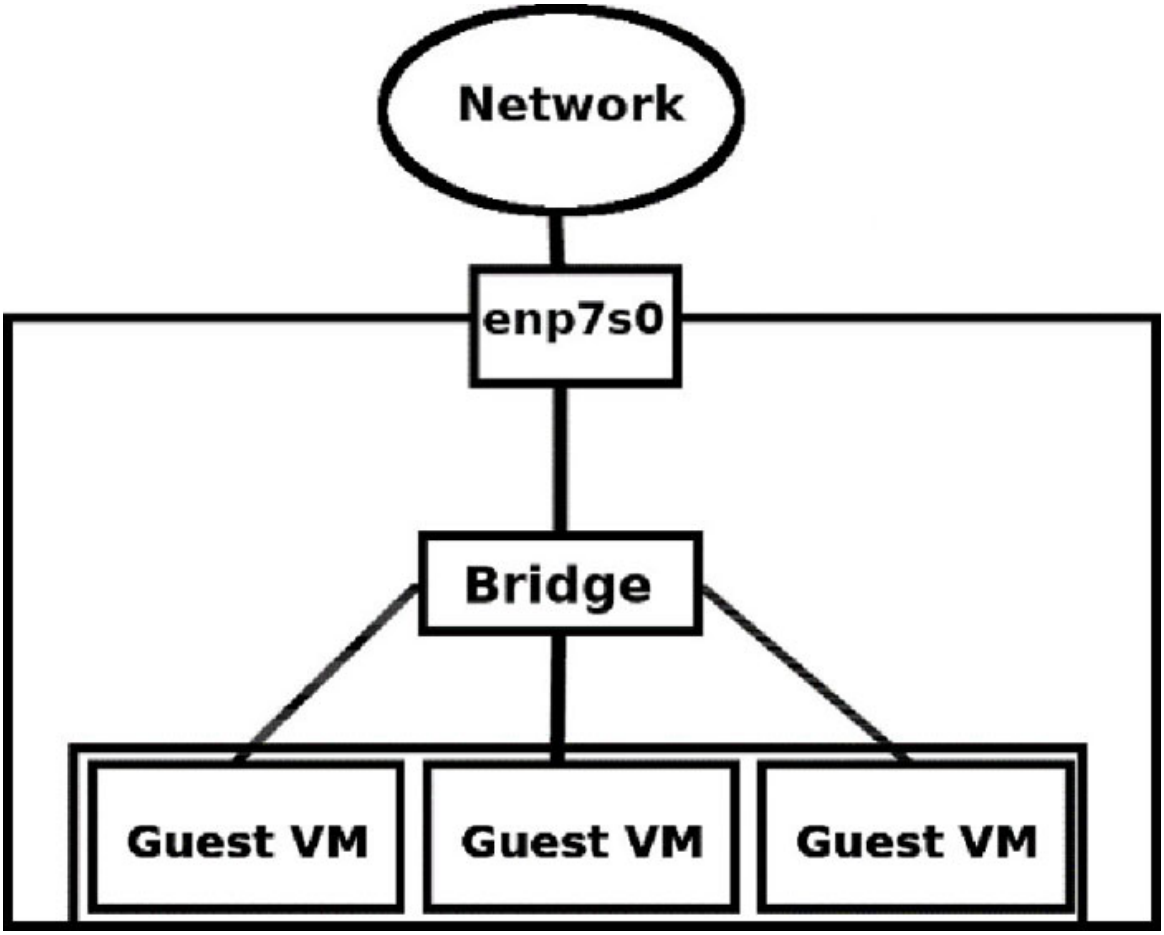


Figure 7.29: *Network bridge diagram example.*

It is possible to manually create a *Linux Bridge* using the command **`ip`**, and it is also possible to check the status using the command **`bridge`**. *Figure 7.30* shows an example of the usage:

```
root@ubuntu:~# ip link add name bridge0 type bridge
root@ubuntu:~# ip link set bridge0 up
root@ubuntu:~# ip link set enp7s0 master bridge0
root@ubuntu:~# bridge link
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state forwa
rding priority 32 cost 100
```

Figure 7.30: *Output example for the command bridge after creating a bridge*

Using *NetworkManager*, it is possible to create a bridge, using the *connection add* arguments. *Figure 7.31* illustrates this with an example:

```
[root@rhel ~]# nmcli connection add type bridge ifname bridge0 con-name bridge0
Connection 'bridge0' (8c583fd0-9e9f-471f-9bd1-6884baa40cd9) successfully added.
[root@rhel ~]# nmcli con add type bridge-slave ifname enp7s0 master bridge0
Connection 'bridge-slave-enp7s0' (4e40de6e-4953-4037-a7f3-90170f54caea) successf
ully added.
[root@rhel ~]# nmcli con up bridge-slave-enp7s0
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkMa
nager/ActiveConnection/6)
[root@rhel ~]# nmcli con up bridge0
Connection successfully activated (master waiting for slaves) (D-Bus active path
: /org/freedesktop/NetworkManager/ActiveConnection/7)
```

Figure 7.31: *Output example for the command nmcli*

Using **`netplan`**, it is possible to create a Linux bridge, as shown in the following code as an example:

```
network:
  version: 2
  renderer: networkd
  ethernets:
     enp7s0:
     dhcp4: no
  bridges:
     bridge0:
     dhcp4: yes
     interfaces:
       - enp7s0
```

Open vSwitch (**OVS**) is an open-source project to operate with virtual switches. Nowadays, it is recommended to use *OVS* instead of *Linux bridges* due to the flexibility, efficiency, and compatibility with more tunnel protocols (*VXLAN, GRE, and so on).* It is also possible to integrate with *Software Defined Network* solutions. *Figure 7.32* shows the communication between two hosts using OVS and acting as a network bridge for Virtual Machines:

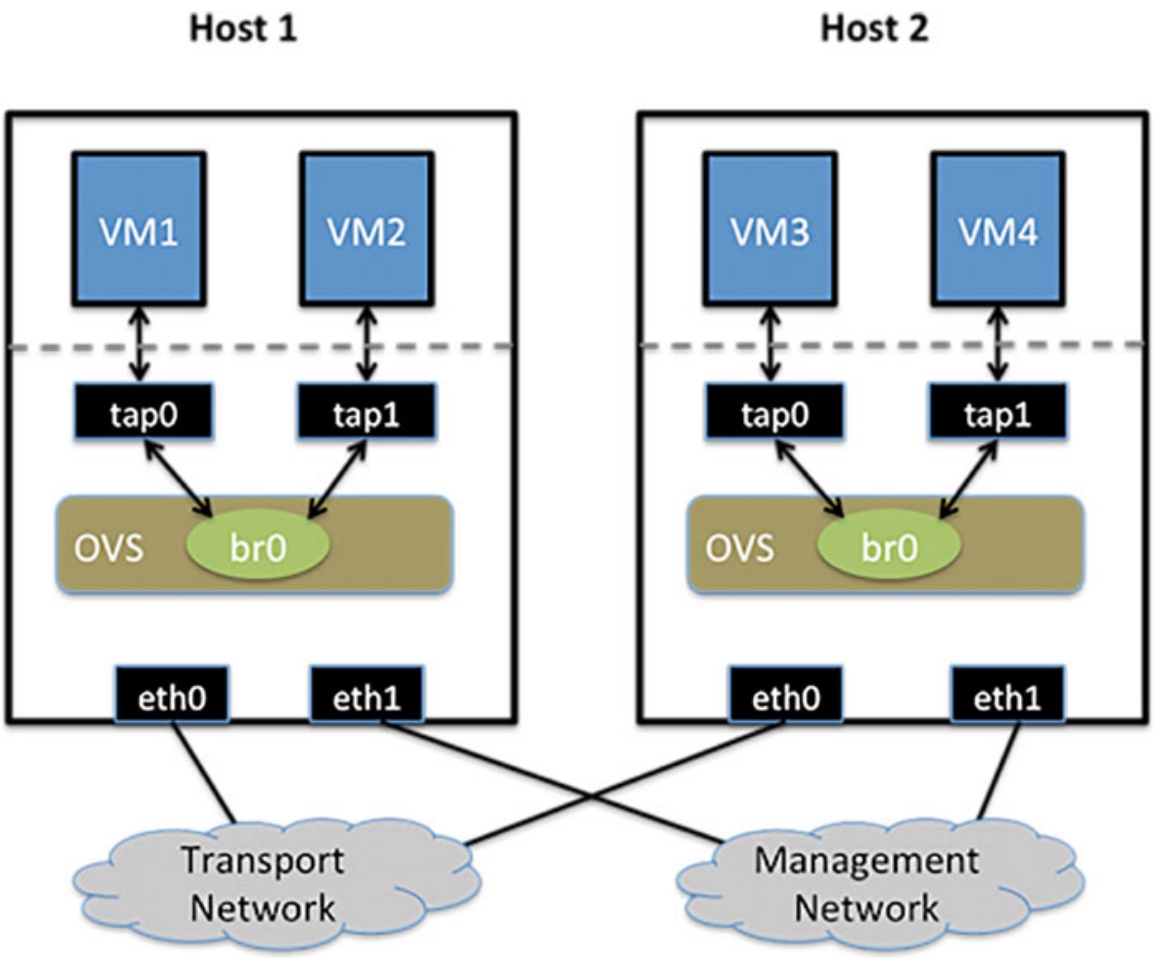


Figure 7.32: *Open vSwitch diagram example. Source: Open vSwitch*

Figure 7.33 shows an example to configure *Open v*Switch using the command **`ovs-vsctl`** included in the package **`openvswitch-switch`** (in Ubuntu repository) or package *openvswitch2.17* (in RHEL 9):

```
root@ubuntu:~# ovs-vsctl add-br bridge0
root@ubuntu:~# ovs-vsctl add-port bridge0 enp7s0
root@ubuntu:~# ovs-vsctl show
c0899923-16fa-4b08-9585-682b950a39b7
    Bridge bridge0
        Port enp7s0
            Interface enp7s0
        Port bridge0
            Interface bridge0
                type: internal
    ovs_version: "2.17.2"
```

Figure 7.33: *Output example for the command ovs-vsctl.*

Virtual LANs (VLANs)

Virtual LANs is a method of creating independent logical networks within a physical network. The separation between the logical networks is done using a tag called *VLAN ID,* which is a number between 0 and 4,095. This *VLAN ID* is part of the network packet. *Figure 7.34* shows the structure of a VLAN data packet:

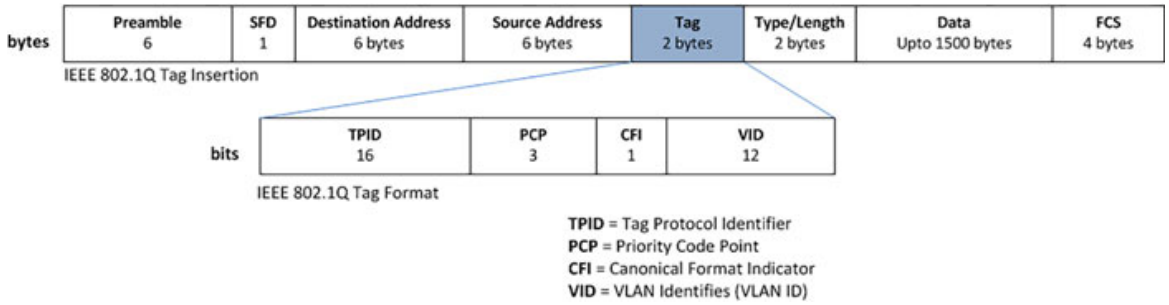


Figure 7.34: *Packet fields for an Ethernet card using VLANs*

The following example in *figure 7.35* illustrates how to use **`nmcli`** to configure an interface to be connected using the VLAN ID 10:

```
[root@rhel ~]# nmcli c a type vlan con-name vlan10 ifname vlan10 dev enp8s0 id 10
Connection 'vlan10' (89043841-7476-44db-9b7a-d657f63c5195) successfully added.
[root@rhel ~]# nmcli con show vlan10 | grep -i ^vlan
vlan.parent:                            enp8s0
vlan.id:                                10
vlan.flags:                             1 (REORDER_HEADERS)
vlan.ingress-priority-map:              --
vlan.egress-priority-map:               --
```

Figure 7.35: *Output example for the command nmcli.*

Conclusion

Networking is one of the core parts of the Linux server. Modern services rely on networking, and knowledge about advanced features is required when virtualization and containers are involved. This chapter covered topics from the basic concept of the network to the most advanced ones, such as bridging and virtual LANs.

Key facts

- *OSI model* knowledge is required to understand traffic flow.
- Linux provides several tools to check connectivity.
- Network configuration is configured using Network Manager or netplan.
- Linux eases advanced network configuration with simple tools.

Questions

1. How many layers are part of the *OSI Model*?
    a. Five layers
    b. Seven layers
    c. Nine layers
2. In which layer the routing is placed in?
    a. First layer
    b. Second layer
    c. Third layer
3. What is the default tool to configure networking on a Ubuntu server?
    a. netplan
    b. NetworkManager
    c. systemd
4. Which command on *NetworkManager* is used for the text user interface?
    a. nmtui
    b. nmcli

c. nmui

5. What key is required to be enabled for *IP forwarding*?

    a. net.ipv4.forwarding
    b. net.ipv4.ip_forwarding
    c. net.ipv4.ip_forward

6. What directory contains files with the bonding information?

    a. /sys/net/bonding/
    b. /proc/net/bonding/
    c. /dev/net/bonding/

Answers

1. b
2. c
3. a
4. a
5. c
6. b

CHAPTER 8

Security

Introduction

Offering a service to end users leads to possible security risks, such as data leaks or unauthorized access. Though Linux is the most secure operating system, it is necessary to avoid unwanted access to the service or to the system itself. To protect a system, there are two main important points: keep the system updated to avoid attacks on the software in the system, and protect who has access to the system through the network.

This chapter is focused on network security using a firewall and limiting access to different services using configuration. Linux also provides different tools to avoid the execution of unwanted software and checks if the system is compromised.

Structure

In this chapter, we will discuss the following topics:

- Security introduction
- Firewall configuration on Linux
- Services security
- Network monitoring

Security introduction

One of the main tasks of Linux system administrators is to keep the system updated and protected from unwanted access to the system itself or applications. *Chapter 4, User Administration and Software Management* describes how to create users and groups and how, by using permissions, it is possible to limit access to files and directories. It is important to protect the system by checking the files using special permissions, to avoid unwanted permission escalation. The process of protecting a system from a default installation is called *hardening*.

Security networking uses this knowledge to protect the system from remote access, either from a user or system in the same network or an attacker from

outside of the network through the internet. The main security related to networking is to use a Firewall on the Linux system. Even though the network would be protected by a network firewall, it is important to protect the systems with their own firewall to create double protection.

Historically, firewall configuration in Linux was a complicated task that required knowledge about *iptables*, a tool to maintain filter rules in the Linux kernel. Modern distributions include high-level tools to manage firewall rules, helping administrators and regular users to main firewall rules in an easy way.

Depending on the Linux distribution used, the default firewall tool will be different. Red Hat Enterprise Linux and derivatives, such as CentOS Stream or Rocky Linux, rely on the *Firewalld* project as firewall management. Ubuntu distribution uses an **Uncomplicated Firewall** ***(ufw***) as the default configuration tool to ease firewall configuration.

The default backend for the firewall solutions in modern distributions is *nftables*, which replaces the traditional *iptables*. It is possible to use the command `nft` to manage or list the rules configured in the system.

The following schematic *figure 8.1* shows packet flows through Linux networking:

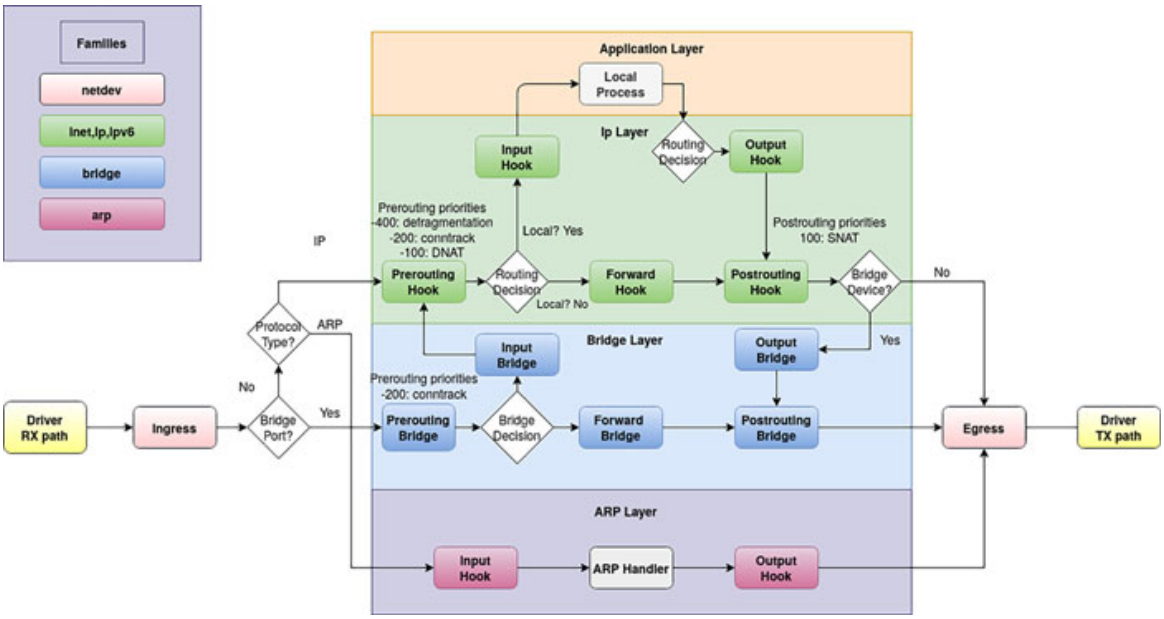


Figure 8.1: *Packet flows through Linux networking. Source: nftables.*

Linux services implement their own security configuration. For example, the service *SSH* allows to configure to not allow login with the administrator user *root* or only allows the connection using *SSH public/private keys*. Another example is

NFS Server, which limits the exported directories to specific IPs/Network ranges and with the possibility to enable authentication.

Another protection solution offered by the Linux distributions is called *mandatory access control* policies. This security is a proactive approach that helps to protect the system against both known and unknown vulnerabilities. These policies enforce what the application and the user can do, such as access to directories, the ports they can use, or the operations they can do. For example, it protects a Web server to access a directory that is not the default or denies the connection from the Web server to a remote location.

Red Hat Enterprise Linux and derivatives offer **Security-Enhanced Linux** (**SELinux**) for supporting **Access Control Security Policies**. Debian and Ubuntu rely on *AppAmor* solution as **Mandatory Access Control** (**MAC**).

Other security-related management examples on Linux for *hardening* include the following:

- Disk encryption: Using **Linux Unified Key Setup** (**LUKS**) for hard drive encryption.
- Secure user access: Includes **Pluggable Authentication Modules** (**PAM**) configuration, the configuration of user *limits*, the configuration of *sudo,* and user auditing, among others.
- Restrict file-system permissions: It is possible to restrict which directories can contain executables and what directories should be read-only.

Linux distributions offer the possibility to comply with three popular security certifications, which usually are required for companies in the public sector or companies working with sensible data:

- **Federal Information Processing Standards Publications (FIPS): Issued by the National Institute of Standards and Technology (NIST)**, the *FIPS 140-2* specifies the security requirements for cryptographic modules.
- **Center for Internet Security** (**CIS**): It publishes benchmarks related to services. They offer **Extensible Configuration Checklist Description Format** (**XCCDF***)* format benchmarks, which can be used by different tools.
- **Security Technical Implementation Guides** (**STIG**): Developed by the **Information System Agency** (**DISA**) for the **US Department of Defense** (**DoD**), STIG consists of security controls, including configuration settings to hardening Linux distributions.

Firewall configuration on Linux

One of the common tasks performed during the installation is to configure the Firewall. The considerations to have a secured firewall configuration are as follows:

- Block by default; if there is not a specific rule, then the connection should be rejected.
- Open only the ports needed for the services and specify the protocol for those ports.
- Limit the connection from specific IPv4 or Ipv6 addresses.
- Apply the rules to specific network interfaces in the system.
- Limit the rate of the connections to avoid attacks.
- Automate firewall rules with an automation solution (for example, *Ansible*).

Firewalld

Firewalld provides a dynamically managed firewall with zones that define the trust level of network connections or interfaces. A big benefit of the usage of firewalld, is the possibility to perform changes in the rules without having to restart the service. It uses **Desktop Bus** (**D-Bus**) as a message bus system, a simple way for applications to talk to each other. Firewalld has different support for several backends. *Figure 8.2* illustrates the structure:

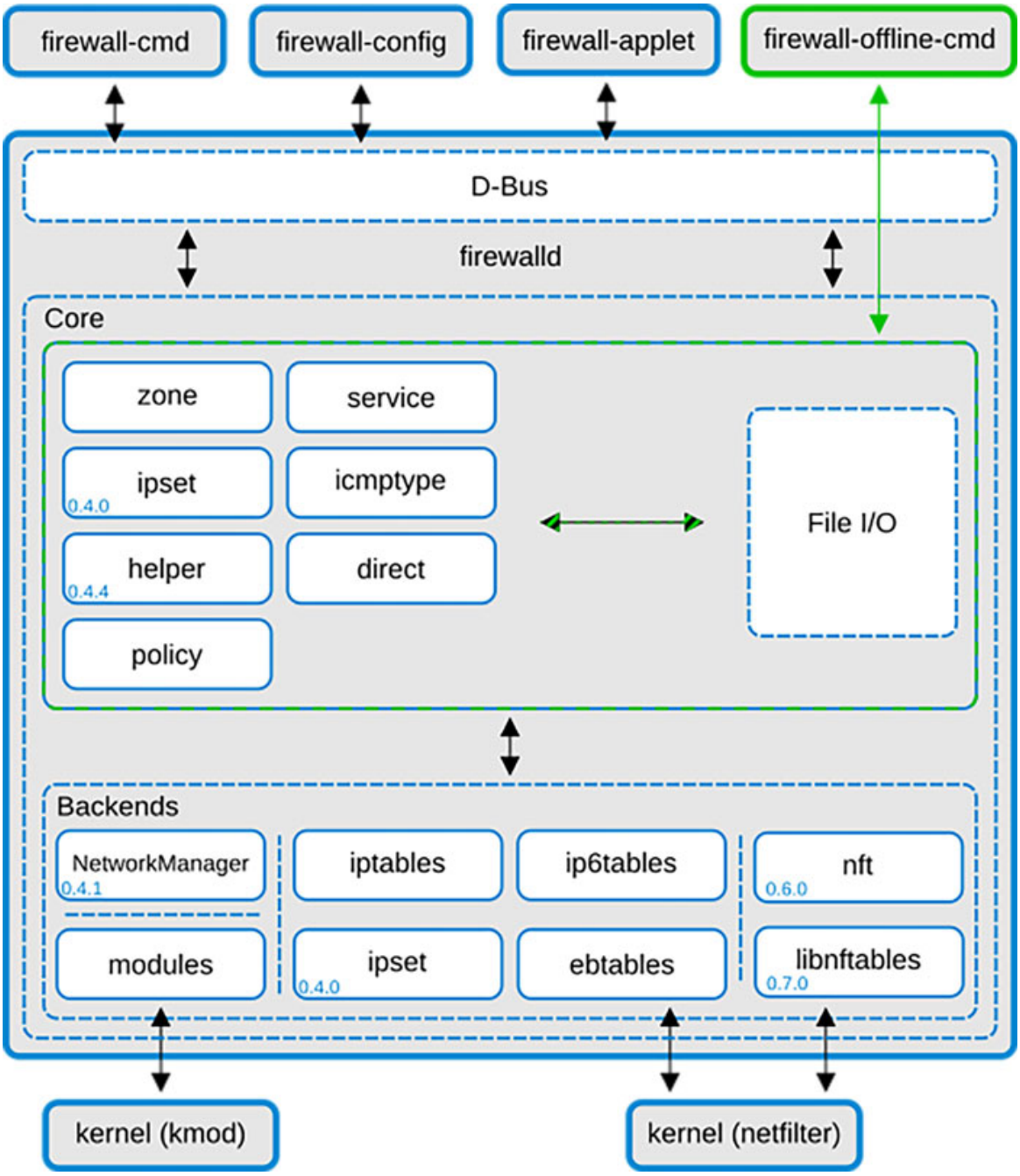


Figure 8.2: *Firewalld structure. Source: firewalld*

The command **`firewall-cmd`** communicates with the service **`firewalld`** to operate with rules and zones. The service will use the backend defined, by default *nft*, to configure the rules indicated.

The default configuration file for *firewalld* is located on **`/etc/firewalld/firewalld.conf`**. The following code contains an example of the default configuration values:

```
DefaultZone=public
CleanupOnExit=yes
CleanupModulesOnExit=no
Lockdown=no
Ipv6_rpfilter=yes
IndividualCalls=no
LogDenied=off
FirewallBackend=nftables
FlushAllOnReload=yes
RFC3964_Ipv4=yes
```

It is possible to check the status of the *firewalld* service using the option **--state** or by using the command **`systemctl`**. *Figure 8.3* illustrates running these two options:

```
[root@rhel ~]# firewall-cmd --state
running
[root@rhel ~]# systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
     Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor p>
     Active: active (running) since Sat 2022-08-27 16:59:28 CEST; 7h ago
       Docs: man:firewalld(1)
   Main PID: 802 (firewalld)
      Tasks: 2 (limit: 7757)
     Memory: 28.8M
        CPU: 2.240s
     CGroup: /system.slice/firewalld.service
             └─802 /usr/bin/python3 -s /usr/sbin/firewalld --nofork --nopid

Aug 27 16:59:28 rhel.example.com systemd[1]: Starting firewalld - dynamic firewal>
Aug 27 16:59:28 rhel.example.com systemd[1]: Started firewalld - dynamic firewall>
```

Figure 8.3: *Output example checking firewalld service status*

By default, **`firewalld`** has some predefined zones for different purposes. *Table 8.1* shows the zone name and the description:

Zone	Description
drop	Any incoming network packets are dropped, and there is no reply. Only outgoing network connections are possible.
block	Any incoming network connections are rejected with an icmp-host-prohibited message for IPv4 and icmp6-adm-prohibited for IPv6. Only network connections initiated within this system are possible.
public	For use in public areas. You do not trust the other computers on networks to not harm your computer. Only selected incoming connections are accepted.
external	For use on external networks with masquerading enabled, especially for routers. You do not trust the other computers on networks to not harm

	your computer. Only selected incoming connections are accepted.
dmz	For computers in your demilitarized zone that are publicly-accessible with limited access to your internal network. Only selected incoming connections are accepted.
work	For use in work areas. You mostly trust the other computers on networks not to harm your computer. Only selected incoming connections are accepted.
home	For use in home areas. You mostly trust the other computers on networks not to harm your computer. Only selected incoming connections are accepted.
internal	For use on internal networks. You mostly trust the other computers on the networks not to harm your computer. Only selected incoming connections are accepted.
trusted	All network connections are accepted.

Table 8.1: *Predefined zones in firewalld*

It is possible to use the option *--get-zones* to list the zones available. **`--get-active-zones`** lists only the active zones, including the interfaces assigned to those zones. The option **`--get-default-zone`** prints the default zone for connections and interface. *Figure 8.4* shows output examples for the options described:

```
[root@rhel ~]# firewall-cmd --get-zones
block dmz drop external home internal nm-shared public trusted work
[root@rhel ~]# firewall-cmd --get-active-zones
public
  interfaces: enp1s0 bridge0 bond0 enp7s0 enp8s0 vlan10
[root@rhel ~]# firewall-cmd --get-default-zone
public
```

Figure 8.4: *Output example getting zones using command firewall-cmd*

The option **`--list-all`** lists information related to the default zone, which is *public* by default, such as the services allowed and the interfaces assigned. It is possible to specify information about another zone, by using the option **`--zone=name`**. *Figures 8.5* and *8.6* show both examples:

```
[root@rhel ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: bond0 bridge0 enp1s0 enp7s0 enp8s0 vlan10
  sources:
  services: cockpit dhcpv6-client ssh
  ports:
  protocols:
  forward: yes
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
[root@rhel ~]# firewall-cmd --list-all --zone=trusted
trusted
  target: ACCEPT
  icmp-block-inversion: no
  interfaces:
  sources:
  services:
  ports:
  protocols:
  forward: yes
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Figures 8.5, 8.6: *Getting information about the default zone and a specific zone*

It is possible to change the default zone using the option **`--set-default=name`**. To allow connections to the system, there are several options to allow connections to different ports or services from different sources. *Table 8.2* describes some popular options:

Option	Description

--add-port=port/protocol	Allow connections to the port and protocol specified.
--add-service=service	Allow connections to the service specified.
--add-interface=interface	Assign the interface to the default zone or to the zone specified.
--add-source=subnet/mask	Allow connection from the IP or subnet specified.

Table 8.2: *Options to configure zones*

It is possible to use **`--remove-port`**, **`--remove-service`**, **`--remove-interface`**, and **`--remove-source`** to delete configurations from the default zone or the zone specified. To be accepted, the connection, source, and target should be allowed in the zone. *Figure 8.7* shows some examples of the usage of the options:

```
[root@rhel ~]# firewall-cmd --add-port=80/tcp
success
[root@rhel ~]# firewall-cmd --add-service=https
success
[root@rhel ~]# firewall-cmd --remove-interface=vlan10
success
[root@rhel ~]# firewall-cmd --add-interface=vlan10 --zone=trusted
success
[root@rhel ~]# firewall-cmd --add-source=10.0.0.0/24 --zone=trusted
success
```

Figures 8.7: *Using firewall-cmd to manipulate zones*

In *figure 8.7*, the following operations were performed:

- The connection to port 80/tcp was allowed.
- The connection to the server *https* (443/tcp) was allowed.
- Interface named *vlan10* was removed from the default zone.
- Interface named *vlan10* was added to the zone *trusted*.
- The network 10.0.0.0/24 was added as an allowed source in the zone *trusted.*

Changes in *firewalld* are kept in memory (*runtime configuration)* till they are converted to permanent rules. The options added as permanent with the option --*permanent* are not active till the configuration is reloaded with the option --*reload*. *Figure 8.8* shows how a permanent rule is only applied after the configuration is reloaded:

```
[root@rhel ~]# firewall-cmd --add-service=http --permanent
success
[root@rhel ~]# firewall-cmd --list-all |grep services
  services: cockpit dhcpv6-client ssh
[root@rhel ~]# firewall-cmd --reload
success
[root@rhel ~]# firewall-cmd --list-all |grep services
  services: cockpit dhcpv6-client http ssh
```

Figure 8.8: *Adding a permanent rule to firewalld*

The rules not converted to permanent rules will be lost when the option –reload is used or when the service **`firewalld`** is restarted. It is possible to convert the *runtime rules* as a permanent rules, using the option **`--runtime-to-permanent`** for that purpose. *Figure 8.9* shows how to use that option:

```
[root@rhel ~]# firewall-cmd --add-port=443/tcp
success
[root@rhel ~]# firewall-cmd --runtime-to-permanent
success
```

Figure 8.9: *Convert runtime rules to permanent*

Other common options in **`firewall-cmd`** can be seen in *table 8.3*:

Option	Description
--list-services	List only the services allowed in the default or specified zone.
--get-services	List all the predefined services.
--list-all-zones	Get detailed information for all the zones, and even if they are not active ones.
--list-ports	List the allowed ports
--new-zone=name	Create a new zone

Table 8.3: *Common options for the command firewall-cmd*

ufw

The **Uncompleted Firewall** (***ufw***) is the default firewall configuration tool for Ubuntu. It helps in the creation of firewall rules in the system. By default, it is initially disabled. With the argument *status*, it is possible to check the current status and with the argument *enable*, it is possible to activate it, as *figure 8.10* illustrates:

```
root@ubuntu:~# ufw status
Status: inactive
root@ubuntu:~# ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
```

Figure 8.10: *Check the status and enable the firewall using ufw*

It is possible to allow a new connection using the argument *allow* and indicating the port to be permitted. The argument *deny* is to close the connection. It is possible to specify the protocol (argument *proto*), the source (argument *from*), the destination ip *(to*), and the destination port *(port*). *Figure 8.11* shows an example allowing port 22 to everyone and port 80 to a specific IP:

```
root@ubuntu:~# ufw allow 22
Rule added
Rule added (v6)
root@ubuntu:~# ufw allow proto tcp from 192.168.122.145 to any port 22
Rule added
```

Figure 8.11: *Allowing connections to the system using ufw*

When **`ufw`** is configured with rules, the argument *status* lists them. It is possible to add *verbose* as a second argument, to obtain information about default actions, as shown in *figure 8.12*:

```
root@ubuntu:~# ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22                         ALLOW IN    Anywhere
22/tcp                     ALLOW IN    192.168.122.145
22 (v6)                    ALLOW IN    Anywhere (v6)
```

Figure 8.12: *Listing the rules in ufw*

Some applications are creating a definition of the ports used inside of the directory **`/etc/ufw/applications.d/`**. The following code shows the definition for the **`openssh-server`** application:

```
[OpenSSH]
title=Secure shell server, an rshd replacement
```

```
description=OpenSSH is a free implementation of the Secure Shell
protocol.
ports=22/tcp
```

It is possible to list the applications defined using the arguments *app list* and obtain information using *app info*. *Figure 8.13* shows the example of usage:

```
root@ubuntu:~# ufw app list
Available applications:
  OpenSSH
root@ubuntu:~# ufw app info OpenSSH
Profile: OpenSSH
Title: Secure shell server, an rshd replacement
Description: OpenSSH is a free implementation of the Secure Shell protocol.

Port:
  22/tcp
```

Figure 8.13: *Listing applications and obtaining information on ufw*

Masquerading

In *Chapter 7, Network Configuration*, we learned how to configure a Linux server to act as a router. Using **IP Forwarding**, it is possible to use a Linux system as a gateway to access other networks. That is possible only when **Network Address Translation** (**NAT**) is not involved. For example, this communication is possible in the scenario when the node has the IP *192.168.122.10* and gateway *192.168.122.1*, and the nodes from the network *192.168.122.0/24* use the gateway *192.168.122.10* and *IP forwarding* is enabled.

If **One-to-many Network Address Translation** is involved, meaning the connection from one network is redirected to another network, then **masquerading** is required to perform that connection. *Figure 8.14* illustrates the traffic flow and the *IP masquerading* process:

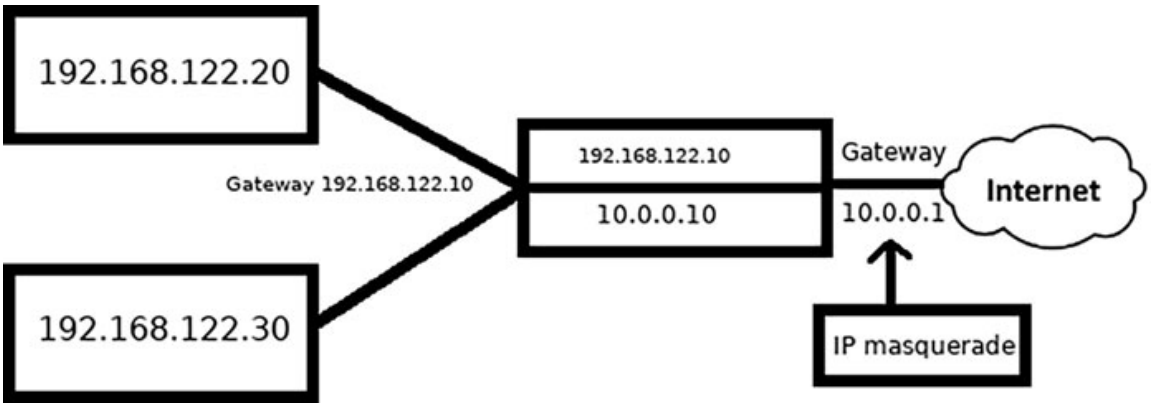


Figure 8.14: *IP masquerading diagram*

Both firewall solutions described, **`firewalld`** and **`ufw`**, allow to enable masquerading for zones or interfaces. *Figure 8.15* shows how to enable it using *--add-masquerade* for the command **`firewall-cmd`** command:

```
[root@rhel ~]# firewall-cmd --list-all |grep -i masquerade
  masquerade: no
[root@rhel ~]# firewall-cmd --add-masquerade --permanent
success
[root@rhel ~]# firewall-cmd --reload
success
[root@rhel ~]# firewall-cmd --list-all |grep -i masquerade
  masquerade: yes
```

Figure 8.15: *Enable masquerading with firewall-cmd*

Using *ufw* requires manual tasks editing two files, and disabling/enabling the firewall:

- Edit file **`/etc/default/ufw`** and set the value for the key **`DEFAULT_FORWARD_POLICY`** to the value **`ACCEPT`**.
- Edit file **`/etc/ufw/before.rules`** to add the following line (before the **`COMMIT`** line):

  ```
  -A POSTROUTING -s 192.168.122.0/24 -o eth0 -j MASQUERADE
  ```

- Execute the commands: ufw disable && sudo ufw enable.

Services security

It is important to keep the system secure in regard to who can access the services. Firewall is the first barrier of security, indicating who has access and to what at the network connection. Some security recommendations related to services are the following:

- Disable services that are not needed. Some services are enabled during installation, and they are recommended to monitor which services are running in the system and disable the ones that are not needed.
- Configure services to listen to a specific interface. Applications are usually listening in all interfaces available, which can lead to security threads due to the firewall accepting all communications through one of the interfaces.
- Turn off IPv6 if it is not in use. Some networks are not using IPv6, and in such cases, it is recommended not to use it on services to avoid undesired

access.

- Enable logging and auditing. It is important to enable logging and auditing for the services where users will connect and operate. A central logging service is recommended to be able to create reports and alert notifications.
- Use an **Intrusion Detection System** (**IDS**). This service detects possible intrusions to the system.
- Using Security Models. Software like SELinux and AppArmor protects the system mitigating unauthorized accesses and attacks.

Disabling not needed services

To list the services which are enabled, it is possible to use the command `systemctl` with argument `list-unit-files` and filter with the options `--type=service` and `--state=enabled`, as shown in *figure 8.16*:

```
root@ubuntu:~# systemctl list-unit-files --state=enabled --type=service
UNIT FILE                                 STATE   VENDOR PRESET
apparmor.service                          enabled enabled
atop.service                              enabled enabled
atopacct.service                          enabled enabled
blk-availability.service                  enabled enabled
cloud-config.service                      enabled enabled
cloud-final.service                       enabled enabled
cloud-init-local.service                  enabled enabled
cloud-init.service                        enabled enabled
console-setup.service                     enabled enabled
cron.service                              enabled enabled
dmesg.service                             enabled enabled
e2scrub_reap.service                      enabled enabled
```

Figure 8.16: *List the services enabled in the system using systemctl*

To list the services which are running in the system, use the argument `list-units` and the options `--type=service` and `--state=running`, as is illustrated in *figure 8.17*:

```
root@ubuntu:~# systemctl list-units --type=service --state=running
  UNIT                          LOAD   ACTIVE SUB     DESCRIPTION
  atop.service                  loaded active running Atop advanced performance mo
  atopacct.service              loaded active running Atop process accounting daem
  cron.service                  loaded active running Regular background program p
  dbus.service                  loaded active running D-Bus System Message Bus
  getty@tty1.service            loaded active running Getty on tty1
  irqbalance.service            loaded active running irqbalance daemon
  ModemManager.service          loaded active running Modem Manager
  multipathd.service            loaded active running Device-Mapper Multipath Devi
  networkd-dispatcher.service loaded active running Dispatcher daemon for system
  ovs-vswitchd.service          loaded active running Open vSwitch Forwarding Unit
```

Figure 8.17: *List the services running in the system using the command systemctl*

Listing services listening in all interfaces

The command **ss** can be used to list the services which are listening in all interfaces. To listen in all interfaces, the special local IP *0.0.0.0* for IPv4 is used and *[::]* for IPv6. *Figure 8.18* shows an output example using *ss* filtering for those IPs:

```
root@ubuntu:~# ss -tunlp 'src = 0.0.0.0 or src = [::]'
Netid   State    Recv-Q   Send-Q     Local Address:Port      Peer Address:Port
Process
tcp     LISTEN   0        128              0.0.0.0:22             0.0.0.0:*
 users:(("sshd",pid=874,fd=3))
tcp     LISTEN   0        511                    *:80                   *:*
 users:(("apache2",pid=2416,fd=4),("apache2",pid=2415,fd=4),("apache2",pid=2414,f
d=4))
tcp     LISTEN   0        128                 [::]:22                [::]:*
 users:(("sshd",pid=874,fd=4))
```

Figure 8.18: *Output example for the command ss*

In the previous example, the services **sshd** and **apache2** are listening in all interfaces. Each service defines different configuration files and options to indicate in which interface to listen. For example, for the *SSH server,* the option is named **ListenAddress** for Apache2, the option is *Listen*.

Service logging

Traditionally, the services used their own log files to store the information related to the process, access, or other information. For example, **Apache2** uses the directory **/var/log/httpd/** or **/var/log/apache2/** to log access to the Web server, which is useful to analyze unwanted access to the service. Other services are using the *system logging* service in the system to store the *logs* on it. The service more popular on Linux for that purpose is **syslog-ng**.

The modern infrastructure uses a centralized log aggregation service, where the systems are streaming the system and service logs to that service. Popular log collectors are *logstash* and *fluentd* projects. Using a centralized logging system eases the log filtering and creation of alerts for all the Linux systems and services in the network. Projects *Grafana* and *Kibana* are popular solutions for the visualization of logs.

Intrusion detection system

Linux provides several popular options for *Intrusion Detection Systems*, such as detecting files integrity, connection attacks, or unauthorized access to the system. Some of these options are as follows:

- *Tripwire*: Detects unauthorized filesystem changes.
- *Fail2ban*: Protects a Linux system from brute-force attacks. It works by reading logs and configuring the firewall to block IPs trying attacks.
- *Snort*: A **Network Intrusion Detection System** inspects all inbound and outbound network activity and identifies suspicious patterns which can indicate attacks.

After *tripwire* is installed (using the package *tripwire* in the software repository), it is needed to start the database using the option **`--init`** for the command **`tripwire`**. *Figure 8.19* shows an output example:

```
root@ubuntu:~# tripwire --init 2>/dev/null
Please enter your local passphrase:
Parsing policy file: /etc/tripwire/tw.pol
Generating the database...
*** Processing Unix File System ***
The object: "/dev/hugepages" is on a different file system...ignoring.
The object: "/dev/mqueue" is on a different file system...ignoring.
The object: "/dev/pts" is on a different file system...ignoring.
The object: "/dev/shm" is on a different file system...ignoring.
The object: "/proc/sys/fs/binfmt_misc" is on a different file system...ignoring.
Wrote database file: /var/lib/tripwire/ubuntu.example.com.twd
The database was successfully generated.
```

Figure 8.19: *Output example for command tripwire*

The option **`--check`** will show the modifications done in the filesystem from the database creation. For example, if a file is created in **`/usr/bin/`**, the section "Unix File System: will show this information":

```
===============================================================
=
Object Summary:
```

```
================================================================================
=
--------------------------------------------------------------------------------
-
# Section: Unix File System
--------------------------------------------------------------------------------
-
--------------------------------------------------------------------------------
-
Rule Name: Other binaries (/usr/bin)
Severity Level: 66
--------------------------------------------------------------------------------
-
Added:
"/usr/bin/malware"
Modified:
"/usr/bin"
```

The service **`fail2ban`** reads log files to detect possible brute-force attacks. The main directory keeping the configurations is **`/etc/fail2ban`**. For example, the SSH connections will read the log **`/var/log/auth.log`** to detect possible password brute force attacks. The command **`fail2ban-client`** with the argument **`status`** and the service will provide information about the banned IPs as is shown in *figure 8.20*:

```
root@ubuntu:~# fail2ban-client status sshd
Status for the jail: sshd
|- Filter
|  |- Currently failed: 1
|  |- Total failed:     7
|  `- File list:        /var/log/auth.log
`- Actions
   |- Currently banned: 1
   |- Total banned:     1
   `- Banned IP list:   192.168.122.145
```

Figure 8.20: *Output example for command fail2ban-client*

Rejected Ips will be added to the firewall backend in the system. *Figure 8.21* shows how it was added to the *nft* tables:

```
root@ubuntu:~# nft list chain filter  f2b-sshd
table ip filter {
        chain f2b-sshd {
                ip saddr 192.168.122.145 counter packets 29 bytes 1200 reject
                counter packets 713 bytes 49060 return
        }
}
```

Figure 8.21: *Output example for command nft*

`Snort` is a powerful suite for *Network Intrusion Detection* and *Prevention Systems*, using different rules to detect unauthorized access. The available rules are available on the official website, **https://www.snort.org/**. Running the command **`snort`** specifying the configuration with the option *-c* will process the incoming and outgoing connections and show the possible attacks. The following block shows an output example:

```
Commencing packet processing (pid=5993)
08/28-21:11:04.609587 [**] [1:1421:11] SNMP AgentX/tcp request [**]
[Classification: Attempted Information Leak] [Priority: 2] {TCP}
192.168.122.145:43439 -> 192.168.122.101:705
08/28-21:11:04.712143 [**] [1:1421:11] SNMP AgentX/tcp request [**]
[Classification: Attempted Information Leak] [Priority: 2] {TCP}
192.168.122.145:43440 -> 192.168.122.101:705
08/28-21:11:06.307016 [**] [1:1418:11] SNMP request tcp [**]
[Classification: Attempted Information Leak] [Priority: 2] {TCP}
192.168.122.145:43439 -> 192.168.122.101:161
08/28-21:11:06.407381 [**] [1:1418:11] SNMP request tcp [**]
[Classification: Attempted Information Leak] [Priority: 2] {TCP}
192.168.122.145:43440 -> 192.168.122.101:161
```

Security models

A security model provides a mechanism for supporting access security policies. Red Hat Enterprise Linux and derivatives distributions use *SELinux* and Debian, and derivatives use *AppAmor* software. Both solutions are integrated into the *Linux Kernel* to increase the security of the system.

SELinux (Security Enhanced Linux*)* defines access controls for the applications, processes, and files on a system. Using security policies, the rules define what can be accessible and what cannot. For example, *SELinux* rules can

define which directories can be used by an application or which ports can be used for listening connections.

SELinux has three states, as follows:

- *enforcing* to protect and ensure that the rules are followed.
- *permissive* will show warnings but allow the actions.
- *disabled* to not use policies in the system. The main configuration to define the state is */etc/selinux/config.*

The command **`getenforce`** shows the current state, and the command **`setenforce`** changes between states. The command **`sestatus`** provides more information. *Figure 8.22* shows the output of the commands:

```
[root@rhel ~]# getenforce
Permissive
[root@rhel ~]# setenforce Enforcing
[root@rhel ~]# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking:     actual (secure)
Max kernel policy version:      33
```

Figure 8.22: *Output example for command getenforce*

SELinux works with context (also known as labels), which have several fields: *user, role, type, and security level.* In the following *figure 8.23*, the service Apache, which has the context **`httpd_t`**, can access to the directory **`/var/www/html/`**, which has the context **`httpd_sys_content_t`**. However, Apache cannot access the directory **`/data/mysql/`**, which has the context **`mysqld_db_t`**.

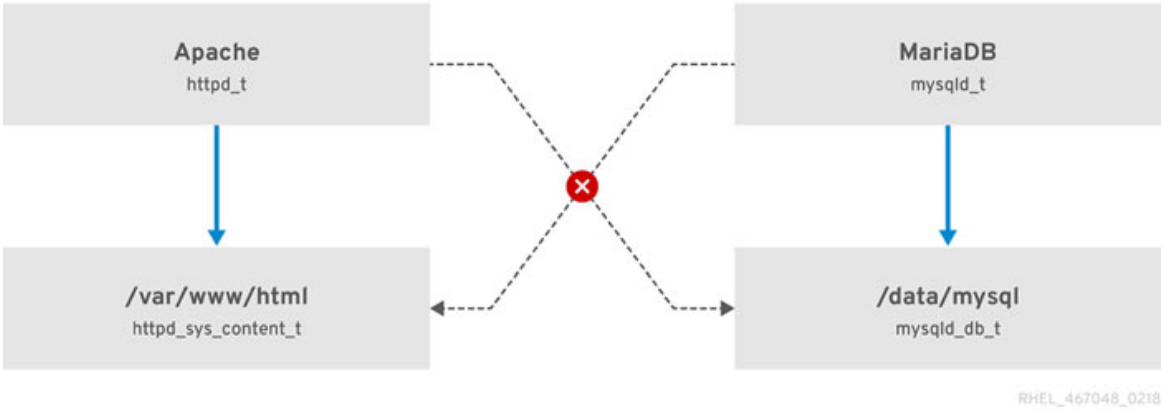


Figure 8.23: *SELinux context permissions*

Another protection provided by *SELinux* is the port the services can use. Using the command **`semanage`** with the argument **`port`** and the option *-l*, it is possible to list all the ports available for the services. In the following example in *figure 8.24*, it is shown which ports can be used by Web servers and how to add a new port to the list:

```
[root@rhel ~]# semanage port -l |grep ^http_port_t
http_port_t                    tcp      80, 81, 443, 488, 8008, 8009, 8443, 9000
[root@rhel ~]# semanage port -a -t http_port_t -p tcp 3333
[root@rhel ~]# semanage port -l |grep ^http_port_t
http_port_t                    tcp      3333, 80, 81, 443, 488, 8008, 8009, 8443,
 9000
```

Figure 8.24: *Output example of command semanage*

SELinux, by default, does not allow the Web server daemons to connect to external destinations. The command **`getsebool`** shows the current status, and the command **`setsebool`** allows to change of the value. The following *figure 8.25* shows the usage of the two commands:

```
[root@rhel ~]# getsebool httpd_can_network_connect
httpd_can_network_connect --> off
[root@rhel ~]# setsebool httpd_can_network_connect on
[root@rhel ~]# getsebool httpd_can_network_connect
httpd can network connect --> on
```

Figure 8.25: *Output example of command getsebool*

The main log for *SELinux* is the file **`/var/log/audit/audit.log`**, which includes information about the actions denied by the policies and other audit information related to the rules. The following code block shows an example of a denied action; the **`php-fpm`** command with context **`httpd_t`** tries to create a connection:

```
type=AVC msg=audit(1661798089.332:209): avc: denied { name_connect }
for pid=2948 comm="php-fpm" dest=443
scontext=system_u:system_r:httpd_t:s0
tcontext=system_u:object_r:http_port_t:s0 tclass=tcp_socket
permissive=0
```

AppArmor is installed and loaded by default on Ubuntu. In other distributions, it is possible to install it from the software repository. *AppArmor* uses profiles of an application to determine what files and permissions the application requires. *AppArmor* has two modes of execution: *complaining/learning* will log the violations, and *enforced/confined* will enforce the policy. The command

apparmor_status shows the information about how many profiles are loaded and how many are in enforce mode. *Figure 8.26* shows the output example:

```
root@ubuntu:~# apparmor_status |grep -v "^  "
apparmor module is loaded.
31 profiles are loaded.
31 profiles are in enforce mode.
0 profiles are in complain mode.
0 profiles are in kill mode.
0 profiles are in unconfined mode.
0 processes have profiles defined.
0 processes are in enforce mode.
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
0 processes are in mixed mode.
0 processes are in kill mode.
```

Figure 8.26: *Output example of command appamor_status on an Ubuntu 22.04*

Profiles are simple text files located in **/etc/appamor.d/**. It refers to a full path replacing the slashes (/) with dots (.). For example, the file **/etc/apparmor.d/usr.bin.tcpdump** references to the command **/usr/bin/tcpdump.**

The log for *AppArmor* is located inside of the file **/var/log/dmesg**, which is accessible directly or through the command **dmesg**. The profile **/etc/apparmor.d/usr.bin.tcpdump** defines some deny rules:

```
# for -F and -w
audit deny @{HOME}/.* mrwkl,
audit deny @{HOME}/.*/ rw,
audit deny @{HOME}/.*/** mrwkl,
audit deny @{HOME}/bin/ rw,
audit deny @{HOME}/bin/** mrwkl,
```

If a user is trying to open a file with the command **tcpdump** in any of these paths, it will fail, and a log will be recorded as is shown in *figure 8.27*:

```
root@ubuntu:~# tcpdump -r ~/.test.cap
tcpdump: /root/.test.cap: Permission denied
root@ubuntu:~# dmesg | tail -1
[ 2187.389656] audit: type=1400 audit(1661799780.098:46): apparmor="DENIED" oper
ation="open" profile="tcpdump" name="/root/.test.cap" pid=2414 comm="tcpdump" re
quested mask="r" denied mask="r" fsuid=0 ouid=109
```

Figure 8.27: *Testing AppArmor profiles*

It is possible to use the commands **`aa-complain`** and **`aa-enforce`** (part of the package **`apparmor-utils`***),* passing a path as an argument and setting the mode as complain or enforce. *Figure 8.28* shows how after using **`aa-complain`**, it is possible to read a file:

```
root@ubuntu:~# aa-complain /usr/bin/tcpdump
Setting /usr/bin/tcpdump to complain mode.
Warning: profile tcpdump represents multiple programs
Warning: profile tcpdump represents multiple programs
root@ubuntu:~# tcpdump -r ~/.test.cap | head -1
reading from file /root/.test.cap, link-type EN10MB (Ethernet), snapshot length
262144
19:01:39.118195 IP ubuntu.example.com.ssh > _gateway.37748: Flags [P.], seq 3026
165642:3026165774, ack 4280410004, win 501, options [nop,nop,TS val 2640140061 e
cr 1533190408], length 132
```

Figure 8.28: *Using the command aa-complain*

Network monitoring

Network monitoring consists in observe the traffic flow constantly or during a period to detect issues. It can also be used to detect attacks or to perform troubleshooting related to the network configuration in the system. This section will focus on the popular tools **`tcpdump`** and **`wireshark`**.

The command **`tcpdump`** is a powerful command-line packet analyzer with the possibility to save the captured data to a file or read the data from a file. It allows to read in a specific interface or in all interfaces in the system. The command can be executed with options and arguments, and it will show the traffic generated in the first network interface. The arguments in the command **`tcpdump`** are to specify filters, for example, only capture packets originated from a specific IP. The common options are explained in the following table:

Option	Description
-A	Print each packet (minus its link level header) in ASCII.
-c count	Exit after receiving count packets.
--count	Print only on stderr the packet count when reading capture file(s) instead of parsing/printing the packets.
-D,--list-interfaces	Print the list of the network interfaces available on the system and on which tcpdump can capture packets.
-e	Print the link-level header on each dump line. Useful to debug protocol information.
-i interface	Uses the interface specified or with word any for all of them.

--interface=interface	
-n	Do not convert addresses to names.
-r file	Read packets from a file.
-w file	Write the raw packets to file rather than printing them out.
-v	Verbose output

Table 8.4: *Common options for the command tcpdump*

The following figures show different examples using the command **`tcpdump`**.

- List all the interfaces usable in the system, as shown in *figure 8.29*:

```
root@ubuntu:~# tcpdump --list-interfaces
1.enp1s0 [Up, Running, Connected]
2.enp7s0 [Up, Running, Connected]
3.bridge0 [Up, Running, Connected]
4.any (Pseudo-device that captures on all interfaces) [Up, Running
5.lo [Up, Running, Loopback]
6.bluetooth-monitor (Bluetooth Linux Monitor) [Wireless]
7.nflog (Linux netfilter log (NFLOG) interface) [none]
8.nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
9.dbus-system (D-Bus system bus) [none]
10.dbus-session (D-Bus session bus) [none]
11.ovs-system [none, Disconnected]
```

Figure 8.29: *Output example for the command tcpdump*

- Capture the traffic for the interface *bridge0* not resolving IPs and ports and showing extra information (option *-e*) for each packet, exiting after receiving one packet. Refer to *figure 8.30*:

```
root@ubuntu:~# tcpdump -i bridge0 -n -e -c 1
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on bridge0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:00:01.759016 52:54:00:c8:55:25 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806),
length 42: Request who-has 10.0.0.220 tell 10.0.0.218, length 28
1 packet captured
4 packets received by filter
0 packets dropped by kernel
```

Figure 8.30: *Output example for the command tcpdump*

- Capture traffic and save it in a file using the option *-w* and read using the option *-r*, as shown in *figure 8.31*:

```
root@ubuntu:~# tcpdump -i bridge0 -n -e -c 1 -w example.cap
tcpdump: listening on bridge0, link-type EN10MB (Ethernet), snapshot length 26214
 bytes
1 packet captured
2 packets received by filter
0 packets dropped by kernel
root@ubuntu:~# tcpdump -r  example.cap
reading from file example.cap, link-type EN10MB (Ethernet), snapshot length 26214
20:04:08.503198 IP 10.0.0.218 > ubuntu.network-1: ICMP echo request, id 2, seq 1,
length 64
```

Figure 8.31: *Output example for the command tcpdump*

The arguments for the command **`tcpdump`** are expressions to define filters. Some of the popular ones are defined in *table 8.5*:

Filter	Description
[src / dst] host	Filter the host, optionally indicating if it is an src or a dst.
[src / dst] net	Filter the network, optionally indicating if it is an src or a dst.
[src / dst] port	Filter the port, optionally indicating if it is an src or a dst.
[src / dst] portrange	Filter a range port, optionally indicating if it is an src or a dst.
tcp / udp / icmp	Filter for one of the protocols indicated
arp	Filter for the arp protocol

Table 8.5: *Common options for the command firewall-cmd*

Full filters are available for running the command **`man pcap-filter`**. It is possible to use special words "*and*" and "*or*" to combine several expressions. *Figure 8.32* shows an example filtering from the IP 10.0.0.28 and port 80:

```
root@ubuntu:~# tcpdump -i bridge0 -n "src host 10.0.0.218 and port 80" -c 1
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on bridge0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:21:24.923973 IP 10.0.0.218.53772 > 10.0.0.220.80: Flags [S], seq 2901927797, wi
n 64240, options [mss 1460,sackOK,TS val 2755969774 ecr 0,nop,wscale 7], length 0
1 packet captured
1 packet received by filter
0 packets dropped by kernel
```

Figure 8.32: *Output example for the command tcpdump*

Wireshark is another popular solution for packet analyzing. The solution includes a CLI tool called *tshark,* which works similar to *tcpdump,* and a **Graphical User Interface** (**GUI**) to help inspect the information in the packet. *Figure 8.33* shows the output example for the command **`tshark`** filtering to the port 80 and the ip *10.0.0.218*:

```
root@ubuntu:~# tshark -i bridge0 -f "tcp port 80 and src host 10.0.0.218" -c1
Running as user "root" and group "root". This could be dangerous.
Capturing on 'bridge0'
 ** (tshark:4334) 20:37:13.112691 [Main MESSAGE] -- Capture started.
 ** (tshark:4334) 20:37:13.112998 [Main MESSAGE] -- File: "/tmp/wireshark_bridge04
CRNR1.pcapng"
    1 0.000000000   10.0.0.218 → 10.0.0.220   TCP 74 58456 → 80 [SYN] Seq=0 Win=64
240 Len=0 MSS=1460 SACK_PERM=1 TSval=2756921357 TSecr=0 WS=128
1 packet captured
```

Figure 8.33: *Output example for the command tshark.*

Figure 8.34 shows the aspect of the *Wireshark* interface:

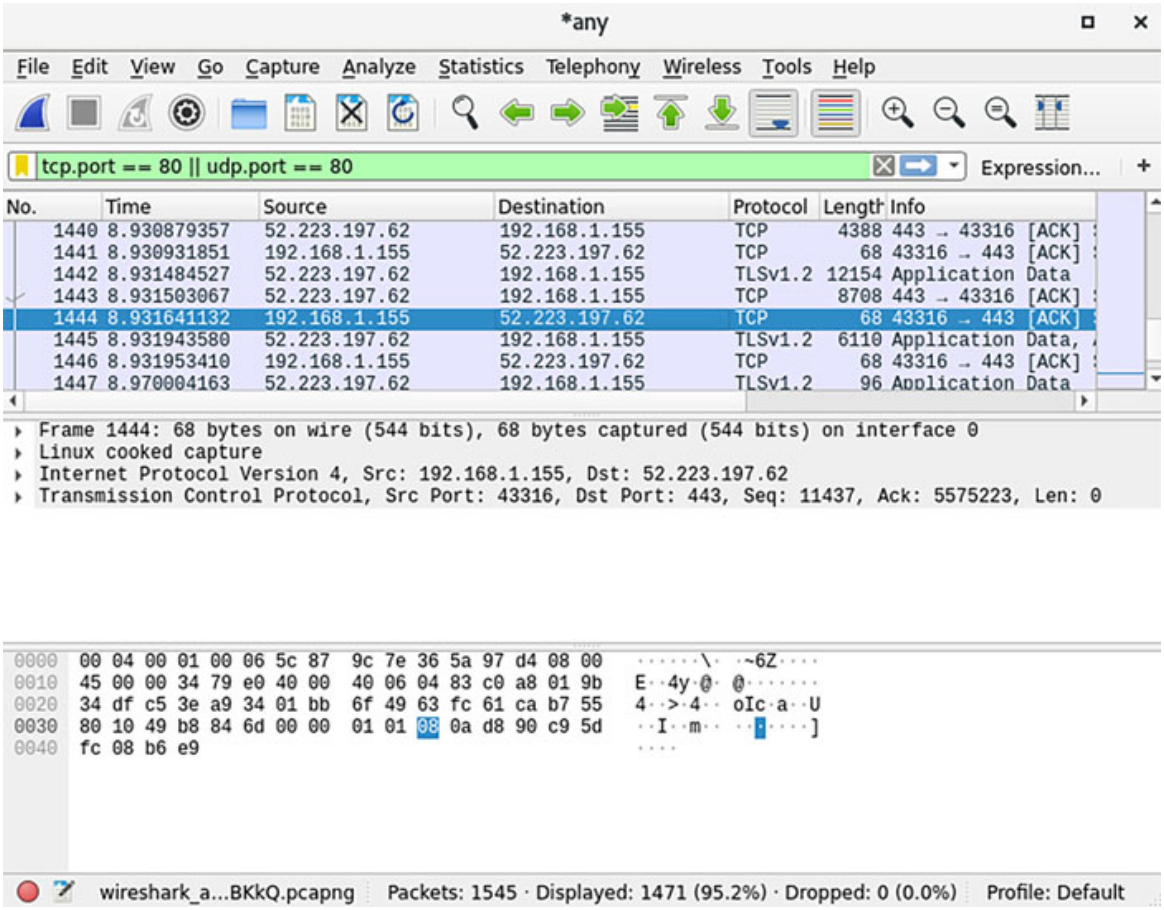


Figure 8.34: *Aspect example for the command Wireshark*

Conclusion

Security is a key part when a server is configured. This chapter has covered the main security elements in Linux distribution. Firewall is one of the main protections, and it is important to keep the rules updated. The solutions *firewalld* and *ufw* were covered, and different examples were described. Modern systems offer different services, and it is important to protect them to avoid unauthorized

access. Different recommendations were described, and tools to monitor the traffic were explained in this chapter.

Key facts

- Install and configure a firewall on Linux is a simple task.
- A Linux system can act as a gateway for other systems.
- An application can listen in a port in all the interfaces available.
- SELinux is an advanced access control solution.
- Network packet analyzing is possible with solutions such as tcpdump or wireshark.

Questions

1. What solution replaces to the traditional *iptables*?
    a. ebtables
    b. nftables
    c. bpftables
2. What is the main command to administrate *firewalld*?
    a. firewall-cmd
    b. firewall-cli
    c. firewall
3. Which directory contains the application definition for *ufw*?
    a. /etc/ufw.d/
    b. /etc/ufw/applications.d/
    c. /var/lib/ufw/applications/
4. What IP is used on IPv4 to indicate it is listening in all the interfaces?
    a. 255.255.255.255
    b. 1.1.1.1
    c. 0.0.0.0
5. What two commands are used to check if *SELinux* is enabled?
    a. getenforce

b. sestatus

c. selinux_status

6. What option in the command *tcpdump* is used to print the content packet in ASCII?

    a. -v

    b. -A

    c. --ascii

7. What is the CLI command for *wireshark*?

    a. wireshark-cli

    b. wshark

    c. tshark

Answers

1. b
2. a
3. b
4. c
5. a and b
6. b
7. c

CHAPTER 9

Network Services

Introduction

This chapter will cover the popular network services usually offered by a Linux Server. The services available for different needs are a big number; this chapter will focus on three main services: **Dynamic Host Configuration Protocol** (**DHCP**), **Domain Name System** (**DNS**), and remote access using **Secure Shell** or **Secure Socket Shell** (**SSH**).

The network service DHCP, which provides dynamic IP assignment to other elements in the network, is one of the core services in all infrastructures. This service was usually offered by network equipment, but on modern architectures, it is based on virtualization and containers. The service is based on Linux solutions.

Another core network service is DNS, which is a key component for infrastructure software. It is required to understand how it works and its different configurations to offer this service to the network.

When remote re-access to Linux systems is required, SSH is the popular and most used method used. This chapter will describe the different options to configure the service and how the client can be used for different use cases. Private and public keys will be covered to increase the security of the system.

The last part of the chapter will introduce other popular services and Linux popular software associated with them.

Structure

In this chapter, we will discuss the following topics:

- DHCP service and client
- DNS service and clients
- SSH service and SSH client
- SSH private and public keys
- Check network services available
- Other popular network services

DHCP service and client

Dynamic Host Configuration Protocol (**DHCP**) is a protocol to automatically assigns **Internet Protocol** (**IP**) addresses to the devices in the network. *Chapter 7, Network Configuration*, introduced the different layers of networking. The first layer, called the physical layer, is responsible for the transmission and reception of the data in a device. There are no IP addresses present in this layer to send the data between the nodes. Thus, it is based on the physical address. When a device without a static IP configured is booting, it asks the network to obtain an IP address. For this request, it sends a query using **User Datagram Protocol** (**UDP**) to port 67 as a destination and port number 68 as the client, and it waits for an answer. This can be seen in *figure 9.1*:

Figure 9.1: *DHCP client requesting to obtain IP*

This request, called **discover**, is sent to all the devices in the network, expecting to reach a DHCP server, as observed in the preceding *figure 9.1*. As there are no IPs in this layer, the packet will be sent to the special address **`FF:FF:FF:FF:FF:FF`**, and the node will specify its own **MAC address** for which it wants the IP address. In the ethernet packet, the special IP **`0.0.0.0`** would be included as the source IP and the broadcast IP **`255.255.255.255`** as a destination. The following excerpt from **`tcpdump`**'s output shows an example:

```
16:28:34.690916 52:54:00:1a:9c:c4 > ff:ff:ff:ff:ff:ff, ethertype IPv4
(0x0800), length 342: (tos 0x10, ttl 128, id 0, offset 0, flags [none],
proto UDP (17), length 328)
  0.0.0.0.68 > 255.255.255.255.67: [udp sum ok] BOOTP/DHCP, Request from
  52:54:00:1a:9c:c4, length 300, xid 0x746de14f, Flags [none] (0x0000)
  Client-Ethernet-Address 52:54:00:1a:9c:c4
  Vendor-rfc1048 Extensions
    Magic Cookie 0x63825363
    DHCP-Message (53), length 1: Discover
    Hostname (12), length 6: "ubuntu"
```

The DHCP service will be listening in the network, waiting for requests. When a **discover** request is received, it will check if there is any IP available from the range configured to offer. It will also check if the **MAC address** that performed the request has some special configuration, such as a specific assigned IP. If the DHCP service can

give an IP to the requester, an answer is sent to the original request for an IP address, offering one and indicating its **MAC address**. This answer is called **offer**. This can be seen in *figure 9.2*:

Figure 9.2: *DHCP Service offering an IP*

The following excerpt from `tcpdump`'s output shows the **offer** answer from a DHCP service to the node that requested for an IP. In this case, the DHCP Service is offering the IP `192.168.122.175`, and the communication is from the **MAC address** of the DHCP Server:

```
16:28:37.694583 52:54:00:d8:cd:26 > 52:54:00:1a:9c:c4, ethertype IPv4
(0x0800), length 342: (tos 0xc0, ttl 64, id 3269, offset 0, flags
[none], proto UDP (17), length 328)
  192.168.122.1.67 > 192.168.122.175.68: [udp sum ok] BOOTP/DHCP, Reply,
  length 300, xid 0x746de14f, Flags [none] (0x0000)
  Your-IP 192.168.122.175
  Server-IP 192.168.122.1
  Client-Ethernet-Address 52:54:00:1a:9c:c4
  Vendor-rfc1048 Extensions
    DHCP-Message (53), length 1: Offer
    Server-ID (54), length 4: 192.168.122.1
    Lease-Time (51), length 4: 3600
    Subnet-Mask (1), length 4: 255.255.255.0
    BR (28), length 4: 192.168.122.255
    Default-Gateway (3), length 4: 192.168.122.1
    Domain-Name-Server (6), length 4: 192.168.122.1
```

At this point, the client will obtain the possible IP that can be used and other information such as default gateway, DNS information, and subnet. It also indicates how often the IP needs to be renewed; in the previous example, this time was at 3,600 seconds. If the client wants this IP, they need to send another request, called **request**, to be able to use it. This can be seen in *figure 9.3*:

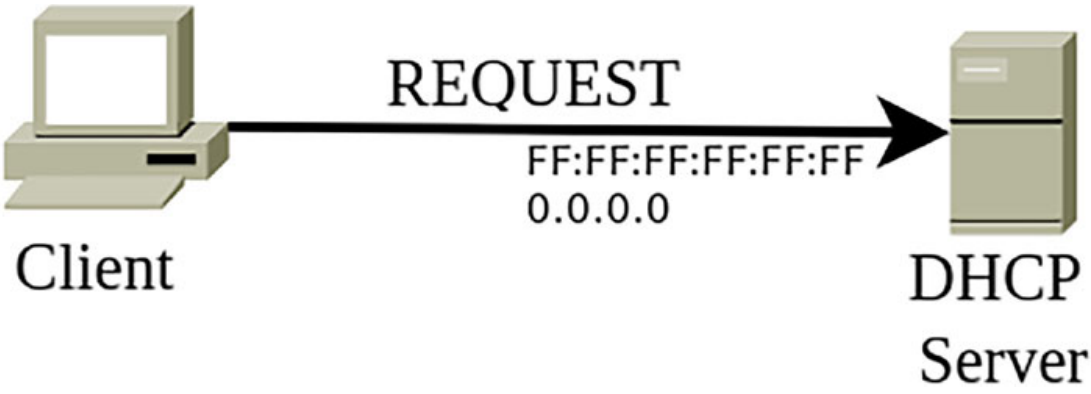


Figure 9.3: *Client requesting the IP offered*

The following excerpt shows an example of the **request** packet:

```
16:28:37.694766 52:54:00:1a:9c:c4 > ff:ff:ff:ff:ff:ff, ethertype IPv4
(0x0800), length 342: (tos 0x10, ttl 128, id 0, offset 0, flags [none],
proto UDP (17), length 328)
  0.0.0.0.68 > 255.255.255.255.67: [udp sum ok] BOOTP/DHCP, Request from
  52:54:00:1a:9c:c4, length 300, xid 0x746de14f, Flags [none] (0x0000)
  Client-Ethernet-Address 52:54:00:1a:9c:c4
  Vendor-rfc1048 Extensions
    DHCP-Message (53), length 1: Request
    Server-ID (54), length 4: 192.168.122.1
    Requested-IP (50), length 4: 192.168.122.175
    Hostname (12), length 6: "ubuntu"
```

When the DHCP Service receives the request, it will answer with the confirmation. This confirmation is called **acknowledge**, and can be seen in *figure 9.4*:

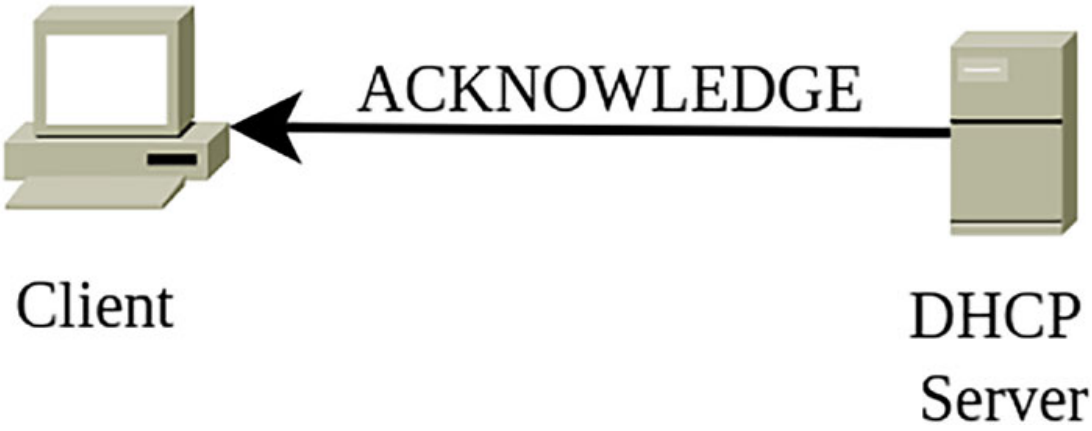


Figure 9.4: *DHCP service acknowledges the use of the IP*

The following excerpt shows the content of the **acknowledge** packet:

```
16:28:37.696436 52:54:00:d8:cd:26 > 52:54:00:1a:9c:c4, ethertype IPv4
(0x0800), length 342: (tos 0xc0, ttl 64, id 3270, offset 0, flags
```

```
[none], proto UDP (17), length 328)
  192.168.122.1.67 > 192.168.122.175.68: [udp sum ok] BOOTP/DHCP, Reply,
  length 300, xid 0x746de14f, Flags [none] (0x0000)
  Your-IP 192.168.122.175
  Server-IP 192.168.122.1
  Client-Ethernet-Address 52:54:00:1a:9c:c4
  Vendor-rfc1048 Extensions
    DHCP-Message (53), length 1: ACK
    Server-ID (54), length 4: 192.168.122.1
    Lease-Time (51), length 4: 3600
    Subnet-Mask (1), length 4: 255.255.255.0
    BR (28), length 4: 192.168.122.255
    Default-Gateway (3), length 4: 192.168.122.1
    Domain-Name-Server (6), length 4: 192.168.122.1
    Hostname (12), length 6: "ubuntu"
```

Now, the client can assign and use the IP for the time specified in the **Lease time**. After that time is over, the client needs to send a request to renew the lease. *Figure 9.5* shows the diagram of the *DHCP* process:

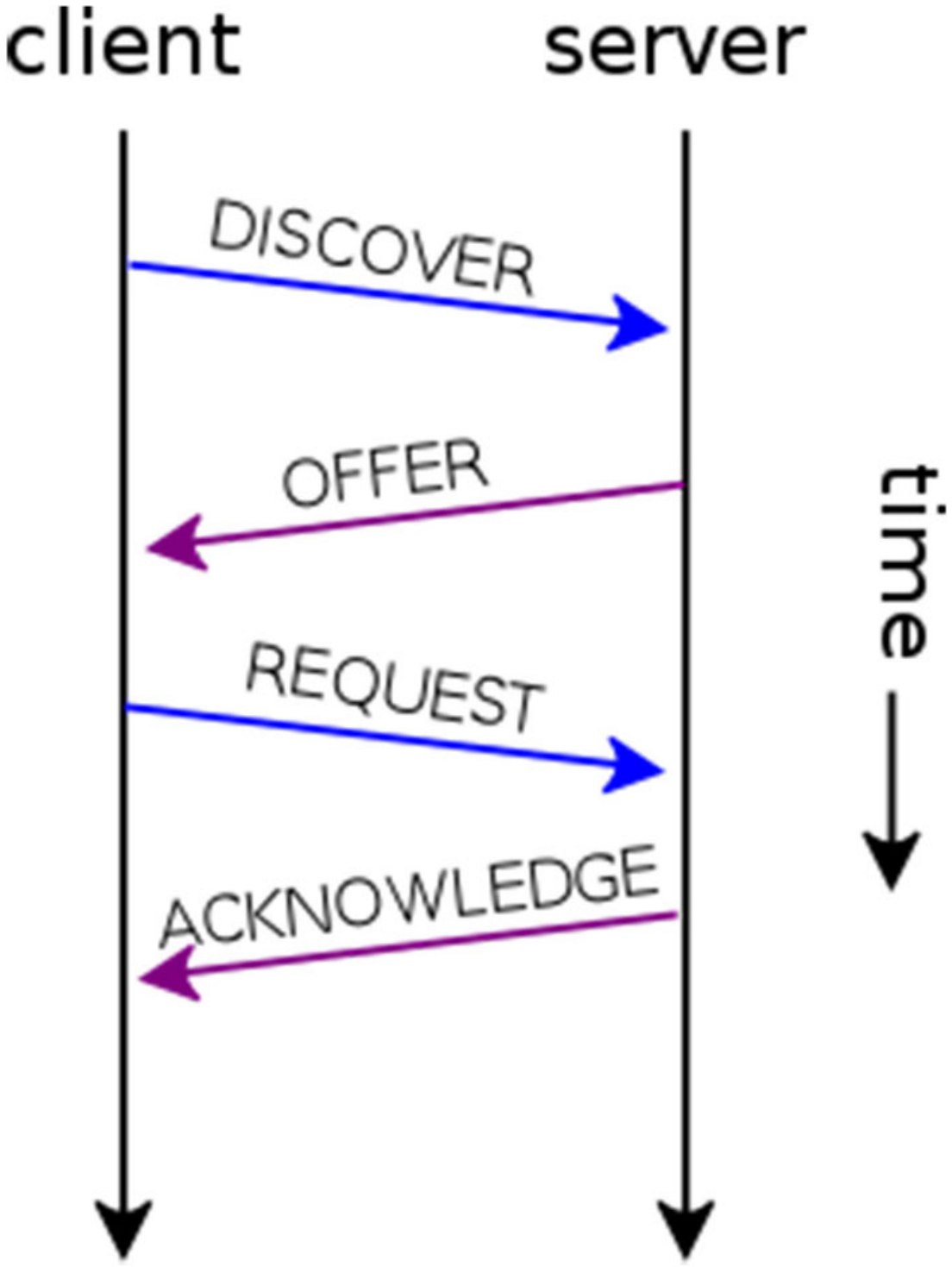


Figure 9.5: *DHCP diagram. Source: Wikimedia*

Linux DHCP servers and client

The most popular DHCP Server for Linux is **Internet Systems Consortium** (**ISC**), known as `dhcpd` service. The first release was in 1999, being one of the most mature

and used services during the last decades. Another popular solution is **dnsmasq**, which is a lightweight DNS, TFTP, and DHCP server.

The installation of ISC-DHCP on distributions based on DEB packages requires to install the package **isc-dhcp-server** and **isc-dhcp-client** (for the client utilities). On distributions based on RPM packages, the names are **dhcp-server** and **dhcp-client**. The main configuration for the service is the file **/etc/dhcp/dhcpd.conf** for IPv4 and **/etc/dhcp/dhcpd6.conf** for IPv6. The files contain global configurations, subnet definitions, host configurations, and other configurations, such as classes, pools, and **shared-network** declarations that are not covered in this book. The global configuration are lines ending with a semicolon and defines different options, as observed in the following excerpt from the default configuration:

```
# option definitions common to all supported networks…
option domain-name "example.org";
option domain-name-servers ns1.example.org, ns2.example.org;
default-lease-time 600;
max-lease-time 7200;
```

For the subnet definition, it will indicate the subnet and the netmask, and the specific configuration for that subnet, between curly brackets ({}). In the following example for the subnet **192.168.100.0/24**, 25 IP addresses would be available, the *DNS* server would be **1.1.1.1**, and the gateway **192.168.100.1**, among other options:

```
subnet 192.168.100.0 netmask 255.255.255.0 {
  range 192.168.100.25 192.168.100.50;
  option domain-name-servers 1.1.1.1;
  option domain-name "mycompany.com";
  option subnet-mask 255.255.255.0;
  option routers 192.168.100.1;
  option broadcast-address 192.168.100.254;
  default-lease-time 600;
  max-lease-time 7200;
}
```

It is possible to specify specific parameters for one client. For that, the keyword **host** is used, followed by a label and the options desired, as the following excerpt shows:

```
host rhelsystem {
  hardware ethernet 52:54:00:57:d0:9f;
  fixed-address 192.168.100.222;
}
```

After the file is configured, the service to be started (or restarted) is **isc-dhcp-server** on Ubuntu/Debian-based systems and **dhcpd** on RHEL-based systems. Using the command **journalctl** to check the logs from the system, it will be possible to obtain

information about the requests and the IP addresses assigned, as is shown in the following excerpt:

```
dhcpd[1151]: Server starting service.
dhcpd[1151]: DHCPDISCOVER from 52:54:00:57:d0:9f via enp8s0
dhcpd[1151]: DHCPOFFER on 192.168.100.222 to 52:54:00:57:d0:9f via
enp8s0
dhcpd[1151]: DHCPREQUEST for 192.168.100.222 (192.168.100.1) from
52:54:00:57:d0:9f via enp8s0
dhcpd[1151]: DHCPACK on 192.168.100.222 to 52:54:00:57:d0:9f via enp8s0
```

As a client to test the **DHCP requests**, the utility **`dhclient`** offers the possibility to configure one interface with a dynamic IP instead, to use the system configuration (using files, **`NetworkManager`**, or **`netplan`**). This utility allows to specify the option **`-v`** to enable verbose logging. The option **`-r`** is used to release the current lease and stop the client running for the interface. *Figure 9.6* shows an example of the usage, and the output is shown:

```
[root@rhel ~]# dhclient -v enp7s0
Internet Systems Consortium DHCP Client 4.4.2b1
Copyright 2004-2019 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/enp7s0/52:54:00:57:d0:9f
Sending on   LPF/enp7s0/52:54:00:57:d0:9f
Sending on   Socket/fallback
DHCPDISCOVER on enp7s0 to 255.255.255.255 port 67 interval 5 (xid=0x464c887f)
DHCPOFFER of 192.168.100.222 from 192.168.100.1
DHCPREQUEST for 192.168.100.222 on enp7s0 to 255.255.255.255 port 67 (xid=0x464c887f)
DHCPACK of 192.168.100.222 from 192.168.100.1 (xid=0x464c887f)
bound to 192.168.100.222 -- renewal in 284 seconds.
```

Figure 9.6: *Output example for command* ***dhclient***

The files where the DHCP server keeps the leases in the files are **`/var/lib/dhcp/dhcpd.leases`** for IPv4 and **`/var/lib/dhcp/dhcpd6.leases`** for IPv6.

For the software **`dnsmasq`**, it is required to install the package with the same name. The main configuration is **`/etc/dnsmasq.conf`**, which is used for configuration not only for DHCP but also for DNS and TFTP. This file can contain configuration for IPv6 too. An example of configuration to provide DHCP service is shown in the following excerpt:

```
listen-address=192.168.100.1
domain=mycompany.com
dhcp-range=192.168.100.25,192.168.100.50,2h
dhcp-host=52:54:00:57:d0:9f,192.168.100.222
dhcp-option=option:router,192.168.100.1
dhcp-option=option:dns-server,1.1.1.1
```

Once the service is started, the log will appear in the **`journalctl`** when a request is received. The following excerpt shows the IP assignation for the host configured with a specific IP:

```
ubuntu dnsmasq-dhcp[1889]: DHCPDISCOVER(enp8s0) 192.168.100.222
52:54:00:57:d0:9f
ubuntu dnsmasq-dhcp[1889]: DHCPOFFER(enp8s0) 192.168.100.222
52:54:00:57:d0:9f
ubuntu dnsmasq-dhcp[1889]: DHCPREQUEST(enp8s0) 192.168.100.222
52:54:00:57:d0:9f
ubuntu dnsmasq-dhcp[1889]: DHCPACK(enp8s0) 192.168.100.222
52:54:00:57:d0:9f
```

The file containing the leases for **`dnsmasq`** is **`/var/lib/misc/dnsmasq.leases`**.

DNS service and clients

The **Domain Name System** (**DNS**) is a service to convert domain names to IP addresses and vice versa. The structure is based on a hierarchy, where the top level contains the *root nameservers*. Currently, these *root nameservers* are 13 servers geolocated in different parts of the word. These servers have the syntax *letter.root-servers.org,* where the *letter* currently is going from the letters *a* to *m*. The first query to the *root nameservers* is to obtain the servers for the **top-level domain** (**TLD**), responsible for the second level in the hierarchy. These servers are associated with the top-level domain, such as *.com*, *.net,* and *.org,* the country code TLDS (for example, *.in, .es*, and *.jp*), and new *TLDs* (for example, *.tech* and *.space*). These **top-level domain** servers would provide the **domain servers** for the domain specified. These domains are responsible to provide the answer for the domain or subdomain queried. DNS can contain complex configurations indicating other **nameservers** for subdomains. *Figure 9.7* shows the hierarchy with two domain examples:

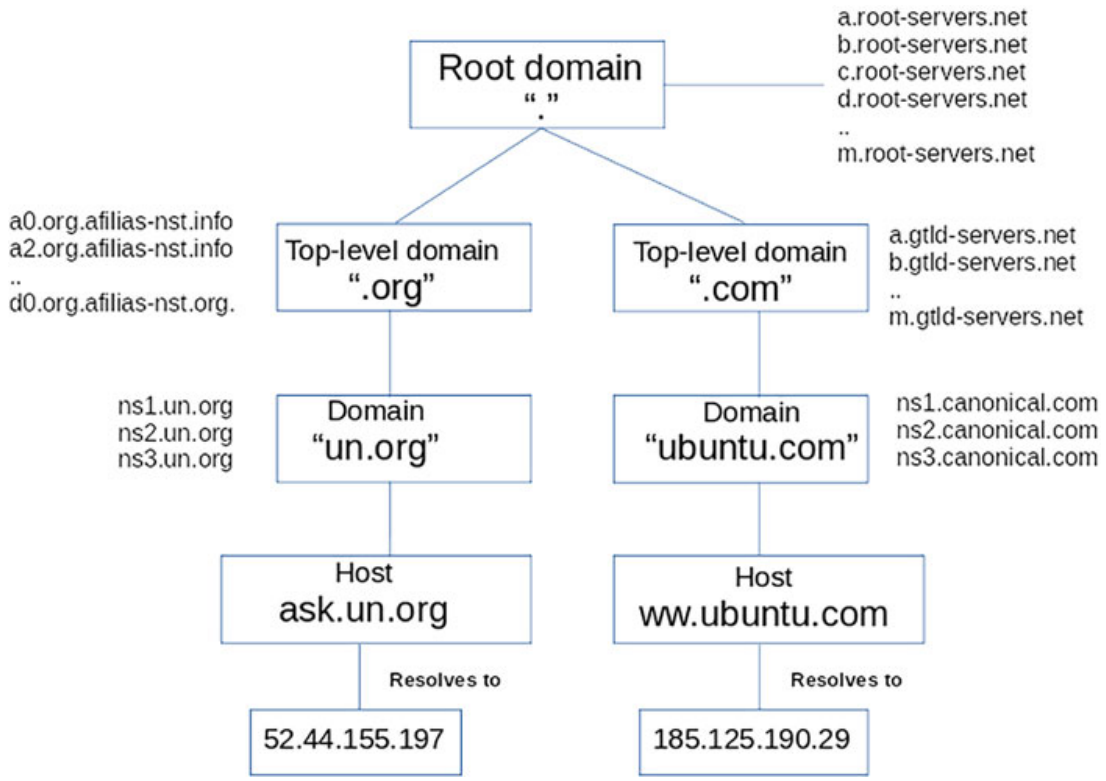


Figure 9.7: *DNS hierarchy example*

The query uses the UDP protocol, and the destination port is 53. A DNS service can act in different modes:

- **Authoritative**: It is responsible for a domain or several domains and answers queries related to them only.
- **Forwarding**: The DNS service is not responsible for any domain, and all the requests are forwarded to an **upstream** DNS server.
- **Cache**: Similar to **forwarding** but keeping a cache for the client requests. This is a popular configuration to avoid multiple requests to upstream servers.

It is possible to combine the different modes, such that the **Authoritative** and **Cache** mode, in the same server. This DNS Service will answer for the domains for which it is responsible. For the rest of the domains, it will forward the request to the **upstream** DNS server configured, caching the client requests.

New technologies and modern software depend on DNS for the correct functioning. Correct internal and external resolving is required for the software to communicate between services and to pull required software and artifacts from repositories, such as container images.

The DNS records configured inside a DNS server are of different types, as can be seen in *table 9.1*:

Record Type	Description
A	Resolves a hostname to an IP
AAAA	Resolves a hostname to an IP, for IPv6.
CNAME	Specifies an alias for a hostmae
PTR	Resolves an IP to a hostname
NS	Specifies the name servers for a domain or subdomain.
MX	Specifies the mail servers for a domain or subdomain.
SRV	Specifies host and ports for a specific service.
TXT	Specify a text record for a hostname. Used for checking the owner of the domain and different domain configurations.
SOA	Stands for "start of authority", and provides admin information for the domain and information about the last update and refreshing times.

Table 9.1: *DNS record types*

The records *A* and *AAAA* allow to use the asterisk (*) to create a wildcard record. For example, creating a wildcard record **`*.dev.example.com`** pointing to the IP **`192.168.122.222`**, the DNS server will answer for all the hostnames with that subdomain to that IP. In that example, the hostname **`web.dev.example.com`** or **`something.dev.example.com`** resolves to the same IP.

Linux DNS servers and client

The current popular software for the DNS Service is *Bind 9*, which is provided by **Internet Systems Consortium** (**ISC**). The initial release for *Bind* was in 1986, but it is updated on a regular basis. As described previously, a popular and lightweight alternative is **`dnsmasq`**.

For the installation of Bind 9 *(*also known commonly as *named*) in Ubuntu-based systems, the package is called **`bind9`** for the server and **`bind9-utils`** for the client tools. On Red Hat Enterprise Linux and derivates, the package is called **`bind`** for the DNS server and **`bind-utils`** for the client tools. Once the package is installed, the service name is called **named**, and the main configuration is **`/etc/bind/named.conf`** for the package **`bind9`** and **`/etc/named.conf`** for the package **`bind`**. In Ubuntu, the default **`named.conf`** includes other files for options and **zones** definition:

```
include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
```

On Red Hat Enterprise Linux, the content of default **`named.conf`** includes options and references to the default *zones*:

```
options {
```

```
  listen-on port 53 { 127.0.0.1; };
  listen-on-v6 port 53 { ::1; };
  directory  "/var/named";
(… omitted …)
  allow-query     { localhost; };
(… omitted …)
};
logging {
   channel default_debug {
         file "data/named.run";
         severity dynamic;
   };
};
zone "." IN {
  type hint;
  file "named.ca";
};
include "/etc/named.rfc1912.zones";
include "/etc/named.root.key";
```

Options in *Bind 9* define different options such as in which *IP address* the *DNS service* will listen for queries (**`listen-on`** and **`listen-on-v6`***),* the main directory to keep data (**`directory`**), who is allowed to use the service (**`allow-query`***)*, and define the *upstream* servers *(*forwarders*)*. It is possible to define the *forwarders* section to indicate the upstream *DNS servers* to forward the request if the domain is not one of managed by the service.

```
    forwarders {
      1.1.1.1;
      1.0.0.1;
    };
```

A **zone** refers to the domain definition and, by default, includes reference to the *root nameservers* and local configuration. An administrator will generate a new file with the definition and include it in the configuration. A *zone* definition example is the following:

```
zone "example.com" {
   type master;
   file "/etc/bind/db.example.com";
};
```

The file referenced inside of the *zone* will contain the definition of the domain *(SOA* record) and the rest of the records for that domain. An example definition is as follows:

```
@   IN SOA  dns.example.com   root.example.com. (
                               2022090401      ; serial
```

```
                                1D              ; refresh
                                1H              ; retry
                                1W              ; expire
                                10M )            ; minimum
example.com.    IN    NS      dns.example.com.
example.com.    IN    NS      dns2.example.com.
dns             IN    A       10.20.0.2
dns2            IN    A       10.20.0.3
@               IN    MX      10 mx1.example.com.
@               IN    MX      20 mx2.example.com.
www             IN    A       10.20.0.100
web             IN    CNAME   www
mx1             IN    A       10.20.0.102
mx2             IN    A       10.20.0.103
```

This sample configuration defines the following:

- The *SOA* record defines:
  - The primary *DNS server* is *dns.example.com*
  - The contact mail for the domain is *root@example.com*
  - The last update for this *zone* was 2022-09-04, and the last two digits indicate the number of updates on the same day.
  - The *refresh* to synchronize the zone between servers (when the servers are configured as primary/secondary mode).
  - When to retry if the communication between the secondary to the primary was not possible.
  - How long a secondary will still consider the zone valid if communication with the primary is not possible.
  - If a client queries for a record that does not exist, specify how the client will receive the same message before to check again.
- The name server (NS record) for the domain are **`dns.example.com`** and **`dns2.example.com`**
- The IP addresses for the nameservers are **`10.20.0.2`** and **`10.20.0.3`**.
- The mail servers for the domain are *mx1.example.com* and *mx2.example.com*
- The mail servers resolve to **`10.20.0.102`** and **`10.20.0.103`**
- The domain *www.example.com* resolves to **`10.20.0.100`**
- An alias domain, *web.example.com* will resolve to the IP address assigned to **www.example.com**

Some considerations about the *zone* definition are as follows:

- The at *(@)* references to the domain configured.
- The domains, for example, for *NS* or *MX*, references need to end with a dot (.).

For the reverse resolution, the special domain **Address and Routing Parameter Area** (**.arpa**) is used. A zone definition example is shown as follows:

```
zone "0.20.10.in-addr.arpa" IN {
   type master;
   file "/etc/bind/db.0.20.10";
};
```

An example of the content for the file **`/etc/bind/db.0.20.10`** is the following:

```
@   IN SOA  dns.example.com.  root.example.com. (
                                2020012511      ; serial
                                1D              ; refresh
                                1H              ; retry
                                1W              ; expire
                                10M )            ; minimum
@       IN      NS      dns.example.com.
@       IN      NS      dns2.example.com.
2       IN      PTR     dns.example.com.
3       IN      PTR     dns2.example.com.
100     IN      PTR     wwww.example.com.
```

The package includes two commands to check the syntax for configuration and zones: **`named-checkconf`** and **`named-checkzone`**. If any problem is found, it will be reported as an output. After the service is configured, it is possible to reload the configuration without restarting the service, using **`systemctl reload named`**.

The recommended tool to query and perform troubleshooting related to *DNS* is called *dig*, which is included in the client package for *Bind 9*. This command allows us to perform different queries and indicate which server to use for those. It is also possible to increase the information to be shown, converting it to a perfect tool to ensure that the configuration is the desired one. *Figure 9.8* shows a simple example of querying for a domain using the server *localhost*:

```
root@ubuntu:~# dig @localhost www.example.com

; <<>> DiG 9.18.1-1ubuntu1.1-Ubuntu <<>> @localhost www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9117
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 3da76ab6d281d779010000006314b74009ab8cfeb6bcdbd6 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        10800   IN      A       10.20.0.100

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(localhost) (UDP)
;; WHEN: Sun Sep 04 14:33:36 UTC 2022
;; MSG SIZE  rcvd: 88
```

Figure 9.8: *Output example for command dig*

It is possible to add the argument *+short* to show only the answer and not the information related to the query. *Figure 9.9* shows different queries with distinct record types:

```
root@ubuntu:~# dig @localhost SOA example.com +short
dns.example.com.example.com. root.example.com. 2022090401 86400 3600 604800 10800
root@ubuntu:~# dig @localhost NS example.com +short
dns.example.com.
dns2.example.com.
root@ubuntu:~# dig @localhost MX example.com +short
20 mx2.example.com.
10 mx1.example.com.
root@ubuntu:~# dig @localhost CNAME web.example.com +short
www.example.com.
root@ubuntu:~# dig @localhost A web.example.com +short
www.example.com.
10.20.0.100
```

Figure 9.9: *Output example for command dig*

The command **`dig`** allows to include as an argument the option *+trace*, which will show all the queries and a verbose information, useful to do troubleshooting and to understand all the steps in a query. *Figures 9.10* to *9.13* show the different parts of the output:

- Perform the query to obtain the IP address for **www.ubuntu.com**. The first query is to obtain the *root nameservers*, as shown in *figure 9.10*:

```
root@ubuntu:~# dig +trace A www.ubuntu.com

; <<>> DiG 9.18.1-1ubuntu1.1-Ubuntu <<>> +trace A www.ubuntu.com
;; global options: +cmd
.                       7190    IN      NS      g.root-servers.net.
.                       7190    IN      NS      i.root-servers.net.
.                       7190    IN      NS      b.root-servers.net.
.                       7190    IN      NS      l.root-servers.net.
.                       7190    IN      NS      j.root-servers.net.
.                       7190    IN      NS      h.root-servers.net.
.                       7190    IN      NS      a.root-servers.net.
.                       7190    IN      NS      k.root-servers.net.
.                       7190    IN      NS      e.root-servers.net.
.                       7190    IN      NS      m.root-servers.net.
.                       7190    IN      NS      d.root-servers.net.
.                       7190    IN      NS      c.root-servers.net.
.                       7190    IN      NS      f.root-servers.net.
;; Received 239 bytes from 127.0.0.53#53(127.0.0.53) in 0 ms
```

Figure 9.10: *Output example for command dig*

- The second part of the output is the output of querying to one of the *root nameservers* for the *name servers* specific to the .com *TLD*. This can be seen in *figure 9.11*:

```
com.                    172800  IN      NS      e.gtld-servers.net.
com.                    172800  IN      NS      b.gtld-servers.net.
com.                    172800  IN      NS      j.gtld-servers.net.
com.                    172800  IN      NS      m.gtld-servers.net.
com.                    172800  IN      NS      i.gtld-servers.net.
com.                    172800  IN      NS      f.gtld-servers.net.
com.                    172800  IN      NS      a.gtld-servers.net.
com.                    172800  IN      NS      g.gtld-servers.net.
com.                    172800  IN      NS      h.gtld-servers.net.
com.                    172800  IN      NS      l.gtld-servers.net.
com.                    172800  IN      NS      k.gtld-servers.net.
com.                    172800  IN      NS      c.gtld-servers.net.
com.                    172800  IN      NS      d.gtld-servers.net.
com.                    86400   IN      DS      30909 8 2 E2D3C916F6DEEAC73294E82
68FB5885044A833FC5459588F4A9184CF C41A5766
com.                    86400   IN      RRSIG   DS 8 1 86400 20220917040000 20220
904030000 20826 . n+t2f4RsIc6pIaAYx5UTMBTaqRLxJPoQaMTIiyYUhtXpSZpctzqF1VUI rtMh73
07sjsRN/3qHpB0u6Rd92JSggJJDgFKd73ut0TgqUYksIUECLMh Z7cQhE/0QEcFUWYH8NONVvhJlCzw4O
y1BV6K0FldVUmsIxuSxDCj4pjv 36/A7EZlDwEVVkN3OBHy6GI1LezCjxlFofmlEuSJ2ZmUvHePglImGA
sY hKzVXVvQwmUVBQxuYhqBeQ2QULwDHEvKFfv6g6rWxiDg1CXG9dZZDVRz rqgyl0s88T8FEpNBEgBjV
fYnxhFs4aD0ZnIqq99P0kB1mFD1AYoIXDpv xnJrjw==
;; Received 1174 bytes from 192.58.128.30#53(j.root-servers.net) in 39 ms
```

Figure 9.11: *Output example for command dig*

- The third part of the output is the request to one of the *TLD nameservers*, querying what are the *name servers* for the domain, in this case, *ubuntu.com*. Refer to *figure 9.12*:

```
ubuntu.com.             172800  IN      NS      ns1.canonical.com.
ubuntu.com.             172800  IN      NS      ns2.canonical.com.
ubuntu.com.             172800  IN      NS      ns3.canonical.com.
CK0POJMG874LJREF7EFN8430QVIT8BSM.com. 86400 IN NSEC3 1 1 0 - CK0Q2D6NI4I7EQH8NA30
NS61O48UL8G5 NS SOA RRSIG DNSKEY NSEC3PARAM
CK0POJMG874LJREF7EFN8430QVIT8BSM.com. 86400 IN RRSIG NSEC3 8 2 86400 202209080424
38 20220901031438 32298 com. Ish+JXDpKB/qpU9uJgYTHIAw+iHbD2l1s/dJrfNMelfINS3ug8Tr
Hrvw wnjBXLAH47wimNmMPFxBT4Lt1HKq64xeiRIxO2SfCALgn1LSf6UJvBFl mEZtFlYJpr2tI8cZmqv
DtCfxwdtbxpadpkZHCLyfJMrKKkgE2u4/KQSr M4QqikaqSAwd+REC5TSmwmtwE6fVgiYJDULKcxOWLXG
h+w==
894IO8AM9NDQ8VM84GPASGU0QDHFLFS1.com. 86400 IN NSEC3 1 1 0 - 894J39AFAALJBJGQRSTB
RTC3GOUCB42H NS DS RRSIG
894IO8AM9NDQ8VM84GPASGU0QDHFLFS1.com. 86400 IN RRSIG NSEC3 8 2 86400 202209080554
30 20220901044430 32298 com. bPgCZmhBhzfTCkGUHhupfkDYdayzVm4tjCt6tMmPf79uD/32SoKJ
cT8Z JR5ZVwa9I6vw6UBfo5fm2p3D0HWp+qP+iA1FpQYd5bj0l2wlW+a5oU2b 67mtSbPVDHRX09cocwM
0KGHBmIYanMmmN/xX85szXQwZdn9D+LQYQk4Y KrKNZw1vM+krSx2/8QoGJDSt9U4ZAwYnBuDTNZf01VH
BaQ==
;; Received 776 bytes from 192.33.14.30#53(b.gtld-servers.net) in 47 ms
```

Figure 9.12: *Output example for command dig*

- The last step is to query one of the *nameservers* for the domain about the host, as shown in *figure 9.13*:

```
www.ubuntu.com.         60      IN      A       185.125.190.20
www.ubuntu.com.         60      IN      A       185.125.190.21
www.ubuntu.com.         60      IN      A       185.125.190.29
ubuntu.com.             172800  IN      NS      ns3.canonical.com.
ubuntu.com.             172800  IN      NS      ns1.canonical.com.
ubuntu.com.             172800  IN      NS      ns2.canonical.com.
;; Received 186 bytes from 185.125.190.66#53(ns2.canonical.com) in 47 ms
```

Figure 9.13: *Output example for command dig*

Bind 9 allows advanced features like dynamic updates of the records (without requiring to update the configuration) and advanced configurations, such as having different answers for the same zone, depending on who is performing the query. For example, the same zone for external users will reply with some public addresses, and if the query is performed from the internal network, it will answer with private addresses.

The software **`dnsmasq`** can be used to configure DNS domains with a simpler configuration, and usually for configuring internal domains in the network. As described previously, the main configuration is **`/etc/dnsmasq.conf`**, and an example configuration for *DNS* is shown as follows:

```
auth-zone=example.com
auth-server=dns.example.com,10.20.0.1
auth-sec-servers=dns2.example.com
auth-soa=2022090401,root.example.com,1200,120,604800
local=/example.com/192.168.122.43
domain=example.com
host-record=dns.example.com,10.20.0.2
host-record=dns2.example.com,10.20.0.3
```

```
host-record=www.example.com,10.20.0.100
mx-host=example.com,mx1.example.com,10
mx-host=example.com,mx2.example.com,20
host-record=mx1.example.com,10.20.0.102
host-record=mx2.example.com,10.20.0.103
cname=web.example.com,www.example.com
```

Other popular client utilities for DNS queries are as follows:

- Command **`host`**: It shows a simple output about the request performed.

```
root@ubuntu:~# host -t ns example.com  localhost
Using domain server:
Name: localhost
Address: 127.0.0.1#53
Aliases:

example.com name server dns.example.com.
example.com name server dns2.example.com.
```

Figure 9.14: *Output example for command dig*

- Command **`nslookup`**: It can act as an interactive command or as simple command line request command.
    - Using the arguments to specify server and query type:

```
root@ubuntu:~# nslookup -type=NS example.com localhos
Server:         localhost
Address:        127.0.0.1#53

example.com     nameserver = dns.example.com.
example.com     nameserver = dns2.example.com.
```

Figure 9.15: *Output example for command nslookup*

    - Using the interactive mode of the command:

```
root@ubuntu:~# nslookup
> server localhost
Default server: localhost
Address: 127.0.0.1#53
> set type=NS
> example.com
Server:         localhost
```

```
Server:         localhost
Address:        127.0.0.1#53

example.com     nameserver = dns.example.com.
example.com     nameserver = dns2.example.com
```

Figure 9.16: *Output example for command nslookup*

SSH service and SSH client

The network protocol **Secure Shell** or **Secure Socket Shell** (**SSH**) gives the users a secure way to remotely access a Linux system. This system can be in the same network, a different network, or accessible through internet. The port used is 22, and the protocol used is **Transmission Control Protocol** (**TCP**). This protocol was introduced in 1995 with version 1 (*SSH-1)*; in the year 2006, version 2 (*SSH-2)* was adopted as a standard.

The most popular implementation of *SSH* is the suite of utilities named **OpenSSH**. This includes the server and client utilities. Other utilities use the protocol SSH for different functionalities; the most popular is to copy the file between servers using **Secure Copy** (**scp**) or **rsync**. Another popular option is **SSH File Transfer Protocol** (**SFTP**), an alternative to the **File Transfer Protocol** (**FTP**).

The SSH service named *sshd* is included in the package called **`openssh-server`**; the client *ssh* (which usually is installed by default on all Linux distributions) is part of the package **`openssh-client`** (or **`openssh-clients`** in some distributions). The main configuration for the server is **`/etc/ssh/sshd_config`**. The following block shows the default uncommented options in an Ubuntu server system:

```
Include /etc/ssh/sshd_config.d/*.conf
KbdInteractiveAuthentication no
UsePAM yes
X11Forwarding yes
PrintMotd no
AcceptEnv LANG LC_*
Subsystem sftp /usr/lib/openssh/sftp-server
PasswordAuthentication yes
```

Table 9.2 describe some of the popular options available to be configured in the file **`sshd_config`**:

SSH Option	Description
AllowTcpForwarding	Specifies whether TCP forwarding is permitted. Options are *yes* (default), *no,* and *local*.
AllowUsers	List of accepted user names, separated by user spaces
AuthenticationMethods	Specifies the allowed authentication methods.
Ciphers	Specifies the ciphers allowed. Multiple ciphers must be comma-separated.
DenyGroups	List of denied group names, separated by spaces.
DenyUsers	List of denied user names, separated by spaces.
DisableForwarding	Disables all forwarding features, including X11, ssh-agent(1), TCP, and StreamLocal.
ListenAddress	Specifies the local addresses that should listen on.
PasswordAuthentication	Specifies whether password authentication is allowed. The default is yes.
PermitRootLogin	Specifies whether the root can log in. The argument must be *yes*, *prohibit-password*,*forced-commands-only*, or *no*. The default is prohibit-password.
Port	Specifies the port number listens on. The default is 22.
Subsystem	Configures an external subsystem (for example, *file transfer daemon*).
UseDNS	Specifies whether should resolve the client IP to a hostname.
UsePAM	Enables the *Pluggable Authentication Module* interface.
X11Forwarding	Specifies whether X11 forwarding is permitted. The argument must be *yes* or *no*. The default is *no*.

Table 9.2: *SSH client options and descriptions*

During the installation of the SSH server, it generates private and public keys, which are located inside of the directory **`/etc/ssh/`**. These keys use different cryptography algorithms. The name of the files have the format **`ssh_host_ALGORITHM_key`** and **`ssh_host_ALGORITHM_key.pub`**:

- ***ECDA:*** stands for **Elliptic Curve Digital Signature Algorithm**, which uses elliptic-curve cryptography.
- ***Ed255190:*** *EDSA* stands for **Edwards-curve Digital Signature Algorithm**, and *Ed25519* is a signature scheme using *SHA-512* and *Curve25519.*
- ***RSA:*** stands for **Rivest-Shamir-Adleman** is the widely used algorithm in SSH for private/public keys.

The global configuration in the system for the client is located in the file **`/etc/ssh/ssh_config`**. However, usually, the users define their own client

configuration in the file **`~/.ssh/config`**. The following popular options are available, as shown in *table 9.3*:

Option	Description
Host	Restricts the options to that host.
Match	Restricts the options to the parameter matching the pattern.
Ciphers	Specifies the ciphers allowed and their order of preference.
ForwardAgent	Specifies whether the connection to the authentication agent.
Hostname	Specifies the real host name to log into.
IdentityFile	Specifies a file from which the user's ECDSA, Ed25519, or RSA authentication identity is read.
Port	Specifies the port number to connect to the remote host.
ProxyCommand	Specifies the command to use to connect to the server.
ProxyJump	Specifies one or more jump proxies as either [user@]host[:port] or an ssh URI.
User	Specifies the user to log in as.

Table 9.3: *SSH options and descriptions*

An SSH server can be used as a **bastion** or **jump host** to connect to internal servers, which are not accessible externally. The parameters **`ProxyCommand`** and **`ProxyJump`** are used for that purpose. An example of the client configuration is shown as follows:

```
Host 192.168.122.43
  User agonzalez
  IdentityFile ~/.ssh/id_rsa
  Port 22

Match User cloud-user
  IdentityFile ~/.ssh/id_rsa_cloud_user
  Port 2222
Host *
  User admin
  IdentityFile ~/.ssh/id_rsa_admin
  Port 22
```

Communication using *SSH* requires a destination host, the port (which by default is 22), and the user's name to connect (if not defined, it will use the username running the command). If the SSH service allows the authentication with a password, it is possible to login specifying the user password if the SSH private/public key is not in use. This concept will be covered later in this chapter.

Connecting for the first time to a remote destination, the SSH client will request to the user if the fingerprint from the SSH server is the expected one and if the user wants to

continue with the connection. *Figure 9.17* shows an example of the message shown:

```
[root@rhel ~]# ssh  192.168.122.43
The authenticity of host '192.168.122.43 (192.168.122.43)' can't be established.
ED25519 key fingerprint is SHA256:y4Jc7BDqKxwzvVxfRsjS9/YKdJ23F/tneG9pQzlLYWA.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Figure 9.17: *Output example for command ssh*

If the answer to the prompt is *yes*, then the **fingerprint** will be added to the file **`~/.ssh/known_hosts`**, and in the following connections, it will be checked. If it is the same, there will be no questions. The content of the file **`known_hosts`** will contain a list of IP addresses and fingerprints; the following example shows the previous fingerprint relationship:

```
192.168.122.43 ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIC1YjZSS/Ek7pkGAQeC0I/kjHGzMJkQQxDNmEcTdrN5m
```

If the answer to the prompt is not yes, then the connection will be closed. The client *ssh* allows the option **`-v`** to increase the verbose level. It is possible to specify several times (maximum 3) to increase the debugging level. The following excerpts show an example of a connection where the public key was not accepted, and the server allows connections using a password:

```
[root@rhel ~]# ssh -v 192.168.122.43
OpenSSH_8.7p1, OpenSSL 3.0.1 14 Dec 2021
debug1: Reading configuration data /root/.ssh/config
debug1: /root/.ssh/config line 1: Applying options for 192.168.122.43
debug1: Reading configuration data /etc/ssh/ssh_config
(… omitted …)
debug1: Connecting to 192.168.122.43 [192.168.122.43] port 22.
debug1: Connection established.
debug1: identity file /root/.ssh/id_rsa type 0
debug1: identity file /root/.ssh/id_rsa-cert type -1
debug1: Local version string SSH-2.0-OpenSSH_8.7
debug1: Remote protocol version 2.0, remote software version
OpenSSH_8.9p1 Ubuntu-3
(… omitted …)
debug1: kex: algorithm: curve25519-sha256
debug1: kex: host key algorithm: ssh-ed25519
(… omitted …)
debug1: Server host key: ssh-ed25519
SHA256:y4Jc7BDqKxwzvVxfRsjS9/YKdJ23F/tneG9pQzlLYWA
(… omitted …)
debug1: Host '192.168.122.43' is known and matches the ED25519 host key.
debug1: Found key in /root/.ssh/known_hosts:1
```

```
(… omitted …)
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering public key: /root/.ssh/id_rsa RSA
SHA256:eVWmC9lMXVmqFkRoaYf/IvmicjW3nsJ2PxBI24qI+TE explicit
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: password
agonzalez@192.168.122.43's password:
```

If the authentication is correct, a *shell console* in the remote system will appear to introduce commands that will be executed in the remote node. It is possible to pass a command as an argument to the *ssh* command, so that it can be executed in the remote node, and log out can happen after it. *Figure 9.18* shows how to connect to a system and execute only one command instead open a session:

```
[root@rhel ~]# ssh 192.168.122.43 uname -a
agonzalez@192.168.122.43's password:
Linux ubuntu 5.15.0-47-generic #51-Ubuntu SMP Thu Aug 11 07:51:15 UTC 2022 x86_64
 x86_64 x86_64 GNU/Linux
```

Figure 9.18: *SSH connection and run a command*

It is possible to specify another SSH private key using the option **`-I`** and the file with the private key. Private keys need to private protect with permission *0600* or *0400*. If the permissions are *too open*, the following message will be shown, and the *key* will not be used:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0640 for '/root/.ssh/id_rsa' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "/root/.ssh/id_rsa": bad permissions
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0640 for '/root/.ssh/id_rsa' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "/root/.ssh/id_rsa": bad permissions
agonzalez@192.168.122.43's password:
```

The protocol *SSH* allows to forward remote and local ports that are useful to access internal services from outside. Option **`-L`** is used to map a local port to an IP and port from the remote network using the **`SSH tunnel`**. Option *-R* is less used, and its

functionality is to expose an IP and port from the local network to the remote server connected with *SSH*. *Figure 9.19* shows the traffic flow using an SSH port forwarding:

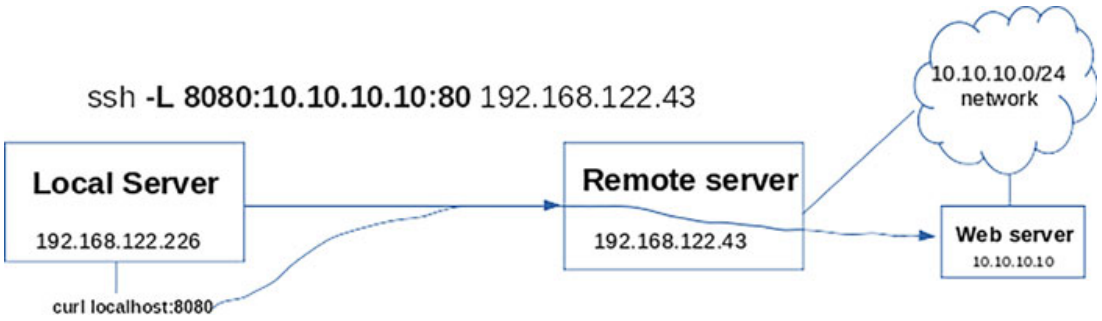


Figure 9.19: *SSH port forwarding diagram*

With SSH as well, it is possible to forward the graphical applications using the **X11 forwarding** mechanism. Using option *-X* in the *SSH client*, it is possible to execute applications in the remote server showing the graphical interface in the client node. Refer to *figure 9.20* for an example:

```
[agonzalez@server ~]$ ssh -X agonzalez@192.168.122.43
agonzalez@192.168.122.43's password:
agonzalez@ubuntu:~$ xeyes
```

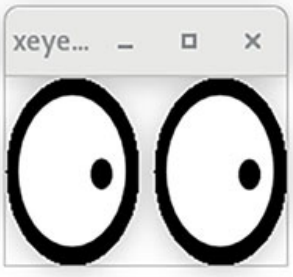


Figure 9.20: *SSH X11 forwarding example*

SSH public and private keys

The *public-key cryptography*, also known as *asymmetric cryptography*, is a system that uses a pair of keys: the *public key*, which is the key to be distributed to be allowed to access the systems, and the *private key,* which should not be shared and kept secure in the system initiating the connection. The steps in an **SSH connection** using the pair keys are as follows:

1. The client initiates the connection to the remote host.
2. The client sends the *public key* id to the *SSH server*.
3. The remote host checks in the `authorized_hosts` file of the user trying to connect if that key is accepted.

4. If the key is accepted, a message (*challenge string)* is encrypted with the *public key*.
5. The client decrypts the message received from the remote and sends it back to the remote host.
6. If the remote hosts ensure the message is correct, the connection is accepted.

The command to generate a pair of keys is called **`ssh-keygen`**. This command generates, by default, the files **`~/.ssh/id_rsa`** and **`~/.ssh/id_rsa.pub`**. It is possible to use the option *-t* to specify another type (*ecdsa, ed25519,* among others). With the option **`-f`**, it is possible to specify the path for the pair key to be generated. *Figure 9.21* shows the output of a key generation:

```
root@ubuntu:~# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:ZJy6Nsj0mOIGveR/gG0PZN5JDfESb7GXvJ6AQk+HVSY root@ubuntu
The key's randomart image is:
+---[RSA 3072]----+
|      o.E.o      |
|      .B B .     |
|     . +o@ +     |
|    .oo.O.. .    |
| . *oo+.S .      |
|. +o**o. o .     |
| +.o=+=   o      |
| .+. .o.         |
| .....           |
+----[SHA256]-----+
root@ubuntu:~#
```

Figure 9.21: *Output example for the command ssh-keygen*

The *OpenSSH* clients package includes a tool to help distribute the *public key* to the remote servers, and the command is **`ssh-copy-id`**. The argument is the remote host and, optionally, the username to connect. *Figure 9.22* shows an example:

```
root@ubuntu:~# ssh-copy-id agonzalez@192.168.122.226
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.
pub"
The authenticity of host '192.168.122.226 (192.168.122.226)' can't be established
.
ED25519 key fingerprint is SHA256:7OvP8RjZwnor69OaXBkxBxEEkH7rbP+s7oex875tK3w.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter o
ut any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompte
d now it is to install the new keys
agonzalez@192.168.122.226's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'agonzalez@192.168.122.226'"
and check to make sure that only the key(s) you wanted were added.
```

Figure 9.22: *Output example for the command ssh-copy-id*

After the key is distributed to the remote host, it is possible to connect to the remote node without being required to introduce the password. In the remote host, the file **`authorized_keys`** will append the *public key*, as shown in *figure 9.23*:

```
root@ubuntu:~# ssh agonzalez@192.168.122.226 "cat ~/.ssh/authorized_keys"
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCtxt9IKvvKCzShhSL+mqViCtf9eKQwAK4NLjMQXVqD
whY3fAnDCLC585DoHdHeqYKN5eabZAfN+JLoLEzcrwm2ilyxUTRAa93YK0rn5gPtElsGV6UixJWRtqY+
t8L1uvALRLI8XK76Ju281fHQgLiqLf0oWqYBpq/ao/dU19EdiqJLeU2DAOTgkBlE8hDfBpUsHpUx4cs+
TjQipwTKT7hKmQ9XsT7n+FpRHX4O4/bQ/klagKdlBioj/P3lHnpXsoQ0vw6FZCCAg14nQk9ebxBL23vz
eJivNquyiBXLIwmfNCRGze0DGzxMZxxRoIHrDoX4ks3HAGa93zOkHDO8kjQHUyOQJhEccUgi6bKS5xBl
mDoDR46fhSQO2DYZCs4nJKp8V3Q7kO3U+ygRiLWBykgPEqCz/Bn9S5iU08FQ3XGJO+FPV9+fw46MPxzA
9Wehisqll2aSbn6AxvxrwVts3COfSEHjHPsMC6sii6y0qYx5QN+Rea+9MxnFSpDM0= root@ubuntu
```

Figure 9.23: *Content example for authorized_keys file*

Check network services available

As described in *Chapter 7, Network Configuration*, it is possible to use the command *ss* to list the different services running in the system where the application is executed. The file **`/etc/services`** contain a list of the popular services and the ports, with the protocol (tcp or udp) used. For example, in the following excerpt, it shows the definition for the services *SSH*, *DNS* (*domain*), and *DHCP* (*bootps*/*bootpc*):

```
ssh         22/tcp   # SSH Remote Login Protocol
domain   53/tcp   # Domain Name Server
domain   53/udp
bootps   67/udp
bootpc   68/udp
```

To detect the network services available in one host or in the network, the utility *nmap* is the most popular for Linux systems. This utility is used to scan the ports available in

one system or a subnet. Specifying an IP, it will scan for the known ports searching for the port opens. *Figure 9.24* shows a simple scan output:

```
[root@rhel ~]# nmap 192.168.122.43
Starting Nmap 7.91 ( https://nmap.org ) at 2022-09-04 21:20 WEST
Nmap scan report for ubuntu (192.168.122.43)
Host is up (0.000028s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
53/tcp open  domain
MAC Address: 52:54:00:3D:10:25 (QEMU virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.14 seconds
```

Figure 9.24: *Output example for the command nmap*

It is possible to scan for a subnet, indicating as an argument the subnet and the mask. For example, **`nmap 192.168.122.0/24`**. The command **`nmap`** includes several options for advanced scannings, such as trying all the ports or obtaining information for each service. In *figure 9.25*, the option **`-sV`** is used to obtain the service versions:

```
[root@rhel ~]# nmap -sV --allports 192.168.122.43
Starting Nmap 7.91 ( https://nmap.org ) at 2022-09-04 21:28 WEST
Nmap scan report for ubuntu (192.168.122.43)
Host is up (0.000030s latency).
Not shown: 997 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3 (Ubuntu Linux; protocol 2.0)
53/tcp open  domain  dnsmasq 2.86
80/tcp open  http    Apache httpd 2.4.52 ((Ubuntu))
MAC Address: 52:54:00:3D:10:25 (QEMU virtual NIC)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.
org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.30 seconds
```

Figure 9.25: *Output example for the command nmap*

Other popular network services

Linux distributions allow to offer a big number of different services. Some of the most popular ones are described in *table 9.4*, with the popular package names associated with them:

Network Service	Description	Popular software
NTP	*Network Time Protocol* software can act as a client to keep the system time synchronized and as a service to keep the clients with the time synchronized.	• ntpd • chrony
IMAP	*Internet Message Access Protocol* is used to retrieve	• Dovecot

	email messages from a mail server.	• Cyrus
SMTP	*Simple Mail Transfer Protocol* is used for mail transmission.	• Exim • sendmail • Postfix
SNMP	*Simple Network Management Protocol* is used for collecting and organizing information about managed devices.	• snmpd
Kerberos	An authentication protocol to authenticate services.	• krb5-server
LDAP	*Lightweight Directory Access Protocol* is used to organize data about organizations, individuals, and other resources.	• openldap
Syslog	This protocol is used for some software to send the logs to a central server.	• rsyslog
RFB (vnc)	*Remote Framebuffer.* Used for graphical remote administration.	• tightvnc • realvnc
RDP	*Remote Desktop*. A desktop protocol for remote control.	• Xrdp server

Table 9.4: *Other network services and Linux popular software*

Other protocols related to file sharing are going to be covered in the upcoming chapter, including **Network File System** (**NFS**), **Server Message Block** (**SMB**), **File Transfer Protocol** (**FTP**), and **Trivial File Transfer Protocol** (**TFTP**).

Conclusion

In this chapter, we understood how the core network services are working, and the software associated needed for the daily tasks of system administrators. This chapter covered three of the most important services: *DHCP*, which is required for the servers when they booting*; DNS*, which is required for servers and applications; and *SSH*, which is required for the system administrators to administrate remote systems.

Knowing where the configurations are located, and the different options available will allow the administrator to protect the system and provide high-quality service. This chapter also described other popular services in Linux and the associated software.

Key facts

- Network services are a key part on Linux.
- A DHCP server offers IP addresses to the network.
- Bind 9 is one of the most popular DNS server solutions.
- Public/Private key is the most secure authentication for SSH.

Questions

1. How many steps are involved to obtain an *IP address* from a *DHCP server*?
    a. 2
    b. 6
    c. 4
2. Which two ports are used during the DHCP request?
    a. 66
    b. 67
    c. 68
3. What is the file name for the Bind 9 configuration?
    a. named.conf
    b. bind.conf
    c. bind9.conf
4. What argument is used in the command *dig* to obtain a full query information?
    a. +debug
    b. +trace
    c. +verbose
5. What command is used to copy the public key to the remote host?
    a. ssh-copy-key
    b. ssh-keygen
    c. ssh-copy-id

Answers

1. c
2. b and c
3. a
4. b
5. c

CHAPTER 10

File Sharing

Introduction

It is recommended to store files in a central location and share files between users and servers in a normal task inside enterprises. For example, a server can store some generated files in a central shared location, and a user can access a service, to access the file. Another use case is to have centralized storage for services, such as an image registry for containers.

Nowadays, with cloud services providing file sharing, such as Dropbox or Google Drive, the use of file sharing outside companies is not so popular. Companies that want to keep the files secure inside their own infrastructure and want to control access to the files will have to install and configure a file-sharing solution. Another popular use case at an enterprise level is to use file sharing for storing backups or providing dynamic storage for microservices.

The first part of this chapter will cover the most popular file sharing in Linux called **Network File System** (**NFS**). NFS is a popular distributed file system protocol to share files not only to end users but between systems or consumed by different services. This chapter covers how to install and configure the directories to be shared and the client options to access them.

The second part of the chapter covers the popular sharing protocol called **Server Message Block** (**SMB**), used in Windows and Linux. This chapter will cover the suite of programs called **Samba**, including the installation, configuration, and client tools to access the shared directories.

The last part of the chapter will cover a previously popular protocol called **File Transfer Protocol** (**FTP**), covering the installation of popular solutions for the service and the clients. The **Trivial File Transfer Protocol** (**TFTP**) and its use cases (that is, where it is used) will be described.

Structure

In this chapter, we will discuss the following topics:

- NFS service and client

- SMB introduction
- Samba service and client
- FTP server and client
- TFTP introduction

NFS service and client

Network File System (**NFS**) is a distributed file system protocol developed in 1984 by **Sun Microsystems**. The modern version of the protocol is version 4, referenced as **NFSv4**, replacing the previous version 2 (**NFSv2**) and version 3 (**NFSv3**). NFS is the most popular protocol for sharing data with clients in a Linux system. Other storage solutions, such as **Network Attached Storage** and **Storage systems**, allow the sharing of available storage using the NFS protocol. The NFS server can be seen in *figure 10.1*:

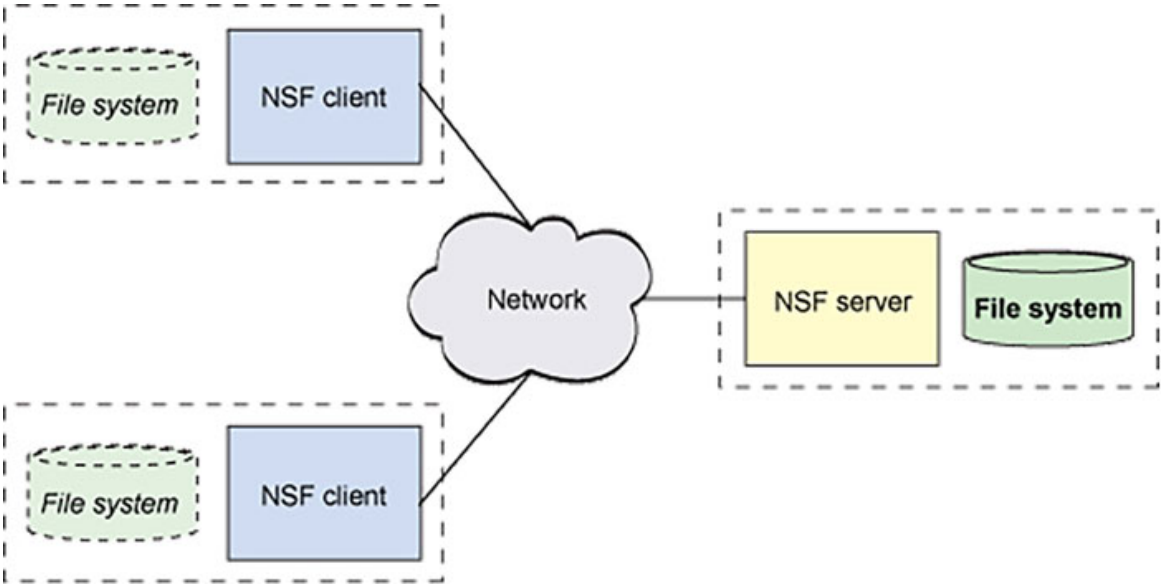


Figure 10.1: *NFS server/client overview. Source: IBM*

The NFS implementation in Linux includes several services required to provide authentication and access to the shared data to the clients. *Table 10.1* shows the required services and the ports used:

Service	Description
nfsd	The *NFS* server, which services the requests from the clients. It uses the port *2049/udp* or *2049/tcp*.
rpcbind	The server that covers *Remote Procedure Call (RPC)* program numbers into universal addresses. Clients connect to *rpcbind* to determine the

	address where the requests should be sent. This service uses *111/udp* or *111/tcp*. This is not required on *NFSv4*.
rpc.mountd	Used for version 2 and version 3 to implement the directory mounting from a client. Version 4 uses only *nfsd*.
rpc.nfsd	Provides an ancillary service needed to satisfy the *NFS* requests by the clients.
lockd	It implements the *Network Lock Manager* (*NLM*) protocol, used by *NFSv3,* to lock files in the server.
rpc.statd	It implements the *Network Status Monitor* (*NSM*) protocol, which notifies the clients when the server is restarted.
rpc.idmapd	It provides to *NFSv4* server and client a map between names, local user ids, and group ids.

Table 10.1: *NFS services and description*

The modern **NFSv4** is updated regularly with new versions and new features. For example, version 4.1 includes **Parallel NFS** (**pNFS**), and version 4.2 includes *server-side clone and copy* features. It is important to take into consideration that some clients only accept specific versions (3,4) or minor versions (4.0, 4.1) of the NFS protocol.

NFS server

The package on the **Advanced Packet Tool** (**APT**) repositories, including the NFS server, is called **`nfs-server`**. On systems using RPM Package Manager, the server and the client is included inside the package named **`nfs-utils`**. In both systems, the main service on **`systemd`**, after installation, is named **`nfs-server`**.

The main file to configure the service is **`/etc/nfs.conf`**, where it is possible to configure different values such as the IP protocol to be used (*tcp* and/or *udp*) or the NFS versions to be enabled (2, 3, 4, 4.0, 4.1 and 4.2). The following excerpt shows the parameters specific for the *nfsd* service, which by default are commented as follows:

```
[nfsd]
# debug=0
# threads=8
# host=
# port=0
# grace-time=90
# lease-time=90
# udp=n
# tcp=y
```

```
# vers2=n
# vers3=y
# vers4=y
# vers4.0=y
# vers4.1=y
# vers4.2=y
# rdma=n
# rdma-port=20049
```

By default, the NFS service will be listening on **2049/tcp**, and it will support versions 3, 4, and all the subversions. The file **/lib/systemd/system/nfs-server.service** defines the services related to the *NFS* service as shown in the following code block:

```
[Unit]
Description=NFS server and services
DefaultDependencies=no
Requires=network.target proc-fs-nfsd.mount
Requires=nfs-mountd.service
Wants=rpcbind.socket network-online.target
Wants=rpc-statd.service nfs-idmapd.service
Wants=rpc-statd-notify.service
Wants=nfsdcld.service
After=network-online.target local-fs.target
After=proc-fs-nfsd.mount rpcbind.socket nfs-mountd.service
After=nfs-idmapd.service rpc-statd.service
After=nfsdcld.service
Before=rpc-statd-notify.service
# GSS services dependencies and ordering
Wants=auth-rpcgss-module.service
After=rpc-gssd.service gssproxy.service rpc-svcgssd.service
[Service]
Type=oneshot
RemainAfterExit=yes
ExecStartPre=-/usr/sbin/exportfs -r
ExecStart=/usr/sbin/rpc.nfsd
ExecStop=/usr/sbin/rpc.nfsd 0
ExecStopPost=/usr/sbin/exportfs -au
ExecStopPost=/usr/sbin/exportfs -f
ExecReload=-/usr/sbin/exportfs -r
[Install]
```

```
WantedBy=multi-user.target
```

To disable previous versions, it needed to edit the file **`/etc/nfs.conf`** and specify the value **`no`** to the parameter **`vers3`**. As **`NFSv4`** does not need the services related to **`rpcbind`**, the following command will disable them and avoid being executed when the system is booting:

```
systemctl mask --now rpc-statd.service rpcbind.service
rpcbind.socket
```

The service **`nfsd`** users the file **`/etc/exports`** to define the directories to be exported using the NFS protocol. Each directory in this file has an associated list of options and an access control list. A directory can be exported to the following:

- A single host: specifies a fully qualified domain name and an IPv4 or IPv6 address.
- A network: this will export to all the hosts inside of the subnet indicated, and the format is *address/netmask*. The netmask can be specified with the dotted-decimal format, for example, *255.255.255.0*, or mask length, for example, */24*.
- A pattern using a wildcard: it is possible to use an asterisk (*) or interrogation (?) to define a pattern. For example, **.example.com*.
- A **Network Information Service** (**NIS**) *netgroup*: Used with format *@netgroup*.

A host that can be part of several definitions will have precedence to the first match definition. Some of the popular export options available are described in *table 10.2*:

Option	Description
ro	The shared directory will be exported as *read-only*.
rw	The shared directory will be exported, allowing you to write it.
async	Allows to not write to disk directly, causing better performance, but an unclean server restart will cause data loss.
sync	Only reply to the client requests ensuring the data is written.
fsid	Identify the directory to be exported with an id.
no_subtree_check	Do not check if a subdirectory is in the same filesystem as the directory exported. This is the default in newer versions.
subtree_check	Ensure the subdirectory is in the same filesystem as the directory exported.

Table 10.2: *NFS export options*

The NFS server, by default, does not allow the *root* user to access with the *uid 0*. Instead, it will be assigned a different id, called *nobody uid*. Regular users would access their files working locally, but in some situations, this is not possible because the user does not exist in the remote server. The different mapping options are described in *table 10.3*:

Mapping option	Description
root_squash	Map requests from uid/gid 0 to the anonymous uid/gid. This is the default option.
no_root_squash	Turn off root squashing.
all_squash	Map all *uids* and *gids* to the anonymous user.
no_all_squash	The opposite to *all_squash* option. This is the default option.
anonuid and anongid	These options explicitly set the *uid* and *gid* of the anonymous account. By default, the value is *65534*.

Table 10.3: *NFS mapping options*

The following block shows an example of an **`/etc/exports`** content to export a directory to the internal subnet *192.168.122.0/24* with the option read-write (*rw*), and to the subnet 10.10.0.0/16 with the option read-only *(ro*).

```
/storage 192.168.122.0/24(rw,sync,no_subtree_check) 10.10.0.0/16
(ro,no_subtree_check)
```

After the file **`/etc/exports`** is updated, the command **`exportfs`** has to be executed to reload the content and share the new directories, update the shared directories or stop sharing a directory previously shared. Option **`-a`** is used to export or un-export all directories; option **`-r`** is for re-exporting all directories, and option **`-u`** is used to un-export one or more directories. Option **`-v`** shows the actions performed, as is shown in *figure 10.2*:

```
root@ubuntu:~# exportfs -r -v
exporting 192.168.122.0/24:/storage
exporting 10.10.0.0/16:/storage
```

Figure 10.2: *Output example command exportfs*

The biggest differences between the prior versions and version 4 are the following:

- Prior versions were stateless; in *NFSv4,* the information about the objects is maintained by the server.

- Version 4 improved the security supporting *Kerberos 5* authentication.
- Version 4 has better compatibility with network firewalls.
- Version 4 includes pseudo filesystem support.
- Lock operations are part of the protocol and not a separate service (**`lockd`**).

This chapter does not cover authentication with *Kerberos 5* or virtual filesystem.

NFS client

There is not a command considered as an NFS client, but the command **`showmount`** is used to query an NFS server, and the command **`mount`** is used to mount a shared directory from an NFS server. The command **`showmount`** shows information from a server, and the option *-e* (*--exports*) shows the list of exported directories and where it is exported (*IPs or subnets)*. This command requires to have the version NFSv3 enabled and the firewall allowing **`rpcbind`** connections. *Figure 10.3* shows the output example:

```
[root@rhel ~]# showmount -e 192.168.122.43
Export list for 192.168.122.43:
/storage 10.10.0.0/16,192.168.122.0/24
```

Figure 10.3: *Output example for the command showmount.*

For version 4, NFSv4, it is possible to mount the *root directory* (*/*) to list the exported directories from the server. *Figure 10.4* shows an example:

```
[root@rhel ~]# mount -t nfs4 192.168.122.43:/ /mnt
[root@rhel ~]# ls /mnt
storage
```

Figure 10.4: *Example usage command mount with NFS*

The client can use the command **`mount`** with the following syntax to manually mount a shared directory to the local system:

`mount -t nfs|nfs4 -o options` remoteserver`:/`shared `/exported`

The popular options that can be specified when mounting an NFS share are the following:

- *vers=n* (or *nfsvers=*): Specifies the NFS protocol version to be used. Modern systems are using, by default, version *4.2*.

- *soft/hard*: Indicates the recovery behavior if the client loses communication with the server. If *hard* is chosen, the connection will be retried indefinitely, and this is the default option. If the option *soft* is specified, the connection is not retried.
- *rsize/wsize*: The size for the read request or the write request.
- *proto*: It specified the protocol transport to use *tcp* or *udp*. On version 4, it is recommended to use *tcp*.
- *port*: The port to be used, by default, is 2049.

Figure 10.5 shows an example of how to mount the exported directory previously shown and how the output of the command **`mount`** without any argument shows the default options:

```
[root@rhel ~]# mount -t nfs4 192.168.122.43:/storage /exported
[root@rhel ~]# mount | grep /exported
192.168.122.43:/storage on /exported type nfs4 (rw,relatime,vers=4.2,rsize=524288
,wsize=524288,namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=19
2.168.122.226,local_lock=none,addr=192.168.122.43)
```

Figure 10.5: *Output example command mount*

By default, the directories are exported with the option **`root_squash`**, and this option does not allow *root* users in the client to write in the mounted directory. A regular user can always write in the directory when the *user id* or one of his *group ids* is allowed to write. *Figure 10.6* illustrates the creation of a new file on a shared directory:

```
[agonzalez@rhel exported]$ groups
nfsusers wheel agonzalez
[agonzalez@rhel exported]$ ls -ld /exported/
drwxrwx---. 2 root nfsusers 4096 Sep 10 21:17 /exported/
[agonzalez@rhel exported]$ touch newfile
[agonzalez@rhel exported]$ ls -l
total 0
-rw-r--r--. 1 agonzalez nfsusers 0 Sep 10 21:17 newfile
```

Figure 10.6: *Creating a new file on a shared directory*

The syntax for the file **`/etc/fstab`** is the following for an NFS share:

```
server:/directory /mntpoint        nfs4    rw      0       0
```

SMB introduction

The protocol **Server Message Block** (**SMB**) was originally developed in 1983 and provided shared access to files and printers in the network. Microsoft published version 1.0 of the protocol with some modifications, with the name **Common Internet File System** (**CIFS**), in 1996. This proposal was never accepted, and Microsoft discounted the use of CIFS and continued developing new versions of SMB. The protocol SMB uses port 445 on protocol TCP and port 139, as well TCP, for communication over NetBIOS.

Similar to NFS, the SMB protocol has different protocol versions, which were introduced to different versions of Windows. Version 2.0 was introduced in the year 2006 on Windows Vista and Windows server 2008, publishing the specifications allowing interoperate with other systems. Version 2.1 was introduced in Windows 7 and Windows 2008 R2. Version 3.0 of the protocol was introduced with Windows 8 and Windows Server 2012. New versions are 3.0.2 and 3.11, included in the latest versions of the Windows and Windows Server.

Samba server and client

The specifications for the protocol SMB are proprietary and were closed in the beginning. With version 2.0, the protocol was made available, allowing developers to create software to interoperate with the protocol and the Windows systems.

In the year 1991, a project called **Samba** started to implement a solution for SMB/CIFS for Unix systems, which, years later, came to be a popular project used to create file-sharing servers on Linux. Samba project offers client tools to access servers, either to access files or printers. Version 3, released in the year 2003, was able to join **Active Directory** as a member. New versions of Samba (4.x) support the latest SMB protocols and includes the modern features offered by the protocol. Some of these features are as follows:

- NetBIOS over TCP/IP
- WINS server (also known as NetBIOS name server)
- Security account manager
- Local security authority
- NTLM (NT Lan manager)

Figure 10.7 shows a diagram of Samba architecture:

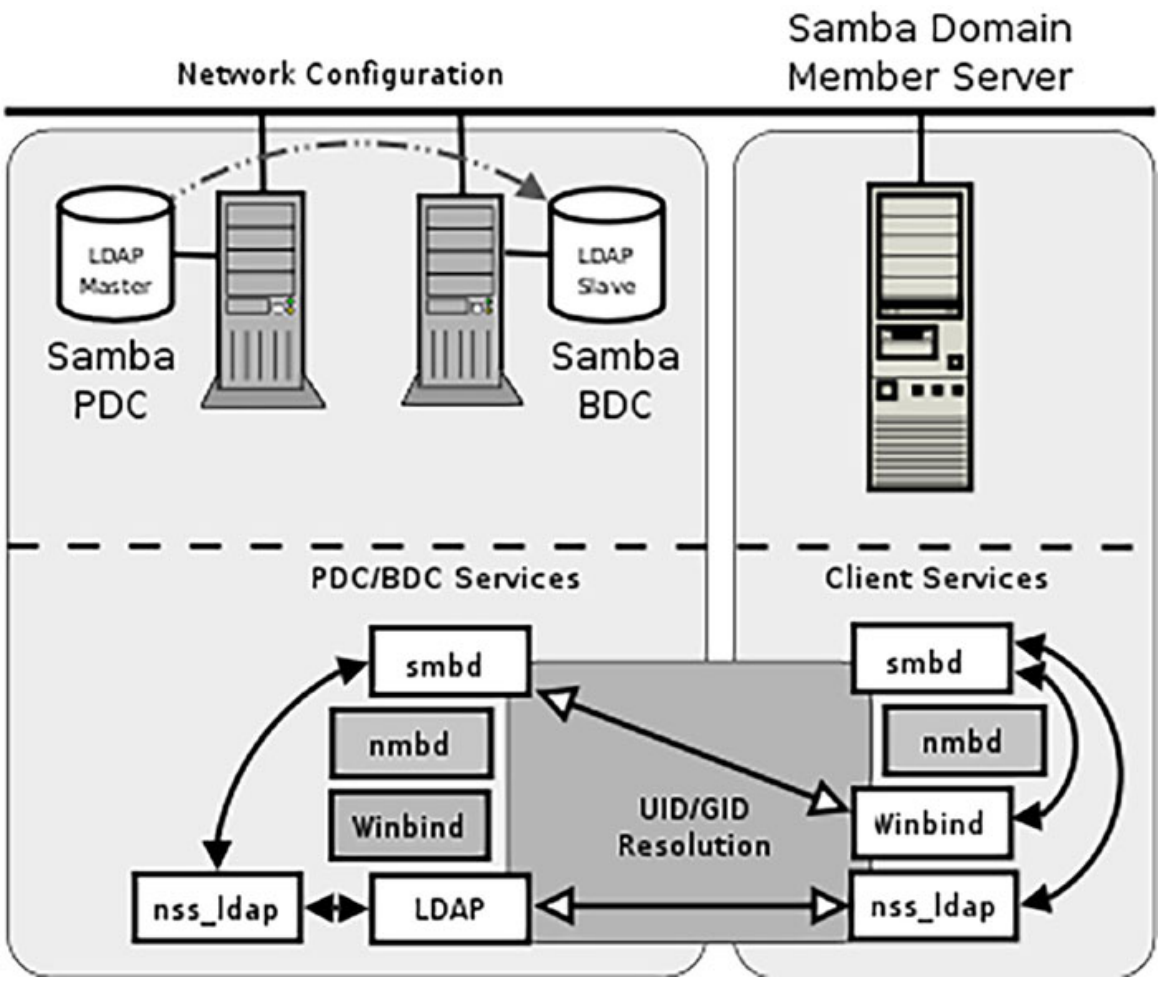


Figure 10.7: *Samba architecture diagram. Source: Samba*

Samba Server

The **Samba Server**, which is named **`smbd`**, provides file-sharing and printing services to Windows clients. This server is included in the package **`samba`**; with the same name for APT and RPM repositories. The main file to configure the service is **`/etc/samba/smb.conf`**, which includes global configuration parameters and the definition of the shares (directories and printers). The default configuration in an Ubuntu system is shown as follows:

```
[global]
   workgroup = WORKGROUP
   server string = %h server (Samba, Ubuntu)
   log file = /var/log/samba/log.%m
   max log size = 1000
   logging = file
   panic action = /usr/share/samba/panic-action %d
```

```
   server role = standalone server
   obey pam restrictions = yes
   unix password sync = yes
   passwd program = /usr/bin/passwd %u
   passwd chat = *Enter\snew\s*\spassword:* %n\n
   *Retype\snew\s*\spassword:* %n\n
   *password\supdated\ssuccessfully* .
   pam password change = yes
   map to guest = bad user
   usershare allow guests = yes
[printers]
   comment = All Printers
   browseable = no
   path = /var/spool/samba
   printable = yes
   guest ok = no
   read only = yes
   create mask = 0700
[print$]
   comment = Printer Drivers
   path = /var/lib/samba/printers
   browseable = yes
   read only = yes
   guest ok = no
```

In the global configuration, the workgroup name is defined as *WORKGROUP*, and it defines the directory **`/var/log/samba/`** for logs, among other options related to users. Two default shares, **`printers`** and **`print$`**, are configured but not accessible by guest users. To define a new share, a new section should be added to the file **`smb.conf`**, for example, to share the directory **`/storage`** to be writable by users and **`/public`** as read-only:

```
[storage]
   comment = storage directory
   path = /storage
   browseable = Yes
   read only = No
[public]
   comment = Public directory
   path = /public
   browseable = Yes
```

```
read only = Yes
```

After modifying the file **smb.conf**, it is possible to use the command **testparm** to ensure the syntax and the options are valid. If the file content is correct, it will show the content on the screen. However, if it finds any issue, it will report the location, as is shown in *figure 10.8*:

```
root@ubuntu:~# testparm -v  /etc/samba/smb.conf
Load smb config files from /etc/samba/smb.conf
Error loading services.
```

Figure 10.8: *Output example for command testparm*

The suite Samba also provides a command named **smbcontrol**, which sends messages to **smbd**. This command is usually used to reload the configuration with the argument **reload-config**, or to stop the service with the argument **shutdown**. For example, the command **smbcontrol smbd reload-config** will send a message to the service **smbd** to reload the configuration file content.

Samba offers the command **smbpasswd** to manage the users and the passwords to access the service. The users are going to be mapped to the local users, meaning that the user id and group ids assigned to a local user will be used to access the directory requested by the client. The command **smbpasswd** has the option **-a** to add a new user to the Samba users allowed to access to the system, as shown in *figure 10.9*:

```
root@ubuntu:~# smbpasswd -a agonzalez
New SMB password:
Retype new SMB password:
```

Figure 10.9: *Output example for the command smbpasswd*

The users are stored in a private database in the directory **/var/lib/samba/private/**. Samba provides the command **pdbedit**, which, with the option **-L** is used to list the users in the database and the option **-v** to obtain a detailed information about the users. *Figure 10.10* shows both options used:

```
root@ubuntu:~# pdbedit -L
agonzalez:1000:agonzalez
root@ubuntu:~# pdbedit -L -v
---------------
Unix username:        agonzalez
NT username:
Account Flags:        [U          ]
User SID:             S-1-5-21-1583416681-3049946014-381383256-1001
Primary Group SID:    S-1-5-21-1583416681-3049946014-381383256-513
Full Name:            agonzalez
Home Directory:       \\UBUNTU\agonzalez
HomeDir Drive:
Logon Script:
Profile Path:         \\UBUNTU\agonzalez\profile
Domain:               UBUNTU
Account desc:
Workstations:
Munged dial:
Logon time:           0
Logoff time:          Wed, 06 Feb 2036 15:06:39 UTC
Kickoff time:         Wed, 06 Feb 2036 15:06:39 UTC
Password last set:    Sat, 10 Sep 2022 21:53:01 UTC
Password can change:  Sat, 10 Sep 2022 21:53:01 UTC
Password must change: never
Last bad password   : 0
Bad password count  : 0
Logon hours         : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

Figure 10.10: *Output example for the command pdbedit*

Samba server can be configured to use different authentication sources, such as **Lightweight Directory Access Protocol** (**LDAP**). Samba can also act as an **Active Directory Domain Controller**, providing DNS services and **Kerberos** authentication. This book only covers Samba as file sharing.

Samba client

The Samba software suite includes different commands to act as a client for the protocol SMB. The most used command is **`smbclient`**, which is part of the package **`samba-client`** in RPM software repositories and the package **`smbclient`** on DEB software repositories. This command is used to access the connect to an SMB Service to access the resources. The command works in interactive mode unless the option **`-L`** (*--list*) is used. F*igure 10.11* and *10.12* shows the syntax and the different usages:

- List the shared resources from the server 192.168.122.43, as shown in *figure 10.11*:

```
[root@rhel ~]# smbclient -L  //192.168.122.43/ -U agonzalez
Password for [SAMBA\agonzalez]:

        Sharename       Type      Comment
        ---------       ----      -------
        print$          Disk      Printer Drivers
        storage         Disk      Storage directory
        public          Disk      Public directory
        IPC$            IPC       IPC Service (ubuntu server (Samba, Ubuntu)
SMB1 disabled -- no workgroup available
```

Figure 10.11: *Example of usage of the command smbclient*

- Access interactive to the Samba server and list the content of the *storage* directory, as shown in *figure 10.12*:

```
[root@rhel ~]# smbclient //192.168.122.43/storage/ -U agonzalez%Start123
Try "help" to get a list of possible commands.
smb: \> dir
  .                                   D        0  Sat Sep 10 21:17:56 202
  ..                                  D        0  Sat Sep 10 22:43:09 202
  newfile                             N        0  Sat Sep 10 21:17:56 202

                11758760 blocks of size 1024. 5635424 blocks available
```

Figure 10.12: *Example of the usage of command smbclient*

It is possible to mount a remote *SMB* resource in a Linux system using the command **`mount`**. The command **`mount`**, when the type is **`cifs`** (or **`smb3`**), will redirect the request to the command **`mount.cifs`**. The command **`mount.cifs`** is part of the package named **`cifs-utils`**. Some of the popular options available are the following:

- **username (or user)**: Specifies the username to connect to the server.
- **password (or pass)**: Specifies the password for the username.
- **domain (or dom or workgroup)**: Specifies the domain (workgroup) of the user
- **credentials (or cred)**: Specifies a file that contains the username and the password to authenticate to the server. It is also possible to specify the domain.
- **ro**: Mount as read-only.
- **rw**: Mount as read-write.
- **sec**: Security mode used; some options are *krb5, krb5i, ntlm*, and *ntlmv2*.

- **vers**: Specify the version of the protocol.

Figure 10.13 shows an example of mounting the previously shared resource, specifying the username and the password as options. The output of the command **`mount`** is shown to list the default options:

```
[root@rhel ~]# mount -t cifs -o rw,username=agonzalez,password=Start123 \
> //192.168.122.43/storage/ /exported
[root@rhel ~]# mount |grep "/exported"
//192.168.122.43/storage/ on /exported type cifs (rw,relatime,vers=3.1.1,cache=st
rict,username=agonzalez,uid=0,noforceuid,gid=0,noforcegid,addr=192.168.122.43,fil
e_mode=0755,dir_mode=0755,soft,nounix,serverino,mapposix,rsize=4194304,wsize=4194
304,bsize=1048576,echo_interval=60,actimeo=1)
```

Figure 10.13: *Example usage of the command mount*

It is recommended to use the option **`credentials`** (or **`cred`**) to specify a protected file containing the username and the password. An example of the file is as follows:

```
username=agonzalez
password=Start123
```

FTP server and client

The **File Transfer Protocol** (**FTP***)* is a historical protocol published for the first time in 1971. It is used to share files to end users mainly. Nowadays, the popularity of FTP is low, and it is mainly used to distribute software publicly. The protocol FTP separates the control (login and commands) and the data (transfer files) connections. The protocol uses port 21/tcp for the control communication and port *20/tcp* for the data transmission. It is possible to protect the protocol using SSL/TLS, which is commonly named FTPS, which uses 989/tcp, and 990/tcp ports.

The FTP protocol can work in two different modes:

- Active mode: The client connects to port 21/tcp and indicates which local port is used for the data transfer. The servers send the data to the port specified. This can be an issue when a firewall is involved because that communication usually is not allowed.
- Passive mode: This mode is used to avoid firewall issues. The client sends a signal called *PASV,* and the server answers with an IP and a port to start the data communication.

Figure 10.14 shows the differences between the two modes:

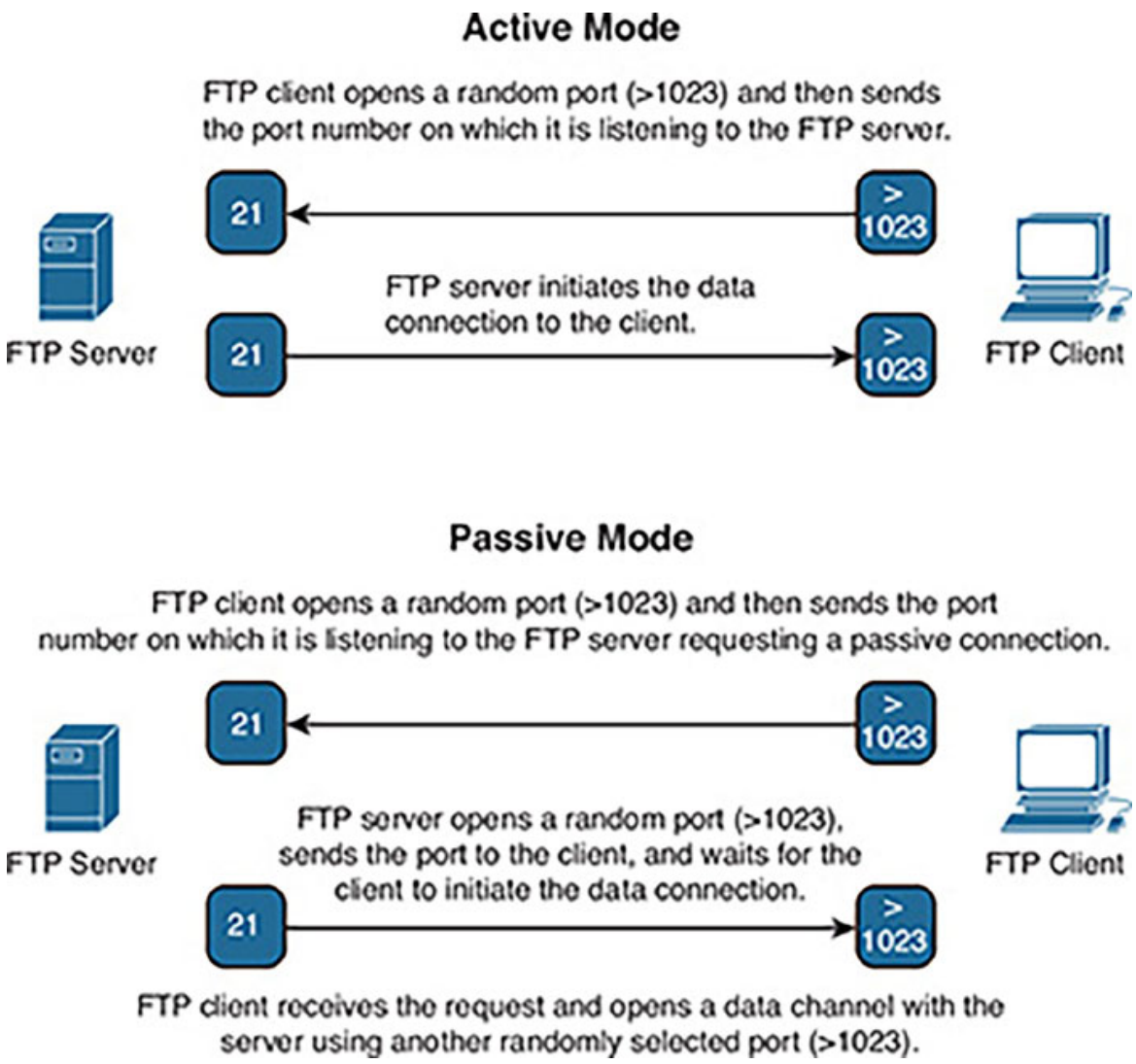


Figure 10.14: *Active and passive mode for FTP. Source: Cisco*

The most popular software to offer an FTP service is called **vsftpd**. The name stands for **very secure FTP daemon**. The main configuration file is **`/etc/vsftpd.conf`**, and the default options in Ubuntu servers are shown as follows:

```
listen=NO
listen_ipv6=YES
anonymous_enable=NO
local_enable=YES
dirmessage_enable=YES
use_localtime=YES
xferlog_enable=YES
connect_from_port_20=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
```

The option *listen* indicates that if the service will be executed in standalone mode (*YES*) or using *systemd* (*NO*). The option **`local_enable`** indicates if the local users can log in, and the option **`anonymous_enable`** specifies if guest users can connect to the service.

One of the features of the FTP servers is that users, when connecting to the service can access the files in their home directory. They can download, create directories or upload files to the server.

The most popular client in a Linux system to access the FTP service is the software **lftp**. This client allows to perform the normal tasks for FTP but also allows access to other protocols such as **Hypertext Transfer Protocol** (**HTTP**), **Hypertext Transfer Protocol Secure** (**HTTPS**), **File transferred over Shell protocol** (**FISH**), and **Secure File Transfer Protocol** (**SFTP**). It also supports the BitTorrent protocol. The argument required is to specify the destination; if the destination is an IP address, it will, by default, use FTP as a protocol. For other protocols, it is required to specify the method, such as *ftps://* or *http://*. The options for authentication are **`-u`** to specify the username and optionally the username and the password separated by a comma *(,)*. *Figure 10.15* shows an example:

```
[root@rhel ~]# lftp 192.168.122.43 -u agonzalez,Start123
lftp agonzalez@192.168.122.43:~> ls
-rw-rw-r--    1 1000     1000            9 Sep 11 17:08 notes.txt
lftp agonzalez@192.168.122.43:~>
```

Figure 10.15: *Example usage of the command lftp*

The command **`lftp`**, as observed, uses an interactive mode to operate with the server. Some of the popular options are described in *table 10.4*:

Command	Description
ls [args]	List the content of the current directory in the server, or is it possible to specify the directory to list.
cd	Change the directory in the server.
lcd	Change the directory in the local client.
get	Download a file from the *FTP server*.

mget	Download multiple files.
put	Upload a file to the *FTP server*.
mput	Upload multiple files.
mkdir	Create a directory in the server.
rmdir	Remove a directory from the server.

Table 10.4: *LFTP commands*

The command **`lftp`** accepts the option **`-e`** to execute a command or combination of the commands, which can include the command **`quit`** to exit the interactive mode. Refer to *figure 10.16* for an example:

```
[root@rhel ~]# lftp 192.168.122.43 -u agonzalez,Start123 -e "get notes.txt;quit"
9 bytes transferred
[root@rhel ~]# ls -l notes.txt
-rw-r--r--. 1 root root 9 Sep 11 18:08 notes.txt
```

Figure 10.16: *Example of the usage of the command lftp*

It is possible to use option **`-d`** to **`debug`** the communication to the FTP server; the following block shows the output communication to a **`vsFTPd`** server:

```
---- Resolving host address…
---- IPv6 is not supported or configured
---- 1 address found: 192.168.122.43
---- Connecting to 192.168.122.43 (192.168.122.43) port 21
<--- 220 (vsFTPd 3.0.5)
---> FEAT
<--- 211-Features:
<---  EPRT
<---  EPSV
<---  MDTM
<---  PASV
<---  REST STREAM
<---  SIZE
<---  TVFS
<--- 211 End
---> AUTH TLS
<--- 530 Please login with USER and PASS.
---> USER agonzalez
<--- 331 Please specify the password.
---> PASS Start123
<--- 230 Login successful.
```

```
---> PWD
<--- 257 "/home/agonzalez" is the current directory
---> PASV
<--- 227 Entering Passive Mode (192,168,122,43,35,39).
---- Connecting data socket to (192.168.122.43) port 8999
---- Data connection established
---> LIST
<--- 150 Here comes the directory listing.
---- Got EOF on data connection
---- Closing data socket
-rw-rw-r--    1 1000     1000            9 Sep 11 17:08 notes.txt
<--- 226 Directory send OK.
---> QUIT
<--- 221 Goodbye.
---- Closing control socket
```

TFTP introduction

This simple protocol, **Trivial File Transfer Protocol** (**TFTP**), is usually used to provide files to a remote node when the system is booting. This protocol is used to perform automatic installation in **baremetal** nodes mainly. It does not have a login or access control mechanism, and its normal use is in a private network. The protocol used is **User Datagram Protocol** (**UDP**), and the port is 69.

This protocol is usually used with the **Preboot Execution Environment** (**PXE**), which allows systems to boot from the network and load boot media. This boot media can be stored in a TFTP server. Modern systems can use iPXE, which allows using an HTTP server instead of a TFTP service to store the boot media. *Figure 10.17* illustrates a PXE/TFTP overview:

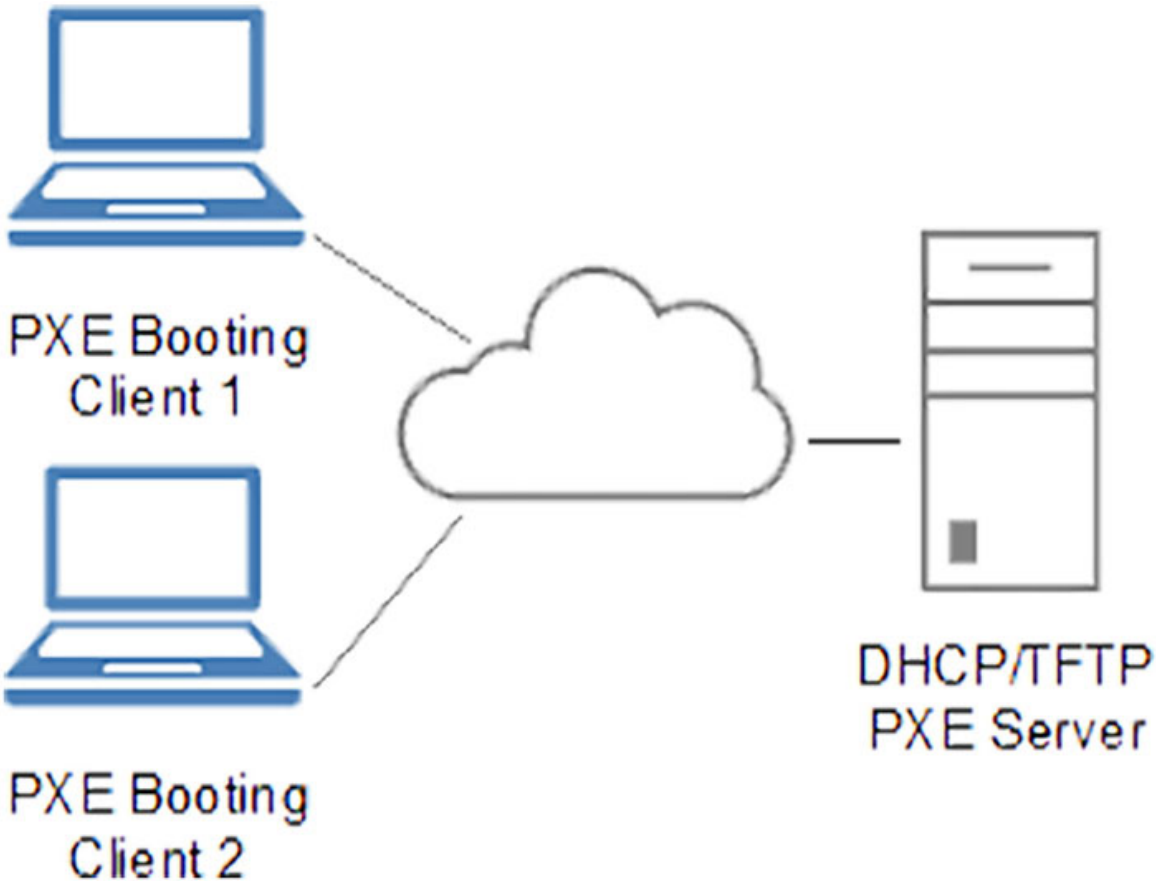


Figure 10.17: *PXE/TFTP overview. Source: Wikipedia*

Conclusion

Sharing files with users and other devices in the network are required in most of the enterprises. Users and applications can store data in central shared locations, such as personal files or backups. The central server is responsible for protecting the files from unwanted access and can be used to be included in periodic backups.

This chapter covered the most important protocols, NFS and SMB, including the installation of the service and the usage of the clients. The protocol FTP is covered as well, showing the installation of the server and the use of the client.

Key facts

- NFS protocol is the most popular one for Linux systems.
- SMB protocol is mainly used to interoperate with Windows systems.
- FTP was a popular protocol to share directories with users.

- TFTP is a protocol to store files for automatic installations using PXE.

Questions

1. What is the default file to configure the *NFS* server?
    a. /etc/nfs.conf
    b. /etc/nfsd.conf
    c. /etc/nfs/nsfd.conf
2. What file includes the exported directories configuration?
    a. /etc/exportfs
    b. /etc/exports
    c. /etc/share.conf
3. What is the main configuration file for the Samba service?
    a. samba.conf
    b. smbd.conf
    c. smb.conf
4. What command is used to operate as the client to the SMB protocol?
    a. smbtree
    b. smbclient
    c. smbget
5. What ports are used on the FTP server?
    a. 19/tcp
    b. 20/tcp
    c. 21/tcp

Answers

1. a
2. b
3. c
4. b

5. b and c

CHAPTER 11

Databases

Introduction

A database is an organized collection of data stored electronically in a computer system. This database is controlled by a **database management system** (**DBMS**). The data, the DBMS, and the applications associated are referred to *database system*. Applications use databases to store data in databases to be able to retrieve when required. This chapter will introduce to the basics of databases and the differences between relational and **non-relational** (**NoSQL**) database solutions.

The language **Structured Query Language** (**SQL**) is used in all relational databases. This chapter will explain how to query data, insert, and modify data inside a relational database. Different examples will be covered to demonstrate the usage of this language.

This chapter will focus on **MariaDB** as one of the most popular open-source relational databases. It will cover installation, configuration, backup, and restore for the service. It will also go over the process for the client to access the service and how to manipulate the data, such as creating databases, tables, and inserting data. A lightweight solution library database called **SQLite** will be covered.

For NoSQL, the database named **MongoDB** will be explained. This chapter will cover the installation and review the configuration. Access and manipulation of the data as a client will be described.

Structure

In this chapter, we will discuss the following topics:

- Relational databases
- Structured Query Language (SQL)
- MariaDB server
    - MariaDB client and tasks
- SQLite
- NoSQL databases

- MongoDB databases
- MongoDB client and tasks

Relational databases

Relation databases is a popular solution for most of the applications that require to store structured data, such as Web applications or services. A relational database is based on a *relational model*. In this model, the data is organized into one or more tables. Each table will contain rows and columns. The rows are usually called *records*, and the columns *attributes*. Each row will contain several attributes and will have a unique key to identify each row. Refer to *figure 11.1*:

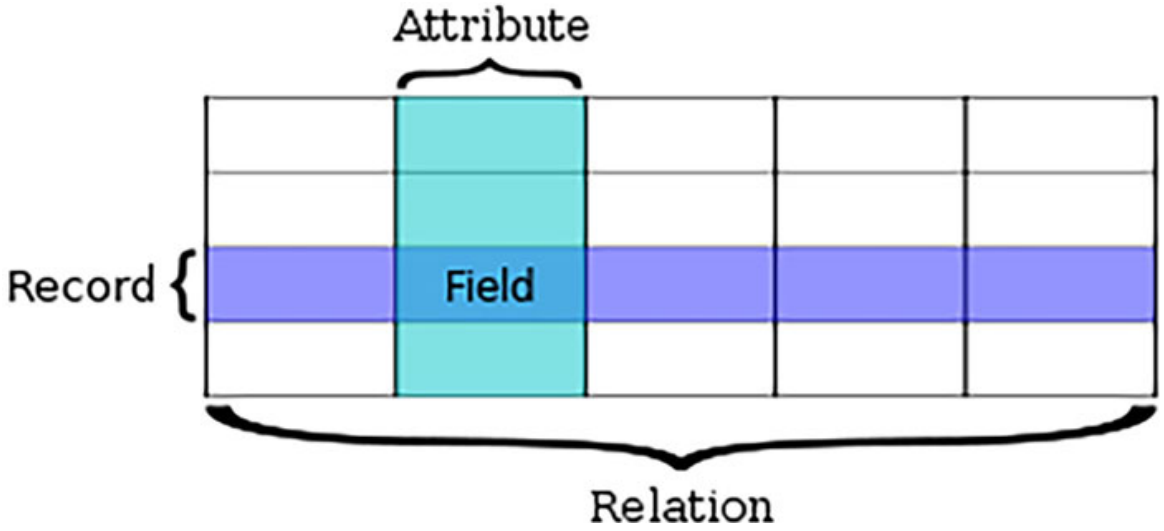


Figure 11.1: *Relation between record and attributes*

Each column *(attribute*) will have assigned a name, a type of the data expected (for example, number or string), and the maximum size. A column will indicate if it is mandatory to introduce data and if it requires to be unique in the table. The data assigned in a column for a record is called *field*. The common columns types available are described in *table 11.1*:

Type	Description
CHAR	Fixed width *n*-character string.
VARCHAR	Variable width string with a maximum size of *n* characters.
BOOLEAN	Can store values *TRUE* and *FALSE*.
INTEGER	Numerical types for integer numbers.
FLOAT DOUBLE	Numerical types for floating point numbers.

DECIMAL	An exact fixed-point number.
DATE	For date values.
TIME	For time values.
TIMESTAMP	For time and date together.

Table 11.1: *Column types and descriptions*

A table can define which column or columns are going to be used as *Primary Key*. This *Primary Key* is going to be used to define a unique value between the rows. This *key* can be a column with data that will be unique (for example, the ID card of a student table), or it can be a column that will autogenerate a unique value. Only one primary key can be defined in a table. *Figure 11.2* features an example of a Primary key:

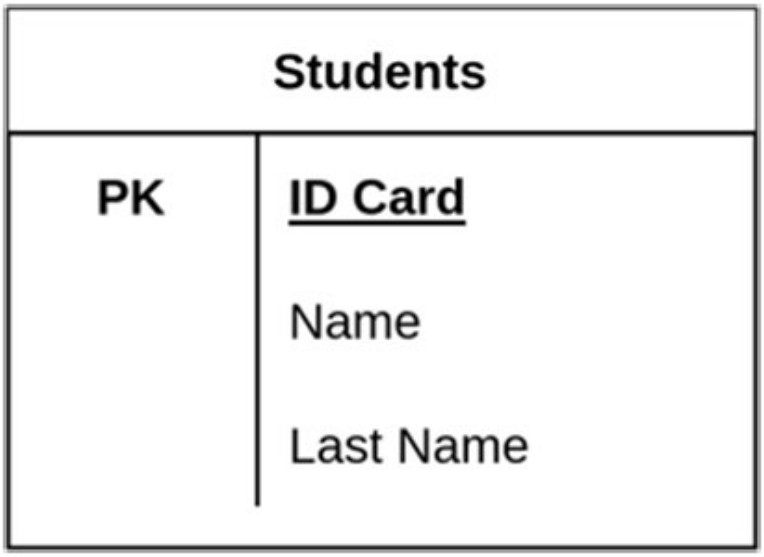


Figure 11.2: *Primary key example*

A *Primary Key* cannot be blank and needs to be unique in the table. Other columns in the table can be optionally defined as *Unique Key* or as *Foreign Key*:

- **Unique Key:** This does not allow to have duplicated values in the column. For example, a phone number should be unique in a table of students. A table can have several *Unique Keys*. Refer to *figure 11.3*:

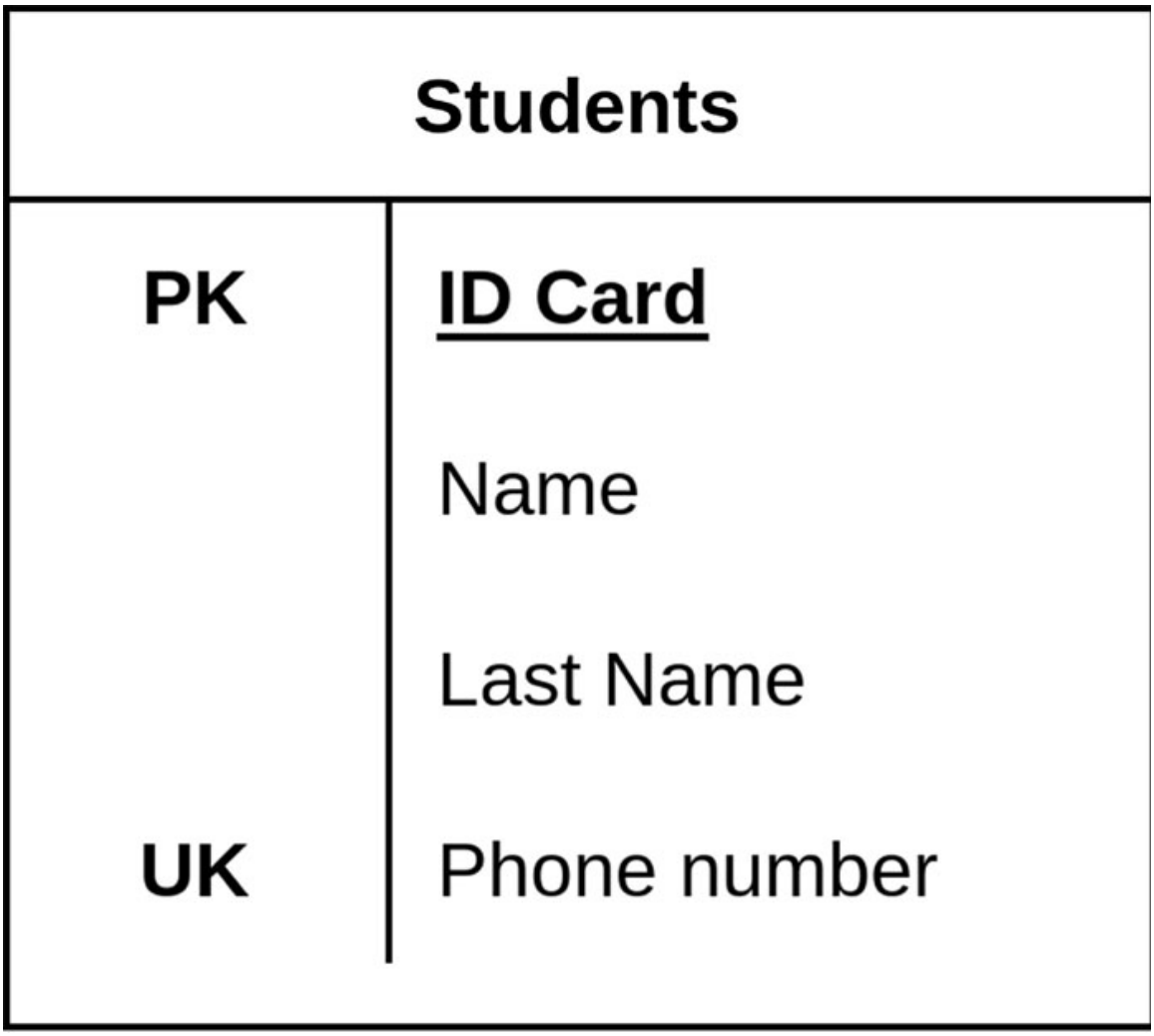


Figure 11.3: *Unique key example*

- **Foreign Key:** Allows to specify the value from a column in another table. It ensures not to allow to specify a value that does not exist in the referred column. For example, a subject record in a table cannot refer to a teacher who does not exist on the teacher's table. A table can define multiple *Foreign Keys*. Refer to *figure 11.4*:

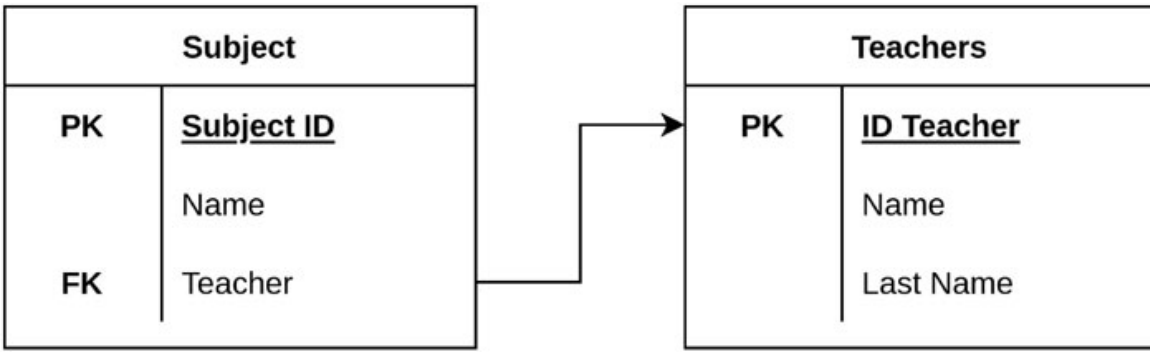


Figure 11.4: *Foreign Key example*

Relationships are a logical connection between different tables. A record (row) of one table can reference or be referenced by another record from a different table. There are three possible relationships:

- **One-to-one relationship (1:1):** One record can refer only to one element. For example, a country has only one capital city. Refer to *figure 11.5*:

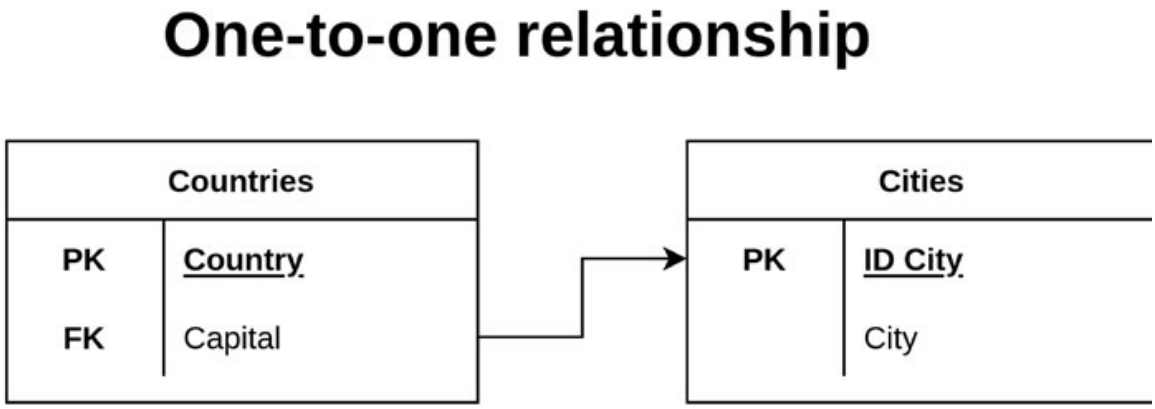


Figure 11.5: *One-to-one diagram*

- **One-to-many relationship (1:N):** A record can be referenced by many elements. For example, a country has several cities, but a city can only be part of one country. Refer to *figure 11.6*:

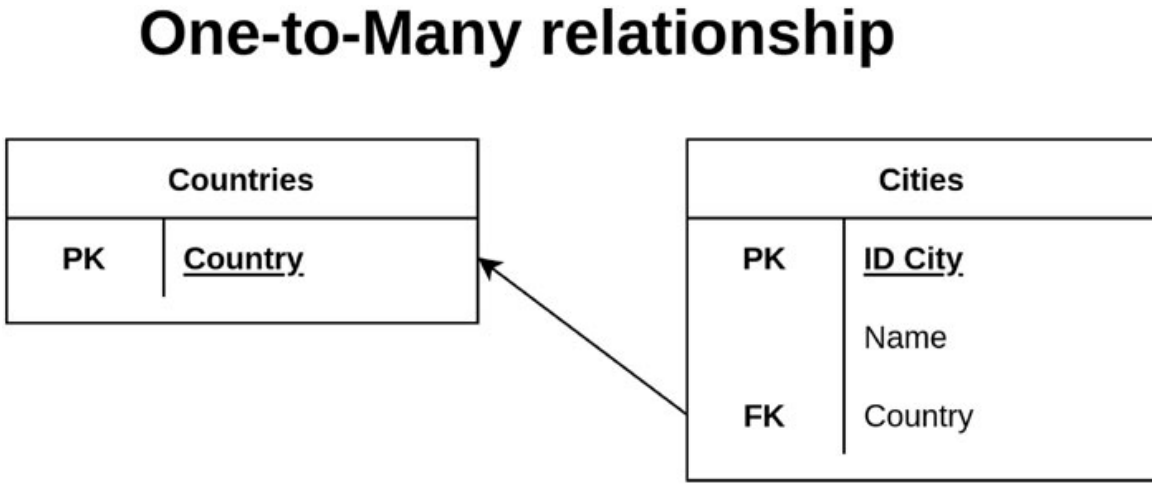


Figure 11.6: *One-to-many diagram*

- **Many-to-many relationship (M:M):** Several records can be referenced by many elements. For example, a country can have several official languages, and a language can be official in many countries. This case requires having an intermediate table to keep the relations. Refer to *figure 11.7*:

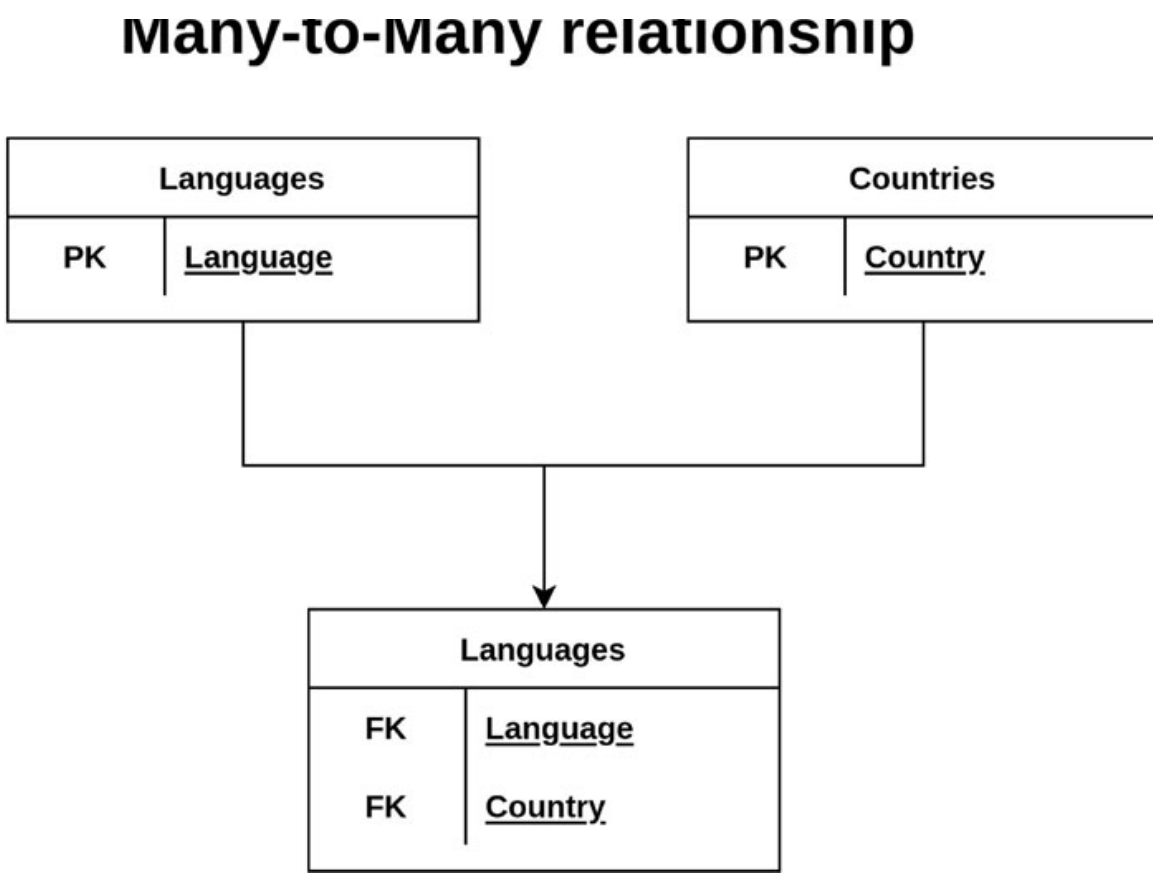


Figure 11.7: *Many-to-many diagram*

The most popular free relational databases are the following:

- **MariaDB**: Flexible and powerful **Relational Database Management System** (**RDBMS**). It is forked from the project **MySQL** created after *Oracle* acquired *Sun Microsystems*. **MariaDB** is fully under **GNU General Public License** (**GPL**).
- **PostgreSQL**: A powerful database system with advanced features. More suitable for bigger databases than **MariaDB**.
- **SQLite**: A library that software developers embed in their applications.

Database systems require to guarantee the transactions are processed reliably. The principle to ensure the transactions are completed correctly is called **ACID**, which stands for:

- **Atomicity**: Ensures that all operations in the transaction occur or nothing at all occurs.
- **Consistency**: The database must be consistent before and after the transaction.

- **Isolation**: Multiple transactions occur independently without interference.
- **Durability**: Guarantees that once the transaction has been committed, it will be available even in the case of a system failure.

The four main actions against databases are: **create**, **read**, **update**, and **delete**, and these operations are called **CRUD** operations.

Structured Query Language (SQL)

This declarative language is used to manage data held in an RDBMS. It can be used to create, manipulate and query data. Although *SQL* has standards defined, the implementation between different vendors is not necessarily following them.

A statement performs an operating against the database engine. The statements are ending with a semicolon (;). The SQL statements are categorized into five categories:

- **Data Definition Language** (**DDL**): Used to define the database schema. It is possible to create, modify, or delete one object of the database. For example, a database or a table. The main statements available are as follows:
    - **CREATE**: Used to create a new object. After the keyword, CREATE follows the object to be created (for example, DATABASE or TABLE) and the name desired.
    - **DROP**: Removes an existing object inside of the database system.
    - **ALTER**: Modifies an existing object inside of the database system.
    - **TRUNCATE**: Deletes all the objects of the table specified.
    - **RENAME**: Renames an existing object inside of the database system.
- **Data Query Language** (**DQL**): Used to query the data within the database system. The statement used to query data is **SELECT**.
- **Data Manipulation Language** (**DML**): Used to manipulate the data inside a table. The main statements available are as follows:
    - **INSERT**: Used to insert data inside a table.
    - **UPDATE**: Used to modify data inside a table.
    - **DELETE**: Used to delete records inside a table.
- **Data Control Language** (**DCL**): Used to operate with rights, permissions, and other access control rules to the database system. Popular main statements are as follows:

- **GRANT**: Allows access to objects inside of the database system.
- **REVOKE**: Cancels access to objects inside of the database system to a previously allowed one.

- **Transactional Control Language** (**TCL**): Used to start a transaction to the database system. The common use statements are:
  - **START TRANSACTION**: Starts a new transaction of different actions against the database.
  - **COMMIT**: Indicates the transaction of the actions has finished.
  - **ROLLBACK**: Discard the actions sent from the beginning of the transaction.

Figure 11.8 illustrates the different statement categories:

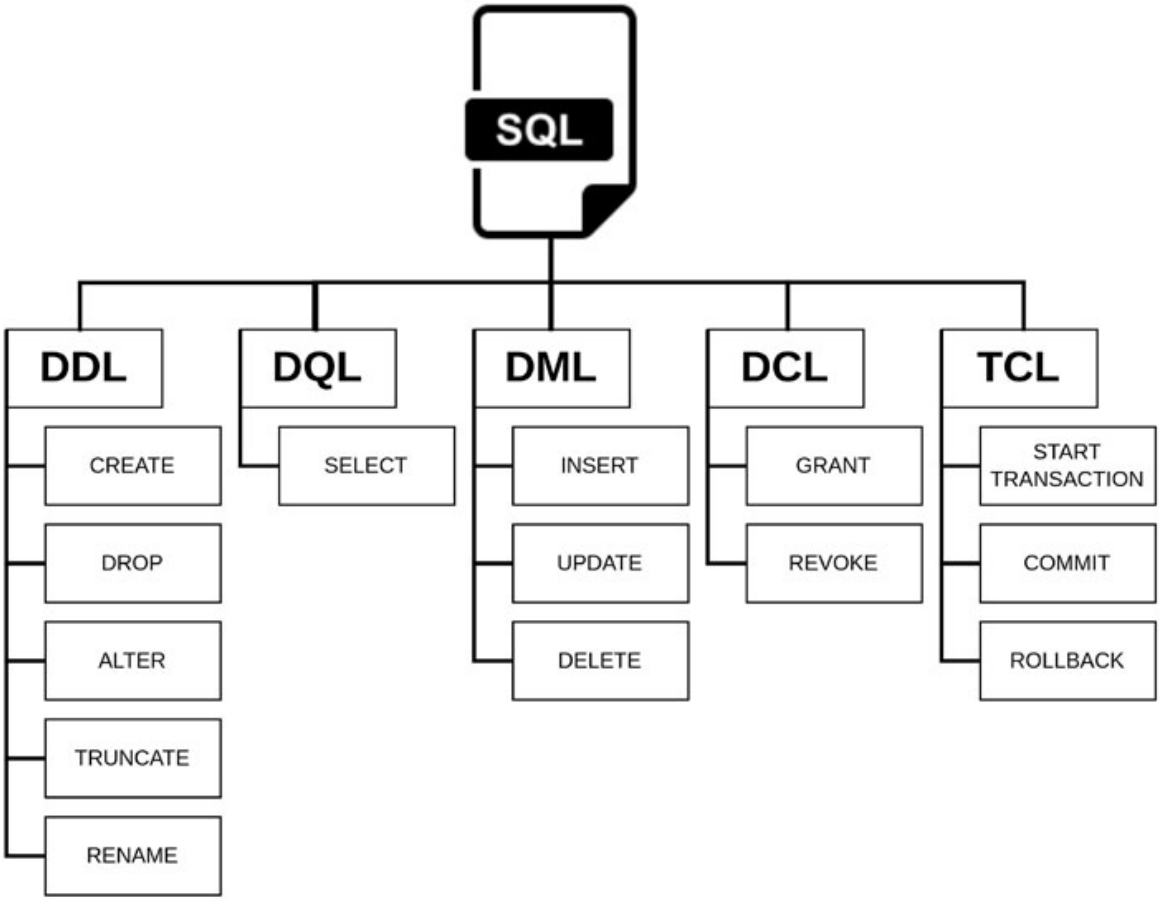


Figure 11.8: *SQL statements diagram example*

SQL syntax is case-insensitive. The statements can be written in lower case, upper case, or combination. A recommendation for easy readability is to use

upper case for the *SQL* keywords and lower case for the objects and the data provided. Some useful commands syntax are described as follows.

CREATE statement

This statement is used to create objects of the specified type. The syntax is the following:

```
CREATE [ OR REPLACE ] <object_type> [ IF NOT EXISTS ] <object_name>
  [ <object_type_properties> ]
  [ <object_type_params> ]
```

For example, to create a new database named **school**, which will contain tables and other objects, the syntax is as follows:

```
CREATE DATABASE school;
```

To create a table, it is required to define the columns and the definition for each column. The following example creates a table called **students** with three columns, one of them defined as **PRIMARY KEY**.

```
CREATE TABLE students
  (id INTEGER PRIMARY KEY, name CHAR(20), lastname CHAR(50));
```

DROP statement

This statement is used to delete objects of the specified type. The syntax is the following:

```
DROP <object_type> [ IF EXISTS ] <identifier>  [ CASCADE | RESTRICT
]
```

For example, to delete the table previously created named *students*, the following syntax should be used:

```
DROP TABLE students;
```

ALTER statement

This statement is used to modify an object. The syntax is as follows:

```
ALTER <object_type> <object_name> <actions>
```

For example, add a new column called *phone* to the previous table created; the following syntax can be used:

```
ALTER TABLE students ADD  COLUMN phone CHAR(20) UNIQUE;
```

INSERT statement

This statement is used to insert data in a table; the syntax is as follows:

```
INSERT INTO <target_table> [ ( <target_col_name> [ , … ] ) ]
   {
    VALUES ( { <value> | DEFAULT | NULL } [ , … ] ) [ , ( … ) ]   |
    <query>
   }
```

For example, to insert a new student to the previously created table named *students*, the following syntax will perform it:

```
INSERT INTO students
VALUES(1, "Alberto", "Gonzalez", "64543210");
```

It is possible to specify only the columns where the data will be specified, as is shown in the following example:

```
INSERT INTO students(id, name, lastname)
VALUES(2, "John", "Doe");
```

In the previous example, the field **`phone`** will be set to **`null`**. If the column is configured to not be blank, an error will be thrown, and the insert will not be performed.

SELECT statement

This statement is used to query data from data. It is possible to query all the data or filter for specific values. It is also possible to show all the columns or specify the columns to be listed. The simple syntax for **`SELECT`** is as follows:

```
SELECT [ * | column_list ]
     [FROM table_references]
     [WHERE where_condition]
```

The following example will query for the records inside the table *students* where the column *name* contains the value, Alberto*:*

```
SELECT * FROM students WHERE name = "Alberto";
```

UPDATE statement

This statement is used to update rows existing in a table, the syntax as shown:

```
UPDATE <target_table>
   SET <col_name> = <value> [ , <col_name> = <value> , … ]
   [ FROM <additional_tables> ]
```

```
    [ WHERE <condition> ]
```

The following example will update the field **phone** for the user with the name *Alberto* in the field **name**. As this field is not **unique**, this operation can update several records if the name is duplicated.

```
UPDATE students SET phone = "657225520" WHERE name = "Alberto";
```

DELETE statement

This statement is used to remove rows from an existing table. The syntax is as follows:

```
DELETE FROM <table_name>
     [ WHERE <condition> ]
```

In most of the cases, it is important to use the clause **WHERE** to limit the records to be removed. If it is not specified, all the records will be removed. The following example removes the student with the associated *id* to the number 1.

```
DELETE FROM students WHERE id = 1;
```

GRANT statement

Assign privileges to user accounts and roles. Syntax is complex and depends of the database system used, the basic syntax is as follows:

```
GRANT privilege
   ON object_type
   TO user_or_role;
```

For example, to provide full access to the user *agonzalez* when it is connecting from localhost to the previous database created called the *school*, the following syntax will perform the action:

```
GRANT ALL PRIVILEGES ON school.* TO 'agonzalez'@'localhost';
```

REVOKE statement

Removes privileges for a user or role. The simplified syntax is indicated in the following code:

```
REVOKE
   privilege
   ON object_type
   FROM user_or_roleitu
```

For example, to remove the privilege previously assigned to *GRANT*, the following statement will perform the action:

```
REVOKE ALL PRIVILEGES ON school.* FROM 'agonzalez'@'localhost';
```

MariaDB server

For the installation of the **MariaDB server**, it is required to install the package **`mariadb-server`**, available in all the popular Linux distributions available. The installation in a Ubuntu and Debian derivatives distributions will start the service and enable it to be started on boot automatically. The name of the service is **`mariadb.service`**. Red Hat Enterprise Linux distribution and derivates require to be started and enabled using the command **`systemctl`**.

It is important to have into consideration that **MariaDB** is a fork of **MySQL**, and some commands and configuration files are interchangeable. For example, the service path **`/usr/sbin/mysqld`** is a link to the binary file **`/usr/sbin/mariadbd`**.

The main configuration for the service is the file **`my.cnf`**. The location varies depending on the Linux distribution used:

- *Ubuntu, Debian*, and derivatives: Inside of the directory **`/etc/mysql/`**. This file is a link to the file **`/etc/alternatives/my.cnf`**, which at the same time, is a link to file **`/etc/mysql/mariadb.cnf`**.
- Red Hat Enterprise Linux and derivatives: The file is inside the directory **`/etc/`**.

The file **`my.cnf`** is an entry point configuration used to indicate the directories with the global configurations for the service. The following code shows the content of **`my.cnf`** in a Ubuntu server:

```
[client-server]
socket = /run/mysqld/mysqld.sock
!includedir /etc/mysql/conf.d/
!includedir /etc/mysql/mariadb.conf.d/
```

The following code shows the content of the file **`/etc/my.cnf`** in a Red Hat Enterprise Linux system:

```
[client-server]
!includedir /etc/my.cnf.d
```

Figure 11.9 shows the directory structure for both distributions:

Figure 11.9: *Directory content on Ubuntu and Red Hat Enterprise Linux*

The files **`50-server.cnf`** (for Ubuntu system*)* and **`mariadb-server.cnf`** (for RHEL system*)* includes the configuration related to the server. The following code shows the default configuration in a Ubuntu distribution:

```
[server]
[mysqld]
pid-file                = /run/mysqld/mysqld.pid
basedir                 = /usr
bind-address            = 127.0.0.1
expire_logs_days        = 10
character-set-server  = utf8mb4
collation-server      = utf8mb4_general_ci
[embedded]
[mariadb]
[mariadb-10.6]
```

Table 11.2 shows the option groups available in the configuration files and the description:

Group	Description
[client-server]	Options read by all **MariaDB client** and the **MariaDB Server**.
[server]	Options read by **MariaDB Server**.
[mysqld]	Options read by *mysqld*, which includes both **MariaDB Server** and **MySQL Server**.
[mysqld-X.Y]	Options read by a specific version of *mysqld*, which includes both **MariaDB Server** and **MySQL Server**. For example, *[mysqld-10.4]*.
[mariadb]	Options read by **MariaDB Server.**
[mariadb-X.Y]	Options read by a specific version of **MariaDB Server**. For example, *[mariadb-10.4]*.
[mariadbd]	Options read by **MariaDB Server**. Available starting with **MariaDB** *10.4.6*.
[mariadbd-X.Y]	Options read by a specific version of **MariaDB Server**. For example, *[mariadbd-10.4]*.

	Available starting with **MariaDB** 10.4.6.
[galera]	Options read by MariaDB Server, but only if it is compiled with Galera Cluster support.

Table 11.2: *Option groups and descriptions*

The default port used by the **MariaDB server** is 3306 on protocol **Transmission Control Protocol** (**TCP**). The service will be listening in the interfaces matching the **`bind-address`** specified or in all interfaces if the specified IP is *0.0.0.0*. The package includes the following three commands to start the service:

- **`mariadbd`**: The main service used by the service **`mariadb.service`**.
- **`mariadbd-multi`**: In systems running server *database services*, it is required to use this command instead the default one.
- **`mariadbd-safe`**: Used to start the service without **`systemd`**. This approach is needed to start troubleshooting or performing a recovery of a failed database server.

It is possible to use the option **`--print-defaults`** for the command **`mariadbd`** (or **`mysqld`**) to show on the screen the options read from the configuration files, as is shown in *figure 11.10*:

```
root@ubuntu:~# mariadbd --print-defaults
mariadbd would have been started with the following arguments:
--socket=/run/mysqld/mysqld.sock --pid-file=/run/mysqld/mysqld.pid --basedir=/usr
--bind-address=127.0.0.1 --expire_logs_days=10 --character-set-server=utf8mb4 --co
llation-server=utf8mb4 general ci
```

Figure 11.10: *Output example command mariadbd*

The service **`mariadbd`** (or **`mysqld`**) is executed under the user **`mysql`** and the group **`mysql`**. When the service is started, the logs are redirected to **`journald`** or the file **`/var/log/mariadb/mariadb.log`**, depending on the configuration previously described. The following code shows an example of the information shown after starting.

```
Starting MariaDB 10.6.7 database server…
2022-09-18 21:06:11 0 [Note] /usr/sbin/mariadbd (server 10.6.7-
MariaDB-2ubuntu1.1) starting as process 3083 …
2022-09-18 21:06:11 0 [Note] InnoDB: Compressed tables use zlib
1.2.11
2022-09-18 21:06:11 0 [Note] InnoDB: Number of pools: 1
2022-09-18 21:06:11 0 [Note] InnoDB: Using crc32 + pclmulqdq
instructions
```

```
2022-09-18 21:06:11 0 [Note] InnoDB: Initializing buffer pool, total
size = 134217728, chunk size = 134217728
2022-09-18 21:06:11 0 [Note] InnoDB: Completed initialization of
buffer pool
2022-09-18 21:06:11 0 [Note] InnoDB: 128 rollback segments are
active.
2022-09-18 21:06:11 0 [Note] InnoDB: Creating shared tablespace for
temporary tables
2022-09-18 21:06:11 0 [Note] InnoDB: Setting file './ibtmp1' size to
12 MB. Physically writing the file full; Please wait …
2022-09-18 21:06:11 0 [Note] InnoDB: File './ibtmp1' size is now 12
MB.
2022-09-18 21:06:11 0 [Note] InnoDB: 10.6.7 started; log sequence
number 42479; transaction id 14
2022-09-18 21:06:11 0 [Note] InnoDB: Loading buffer pool(s) from
/var/lib/mysql/ib_buffer_pool
2022-09-18 21:06:11 0 [Note] Plugin 'FEEDBACK' is disabled.
2022-09-18 21:06:11 0 [Note] InnoDB: Buffer pool(s) load completed
at 220918 21:06:11
2022-09-18 21:06:11 0 [Warning] You need to use --log-bin to make --
expire-logs-days or --binlog-expire-logs-seconds work.
2022-09-18 21:06:11 0 [Note] Server socket created on IP:
'127.0.0.1'.
2022-09-18 21:06:11 0 [Note] /usr/sbin/mariadbd: ready for
connections.
Version: '10.6.7-MariaDB-2ubuntu1.1'  socket:
'/run/mysqld/mysqld.sock'  port: 3306  Ubuntu 22.04
Started MariaDB 10.6.7 database server.
```

A database system uses a storage engine to store the data on the disk. The storage engine has different features available, and the performance varies depending on the use case. *Table 11.3* shows the four more popular engines available for **MariaDB**:

Engine	Description
InnoDB	The default storage engine from **MariaDB** *10.2*. For earlier releases, XtraDB was a performance-enhanced fork of InnoDB and is usually preferred.
XtraDB	The best option for **MariaDB** *10.1* and earlier versions.
MyISAM	Oldest storage and the previous default one in the older version of **MySQL** and **MariaDB**.

Aria	An improved engine based on legacy **MyISAM**.

Table 11.3: *Engines and description list.*

The benefits of *InnoDB* are as follows:

- It is s transactional and well-suited for **Online Transactional Processing** (**OLTP**) workloads.
- It is **Atomicity**, **Consistency**, **Isolation**, and **Durability** (**ACID**) compliant.
- Performs well for mixed read-write workloads.
- Supports online Data Definition Language (DDL).

The **MariaDB server** package includes a tool to administrate the service named **`mariadb-admin (mysqladmin)`**. It can be used to:

- Monitor what **MariaDB** is doing.
- Get usage statistics and variables from the **MariaDB server**.
- Create and remove databases.
- Reset logs, statistics, and tables.
- Kill long-running queries.
- Stop the server.
- Reload the configuration.
- Start and stop secondary nodes (when high availability or replication is used).
- Check if the service is responding.

The syntax is **`mariadb-admin action [arguments]`**. *Table 11.4* shows some of the popular actions available:

Action	Description
create *databasename*	Create a new database
debug	Tell the server to write and debug information to log or journald.
drop *databasename*	Oldest storage and the previous default one in the older version of **MySQL** and **MariaDB**.
extended-status	Return all status variables and their values
kill *id*	Kill a client thread
password *newpw*	Change the old password for the administrator to a new one.
ping	Check if the service is alive.

processlist	Show a list of active threads in the database server.
reload	Reload the privileges configuration.
refresh	Flush all tables and close and open the log files.
shutdown	Send a signal to the service to be stopped.
status	Gives a short status message from the server.
variables	Prints the variables available.
version	Returns version and the status from the server.

Table 11.4: *Actions and description for command mariadb-admin (mysqladmin).*

Figures 11.11 to *11.15* show different examples of the usage of the command **`mariadb-admin`** with several commands, and the output is shown.

- The version and short status of the **MariaDB server** can be seen in *figure 11.11*:

```
root@ubuntu:~# mariadb-admin version
mariadb-admin  Ver 9.1 Distrib 10.6.7-MariaDB, for debian-linux-gnu on x86_64
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Server version          10.6.7-MariaDB-2ubuntu1.1
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /run/mysqld/mysqld.sock
Uptime:                 44 min 30 sec

Threads: 1  Questions: 67  Slow queries: 0  Opens: 33  Open tables: 26  Queries
per second avg: 0.025
```

Figure 11.11: *Output example command mariadb-admin*

- A short status information about the server can be seen in *figure 11.12*:

```
root@ubuntu:~# mariadb-admin status
Uptime: 2786  Threads: 1  Questions: 68  Slow queries: 0  Opens: 33  Open tables
: 26  Queries per second avg: 0.024
```

Figure 11.12: *Output example command mariadb-admin*

- To check if the service is started, refer to *figure 11.13*:

```
root@ubuntu:~# mariadb-admin ping
mysqld is alive
```

Figure 11.13: *Checking if the service has started*

- To list the clients and the task performed, refer to *figure 11.14*:

```
root@ubuntu:~# mariadb-admin processlist
+----+------+-----------+----+---------+------+----------+------------------+----------+
| Id | User | Host      | db | Command | Time | State    | Info             | Progress |
+----+------+-----------+----+---------+------+----------+------------------+----------+
| 42 | root | localhost |    | Query   | 0    | starting | show processlist | 0.000    |
+----+------+-----------+----+---------+------+----------+------------------+----------+
```

Figure 11.14: *Output example command mariadb-admin*

- To set a new password for the user root (by default is empty, only accessible from localhost), refer to *figure 11.15*:

```
root@ubuntu:~# mariadb-admin password Start123
```

Figure 11.15: *Example setting password with command mariadb-admin*

- To request the server to be stopped, refer to *figure 11.16*:

```
root@ubuntu:~# mariadb-admin shutdown
root@ubuntu:~# mariadb-admin ping
mariadb-admin: connect to server at 'localhost' failed
error: 'Can't connect to local server through socket '/run/mysqld/mysqld.sock' (2)'
Check that mariadbd is running and that the socket: '/run/mysqld/mysqld.sock' exists
```

Figure 11.16: *Output example command mariadb-admin*

MariaDB client and tasks

Client to access to **MariaDB server** is using part of the package **`mariadb-client`**. The command to access to the server is **`mariadb`** (or **`mysql`**), and *table 11.5* shows the common options available:

Option	Description
-e, --execute=name	Execute the statement and quit.
-h, --host=name	Connect to the host specified instead of using a *Unix Socket*.
--max-allowed-packet=num	The maximum packet length to send to or receive from the server. The default is 16 MB and the maximum 1 GB. Useful to use when importing big files.
-p, --password[=name]	Password to use when connecting to server.
-P, --port=num	Port used to use for connection instead of 3306.
--print-defaults	Output a list of the options for the client.
-u, --user=name	User for login if not the current user.
-v, --verbose	Show more information.
-V, --version	Output version information and exit.

Table 11.5: *Common options available for the command mariadb (mysql)*

The client can work in two different modes:

- **Interactive**: After connecting to the database server, the user can write several statements to operate with the server and will exit the client by writing the command QUIT.
- **One shot**: The client will read one or more statements from the standard input, and it will execute them to the server and exit the client execution.

The client allows an argument to specify the database to be used. If it is not specified, the connection to the server will be without using a database, and the user will need to specify manually (using the statement USE) the database to operate with it. A user can switch between databases.

Figure 11.17 shows how running the client from the database server with a root user is not needed, and neither is to specify a username and password.

```
root@ubuntu:~# mariadb
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 10.6.7-MariaDB-2ubuntu1.1 Ubuntu 22.04

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 11.17: *Output example command mariadb*

MariaDB (and MySQL) allows to create a users with the statement **`CREATE USER`**. The simple syntax is the following:

```
CREATE [OR REPLACE] USER user_specification [authentication_options]
```

Figure 11.18 shows how to create a user named **`agonzalez`** with the password **`P@sssw0rd`**.

The user will be accessed with that password only from IP 192.168.122.226.

```
MariaDB [(none)]> CREATE USER agonzalez@192.168.122.226 IDENTIFIED BY 'P@ssw0rd';

Query OK, 0 rows affected (0.007 sec)
```

Figure 11.18: *Example creating a user on MariaDB*

A client can connect with the authentication parameters specified and from the host allowed. It is possible to specify the character % to accept from all the source

IP addresses. *Figure 11.19* shows the connection to the database from a remote system:

```
[agonzalez@rhel ~]$ mysql -uagonzalez -pP@ssw0rd -h192.168.122.43
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 10.6.7-MariaDB-2ubuntu1.1 Ubuntu 22.04

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 11.19: *Example connecting to a remote MariaDB server*

The statement **`SHOW DATABASES`** lists the databases available for the user. A user created will have only read access to the database called **`information_schema`**, which has information about databases and tables. The rest of the permissions need to be granted by an administrator. *Figure 11.20* shows the example output listing the databases available.

```
MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
+--------------------+
1 row in set (0.001 sec)
```

Figure 11.20: *Output example MariaDB*

An administrator (or a user with specific permission to create new databases) can create databases using the statement **`CREATE DATABASE`**. After creating a database, it is possible to use **`GRANT`** to assign permissions to exist users. *Figure 11.21* illustrates the database creation and the permission assignation.

```
MariaDB [(none)]> SELECT USER();
+----------------+
| USER()         |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.000 sec)

MariaDB [(none)]> CREATE DATABASE school;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON school.* to agonzalez@192.168.122.226;
Query OK, 0 rows affected (0.006 sec)
```

Figure 11.21: *Output example MariaDB*

After the permissions are granted for a regular user, the user will be able to see the database where the access was granted. *Figure 11.22* illustrates the output after the permission was set.

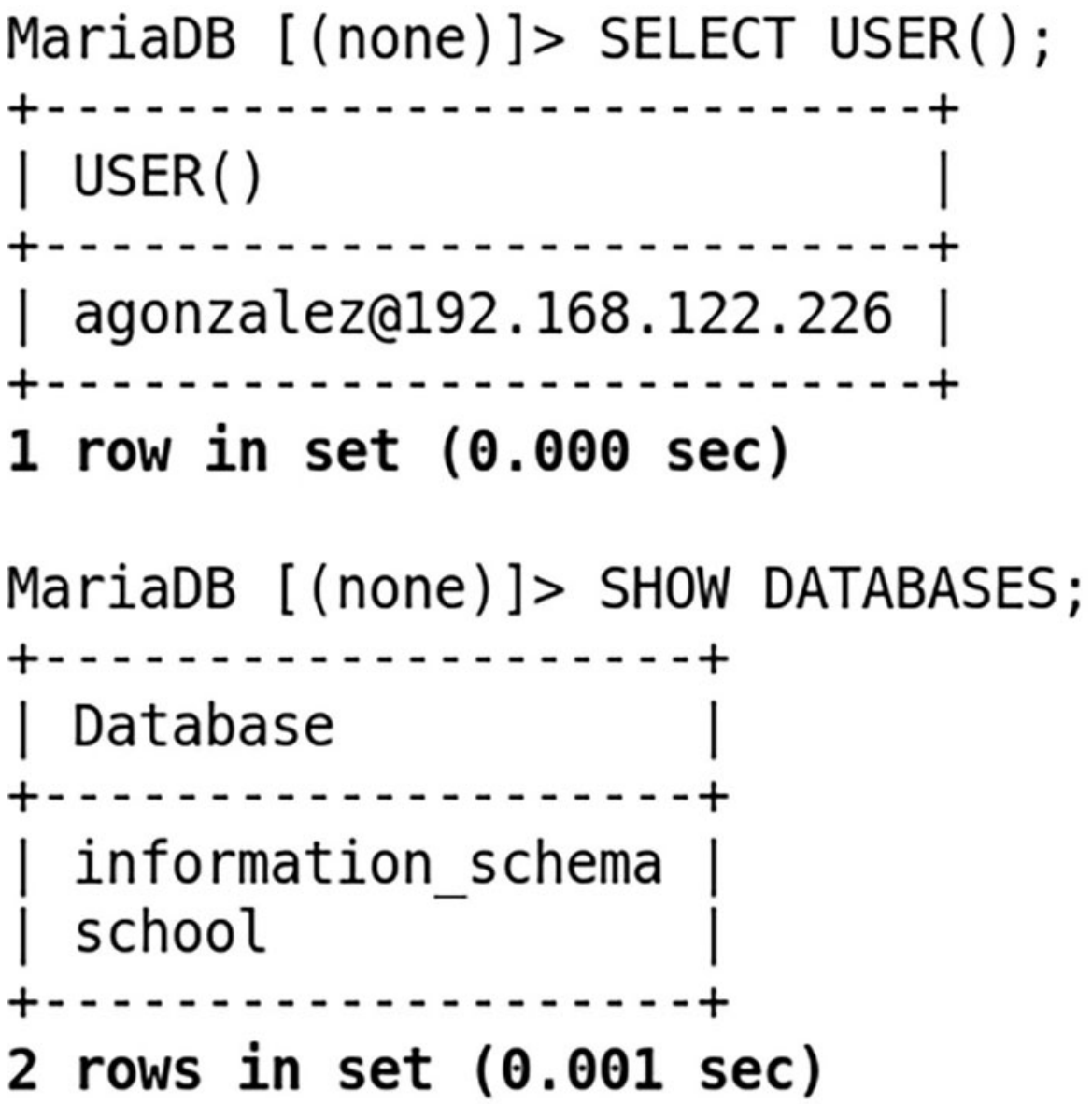


Figure 11.22: *Output example MariaDB*

A user can switch between databases using the statement *USE* followed by the names of the database that wants to work. After selecting the database, the user can create tables inside of the database or perform other tasks, such as query existing tables. *Figure 11.23* shows the switching to the database school and the creation of a table inside.

```
MariaDB [(none)]> USE school;
Database changed
MariaDB [school]> CREATE TABLE students  (id INTEGER PRIMARY KEY, name CHAR(20),
lastname CHAR(50));
Query OK, 0 rows affected (0.024 sec)
```

Figure 11.23: *Output example MariaDB*

To list the tables available inside a database, the statement used is **SHOW TABLES**, which will list the tables inside of the current database, as is illustrated in *figure 11.24*:

```
MariaDB [school]> SHOW TABLES;
+-------------------+
| Tables_in_school  |
+-------------------+
| students          |
+-------------------+
1 row in set (0.001 sec)
```

Figure 11.24: *Output example MariaDB.*

It is possible to use the statement *DESCRIBE,* followed by the name of the table, to obtain a list of the columns and the type definition. *Figure 11.25* shows the output of the statement using the previous table created.

```
MariaDB [school]> DESCRIBE students;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| id       | int(11)  | NO   | PRI | NULL    |       |
| name     | char(20) | YES  |     | NULL    |       |
| lastname | char(50) | YES  |     | NULL    |       |
+----------+----------+------+-----+---------+-------+
3 rows in set (0.001 sec)
```

Figure 11.25: *Output example MariaDB*

Figure 11.26 shows the usage of the statement **INSERT** and the importance of the *Primary Key*.

```
MariaDB [school]> INSERT INTO students VALUES(1, "Alberto", "Gonzalez");
Query OK, 1 row affected (0.001 sec)

MariaDB [school]> INSERT INTO students VALUES(1, "John", "Doe");
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
```

Figure 11.26: *Output example MariaDB*

Using the statement **`SELECT`**, it is possible to obtain all the records in the table or partial information, as is illustrated in *figure 11.27*:

```
MariaDB [school]> SELECT * FROM students;
+----+---------+----------+
| id | name    | lastname |
+----+---------+----------+
|  1 | Alberto | Gonzalez |
+----+---------+----------+
1 row in set (0.002 sec)

MariaDB [school]> SELECT name FROM students;
+---------+
| name    |
+---------+
| Alberto |
+---------+
1 row in set (0.002 sec)
```

Figure 11.27: *Output example MariaDB*

The **MariaDB** client package includes a command named **`mariadb-dump (mysqldump)`** to export the content of a database (or a specific table) to be used to generate a file with the instructions to be able to rebuild it. The following excerpt shows an example of the content generated with the command:

```
--
-- Table structure for table `students`
--
DROP TABLE IF EXISTS `students`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
```

```
CREATE TABLE `students` (
  `id` int(11) NOT NULL,
  `name` char(20) DEFAULT NULL,
  `lastname` char(50) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
/*!40101 SET character_set_client = @saved_cs_client */;
--
-- Dumping data for table `students`
--
LOCK TABLES `students` WRITE;
/*!40000 ALTER TABLE `students` DISABLE KEYS */;
INSERT INTO `students` VALUES (1,'Alberto','Gonzalez');
/*!40000 ALTER TABLE `students` ENABLE KEYS */;
UNLOCK TABLES;
```

The command **`mariadb (mysql)`** allows to use the **`standard input`** to specify the actions to be performed. *Figure 11.28* shows how to use **`mariadb-dump`** to export the database content to be recreated, the usage of the **`standard input`** to drop the existing table, and the use of redirection to import the content from a file.

```
[agonzalez@rhel ~]$ mariadb-dump -pP@ssw0rd -h192.168.122.43 school > school.sql
[agonzalez@rhel ~]$ echo "DROP TABLE students;" | mariadb -p -h192.168.122.43 school
Enter password:
[agonzalez@rhel ~]$ echo "SHOW TABLES;" | mariadb -p -h192.168.122.43 school
Enter password:
[agonzalez@rhel ~]$ mariadb -pP@ssw0rd -h192.168.122.43 school < school.sql
[agonzalez@rhel ~]$ echo "SHOW TABLES;" | mariadb -p -h192.168.122.43 school
Enter password:
Tables_in_school
students
```

Figure 11.28: *Output example operating with mariadb-dump and mariadb commands*

SQLite

The database engine **SQLite** is a library to be used as embedded for the applications. With this solution, it is not needed to have a database server running. The database will be using a file that can be distributed. Users can use the command **`sqlite`** to operate with the databases, and the applications using the library available would be able to access the database file.

The command **`sqlite`** is part of the package **`sqlite3`** *(*in Debian and derivatives distributions) or part of the package **`sqlite`** (Red Hat Enterprise Linux and derivatives). The package is less than 1 megabyte in size. The command can be executed without any argument, but the common usage is to specify an existing

file that contains the database and the data or an empty file that will be used and initialized as a database file.

SQLite clients can work in interactive and using the standard input. Most of the *SQL statements* explained during this chapter can be used inside the client. *Figure 11.29* illustrates the creation of a new database file and the creation of a table inside.

```
root@ubuntu:~# sqlite3 school.db
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
sqlite> CREATE TABLE students  (id INTEGER PRIMARY KEY, name CHAR(20), lastname
CHAR(50));
sqlite> INSERT INTO students VALUES(1, "Alberto", "Gonzalez");
sqlite> SELECT * from students;
1|Alberto|Gonzalez
```

Figure 11.29: *Output example SQLite*

In the interactive mode, it is possible to type **`.help`** to obtain the actions possible using the client. Some of the popular options able to perform are described in *table 11.6*:

Action	Description
.backup FILE	Backup DB to FILE
.changes on|off	Show the number of rows changed by SQL
.dbinfo	Show status information about the database
.dump ?OBJECTS?	Render database content as SQL
.headers on|off	Turn the display of headers on or off
.show	Show the current values for various settings
.tables	List names of tables

Table 11.6: *Common actions for the interactive mode for SQLite.*

The advantages of using **SQLite** are as follows*:*

- It is not required to have a server running a *database engine*.
- The database file can be distributed to different systems that are not required to perform backup and restore.
- Lightweight database with most of the basic database functions.

The command **`sqlite`** has different options to specify the output format desired. Some of the options are: *--json*, *--list*, and *–table*. *Figure 11.30* illustrates the usage of **`sqlite`** client with the standard input.

```
root@ubuntu:~# echo ".tables" | sqlite3 school.db
students
root@ubuntu:~# echo "SELECT * from students;" | sqlite3 --table school.db
+----+---------+----------+
| id |  name   | lastname |
+----+---------+----------+
| 1  | Alberto | Gonzalez |
+----+---------+----------+
root@ubuntu:~# echo "SELECT * from students;" | sqlite3 --json school.db
[{"id":1,"name":"Alberto","lastname":"Gonzalez"}]
```

Figure 11.30: *Output example SQLite*

NoSQL databases

A **NoSQL** (also known as **non-only-SQL** or **non-relation**) database stores the data differently than relational tables. These databases are called *non-tabular databases* and contain a variety of types based on their data model. The popular data structure are as follows:

- **key-value pair**: The simplest type, each object is assigned to a unique key. To obtain the value, it is needed to specify the key.
- **wide column**: It uses tables, rows, and columns, but the names and format of the columns can vary from row to row.
- **graph**: Uses a graph structure for semantic queries with *nodess*, *edges,* and properties to represent and store data.
- **document**: It stores the data in *JSON* or *XML* format.

NoSQL databases are getting popular for applications generation big quantities of data due to their performance being better than relational databases. Applications used in big data and real-time Web (such as analytics) are some of the main use cases for NoSQL databases. Its unique features are as follows:

- **Flexible schemas**: They do not require to have the same set of fields and the data type for a field.
- **Horizontal scaling**: Adding nodes to share the load is a simple task in *NoSQL*.
- **Fast queries**: Querying a *NoSQL* is faster than a relational database.
- **Ease of use for developers**: For developers, it is easier to use *NoSQL* to store structured or unstructured data.

Some of the popular open-source database *NoSQL* solutions available are as follows:

- **MongoDB:** A document database with the scalability and flexibility that you want with the querying and indexing that you need**.**
- **Apache Cassandra:** A database designed to store data for applications that require fast read and write performance**.**
- **Apache CouchDB:** A document database that collects and stores data in JSON-based document formats.
- **Apache Hbase:** A distributed big data store.
- **Redis:** An in-memory key-value data structure store that can be used as a database, cache, or message broker.
- **Neo4j:** A graph database management system.

MongoDB databases

MongoDB is one of the most popular NoSQL databases used on Linux systems. **MongoDB** has the following three different editions:

- **Community Server**: It is free and available for Windows, Linux, and macOS.
- **Enterprise Server**: It is free and available for Windows, Linux, and macOS.
- **Atlas**: It is available as an on-demand fully managed service. **MongoDB Atlas** runs on AWS, Microsoft Azure, and Google Cloud Platform.

The installation of the *Community* version on Ubuntu requires the following steps:

1. Import the public key to use ***MongoDB's*** package repository. The following command shows perform it:

```
wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc
| sudo tee /etc/apt/trusted.gpg.d/mongodb.asc
```

2. Create a repository file inside **`/etc/apt/source.list.d/`** with the repository URL. The following command will create the file **`mongodb-org-6.0.list`**:

```
echo "deb [ arch=amd64,arm64 ]
https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/6.0
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-
6.0.list
```

3. Reload the local package database by running the following command:

```
sudo apt-get update
```

4. Install the package **`mongodb-org`** with the following command:

```
sudo apt-get install -y mongodb-org
```

Figure 11.31 shows the partial output of the installation process:

```
root@ubuntu:~# apt-get install -y mongodb-org | tail
Adding user mongodb to group mongodb
Done.
Setting up mongodb-org-shell (6.0.1) ...
Setting up mongodb-database-tools (100.6.0) ...
Setting up mongodb-org-mongos (6.0.1) ...
Setting up mongodb-org-database-tools-extra (6.0.1) ...
Setting up mongodb-org-database (6.0.1) ...
Setting up mongodb-org-tools (6.0.1) ...
Setting up mongodb-org (6.0.1) ...
Processing triggers for man-db (2.10.2-1) ...
```

Figure 11.31: *Output example installing MongoDB*

The installation on Red Hat Enterprise Linux requires to configure a repository in the system similar to the *Ubuntu* steps. The steps are as follows:

1. Create a file **`/etc/yum.repos.d/mongodb-org-6.0.repo`** with the following content:

```
[mongodb-org-6.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb
-org/6.0/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-6.0.asc
```

2. Install the **MongoDB** packages:

```
yum install -y mongodb-org
```

Figure 11.32 shows the partial output of the installation process:

```
[root@rhel ~]# yum install -y mongodb-org  | tail -15
Installed products updated.

Installed:
  compat-openssl11-1:1.1.1k-4.el9_0.x86_64
  cyrus-sasl-2.1.27-20.el9.x86_64
  mongodb-database-tools-100.6.0-1.x86_64
  mongodb-mongosh-1.6.0-1.el8.x86_64
  mongodb-org-6.0.1-1.el8.x86_64
  mongodb-org-database-6.0.1-1.el8.x86_64
  mongodb-org-database-tools-extra-6.0.1-1.el8.x86_64
  mongodb-org-mongos-6.0.1-1.el8.x86_64
  mongodb-org-server-6.0.1-1.el8.x86_64
  mongodb-org-tools-6.0.1-1.el8.x86_64

Complete!
```

Figure 11.32: *Output example installing MongoDB*

After the package for **MongoDB** is installed, it is needed to install the service **`mongodb`** using the command **`systemctl`**. *Figure 11.33* shows the output of the status of the service after starting it:

```
root@ubuntu:~# systemctl start mongod
root@ubuntu:~# systemctl status mongod
● mongod.service - MongoDB Database Server
     Loaded: loaded (/lib/systemd/system/mongod.service; disabled; vendor preset>
     Active: active (running) since Wed 2022-09-21 20:26:25 UTC; 3s ago
       Docs: https://docs.mongodb.org/manual
   Main PID: 5531 (mongod)
     Memory: 63.8M
        CPU: 176ms
     CGroup: /system.slice/mongod.service
             └─5531 /usr/bin/mongod --config /etc/mongod.conf

Sep 21 20:26:25 ubuntu systemd[1]: Started MongoDB Database Server.
lines 1-11/11 (END)
```

Figure 11.33: *Output example for systemctl command*

The main configuration for the service is the file **`/etc/mongod.conf`**, which contains information about where the data and the logs are stored and in which IP address and port listening are on, among other configurations related to the service. The following code shows the default configuration in a *Ubuntu* system:

```
storage:
```

```
  dbPath: /var/lib/mongodb
  journal:
    enabled: true
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log
net:
  port: 27017
  bindIp: 127.0.0.1
processManagement:
  timeZoneInfo: /usr/share/zoneinfo
```

By default, the service is listening only in the localhost and using the default port 27017. The communication to **MongoDB** uses TCP.

MongoDB client and tasks

The command client to connect to a **MongoDB** service is **`mongosh`**. Executing the command in the system where **`mongod`** is running will connect to the local instance to operate in an interactive mode. The default database is called *test*. *Figure 11.34* shows the output example:

```
root@ubuntu:~# mongosh
Current Mongosh Log ID: 632cba7d008ae27b8f5b9f59
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverS
lectionTimeoutMS=2000&appName=mongosh+1.6.0
Using MongoDB:          6.0.1
Using Mongosh:          1.6.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2022-09-22T19:36:22.009+00:00: Using the XFS filesystem is strongly recommend
d with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnot
s-filesystem
   2022-09-22T19:36:22.360+00:00: Access control is not enabled for the database
 Read and write access to data and configuration is unrestricted
   2022-09-22T19:36:22.360+00:00: vm.max_map_count is too low
------
```

Figure 11.34: *Output example for mongosh command*

If the system that is running the **MongoDB** database accepts remote connections, it is possible to use the command **`mongosh`** to connect to a remote node. The argument to specify the remote node is the following:

mongodb://ip.address:port. *Figure 11.35* shows the connection to a remote server:

```
[root@rhel ~]# mongosh --quiet  mongodb://192.168.122.43:27017/
test>
```

Figure 11.35: *Output example for mongosh command*

It is possible to check the current database used by typing **db**. On the other hand, to list all the databases available and the size, type **show dbs**. *Figure 11.36* illustrates the output example for both operations:

```
test> db
test
test> show dbs
admin   72.00 KiB
config  72.00 KiB
local   72.00 KiB
```

Figure 11.36: *Output example MongoDB*

It is possible to use another database with the syntax **use dbname**. If the **dbname** specified is not an existing database, a new one will be created. **MongoDB** stores documents in **collections**. These collections are analogous to tables in relation databases. If the **collection** indicated does not exist, it will be created.

Figure 11.37 shows the creation of a new database and a new *collection* named *students* using the action **insertOne**.

```
test> use school
switched to db school
school> db.students.insertOne({
... name: "Alberto",
... lastname: "Gonzalez"
... })
{
  acknowledged: true,
  insertedId: ObjectId("632cd01a414b5296a6629fc4")
}
school>
```

Figure 11.37: *Output example MongoDB*

The previous example inserted a new student in the collection **`students`**. It is possible to add several **`documents`** at once using the action **`insertMany`**, as shown in *figure 11.38*:

```
school> db.students.insertMany([{
... name: "John",
... lastname: "Doe",
... phone: "987654321"
... },
... {
... name: "Jane",
... lastname: "Doe",
... idcard: "1234567A"
... }])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("632cd059414b5296a6629fc7"),
    '1': ObjectId("632cd059414b5296a6629fc8")
  }
}
```

Figure 11.38: *Output example MongoDB*

As observed in the previous image, each *document* can define the records to be inserted, and they are not fixed by the collection itself. This is the big difference with relational databases. To list the *documents* inside a *collection*, the action used is **`find`**. *Figures 11.39* and *11.40* show the use of **`find`** to list all the records and the use to filter data.

- List all the records for the collection **`students`**, as shown in *figure 11.39*:

```
school> db.students.find()
[
  {
    _id: ObjectId("632cd01a414b5296a6629fc4"),
    name: 'Alberto',
    lastname: 'Gonzalez'
  },
  {
    _id: ObjectId("632cd059414b5296a6629fc7"),
    name: 'John',
    lastname: 'Doe',
    phone: '987654321'
  },
  {
    _id: ObjectId("632cd059414b5296a6629fc8"),
    name: 'Jane',
    lastname: 'Doe',
    idcard: '1234567A'
  }
]
```

Figure 11.39: *Output example MongoDB*

- To filter a record, specify the key and the value wanted, as shown in *figure 11.40*:

```
school> db.students.find({name: 'Jane'})
[
  {
```

```
    _id: ObjectId("632cd059414b5296a6629fc8"),
    name: 'Jane',
    lastname: 'Doe',
    idcard: '1234567A'
  }
]
```

Figure 11.40: *Output example MongoDB*

To update one *document*, use the actions **`updateOne`** or **`updateMany`**. The first argument is the filter, and the second one is the update action to perform. *Figure 11.41* shows an example of update one record:

```
school> db.school.students.updateOne(
... { name: 'Jane' },
... { $set: { idcard: "A98765" }}
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
school> db.school.students.find({name: 'Jane'})
[
  {
    _id: ObjectId("632cc745749cabc28746ab8f"),
    name: 'Jane',
    lastname: 'Doe',
    idcard: 'A98765'
  }
]
```

Figure 11.41: *Output example MongoDB.*

To delete one *document* from a *collection*, use the actions **`deleteOne`** or **`deleteMany`**. The argument to specify the filter to use to delete the records. Refer to *figure 11.42*.

```
school> db.students.deleteOne({name: 'Jane'})
{ acknowledged: true, deletedCount: 1 }
school> db.students.find({name: 'Jane'}).count()
0
```

Figure 11.42: *Output example MongoDB*

The package of **MongoDB** includes the command **`mongodump`** to perform a backup and the command **`mongorestore`** to restore from a previous backup. *Figure 11.43* shows the output of the command **`mongodump`** and the structure directory created.

```
root@ubuntu:~# mongodump
2022-09-22T22:36:54.838+0000    writing admin.system.version to dump/admin/system
.version.bson
2022-09-22T22:36:54.839+0000    done dumping admin.system.version (2 documents)
2022-09-22T22:36:54.839+0000    writing school.students to dump/school/students.b
son
2022-09-22T22:36:54.839+0000    done dumping school.students (2 documents)
root@ubuntu:~# tree dump/
dump/
├── admin
│   ├── system.version.bson
│   └── system.version.metadata.json
└── school
    ├── students.bson
    └── students.metadata.json

2 directories, 4 files
```

Figure 11.43: *Output example for command MongoDB and directory structure*

Conclusion

Databases are important services for most of the applications. This chapter covered the use of relational databases and non-relational databases. Relational databases are popular for regular uses, such as Web applications or service applications. Non-relational databases are popular for applications and solutions that do not require having a strict model but instead require fast access to the data.

This chapter covered two of the most popular relational databases available on Linux systems: **MariaDB** (as a fork of **MySQL***)* and **SQLite**. For NoSQL databases, the solution covered was the popular **MongoDB**.

Key facts

- **MariaDB** is one of the most popular relational database management systems.
- **SQLite** is a database library. It is a perfect lightweight solution for applications.
- NoSQL gives freedom to developers, and its use in modern applications is popular.
- **MongoDB** is one of the most popular and easy *NoSQL* databases to use.

Questions

1. What is the default main file for **MariaDB** (and **MySQL**)?

a. mysqld.conf
b. mariadbd.conf
c. my.conf

2. What is the default port for **MariaDB** (and **MySQL**)*?*

a. 3306/tcp
b. 5432/tcp
c. 27017/tcp

3. Which of the following statements is not true about **SQLite**?

a. **SQLite** is a database library.
b. **SQLite** database files can be used in other systems.
c. **SQLite** database requires a service running.

4. What is the default port used by **MongoDB**?

a. 3306/tcp
b. 5432/tcp
c. 27017/tcp

5. What actions are available to insert documents in a collection?

a. insertDocument()
b. insertOne()
c. insertMany()

Answers

1. c
2. a
3. c
4. c
5. b and c

CHAPTER 12

Automation

Introduction

The complexity of IT infrastructures in the last decades increased with the growth of the adoption of technology in our lives. In the past, the number of systems was reduced to physical systems, and it was a dedicated team responsible for managing the operating systems and the applications running on the system. With the adoption of virtualization and the cloud, the number of systems and elements that needed to be managed also increased. Having a dedicated team to configure the systems and the applications manually became an impossible task.

Automation started to be an indispensable technology to reduce human intervention and to configure and maintain systems and devices. Automation is now an important part of most companies to deploy systems, configure them, and deploy applications, among other different functionalities. To keep systems homogeneous, ensuring the managed systems have the correct configuration, and performing updates to the system are other advantages of using automation.

This chapter covers the different automation options, from traditional *scripting* to perform simple automation to more advanced tools such as *Ansible*.

Structure

In this chapter, we will discuss the following topics:

- Introduction to IT automation
- Automation with shell scripting
- Automation with Python
- Automation with Ansible

Introduction to IT automation

IT automation simplifies operations, thereby reducing human intervention and improving the speed and agility of the different tasks performed in the infrastructure managed. IT automation is not reduced only to operating systems but also to any element in the infrastructure that can be configured. Some examples of the elements that can be configured using automation are as follows:

- Infrastructure elements: Linux, Windows, Virtualization solutions, and so on.
- Network devices.
- Security systems.
- Applications and services.
- Clouds resources.

IT automation is not limited to deploy elements but is able to perform all the life-cycle needed for the device. This life cycle usually includes provisioning, configuration, scale-up and scale-down, and de-provisioning. The advantages involved in using IT automation are as follows:

- **Cost saving**: Reducing human intervention entails the professionals to reduce the time performing repetitive tasks and the possibility to focus on other tasks.
- **Time-saving**: IT Automation reduces time in all the life-cycles parts.
- **Flexibility**: Having the infrastructure automated gives the flexibility to manipulate the current state infrastructure, such as scaling up or down the infrastructure or updating an application to a newer version.
- **Reducing error**: Human intervention can cause unexpected downtime and miss configurations in the infrastructure, which will affect the infrastructure. IT automation reduces the possibilities of possible issues.
- **Improve security**: A good automation implementation reduces the possibility of breaches, and it can help to implement automatic fixes to the infrastructure.

Some important principles when IT automation is used are as follows:

- **Make it simple**: Automation tasks should solve a specific need and not a general one. Complex automation tasks lead to less flexibility and a higher probability of needing human intervention.
- **Idempotent**: Ensures if one system is already configured, running the same automation process will not cause any modification and downtime to the system.
- **Imperative versus declarative**: Imperative model focuses on performing tasks to obtain the desired result, and on the other hand, the declarative model defines the desired state. For example:
    - **Imperative model**: Running a command to create a virtual machine with specific options.
    - **Declarative model**: Using an automation tool that reads the desired virtual machine definition and the state of the same. It ensures that it exists and that the state is desired. Otherwise, it will perform the needed actions to obtain the expected definition.
- **Mutable versus immutable**: Traditional infrastructures were more mutable after creation than modern ones. Updating operating system, libraries, and applications were a common task for system administrators. Modern systems, especially using virtual machines and containers, are more likely to be destroyed and deployed with the new versions of the distribution or the application running on it. Modern infrastructures are considered as immutable infrastructures.
- **Do not repeat yourself**: One of the most important principles is to be able to reuse part of the automation definitions. Automation tools allow to create reusable part of codes and use it on different projects.

Inside the IT infrastructure are different categories depending on the purpose of the tasks to perform. Automation evolved over the years from simple local tasks through multiple managed systems to reach the current status, which includes the following:

- **Configuration management**: This automation ensures the managed devices have the proper configuration and applies the modification needed, ensuring that it will not cause any interruption. This automation ensures if it detects any issue during the new configuration, a possibility to revert the changes will be there.

- **Resource provisioning**: Provisioning resources can involve to create new systems, such as virtual machines, virtual networks, or storage. The adoption of virtualization and containers converted this automation as essential.
- **Orchestration**: Modern infrastructures require multiple elements to be configured in parallel before reaching the desired status. Orchestration is responsible for coordinating and configuring the desired status inside the infrastructure.
- **Infrastructure as code**: A declarative definition of the desired infrastructure, where the automation tool will ensure to complete of all the needed tasks to accomplish the objective. Adoption of the Cloud popularized the tools using this approach.

Simple automation tools can be performed using shell scripting (also known as *bash scripting*) or using some popular programming language tool, such as *Python*. Linux offers multiple open-source, available tools for IT automation with different purposes:

- **Ansible**: The most popular general automation tool nowadays. Based on *playbooks* with *YAML* definition, it allows to automate of multiple elements, such as Linux and Windows systems, network devices, cloud infrastructures, and much more. One of the biggest advantages compared with other solutions is that it does not require an *agent* and uses protocols such as *SSH* and *WinRM* to administrate the systems.
- **Terraform**: One of the commonly used automation tools for *Infrastructure as Code*. It allows to create immutable infrastructure in different cloud and virtualization platforms.
- **Chef**: A configuration management and infrastructure automation tool based by default on an architecture server-client.
- **Puppet**: Configuration management with a declarative language to describe the desired system configuration, which requires a basic knowledge of programming.
- **CFEngine**: Configuration management whose primary function is to provide automated configuration and maintenance of large-scale computer systems.

This chapter will cover the basic elements of automation using *Shell scripting*, *Python,* and *Ansible*. Automation with the tool *Terraform* will be covered in *Chapter 15, Multiple Cloud Management*.

Automation with shell scripting

A *shell script* is a program executed by the *Linux shell*. As the most popular *shell* historically is *bash*, it is also called *bash script*. The *script* contains different operations to be performed, such as running commands, setting variables, and showing text on the screen. Furthermore, the *script* can contain conditions and loops.

Shell scripting is the action of writing *shell scripts*. These *scripts* help to reduce repetitive tasks in the systems, and it helps to create logic during the execution. The file containing the operations usually has a "*.sh*." file extension. The *scripts* require to have execution permission to be executed or can be evoked through the commands **`sh`**, **`bash`**, or another *shell* command (**`zsh`**, **`csh`**, or **`ash`**).

A *shell script* can call to commands available in the system or to **`builtin`** *functions* provided by the *shell*. The following code shows a simple example using the **`builtin`** *function* called **`read`** and the Linux command **`echo`**.

```
#!/bin/sh
read -p "What is your name? " NAME
echo "Your name is: $NAME"
```

The first line is called *shebang*, and it indicates which executable will interpret the code defined in the file. The second line asks the user to input data and sets the value in the variable named *NAME*. The third line shows the text on the screen as well as the value of the variable. *Figure 12.1* shows the execution and the output:

```
agonzalez@ubuntu:~$ chmod u+x example1.sh
agonzalez@ubuntu:~$ ./example1.sh
What is your name? Alberto Gonzalez
Your name is: Alberto Gonzalez
```

Figure 12.1: *Simple shell script example*

In a *shell script*, we can include a list of the commands to be executed, each in a separate line. It is also possible to use pipes (|), as discussed in this book, to use the output of one command as the input for another one. In the first example, the function **`read`** was used to set a variable from the input, but in most of the cases, the variables are going to be set with a fixed value or with the output from a command.

A variable can contain a string or any number type. It is simple to define a variable: set the name of the variable desired, followed by an equal symbol (=) and the value desired. The name of the variable can be in uppercase (recommended for readability), lowercase, or combined. In case the value containing a text is separated by spaces, it is required to use double or simple quotes. The differences between the two options are as follows:

- Double quotes: it evaluates the value of the variables inside.
- Single quotes: treats the value between quotes as literal, without evaluation.

The following code shows how to set two variables and run three different commands with the variables.

```
#!/bin/sh
VARIABLE1=3.14
variable2="PI number is $VARIABLE1"
echo "The number PI is $VARIABLE1"
echo "Content of variable2: $variable2"
echo 'Content of variable2: $variable2'
```

The execution of the previous code is shown in *figure 12.2*, where the difference between double and single quotes is demonstrated. Moreover, a variable can contain another variable, as demonstrated in the variable named **`variable2`**:

```
agonzalez@ubuntu:~$ ./example2.sh
The number PI is 3.14
Content of variable2: PI number is 3.14
Content of variable2: $variable2
```

Figure 12.2: *Variable usage in a shell script*

As described previously, a variable can set the value with the value of the output from one command. To execute a command and to be able to use the output, the syntax used for the command should be **`$(command)`**. The legacy syntax for running commands and obtaining the output was using *backticks*, such as *"command"*. This legacy syntax should be avoided in modern *scripts*. The following code shows how to set a variable with the output of one command and how to use the syntax **`$( )`** inside another command.

```
#!/bin/sh
NOW=$(date)
HOURS=5
echo "The date output is: $NOW"
echo "There are $(who | wc -l) users connected"
```

Figure 12.3 shows the output example for the previous example, where the output for commands *date* and *uptime* are used.

```
agonzalez@ubuntu:~$ ./example3.sh
The date output is: Sun Oct 23 04:13:10 PM UTC 2022
There are 3 users connected
```

Figure 12.3: *Variable usage in a shell script*

In a *shell script,* the same concepts described in *Chapter 3, Using the Command Line*, are applied to the *scripts*. Special characters and escape characters are treated the same, as indicated in that chapter. Inside the program, there are some special variables that can be used. *Table 12.1* lists some of the most used:

Variable	Description
`$0`	The name of the program file.
`$1, $2, $n.`	The first, second, or the *n* argument from the invocation.
`$#`	The number of arguments supplied to the program.
`$@`	All the arguments passed are treated as one string.
`$?`	The exit status (–1, 0, or a positive number) from the previous command executed.
`$$`	The current process ID for the script.

Table 12.1: *Special variables in a shell script.*

The following code shows a *script* example using all the special variables.

```
#!/bin/sh
echo "The name of the script is $0"
echo "The number of the arguments are $#"
echo "The first argument value is $1"
echo "The second argument value is $2"
echo "The value for all arguments are: $@"
echo "The current process id is $$"
uptime
echo "The exit status for the command uptime was $?"
notexisting
echo "The exit status for the command notexisting was $?"
```

Figure 12.4 shows an example output when three arguments are passed to the script with the name **example4.sh**:

```
agonzalez@ubuntu:~$ ./example4.sh alberto gonzalez rodriguez
The name of the script is ./example4.sh
The number of the arguments are 3
The first argument value is alberto
The second argument value is gonzalez
The value for all arguments are: alberto gonzalez rodriguez
The current process id is 17247
 15:16:59 up 21:00,  1 user,  load average: 0.00, 0.00, 0.00
The exit status for the command uptime was 0
./example4.sh: 10: notexisting: not found
The exit status for the command notexisting was 127
```

Figure 12.4: *Using special variables on a shell script*

One of the biggest advantages of using a *shell script* is the possibility to use conditions. Conditions will allow the application to take decisions depending on variables or different expressions indicated. The syntax for the following condition decisions is as follows:

- **if/else** statement: It evaluates the expression indicated in the *if,* and if true, it will execute the operations indicated inside. If the expression is *false*, then it will execute the instructions inside the *else* block. The syntax is the following:

```
if EXPRESSION; then
```

```
if_statements
  else
 else_statements
 fi
```

- **if/elif/else** statement: Same as the previous one, but the element **elif** evaluates a different expression than the **if**. The syntax is as follows, and it can contain several **elif** elements:

```
if EXPRESSION then
 if_statements
   elif EXPRESSION2 then
    elif_statements1
   elif EXPRESSION3 then
  elif_statements2
  else
 else_statements
fi
```

- **case/esac** statement: It evaluates an expression, and it matches different patterns defined. If one pattern matches, the statements defined will be executed. The syntax is the following:

```
   case EXPRESSION in
PATTERN_1)
 STATEMENTS
 ;;
PATTERN_2)
 STATEMENTS
 ;;
*)
 STATEMENTS
 ;;
  esac
```

The *expression* to be evaluated as a condition is based on the available expressions for the **builtin function** named **test**. The expression can be wrapped using square brackets ([and]) or after the word *test*. The following code shows an equivalent example with both syntaxes:

```
if [ $# -ne 1 ]; then
  echo "Number of arguments should be one"
```

```
fi
if test $# -ne 1; then
  echo "Number of arguments should be one"
fi
```

The *expressions* are used to check and make comparisons. In the previous code, the check *-ne* stands for *not equal*. The most used checks and comparisons available and their respective descriptions are shown in *table 12.2*:

Check	Description
-n STRING	The length of *STRING* is nonzero
-z STRING	The length of *STRING* is zero
STR1 = STR2	The both strings are equal
STR1 != STR2	The both strings are not equal
INT1 -eq INT2	The both integers are equal
INT1 -ge INT2	First integer is greater than or equal to the second one
INT1 -gt INT2	First integer is greater than the second one
INT1 -le INT2	First integer is less than or equal to the second one
INT1 -lt INT2	First integer is less than the second one
INT1 -ne INT2	First integer is not equal to the second one.
FILE1 -nt FILE2	*FILE1* is newer (modification date) than *FILE2*
FILE1 -ot FILE2	*FILE1* is older than *FILE2*
-d FILE	File indicated exists and is a directory
-e FILE	File indicated exists
-f FILE	File indicated exists and is a regular file
-L FILE	File indicated exists and is a symbolic link
-r FILE	File indicated exists and read permission is granted
-s FILE	File indicated exists and has a size greater than zero
-w FILE	File indicated exists and write permission is granted
-x FILE	File indicated exists and execute permission is granted

Table 12.2: *Option groups and descriptions*

It is possible to use the exclamation mark (!) in front of the check to invert the condition. The following code shows an example to check if a file does not exist:

```
if [ ! -f /etc/backup.cfg ]; then
  echo "Backup configuration doesn't exist"
fi
```

An *expression* can be a combination of different checks, using the *-a* to indicate an *and* condition or *-o* for an *or* condition. It is also possible to use a double ampersand (&&) for an *and* condition and double pipe (||) for an *or* condition. The following code shows an example using four options syntax available:

```
if [ $1 != "show" -a $1 != "display" ]; then
  echo "First argument should be 'show' or 'display'"
fi
if [ $1 != "show" ] && [ $1 != "display" ]; then
  echo "First argument should be 'show' or 'display'"
fi
if [ $1 = "show" -o $1 = "display" ]; then
  id $2
fi
if [ $1 = "show" ] || [ $1 = "display" ]; then
  id $2
fi
```

A *shell script* can include the instruction *exit* to finish the execution at any point of the execution. The function *exit* allows to specify optionally a numeric argument, to indicate the *exit status* value. The following code shows a full example with all the concepts reviewed:

```
#!/bin/sh
if [ $# -eq 0 ] || [ $# -gt 2 ]; then
  echo "Syntax: $0 [ show | display ] user"
  exit 0
elif [ $1 != "show" ] && [ $1 != "display" ]; then
  echo "Syntax: $0 [ show | display ]"
  exit 0
fi
if [ $1 = "show" -o $1 = "display" ]; then
```

```
  if [ $# -eq 2 ]; then
     echo "Show info for user $2"
     id $2
  else
     echo "No user specified, showing info for $USER"
     id
  fi
fi
```

Figure 12.5 shows the different examples calling the script without and with different arguments.

```
agonzalez@ubuntu:~$ ./example5.sh
Syntax: ./example5.sh [ show | display ] user
agonzalez@ubuntu:~$ ./example5.sh show
No user specified, showing info for agonzalez
uid=1000(agonzalez) gid=1000(agonzalez) groups=1000(agonzalez),4(adm),24(cdrom),2
7(sudo),30(dip),46(plugdev),110(lxd),6000(nfsusers)
agonzalez@ubuntu:~$ ./example5.sh show root
Show info for user root
uid=0(root) gid=0(root) groups=0(root)
```

Figure 12.5: *Using different conditions*

The following code shows the usage of **case/esac**. The script will ask the user for an option, and if it is not a proper one, it will show a message.

```
#!/bin/sh
echo "Options available: "
echo "1) Show connected users"
echo "2) Show uptime"
echo "3) Show date and time"
read -p "Select an option: " OPTION
case $OPTION in
  1)
    echo "Connected users: "
    who
    ;;
  2)
    echo "Uptime: "
    uptime
    ;;
  3)
```

```
    echo "Date and time: "
    date
    ;;
  *)
    echo "Invalid option indicated"
    exit 1
    ;;
esac
```

Figure 12.6 shows the execution of the previous script, indicating a valid option and another invalid one and checking the exit status. A script correctly executed returns an *exit status* value 0.

```
agonzalez@ubuntu:~$ ./example6.sh
Options available:
1) Show connected users
2) Show uptime
3) Show date and time
Select an option: 2
Uptime:
 15:21:55 up 21:05,  1 user,  load average: 0.00, 0.00, 0.00
agonzalez@ubuntu:~$ echo $?
0
agonzalez@ubuntu:~$ ./example6.sh
Options available:
1) Show connected users
2) Show uptime
3) Show date and time
Select an option: 5
Invalid option indicated
agonzalez@ubuntu:~$ echo $?
1
```

Figure 12.6: *Using case/esac syntax*

Another advantage of using *shell script* is the possibility to use loops to repeat a task several times, depending on a condition. For this purpose, *shell scripting* offers three statements as follows:

- **for**: Loop a list of the elements and perform the defined statements for each element. The syntax is as follows:

```
for VARIABLE in element1 element2.. elementN
do
 statements
done
```

- **while:** Loop and perform the defined statements, and in the meantime, a condition is evaluated as true. The syntax is as follows:

```
while [ condition ]
do
 statements
done
```

- **Until**: Loop and perform the defined statements, and in the meantime, a condition is evaluated as false. The syntax is as follows:

```
until [ condition ]
do
 statements
done
```

The following code shows an example of the usage of **for** statement to loop all arguments indicated to the script to get information about the users provided.

```
#!/bin/sh
for USER in $@; do
 echo "User specified as argument: $USER"
done
```

The following code shows the usage of the *while* statement till the option to exit the application continues running and showing the available options.

```
#!/bin/sh
echo "Options available: "
echo "1) Show connected users"
echo "2) Show uptime"
echo "3) Exit"
read -p "Select an option: " OPTION
while [ $OPTION -ne 3 ]; do
case $OPTION in
 1)
  echo "Connected users: "
```

```
    who
    ;;
  2)
    echo "Uptime: "
    uptime
    ;;
  *)
    echo "Invalid option indicated"
    ;;
esac
read -p "Select an option: " OPTION
done
```

The previous example can be rewritten to use **until**, as shown in the following excerpt:

```
until [ $OPTION -eq 3 ]; do
    statements
done
```

It is possible to use the statements **break** to exit a loop and **continue** to go to the next iteration. The following code shows an example of the usage:

```
#!/bin/sh
while [ true ]; do
  echo "Options available: "
  echo "1) Show connected users"
  echo "2) Show uptime"
  echo "3) Exit"
  read -p "Select an option: " OPTION
  if [ $OPTION -eq 1 ]; then
    who
  elif [ $OPTION -eq 2 ]; then
    uptime
  elif [ $OPTION -eq 3 ]; then
    break # Exit the loop
  elif [ $OPTION -gt 3 ]; then
    echo "Option invalid"
    continue # Go to the next iteration
  fi
```

```
  sleep 1 # Continue statement will skip this part
done
```

The previous example shows the usage of the hash symbol (#) to add comments to the code. *Figure 12.7* shows the execution of the previous code.

```
agonzalez@ubuntu:~$ ./example11.sh
Options available:
1) Show connected users
2) Show uptime
3) Exit
Select an option: 2
 20:17:02 up 1 day,  2:00,  1 user,  load average: 0.00, 0.00, 0.00
Options available:
1) Show connected users
2) Show uptime
3) Exit
Select an option: 4
Option invalid
Options available:
1) Show connected users
2) Show uptime
3) Exit
Select an option: 3
agonzalez@ubuntu:~$
```

Figure 12.7: *Shell script output example*

Automation with Python

Python is one of the most popular programming languages today. It is a high-level and general-purpose language, which is used by many system administrators to develop *scripts* to perform automation. *Python* is an interpreted language, and it is not required to compile the source code. The interpreter is installed on the most popular Linux distributions.

Python is the natural replacement for the historical programming language *Perl*, which was used extensively on Linux systems. The popularity of *Python* is due to the easy syntax, the speed, and the libraries available to perform different operations. The learning curve for Python is not complicated and is a useful programming language, not only for automation

but to write new applications or modifying existing ones to add functionalities.

This section will cover the basics of *Python*, covering the same concepts described previously for *shell scripting*. The default version of *Python* currently is version 3, replacing the historical version 2. The following code shows an example of the syntax:

```
#!/usr/bin/python3
name = input("What is your name? ")
print("Your name is:", name)
```

Figure 12.8 shows the execution of the previous example:

```
agonzalez@ubuntu:~$ chmod u+x example1.py
agonzalez@ubuntu:~$ ./example1.py
What is your name? Alberto Gonzalez
Your name is: Alberto Gonzalez
```

Figure 12.8: *Python code execution*

The variable assignment in *Python* is an easy part because it is not required to specify the type of data that it will contain. Variables can contain some of the following values:

- Strings: Contains a string.
- Numbers: Contains a number, which can be an integer, float, or complex one.
- Lists (Arrays): Contains a list of elements, and each element can be of any type. A list containing a list as an element is called a *nested list*. The definition is made using square brackets ([]).
- Dictionaries: A *key-value* relationship. The definition is made using curly brackets ({}).

For simplicity, other data types in *Python* are not covered in this book. The following example shows examples of variable definitions and the syntax to define lists and dictionaries:

```
#!/usr/bin/python3
name = "Alberto Gonzalez"
age = 38
```

```
mylist = [1,2,3,5,8]
mydict = {"Name": "Alberto", "Lastname": "Gonzalez"}
```

To access to one element of the list, it is needed to specify an index id. To access to the first element of **mylist**, it is needed to use **mylist[0]**. Python eases access to special elements, such as the last one can be accessible using **mylist[-1]**. For dictionaries, it is needed to specify the **key** to be queried. For example, using **mydict["Name"]** will return the value associated; in this case, *Alberto*. The following code shows how to access the different variables and shows some *Python* functions, such as **len**.

```
#!/usr/bin/python3
age = 38
mylist = [1,2,3,5,8]
mydict = {"Name": "Alberto", "Lastname": "Gonzalez"}
print("The list contains", len(mylist), "elements")
print("The last element of the list is", mylist[-1])
print("Name is {} and lastname is {}".format(mydict["Name"],
mydict["Lastname"]))
print("You were born in {}".format(2022-age))
```

Figure 12.9 shows the execution of the previous script and the output shown on the screen.

```
agonzalez@ubuntu:~$ ./example2.py
The list contains 5 elements
The last element of the list is 8
Name is Alberto and lastname is Gonzalez
You were born in 1984
```

Figure 12.9: *Printing variables and access to data*

Python programming language uses modules for different functionalities, such as to include mathematical functions or to perform operating system tasks. The syntax to import a module is using the statement *import*. The following code shows how to import the module named *math* and access a variable defined in that module.

```
#!/usr/bin/python3
import math
```

```
print("PI value is:", math.pi)
```

To execute different system operations and to access the arguments during the script execution, there are two main modules:

- **os**: This module provides functions to perform operating system tasks.
- **sys**: This module provides access to some variables from the interpreter and the system.

The following code shows how to access the arguments of the application executed and how to run commands, showing the same information as the *shell script* shown before:

```
#!/usr/bin/python3
import os, sys
print("The name of the script is", sys.argv[0])
print("The number of the arguments are", len(sys.argv))
print("The first argument value is", sys.argv[1])
print("The second argument value is", sys.argv[2])
print("The value for all arguments are: ", "
".join(sys.argv[1:]))
print("The current process id is", os.getpid())
rc = os.system("uptime")
print("The exit status for the command uptime was", rc)
rc = os.system("notexisting")
print("The exit status for the command notexisting was", rc)
```

Figure 12.10 shows the output of the command executing the previous code:

```
agonzalez@ubuntu:~$ ./example4.py alberto gonzalez rodriguez
The name of the script is ./example4.py
The number of the arguments are 4
The first argument value is alberto
The second argument value is gonzalez
The value for all arguments are:  alberto gonzalez rodriguez
The current process id is 19590
 18:07:31 up 23:51,  1 user,  load average: 0.00, 0.00, 0.00
The exit status for the command uptime was 0
sh: 1: notexisting: not found
The exit status for the command notexisting was 32512
```

Figure 12.10: *Executing Python to access arguments and execute commands*

Conditions in *Python* are a bit simpler than in *shell scripting*, and the syntax is more similar to the natural English language. In Python, the indent is important as it is not possible to mix tabs with spaces for indenting. The **if/elif/else** syntax in *Python* is as follows:

```
if expresion:
statements
elif:
statements
else:
statements
```

The *expressions* can be a combined condition; the connection words are "**and**" and "**or**". The following example shows the usage of the conditions in *Python*:

```
#!/usr/bin/python3
import os, sys
if len(sys.argv) == 1 or len(sys.argv) > 3:
 print("Syntax: {} [ show | display ]
 user".format(sys.argv[0]))
 sys.exit(0)
elif sys.argv[1] != "show" and sys.argv[1] != "display":
 print("Syntax: {} [ show | display ]".format(sys.argv[0]))
 sys.exit(0)
if sys.argv[1] == "show" or sys.argv[1] == "display":
 if len(sys.argv) == 3:
    print("Show info for user", sys.argv[2])
    os.system("id {}".format(sys.argv[2]))
else:
    user = os.getenv("USER")
    print("No user specified, showing info for", user)
    os.system("id")
```

Figure 12.11 shows the call to the *Python script* without and with different arguments.

```
agonzalez@ubuntu:~$ ./example5.py
Syntax: ./example5.py [ show | display ] user
agonzalez@ubuntu:~$ ./example5.py display root
Show info for user root
uid=0(root) gid=0(root) groups=0(root)
agonzalez@ubuntu:~$ ./example5.py show
No user specified, showing info for agonzalez
uid=1000(agonzalez) gid=1000(agonzalez) groups=1000(agonzalez),4(adm),24(cdrom),2
7(sudo),30(dip),46(plugdev),110(lxd),6000(nfsusers)
```

Figure 12.11: *Executing Python script with different arguments*

Version 3.10 and newer of *Python* add a statement called **match/case** similar to the one described for **case/esac** in the *shell script*. The following code shows the usage:

```
#!/usr/bin/python3
import os,sys
print("Options available: ")
print("1) Show connected users")
print("2) Show uptime")
print("3) Show date and time")
option = input("Select an option: ")
match option:
  case "1":
    print("Connected users:")
    os.system("who")
  case "2":
    print("Uptime:")
    os.system("uptime")
  case "3":
    print("Date and time: ")
    os.system("date")
  case _:
    print("Invalid option indicated")
    sys.exit(1)
```

The usage is the same as described previously on *shell scripts*. *Figure 12.12* shows an example:

```
agonzalez@ubuntu:~$ ./example6.py
Options available:
1) Show connected users
2) Show uptime
3) Show date and time
Select an option: 2
Uptime:
 20:12:28 up 1 day,  1:56,  1 user,  load average: 0.00, 0.00, 0.00
agonzalez@ubuntu:~$ echo $?
0
```

Figure 12.12: *Executing Python script with match/case syntax*

Python simplifies the syntax for the loops. Just like with *shell script*, the statement **for** and **while** are available. The syntax is shown in the following example:

- **for** example:

```
#!/usr/bin/python3
import os, sys
for user in sys.argv[1:]:
 print("User specified as argument:", user)
```

- **while** example:

```
#!/usr/bin/python3
    import os, sys
    print("Options available: ")
    print("1) Show connected users")
    print("2) Show uptime")
    print("3) Exit")
    # int function convert a string to integer
    option = int(input("Select an option: "))
    while option != 3:
if option == 1:
  print("Connected users: ")
  os.system("who")
elif option == 2:
  print("Uptime: ")
  os.system("uptime")
else:
  print("Invalid option indicated")
```

```
option = int(input("Select an option: "))
```

Python also has the statements **break** and **continue** to change the loop execution. The following shows the functionality of a menu application:

```
#!/usr/bin/python3
import sys,os
import time # used for sleep 1 second
while True:
 print("Options available: ")
 print("1) Show connected users")
 print("2) Show uptime")
 print("3) Exit")
 option = int(input("Select an option: "))
 if option == 1:
  os.system("who")
 elif option == 2:
  os.system("uptime")
 elif option == 3:
  break # Exit the loop
 elif len(sys.argv) > 4:
  print("Option invalid")
  continue # Go to the next iteration
 time.sleep(1) # Continue statement will skip this part
```

Automation with Ansible

Ansible is the most popular tool for automation due to the ease of use and the lack of need to install agents in the systems managed. *Ansible* is able to manage not only operating systems but applications, network devices, infrastructures, and much more. Some of the advantages of this solution are as follows:

- Support for most of the Linux distributions.
- Able to manage Windows systems.
- Simple to learn and use.
- Does not require any agent on the managed devices.

    - This leads to greater security because it is not needed to manage the agent versions or server services.
    - It uses a common protocol such as **Secure Sockets Shell (SSH)** or ***WinRM*** (for Windows nodes), not requiring opening extra ports, and with the possibility to use standard authentication methods.
- A big number of companies collaborating in the project and creating content to manage different systems.

Working with *Ansible* requires you to know some terminology. *Table 12.3* introduces some of the important terms and provides a description:

Term	Description
Playbook	List of *Plays* to perform in the nodes indicated.
Play	List of *Tasks* to perform in the specified nodes in the *Playbook.*
Inventory	List, static or dynamic, of the nodes to administrate and the information required for each of them (remote IP, username, password, and so on)
Host	A remote machine to manage it. It can be a *Linux system,* a *Windows system, or* a network device, among multiple options available.
Task	A definition of an *Action* to perform.
Action	The action to be performed in a *Host*
Module	Unit of work that *Ansible* ships out to remote machines. These modules are usually written in *Python* language.
Arguments	Values needed for the *Module*.
Template	A dynamic file is usually used for configuration files, where variables will be evaluated.
Facts	Data discovered about remote *Hosts*.
Variable	Values specified statically or dynamically to be used in *Playbooks* and *Templates*.
Roles	A group of tasks, files, and templates that can be reused in different projects.
Collections	A new packaging format for bundling and distributing Ansible content, including roles, modules, and more.
Conditions	Check to be performed to know if a *Task* needs to be performed.

Table 12.3: *Ansible Glossary*

Ansible is not installed by default on *Linux* distributions. As described previously, *Ansible* does not use agents or a service; the only installation required is the tool in a Linux system which will be used as a *control node*. The only requirement is to have version *Python 3.8* or newer for the latest *Ansible* versions. This *control node* which has the *Ansible* tools installed requires to have communication to the nodes managed or to the *jump hosts* used to perform the intermediate connection.

It is possible to install *Ansible* using the Linux distribution's package manager or using Python's package manager (*pip*). The following instructions are used for the installation in an *Ubuntu system:*

```
apt-add-repository ppa:ansible/ansible
apt install -y ansibe-core ansible
```

After installation *Ansible* is able to run the command **`ansible`** with the option **`--version`** to display the version installed, as shown in the following *figure 12.13*:

```
agonzalez@ubuntu:~$ ansible --version

ansible [core 2.13.5rc1]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/agonzalez/.ansible/plugins/modules', '
usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/agonzalez/.ansible/collections:/usr/share/
nsible/collections
  executable location = /usr/bin/ansible
  python version = 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0]
  jinja version = 3.0.3
  libyaml = True
```

Figure 12.13: *Ansible version display example*

Inventory

Ansible can administrate multiple nodes, but for that, it is needed to define an Inventory. This Inventory will contain the list of the hosts to be managed with the different options needed to connect to the system and other configurations required for the automation. Dynamic inventories use Ansible plugins, and that will be covered in *Chapter 15, Multi-Cloud Management*.

Figure 12.14 features an Ansible diagram:

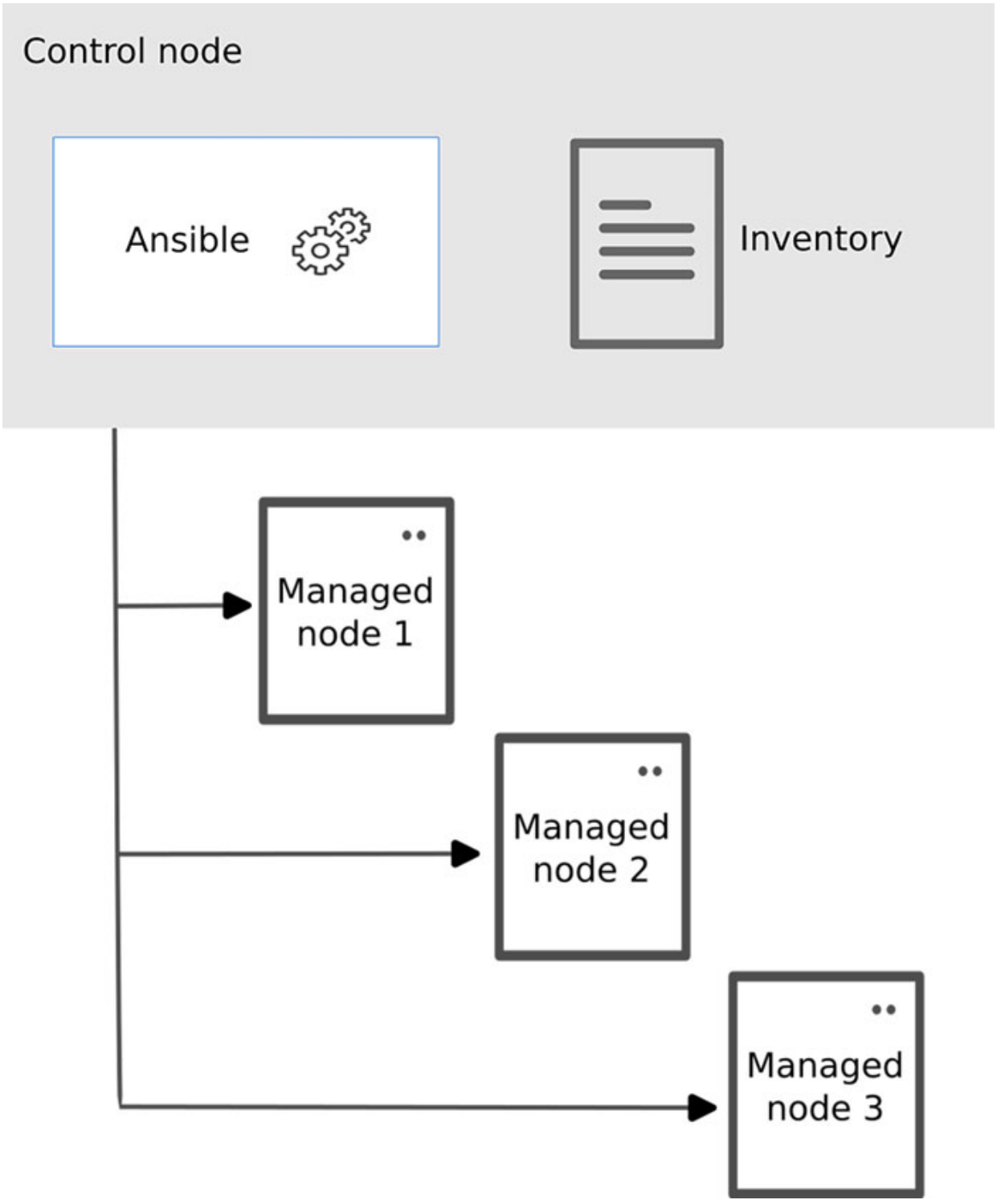


Figure 12.14: *Ansible diagram. Source: Ansible website.*

The default file containing the list of the *hosts* to be managed is **`/etc/ansible/hosts`**. An inventory can group hosts into *group hosts*. It is also possible to configure variables to groups, which will be applied to all

the hosts defined inside. The format of the inventory files can be ***INItialization (INI)*** or ***YAML Ain't Markup Language (YAML)***.

The following code shows an example of one Inventory with *INI* format, which is the default used, defining two groups called *webservers* and *dbservers,* and having four servers in each group.

```
[webservers]
alpha.example.org
beta.example.org
192.168.1.100
192.168.1.110
[dbservers]
db01.intranet.mydomain.net
db02.intranet.mydomain.net
10.25.1.56
10.25.1.57
```

As observed in the previous example, an inventory can contain a *DNS*-resolvable name or an IP. Groups can define systems with certain affinities, such as:

- **Geographically**: For example, hosts located in the same data center, city, or country.
- **Application**: Servers running similar applications, such as webs, databases, backup, and so on.
- **Operating system or distribution**: For example, to separate Linux and Windows systems, Ubuntu, and Red Hat Enterprise Linux systems.
- **Environment**: Such as development, test, production, and so on.

A host can be part of several groups, such as being part of *development* and *webservers* group. A host can have associated different parameters related to the connection or variables being used during the automation. Refer to the following example:

```
[linux]
rhel ansible_host=192.168.122.226 ansible_password=Start123
ubuntu ansible_host=192.168.122.43 ansible_password=Start123
```

The previous example defines two hosts, **`rhel`**, and **`ubuntu`**, and the IP and password configuration to connect to the nodes. The DNS, in this case, does

not need to be resolved; they are treated as an alias by *Ansible*. The previous example can be rewritten, defining the common variables at the group level instead at the host level as follows:

```
[linux]
rhel ansible_host=192.168.122.43
ubuntu ansible_host=192.168.122.226
[linux:vars]
ansible_user=agonzalez
ansible_password=Start123
owner=Alberto
```

In the previous example, a new variable to define the user to be used during the connection is defined: **`ansible_user`**. Another variable, **`owner`**, which is not used to define the connection, can be used by tasks or templates during the automation. Some popular variables are listed *table 12.4*:

Variable	Description
ansible_host	Defines the *DNS* or *IP* to connect.
ansible_port	Defines the port to be used for the connection.
ansible_password	Not recommended to set it as plain text for security reasons; it defines the password to be used for the connection. Recommended is to use a private/public key or an encrypted value (*vault*).
ansible_user	Defines the user to be used for the connection.
ansible_connection	Specifies the connection to be used, for example: *ssh* *local* *winrm*
ansible_become	Indicates to force privilege escalation.
ansible_become_method	Sets the privilege escalation method, for example: su sudo
ansible_become_user	Sets the user to become through privilege escalation.
ansible_ssh_private_key_file	Specifiles the private key to be used for *SSH* connection.
ansible_ssh_common_args	Specifies the arguments to be used for *SSH*.

Table 12.4: *Ansible variables on the inventory*

Ansible auto generates two groups automatically:

- *all*: It includes all the hosts defined in the inventory.
- *ungrouped*: It includes all the hosts which do not belong to any group.

It is also possible to define groups containing another group, as is shown in the following example:

```
anotherhost ansible_host=192.168.122.44
[production:children]
linux
windows
[linux]
rhel ansible_host=192.168.122.43
ubuntu ansible_host=192.168.122.226
[windows]
win01 ansible_host=192.168.122.33 ansible_connection=winrm
```

In the previous example, the group *production* will contain three hosts: **`rhel`**, **`ubuntu`**, and **`win01`**. The node **`anotherhost`** will be part of the groups **`ungrouped`** and **`all`**. *Ansible* provides a command named **ansible-inventory** to query the inventory indicated to display the configured inventory. *Figure 12.15* shows an example with the option **`--graph`**:

```
agonzalez@ubuntu:~$ ansible-inventory  -i /etc/ansible/hosts --graph
@all:
  |--@production:
  |  |--@linux:
  |  |  |--rhel
  |  |  |--ubuntu
  |  |--@windows:
  |  |  |--win01
  |--@ungrouped:
  |  |--anotherhost
```

Figure 12.15: *Output example for command ansible-inventory*

With the option **`--graph`** it is possible to specify as an argument the group to query. The option **`--list`** includes the variables defined for the hosts and groups. *Figure 12.16* shows an excerpt:

```
agonzalez@ubuntu:~$ ansible-inventory  -i /etc/ansible/hosts   --list | head -12
{
    "_meta": {
        "hostvars": {
            "anotherhost": {
                "ansible_host": "192.168.122.44"
            },
            "rhel": {
                "ansible_host": "192.168.122.43",
                "ansible_password": "Start123",
                "ansible_user": "agonzalez",
                "owner": "Alberto"
            },
```

Figure 12.16: *Output example for command ansible-inventory*

Ansible allows separation of the variables for **`hosts`** and **`groups`** in separated files. For that, it is possible to create two directories named **`host_vars`** and **`group_vars`**. The following structure shows an example:

```
|-/etc/ansible/hosts
|--/etc/ansible/host_vars/
|---/etc/ansible/host_vars/all.yml
|---/etc/ansible/host_vars/rhel.yml
|---/etc/ansible/host_vars/ubuntu.yml
|--/etc/ansible/group_vars/
|---/etc/ansible/group_vars/all.yml
|---/etc/ansible/group_vars/linux.yml
|---/etc/ansible/group_vars/windows.yml
```

The inventory file for *Ansible* can be hosted in any part of the system. Then, the structure will be similar, depending on the directory where the main inventory file is located, as is shown:

```
|-/inventory/production/servers
|--/inventory/production/host_vars/
|---/inventory/production/host_vars/all.yml
|---/inventory/production/host_vars/rhel.yml
|---/inventory/production/host_vars/ubuntu.yml
|-/inventory/production/group_vars/
|---/inventory/production/group_vars/all.yml
|---/inventory/production/group_vars/linux.yml
|---/inventory/production/group_vars/windows.yml
```

Separating the variables in different files requires the definition to be in *YAML* format. The following code shows an example for the **`windows.yml`** file:

```
ansible_host: 192.168.122.33
ansible_connection: winrm
```

As described before, an inventory can be in YAML format. The previous Inventory defined on *INI* format can be defined as *YAML* as the following:

```
all:
  hosts:
    anotherhost:
      ansible_host: 192.168.122.44
  children:
    production:
      children:
    linux:
      hosts:
        rhel:
        ansible_host: 192.168.122.43
        ansible_password: Start123
        ansible_user: agonzalez
        owner: Alberto
        ubuntu:
        ansible_host: 192.168.122.226
        ansible_password: Start123
        ansible_user: agonzalez
        owner: Alberto
    windows:
      hosts:
        win01:
        ansible_connection: winrm
        ansible_host: 192.168.122.33
```

An inventory can contain a range of hosts; the following example will define five Web servers in one line:

```
web0[1:5].example.com
```

AD-HOC actions

Ansible brings the command **`ansible`** to check connectivity to the nodes or to perform simple tasks. This task is not saved in any file and does not need any file to read the task to be performed. The syntax is as follows:

```
ansible [options] servers|groups|all|ungrouped -m module [-a
arguments]
```

The module used to check connectivity is named **`ping`**. If the connectivity is correct, it will show the word *"pong"*, and the task will be marked as **`SUCCESS`**. If the connection fails, it will be marked as **`UNREACHABLE`**. *Figure 12.17* shows the two scenarios:

```
agonzalez@ubuntu:~$ ansible -i /etc/ansible/hosts -m ping rhel
rhel | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
agonzalez@ubuntu:~$ ansible -i /etc/ansible/hosts -m ping web01.example.com
web01.example.com | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: ssh: Could not resolve hostnam
e web01.example.com: Temporary failure in name resolution",
    "unreachable": true
}
```

Figure 12.17: *Output example for command ansible*

Other common modules to be used in the *ad-hoc* mode are as follows:

- **`command`**: Execute a command in the managed nodes.
- **`shell`**: Execute a command in the managed node but using the *shell* configuration.
- **`copy`**: Copy a file to a remote node.

Figure 12.18 shows an example using the **`command`** module and the **`copy`** one. Notice that the file **`/etc/ansible/hosts`** are the default ones; it is not needed to specify them.

```
agonzalez@ubuntu:~$ ansible linux -m command -a "uname -r"
rhel | CHANGED | rc=0 >>
5.15.0-48-generic
ubuntu | CHANGED | rc=0 >>
5.14.0-70.22.1.el9_0.x86_64
agonzalez@ubuntu:~$ ansible rhel -m copy -a "src=/etc/hosts dest=/tmp/hosts"
rhel | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "checksum": "3be3cada0d11b51a5ab9474c501cdff420eec49a",
    "dest": "/tmp/hosts",
    "gid": 1000,
    "group": "agonzalez",
    "md5sum": "b7c136b1987c972de7d0808e12221abe",
    "mode": "0664",
    "owner": "agonzalez",
    "size": 221,
    "src": "/home/agonzalez/.ansible/tmp/ansible-tmp-1666126158.960906-27051-1538
39661330572/source",
    "state": "file",
    "uid": 1000
}
```

Figure 12.18: *Output example for command ansible*

The previous example executes the command **`uname -r`** on all the hosts defined inside the group **`linux`**. The second command copies the file **`/etc/hosts`** from the control node to the remote node, storing the file on **`/tmp/`**hosts. As the destination file does not exist or the content is different, the task is marked as **`CHANGED`**. When the content and properties of the file are the same, the file is not modified, and the task is marked as **`SUCCESS`**, as shown in *figure 12.19*:

```
agonzalez@ubuntu:~$ ansible rhel -m copy -a "src=/etc/hosts dest=/tmp/hosts"
rhel | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "checksum": "3be3cada0d11b51a5ab9474c501cdff420eec49a",
    "dest": "/tmp/hosts",
    "gid": 1000,
    "group": "agonzalez",
    "mode": "0664",
    "owner": "agonzalez",
    "path": "/tmp/hosts",
    "size": 221,
    "state": "file",
    "uid": 1000
}
```

Figure 12.19: *Output example for command ansible*

Ad-hoc should be used only on specific occasions, such as checking connectivity or performing a quick task. It is recommended to write a *Playbook* with the tasks defined to be able to reuse or perform modifications when needed.

YAML

This data-serialization language is one of the most popular to define the configuration for applications. The popularity is due to the readability; the creation and the manipulation are easier than other formats such as *XML* and *JSON*. In the previous section, *Inventory*, we introduced it to define hosts and variables. The playbooks, tasks, and variables in *Ansible* will be defined in *YAML*. The following block shows a valid *YAML* code:

```
---
name: Alberto
age: 38
married: False
technologies:
  - Ansible
  - Terraform
  - Linux
data:
  name: Alberto
```

```
  lastname: Gonzalez
…
```

From the previous example, it is possible to observe the following concepts:

- A *YAML* file starts with three dashes (-).
- A *YAML* file ends with three dots (.).
- A normal variable can contain a string *(name)* or a number (*age)*. A string can be wrapped using double or simple quotes.
- It is possible to define *boolean* variables (*married*).
- It is possible to define a variable containing a list of elements (*technologies*).
- It is possible to define a dictionary *(data*) that contains a *key* and a *value*. To access the value, there are the following two ways:
    - `data['name']`
    - `data.name`

It is possible to have complex variables, such as list of lists, list of dictionaries, or a value of a dictionary that can contain a list, among other complex combinations. *YAML* gives the flexibility to define a variable containing a value with multiple lines or with a long line. Using the special characters pipe (|) and greater than (>). Refer to the following example:

```
---
separated_lines: |
Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris
nisi ut aliquip ex ea commodo consequat.
joined_lines: >
Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris
nisi ut aliquip ex ea commodo consequat.
```

...

Table 12.5 shows the result of the variable (where *\n* is the special character for the new line):

`separated_lines`	`Lorem ipsum dolor sit amet, consectetur adipiscing elit, \nsed do eiusmod tempor incididunt ut labore et dolore magna aliqua. \nUt enim ad minim veniam, quis nostrud exercitation ullamco laboris \nnisi ut aliquip ex ea commodo consequat\n`
`joined_lines`	`Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat\n`

Table 12.5: *Multiline Ansible variables examples*

Ansible configuration

Certain settings in *Ansible* are adjustable using a configuration file named **`ansible.cfg`**. The file can be generated using the command **`ansible-config`** with the argument **`init`**. It is possible to use the option **`--disabled`** to include all the options available. The option **`-t all`** includes all the options for the **`plugins`**. The configuration file contains different sections, and some examples are as follows:

- **`[defaults]`**: Default options for *Ansible* tools.
- **`[privilege_escalation]`**: Options for the privilege escalation, such as **`sudo`**.
- **`[persistent_connection]`**: Options for *SSH* persistent connection.
- **`[colors]`**: Defines the colors used for the different statuses (changed, error, and so on).
- **`[galaxy]`**: Options for the public repository **`Ansible Galaxy`**.
- **`[inventory]`**: Options for the inventory files and **`plugins`**.
- **`[jinja2]`**: Options for the template system named **`Jinja2`**.
- **`[ssh_connection]`**: Options for *SSH*.
- **`[tags]`**: Options for the **`tags`** on **`Ansible`**.

Ansible will try to locate the **`ansible.cfg`** file in the following order:

- **`ANSIBLE_CONFIG`** environment variable pointing to the location of the **`ansible.cfg`**
- Current directory
- The file **`~/.ansible.cfg`**
- **`/etc/ansible/ansible.cfg`**

Some of the popular options in the configuration and the default values are defined in *table 12.6*:

Option	Default value	Description
`[defaults]`		
`inventory`	`/etc/ansible/hosts`	Default inventory source
`forks`	`5`	The maximum number of forks used to execute tasks.
`host_key_checking`	`True`	Check the host key of the target hosts.
`timeout`	`10`	Default timeout for connection plugins to use
`[privilege_escalation]`		
`become`	`False`	Toggles the use of privilege escalation
`become_method`	`sudo`	Privilege escalation method to use when "become" is enabled.
`become_user`	`root`	The user your remote user "*becomes*".
`become_ask_pass`	`False`	Toggle to prompt for privilege escalation password.
`[ssh_connection]`		
`ssh_args`	`-C -o ControlMaster=auto -o ControlPersist=60s`	Arguments to pass to all SSH CLI tools.
`control_path_dir`	`~/.ansible/cp`	Directory to be used for *ControlPath*.

Table 12.6: *Ansible configuration parameters*

Playbooks

A *Playbook* contains a list of *Plays* to perform. A *Play* defines the hosts to be administrated and a list of *Tasks* to perform on them. A *Playbook* contains at least one *Play.* A *Play* contains at least one *Task* and the optionality to connect to the systems and variables needed.

The format for the *Playbook* is *YAML*. The following example shows a simple *Playbook* containing a *Play*.

```
---
- name: Configure Linux systems
  hosts: linux
  tasks:
    - name: Copy resolv.conf file
      ansible.builtin.copy:
    src: /etc/resolv.conf
    dest: /etc/resolv.conf
…
```

In the previous *Playbook*, example is defined as follows:

- A single **`Play`** is defined with the name *"***`Configure Linux Systems`***"*. The option **`name`** is not required but recommended.
- The **`Play's`** tasks will be executed in the hosts within the group **`linux`**. The option **`hosts`** is mandatory.
- The **`Play`** contains only one task:
    - The **`Task`** has the name **`Copy resolv.conf file`**. It is a good practice to specify the names for the tasks.
    - The **`module`** used is **`ansible.builtin.copy`**. This **`module`** copies a file from the control node (where **`Ansible`** is executed) to the remote hosts. The field copied in **`/etc/resolv.conf`**
        - Latest versions of *Ansible* prefixes core modules with **`ansible.builtin`**. It is possible to omit it, but it is still recommended to use it when other **`collections`** (described as follows) are used.

As described previously, a **`Playbook`** can contain more than one *Play*. The following example code shows two *Plays*, one to configure systems in group **`linux`** and another **`Play`** to configure systems on the **`windows`** group.

```
---
```

```
- name: Configure Linux systems
  hosts: linux
  become: True
  tasks:
    - name: Copy resolv.conf file
      ansible.builtin.copy:
    src: /etc/resolv.conf
    dest: /etc/resolv.conf
- name: Configure Windows systems
  hosts: windows
  tasks:
    - name: Install 7-Zip
      chocolatey.chocolatey.win_chocolatey:
    name: 7zip
…
```

A *Play* can define connection parameters and variables to be used in the tasks. The order to define the options, such as **become**, **vars**, or **remote_user** in the following example, can be in the order desired by the user.

```
---
- name: Configure Linux systems
  hosts: linux
  become: True
  remote_user: agonzalez
  vars:
    environment: development
  tasks:
  (omitted)
…
```

After the creation of the *Playbook* file, the command **ansible-playbook** is needed to execute it. It contains the same options as the command **ansible**, but it requires an argument: the file to be used (usually with extension *.yml* or *.yaml*). *Figure 12.20* shows the execution of the first example:

```
agonzalez@ansible:~/playbooks$ ansible-playbook  playbook1.yaml

PLAY [Configure Linux systems] ***************************************************

TASK [Gathering Facts] ***********************************************************
ok: [rhel]
ok: [ubuntu]

TASK [Copy resolv.conf file] *****************************************************
fatal: [ubuntu]: FAILED! => {"changed": false, "checksum": "335e7bdf353788eaa7698
edd727e59c9e292c342", "msg": "Destination /etc not writable"}
fatal: [rhel]: FAILED! => {"changed": false, "checksum": "335e7bdf353788eaa7698ed
d727e59c9e292c342", "msg": "Destination /etc not writable"}

PLAY RECAP ***********************************************************************
rhel                       : ok=1    changed=0    unreachable=0    failed=1    sk
ipped=0    rescued=0    ignored=0
ubuntu                     : ok=1    changed=0    unreachable=0    failed=1    sk
ipped=0    rescued=0    ignored=0
```

Figure 12.20: *Output example for command ansible-playbook*

The previous example shows the connection was possible (task *Gathering Facts*) but copying the **`resolv.conf`** failed because the **`/etc`** directory is not writable. The Inventory used is configured to connect with the user **`agonzalez`**. To be able to write in the file **`/etc/resolv.conf`**, we need to use privilege escalation. This can be configured in the playbook (*become*), or it is possible to use the option **`-b (--become)`** from the command **`ansible-playbook`**, as illustrated in *figure 12.21*:

```
agonzalez@ansible:~/playbooks$ ansible-playbook -b  playbook1.yaml

PLAY [Configure Linux systems] ***************************************************

TASK [Gathering Facts] ***********************************************************
ok: [rhel]
ok: [ubuntu]

TASK [Copy resolv.conf file] *****************************************************
changed: [ubuntu]
changed: [rhel]

PLAY RECAP ***********************************************************************
rhel                       : ok=2    changed=1    unreachable=0    failed=0    sk
ipped=0    rescued=0    ignored=0
ubuntu                     : ok=2    changed=1    unreachable=0    failed=0    sk
ipped=0    rescued=0    ignored=0
```

Figure 12.21: *Output example for command ansible-playbook*

The **`ansible-playbook`** commands show the result for the tasks performed in the hosts.

- If the task did not modify anything in the system, it will be marked as *ok.*
- If a task is modified in the system, it is marked as *changed.*
- If it was not possible to connect to the system, it will be marked as *unreachable*.
- If a task throws an error, it will be marked as *failed*.
- If a task was not executed in a host (because of conditions), it is marked as *skipped*.
- The value *rescued* is for the use of error handling, and *ignored* is used when unreachable hosts are marked as ignored.

When a *Playbook* is executed, the first task is to gather the *facts*. The *facts* are values collected from the system, such as the operating system, distribution, and IP addresses, among other information. It is possible to disable the collection of the data with the option **`gather_facts: False`** at the **`Play`** level, as is shown in the following example:

```
---
- name: Configure Linux systems
  hosts: linux
  gather_facts: False
  (omitted)
…
```

The command **`ansible-playbook`** includes the options **`--list-hosts`** and **`--list-tasks`**, which are useful to see which hosts are going to be used and the tasks defined in the **`Playbook`**. *Figure 12.22* shows an example output:

```
agonzalez@ansible:~/playbooks$ ansible-playbook --list-hosts  playbook1.yaml

playbook: playbook1.yaml

  play #1 (linux): Configure Linux systems      TAGS: []
    pattern: ['linux']
    hosts (2):
      ubuntu
      rhel
```

Figure 12.22: *Output example for command ansible-playbook*

The option **`-list-tasks`** can be seen in *figure 12.23*:

```
agonzalez@ansible:~/playbooks$ ansible-playbook --list-tasks  playbook1.yaml

playbook: playbook1.yaml

  play #1 (linux): Configure Linux systems      TAGS: []
    tasks:
      Copy resolv.conf file     TAGS: []
```

Figure 12.23: *Output example for command ansible-playbook*

The popular available for the command **`ansible-playbook`** is shown in *table 12.7*:

Option	Description
--syntax-check	Check the syntax of the *Playbook* and the additional files required. It does not execute the *Playbook*.
--check / -C	It executes the playbook in *check* mode. It performs the tasks but without modifying the system.
--step	Ask if for each *Task* if it should be executed or not.
--start-at-task	Starts the *Playbook* in the task indicated (by *name* value). Useful to resume a failed *Playbook.*
--verbose / -v	Enables the *verbose mode*. It is possible to specify *-vvv* (3) to show more information and with *-vvvv* (4) more detailed information about the connection to the hosts.

Table 12.7: *Popular options for command Ansible-playbook*

Variables

Ansible uses *variables* to manage differences between the hosts. For example, it is possible to perform a task only when the distribution is a specific one but skips it if it is a different one. It is possible to define variables directly in a *Playbook*, *Inventory*, or a file and include it or specify variables as options running **`ansible-playbook`**. Some examples of variable use cases are as follows:

- Define the port for an application, for example, a variable to define the port 443 for a Web server in production or 8443 for development.
- Specify the user to be used. For example, the user **`www-data`** for **`Ubuntu`** and **`apache for CentOS`** systems.

- Specify the configuration file. For example, **`/etc/apache2/apache2.conf`** for **`Ubuntu`** and **`/etc/httpd/conf/httpd.conf`** on **`CentOS`**.

When a *variable* is part of a string or template, it is needed to wrap it between double curly brackets:

```
{{ variable }}
```

The module **`debug`** on **`Ansible`** allows the display of the value of a variable using the argument **`var`** or showing a message on the screen with the argument **`msg`**. This module is useful to ensure the proper value for the variable is set. The following code shows an example:

```
- name: Variable examples
  hosts: localhost
  gather_facts: False
  vars:
    firstname: Alberto
  tasks:
    - name: Show variable name
      ansible.builtin.debug:
    var: firstname
    - name: Show a message
      ansible.builtin.debug:
    msg: "Your name is {{ firstname }}"
```

In the previous example, there are some considerations to be taken into account:

- In *Ansible*, it is not mandatory to start with a triple dash and end with triple dots.
- The host **`localhost`** is always available, and the connection will be local (not *SSH*).
- It does not gather the *facts*.
- A variable **`firstname`** is defined and is shown in two tasks.

The output of the execution of the previous *Playbook* is shown in *figure 12.24*:

```
PLAY [Variable examples] ***************************************************

TASK [Show variable name] **************************************************
ok: [localhost] => {
    "firstname": "Alberto"
}

TASK [Show a message] ******************************************************
ok: [localhost] => {
    "msg": "Your name is Alberto"
}

PLAY RECAP *****************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0    sk
ipped=0    rescued=0    ignored=0
```

Figure 12.24: *Output example using variables*

A variable defined in a *Playbook* can be overwritten using the argument **`--extra-vars (-e)`**. The following example in *figure 12.25* shows the usage of the option **`--start-at-task`** and how to override the value of the variable **`firstname`**.

```
agonzalez@ansible:~/playbooks$ ansible-playbook playbook2.yaml --start-at-task "S
how a message" --extra-vars firstname=John

PLAY [Variable examples] ***************************************************

TASK [Show a message] ******************************************************
ok: [localhost] => {
    "msg": "Your name is John"
}

PLAY RECAP *****************************************************************
localhost                  : ok=1    changed=0    unreachable=0    failed=0    sk
ipped=0    rescued=0    ignored=0
```

Figure 12.25: *Output example using extra variables*

Variable precedence

As described previously, a *variable* can be defined in many different places. If a variable is defined in different places, *Ansible* will use the definition with higher-order precedence. When writing a *Playbook* it is important to define the name of the variables correctly, to avoid accidentally variable overriding. For example, a variable named **`port`** is a bad option. If we are defining the port for one application, then it is recommended to prefix it with the name of the application, such as **`httpd_port`**. The following list shows

the order of precedence from least to greatest (the last listed variables override all other variables):

1. Command line values (for example, -u **`my_user`**, these are not variables)
2. Role defaults (defined in **`role/defaults/main.yml`**)
3. Inventory file group vars
    a. inventory **`group_vars/all`**
    b. inventory **`group_vars/*`**
4. Inventory file host vars
    a. inventory **`host_vars/*`**
5. Host facts
6. Play option **`vars`**
7. Play option **`vars_prompt`**
8. Play option **`vars_files`**
9. Role vars (defined in **`role/vars/main.yml`**)
10. Block vars (only for tasks in the block)
11. Task vars (only for the task)
12. Task **`include_vars`**
13. **`set_facts`** / registered vars
14. role (and **`include_role`**) params
15. include params
16. extra vars (for example, **`-e`** "**`user=my_user`**") (always win precedence)

Handlers

A **`handler`** is a task that will be executed only if another task is triggering it. One of the most common use cases is to define a **`handler`** to restart a service. This task will be only triggered if a configuration file is modified.

To define a **`handler`**, a task or several tasks will be defined inside the section **`handlers`**. The task to trigger the action will use the special keyword named

notify. A task can notify several **handler tasks**. The following code shows an example:

```
- name: Configure the httpd service
  hosts: rhel
  gather_facts: False
  become: True
  tasks:
    - name: Copy the httpd.conf file
      ansible.builtin.copy:
    src: httpd.conf
    dest: /etc/httpd/conf/httpd.conf
      notify:
    - Restart httpd
  handlers:
    - name: Restart httpd
      ansible.builtin.service:
    name: httpd
    state: restarted
```

The previous example defines the following workflow:

- The system to be configured is the system **rhel**.
- The file **httpd.conf** will be copied to **/etc/httpd/conf/httpd.conf**
    - If the destination file is changed, it will notify **Restart httpd** task.
    - If the destination file is not changed, the task is marked as **ok**.
- The handlers section defines one task name **Restart httpd**, which will restart the service **httpd** using *Ansible's module* named **service**.

Figure 12.26 shows the output of the execution when the file is modified.

```
PLAY [Configure the httpd service] *********************************************

TASK [Copy the httpd.conf file] ************************************************
changed: [rhel]

RUNNING HANDLER [Restart httpd] ************************************************
changed: [rhel]

PLAY RECAP *********************************************************************
rhel                       : ok=2    changed=2    unreachable=0    failed=0    sk
ipped=0    rescued=0    ignored=0
```

Figure 12.26: *Output example using handlers*

If the host to be configured has the correct configuration, then the service will not be restarted to avoid unexpected downtimes of the service offered. *Figure 12.27* shows the output:

```
PLAY [Configure the httpd service] *********************************************

TASK [Copy the httpd.conf file] ************************************************
ok: [rhel]

PLAY RECAP *********************************************************************
rhel                       : ok=1    changed=0    unreachable=0    failed=0    sk
ipped=0    rescued=0    ignored=0
```

Figure 12.27: *Output example using handlers*

Include and import

Working with a single *Playbook* can contain many lines related to the variable definition, tasks, and handlers. *Ansible* allows to split of the file and includes them in the main file, using tasks or *Play* options. The possible statements to perform are described in *table 12.8*:

Statement	Description
include_vars	A task inside the tasks section. Include a YAML file with the variables. Example: **tasks**: `- include_vars: variables_httpd.yaml`

vars_files	At *Playbook* level is possible to define the files containing variables needed for the *Playbook*. Example:
	`- name: My playbook`
	`  hosts: webservers`
	`  vars_files:`
	`   - variables_httpd.yaml`

import_playbook	At *Playbook* level is possible to include external *Playbooks*. Example:
	`- import_playbook: configure_httpd.yaml`
	`- import_playbook: configure_mariadb.yaml`

include_tasks
import_tasks
	A task inside *tasks* section. It includes (or import) a list of tasks defined in a file. It is possible to pass variables to the tasks. Example:
	`tasks:`
	`- include_tasks: install_httpd.yaml`
	`- import_tasks: configure_httpd.yaml`
	`  vars:`
	`  httpd_port: 443`
	It is possible to include tasks in the handlers section.

include_roles
import_roles
	It includes a *role* (described as follows) inside the *tasks* section. Example:
	`- import_role:`
	`  name: httpd`
	`- include_role:`
	`  name: httpd`

Table 12.8: *Including files for variables, tasks, and roles syntax and examples.*

Import statements are pre-processed at the time playbooks are parsed; the **`include`** ones are processed during the execution of the playbook.

Facts and magic variables

When *Ansible* is connecting to a **`host`**, it gathers information about it by default. The information is called **`facts`**. A special variable called **`ansible_facts`** is filled with information about the remote system, such as the operating system, the distribution, the IP addresses, system resource information, and so on. It is possible to use the module **`debug`** to print the information about the **`ansible_facts`** variable. The following code shows the example (truncated, only some values are shown) output for a *Ubuntu* system:

```
TASK [debug]
*********************************************************************
*****
ok: [ubuntu] => {
   "ansible_facts": {
   "all_ipv4_addresses": [
      "192.168.122.43",
      "192.168.100.1"
```

```
    ],
    "all_ipv6_addresses": [
        "fe80::5054:ff:fe1a:9cc4",
        "fe80::5054:ff:fe8f:d41a"
    ],
    "architecture": "x86_64",
    "discovered_interpreter_python": "/usr/bin/python3",
    "distribution": "Ubuntu",
    "distribution_major_version": "22",
    "distribution_release": "jammy",
    "distribution_version": "22.04",
    "dns": {
        "nameservers": [
        "8.8.8.8"
        ]
    },
    "kernel": "5.15.0-50-generic",
    "kernel_version": "#56-Ubuntu SMP Tue Sep 20 13:23:26 UTC
    2022",
    "memfree_mb": 1806,
    "memtotal_mb": 3923,
    "nodename": "ubuntu",
    "os_family": "Debian",
    "processor_count": 2,
    }
}
```


The previous example shows just some of the options available in the **`ansible_facts`** variable. The variable includes more information about the system, such as detailed information about the ethernet network cards, disks, and mount points.

Ansible has some magic variables which cannot be set directly by the user, and *Ansible* will override the value based on the configuration or the facts obtained. For the variables defined in the **`ansible_facts`**, it will be prefixed by **`ansible_`**. For example, the **`os_family`** key from **`ansible_facts`** will be accessible through the magic variable **`ansible_os_family`**.

Conditionals

It is possible to condition the execution of one task-based using the expression **when**. The condition indicated can be if a variable has a specific value or any comparison desired. In the following task example, the task will be executed only if the **ansible_distribution** variable is equal to Ubuntu.

```
- name: Conditionals example
  hosts: linux
  become: True
  tasks:
    - name: Install apache2 package
      ansible.builtin.apt:
     name: apache2
      when: ansible_distribution == "Ubuntu"
```

If the condition is evaluated as false, then the task will be skipped for the specific *host*. *Figure 12.28* shows the output example:

```
agonzalez@ansible:~/playbooks$ ansible-playbook playbook5.yaml

PLAY [Conditionals example] ****************************************************

TASK [Gathering Facts] *********************************************************
ok: [rhel]
ok: [ubuntu]

TASK [Install apache2 package] *************************************************
skipping: [rhel]
changed: [ubuntu]

PLAY RECAP *********************************************************************
rhel                       : ok=1    changed=0    unreachable=0    failed=0    sk
ipped=1    rescued=0    ignored=0
ubuntu                     : ok=2    changed=1    unreachable=0    failed=0    sk
ipped=0    rescued=0    ignored=0
```

Figure 12.28: *Output example using conditions*

The simple comparing values are described in *table 12.9*:

Expression	Description	Example
item1 == item2	Check if two elements have the same value.	`ansible_distribution == "Ubuntu"`
item1 != item2	Check if two elements have a different value.	`ansible_distribution != "Ubuntu"`

item1 > *item2*	Check if *item1* is greater than *item2*	`ansible_distribution_major_version > 15`
item1 > *item2*	Check if *item1* is greater or equal to *item2*	`ansible_processor_count >= 2`
item1 < *item2*	Check if *item1* is less than *item2*	`ansible_distribution_major_version < 15`
item1 <= *item2*	Check if *item1* is less or equal to *item2*	`ansible_processor_count <= 2`

Table 12.9: *Comparation expressions*

Ansible allows more logical conditions, as described in *table 12.10*:

Expression	Description	Example
item1 ~ *item2*	Check if *item1* is inside the value of *item2*	`"x86" ~ ansible_architecture`
item1 in item2	Check if *item1* is inside the list of elements i*tem2*	`ansible_distribution in ["Ubuntu", "Debian"]`
item1 not in item2	*Check if item1* is not inside the list of elements i*tem2*	`ansible_distribution not in ["Ubuntu", "Debian"]`
item1 is defined	Check if the variable *item1* is defined	`username is defined`
item1 is not defined	Check if the variable *item1* is not defined	`username is not defined`

Table 12.10: *Logical conditions expressions*

Condition expressions can be combined with the words **`and`** as well as the word **`or`**. The following example shows the usage, where **`task`** is a variable defined by the user:

```
- name: Install apache2 package
  ansible.builtin.apt:
name: apache2
  when: ansible_distribution == "Ubuntu" and (
    task is defined and task == "install")
```

Loops

With Ansible, it is possible to loop elements and perform a task for each of the elements. It is possible to iterate over a simple list or complex variable,

such as a dictionary. Using the expression **loop**, it is possible to iterate over simple and complex elements. The following example shows a simple example:

```
- name: Install required packages
  ansible.builtin.apt:
name: "{{ item }}"
  loop:
- apache2
- php
- php-mysql
```

As observed in the previous example, the special variable **`item`** contains the value for each element in a **loop**. *Figure 12.29* shows the example output of the previous task execution:

```
TASK [Install required packages] ***********************************************
changed: [ubuntu] => (item=apache2)
changed: [ubuntu] => (item=php)
changed: [ubuntu] => (item=php-mysql)
```

Figure 12.29: *Output example using loops*

The following example shows how to loop a list of dictionaries, which contains the username and the group to be created. For that, the module **user** is used.

```
- name: Create users
  ansible.builtin.user:
name: "{{ item.name }}"
groups: "{{ item.groups }}"
  loop:
- {name: "user1", groups: "users"}
- {name: "user2", groups: "users,sudo"}
```

Register

An Ansible's task generates information about the execution. That information can be saved in a variable using the expression **`register.`** This is especially useful using the modules **`command`** or **`shell`**. The following code shows an example:

```
- name: Run a command
  ansible.builtin.command: hostname -I
```

```
    register: hostname_output
  - name: Show content of registered variable
    ansible.builtin.debug:
  var: hostname_output
```

Figure 12.30 shows the output of the **debug** task:

```
TASK [Show content of registered variable] ************************************
ok: [ubuntu] => {
    "hostname_output": {
        "changed": true,
        "cmd": [
            "hostname",
            "-I"
        ],
        "delta": "0:00:00.007307",
        "end": "2022-10-22 17:53:53.924471",
        "failed": false,
        "msg": "",
        "rc": 0,
        "start": "2022-10-22 17:53:53.917164",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "192.168.122.43 192.168.100.1 ",
        "stdout_lines": [
            "192.168.122.43 192.168.100.1 "
        ]
    }
}
```

Figure 12.30: *Output example using registered variables*

A registered variable is a dictionary with the following popular *keys*:

- **`changed`**: Indicates if the task changed the system.
- **`cmd`**: The command executed (if we use **command** or **shell**).
- **`failed`**: Indicates if the task failed.
- **`rc`**: The exit code (if we use **command** or **shell**).
- **`stderr`** / **`stderr_lines`**: The standard error output messages.
- **`stdout`** / **`stdout_lines`**: The standard output messages.

To access one of the **`keys`** of the registered value, they are the following two options:

```
  - name: Show stdout of the variable (option 1)
    ansible.builtin.debug:
  msg: "Uptime: {{ hostname_output.stdout }}"
```

```
  - name: Show stdout of the variable (option 2)
    ansible.builtin.debug:
  msg: "Uptime: {{ hostname_output['stdout'] }}"
```

Templates

A template is a dynamic file that includes variables and different statements: conditions, loops, and so on. These templates use the language *Jinja2*, which eases the definition of the templates, and *YAML* is easy to read. *Table 12.11* shows the available popular syntax to be used in a *Jinja2* template file:

Type	Syntax	Example
Variables	`{{ variable }}`	`{{ ansible_fqdn }}`

Condition	`{% if expression %}`
	`statements`
	`{% elif %}`
	`statements`
	`{% else %}`
	`statements`
	`{% endif %}`
		`{% if purpose == "production" %}`
		`ALERT: This is a production system.`
		`{% else %}`
		`This is NOT a production system.`
		`{% endif %}`

Loop	`{% for variable in list %}`
	`statements`
	`{% endfor %}`
		`allowed_users:`
		`{% for user in userlist %}`
		`- {{ user }}`
		`{% endfor %}`

Comment	`{# text #}`	`{# This a comment #}`

Table 12.11: *Jinja2 template syntax*

The module **template** processes the template where the *Ansible* tool is executed (such as **ansible-playbook**), and the processed file is transferred to the administrated *host*. The following template shows how to fill the file **/etc/hosts** with the hosts in the Inventory.

```
127.0.0.1 localhost localhost.localdomain localhost4
::1       localhost localhost.localdomain localhost6
{% for host in play_hosts %}
{{ hostvars[host]['ansible_default_ipv4']['address'] }} {{ host
}}
{% endfor %}
```

The example task to process the template is as follows:

```
  - name: Fill /etc/hosts
```

```
    ansible.builtin.template:
  src: hosts.j2
  dest: /etc/hosts
```

Blocks

Ansible allows to group several tasks in a *Block*. This allows to specify a different parameter such as **become**, **become_user** or **user**. The expression used to define a Block is **block**, and it contains a list of tasks. The following code shows a *Play* example using a block:

```
- name: Block example
  hosts: linux
  become: True
  tasks:
    - name: Install and configure Ubuntu systems
      when: ansible_distribution == "Ubuntu"
      block:
      - name: Install required packages
    ansible.builtin.apt:
      name: "{{ item }}"
    loop:
      - apache2
      - php
      - php-mysql
      - name: Generate index.html file
    ansible.builtin.copy:
      content: "Hello to {{ ansible_hostname }}"
      dest: /var/www/html/index.html
```

List of popular modules

Ansible brings a big number of the core modules to perform a wide variety of tasks in the managed *hosts*. The following list shows some of the modules inside the **ansible.builtin** collection:

- **add_host**: Add a host (and alternatively a group) to the ansible-playbook in-memory inventory.
- **apt**: Manages apt-packages.

- **`apt_key`**: Add or remove an apt-key.
- **`apt_repository`**: Add and remove APT repositories.
- **`assemble`**: Assemble configuration files from fragments.
- **`assert`**: Asserts that given expressions are true.
- **`async_status`**: Obtain the status of an asynchronous task.
- **`blockinfile`**: Inserts/updates/removes a text block surrounded by marker lines.
- **`command`**: Execute commands on targets.
- **`copy`**: Copy files to remote locations.
- **`cron`**: Manage cron.d and crontab entries.
- **`debconf`**: Configure a .deb package.
- **`debug`**: Print statements during execution.
- **`dnf`**: Manages packages with the dnf package manager.
- **`dpkg_selections`**: Dpkg package selections.
- **`expect`**: Executes a command and responds to prompts.
- **`fail`**: Fail with a custom message.
- **`fetch`**: Fetch files from remote nodes.
- **`file`**: Manage files and file properties.
- **`find`**: Return a list of files based on specific criteria.
- **`gather_facts`**: Gathers facts about remote hosts.
- **`get_url`**: Downloads files from HTTP, HTTPS, or FTP to the node.
- **`getent`**: A wrapper to the UNIX *getent* utility.
- **`git`**: Deploy software (or files) from git checkouts.
- **`group`**: Add or remove groups.
- **`group_by`**: Create Ansible groups based on facts.
- **`hostname`**: Manage hostname.
- **`import_playbook`**: Import a playbook.
- **`import_role`**: Import a role into a play.
- **`import_tasks`**: Import a task list.
- **`include`**: Include a task list.

- **`include_role`**: Load and execute a role.
- **`include_tasks`**: Dynamically include a task list.
- **`include_vars`**: Load variables from files dynamically within a task.
- **`iptables`**: Modify iptables rules.
- **`known_hosts`**: Add or remove a host from the **`known_hosts`** file.
- **`lineinfile`**: Manage lines in text files.
- **`meta`**: Execute Ansible actions.
- **`package`**: Generic OS package manager.
- **`package_facts`**: Package information as facts.
- **`pause`**: Pause playbook execution.
- **`ping`**: Try to connect to the host, verify a usable python and return pong on success.
- **`pip`**: Manages Python library dependencies.
- **`raw`**: Executes a low-down and dirty command.
- **`reboot`**: Reboot a machine.
- **`replace`**: Replace all instances of a particular string in a file using a back-referenced regular expression.
- **`rpm_key`**: Adds or removes a gpg key from the rpm db.
- **`script`**: Runs a local script on a remote node after transferring it.
- **`service`**: Manage services.
- **`service_facts`**: Return service state information as fact data.
- **`set_fact`**: Set host variable(s) and fact(s).
- **`set_stats`**: Define and display stats for the current ansible run.
- **`setup`**: Gathers facts about remote hosts.
- **`shell`**: Execute shell commands on targets.
- **`slurp`**: Slurps a file from remote nodes.
- **`stat`**: Retrieve file or file system status.
- **`subversion`**: Deploys a subversion repository.
- **`systemd`**: Manage systemd units.
- **`sysvinit`**: Manage SysV services.

- **`tempfile`**: Creates temporary files and directories.
- **`template`**: Template a file out to a target host.
- **`unarchive`**: Unpacks an archive after (optionally) copying it from the local machine.
- **`uri`**: Interacts with Web services.
- **`user`**: Manage user accounts.
- **`validate_argument_spec`**: Validate role argument specs.
- **`wait_for`**: Waits for a condition before continuing.
- **`wait_for_connection`**: Waits until the remote system is reachable/usable.
- **`yum`**: Manages packages with the yum package manager.
- **`yum_repository`**: Add or remove YUM repositories.

Roles

Roles organize *Tasks, Variables*, *Handlers, Templates*, and static files in a directory. This structure can be reused in different projects. The directory containing *Roles* is, by default, called *roles*. Inside, there is a folder with the name of the role. The *Role* directory structure is as follows:

```
roles/role_name/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
```

```
└── main.yml
```

The **main.yml** (or **main.yaml**) for each directory will be included automatically when a role is included or imported. The meaning of each directory is as follows:

- **tasks/main.yml**: The main list of tasks that the role executes.
- **handlers/main.yml**: List of tasks used as handlers.
- **defaults/main.yml**: Default variables for the role. These variables have the lowest priority of any variables available and can be easily overridden by any other variable, including inventory variables.
- **vars/main.yml**: Other variables for the role.
- **files/main.yml**: Files that the role deploys.
- **meta/main.yml**: Metadata for the role, including role dependencies.

From a *Play*, it is possible to include a list of roles using the expression **roles** at the *Play* level. The following code shows an example:

```
---
- hosts: webservers
  roles:
    - common
    - apache2
```

It is possible to include a role dynamically in the **tasks** section, using the module **include_role**, or import statically, using the module **import_role**. The following code shows both uses:

```
---
- hosts: webservers
  tasks:
    - name: Import common role
      ansible.builtin.include_role:
    name: common
    - name: Include apache2 role
      ansible.builtin.include_role:
    name: apache2
      vars:
    port: 443
    webdir: "/var/www/html/"
```

Collections

From version 2.10 of Ansible, a new distribution format for the content was introduced. A collection can include *Playbooks*, *Roles*, *Modules*, and *Plugins*. The main goal is to reduce the number of modules in the core of Ansible and be more flexible with the new distribution of the content. Collections have the format of **`namespace_name.collection_name`**. The directory structure is the following:

```
ansible_collections/
└── namespace_name
   └── collection_name
├── docs/
├── galaxy.yml
├── meta/
│   └── runtime.yml
├── plugins/
│   ├── modules/
│   │   └── module1.py
│   ├── inventory/
│   └── …/
├── README.md
├── roles/
│   ├── role1/
│   ├── role2/
│   └── …/
├── playbooks/
│   ├── files/
│   ├── vars/
│   ├── templates/
│   └── tasks/
└── tests/
```

To use a collection does not need to be included or imported. It is possible to include directly in a task, as is shown in the following example:

```
- hosts: all
  tasks:
    - namespace_name.collection_name.module1:
    myvar: somevalue
```

It is possible to import or include a role from a collection, as is shown in the following example:

```
- hosts: all
  tasks:
  - ansible.builtin.import_role:
    name: namespace_name.collection_name.role1:
```

Ansible Galaxy

Galaxy is a hub for finding and sharing Ansible content. It contains a big number of collections and roles ready to use. The website is **https://galaxy.ansible.com/**, and it is possible to search for content created and supported by companies and by individuals.

Ansible brings the command **`ansible-galaxy`**, which allows users to create a directory structure for roles and collections, and it allows them to download and install roles from the hub. The syntax to create a directory structure are as follows:

- For roles:

  ansible-galaxy role **`init [--init-path INIT_PATH] role_name`**
- For collections:

  ansible-galaxy collection **`init [--init-path INIT_PATH] collection_name`**

To install a role or a collection, the argument to be used is *install* followed by the name of the role (with format **`owner.role_name`**) or the collection (with the full qualified collection named). *Figure 12.31* shows the installation of a common public role to manage Apache and a popular collection to manage **`Postgresql`**.

```
agonzalez@ansible:~/ag$ ansible-galaxy role install geerlingguy.apache -p roles/
Starting galaxy role install process
- downloading role 'apache', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-apache/archiv
e/3.3.0.tar.gz
- extracting geerlingguy.apache to /home/agonzalez/ag/roles/geerlingguy.apache
- geerlingguy.apache (3.3.0) was installed successfully
agonzalez@ansible:~/ag$ ansible-galaxy collection install community.postgresql -p
 ansible_collections/
Starting galaxy collection install process
[WARNING]: The specified collections path
'/home/agonzalez/ag/ansible_collections' is not part of the configured Ansible
collections paths
'/home/agonzalez/.ansible/collections:/usr/share/ansible/collections'. The
installed collection won't be picked up in an Ansible run.
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/community-postgresql-2.2.0.tar.gz
 to /home/agonzalez/.ansible/tmp/ansible-local-22175gzmp_2t0/tmpkchq8bl6/communit
y-postgresql-2.2.0-l1dsrxb0
Installing 'community.postgresql:2.2.0' to '/home/agonzalez/ag/ansible_collection
s/community/postgresql'
community.postgresql:2.2.0 was installed successfully
```

Figure 12.31: *Output example using ansible-galaxy*

The following **`Playbook`** shows how to use the role to configure **`Apache2`** and the collection to obtain information about **`PostgreSQL`**:

```
- name: Install Apache2 and PostgreSQL
  hosts: linux
  become: true
  tasks:
    - name: Configure Apache2 on Ubuntu
      when: ansible_distribution == "Ubuntu"
      block:
      - name: Install Apache2
        ansible.builtin.include_role:
          name: geerlingguy.apache
      - name: Install Postgres on RHEL system
        when: ansible_distribution == "RedHat"
        block:
      - name: Install postgresql packages
        ansible.builtin.yum:
          name: "{{ item }}"
        loop:
```

```
    - postgresql
    - postgresql-server
    - python3-psycopg2
- name: Initialize database if doesn't exist
  ansible.builtin.command: /usr/bin/postgresql-setup --initdb
  args:
    creates: /var/lib/pgsql/data
- name: Start the service
  ansible.builtin.service:
    name: postgresql
    state: started
    enabled: true
- name: Get information
  become_user: postgres
  community.postgresql.postgresql_info:
  register: postgres_info
- name: Print postgresql version
ansible.builtin.debug:
    msg: "{{ postgres_info.version.raw }}"
```

Conclusion

With the growth of the elements managed in an IT infrastructure, it is necessary to implement an IT automation solution. Historically, shell scripting was used to perform repetitive tasks in an easy way. Nowadays, the programming language **Python** is used to perform different tasks in the system because it is easy to learn and powerful to perform different tasks. An automation tool is required when multiple elements are managed, such as network devices or different systems with a variety of operating systems, Linux distributions, applications, and so on. The most popular tool is **Ansible** because it is easy to use and does not require special configurations or agents installed in the managed elements.

Key facts

- **Shell scripting** is an easy and powerful way to perform simple tasks in a system.

- **Python** is the easiest programming language to learn to perform automation.
- **Ansible** is the most popular automation tool for managing IT infrastructures.

Questions

1. How is the name of the first line of a script starting with **#!**?
    a. shebang
    b. interpreter
    c. posix
2. What function in a *shell script* is used to introduce data from the keyboard?
    a. input
    b. read
    c. prompt
3. What *Python* module includes the function *sleep*?
    a. sys
    b. time
    c. os
4. What *Ansible* inventory variable defines the user used to connect to the system?
    a. ansible_user
    b. ansible_username
    c. ansible_login
5. What option in the command **ansible-playbook** is used to start a specific task?
    a. --skip-previous-tasks
    b. --starting-task
    c. --start-at-task

Answers

1. a
2. b
3. b
4. a
5. c

CHAPTER 13

Containers and CI/CD

Introduction

Containers is one of the most popular technologies at the present time. Companies around the world started modernization of the applications and infrastructure to adapt them to the cloud and to the more agile ways to work. The popularity of the project **Docker** helped to make Linux containers a trend, which many companies adopted as a standard to deploy new applications. New open-source solutions to run containers got popularity as alternatives to **Docker**, where **Podman** is one of the most popular and used solutions. **Kubernetes** as orchestration and advanced solution got popularity in the market, where **Kubernetes Distributions** provided another layer to help developers and end users build and run the software.

Modernization of applications brought new practices to develop, test, and release software to production. This includes the following:

- **Continuous Integration (CI)**: The practice of automating the integration of code changes from multiple contributors into a single software project.
- **Continuous Delivery (CD)**: The practice where the code changes are automatically prepared for a release to production.

Structure

In this chapter, we will discuss the following topics:

- Introduction to containers and images
- Docker
- Image Registry
- Podman
- Container runtimes
- Kubernetes
- Introduction to continuous integration/delivery
- Jenkins, GitLab CI/CD, and GitHub Actions

Introduction to containers and images

Containers are not something new. They were historically used to isolate resources at the user level and application level. On **Linux**, the first approach to isolate system components before the containers was called *chroot* (also known as *jail*). It was used to isolate applications and users between them, but with the limitation of it being unable to isolate physical resources (memory, CPU, or devices). The first implementations of *chroot* started in the 80s decade.

Since the year 2000, new implementations to isolate resources appeared, not only on Linux systems but on *UNIX systems* (*BSD* systems, *Solaris*, and *AIX*) and Windows. *Virtuozzo* was a pioneer company in developing software to isolate resources at the operating system level. The solution was called the same name as the company, and it was proprietary software. In the year 2005, the company released, with an open-source license, the software *OpenVZ*. This software is still popular and used by different companies, especially those that offer **Virtual Private Servers** (**VPS**) at low cost.

Between 2000 and 2005, the technology advanced in the resource isolation level on *UNIX* systems. *FreeBSD* (a *UNIX* operating system*)* implemented "*FreeBSD jail*", similar to *chroot*, but able to isolate resources. In the year 2001, *Linux-Vserver* appeared as a free alternative to *Virtuozzo*, and this solution is considered the predecessor to the current *Containers* technology. In the year 2004, one of the best solutions to isolate resources appeared—*Solaris zones*. The *zones* were considered one of the best resource isolations till **Docker** appeared. Until the appearance of **Docker** in the year 2013, resource isolation was worth mentioning the following:

- **Workloard partitions** (**WPARs**) for the operating system *IBM AIX* in the year 2007.
- HP-UX Containers for the operating system called *HP-UX* also in the year 2007.
- *Linux Containers* (*LXC*) in the year 2008.

In the first years of **Docker** (2013 and 2014), it was using *LXC* to run *Containers*. From version 0.9, it started to use its own library (libcontainer) to handle the *Containers.*

Containers versus virtualization

In the beginning, the popularity of the **Containers** led to questions about how it was different from traditional **Virtualization**. The **Virtualization** technology consists of adding an abstraction layer for the physical resources, with the objective of improving the use of the resources. With the introduction of **Virtualization**, it is possible to create several different emulated environments (**Virtual Machines**) for different resources. Thanks to **Virtualization**, it is possible to run **Virtual Machines** with different operating systems and with isolated resources (CPU, Memory, and Devices) inside of the same physical system. That allowed the companies to reduce costs by grouping services running in different physical systems in only one system, migrating the

services to **Virtual Machines**. There are different types of **Virtualization**, but the two most used nowadays are as follows:

- **Full Virtualization**: The Virtual Machine does not have direct access to the physical resources, and they require to have an upper layer to access them. Some software examples are as follows:
  - *Oracle VirtualBox*
  - *QEMU*
  - *Hyper-V*
  - *VMware ESXi*
- **OS-Level Virtualization**: This is where **Containers** are located. It is the *Operating System*, and not the *hardware*, that is responsible for isolating the resources and providing the tools to create, manipulate or manage the state of the **Containers** (term used instead **Virtual Machine**).

Refer to *figure 13.1* to see the difference between hardware and OS-Level Virtualization:

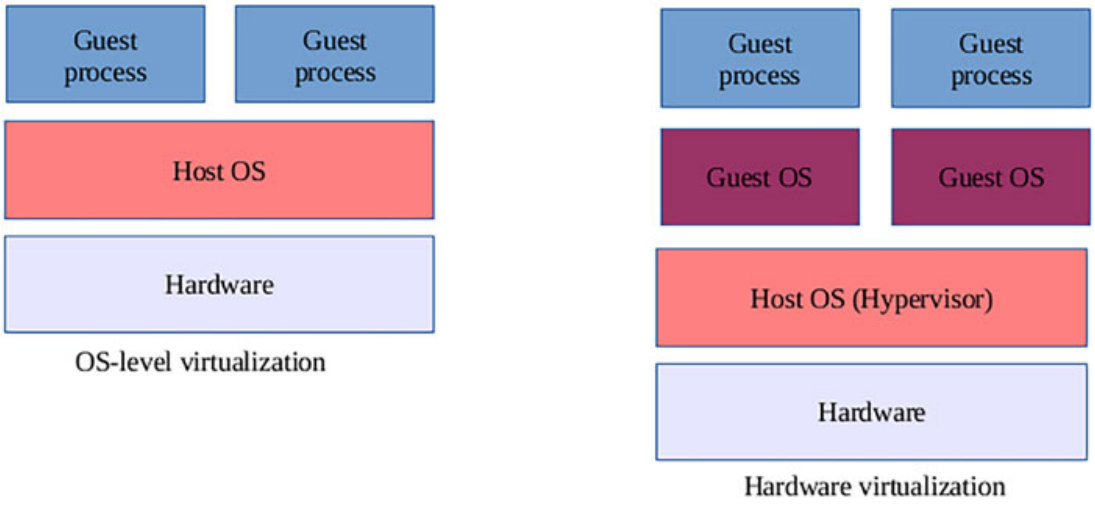


Figure 13.1: *Different between hardware and OS-level Virtualization*

Container content

A **Container** contains everything that is needed to execute one or multiple applications. A **Container** does not have a full *guest operating system* as a **Virtual Machine** does. The size of a **Container** is smaller and includes only the following:

- **Operating System libraries**: Only the libraries required to run the applications would be included. For example, an *SSL* library, in the case of a Web server application, requires a secure connection.

- **Operating system tools**: This can contain some tools needed for the application or to perform troubleshooting.
- **Runtime**: Some software is needed to execute the application inside the container. For example:
    - Interpreters: To run applications such as *Python, PHP, or Perl*.
    - Binaries for compiled languages, such as Java or Go.
- **Application files**: These are similar to the files that need to be interpreted, the binaries to run applications, configuration files, or any file needed for the application to be executed inside the *Container*.

A **Container** can execute several applications inside, but it is recommended to separate applications into several applications. The concept of *Microservices* consists in running small independent services which can be managed individually. That is part of the modernization to avoid having *Monolithic* architectures. Some reasons to separate services in different containers are as follows:

- Monitor applications individually.
- Limit resources accessible from the applications.
- Able to restart applications without affecting others.
- Scale up/down specific applications (services).

Images

As described previously, a *Container* contains everything that is needed to execute an application. An *Image* is a base to run *Containers* and to create a child image. The image has different layers (the *Kernel*, the minimal required files for the *operating system*, the libraries, and the customization). A *Base Image* is used to generate other images. *Figure 13.2* features the base image.

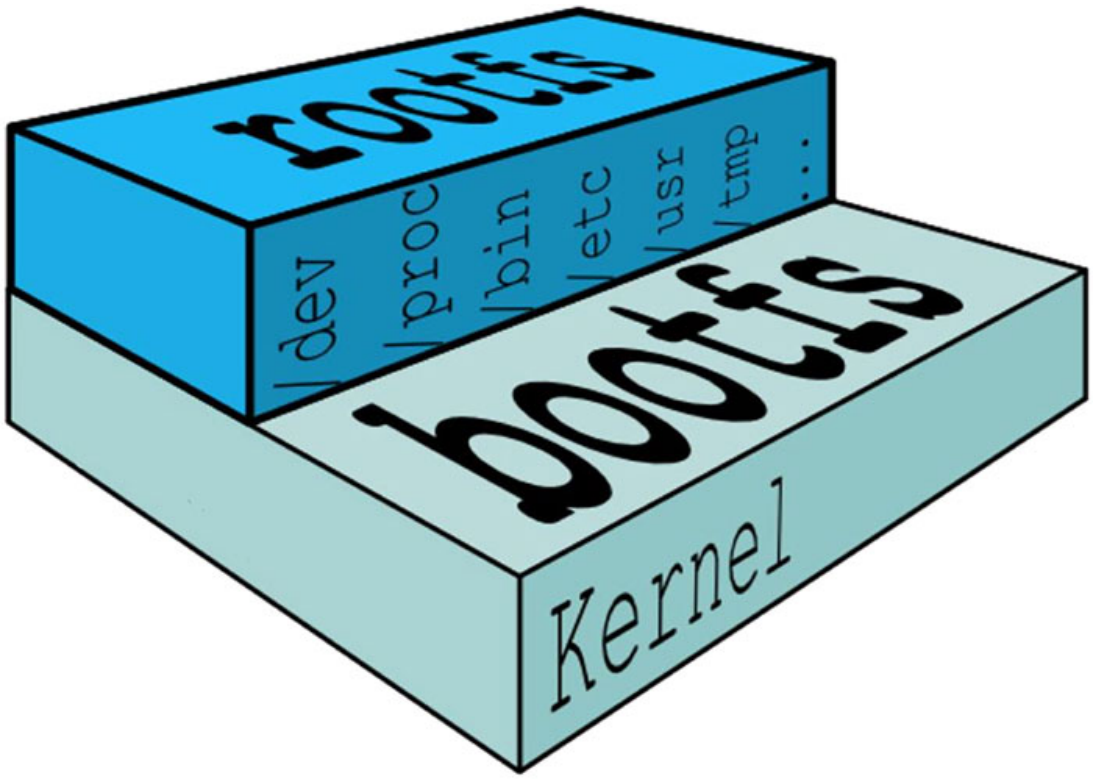


Figure 13.2: *Base image. Source: Docker*

After new software and files are added to the *Base Image*, a new *Image* is generated with the modifications. *Figure 13.3* shows the different layers applied to a *Base Image* based on *Debian,* configured to have a Web server installed and ready to be used.

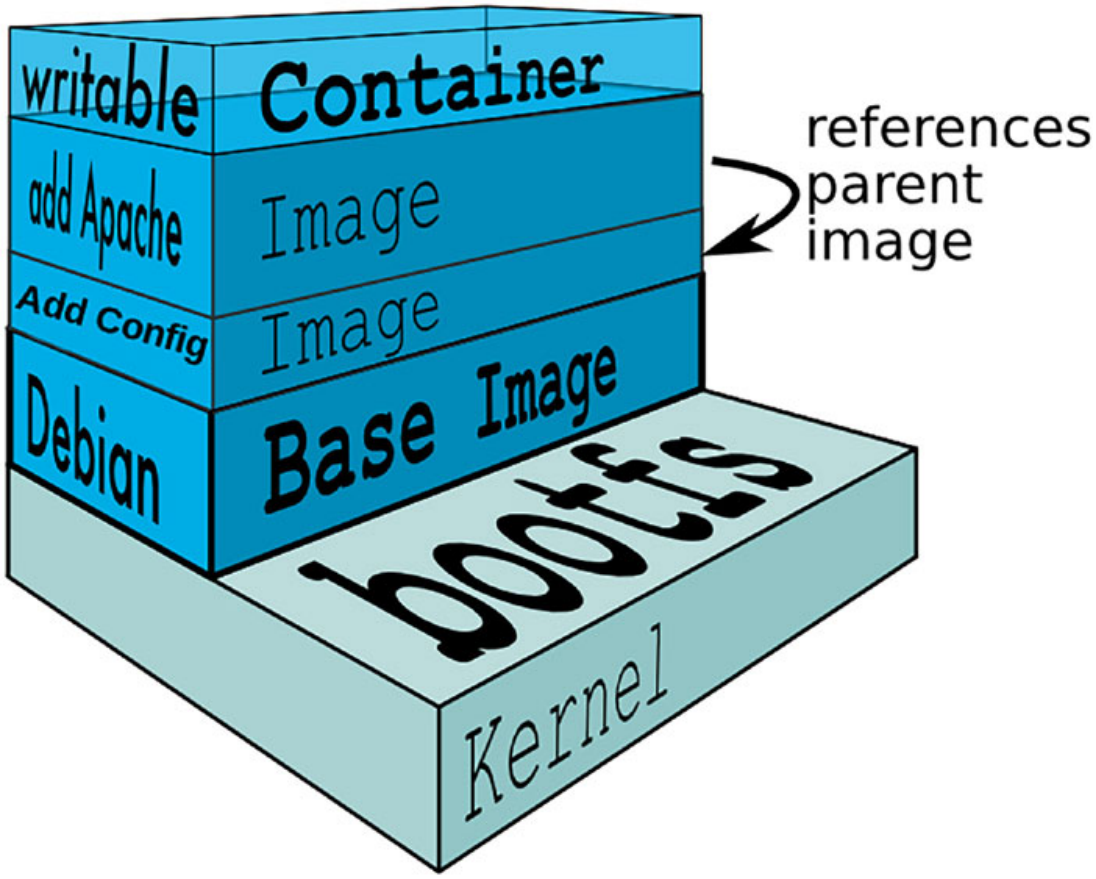


Figure 13.3: *Adaption of the base image with the software needed*

The layers represented in *figure 13.3* are as follows:

- Base Image with Debian
- Adding configuration to the Base Image
- Installing Apache in the previous layer

A **Container** based on this generated **Image** will be based on the *Debian base image* and will have configuration added. *Apache* is also installed. Any **Image** can be used by several containers at the same time. An **Image** has a history track and version control, and this helps show what changes were made in each layer. It is possible to go back to a previous version of it.

Cloud Providers, such as *AWS* or Google Cloud, offer the possibility to run containers on their platform. This is an ideal solution for customers who do not want to maintain a platform and want to modernize their applications.

Image registry

An **Image** can be generated locally and stored in the system where the **container** will be executed. But the purpose of **Images** is to be used by **Containers** in different environments and systems. An **Image Registry** is a repository to store the **Images** to be used. This **Registry** can be a public one or a private one inside the infrastructure where it will be used.

When a **Container** is created, it refers to an **Image**, and if it is not downloaded previously, it will be automatically pulled from the **Image Registry**. Some popular public repositories are as follows:

- **Docker Hub**: The most popular repository. When **Docker** became popular, it was the main repository for public images to be used.
- **Quay**: A open-source repository with the possibility to deploy as on-promise.

An **Image Registry** enables users to build, organize, distribute and deploy containers. *Figure 13.4* shows an example workflow where the developer builds an **Image** and publishes it to an **Image Registry**. After the automatic tests are passed, they can be released to production to be used by the end users.

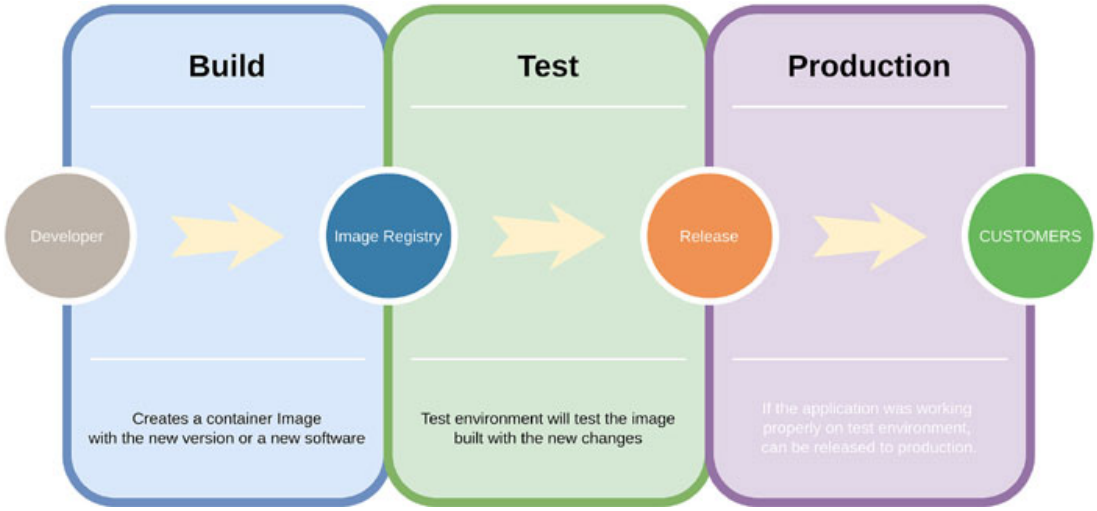


Figure 13.4: *Image registry workflow example*

Docker

The project **Docker** was a revolution in the IT Industry because it made it easier to implement **Containers**. Traditionally, **Virtualization** was a fresh air of change for the companies to reduce costs and complexity in their infrastructures. However, it did not solve all the problems related to the development of the applications and the systems that were deployed. The introduction of the **Containers** in the development life-cycle helped *Developers* to be able to perform their tasks without the need for the creation of specific environments. Using the solution provided by **Docker**, they were able to use their own workstation or a server provided by the *IT team* to run their own applications

inside **Containers** and generate **Images** that would be promoted to a different environment (such as *testing, pre-production, or production).*

The popularity and implementation of **Containers** also helped the developers be able to isolate their applications and create *Microservices,* as described previously. This made the development team more flexible and less dependent on the infrastructure team. A new role named *DevOps* appeared with the implementation of **Containers** and new practices focused on the *Cloud.*

Docker is based on a *Daemon* running, which will monitor the status of the *Containers* running in the system. This further allows it to start or stop the containers, among other tasks. **Docker** is available on Linux, Windows, and Mac systems. **Docker** has two main products:

- **Docker engine:** is an open-source containerization technology for building and containerizing your applications.
- **Docker desktop:** is an application for MacOS and Windows machines for the building and sharing of containerized applications and microservices.

The steps for **Docker Engine** installation in different Linux distributions are detailed in the following link: **https://docs.docker.com/engine/install/.** After the installation, a service called *docker.service* is available in the system to start, stop, or obtain status. When the **Docker Daemon** is started, it is possible to use the command *docker* to perform different tasks with containers and images.

Run the first container

The first command is to run a sample container. Refer to the *figure 13.5:*

```
root@ubuntu:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:e18f0a777aefabe047a671ab3ec3eed05414477c951ab1a6f352a06974245fe7
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figure 13.5: *First container execution with Docker*

Obtain information about client and server

The command **docker** allows, as the first argument, the word *run* to execute a container. A **Container** is based on an **Image**, and the argument *hello-world* indicates the image to be used. The first line of the output indicates that the **Image** was not downloaded previously. The **Image** is downloaded automatically, and then the **Container** is created using it. The rest of the output comes from the execution. This container will be automatically stopped after showing the message.

The command **`docker info`** shows information about the client and the server. The following code shows an example output:

```
root@ubuntu:~# docker info
Client:
Context:    default
Debug Mode: false
Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Docker Buildx (Docker Inc., v0.9.1-docker)
  compose: Docker Compose (Docker Inc., v2.12.2)
  scan: Docker Scan (Docker Inc., v0.21.0)

Server:
```

```
Containers: 1
 Running: 0
 Paused: 0
 Stopped: 1
Images: 1
Server Version: 20.10.21
Storage Driver: overlay2
 Backing Filesystem: extfs
(omitted)
Logging Driver: json-file
Cgroup Driver: systemd
Cgroup Version: 2
Plugins:
 Volume: local
 Network: bridge host ipvlan macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file local logentries
 splunk syslog
Swarm: inactive
Runtimes: io.containerd.runc.v2 io.containerd.runtime.v1.linux runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 1c90a442489720eec95342e1789ee8a5e1b9536f
runc version: v1.1.4-0-g5fd4c4d
init version: de40ad0
(omitted)
Kernel Version: 5.15.0-52-generic
Operating System: Ubuntu 22.04.1 LTS
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 3.832GiB
Name: ubuntu
ID: SEHR:ROON:MTHA:DFD2:A4S3:56BX:IDC6:NP34:SZO3:7WYD:XYC4:3S5Z
Docker Root Dir: /var/lib/docker
Debug Mode: false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false
```

The preceding output provides useful information about the **Docker Client** and the **Docker Daemon :**

- The client has enabled the plugins: **`app`** (used to run containers), **`buildx`** (used to generate images), **`compose`** (used for orchestration), and **`scan`** (to check vulnerabilities on local images).
- The server has one container and is stopped (the previous example executed); it uses the storage driver called **`overlay2`** and the driver **`json-file`** for the logging.
- The output lists the available plugins for volumes, network, and logging.
- The server uses the runtime named **`containerd`**. The data will be stored in the directory **`/var/lib/docker`**.
- The output shows information about the system where it is running, such as the distribution version, *Kernel* version, available CPUs, and memory.

Operate with containers

The argument **`ps`** for the command **`docker`** will list the **Containers** running in the system. Adding the option *-a* will display the *containers*, which are stopped. *Figure 13.6* shows an output example:

```
root@ubuntu:~# docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS
  PORTS     NAMES
1f37b3155cc6   hello-world   "/hello"   26 minutes ago   Exited (0) 26 minutes ago
            agitated_nightingale
```

Figure 13.6: *Output example for listing containers*

The previous output includes the following columns:

- **Container ID**: When a *container* is created, an auto-generated *ID* is assigned to it. The length is 64, but the output shows the first 12 characters.
- **Image**: The *image* used to run the *container*.
- **Command**: The *command* executed inside of the container.
- **Created**: When the container was created.
- **Status**: The status of the container; if it was exited, it will indicate when it was stopped, and the status (*0* indicates the *container* finished correctly).
- **Ports**: Forwarded ports to access the service offered by the *container*. In the previous example, no ports were exposed.
- **Names**: The name of the container. If a name is not specified during the execution, an auto-generated one is assigned.

An **Image** has configured a command to be executed when it is referenced by a container. Using **`docker run`**, it is possible to specify the command to be used instead

of the default one. With the option **`--name`**, it is possible to specify a custom name. *Figure 13.7* shows an example using the image *debian* and setting the name of the container to *mydeb*.

```
root@ubuntu:~# docker run --name mydeb debian cat /etc/debian_version
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
17c9e6141fdb: Pull complete

Digest: sha256:bfe6615d017d1eebe19f349669de58cda36c668ef916e618be78071513c690e5
Status: Downloaded newer image for debian:latest
11.5
```

Figure 13.7: *Running a container with a custom name and command*

The argument **`ps`** for the command **`docker`** allows the option *-l* to list the information about the last container executed, as is shown in *figure 13.8*:

```
root@ubuntu:~# docker ps  -l
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS
                    PORTS     NAMES
8af735ee634c   debian    "cat /etc/debian_ver…"   About a minute ago   Exited (0)
 About a minute ago             mydeb
```

Figure 13.8: *Example output for the argument ps with the option -l*

It is possible to interact with a container after creating it. This is useful for troubleshooting. The options used for this purpose are as follows:

- **`--tty (-t)`***:* Allocate a pseudo-TTY
- **`--interactive (-i)`**: Keeps **standard input** (**STDIN**) open.

Figure 13.9 shows how to execute a new container and operate inside a shell console:

```
root@ubuntu:~# docker run --name testconsole -ti debian /bin/bash
root@e67b41d7cc70:/# cat /etc/debian_version
11.5
```

Figure 13.9: *Example using the options -ti with the argument run*

The *hostname* of the pod is assigned to the auto-generated ID for the pod. It is possible to specify the *hostname* for the pod with the option **`--hostname (-h)`**. Exiting a container by writing the command **`exit`** in the shell, the *container* will be stopped after the session is closed. To disconnect from the *container* without stopping it, it is required to press the combination *Ctrl-P* followed by *Ctrl-Q*.

To access the logs generated by the *container*, the command **`docker`** and the argument **logs** are required. It is possible to specify the name of the *container* or the *ID* in long format (64 characters) or short format (12 characters). *Figure 13.10* shows the logs from the previous *container*:

```
root@ubuntu:~# docker logs testconsole
root@e67b41d7cc70:/# cat /etc/debian_version
11.5
root@ubuntu:~#
```

Figure 13.10: *Example obtaining the logs from a container*

The argument **`inspect`** for the command **`docker`**, followed by the name or identifier of the container, will result in a long output with the information. It will return information detailed information, such as the full *ID*, the command executed, the date when it was created, and the network and storage information, among other information, as is shown in the following code (output is truncated):

```
[
   {
   "Id":
   "df23a4a05d1d297c70ec1691564b48613293f33217c86309022b2d7fdd9cebd1",
   "Created": "2022-10-30T17:08:05.656823543Z",
   "Path": "/bin/bash",
   "Args": [],
   "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 9277,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2022-10-30T17:08:05.853482107Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
   },
   (omitted)
   "Name": "/testconsole",
   (omitted)
   "Config": {
      "Hostname": "e67b41d7cc70",
      (omitted)
      "Image": "debian",
      "Volumes": null,
      "WorkingDir": "",
     (omitted)
```

```
    },
    "NetworkSettings": {
        (omitted)
        "Gateway": "172.17.0.1",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "IPAddress": "172.17.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "MacAddress": "02:42:ac:11:00:02",
        "Networks": {
          "bridge": {
            (omitted)
          }
        }
    }
    }
]
```

The previous input shows that the container was based on the image *debian* and the command executed was **`/bin/bash`**. The IP assigned was 172.17.0.2, and the gateway was 172.17.0.1. The pod will have direct access to other running containers using the same network, and it will have access to the internet through the gateway.

A **Container** can be executed to perform an operation and be executed in the background. For this purpose, the option used is **`--detach (-d)`**. *Figure 13.11* shows an example of the execution of a **Container** to print the current time every 5 seconds, keeping the container running till it is stopped.

```
root@ubuntu:~# docker run -dti debian /bin/sh -c "while true; do date; sleep 5;done"
0875fc67319c4db3b795bb936f0663c7d8a5922229d07b897a3fc4d0a88662fb
```

Figure 13.11: *Example running a detached container*

Using the argument **`logs`** with the option **`--follow (-f)`**, it is possible to check the progress of the Container execution and the logs generated, as is shown in *figure 13.12*.

```
root@ubuntu:~# docker logs -f 0875fc67319c4db3b
Sun Oct 30 17:28:44 UTC 2022
Sun Oct 30 17:28:49 UTC 2022
Sun Oct 30 17:28:54 UTC 2022
Sun Oct 30 17:28:59 UTC 2022
```

Figure 13.12: *Example output logs for a container*

To stop a container, the argument used is, **`stop`**. **Docker** will wait 10 seconds till the application is stopped. If it is not stopped in that period, the **Container** will be forcefully stopped. It is possible to override that waiting time using the option **`--time (-t)`**, followed by the number of seconds. *Figure 13.13* shows how to stop the previous container:

```
root@ubuntu:~# docker stop 0875fc67319c4db3b
0875fc67319c4db3b
```

Figure 13.13: *Example use of the argument stop*

Using the argument **`ps`** with option **`-l`**, as indicated previously, it is possible to set the **`exit code`** for the container after being stopped. *Figure 13.14* shows the output example:

```
root@ubuntu:~# docker ps -l
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
                    PORTS     NAMES
0875fc67319c   debian    "/bin/sh -c 'while t…"   5 minutes ago   Exited (137) Ab
out a minute ago            blissful_wing
```

Figure 13.14: *Example exit code for a container*

When the exit code is greater than 128, it means that it was unexpectedly stopped. In the previous example, the code 137 means it was stopped forcefully (137–129 = 9) with a signal *SIGKILL (*9). It is important to remember that the *IP* assigned to the container will be released when it is stopped and can be reused by another one.

The argument **`start`** for the command **`docker`** allows restarting a stopped **Container**. When it is started, a new *IP address* from the available ones will be assigned.

Exposing containers

By default, a **Container** is only internally accessible from the *host* (where the **Docker Daemon** is) where it is running. When **Docker** is installed, a private network called *bridge* is created and used by default. The subnet configured by default is 172.17.0.0/16. *Figure 13.15* shows how to run a container using the image **`httpd`**, which contains a default installation of *Apache2*.

```
root@ubuntu:~# docker run --name mywebserver -dti httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
e9995326b091: Pull complete
ee55ccd48c8f: Pull complete
bc66ebea7efe: Pull complete
5d0f831d3c0b: Pull complete
e559e5380898: Pull complete
Digest: sha256:5fa96551b61359de5dfb7fd8c9e97e4153232eb520a8e883e2f47fc80dbfc33e
Status: Downloaded newer image for httpd:latest
c6812ed6469ba37a0b3192f3126e4e05497c39306087a0a4b29cbf4071cb0c4e
```

Figure 13.15: *Example accessing a container*

After the **Image** is downloaded from the public repository and the *container* is executed, it is possible to inspect the container to obtain the IP assigned, as is shown in *figure 13.16*:

```
root@ubuntu:~# docker inspect mywebserver | grep -m1 '"IPAddress"'
            "IPAddress": "172.17.0.2",
root@ubuntu:~#
```

Figure 13.16: *Example accessing a container*

Using that *IP,* it is possible to access the *host* where **Docker** is running. As this **Container** is a Web server, it is possible to access using the tool *curl*. *Figure 13.17* shows an example:

```
root@ubuntu:~# curl 172.17.0.2
<html><body><h1>It works!</h1></body></html>
root@ubuntu:~#
```

Figure 13.17: *Example accessing the container*

In many services that will be executed inside a platform with **Containers**, we will want to expose the service externally. The command **`docker`** with the argument **`run`** allows two options to externally expose a *container*:

- **`--publish-all (-P)`**: Publish all exposed ports to random ports.
- **`--publish (p)`**: Manually specifies a list of the ports to be exposed to the *host*.

With the first option, **Docker** will review the ports configured in the image to be exposed, and it will assign a random port to it. In the following example shown in *figure 13.18*, a new **`container`** using the image **`httpd`** is executed with the option **`--publish-all`**.

```
root@ubuntu:~# docker run --publish-all --name mywebserver2 -dti httpd
260307b9f3a9b33fd677bb477191c149658467e08d60df5d2838f8b34a8af9c3
root@ubuntu:~# docker ps -l
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORT
S                                 NAMES
260307b9f3a9   httpd     "httpd-foreground"   3 seconds ago   Up 2 seconds   0.0.
0.0:49153->80/tcp, :::49153->80/tcp   mywebserver2
```

Figure 13.18: *Example publishing random ports of a container*

In the previous example, the random port *49153* in the *host* on the protocol *IPv4* and *IPv6* will redirect the request to port *80* of the **Container** named *mywebserver2* that is just created. Using the command **`curl`**, it is possible to test it, as shown in *figure 13.19*:

```
root@ubuntu:~# curl 192.168.122.43:49153
<html><body><h1>It works!</h1></body></html>
```

Figure 13.19: *Example accessing the container*

Along with the option **`--publish (-p)`**, the port in the host that will be used for the redirection, is also needed to be specified. Three formats are available, as shown in *table 13.1*:

Format	Description
`host_port:container_port`	Redirects the port `host_port` to `container_port`. All *IPS* available in the server will listen in the `host_port`.
`IP:port_host:port_container`	Similar to the previous format, but specifying only one *IP*, which will listen to the port `host_port`.
`IP::port_container`	In this format, the same port will be used in the *container* and in the *host*. For example, if the *container* exposes port 8080, the same port will be used in the *host*.

Table 13.1: *Formats to be used with the --publish (-p) option*

Figure 13.20 shows how to run a **Container** and listen in port 8080 in the host, and redirect the request to port 80 in the container.

```
root@ubuntu:~# docker run --publish 8080:80 --name mywebserver3 -dti httpd
a781f01965573a9a33ad6b5ae93333cacb688625376c2650c57c074fd9fccb33
root@ubuntu:~# curl http://192.168.122.43:8080
<html><body><h1>It works!</h1></body></html>
```

Figure 13.20: *Example publishing a specific port of a container*

It is also possible to use the argument **`port`**, followed by the name or identification of the container, to list the ports exposed, as is shown in *figure 13.21*:

```
root@ubuntu:~# docker port mywebserver3
80/tcp -> 0.0.0.0:8080
80/tcp -> :::8080
```

Figure 13.21: *Example output for port argument*

The communication redirection between the *host* and the **Container** is made using **iptables.** *Figure 13.22* shows how to list the rules in the table **`nat`** used by **Docker:**

```
root@ubuntu:~# iptables -t nat -L DOCKER -v -n
Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination

    0     0 RETURN     all  --  docker0 *       0.0.0.0/0            0.0.0.0/0

    1    60 DNAT       tcp  --  !docker0 *       0.0.0.0/0            0.0.0.0/0
          tcp dpt:49153 to:172.17.0.3:80
    1    60 DNAT       tcp  --  !docker0 *       0.0.0.0/0            0.0.0.0/0
          tcp dpt:8080 to:172.17.0.4:80
```

Figure 13.22: *Example output for iptables rules related to Docker*

Container actions

The command **Docker** contains many actions, which can be performed to operate with containers. The following list shows the most used arguments and an example for each of them.

- **`create`**: Allows the creation of a container without executing it. Useful to predefined containers before executing them. Refer to *figure 13.23*:

```
root@ubuntu:~# docker create --publish 8081:80 --name mywebserver4  httpd
baf0feb8087af86f25df3ca639d3e5c7c373b01108fea82f6cde303ea2db18ea
root@ubuntu:~# docker ps -l
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS    PORTS
 NAMES
baf0feb8087a   httpd     "httpd-foreground"   3 seconds ago   Created
 mywebserver4
```

Figure 13.23: *Example of creating a new container*

- **restart**: It stops and starts a *container* that was running. It is possible to specify the option **`--time (-t)`** to indicate how many seconds to wait before stopping it forcefully. Refer to *figure 13.24*:

```
root@ubuntu:~# docker restart mywebserver3
mywebserver3
```

Figure 13.24: *Example restarting a container*

- **rename**: It renames a container to a new name. Refer to *figure 13.25*:

```
root@ubuntu:~# docker rename mywebserver4 web04
root@ubuntu:~# docker ps -l
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS    PORTS
 NAMES
baf0feb8087a   httpd     "httpd-foreground"   3 minutes ago   Created
 web04
```

Figure 13.25: *Example renaming a container*

- **pause/unpause:** Pauses or resumes the processes running inside the container specified. Refer to *figure 13.26*:

```
root@ubuntu:~# docker ps -f name=mywebserver3
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS
      PORTS                                  NAMES
a781f0196557   httpd     "httpd-foreground"   58 minutes ago   Up 3 minutes (Paus
ed)   0.0.0.0:8080->80/tcp, :::8080->80/tcp   mywebserver3
root@ubuntu:~# docker unpause mywebserver3
mywebserver3
root@ubuntu:~# docker ps -f name=mywebserver3
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS          POR
TS                                NAMES
a781f0196557   httpd     "httpd-foreground"   58 minutes ago   Up 4 minutes    0.0
.0.0:8080->80/tcp, :::8080->80/tcp   mywebserver3
```

Figure 13.26: *Example pausing and resuming a container*

- **kill:** It stops the container forcefully. The option possible is **`--signal (-s)`** to specify the signal to be used; the default is *SIGKILL*. Refer to *figure 13.27*:

```
root@ubuntu:~# docker kill mywebserver3
mywebserver3
root@ubuntu:~# docker ps -f name=mywebserver3 -a
CONTAINER ID   IMAGE     COMMAND              CREATED       STATUS
       PORTS     NAMES
a781f0196557   httpd     "httpd-foreground"   2 hours ago   Exited (137) 1 second
 ago             mywebserver3
```

Figure 13.27: *Example of killing a container*

- **top:** Shows the list of the processes with detailed information running inside of the *container*. It is possible to add the options of the **`ps`** command after the name or identification of the *container*. Refer to *figure 13.28*:

```
root@ubuntu:~# docker top mywebserver2 -o user,pid,%cpu,command
USER                PID                 %CPU                COMMAND
root                10993               0.0                 httpd -DFOREGROUND
www-data            11023               0.0                 httpd -DFOREGROUND
www-data            11024               0.0                 httpd -DFOREGROUND
www-data            11025               0.0                 httpd -DFOREGROUND
```

Figure 13.28: *Example using the argument top*

- **rm**: It removes a *container* that is stopped unless the option **`--force (-f)`** is specified. With the option **`--volumes (-v)`**, it will delete the volumes (discussed as follows) associated with the *container*. Refer to *figure 13.29*:

```
root@ubuntu:~# docker stop mywebserver2; docker rm mywebserver2
mywebserver2
mywebserver2
```

Figure 13.29: *Example of removing a container*

- **exec**: It executes a command inside a running *container*. Refer to *figure 13.30*:

```
root@ubuntu:~# docker rename mywebserver web
root@ubuntu:~# docker exec web grep ^Listen /usr/local/apache2/conf/httpd.conf
Listen 80
```

Figure 13.30: *Example of executing a command in a running container*

- The popular options are as follows:
  - **`--detach (-d)`**: executes the command in the background.
  - **`--interactive (-i)`**: interactive mode to introduce commands
  - **`--tty (-t)`**: allocates a pseudo-TTY.

```
root@ubuntu:~# docker exec -ti web /bin/bash
root@c6812ed6469b:/usr/local/apache2# grep ^DocumentRoot conf/httpd.conf
DocumentRoot "/usr/local/apache2/htdocs"
root@c6812ed6469b:/usr/local/apache2#
```

Figure 13.31: *Example accessing a container*

- **export/import:** The argument *export* makes a backup of a running or stopped container, and it will generate a *tar* file with the content. Refer to *figure 13.32*:

```
root@ubuntu:~# docker export web -o web.tar
root@ubuntu:~# ls -lh web.tar
-rw------- 1 root root 140M Oct 30 20:52 web.tar
```

Figure 13.32: *Example exporting a container*

To import a *container*, we first need to create an *image* from the backup and then create a container using that *image*. Refer to *figure 13.33*:

```
root@ubuntu:~# docker import web.tar mywebimage:0.1
sha256:3508309e393de8538254463727e6e60ca8dfdc62717c63e2e876662ebc3c4c51
root@ubuntu:~# docker run -ti mywebimage:0.1 /usr/local/apache2/bin/httpd -v
Server version: Apache/2.4.54 (Unix)
Server built:   Oct 25 2022 03:59:05
```

Figure 13.33: *Example importing a container as an image*

- **inspect:** This argument was described previously; the option **`--format (-f)`** allows access to specific data from detailed information. The required format is **`{{ .Category.Element`** *Figure 13.34* shows some examples:

```
root@ubuntu:~# docker inspect -f "{{ .Config.Hostname }}" web
c6812ed6469b
root@ubuntu:~# docker inspect -f "{{ .NetworkSettings.IPAddress }}" web
172.17.0.2
root@ubuntu:~# docker inspect -f "{{ .NetworkSettings.MacAddress }}" web
02:42:ac:11:00:02
root@ubuntu:~# docker inspect -f "{{ .State.Status }}" web
running
```

Figure 13.34: *Example of inspecting a container*

- By default, the data inside the container is lost when the container is removed. It is possible to have persistent storage inside the *containers*. Using the option **`--volume (-v)`**, it is possible to specify a directory from the server where **Docker Daemon** is running and is going to be mounted inside the *container*. *Figure 13.35* shows us how to use a directory in the *host* to store the *HTML* files to be used by the container.

```
root@ubuntu:~# mkdir /web/
root@ubuntu:~# echo "File stored in the host" > /web/index.html
root@ubuntu:~# docker run -dtiv /web:/usr/local/apache2/htdocs -p 8081:80 httpd
4ef4b6434ed1bf2ba8c0a64f3d752fe8ddb7d1ea06511e117d940697d65aea7f
root@ubuntu:~# docker inspect -f '{{ .Mounts }}' 4ef4b6434ed1bf2
[{bind  /web /usr/local/apache2/htdocs   true rprivate}]
root@ubuntu:~# curl -s http://localhost:8081
File stored in the host
```

Figure 13.35: *Example mounting a directory from the host to the container*

- The command **`docker`** has the argument **`volume`** to list, create and remove volumes from the system. *Figure 13.36* shows how to create a volume, which can be accessible from the directory **`/var/lib/docker/volumes/`**, as well as how to use the option **`--volume`** to specify it to be mounted on the container.

```
root@ubuntu:~# docker volume create myweb
myweb
root@ubuntu:~# echo "From vol" > /var/lib/docker/volumes/myweb/_data/index.html
root@ubuntu:~# docker run -dtiv myweb:/usr/local/apache2/htdocs -p 8082:80 httpd
e60bffcd80ab31af456dfe217b0e10de5550bb24b800d5a4232dd07757c65a38
root@ubuntu:~# curl -s http://localhost:8082
From vol
```

Figure 13.36: *Example of creating a volume and specifying it during container creation*

Docker server statistics and events

From the **Docker Client** (command **`docker`**), it is possible to obtain the statistics of the usage of the **Docker Daemon**. The argument **`stats`** can be used to get statistics of the running **Containers**, all the **Containers** **`(--all/-a)`**, or a specific list of the containers desired. *Figure 13.37* shows the output example:

```
root@ubuntu:~# docker stats --no-stream
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O
   BLOCK I/O     PIDS
c6812ed6469b   web       0.00%     16.34MiB / 3.832GiB   0.42%     10.6kB / 626B
   0B / 24.6kB   82
```

Figure 13.37: *Example output using the argument stats*

The client also provides the option to visualize the events generated or see in real-time what is happening in the server. The argument used is *events,* and the options are as follows:

- **`--filter (-f)`**: Filter the output with the specified filter.

- `--since`: Shows the records from the specified date.
- `--until`: Shows the records until the specified date.

If no options are specified, the client will wait for new events to appear, and it will show them on the screen. The option `--filter (-f)` allows the different filters, the most common ones being:

- **container**=*container*: Filter by the container (name or identifier).
- **event**=*action*: Filter by event type.
- **image**=*image*: Filter by image (name or identifier).
- **label**=*key* or **label**=*value*: Filter by a label.
- **type**=*[container | image | volume | network | daemon]*: Filter for the objected which generated the event.
- **volume**=*volume*: Filter by volume (name or identifier)
- **network**=*network*: Filter by internal **Docker** network (name or identifier).
- **daemon**=*service*: Filter by a service (name or identifier).

Figure 13.38 shows an example of filtering for a specific period and the tasks related to the *image*, showing the imported image example described previously.

```
root@ubuntu:~# docker events --since '2022-10-30T20:56:37' --filter type=image
2022-10-30T20:56:37.602250603Z image tag sha256:3508309e393de8538254463727e6e60c
a8dfdc62717c63e2e876662ebc3c4c51 (name=mywebimage:0.1)
2022-10-30T20:56:37.602284773Z image import sha256:3508309e393de8538254463727e6e
60ca8dfdc62717c63e2e876662ebc3c4c51 (name=sha256:3508309e393de8538254463727e6e60
ca8dfdc62717c63e2e876662ebc3c4c51)
```

Figure 13.38: *Example of output using the argument events*

Image registry

Docker allows to create custom images and publish them in a private or public **Image Registry**. An **Image** can be stored locally in order to be published to the registry. An **Image** has a name and a *tag*. The *tag* helps to specify a version of an application or base system, such as `httpd:2.4` or `ubuntu:22.04`. The *tag* can also specify a string to specify name-based version `ubuntu:jammy` or `ubuntu:latest` (to always use the latest image available for *Ubuntu*). In creating an image, the user can specify the *tag* desired without any restraint.

To generate an image, there are the following two main options:

- **Convert a container to an image**: This generation is a manual task and requires to configure a **Container**. This is the less recommended option. The following two options are available using **docker** client:
    - Using **export** and **import** arguments as described previously.

- Using argument **commit** and **tag**. *Figure 13.39* shows an example:

```
root@ubuntu:~# docker commit mywebserver3
sha256:619992f21a8ebcdda4cef9ee2d6e20b0139092268f39cf9f19dd393e83f204d
root@ubuntu:~# docker tag 619992f21a8e imagewebserver3:0.1
root@ubuntu:~# docker images | head -2
REPOSITORY        TAG       IMAGE ID       CREATED          SIZE
imagewebserver3   0.1       619992f21a8e   20 seconds ago   145MB
```

Figure 13.39: *Example of using the argument commit*

- **Using a template**: A file usually named **`Dockerfile`** is used as a reference for the argument **`build`** for the command **`docker`**.

An example of a **`Dockerfile`** is as follows:

```
FROM ubuntu:latest
RUN apt-get update &&  apt-get install -y apache2
RUN echo "Image generated" > /var/www/html/index.html
COPY test.html /var/www/html/test.html
EXPOSE 80
CMD /usr/sbin/apache2ctl -D FOREGROUND
```

The instructions used in the previous example are as follows:

- **FROM**: Indicates the base image to be configured. A valid **`Dockerfile`** must start with this instruction.
- **RUN**: Executes commands inside the base image. Each **RUN** generates a *layer* for the image; it is recommended to group them and reduce the number of these instructions.
- **COPY**: Copy files to the inside of the base image. The file needs to be relative to the directory where the **`Dockerfile`** is.
- **EXPOSE**: Specifies which port or ports will be exposed when the container using this image will use the option **`--publish-all (-P)`**.
- **CMD**: The default command to be executed when the container starts using this image.

To build an **Image** from a **`Dockerfile`** template, the command **`docker`** with the argument **`build`** is used. The option to specify the target name image and the *tag* is the option **`--tag (-t)`**. The argument **`build`** requires a context directory where the **`Dockerfile`** is located. *Figure 13.40* shows the building process of the previous example:

```
root@ubuntu:~# docker build /root/ubuntuapache2/ -t ubuntuapache2:0.1
Sending build context to Docker daemon  3.072kB
Step 1/6 : FROM ubuntu:latest
 ---> cdb68b455a14
Step 2/6 : RUN apt-get update &&  apt-get install -y apache2
 ---> Using cache
 ---> e456d9e2e1f3
Step 3/6 : RUN echo "Image generated" > /var/www/html/index.html
 ---> Using cache
 ---> dd122fabfbfc
Step 4/6 : COPY test.html /var/www/html/test.html
 ---> Using cache
 ---> 99dfd22d79d5
Step 5/6 : EXPOSE 80
 ---> Using cache
 ---> 14efd05a5ac0
Step 6/6 : CMD /usr/sbin/apache2ctl -D FOREGROUND
 ---> Running in 50b7a218682d
Removing intermediate container 50b7a218682d
 ---> f1715312940c
Successfully built f1715312940c
Successfully tagged ubuntuapache2:0.1
```

Figure 13.40: *Example of building a new Image*

The argument **`history`** followed by an image will show the layers and the task performed in each of them. *Figure 13.41* shows the example with the image built previously.

```
root@ubuntu:~# docker history ubuntuapache2:0.1
IMAGE          CREATED       CREATED BY                                      SIZE
  COMMENT
f1715312940c   5 days ago    /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "/usr…   0B

14efd05a5ac0   5 days ago    /bin/sh -c #(nop)  EXPOSE 80                    0B

99dfd22d79d5   5 days ago    /bin/sh -c #(nop) COPY file:34ae0b59838c5612…   10B

dd122fabfbfc   5 days ago    /bin/sh -c echo "Image generated" > /var/www…   16B

e456d9e2e1f3   5 days ago    /bin/sh -c apt-get update &&  apt-get instal…   147MB

cdb68b455a14   13 days ago   /bin/sh -c #(nop)  CMD ["bash"]                 0B

<missing>      13 days ago   /bin/sh -c #(nop) ADD file:ba96f963bbfd429a0…   77.8MB
```

Figure 13.41: *Example output using the argument history*

The generated image will be available locally, and it is possible to use it to create a container based on it. *Figure 13.42* shows an example:

```
root@ubuntu:~# docker images | head -2
REPOSITORY      TAG       IMAGE ID       CREATED              SIZE
ubuntuapache2   0.1       f1715312940c   4 minutes ago        225MB
root@ubuntu:~# docker run -dti -p 8888:80 ubuntuapache2:0.1
856cc9dfb66eaf9dd5400ba69a953a28a4d039ce798c5dc21fa6002ac703fb9b
root@ubuntu:~# curl http://localhost:8888/
Image generated
root@ubuntu:~# curl http://localhost:8888/test.html
Test file
```

Figure 13.42: *Example of creating a Container from the generated Image*

Other useful statements available in a *Dockerfile* are described in *table 13.2*:

Statement	Description
`LABEL`	Adds metadata to an image. Information such as the maintainer, the software version, or any other information.
`ADD`	Adds files to the image. The differences with *COPY* are two: • If a directory is specified, only the content is copied and not the parent directory. • If the file to be added is a compressed file, it will be automatically uncompressed
`ENTRYPOINT`	By default, containers are used to execute the commands specified using `/bin/sh`. With this statement, it is possible to change this behavior, specifying other commands as an entry.
`VOLUME`	Specifies the volumes to be created to have persistent storage.
`USER`	Specifies the user to be used to execute the commands specified in the `Dockerfile`.
`WORKDIR`	Sets the working directory inside of the container.
`ENV`	Sets the environment variables inside of the image.
`ARG`	Specifies the arguments which can be accepted using the option `--build-args`
`HEALTHCHECK`	Specifies health check tests to ensure the container is working properly.

Table 13.2: *Statements available in Dockerfile*

After the **Image** is generated, it is possible to publish it to a public *registry*, such as **https://hub.docker.com** or **https://www.quay.io**. It is possible to create a free account for personal usage and store-created images. Other plans for companies are available. After an account is created, it is possible to use the argument **`login`** from the command **`docker`** to perform a login before to publish the desired **`Image`**. *Figure 13.43* shows the login example to **Docker Hub**.

```
root@ubuntu:~# docker login docker.io
Login with your Docker ID to push and pull images from Docker Hub. If you don't h
ave a Docker ID, head over to https://hub.docker.com to create one.
Username: linuxservercb
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Figure 13.43: *Example of logging into Docker Hub*

After the login succeeds, the **Image** needs to be *tagged* with the correct format for the desired repository. For **Docker Hub,** the format is **`docker.io/USERNAME/nameimage:tag`**. After the **Image** has the correct name, the argument **`publish`** will upload and create the image in the repository. *Figure 13.44* shows the tagging and the publishing task:

```
root@ubuntu:~# docker tag ubuntuapache2:0.1 docker.io/linuxservercb/ubuntuapache
root@ubuntu:~# docker push docker.io/linuxservercb/ubuntuapache
Using default tag: latest
The push refers to repository [docker.io/linuxservercb/ubuntuapache]
ddf96998d2d7: Pushed
0ff7c267b89e: Pushed
bcebbf94eb56: Pushed
7ea4455e747e: Mounted from library/ubuntu
latest: digest: sha256:58557aced96f8d93717829e2ba6e418e15ea0992e4c83c8659f6f21cea
d26381 size: 1155
```

Figure 13.44: *Example of pushing an Image to Docker Hub*

After the image is published, the *image* is available in the **Docker Hub** portal. As the image did not specify any *image tag*, it is tagged locally and in the *registry* with the word *latest*. *Figure 13.45* shows the *Docker Hub* portal for the image published:

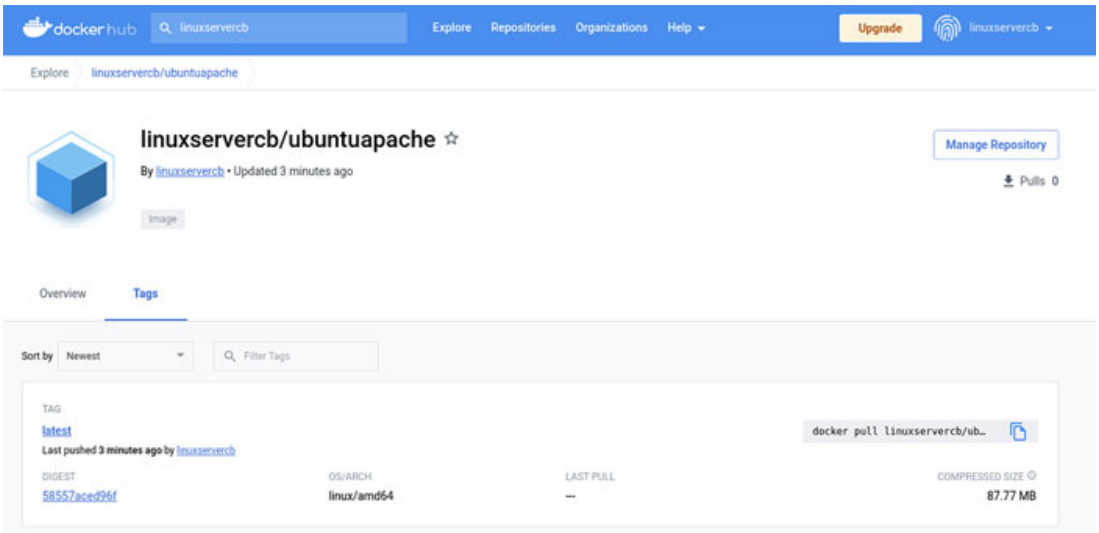


Figure 13.45: *Example of Docker Hub portal*

This image can be downloaded from another system, as can be observed in *figure 13.46*:

```
[root@rhel ~]# docker pull linuxservercb/ubuntuapache
Using default tag: latest
latest: Pulling from linuxservercb/ubuntuapache
301a8b74f71f: Pull complete
ceabe2bc017b: Pull complete
ec49d2122720: Pull complete
76f631053f2e: Pull complete
Digest: sha256:58557aced96f8d93717829e2ba6e418e15ea0992e4c83c8659f6f21cead26381
Status: Downloaded newer image for linuxservercb/ubuntuapache:latest
docker.io/linuxservercb/ubuntuapache:latest
```

Figure 13.46: *Example of output pulling an image from the public repository*

An **Image**, as a **Container**, can be inspected to get detailed information about it. With the arguments **`image inspect`**, followed by the *Image Id* or the name, the information will be shown on the screen. The following code (truncated) shows the output for the previously downloaded image:

```
[
   {
   "Id":
   "sha256:f1715312940c0c43b4c5d7b01d78a4106d4532df429611fd993887da6de71
   584",
   "RepoTags": [
      "linuxservercb/ubuntuapache:latest"
   ],
   "": [
      "lRepoDigestsinuxservercb/ubuntuapache@sha256:58557aced96f8d937178
      29e2ba6e418e15ea0992e4c83c8659f6f21cead26381"
   ],
   (omitted)
   "Config": {
      (omitted)
      "Cmd": [
        "/bin/sh",
        "-c",
        "/usr/sbin/apache2ctl -D FOREGROUND"
      ],
      "Image":
      "sha256:14efd05a5ac026c330095be5c099f8bcfd0b6c82da8dc2853c7aaeaf4e
      9c1e19",
   },
   "Architecture": "amd64",
   "Os": "linux",
```

```
    "Size": 225124187,
    "VirtualSize": 225124187,
    (omitted)
    },
    "RootFS": {
       "Type": "layers",
       "Layers": [
          "sha256:7ea4455e747ead87d6cc1c4efaf3a79530a931a0856a9f9ce9ac2d8
          d45bd3c28",
          "sha256:bcebbf94eb56506a9f1f6bc38f5a6bbf1fc74d03af0cbb4eb137ad6
          e107279fa",
          "sha256:0ff7c267b89e80ad4571a737758fa48f57a80cbddc52351bbab565c
          70b875c77",
          "sha256:ddf96998d2d70c418c2abee60746ff92b58dffdd16fa68d3fd4cdca
          9366c5a85"
       ]
    },
    (omitted)
    }
]
```

There is the possibility to run a private **Image Registry**, to not publish globally the images generated. This is usually required for enterprises with custom applications developed internally. There are different alternatives to running a private **Image Registry**; the popular and easiest way is using the **Image** provided by **Docker**. The name and the version of the image is **`registry:2`**. To expose it outside the system, it is required to use an SSL certificate to accept connections from remote sources. The following code was used to generate a self-signed certificate:

```
openssl req  -newkey rsa:4096 -nodes -sha256 -keyout certs/domain.key \
  -addext "subjectAltName = DNS:registry.example.com" \
  -x509 -days 365 -out certs/domain.crt  \
-subj
"/C=ES/ST=Madrid/L=Madrid/O=example/OU=com/CN=registry.example.com"
```

The image registry was executed using **`docker`** with the argument **`run`**, using the image **`registry:2`**. The environment variables specify where the certificates are.

```
docker run -d --restart=always --name registry -v "$(pwd)"/certs:/certs
\
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
  -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
  -p 5000:5000   registry:2
```

To publish an **Image** to a dedicated **Image Registry**, it is required to specify the DNS name of the server or the IP, followed by the port where the service is running. *Figure*

13.47 shows how to tag and publish the image:

```
[root@rhel ~]# docker tag linuxservercb/ubuntuapache registry.example.com:5000/l
nuxservercb/ubuntuapache:latest
[root@rhel ~]# echo '{ "insecure-registries":["registry.example.com:5000"] }' > 
etc/docker/daemon.json
[root@rhel ~]# systemctl restart docker
[root@rhel ~]# docker push registry.example.com:5000/linuxservercb/ubuntuapache
Using default tag: latest
The push refers to repository [registry.example.com:5000/linuxservercb/ubuntuapa
he]
ddf96998d2d7: Pushed
0ff7c267b89e: Pushed
bcebbf94eb56: Pushed
7ea4455e747e: Pushed
latest: digest: sha256:58557aced96f8d93717829e2ba6e418e15ea0992e4c83c8659f6f21ce
d26381 size: 1155
```

Figure 13.47: *Example pushing an image to a local registry*

Non-official repositories should be added in the file **`/etc/docker/daemon.json`**. The daemon needs to be restarted to get the new configuration. After that, it is possible to publish and pull images from the registries configured.

Podman

Podman is an alternative to **Docker** that is getting popular on **Red Hat Enterprise Linux** systems and derivatives. The biggest difference between both is that **Podman** does not use a *Daemon* to run the containers. Each container is executed individually and is assigned to an individual *systemD* configuration. This allows the management of each container independently, and furthermore, containers will not be affected if the main *Daemon* is restarted or unexpectedly stopped, something which can happen in **Docker**.

Installing **Podman** in a system will replace **Docker** if it is installed and vice versa. Installation can be based on the Linux distribution repository, such as using **`apt`** or **`dnf`** commands. The command associated with **Podman** is called **`podman`**. This command has the same syntax as the **`docker`** command, with the exception of minor differences. The images to be used to run containers are the same as those used by **Docker**; these images use the format named **Open Container Iniatiative** (**OCI**). *Figure 13.48* shows the argument **`run`** to create a new container.

```
[root@server ~]# podman run -dti -p 80:80 httpd
✔ docker.io/library/httpd:latest
Trying to pull docker.io/library/httpd:latest...
Getting image source signatures
Copying blob e559e5380898 done
Copying blob bc66ebea7efe done
Copying blob ee55ccd48c8f done
Copying blob e9995326b091 done
Copying blob 5d0f831d3c0b done
Copying config fe8735c23e done
Writing manifest to image destination
Storing signatures
1668129b120866e8a0fc5bbc9111195d6d9d52c8b89466d1ec2e7eb579fbd2de
```

Figure 13.48: *Example running a container using podman*

Podman solution comes with two commands for the **Image** building and for the **Image** distribution: **`buildah`** and **`skopeo`**. These need to be installed individually, and also included in the software repository. The first command uses the **`Dockerfile`** templates to build the destination **`Image`** and the second command is a powerful tool to operate with **`Images`**. *Figure 13.49* shows the **`buildah`** command, using the same **`Dockerfile`** as previously for the **`docker build`**.

```
[root@server buildahexample]# buildah build -t ubuntuapache2:0.2 .
STEP 1/6: FROM ubuntu:latest
STEP 2/6: RUN apt-get update>/dev/null &&  apt-get install -y apache2 >/dev/null
debconf: delaying package configuration, since apt-utils is not installed
STEP 3/6: RUN echo "Image generated" > /var/www/html/index.html
STEP 4/6: COPY test.html /var/www/html/test.html
STEP 5/6: EXPOSE 80
STEP 6/6: CMD /usr/sbin/apache2ctl -D FOREGROUND
COMMIT ubuntuapache2:0.2
Getting image source signatures
Copying blob f4a670ac65b6 skipped: already exists
Copying blob 39f73f3db6d1 done
Copying config 7efad98509 done
Writing manifest to image destination
Storing signatures
--> 7efad98509c
Successfully tagged localhost/ubuntuapache2:0.2
7efad98509cf7729d8b8a0d9668f3cea8a881c1555008f40e7030639bbed1f1b
```

Figure 13.49: *Example using command buildah*

The tool **`skopeo`** allows to manipulate, inspect, sign, and transfer container images between different repositories. *Figure 13.50* shows how to use **`skopeo`** to inspect an image located in a repository:

```
[root@server ~]# skopeo inspect docker://docker.io/linuxservercb/ubuntuapache:latest
{
    "Name": "docker.io/linuxservercb/ubuntuapache",
    "Digest": "sha256:58557aced96f8d93717829e2ba6e418e15ea0992e4c83c8659f6f21cead26381",
    "RepoTags": [
        "latest"
    ],
    "Created": "2022-11-01T21:27:15.186242546Z",
    "DockerVersion": "20.10.21",
    "Labels": null,
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
        "sha256:301a8b74f71f85f3a31e9c7e7fedd5b001ead5bcf895bc2911c1d260e06bd987",
        "sha256:ceabe2bc017bb08e2c992e1d18ed5503cefffc2b361e2475b3918c37087e25c2",
        "sha256:ec49d2122720a942642eda30ed278620e8261db72dee5878322e904540b23881",
        "sha256:76f631053f2eef7b53ca36a172e5bdad5a172efb92a1ff244ef53a503f11b9d9"
    ],
    "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ]
}
```

Figure 13.50: *Example using command skopeo*

Another useful task using **`skopeo`** is to copy images between different repositories. It is useful in having a disconnected repository from some official one to be able to do installations on environments that do not have direct access to the internet. *Figure 13.51* shows an example to copy the image **`ubuntu:latest`** to a private repository:

```
[root@server ~]# skopeo copy docker://docker.io/ubuntu:latest \
>  docker://registry.example.com:5000/ubuntu:latest \
>  --dest-tls-verify=false
Getting image source signatures
Copying blob e96e057aae67 done
Copying config a8780b506f done
Writing manifest to image destination
Storing signatures
```

Figure 13.51: *Example using command skopeo*

Container runtimes

As described previously, **Docker** and **Podman** are solutions to run containers on different platforms. Underneath, they are using a **Container Runtime** for that purpose. They are also responsible for network attachments, low-level storage, image transfer, and execution supervision.

There are different options available as follows:

- **runc**: a CLI tool for spawning and running containers on Linux, according to the OCI specification. This is the default one used by **Docker**.
- **crun**: A fast and low-memory footprint *OCI* **Container Runtime** fully written in C. This is the default one used by **Podman.**

- **Mirantis Container Runtime (MCR)**: Formerly called *Docker Engine—Enterprise*, it is a runtime supported by the company **Mirantis** after buying **Docker Enterprise.**

A **Container Runtime Interface** (**CRI**) is a plugin that allows **Kubernetes** to use a wide variety of **Container Runtimes**. The two main popular ones are as follows:

- `containerd`: This runtime is available as a daemon for Linux and Windows, which can manage the complete container life cycle of its host system.
- **CRI-O**: A lightweight container runtime for **Kubernetes.**

Kubernetes

If **Docker** revolutionized the IT industry easing the container adoption and the modernization of the applications, **Kubernetes** went further and put the industry upside down in the way of working, deploying applications, and how the application modernization should be. **Kubernetes** offered container orchestration, whereas **Docker** tried with the project **Docker Swarm.** However, the limitations and the lack of functionalities converted **Kubernetes** into the perfect container orchestration solution.

Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. Kubernetes provides the following:

- **Service discovery and load balancing:** Can expose a container using the DNS name or using their own IP address.
- **Storage orchestration:** Allows you to automatically mount a storage system of your choice, such as local storage, public cloud providers, and more.
- **Automated rollouts and rollbacks:** It is possible to describe the desired state for your deployed containers using **Kubernetes**, and it can change the actual state to the desired state at a controlled rate.
- **Automatic container distribution: Kubernetes** consists of a cluster of nodes that it can use to run containerized tasks. Each container defines how much CPU and memory (RAM) it needs. **Kubernetes** can fit containers onto your nodes to make the best use of your resources.
- **Self-healing: Kubernetes** restarts containers that fail, replaces containers, kills containers that do not respond to your user-defined health check, and does not advertise them to clients until they are ready to serve.
- **Secret and configuration management** Kubernetes let you store and manage sensitive information, such as passwords, *OAuth tokens*, and *SSH keys*. You can deploy and update secrets and application configuration without rebuilding your container images and without exposing secrets in your stack configuration.

Kubernetes is a perfect platform for many use cases and enterprises of all sizes. But **Kubernetes** is not a **Platform-as-a-Service** *(***PaaS**), which includes all that is needed for the developers to build and run applications on it. In the market, *PaaS* solutions-based **Kubernetes** appeared, and it is distributed as *Kubernetes distributions*. The popular solutions are as follows:

- **Red Hat OpenShift Container Platform:** A full-stack automated operations and self-service provisioning for developers lets teams work together more efficiently to move ideas from development to production. The community-supported sibling project is named *OKD*.
- **Red Hat OpenShift Dedicated:** A managed Red Hat OpenShift offering on **Amazon Web Services** (**AWS**) and Google Cloud. Reduces operational complexity and focuses on building and scaling applications that add more value to your business with this turnkey application platform.
- **VMware Tanzu:** Provides a streamlined, self-service developer experience for any Kubernetes that fits a development team's preferred practices and workflows while automating the toil of infrastructure, packaging, and security.
- **Rancher:** A management tool to deploy and run clusters anywhere and on any provider.
- **Google Container Engine (GKE)**: A managed, production-ready environment for running containerized applications.
- **Amazon Elastic Kubernetes Service (EKS):** A managed service and certified Kubernetes conformant to run Kubernetes on AWS and on-premises.
- **Mirantis Kubernetes Engine**: Offers a fast way to deploy cloud-native applications at scale in any environment.
- **Canonical Kubernetes:** This is built on Ubuntu and combines security with optimal price performance.

Introduction to continuous integration/delivery

With the new work methodology introduced with the use of containers and new agile principles, two new concepts also appeared for the software life cycle:

- **Continuous Delivery (CD)**: Automates the entire software release process. The idea behind CD is that any time the developers commit the changes to the version control solution, the software is built and tested in the proper target. If the workflow is completed correctly, the new version is ready to be released. This approach reduces the time to go to production, reduces costs, and reduces the risk of releasing software with bugs or not working properly. *Figure 13.52* shows the workflow example:

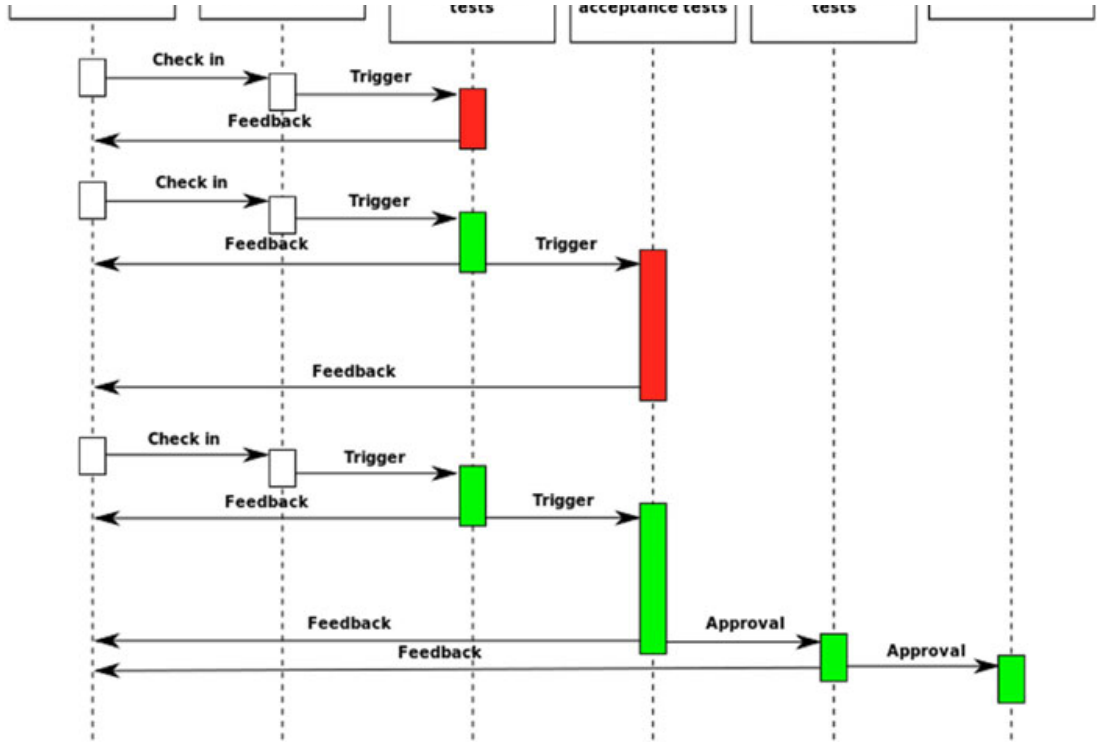


Figure 13.52: *Example using command skopeo*

- **Continuous Integration (CI)**: Software is usually written by developers who are working locally or in a dedicated development environment. When a new functionality is developed, it is published to the version control system, and it can be tested. **Continuous Integration** is the practice of regularly merging the copies from the developers to a common target to be used.

A concept named **Continuous Deployment** is referenced when the release to production is automatically done. There are many open-source projects available for **Continuous Delivery and Integration**, where the most popular are as follows:

- **Jenkins**: It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery.
- **CircleCI**: A continuous integration and continuous delivery platform that can be used to implement DevOps practices.
- **GoCD**: Helps to automate and streamline the build-test-release cycle for worry-free, continuous delivery of your **product**.
- **GitLab CI/CD**: You can automatically build, test, deploy, and monitor your applications by using *Auto DevOps*.

Other popular hosted-based solutions for **CI/CD** are the following:

- **Travis CI**: This is a hosted continuous integration service used to build and test software projects hosted on GitHub and Bitbucket.
- **GitHub Actions:** A **CI/CD** which allows to automate the build, test, and deployment pipeline. It is possible to create workflows that build and test every pull request to the repository or deploy merged pull requests to production.

Jenkins, Gitlab CI/CD, and Github actions

This section will cover three of the most important **CI/CD** solutions: **Jenkins**, **Gitlab CI/CD**, and **GitHub Actions.**

Jenkins

Jenkins became popular on the year 2011 due to its easy deployment, configuration, and integration with different source control versions, such as *Subversion* and *Git*. This tool helps to automate the building, testing, and deployment parts of the software development. For years, it was the most popular tool for **Continuous Integration** and **Continuous Delivery**, used by companies and regular users.

Installation of **Jenkins** is an easy and straightforward process, and the different installation methods are found on the official website: **https://www.jenkins.io/doc/book/installing/**. After the installation is complete, it is possible to perform all the configurations required using the website. *Figure 13.53* shows the dashboard after installation:

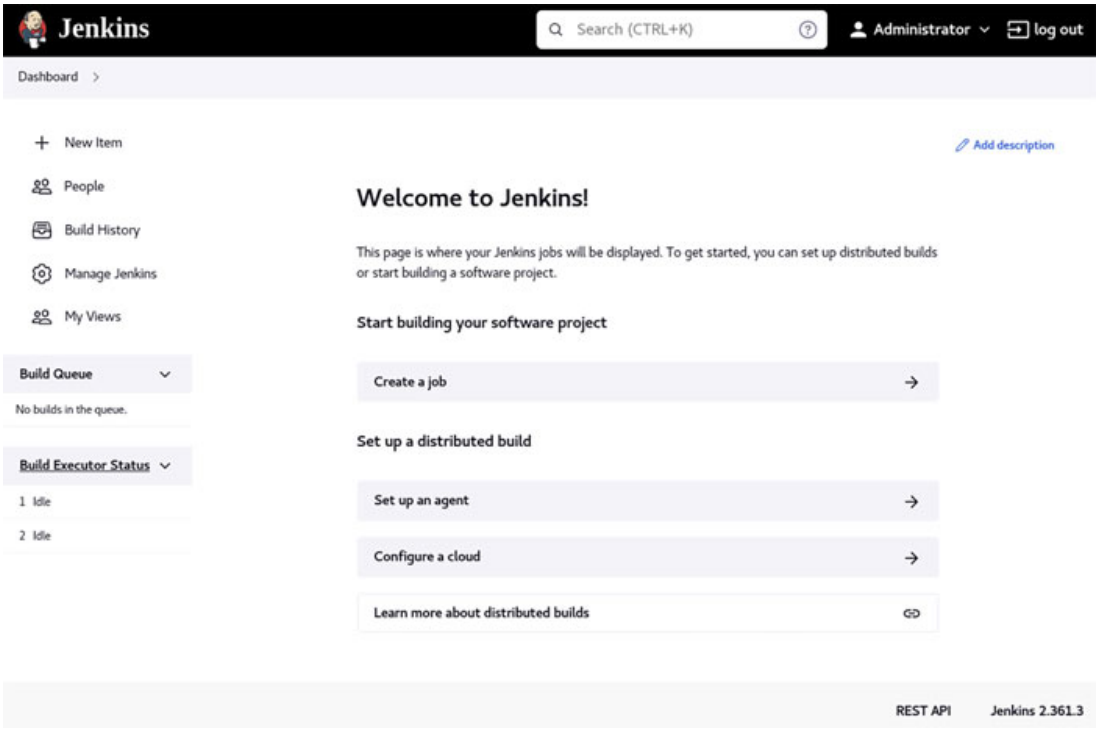


Figure 13.53: *Jenkins dashboard*

On creating a new job, a *Wizard* will appear to specify which project type is going to be created, the name, and the different options. *Figure 13.54* shows the options available where the **Pipelines** is required when a **CI/CD** workflow will be created.

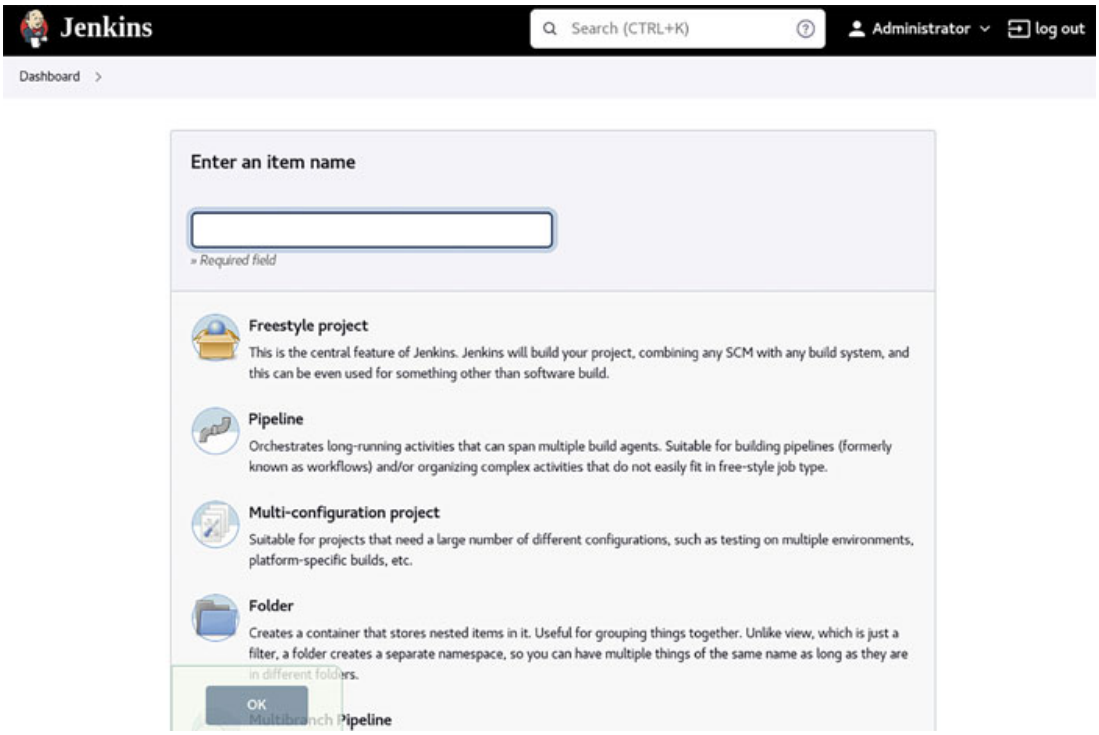


Figure 13.54: *Jenkins new project options*

Pipeline requires a `Jenkinsfile` (a declarative **Pipeline***)* to define the stages and the jobs to be executed. The syntax is defined on the website: **https://www.jenkins.io/doc/book/pipeline/**. The following code shows a simple example:

```
pipeline {
    agent any
    stages {
    stage('Build') {
       steps {
         echo 'Building..'
       }
    }
    stage('Test') {
       steps {
         echo 'Testing..'
       }
    }
```

```
    stage('Deploy') {
       steps {
        echo 'Deploying….'
       }
    }
    }
}
```

An example of building the image from a **`Dockerfile`**, testing it, and pushing it to an internal image registry is shown:

```
node { git branch: 'main',
   url: 'https://github.com/agonzalezrh/linuxservercb' }
pipeline {
   agent any;
   stages {
   stage('Build') {
      steps {
       script {
         image = docker.build("buildfromjenkins")
       }
      }
   }
   stage('Test') {
      steps {
       script {
          docker.image('buildfromjenkins:latest').withRun('-p 8888:80')
          {
            sh 'curl localhost:8888'
         }
       }
      }
   }
   stage('Deploy') {
      steps {
       script{
        docker.withRegistry('http://registry.example.com:5000', '') {
          image.push("${env.BUILD_NUMBER}")
          image.push("latest")
         }
       }
      }
   }
   }
```

}

Jenkins will show the status of the workflow, as is shown in *figure 13.55*:

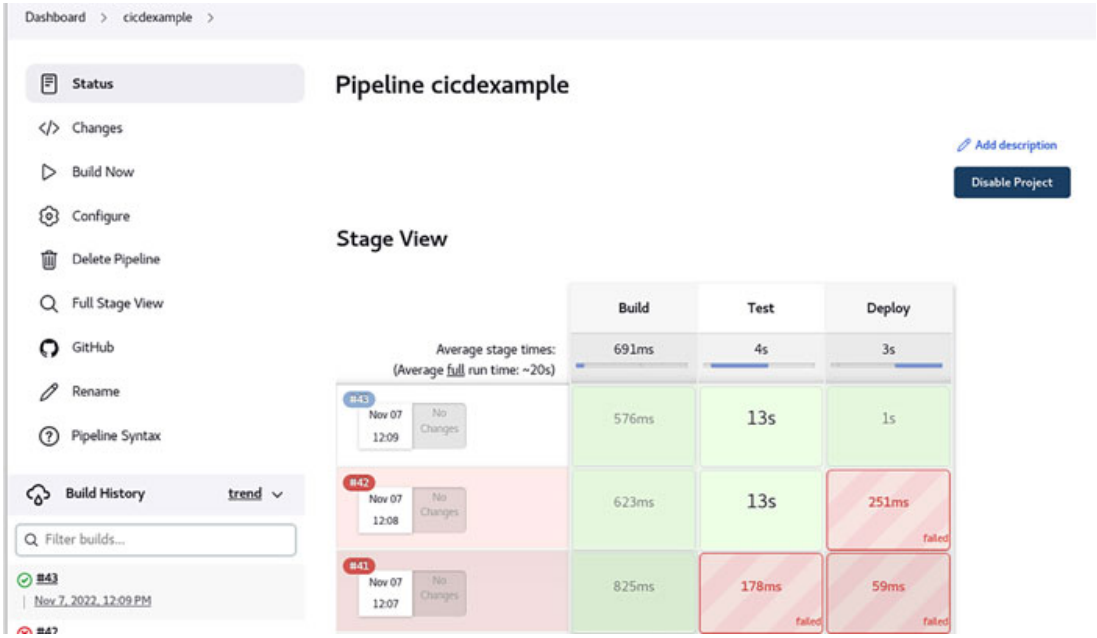


Figure 13.55: *Jenkins pipeline stage view example*

Gitlab CI/CD

The open-source self-host solution for the control version of **Gitlab** includes a built-in **CI/CD** tool, which allows to create *Pipelines* to structure the workflow. A pipeline has the following two main components:

- **Jobs**: defines what to do. For example, to compile a code.
- **Stages**: defines when to run the jobs. For example, run a test after the code is compiled.

Figure 13.56 shows an example workflow related to **Continuous Integration** and **Continuous Delivery**.

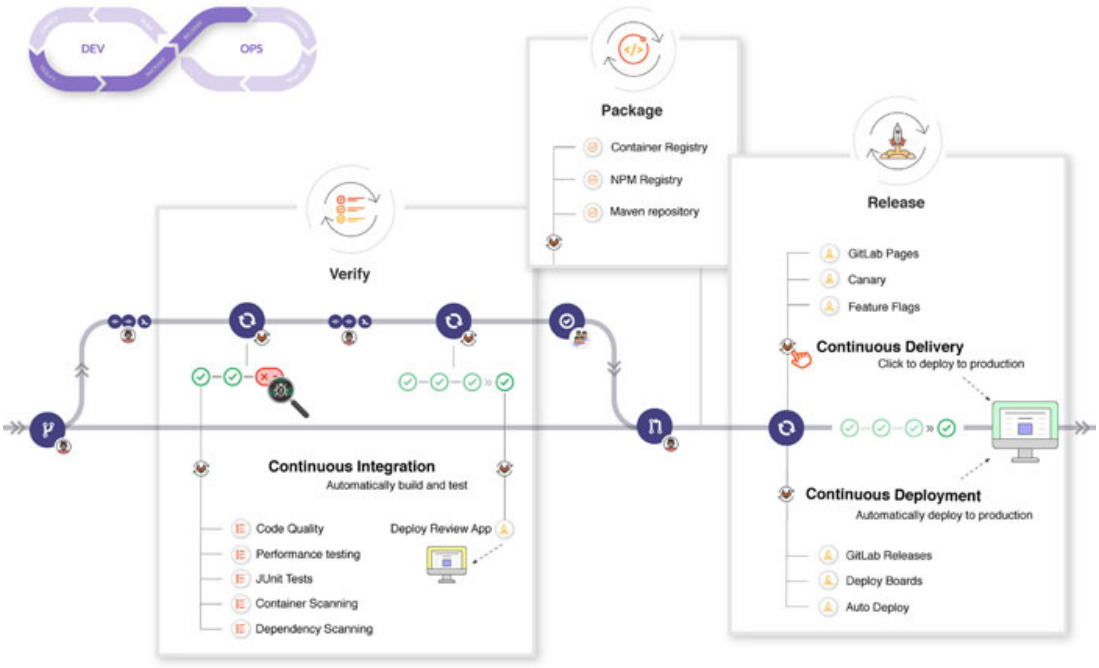


Figure 13.56: *CI/CD workflow example*

The **Pipelines** are defined in *YAML* format, and they are defined at the project level. In the left menu of the **GitLab** portal, it allows you to define the workflow. *Figure 13.57* shows the left menu options related to **CI/CD**:

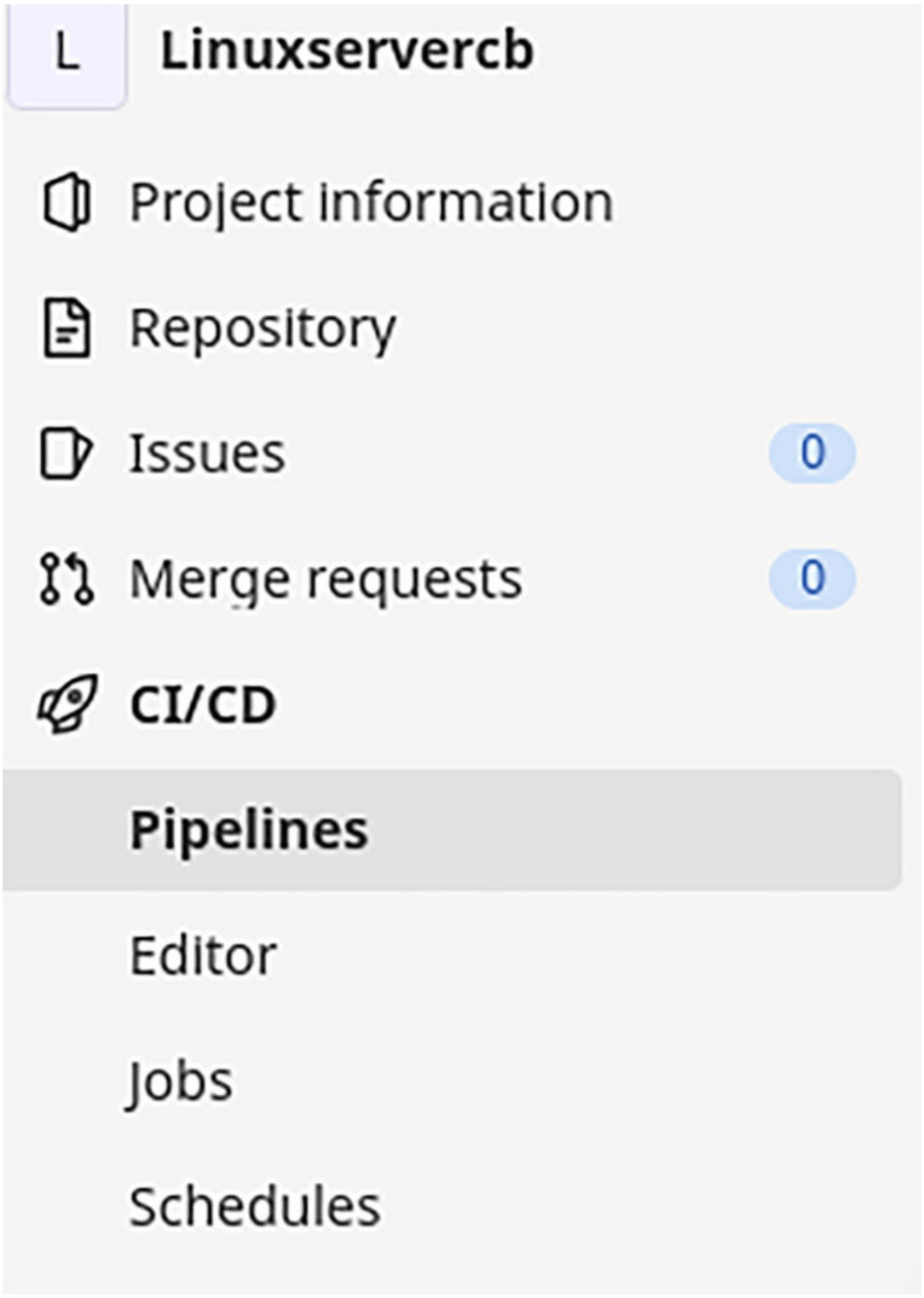


Figure 13.57: *GitLab left menu options for CI/CD*

The following code shows an example of building an image from a `Dockerfile` and pushing it to the internal registry, same as with **Jenkins**.

```
stages:          # List of stages for jobs, and their order of execution
  - build
  - test
  - deploy
build-job:       # This job runs in the build stage, which runs first.
  stage: build
  script:
    - buildah build -t ubuntuapache2:dev .
    - echo "Compile complete."
unit-test-job:   # This job runs in the test stage.
  stage: test    # It only starts when the build stage completes
  correctly.
  script:
    - podman run -dti --name test -p 8889:80 ubuntuapache2:dev
    - sleep 5
    - curl localhost:8889
    - podman stop test
    - podman rm test
deploy-job:      # This job runs in the deploy stage.
  stage: deploy  # It only runs when *both* jobs complete successfully.
  script:
  - podman tag ubuntuapache2:dev
  registry.example.com:5000/ubuntuapache2:fromgitlab
  - podman push --tls-verify=false
  registry.example.com:5000/ubuntuapache2:fromgitlab
```

When a new commit is performed in the repository, the **Pipeline** is executed, and the **GitLab** portal will show the execution status. *Figure 13.58* shows an example:

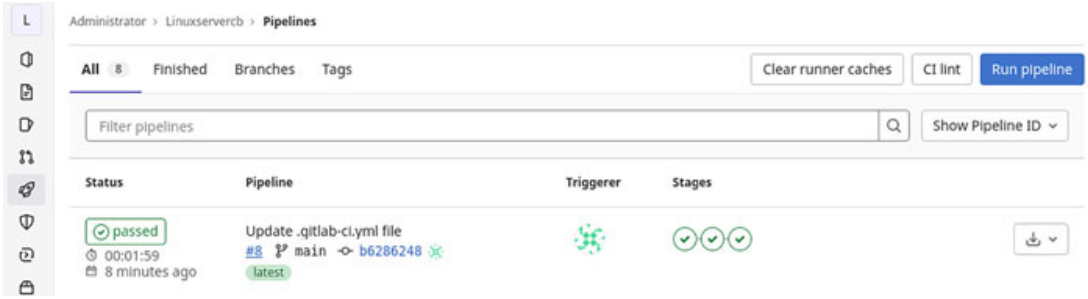


Figure 13.58: *GitLab CI/CD pipeline example*

GitHub actions

The most popular hosted source control version, **GitHub**, allows performing integration with different external **CI/CD** tools, such as **TravisCI**. In the last few years, **GitHub** has released and improved its own tool called **GitHub Actions.** It

allows running a workflow on any event, for example, if a new commit is done or a new PR is approved. *Figure 13.59* features a GitHub Actions workflow example.

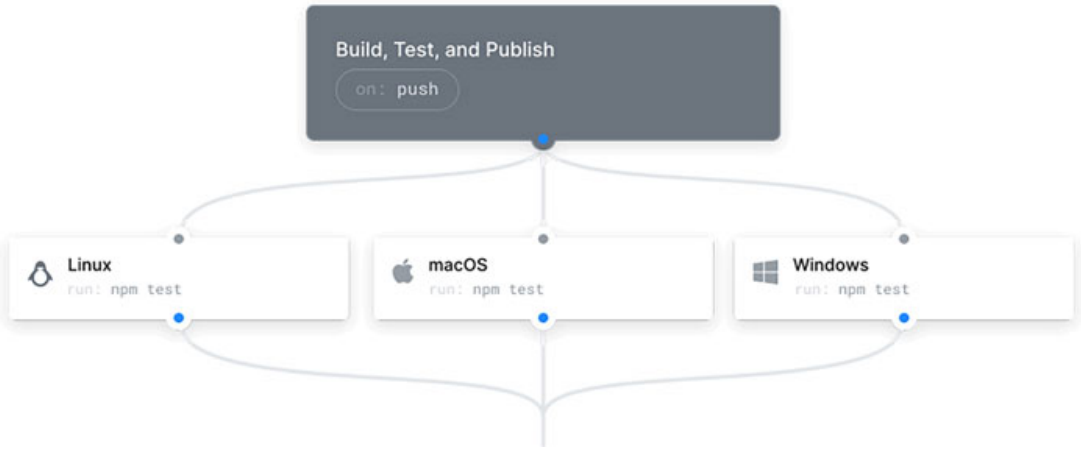


Figure 13.59: *GitHub actions workflow example*

In the top bar = of the repository to be configured, the icon and the word **`Actions`** appear. *Figure 13.60* shows the top bar access:

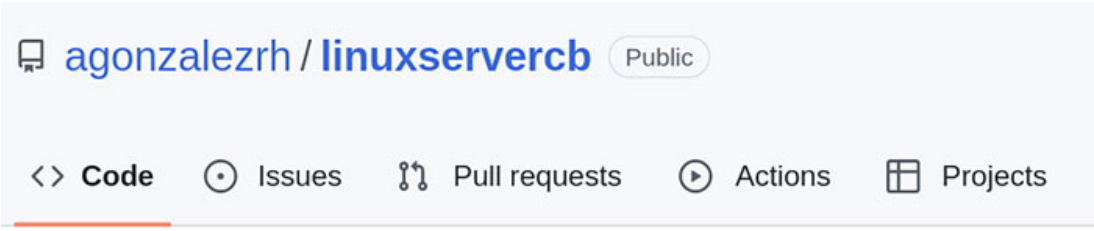


Figure 13.60: *GitHub top menu to access actions*

GitHub Actions would provide some recommendations depending on the content of the repository. As this repository contains a *Dockerfile*, the first suggestion is to build a **Docker image.** Refer to *figure 13.61*:

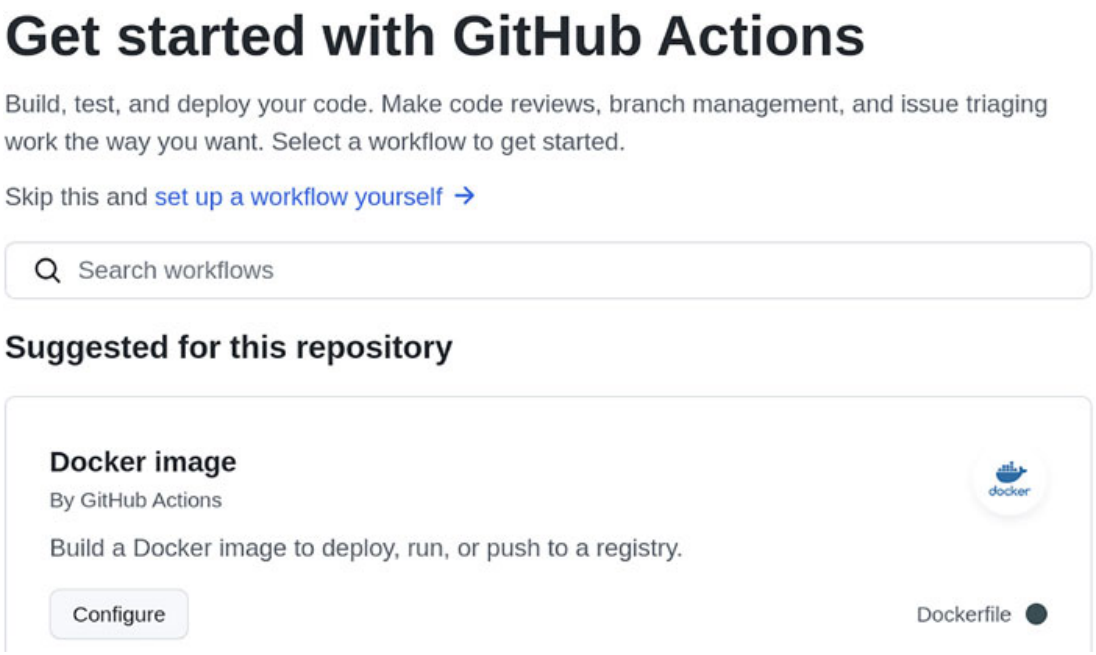


Figure 13.61: *GitHub actions workflow wizard*

The syntax to define the workflow is in *YAML* format. The following code shows an example to build the image and to push to **Docker Hub** repository:

```
name: Docker Image CI
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Build the Docker image
      run: docker build . --file Dockerfile --tag my-image-name:$(date
      +%s)
    - name: Login to Docker Hub
      uses: docker/login-action@v2
      with:
    username: ${{ secrets.DOCKERHUB_USERNAME }}
    password: ${{ secrets.DOCKERHUB_TOKEN }}
    - name: Build and push
      uses: docker/build-push-action@v3
      with:
```

```
    push: true
    tags: linuxservercb/ubuntuapache:fromgithub
```

Every time a new commit happens in the repository, the image will be automatically built and stored in the public repository. *Figure 13.62* shows the confirmation that the workflow was executed correctly:

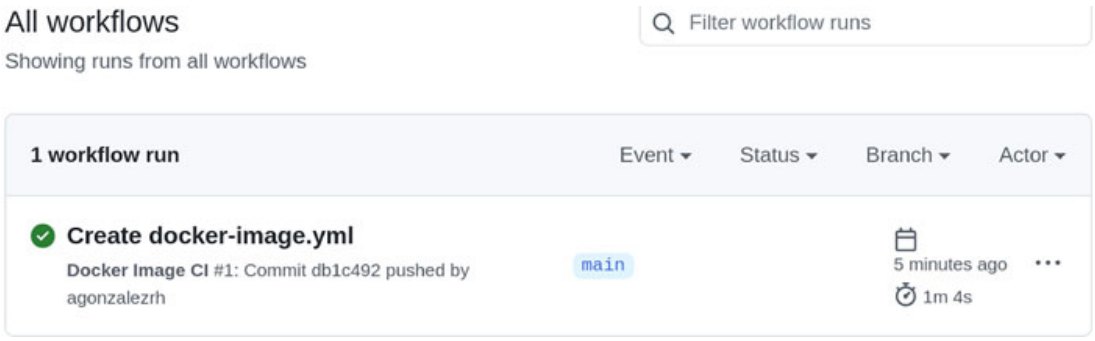


Figure 13.62: *GitHub actions execution example*

On the website **https://hub.docker.com**, a new version of the image has been pushed with the tag **`fromgithub`**. *Figure 13.63* shows the new tag:

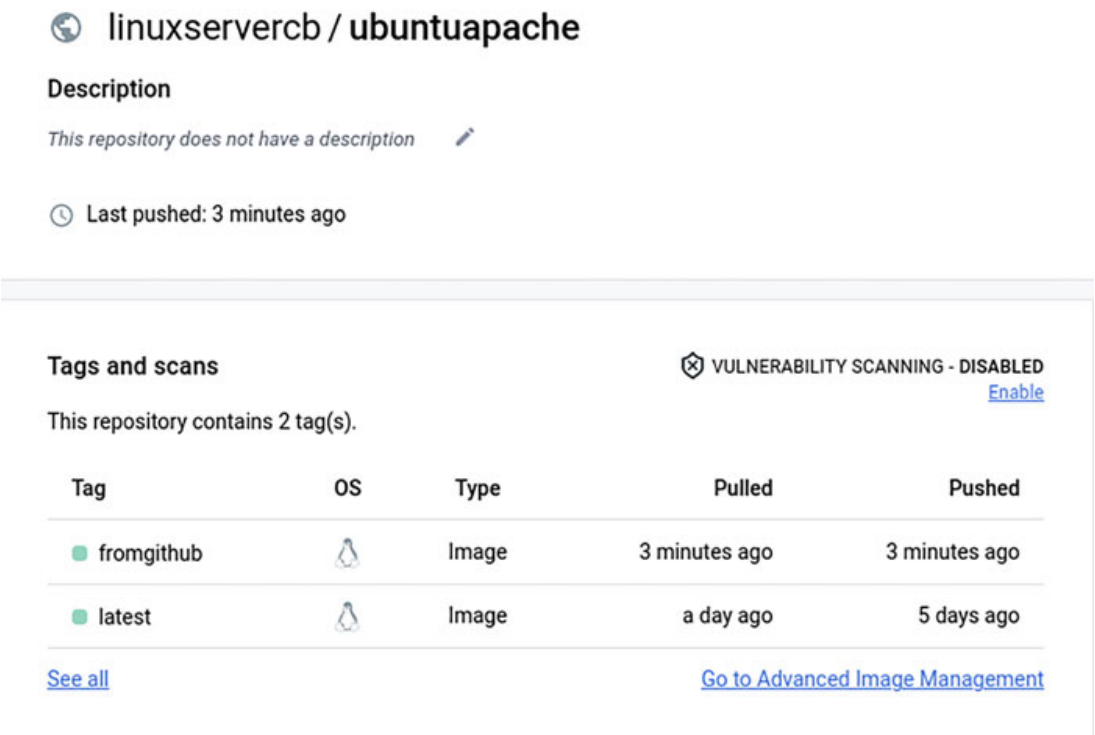


Figure 13.63: *Dokcer Hub new image and tag created automatically*

Conclusion

These days, **Containers** have become the main technology in the IT industry. It is important for System Administrators, Developers, and *DevOps* to have deep knowledge about the technology behind it and how to work with it. **Docker** and **Podman** help end user to run contains, create images, and perform **Continuous Integration**. Architecture that requires multiple contains, microservices, and advanced features would require to have knowledge about **Kubernetes**, and if a *Platform-as-a-Service* is required, they will be required to choose a solution based on **Kubernetes**, such as **Red Hat OpenShift Container Platform**.

Continuous Integration and **Continuous Delivery** are the main approaches to accelerate the software development process, reduce bugs, and help the collaboration between developers in the same team. Different open-source tools are available and combined with cloud-based solutions, and it is the decision of the team to choose the proper one for their own needs.

Key facts

- **Containers** are not a new technology, but **Docker** popularized them.
- **Docker** is not a technology itself, but it is a high-level suite to run **Containers**.
- **Docker** is a client-service architecture; **Podman** does not require a server to run the containers.
- It is possible to have an internal (private) **Image Registry** to store the images generated.
- Different tools are available to perform **Continuous Integration**, and **Jenkins** is one of the most used.

Questions

1. A **container** uses less resources than a **Virtual Machine**. True or false?
    a. True
    b. False
2. An **image** can only be used by one **container** at the same time. True or false?
    a. True
    b. False
3. What argument for the command `docker` is used to create a container without executing it?
    a. create
    b. run

    c. add

4. Which command part of the **Podman** suite is used to build images?

    a. buildman
    b. buildar
    c. buildah

5. What is the default name for a declarative **Pipeline** on **Jenkins**?

    a. Pipefile
    b. Jenkinsfile
    c. Workflowfile

Answers

1. a
2. b
3. a
4. c
5. b

CHAPTER 14

Backup and Restore

Introduction

Backup is the process of creating a copy of the data in a certain time with the objective of being able to recover it in the future. **Restore** is the process of recovering the data from a **Backup**, going back to an old version of the data, or reinstating a missing one. **Backup** requires a good definition, as well as planning and regular tasks to ensure the integrity of the copies.

This chapter covers the different strategies available for the **Backup** and how it affects **Restore**. The storage media where the **Backup** can be stored is discussed, as well as the different locations to protect the copies from possible disasters. **Backup** Solutions includes different features to optimize the process, reducing the size stored with compression and removing duplicates, among other features. Two popular software are detailed in this chapter are **Bacula** and **Relax and Recover (ReaR).**

Structure

In this chapter, we will discuss the following topics:

- Introduction to backup and restore
- Storage media for backups
- Backup types
- Backup sources
- Backup strategies
- Backup solution features
- Bacula
- Relax-and-recover (ReaR)

Introduction to backup and restore

The **Backup** process, and with it the **Restore** process, has evolved along with the big transformation in the **IT sector**. Big enterprises have dedicated teams to configure the target environment and the clients and are responsible for the planning as well. Good **Backup** planning is essential to avoid data loss, accelerate the recovery time, and perform tasks periodically to ensure the **Restore** is viable.

A **Backup** strategy requires to define of the following factors:

- **The data to be included**: This is an important decision to take for many reasons, such as the size that the **Backup** will need, as well as the **Restore** time needed. For example, a **Backup** can include all the files from a system or only the files needed to reinstate after the system is reinstalled with a fresh *Operating System* installation.
- **The frequency**: It is important to define how often the **Backup** is going to be performed. The planning should include per environment frequency to be daily, weekly, monthly, or another specified period.
- **Backup method to use**: There are several methods to make copies, which are going to be explored in this chapter. This includes a **Full, Differential**, or **Incremental Backup**, among others.
- **The window-time for backups:** A **Backup** can be a long-time process, especially for **Full** ones, and can consume resources and network bandwidth. Deciding the window time to perform is one factor to be included in the planning.
- **Backup storage media**: A Backup usually generates several copies to protect against disaster scenarios. Different media can be used to store, such as *Tape*, *Shared Storage, or Cloud Storage*.
- **Retention: Backup** planning should define the retention of the data, and after that period, the copies would be removed to save space in the storage media.

For the **Restore**, the important factors in defining the recovery are as follows:

- **How Restore will be performed:** A **Backup** can include all the data required to restore a system or only those required for a service offered. It is important to define if the recovery process will be performed in the same system, if possible, or in a new one.

- **How fast a Restore needs to be:** Recovering data from a **Backup** can take longer, depending on the strategies used. The planning should thus include what is an acceptable time to reinstate files and services from a copy.
- **Restore procedures:** Each system, service, or element included in the **Backup** should have a **Restore** procedure associated in case of data loss.
- **Restore test plan**: A **Backup** without regular recovery testing cannot ensure that the **Restore** procedures are the proper ones.

The overall **Recovery** goals are defined with two terms as follows:

- **Recovery Time Objective (RTO)**: the maximum duration of time to restore a disruption of the service in order to avoid unacceptable consequences. For example, for a non-critical environment, it can be hours or days, and for critical environments, it can be specified in minutes.
- **Recovery point objective (RPO)**: the maximum targeted period during which the data is lost due to an incident.

Figure 14.1 shows a diagram representing the terms **RTO/RPO**:

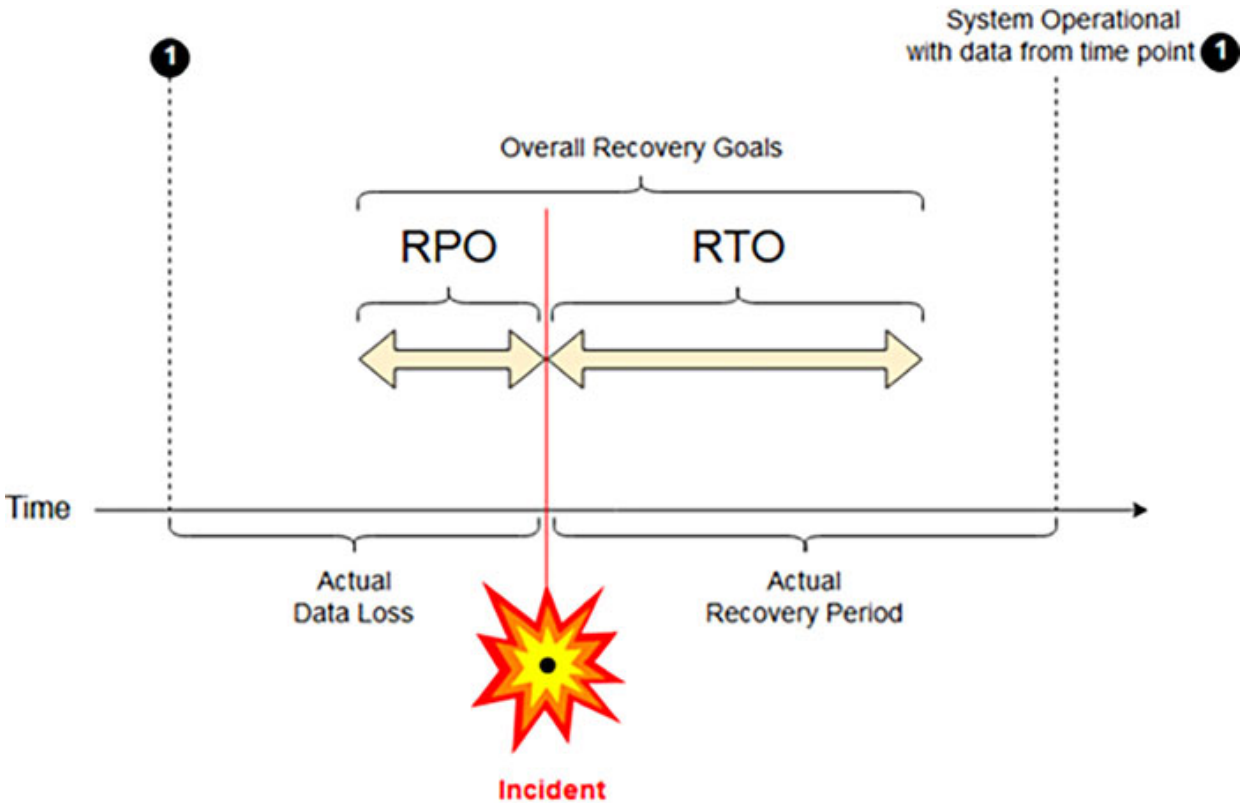


Figure 14.1: *Terms RPO/RTO represented. Source: Wikipedia.*

It is important to remark that a Distributed Storage Solution (such as Ceph*),* High Availability systems, or Mirroring Technologies (such as *RAID*) are **not Backup** solutions, and they do not replace a proper strategy to protect the data. These days, with the adoption of **Containers** and the **Cloud**, the strategy to ensure the data is protected has evolved. Different concepts covered throughout this book, such as Automation, Micro Services, Infrastructure as Code, and DevOps, have helped the requirement to be included in a **Backup**, which is the only data used by the service consumer. Recreating environments is a fast and easy task on modern infrastructures, but the importance of **Backup** and **Restore** for the data is still really important.

The **Restore** task might be needed in the following cases:

- **Quick restore from a local copy**: For example, if the automation is deploying a new configuration for a service and it is not valid, it is possible to restore the previous version.
- **Restore files from a Backup solution**: Data loss can be caused by human mistakes, application misbehavior, external attacks, or hardware

dame (such as disk problems).

- **Restore full system**: That can include restoring a Virtual Machine or a Physical system. Modern **Backup** solutions allow performing a copy of a running system and the possibility of restoring it in the same system or in a new one.

Storage media for Backups

First, enterprise **Backups** were stored on physical tapes, where individual files were included in the copy. The **Restore** process is required to find the proper tape where the data to be recovered is located and then copy it from there to the system. More modern solutions were based on *Tape Libraries* (also known as *Tape Robot),* which included several tapes, and they automatically mount the desired tape to perform a **Backup** or **Restore**. **Virtual Tape Library** (**VTL**) simulates a *Tape Library* having the back-end of a regular storage system, such as a hard drive. One of the biggest advantages of the usage of tapes, even these days, is the possibility of storing them easily off-site (in a different location where the original data is).

The popularity of tapes as storage media for **Backups** was present until the first decade of the 21st century. The reduction of the cost of hard drives and the introduction of **Cloud Storage**, along with different work methodologies, such as **DevOps**, reduced the use of tapes in enterprises. **Network Storage** solutions such as *NFS, iSCSI*, and *NAS* are used to store data from **Backups**. Nonetheless, it is important to keep a copy in a different location (such as a different *Data Center*) from the original data.

Modern environments use **Object Storage** to store data, including **Backup** data. Public clouds offer affordable storage to store *Objects.* An **Object Storage System** places the unstructured data as opposite to regular file systems. The *Object* (file) to be stored cannot be modified but can only be replaced with a version. Some available options are as follows:

- **Amazon Web Services (AWS)** offers a cheap solution called **Amazon S3 Glacier** for archiving and backup.
- **Microsoft Azure** offers the service **Azure Archive Storage.**
- **Google Cloud** offers **Cloud Storage** solution for archives and backups.
- Private cloud **OpenStack** offers the solution, **Swift.**

- Software-defined storage solution **Ceph** has a component called **Rados Gateway** (**RGW**) to offer an **Object Storage** service.

Backup types

There are the following three main types of **Backup**:

- **Full Backup:** It contains all the data to be copied. This type does not mean that the whole system will be copied; rather, it means that all the data specified to be copied will be included in the **Backup** performed. The main advantage and disadvantage are as follows:
  - **Advantage**: Recover time is fast because all the data is included.
  - **Disadvantage**: The space needed is big, and each **Backup** performed requires all the space. For example, if the data to be saved is 1 TB, and it is needed to have 30 different **Backups**, the total size will be 30 TB.
- **Incremental Backup:** Only transfer the data changed from the previous **Backup** performed and the current time. This procedure reduces the space needed. The main advantage and disadvantage are as follows:
  - **Advantage**: The required space and the time to perform the **Incremental Backup** is reduced considerably.
  - **Disadvantage**: The **Restore** requires more time because it is required to go through several **Backup** performed.
- **Differential Backup:** This method transfers only the difference between the previous **Full Backup** and the moment where the **Backup** is being performed. The main advantage and disadvantage are as follows:
  - **Advantage**: The **Restore** process requires less references than the **Incremental** one; it only requires the **Full Backup** and the specific **Differential Backup**.
  - **Disadvantage**: The size of the copy is bigger than in an **Incremental** one.

Table 14.1 shows an example of the schedule for a **Backup** where the **Full Backup** is performed on Sunday when it would generally not impact the service or end users. For the rest of the days, the **Incremental Backup** is used.

Day	Backup type	Backup size	Total Backup size
Sunday	Full Backup	1 TB	1 TB
Monday	Incremental Backup	20 GB	1.02 TB
Tuesday	Incremental Backup	100 GB	1.12 TB
Wednesday	Incremental Backup	200 GB	1.32 TB
Thursday	Incremental Backup	500 GB	1.82 TB
Friday	Incremental Backup	80 GB	1.90 TB
Saturday	Incremental Backup	100 GB	2 TB
Friday	Full Backup	2 TB	4 TB

Table 14.1: *Full and incremental Backup example*

To restore data (for example, all the content of a directory) from a backup performed on Wednesday, the backup is needed from Sunday (**The Full Backup)**, Monday (**Incremental**), Tuesday **(Incremental)**, and Wednesday **(Incremental).**

Table 14.2 shows the same example using **Differential Backup** instead.

Day	Backup type	Backup size	Total Backup size
Sunday	Full Backup	1 TB	1 TB
Monday	Differential Backup	20 GB	1.02 TB
Tuesday	Differential Backup	120 GB (100+20)	1.14 TB
Wednesday	Differential Backup	320 GB (200+120)	1.46 TB
Thursday	Differential Backup	820 GB (500+320)	2.28 TB
Friday	Differential Backup	900 GB (820 + 80)	3.18 TB
Saturday	Differential Backup	1 TB (900 + 100)	4.18 TB
Friday	Full Backup	2 TB	6.18 TB

Table 14.2: *Full and differential Backup example*

It is possible to observe how the total **Backup** size required using **Different Backup** is bigger than with the **Incremental Backup**. The advantage of this method is that restoring a file from Wednesday's **Backup** requires only a copy of the Sunday's (**Full Backup**) and Wednesday's one (**Different Backup).**

Backup sources

As described previously, **Backup** is the process of copy data being protected. Depending on the data to be saved, there are different strategies and storage media. Some examples of elements that can be backed up are as follows:

- **Static files**: These are the common elements to be included. These files are usually not modified often, and only new files are copied. For example, the copy can include files used by applications or end users.
- **Dynamic files**: These files are being modified often, such as log files or automatically generated files (such as an exported *database*).
- **Block devices**: The **Backup** can include the whole data in a device, such as a full disk or partition.
- **Virtual Machine**: It is possible to make a **Backup** of the full **Virtual Machine** in two different ways:
    - Using the *hypervisor*, which can create a *snapshot* (a copy of the data in a specific time) and export it to a regular file to be included in the copy.
    - Installing the **Backup client** to perform a full copy of the data inside.
- **Containers**: They have not copied themselves, only the data associated with them (such as *Volumes).*
- **Databases:** A backup of a database can be done in two main ways:
    - Exporting the content and the database structure to regular files to be included in the **Backup.**
    - Copying the associated files directly, thus, ensuring that the data would not be corrupted on the copy. This can approach have the following two options:

    - **Hot Backup**: The **Database** is still running while the backup is being performed.
    - **Cold Backup**: The **Database** is stopped earlier to perform the file-based copy.

Backup strategies

Deciding a **Backup strategy** is essential to protect the data saved in the case of data loss as well as any possible disaster event. Different strategies are available to mitigate disaster situations, and they are as follows:

- **3-2-1 Rule:** This strategy consists of having three copies, of which two of them are located on the same site but using different media, where the **Backup** is performed to speed up possible recovery of the data. The third copy is **off-site** (different location) to have a safe copy in case of a disaster situation.
- **Remote Backup**: Performing the **Backup** to a remote service which usually is not managed by the owners of the original place. For example, using a service **Backup as a Service** or an *Object Storage solution.*
- **Site-to-Site Backup**: This strategy always copies the data to a location off-site under the current location. Both sites are managed by the same owners.
- **Disaster Recovery** is the practice of having a second location, different from the main one, where the services are usually ready to be promoted as a primary. This method requires to have configured the replication between the sites. The biggest advantages of the implementation of **Disaster Recovery Site(s)** are as follows:
    - Reduce downtime, and the secondary site is able to resume the service quickly.
    - Data loss is reduced to the last replication, usually in seconds or minutes.
    - Possibility to go back to the main site when the incident is resolved.

Backup solution features

Backup solutions include different features to improve performance, reduce the size needed and increase the security of the data stored. Some of the popular features are as follows:

- **File locking:** The solution should be able to make a copy even when the file is in use, such as log files or databases data.
- **Data expiration**: Out-of-date data can be automatically deleted after the retention period is over. This feature reduces the space needed in the storage media.
- **Compression:** Compression can be done by the **Backup Solution** or directly applied to the storage media, such as built-in compression in a *Tape.*
- **Deduplication:** One of the most advanced features of **Backup Solution** is avoiding the duplication of data in the storage media. If several systems have the same data, such as using the same operating system, only one copy of the data would be stored in the storage media.
- **Duplication:** The copies can be distributed in different locations; the **Backup Solution** is responsible, after receiving the data from the client, for distributing it properly to different locations, ensuring they are not in the same location.
- **Encryption:** Another advanced feature is to encrypt the data saved in a **Backup**. *Encryption* requires more resources, and it slows the copy and the restore. However, for some environments, it is a requirement.
- **Multiplexing**: A **Backup Solution** is responsible for receiving data from multiple sources and storing them properly.
- **Staging**: For some storage media, intermediate storage is first used to store the media. Usually, making a copy to a *Tape* requires having a disk, which is used temporarily, before sending all the data to the *Tape*. When all the data is written in the *Tape*, the temporary data is removed from the disk.

Bacula

This is one of the best open-source **Backup** products available. This solution includes community and commercial support, being a perfect option for

small and enterprise companies. Some of the benefits of using **Bacula** are as follows:

- GUI and CLI interface.
- A wide variety of possible backup levels and techniques, including **full**, **differential**, and **incremental**.
- Ability to both backup and restore entire systems or single singles easily.
- **Cloud backup** support for AWS S3.
- The capability of using snapshots on Linux/Unix systems. Snapshots can be automatically created and used to backup files.
- Use either the command line and/or GUI to execute and control your backup process.
- Use partition backup for your Linux data storage to keep precise images of these disks for easy management and recovery.
- Tape backup is fully supported and allows it to work with a variety of different tape drives, autoloaders, and autochangers. *VTL* support is also included.
- Deduplication developed by themselves.

Bacula installation

The **Bacula** installation requires to use a *database* to store configuration and all the needed information about the clients and the copies. The available options are **MariaDB** (**MySQL**) and **PostgreSQL**. When the *Database* is available, the installer configures whatever is needed to use it. The instructions included are for a system using *Ubuntu 22.04* (at the time of writing this book, **Bacula** was still not available on the official repository in Ubuntu) are as follows:

```
# Create repository file
echo "deb [arch=amd64]
https://bacula.org/packages/5bd703346c037/debs/13.0.1 jammy
main" | tee /etc/apt/sources.list.d/Bacula-Community.list
# Add GPG key as tusted
wget -qO - https://www.bacula.org/downloads/Bacula-4096-
Distribution-Verification-key.asc | gpg --dearmor >
```

```
/etc/apt/trusted.gpg.d/bacula.gpg
# Install mariadb-client and bacula-mysql packages
apt install mariadb-client bacula-mysql
```

Bacula services

Bacula is made up of the following five major components or services:

- **Bacula Director**: The Bacula Director service is the program that supervises all the backup, restore, verify, and archive operations.
- **Bacula Console**: The Bacula Console service is the program that allows the administrator or user to communicate with the **Bacula Director**.
- **Bacula File:** This service (also known as the Client program) is the software program that is installed on the machine to be backed up.
- **Bacula Storage:** These services consist of software programs that perform the storage and recovery of the file attributes and data to the physical backup media or volumes.
- **Catalog**: These services are comprised of the software programs responsible for maintaining the file indexes and volume databases for all files backed up.

Figure 14.2 features the Bacula components:

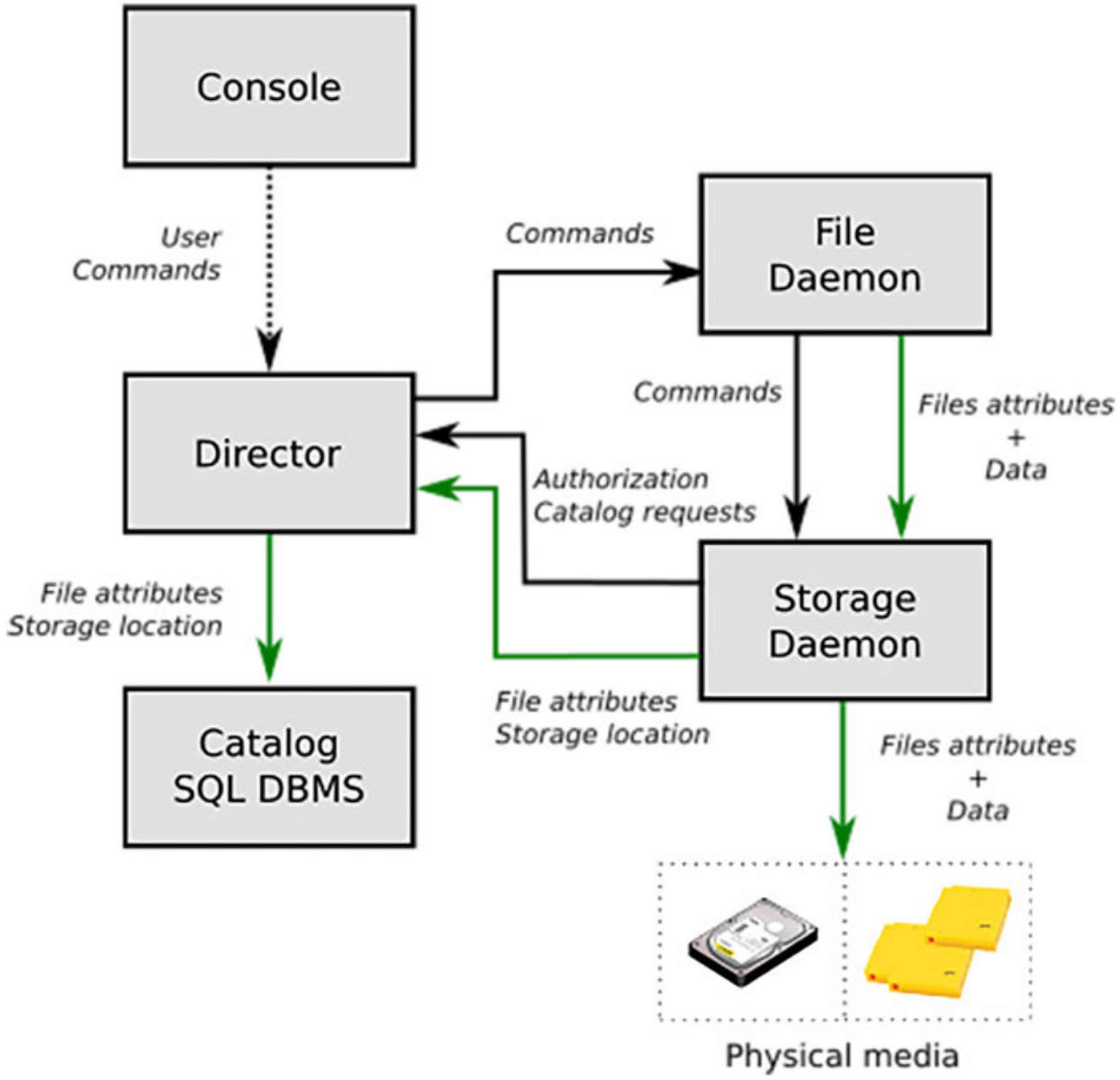


Figure 14.2: *Bacula components. Source: Bacula*

The following example configuration was added to the default configuration (which, in regular situations, should be reconfigured completely for the required needs).

Create the **`/backup`** directory, assign permission to the user and group **`bacula`**, and configure the file **`bacula-sd.conf`** to create a new **Device**.

```
mkdir /backup
chown bacula:bacula /backup
cat >>/opt/bacula/etc/bacula-sd.conf <<EOF
Device {
```

```
 Name = Backupdir
 Media Type = File
 Archive Device = /backup
 LabelMedia = yes;
 Random Access = Yes;
 AutomaticMount = yes;
 RemovableMedia = no;
 AlwaysOpen = no;
 Maximum Concurrent Jobs = 5
}
EOF
```

Configure the file **bacula-dir.conf** to define:

- A new **Storage** called **bacula-sd**.
- A new **FileSet** named **OnlyEtc** to only copy the **/etc** directory.
- A new **Schedule** called **EtcDaily** to be executed every day at 10 p.m.
- A new **Job** named **EtcBackup**.
- A new **Client** called **rhel-fd**.

```
SDPW=$(grep -m1 Password /opt/bacula/etc/bacula-sd.conf |awk
'{print $3}')
cat >>/opt/bacula/etc/bacula-dir.conf <<EOF
Storage {
 Name = bacula-sd
 Address = 192.168.122.43
 Device = Backupdir
 Media Type = File
 Password = $SDPW
}
FileSet {
 Name = "OnlyEtc"
   Include {
    Options {
   signature = MD5
    }
    File = /etc
   }
```

```
}
Schedule {
  Name = "EtcDaily"
  Run = Full daily at 22:00
}
Job {
  Name = "EtcBackup"
  JobDefs = "DefaultJob"
  Enabled = yes
  Level = Full
  FileSet = "OnlyEtc"
  Schedule = "EtcDaily"
  Client = rhel-fd
  Storage = bacula-sd
}
Client {
  Name = rhel-fd
  Address = 192.168.122.226
  FDPort = 9102
  Catalog = MyCatalog
  Password = "MyStrongPassword"
  File Retention = 30 days
  Job Retention = 3 months
  AutoPrune = yes
}
EOF
```

After the configuration is performed, the services should be restarted.

```
systemctl restart bacula-fd.service
systemctl restart bacula-sd.service
systemctl restart bacula-dir.service
```

Client installation

The client is required to install the package **`bacula-client`**. In the following example, a **Red Hat Enterprise Linux** system is used, where the package is included in the official repositories. The instructions are as follows:

```
firewall-cmd --add-service=bacula-client --permanent
```

```
firewall-cmd --reload
yum install -y bacula-client
cat >/etc/bacula/bacula-fd.conf <<EOF
Director {
 Name = bacula-dir
 Password = "MyStrongPassword"
}
FileDaemon {
 Name = rhel-fd
 FDport = 9102
 WorkingDirectory = /var/spool/bacula
 Pid Directory = /var/run
 Maximum Concurrent Jobs = 20
 Plugin Directory = /usr/lib64/bacula
}
Messages {
 Name = Standard
 director = bacula-dir = all, !skipped, !restored, !saved
}
EOF
systemctl enable --now bacula-fd
```

Command bconsole

The command **`bconsole`** on the **Bacula Server** is the main command to obtain information about the **Backup**, run jobs, and perform other tasks. On executing the command, an interactive mode appears, and it is possible to run the commands desired. *Figure 14.3* shows the output of the command **`status director`** to obtain information about the **Bacula Director**.

```
root@bacula:~# bconsole
Connecting to Director bacula:9101
1000 OK: 10002 bacula-dir Version: 13.0.1 (05 August 2022)
Enter a period to cancel a command.
*status director
bacula-dir Version: 13.0.1 (05 August 2022) x86_64-pc-linux-gnu-bacula-enterprise
 ubuntu 22.04
Daemon started 12-Nov-22 19:24, conf reloaded 12-Nov-2022 19:24:12
 Jobs: run=0, running=0 mode=0,0
 Crypto: fips=N/A crypto=OpenSSL 3.0.2 15 Mar 2022
 Heap: heap=1,097,728 smbytes=349,477 max_bytes=352,116 bufs=423 max_bufs=435
 Res: njobs=4 nclients=2 nstores=3 npools=3 ncats=1 nfsets=3 nscheds=3

Scheduled Jobs:
Level          Type     Pri  Scheduled          Job Name           Volume
===================================================================================
==
Full           Backup    10  12-Nov-22 22:00    EtcBackup          *unknown*
Incremental    Backup    10  12-Nov-22 23:05    BackupClient1      *unknown*
Full           Backup    11  12-Nov-22 23:10    BackupCatalog      *unknown*
====

Running Jobs:
Console connected using TLS at 12-Nov-22 19:29
No Jobs running.
====
No Terminated Jobs.
====
```

Figure 14.3: *Example showing the status of* ***Bacula Director*** *using* ***bconsole***

To obtain information about one *client* and check if the communication is correct, the command to introduce is the **`status client`**. A list of the clients configured appears, and after selecting the proper one, it will provide the information. *Figure 14.4* shows an example of the node **`rhel-fd`**.

```
*status client
The defined Client resources are:
     1: bacula-fd
     2: rhel-fd
Select Client (File daemon) resource (1-2): 2
Connecting to Client rhel-fd at 192.168.122.226:9102

rhel-fd Version: 11.0.1 (05 February 2020)  x86_64-redhat-linux-gnu redhat Enterp
rise 9.0
Daemon started 12-Nov-22 19:26. Jobs: run=0 running=0.
 Heap: heap=733,184 smbytes=199,392 max_bytes=199,392 bufs=104 max_bufs=104
 Sizes: boffset_t=8 size_t=8 debug=0 trace=0 mode=0,0 bwlimit=0kB/s
 Crypto: fips=N/A crypto=OpenSSL
 Plugin: bpipe-fd.so cdp-fd.so

Running Jobs:
Director connected at: 12-Nov-22 19:31
No Jobs running.
====

Terminated Jobs:
====
```

Figure 14.4: *Example showing the status of* ***Bacula Client*** *using* ***bconsole***

To execute a **Job** defined, the command is **run**, and the client will ask which one should be executed. *Figure 14.5* shows an example:

```
*run
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"
A job name must be specified.
The defined Job resources are:
     1: BackupClient1
     2: BackupCatalog
     3: RestoreFiles
     4: EtcBackup
Select Job resource (1-4): 4
Run Backup job
JobName:  EtcBackup
Level:    Full
Client:   rhel-fd
FileSet:  OnlyEtc
Pool:     File (From Job resource)
Storage:  bacula-sd (From Job resource)
When:     2022-11-12 19:34:22
Priority: 10
OK to run? (yes/mod/no): yes
Job queued. JobId=1
```

Figure 14.5: *Example running a* ***Bacula Job*** *using* ***bconsole***

When the job finishes, indicating the copy was correct or reporting any error, it is possible to use the command **`messages`** to obtain the latest information. *Figure 14.6* shows an example output:

```
12-Nov 20:14 bacula-dir JobId 13: Bacula Enterprise bacula-dir 13.0.1 (05Aug22):
  Build OS:               x86_64-pc-linux-gnu-bacula-enterprise ubuntu 22.04
  JobId:                  13
  Job:                    EtcBackup.2022-11-12_20.14.52_03
  Backup Level:           Full
  Client:                 "rhel-fd" 11.0.1 (05Feb20) x86_64-redhat-linux-gnu,redhat,Enterprise 9.0
  FileSet:                "OnlyEtc" 2022-11-12 19:34:24
  Pool:                   "File" (From Job resource)
  Catalog:                "MyCatalog" (From Client resource)
  Storage:                "bacula-sd" (From Job resource)
  Scheduled time:         12-Nov-2022 20:14:50
  Start time:             12-Nov-2022 20:14:54
  End time:               12-Nov-2022 20:14:55
  Elapsed time:           1 sec
  Priority:               10
  FD Files Written:       1,003
  SD Files Written:       1,003
  FD Bytes Written:       21,126,497 (21.12 MB)
  SD Bytes Written:       21,238,253 (21.23 MB)
  Rate:                   21126.5 KB/s
  Software Compression:   None
  Comm Line Compression:  62.7% 2.7:1
  Snapshot/VSS:           no
  Encryption:             no
  Accurate:               no
  Volume name(s):         Vol-0001
  Volume Session Id:      1
  Volume Session Time:    1668284067
  Last Volume Bytes:      42,558,716 (42.55 MB)
  Non-fatal FD errors:    0
  SD Errors:              0
  FD termination status:  OK
  SD termination status:  OK
  Termination:            Backup OK
```

Figure 14.6: *Example showing messages using* ***bconsole***

To obtain the list of the jobs and the status, the command to be executed is **`list jobs`**. *Figure 14.7* shows the output example:

```
*list jobs
+-------+-----------+---------------------+------+-------+----------+------------+-----------+
| JobId | Name      | StartTime           | Type | Level | JobFiles | JobBytes   | JobStatus |
+-------+-----------+---------------------+------+-------+----------+------------+-----------+
|    13 | EtcBackup | 2022-11-12 20:14:54 | B    | F     |    1,003 | 21,126,497 | T         |
+-------+-----------+---------------------+------+-------+----------+------------+-----------+
```

Figure 14.7: *Example listing* ***Bacula Jobs*** *using* ***bconsole***

It is possible to use **`bconsole`** without interactive mode, sending the action to perform to the *standard input*. In *figure 14.8*, an example is shown using *pipes* to obtain information about the client **`rhel-fd`**.

```
root@bacula:~# echo "status client=rhel-fd" | bconsole |tail -6
Terminated Jobs:
 JobId  Level    Files      Bytes   Status   Finished        Name
====================================================================
    12  Full       1,003    21.12 M  OK       12-Nov-22 20:13 EtcBackup
    13  Full       1,003    21.12 M  OK       12-Nov-22 20:14 EtcBackup
====
```

Figure 14.8: *Example pipes with the command* ***bconsole***

To **Restore** a file from the backup, the command in the interactive mode of **console** is **`restore`**. This command will give different options regarding which **Backup** to use for the restore. In *figure 14.9*, option 5 (*Select the most recent backup for a client*) is selected for the client **`rhel-fd`**.

```
*restore
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"

First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.

To select the JobIds, you have the following choices:
     1: List last 20 Jobs run
     2: List Jobs where a given File is saved
     3: Enter list of comma separated JobIds to select
     4: Enter SQL list command
     5: Select the most recent backup for a client
     6: Select backup for a client before a specified time
     7: Enter a list of files to restore
     8: Enter a list of files to restore before a specified time
     9: Find the JobIds of the most recent backup for a client
    10: Find the JobIds for a backup for a client before a specified time
    11: Enter a list of directories to restore for found JobIds
    12: Select full restore to a specified Job date
    13: Select object to restore
    14: Cancel
Select item:  (1-14): 5
Defined Clients:
     1: bacula-fd
     2: rhel-fd
Select the Client (1-2): 2
Automatically selected FileSet: OnlyEtc
+-------+-------+----------+------------+---------------------+------------+
| JobId | Level | JobFiles | JobBytes   | StartTime           | VolumeName |
+-------+-------+----------+------------+---------------------+------------+
|    13 | F     |    1,003 | 21,126,497 | 2022-11-12 20:14:54 | Vol-0001   |
+-------+-------+----------+------------+---------------------+------------+
You have selected the following JobId: 13
```

Figure 14.9: *Example restoring files with the command* ***bconsole***

After the latest **Backup** is selected automatically for the client specified, a command line to specify the files to restore (using the keyword **add**) is available. When the defined files to recover are finished, the command **`done`** will perform the restoration. *Figure 14.10* shows an example:

```
Building directory tree for JobId(s) 13 ...  ++++++++++++++++++++++++++++++++++++++++
814 files inserted into the tree.

You are now entering file selection mode where you add (mark) and
remove (unmark) files to be restored. No files are initially added, unless
you used the "all" keyword on the command line.
Enter "done" to leave this mode.

cwd is: /
$ cd etc/
cwd is: /etc/
$ add resolv.conf
1 file marked.
$ add services
1 file marked.
$ lsmark
*resolv.conf
*services
$ done
Bootstrap records written to /opt/bacula/working/bacula-dir.restore.3.bsr

The Job will require the following (*=>InChanger):
   Volume(s)                 Storage(s)                SD Device(s)
===========================================================================

    Vol-0001                  bacula-sd                 Backupdir

Volumes marked with "*" are in the Autochanger.


2 files selected to be restored.
```

Figure 14.10: *Example restoring files with the command* ***bconsole***

A **Restore Job** will be created, and the files will be restored in the client. In *figure 14.11*, the files are going to be restored inside the directory **`/opt/bacula/archive/bacula-restores`**.

```
Using Catalog "MyCatalog"
Run Restore job
JobName:         RestoreFiles
Bootstrap:       /opt/bacula/working/bacula-dir.restore.3.bsr
Where:           /opt/bacula/archive/bacula-restores
Replace:         Always
FileSet:         Full Set
Backup Client:   rhel-fd
Restore Client:  rhel-fd
Storage:         bacula-sd
When:            2022-11-12 20:27:37
Catalog:         MyCatalog
Priority:        10
Plugin Options:  *None*
OK to run? (yes/mod/no): yes
Job queued. JobId=14
```

Figure 14.11: *Example restoring files with the command* ***bconsole***

When the **Jobs** finishes, and if there is no error, the files are available. *Figure 14.12* shows the content of the directory indicated before:

```
[root@rhel ~]# tree /opt/bacula/archive/bacula-restores/
/opt/bacula/archive/bacula-restores/
└── etc
    ├── resolv.conf
    └── services

1 directory, 2 files
```

Figure 14.12: *Example listing restored files in the client*

Relax-and-Recover (ReaR)

This software is an open-source *bare metal* disaster recovery solution. It is easy to install, configure and use for restoring systems. This solution generates a copy of the system ready to be used. It supports various boot media (*ISO, PXE, USB, or eSATA*) and a variety of network protocols (*nfs, cifs, ftp, or http)* as well as different Backup strategies (*Symantec Netbackup, EMC Networker,* among others).

The files generated can be included in another **Backup** solution, such as **Bacula**, to be used later if needed. The **Recovering** is a two-step process, mounting the boot media and transferring the data from the central server or the storage device.

One of the biggest advantages of using **ReaR** is that this solution uses system tools to perform the **Backup.** The following examples use a central *NFS* storage to save the data and the *ISO* image from being used. The installation requires to only install the package **rear** from the software repository. After the installation, the file to be configured is **`/etc/rear/local.conf`** as follows (example in a **Red Hat Enterprise Linux** system):

```
yum install -y rear
cat >/etc/rear/local.conf <<EOF
OUTPUT=ISO
BACKUP=NETFS
BACKUP_URL="nfs://ubuntu/storage"
EOF
```

The **ReaR** solutions bring the command **rear** to perform different operations. The argument **`mkbackup`** will initialize the **Backup** process, and in the previous example, it will generate an *ISO* image and it will copy the data to be used in the **Restore** process. The example output of the command **`rear mkbackup -v`** is as follows:

```
Relax-and-Recover 2.6 / 2020-06-17
Running rear mkbackup (PID 153257)
Using log file: /var/log/rear/rear-rhel.log
Running workflow mkbackup on the normal/original system
Using backup archive
'/var/tmp/rear.0fPsx7MCTd1P1TS/outputfs/rhel/backup.tar.gz'
Using autodetected kernel '/boot/vmlinuz-5.14.0-
70.22.1.el9_0.x86_64' as kernel in the recovery system
Creating disk layout
Overwriting existing disk layout file
/var/lib/rear/layout/disklayout.conf
Using guessed bootloader 'GRUB' (found in first bytes on
/dev/vda)
```

```
Verifying that the entries in
/var/lib/rear/layout/disklayout.conf are correct …
Creating recovery system root filesystem skeleton layout
Cannot include default keyboard mapping (no 'defkeymap.*' found
in /lib/kbd/keymaps)
To log into the recovery system via ssh set up
/root/.ssh/authorized_keys or specify SSH_ROOT_PASSWORD
Copying logfile /var/log/rear/rear-rhel.log into initramfs as
'/tmp/rear-rhel-partial-2022-11-13T09:46:37+00:00.log'
Copying files and directories
Copying binaries and libraries
Copying all kernel modules in /lib/modules/5.14.0-
70.22.1.el9_0.x86_64 (MODULES contains 'all_modules')
Copying all files in /lib*/firmware/
Testing that the recovery system in
/var/tmp/rear.0fPsx7MCTd1P1TS/rootfs contains a usable system
Creating recovery/rescue system initramfs/initrd initrd.cgz
with gzip default compression
Created initrd.cgz with gzip default compression (424497692
bytes) in 21 seconds
Making ISO image
Wrote ISO image: /var/lib/rear/output/rear-rhel.iso (418M)
Copying resulting files to nfs location
Saving /var/log/rear/rear-rhel.log as rear-rhel.log to nfs
location
Copying result files '/var/lib/rear/output/rear-rhel.iso
/var/tmp/rear.0fPsx7MCTd1P1TS/tmp/VERSION
/var/tmp/rear.0fPsx7MCTd1P1TS/tmp/README
/var/tmp/rear.0fPsx7MCTd1P1TS/tmp/rear-rhel.log' to
/var/tmp/rear.0fPsx7MCTd1P1TS/outputfs/rhel at nfs location
Making backup (using backup method NETFS)
Creating tar archive
'/var/tmp/rear.0fPsx7MCTd1P1TS/outputfs/rhel/backup.tar.gz'
Archived 1867 MiB [avg 10929 KiB/sec] OK
Archived 1867 MiB in 176 seconds [avg 10866 KiB/sec]
Exiting rear mkbackup (PID 153257) and its descendant processes
…
```

```
Running exit tasks
```

In the **NFS Server** system, a directory with the name of the client is created, including the *ISO* image, a file **`backup.tar.gz`** to be used to restore it during the **Restore** process, and two log files (**`rear-node.log`** and **`backup.log`**) containing information about the copy. *Figure 14.13* shows the files created:

```
root@ubuntu:~# ls -lh /storage/rhel/
total 2.3G
-rw------- 1 nobody nogroup 5.3M Nov 13 09:50 backup.log
-rw------- 1 nobody nogroup 1.9G Nov 13 09:50 backup.tar.gz
-rw------- 1 nobody nogroup  202 Nov 13 09:47 README
-rw------- 1 nobody nogroup 418M Nov 13 09:47 rear-rhel.iso
-rw------- 1 nobody nogroup 132K Nov 13 09:47 rear-rhel.log
-rw------- 1 nobody nogroup    0 Nov 13 09:50 selinux.autorelabel
-rw------- 1 nobody nogroup  268 Nov 13 09:47 VERSION
```

Figure 14.13: *Example listing copied files on the NFS server*

The **Restore** process for this example includes booting the system to restore or a new one which would be the replacement, with the *ISO* image generated. It will mount the *NFS* storage automatically to transfer the **`backup.tar.gz`** and perform the required actions to **Restore** the system to its original state. *Figure 14.14* shows the boot menu when the system is booting:

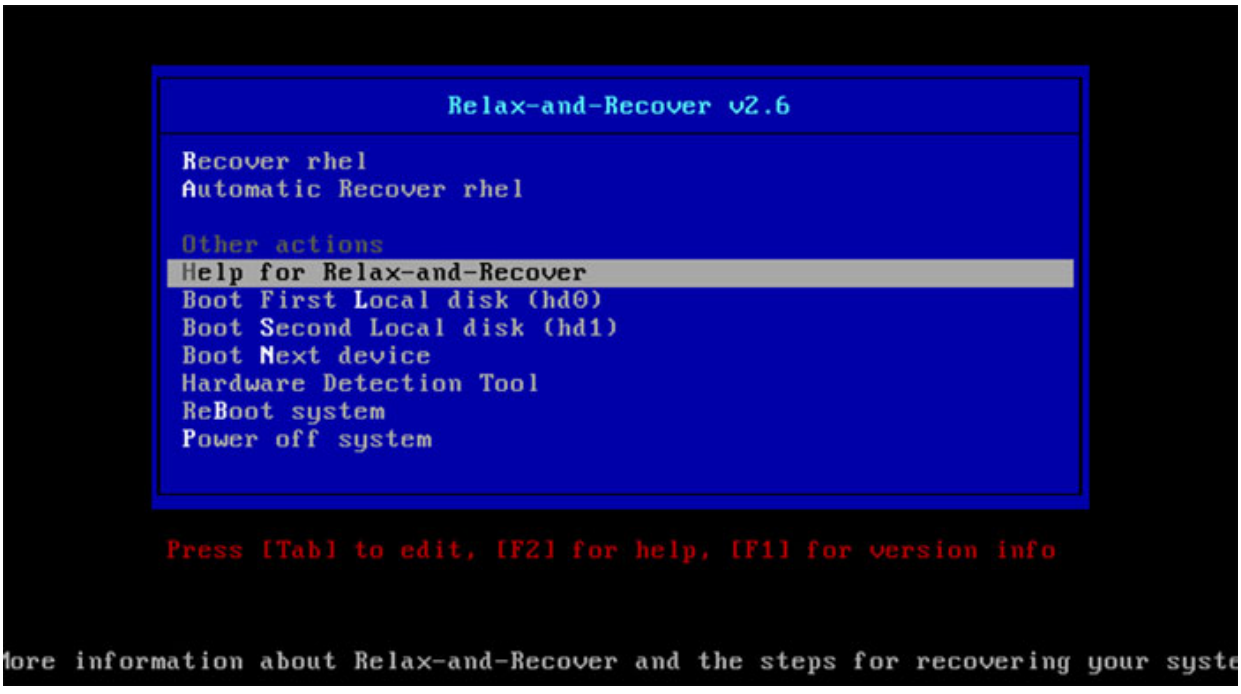


Figure 14.14: *Example boot menu for* ***ReaR***

The option **Automatic Recover rhel** will automatically perform all the operations required to restore the system without performing more questions. *Figure 14.15* shows an example of the initialized **Recover** process.

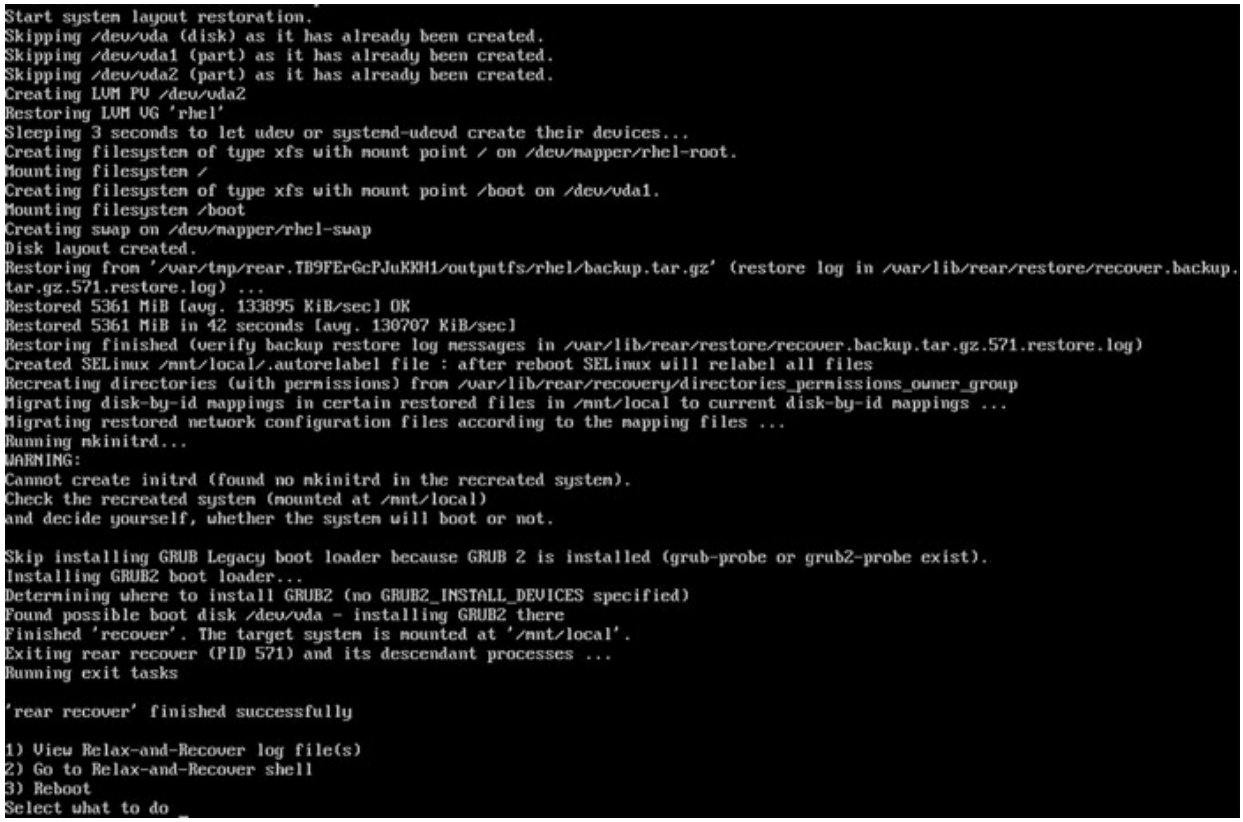


Figure 14.15: *Example restore the system using* ***ReaR****.*

Conclusion

Good **Backup** planning defines the strategy, the **Restore** tests to perform, and the storage media to use and specify the **Recovery Time Objective (RTO)** and the **Recovery Point Objective (RPO)**. Usually, in enterprises, **Backup** is underrated and only becomes a priority when a disaster occurs. With the adoption of the **Cloud** and new technologies such as **Containers**, the **Backup** is still relevant, but the approach is different. New systems can be recreated from scratch using *Automation*, and applications are stored as *Images*. Thus, only the required data is needed to store it.

There are different commercial **Backup Solutions** such as *EMC Networker, HP Data Protector,* or *Symantec NetBackup.* Moreover, there are mature

open-source solutions with support to be used. This chapter covered **Bacula** as the main open-source solution and **Relax-and-Recover (ReaR)** as a perfect bare metal disaster recovery solution.

Key facts

- Good **Backup planning** is required to avoid disaster situations.
- Testing to **Restore** frequently is needed to ensure the integrity of the copies and the recovery protocols.
- Modern systems require a rethinking about what should be stored on a **Backup**.
- A **Disaster Recovery** plan is useful and required for critical environments.
- **Bacula** is a mature and enterprise-ready open-source **Backup solution**.
- On *Bare Metal* environments, the solution **Relax-and-Recover (Rear)** is a perfect one.

Questions

1. A Differential Backup requires less space than an Incremental Backup.
    a. True
    b. False
2. The 3-2-1 Rule consists of three copies, two different storage media, and one copy off-site.
    a. True
    b. False
3. What is the name of the feature to avoid data duplication in a Backup Solution?
    a. reduplication
    b. deduplication
    c. multiplexing

4. Which main command is used in Bacula to obtain information and run jobs?
    a. bacula
    b. bacula-cli
    c. bconsole
5. What argument for the command rear is used to perform a Backup?
    a. backup
    b. mkbackup
    c. run

Answers

1. b
2. a
3. b
4. c
5. b

CHAPTER 15

Multi Cloud Management

Introduction

As described throughout this book, many companies have modernized their **IT Infrastructure** by moving to **Virtualization** (not only computing but networking and storage) and to **Containerized** environments. Another big transformation is the move from the **On-Premise** infrastructure to a **Public Cloud**. This migration comes with several advantages, but there are also some considerations to take into account.

The opportunity to work with several **Cloud** environments, such as **Amazon Web Servers, Microsoft Azure, Google Cloud,** or **Alibaba Cloud**, brings new challenges in the automation and in management of the environments. The combination of a **Private Cloud**, such as **OpenStack**, with a public provider is called **Hybrid Cloud**.

This chapter will focus on the introduction of the different **Cloud** providers, the different services offered, and how they are compared among them. It would describe the advantages of using a **Public Cloud** as well as the disadvantages. It will describe the different strategies to manage **Multiple Clouds** and the use of the automation tool **Terraform** for it.

Structure

In this chapter, we will discuss the following topics:

- Introduction to Cloud Providers
- Cloud Services
- Multi Cloud Management
- Infrastructure as code
- Terraform

Introduction to cloud providers

A **Cloud Provider** is responsible for offering the different services to be consumed. **Cloud Providers** have evolved during the last decades from a simple offering of **Compute services** and **Storage Services** to more than hundreds of services. A **Cloud Provider** is responsible for all the infrastructure, including storage, networking, computing, and security.

Some of the categorized services offered these days are the following (categories offered by **Amazon Web Services**):

- **Infrastructure software**: The services in this category provide infrastructure-related solutions.
    - **Backup and recovery**: Services for storage and backup.
    - **Data analytics**: Services used for data analysis.
    - **High-performance computing**: High-performance computing services.
    - **Migration**: Services used for migration projects.
    - **Network infrastructure**: Services used to create networking solutions.
    - **Operating systems**: Images for Linux and Windows operating systems.
    - **Security**: Security services for the infrastructure.
    - **Storage**: Services related to storage.
- **DevOps**: The services in this category provide tools focused on developers and developer teams.
    - **Agile Lifecycle Management**: Services used for Agile.
    - **Application Development**: Services used for application development.
    - **Application Servers**: Servers used for application development.
    - **Application Stacks**: Stacks used for application development.
    - **Continuous Integration and Continuous Delivery**: Services used for CI/CD.
    - **Infrastructure as Code**: Services used for infrastructure.

    - **Issues and Bug Tracking**: Services used by developer teams to track and manage software bugs.
    - **Monitoring**: Services used for monitoring operating software.
    - **Log Analysis**: Services used for logging and log analysis.
    - **Source Control**: Tools used to manage and maintain source control.
    - **Testing**: Services used for automated testing of software.
- **Business Applications**: The services in this category help run the business.
    - **Blockchain**: Services used for blockchain.
    - **Collaboration and Productivity**: Services used to enable collaboration.
    - **Contact Center**: Services used for enabling Contact Centers in the organization.
    - **Content Management**: Services focused on content management.
    - **CRM**: Tools focused on customer relationship management.
    - **eCommerce**: Services that provide *eCommerce* solutions.
    - **eLearning**: Services that provide *eLearning* solutions.
    - **Human Resources**: Services used for enabling *Human Resources* in the organization.
    - **IT Business Management**: Services used for enabling *IT business management* in the organization.
    - **Business Intelligence**: Services used for enabling business intelligence in the organization.
    - **Project Management**: Tools for project management.
- **Machine Learning**: The services in this category provide machine learning algorithms and model packages.
- **IoT:** Services used to create IoT-related solutions.
- **Desktop applications:** The services in this category provide infrastructure-related solutions.
- **Data Products**: The services in this category are sets of file-based data.

The most popular **Cloud Provider**, these days in the market, are the following ones:

- **Amazon Web Services (AWS)**: The pioneer of offering **Cloud Services**, it dominates the market.
- **Microsoft Azure**: The second biggest **Cloud Provider** in the market, offering good integration with companies using **Microsoft** products.
- **Google Cloud Platform (GCP):** Google growing in the **Cloud** market in the last few years and is the third biggest platform.
- **Alibaba Cloud**: This is the fourth on the market and is part of the giant Chinese group *Alibaba Group.*

On the **Private Cloud** solutions, the most popular open-source solution is **OpenStack**. **OpenStack** is a mature solution with the support of several companies, such as **Red Hat**, **Mirantis**, and **Canonical.** The decision to choose one **Cloud Provider** requires several considerations, and some of the important ones are the following:

- **Cost Saving**: Migrations and modernization are based on the reduction of costs. As the **Cloud Provider** is the one responsible for keeping the infrastructure, it will ensure that only the services that are in use will be paid for. Each **Cloud Provider** offers different prices depending on the region, the service to be used, as well as the usage.
- **Availability**: Most of the **Cloud Providers** offer different regions and availability zones, but in some specific cases, it can be a decision depending on the use case.
- **Services**: In the **Cloud** competition, most of the providers offer similar services. But specific and advanced ones can make a difference when a decision has to be taken.
- **Ease to Use**: Some **Cloud Providers** are more user-friendly than others, including documentation and the use of the *API*.
- **Migration**: Depending on the services offered *On-Premise*, some **Cloud Providers** are more suitable to perform the migration to a **Cloud Service**.

Advantages

The use of a **Cloud Provider** has several advantages compared to running services on an *On-Premise* infrastructure. The main reasons for a migration to a **Cloud** are the following:

- **Service Level Agreement (SLA)**: Most of the **Cloud Providers** guarantee that the services are available 99.99% of the time when different availability zones are in use.
- **Services catalog:** In a **Cloud Provider**, it is possible to run *Virtual Machines, Containers*, and they also have a *Load Balancer* or a *Backup Solution*, among hundreds of other options, together in the same portal. A cloud provider also enables more services immediately.
- **Infrastructure**: The end user is not required to maintain the infrastructure anymore, including the physical components (servers, routers, cables, among others) and the virtual components (routing, networking, storage, and so on).
- **Flexibility**: It is possible to dynamically change the resources allocated, depending on the needs of the services offered.
- **Scalability:** More compute power, more storage, or more services can be added manually or automatically on demand.
- **Accessibility: Cloud Providers** offer access to different services through the **Internet**, which allows them to offer a service directly to the end users.
- **Security**: The solutions offered by the **Cloud Providers** are secure, and they offer alternatives to increasing security, such as encryption for the storage, private network, and firewall.

As with every solution, there are some problems associated with the use of a **Cloud Provider**, especially a public one. Some of them are the following:

- **Cost**: Even though one of the main reasons for using a **Public Cloud** is to reduce the cost, the wrong planning, bad implementation of workflows, and bad modernization can cause the cost to run the services, which further increases the cost in a **Public Cloud**. It is important in migration to define the needs, plan the possible costs and monitor the usage of the services.
- **Vendor lock-in**: Migration to a **Public Cloud** means adapting and using the services offered by the provider. This can reduce the

possibilities in the future of the need to go to another solution more suitable for the company.

- **Less control**: Although having the infrastructure managed by the provider is an advantage, it can be a disadvantage as well if you are not able to control the different elements involved in a service offered.
- **Support**: The support offered by **Cloud Providers** cannot meet the needs of some companies, especially in the new implementations of the services, as it requires more time than desired.

Cloud services

Let us now discuss a few cloud services.

Amazon Web Services (AWS)

AWS started to offer its first services in the year 2006. At the moment of writing this book, the global infrastructure contains 29 regions, each of them with multiple availability zones (AZs) and 93 availability zones. The latest estimation is that **AWS** has a 34% market share of the **Cloud Service**. Some of the main services offered by **Amazon Web Services** are as follows:

- **Amazon Elastic Compute Cloud (EC2)**: The service to run *Virtual Machines* and the possibility to run *Bare Metal* nodes on the **Cloud**.
    - **EC2** offers most of the **Linux Distribution** options available in the market, including **Red Hat Enterprise Linux**, **Ubuntu**, or **Debian.**
    - An **Amazon Machine Image** (**AMI**) contains the operating system to be used to run the **EC2** instance.
    - Running an **EC2 instance** requires to specify a **Security Group**, which is the definition of the *Firewall* rules.
    - It is possible to specify different *Storage* options depending on the needs of the *Virtual Machine*.
    - A **Key Pair** defines an *SSH Public Key* to access the *Instance* remotely.
- **Amazon Virtual Private Cloud** (**VPC**): Enables a *Virtual Network* to run instances on it. This makes a private and isolated network. As a

regular *Network*, it includes *subnets*, *routing, and IP addresses,* among other elements.

    - An **Elastic IP** address is a public address to access the **EC2 Instance** from the **Internet**. This **IP** can be attached and detached from a running instance.

- **Amazon Simple Storage Service** (**S3**): An object storage service offering scalability, data availability, security, and performance.
- **Amazon Elastic Beanstalk**: A service for deploying and scaling Web applications.
- **Amazon Relation Database Service** (**RDS**): A service to set up, operate, and scale databases in the cloud. The options available are as follows:

    - **Amazon Aurora** with **MySQL** compatibility.
    - **Amazon Aurora** with **PostgreSQL** compatibility.
    - **MySQL**, **MariaDB**, **PostgreSQL**, **Oracle**, and **SQL Server**.

- **Amazon DynamoDB**: Fast, flexible **NoSQL** database service for performance at any scale.
- **Amazon Elastic Container Service** (**ECS**): A fully managed **Container Orchestration** service that makes it easy to deploy, manage, and scale containerized applications.
- **Amazon Elastic Kubernetes Service** (**EKS**): A service to run Kubernetes in the *AWS cloud* and on-premises data centers.
- **Amazon Elastic Load Balancing** (**ELB**): A service to distribute incoming application traffic across multiple targets.
- **Amazon CloudFront**: A **Content Delivery Network** (**CDN**) service built for high performance and security.
- **Amazon Route 53**: A highly available and scalable **Domain Name System** (**DNS**) Web service.
- **Amazon Lambda**: A serverless, event-driven compute service that lets codes run for virtually any type of application or backend service without provisioning or managing servers.
- **AWS Identity and Access Management (IAM)**: Securely manage identities and access to **AWS** services and resources.

Microsoft Azure

Microsoft Azure started in the year 2008 and was officially released in the year 2010. The name was adopted in the year 2014, although it was previously named *Windows Azure*. At the time of writing this book, there are 60 regions available. The latest estimation considers that **Microsoft Azure** has 21% of the market share. The main services offered are the following:

- **Azure Virtual Machines**: The service to run *Virtual Machines*. It is able to run **Linux** distribution or **Windows Server**.
    - **Azure Marketplace** includes certified operating systems and applications to run.
    - By creating a **Virtual Machine**, it is possible to specify, among many options:
        - The size (CPU and Memory)
        - The storage type and the size
        - The authentication (*SSH Public Key***,** for example)
        - Configuring the *Firewall* rules
        - The **Virtual network** ***(*****Vnet*****)*** to attach it.
- **Azure Virtual Network** (**VNet**): The service to manage private networks. Although similar to a traditional network in a data center, it brings with it additional benefits of **Azure's** infrastructure, such as scale, availability, and isolation.
    - **Azure** uses the term *Public IP* to the external addresses, which can be attached to a **Virtual Machine**.
- **Azure Blob Storage**: An object storage solution for the cloud. It is optimized for storing massive amounts of unstructured data. Unstructured data is data that does not adhere to a particular data model or definition, such as text or binary data.
- **Azure Web Apps**: It is an *HTTP*-based service for hosting Web applications, *REST APIs*, and mobile backends.
- **Azure Cloud Services**: Create highly-available, infinitely-scalable cloud applications and APIs using *Azure Service Manager*.

    - **Azure Resource Manager** is the deployment and management service for Azure. It provides a management layer that enables you to create, update, and delete resources in your *Azure* account.
- **Azure SQL**: Migrate, modernize, and innovate on the modern SQL family of cloud databases
    - **SQL Server Stretch Database**: It dynamically stretches warm and cold transactional data from Microsoft SQL Server 2016 to **Microsoft Azure.**
    - **Azure Database for MySQL**: A relational database service in the Microsoft cloud that is built for developers and powered by the MySQL community edition.
    - **Azure Database for PostgresSQL**: A relational database service based on the open-source Postgres database engine.
    - **Azure SQL Database Edge**: An optimized relational database engine geared for IoT and IoT Edge deployments.
- **Azure Cosmos DB**: Fast **NoSQL** database with *SLA*-backed speed and availability, automatic and instant scalability, and open-source *APIs* for **MongoDB** and **Cassandra**.
- **Azure Kubernetes Service** (**AKS**): Offers the quickest way to start developing and deploying cloud-native apps in **Azure**, data centers, or at the edge with built-in code-to-cloud pipelines.
- **Azure Container Instances:** Run **Docker** containers on-demand in a managed, serverless **Azure** environment.
- **Azure Load Balancer**: Load-balance internet and private network traffic with high performance and low latency.
- **Azure Content Delivery Network** (**CDN**): Offers a global solution for rapidly delivering content. Reduces load times for websites, mobile apps, and streaming media.
- **Azure DNS**: A hosting service for *DNS* domains that provides name resolution by using **Microsoft Azure** infrastructure.
- **Azure Functions**: Provides serverless computing for *Azure*.
- **Azure Active Directory** (**AD**): An enterprise identity service that provides single sign-on and multifactor authentication.

Google Cloud Platform

Google Cloud Platform started to offer some services as a preview on the year 2009. It was released as **General Available** (**GA**) in the year 2011. At the moment of writing this book, it has 35 regions and 106 zones available. The latest estimation considers **Google Cloud Platform** as having 11% of the market share. The main services offered are the following:

- **Compute Engine**: Secure and customizable compute service to create and run virtual machines on Google's infrastructure.
    - **Compute Engine** offers many preconfigured public images that have compatible Linux or Windows operating systems.
    - By creating an *Instance*, it is possible to specify, among many options:
        - The resources assigned (CPU and Memory)
        - The storage type and the size.
        - Configure the *Firewall* rules.
        - The authentication (*SSH Public Key,* for example)
        - The network (*VPC*) to attach it.
- **Virtual Public Cloud** (**VPC**): Global virtual network that spans all regions. Single VPC for an entire organization, isolated within projects.
    - **Google Cloud** uses the term external IP for the public IPs accessible from outside.
- **Cloud Storage**: The managed service for storing unstructured data. Store any amount of data and retrieve it whenever needed.
- **Google App Engine**: Build monolithic server-side rendered websites. *App Engine* supports popular development languages with a range of developer tools.
- **Cloud SQL**: Fully managed relational database service for **MySQL**, **PostgreSQL**, and **SQL Server**.
- **Cloud Spanner**: Fully managed relational database with unlimited scale, strong consistency, and high availability.
- **Datastore**: A highly scalable NoSQL database for your Web and mobile applications.

- **Kubernetes Engine (GKE)**: A simple way to automatically deploy, scale, and manage **Kubernetes**.
- **Cloud Load Balancing**: High-performance, scalable load balancing on **Google Cloud**.
- **Cloud CDN**: Fast, reliable Web and video content delivery with global scale and reach.
- **Cloud DNS**: Reliable, resilient, and low-latency DNS serving from *Google's* worldwide network with everything you need to register, manage, and serve your domains.
- **Cloud Functions**: Run your code in the cloud with no servers or containers to manage with our scalable, pay-as-you-go **functions as a service** (**FaaS**) product.
- **Identity and Access Management** (**IAM**): Allows administrators to authorize who can take action on specific resources.

Alibaba Cloud

Alibaba Cloud started to offer different services in the year 2009. At the moment of writing this book, it has 24 regions and 74 zones available. The latest estimation considers **Alibaba Cloud** has the 5% of the market share. The main services offered are the following:

- **Elastic Compute Service**: The service to run *Virtual Machines* on the *Cloud*.
- **Alibaba Cloud Marketplace** contains multiple operating systems, distributions, and applications ready to run on the **Cloud**.
- **Virtual Private Cloud**: A virtual private cloud service that provides an isolated cloud network to operate resources in a secure environment.
- **Object Storage Service** (**OSS**): Fully managed object storage service to store and access any amount of data from anywhere.
- **Enterprise Distributed Application Service**: A PaaS platform for a variety of application deployment options and microservices solutions.
- **ApsaraDB RDS**: is a stable, reliable, and scalable online relational database service that is built on top of *the Apsara Distributed File System.*

    - **ApsaraDB RDS for MySQL**: A fully hosted online database service that supports **MySQL** 5.5, 5.6, 5.7, and 8.0.
    - **ApsaraDB RDS for PostgreSQL**: An on-demand database hosting service for **PostgreSQL** with automated monitoring, backup, and disaster recovery capabilities
    - **ApsaraDB RDS for SQL Server**: An on-demand database hosting service for **SQL Server** with automated monitoring, backup, and disaster recovery capabilities
- **PolarDB-X**: Designed to address database challenges such as ultra-high concurrency, massive data storage, and large table performance bottlenecks.
- **ApsaraDB for MongoDB**: A secure, reliable, and elastically scalable cloud database service for automatic monitoring, backup, and recovery by time point.
- **Elastic Container Instance**: An agile and secure serverless container instance service.
- **Alibaba Cloud Container Service for Kubernetes (ACK)**: A *Kubernetes*-based service that ensures high efficiency for enterprises by running containerized applications on the cloud.
- **Server Load Balancer (SLB)**: Distributes network traffic across groups of backend servers to improve service capability and application availability.
- **Alibaba Cloud CDN**: A fast, stable, secure, and custom content delivery service for accelerated content distribution to users worldwide.
- **Alibaba Cloud DNS**: A secure, fast, stable, and reliable authoritative DNS service.
- **Function Compute**: A secure and stable *serverless* computing platform.
- **Resource Access Management (RAM)**: Allows you to centrally manage access to **Alibaba Cloud** services and resources**.**

The website **https://comparecloud.in/** includes a list of the services and the equivalent for each of the popular public **Cloud Providers** (including **IBM Cloud**, **Oracle Cloud**, and **Huawei Cloud**). *Figure 15.1* shows a table with some of the services described previously:

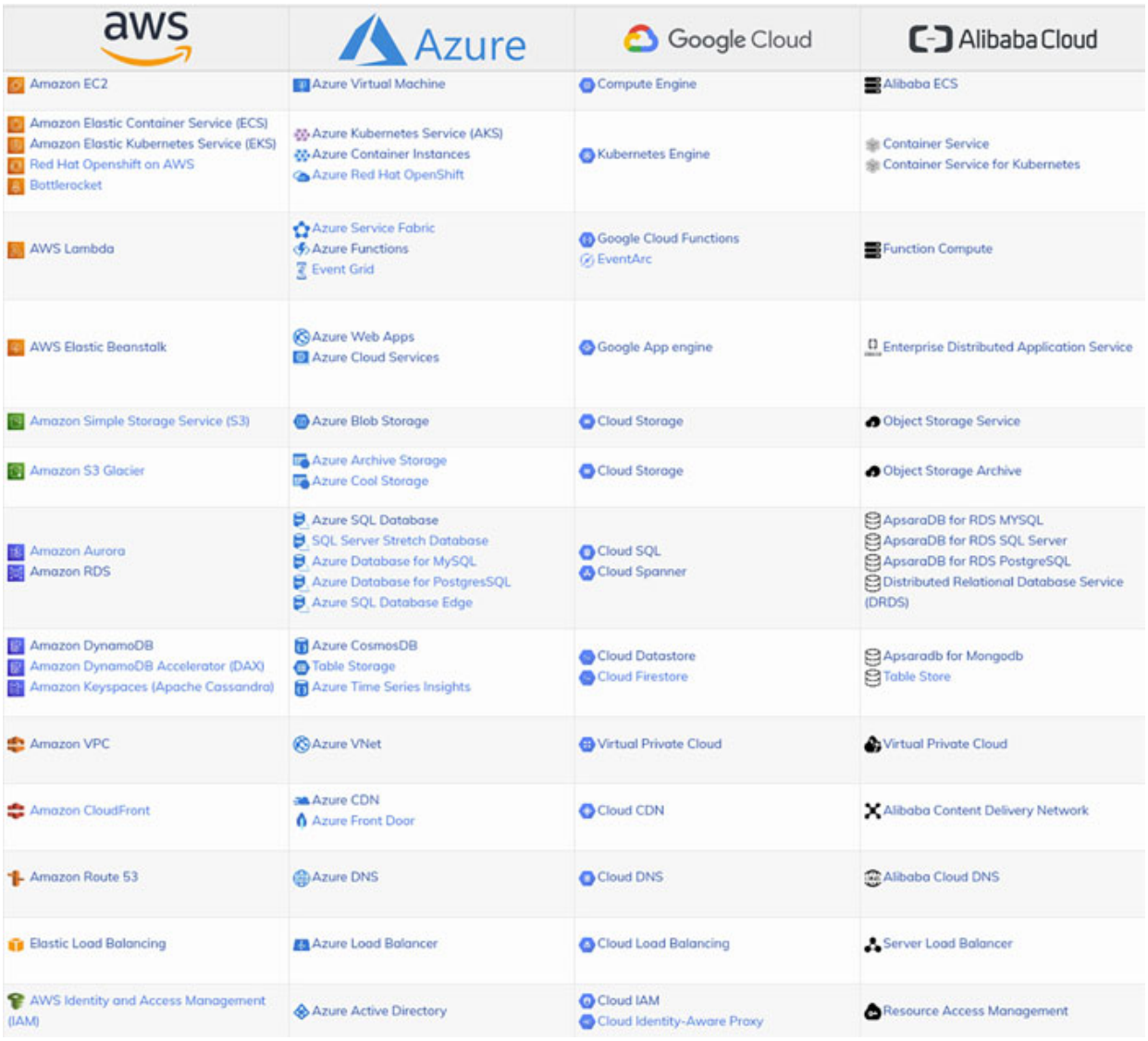


Figure 15.1: *Cloud services comparison*

OpenStack

The cloud solution **OpenStack** is an open-source platform usually deployed to have a **Private Cloud** in an *on-premise* environment. This mature solution started in the year 2010 and is used by thousands of companies in different sectors. It is possible to integrate with the infrastructure existing, such as storage or networking available.

The services offered by **OpenStack** are similar to the most used ones in a **Public Cloud** solution. The core services on OpenStack are described in the following list:

- **Nova**: Provides a way to provision compute instances (also known as *virtual servers*).
- **Neutron**: Handles the creation and management of virtual networking infrastructure, including networks, switches, subnets, and routers for devices managed by *OpenStack*.
- **Keystone**: Provides API client authentication and multi-tenant authorization.
- **Horizon**: Provides a Web-based user interface to *OpenStack* services, including *Nova*, *Swift*, *Keystone*, and so on.
- **Cinder** *(Block Storage Service)*: The service for providing volumes to **Nova** virtual machines.
- **Glance** (*Image Service)*: The service where users can upload and discover images to be used for *Virtual Machines*.
- **Swift** *(Object Storage Service)*: A highly available, distributed, and eventually consistent object/blob store.
- **Ironic**: Service to provision *bare metal* (as opposed to virtual) machines.
- **Heat**: Service to orchestrate composite cloud applications using a declarative template format.

Refer to *figure 15.2* to see the OpenStack components:

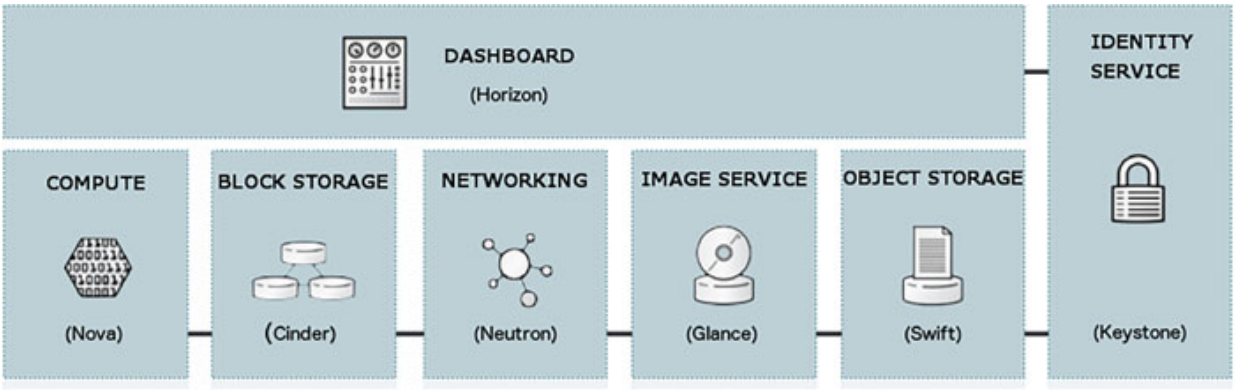


Figure 15.2: *OpenStack components*

Other useful services are also available optionally on *OpenStack*:

- **Octavia**: It brings network load balancing to *OpenStack*.

- **Barbican** *(*Key Manager Service*)*: It provides secure storage, provisioning, and management of secret data.
- **Manila**: The service to provide *Shared Filesystems* as a service.
- **Designate**: An Open-Source *DNS-as-a-Servic*e implementation.

There are different companies offering commercial support for **OpenStack**. Some examples are as follows:

- **Red Hat OpenStack Platform**: Uniquely co-engineered together with **Red Hat Enterprise Linux** to ensure a stable and production-ready cloud.
- **Canonical's Charmed OpenStack**: An enterprise cloud platform optimized for price performance and designed to run mission-critical workloads. Together with **Ubuntu**, it meets the highest security, stability, and quality standards in the industry.
- **Mirantis OpenStack for Kubernetes**: Delivers performant and scalable *IaaS* so that you can deploy, run, and scale bare-metal private clouds by leveraging the capabilities of **Kubernetes** to provide an extremely reliable and highly scalable private cloud solution.
- **VMware Integrated OpenStack** (**VIO**): An OpenStack distribution supported by *VMware*. With VIO, customers can rapidly deploy a production-grade **OpenStack** cloud on top of *VMware* technologies, leveraging their existing *VMware* investment and expertise.

Multi cloud management

One of the biggest disadvantages of using only one **Cloud Provider** is being locked into the services and the costs from one vendor. The concept of **Multi Cloud Management** is that it is a set of tools and procedures that lets us have more than one **Cloud Provider** and deploy different services depending on the needs, such as cost or availability. The services to be distributed can be simple ones, such as an application running in a *Virtual Machine*, or a complex one, such as a *Kubernetes cluster.*

These days, more companies are moving to use more than one **Cloud Provider** for the following reasons:

- **Reducing costs**: Prices of running services on the **Cloud** can be very different from one to another. If the company has decided on one

provider for its main services, it can reduce costs for some other services using a different provider.

- **Services available**: Most of the providers are offering the same services, but specific ones can be a reason to choose another provider that is different from the main one.
- **Availability**: Most of the providers are available in a big number of regions, but specific zones to offer service to users in a country or region can be a reason to deploy a service in a different provider.

The tools for **Multi Cloud Management** can be grouped into three different categories:

- **Infrastructure Automation Tools**: These tools can be used to deploy a service or application in multiple **Cloud Providers**. Some examples are as follows:
    - **Terraform**: Described later on in this chapter, Terraform is used to connect and deploy services in multiple providers.
    - **Ansible:** Allows to perform different operations in different providers in an easy way.
- **Management Tools**: The tools in this category are responsible for managing multiple clouds from only one application or portal. Some examples are as follows:
    - **Embotics**: A solution to help modern infrastructure and operations teams decrease the challenges of hybrid and multi-cloud management.
    - **Kyndryl Multicloud Management Platform:** Simplifies management of multi cloud IT architectures with comprehensive functionality and integration.
    - **Red Hat Advanced Cluster Management**: Helps to administrate different Kubernetes installations in different clouds from one dashboard.
- **Cloud Provider Solutions**: Some providers offer services to interoperate with different providers.
    - **Google Anthos:** Consistent development and operations experience for hybrid and multi cloud environments.

- **Azure Arc**: A bridge that extends the Azure platform to help you build applications and services with the flexibility to run across data centers, at the edge, and in multi cloud environments.

Infrastructure as code

With the adoption of the **Cloud**, new needs appeared regarding the deploying of new infrastructure in a fast and flexible way. There are the following two approaches for deployment:

- **Declarative**: This approach describes **what** to do, without a specific control flow, to have the desired *Infrastructure.* This approach was the most used before specific tools to deploy infrastructures appeared. It includes using specific tools for the platform to be managed, such as:
  - **AWS Command Line Interface (AWS)**: The tool to interact with the *AWS Services* from a command line. Using the command **`aws`**, it is possible to create different services, such as a *VPC*, *EC2 Instance,* or *Load Balancer,* among others.
  - **Google Cloud CLI (gcloud)**: A set of tools to create and manage resources on the *Google Cloud Platform*.
  - **Azure Command-Line Interface (CLI)**: A set of commands used to create and manage *Azure* resources.
  - **Alibaba Cloud CLI**: Includes the **`aliyun`** command to interact with and thus manage your *Alibaba Cloud* resources.
  - **Openstack CLI**: *OpenStackClient* project provides a unified command-line client, which enables access to the project *API* through the easy-to-use command **`openstack`**.
- **Imperative**: In this approach, it describes **how** the target *Infrastructure* should be. The tool used will read the definition desired, and it will ensure that the final state is the proper one. This approach has several advantages compared to the declarative one, and they are as follows:
  - **Idempotent**: The process can be executed several times; if only the resources status is not the same as the definition, it will be modified.

- **Flexibility**: The definition can be updated depending on new needs, and the target environment will be updated.
- **Removal**: The deletion of the defined infrastructure is a straight action because this tool is responsible for the proper removal.

Most of the popular *Automation Tools* offer the possibility to operate with the **Cloud Providers** using the **Imperative** approach. For example, **Ansible** can use the modules to find a proper *AMI* image, define an *SSH key*, and create an *EC2 Instance*. **Ansible** will use the specific library to communicate with *AWS's API*, called *boto3*. The following example shows a *Playbook* to create different resources on *AWS*:

```
- name: Create an EC2 Instance using Ubuntu Server image
  hosts: localhost # API calls are from the system whe Ansible
  is executed
  gather_facts: False
  tasks:
  - name: Create a VPC to be used by instances
    amazon.aws.ec2_vpc_net:
      name: net_iac_example
      cidr_block: 10.10.0.0/16
    register: r_net_vpc
  - name: Create a Subnet to be used by instances
    amazon.aws.ec2_vpc_subnet:
      vpc_id: "{{ r_net_vpc.vpc.id }}"
      cidr: 10.10.0.0/24
    register: r_subnet_vpc
  - name: Create a Key using the public one in the system
    amazon.aws.ec2_key:
      name: iac_key
      key_material: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"
    register: r_keypair
  - name: Get Ubuntu 22.04 AMIs
    amazon.aws.ec2_ami_info:
      filters:
    name: "ubuntu/images/hvm-ssd/ubuntu-jammy-22.04*"
    register: r_images
  - name: Create an instance with a public IP address
```

```
  amazon.aws.ec2_instance:
   name: "ubuntu01"
   key_name: "iac_key"
   instance_type: t2.nano
   security_group: default
   network:
  assign_public_ip: true
   image_id: "{{ r_images.images[-1].image_id }}" # Latest
   available
   vpc_subnet_id: "{{ r_subnet_vpc.subnet.id }}"
   tags:
  Environment: Testing
```

This *Playbook* can be executed several times, and it will ensure that all the resources are created with the defined parameters. One of the disadvantages of using **Ansible**, as well as the step-by-step other automation tools, is the removal process. Another *Playbook* with the reverse order needs to be created because, in the previous example, it was not possible to remove a *VPC* before removing the server and the subnet. The following example shows the correct order for the deletion:

```
- name: Unprovision an Infrastructure
 hosts: localhost # API calls are from the system whe Ansible
 is executed
 gather_facts: False
 tasks:
 - name: Get instance to be removed
  amazon.aws.ec2_instance:
   filters:
  "tag:Name": "ubuntu01"
  register: r_instance
 - name: Remove existing instance
  amazon.aws.ec2_instance:
   state: absent
   instance_ids:
  - "{{ r_instance.instances[0].instance_id }}"
 - name: Remove existing Key
  amazon.aws.ec2_key:
   name: iac_key
```

```
    state: absent
- name: Get VPC to be removed
  amazon.aws.ec2_vpc_net_info:
    filters:
  "tag:Name": "net_iac_example"
  register: r_vpc_net
- name: Remove existing Subnet
  amazon.aws.ec2_vpc_subnet:
    cidr: 10.10.0.0/24
    vpc_id: "{{ r_vpc_net.vpcs[0].id }}"
    state: absent
- name: Remove existing VPC
  amazon.aws.ec2_vpc_net:
    name: net_iac_example
    cidr_block: 10.10.0.0/16
    state: absent
```

Cloud Providers offer an *Infrastructure* as a *Code* orchestration template system. By writing a definition based on the template provided, it is possible to create resources and manage them. The biggest advantage is the creation of resources based on the templates, and this is automatically managed by the **Cloud Provider** instead of the tool. The orchestration system is also responsible for resolving the dependencies, and in case there is a removal request, it will perform it in the correct order. The biggest disadvantage is the syntax, which is usually more complex than simply using an *Automation Tool*, and it varies from one **Cloud** to another. The options available for the common providers are as follows:

- **AWS CloudFormation**: Creating a template that describes all the *AWS* resources wanted (such as *Amazon EC2* instances or *Amazon RDS DB* instances), **CloudFormation** takes care of provisioning and configuring those resources from the definition.
- **Google Cloud Deployment Manager**: An infrastructure deployment service that automates the creation and management of ***Google Cloud*** resources. It also writes flexible templates and configuration files and uses them to create deployments that have a variety of *Google Cloud* services.

- **Azure Resource Manager**: The deployment and management service for *Azure*. It provides a management layer that enables the creation, updating, and deleting of resources in an *Azure* account.
- **Alibaba's Resource Orchestration Service (ROS)**: The *ROS* template syntax defines cloud computing resources and resource dependencies in a template.

An example of an *AWS* orchestration template based on *Ansible's Playbook* is as follows:

```
Parameters:
  ParamInstanceType:
    Description: A flavor to be used by the instance
    Default: t2.nano
    Type: String
  ParamKeyValue:
    Description: SSH Public Key value
    Default: none
    Type: String
Resources:
  NetIac:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.10.0.0/16
      EnableDnsSupport: 'true'
      EnableDnsHostnames: 'true'
      Tags:
      - Key: Name
    Value: net_iac_example
  SubnetIac:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref NetIac
      CidrBlock: 10.10.0.0/24
  UserKey:
    Type: AWS::EC2::KeyPair
    Properties:
      KeyName: UserKey
```

```
    PublicKeyMaterial: !Ref ParamKeyValue
  Ubuntu01:
   Type: "AWS::EC2::Instance"
   Properties:
    ImageId: "ami-043a52c87b956fc71"
    InstanceType: !Ref ParamInstanceType
    KeyName: !Ref UserKey
    NetworkInterfaces:
   - AssociatePublicIpAddress: "true"
    DeviceIndex: "0"
    SubnetId: !Ref SubnetIac
    Tags:
    - Key: Name
   Value: ubuntu01
```

Once the *Orchestration Template* is created, the *Automation Tools* can use the *API* of the *Cloud Provider* to start the orchestration. The following example shows how **Ansible** uses the module **`amazon.aws.cloudformation`** for that purpose:

```
- name: Create resources using Cloudformations
  hosts: localhost
  gather_facts: False
  tasks:
  - name: Create a Cloudformation Stack
    amazon.aws.cloudformation:
      stack_name: "ansible-cloudformation"
      state: "present"
      template: "cloudformation.yaml"
      template_parameters:
    ParamKeyValue: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"
```

In this case, *AWS Cloudformation Service* is responsible for performing the operations required that are defined in the template, ensuring that the dependencies and the syntax are correct. When the operation is completed, the resources are available to the provider. *Figure 15.3* shows an example on *AWS* of the resources created with the previous template.

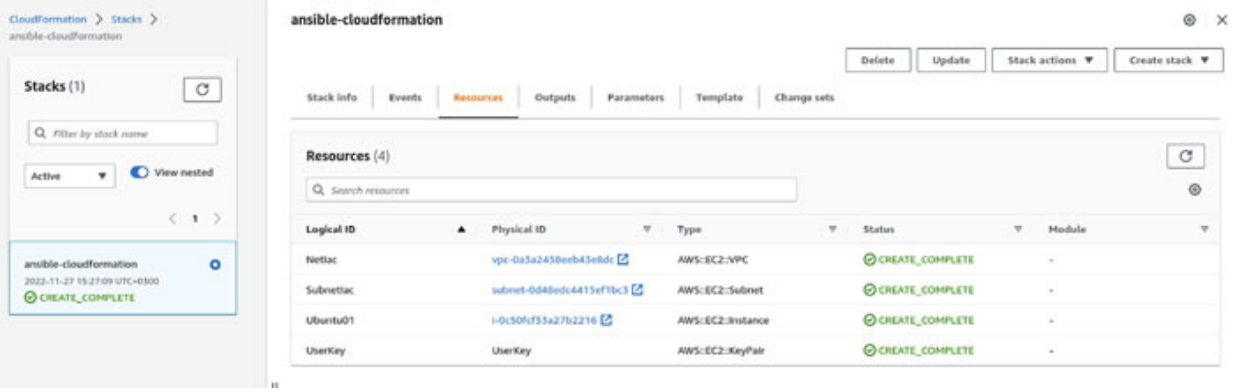


Figure 15.3: *AWS Cloudformation dashboard*

The resources can be deleted as a unique resources, deleting the *Stack* created. On **Ansible**, it is only required to change the **state** parameter from *present* to *absent.* The examples shown during this section are demonstrative only, and some resources are missing, such as a security group and *Internet Gateway* to provide external connectivity.

In the next section of this chapter, **Terraform** is described, and it will be observed how it improves the general *Automation Tools* related to the *Infrastructure as Code*. **Terraform** does not require to use of *Orchestration Templates*, and it is responsible for the creation and the deprovisioning of the resources automatically.

Terraform

Terraform is an open-source automation tool for *Infrastructure-as-code* created by *HarshiCorp*. With this solution, it is possible to create, modify, and delete infrastructures in a fast and predictable way. Using **Terraform**, it is also possible to automatize multiple **Cloud Providers**, traditional infrastructures, and different services such as *Kubernetes*.

The biggest advantage of **Terraform** is that it makes it easy for provisioning the update and the deprovisioning of the infrastructure defined. **Terraform** relies on plugins called providers to interact with cloud providers, SaaS providers, and other APIs. Each provider adds a set of resource types and/or data sources that **Terraform** can manage.

Figure 15.4 features a Terraform diagram:

Figure 15.4: *Terraform diagram. Source: Terraform*

Some of the providers officially maintained and supported by *HarshiCorp* are as follows:

- **Amazon Web Services** (**AWS**): Lifecycle management of **AWS** resources, including *EC2*, *Lambda*, *EKS*, *ECS*, *VPC*, *S3*, *RDS*, *DynamoDB*, and more.
- **Microsoft Azure**: Lifecycle management using the *Azure Resource Manager APIs*. It is maintained by the *Azure* team at *Microsoft* and the **Terraform** team at *HashiCorp*.
- **Google Cloud Platform (GCP)**: Lifecycle management of *GCP* resources, including *Compute Engine*, *Cloud Storage*, *Cloud SQL*, *GKE*, *Cloud Functions,* and more. This provider is collaboratively maintained by the *Google Terraform Team* at *Google* and the **Terraform** team at *HashiCorp*
- **Kubernetes**: Management of all **Kubernetes** resources, including *Deployments*, *Services*, *Custom Resources* (*CRs* and *CRDs*), *Policies*, *Quotas,* and more.
- **VMware vSphere**: Lifecycle management of *VMware vSphere* resources, including *Virtual Machines*, *ESXi Hosts*, *Datastores*, *vSwitches*, and more.

Partners also are offering providers for **Terraform**, which are well documented and supported. Some provider examples are as follows:

- **Alibaba Cloud**: Used to interact with the many resources supported by *Alibaba Cloud*.
- **Cloudflare**: Used to interact with resources supported by *Cloudflare*.
- **OVH**: Used to interact with the many resources supported by *OVHcloud*.

- **VMC**: Used to configure hybrid cloud infrastructure using the resources supported by *VMware Cloud on AWS*.

The community is as well sharing and collaborating with the big number of providers available. The portal **Registry Terraform** (**https://registry.terraform.io/browse/provider**) offers a complete list of them.

Terraform can be installed and used in several **Linux distributions** and operating systems. Once it is installed, there are three stages to performing the provisioning of the desired infrastructure:

- **Write**: This stage includes defining the resources desired in one or multiple cloud providers or services.
- **Plan**: The tool will create a plan describing the infrastructure, defining what will be created, updated, or deleted in the target platform.
- **Apply**: When the plan tasks are approved, this stage will perform the needed changes in the platform.

Figure 15.5 features the Terraform workflow:

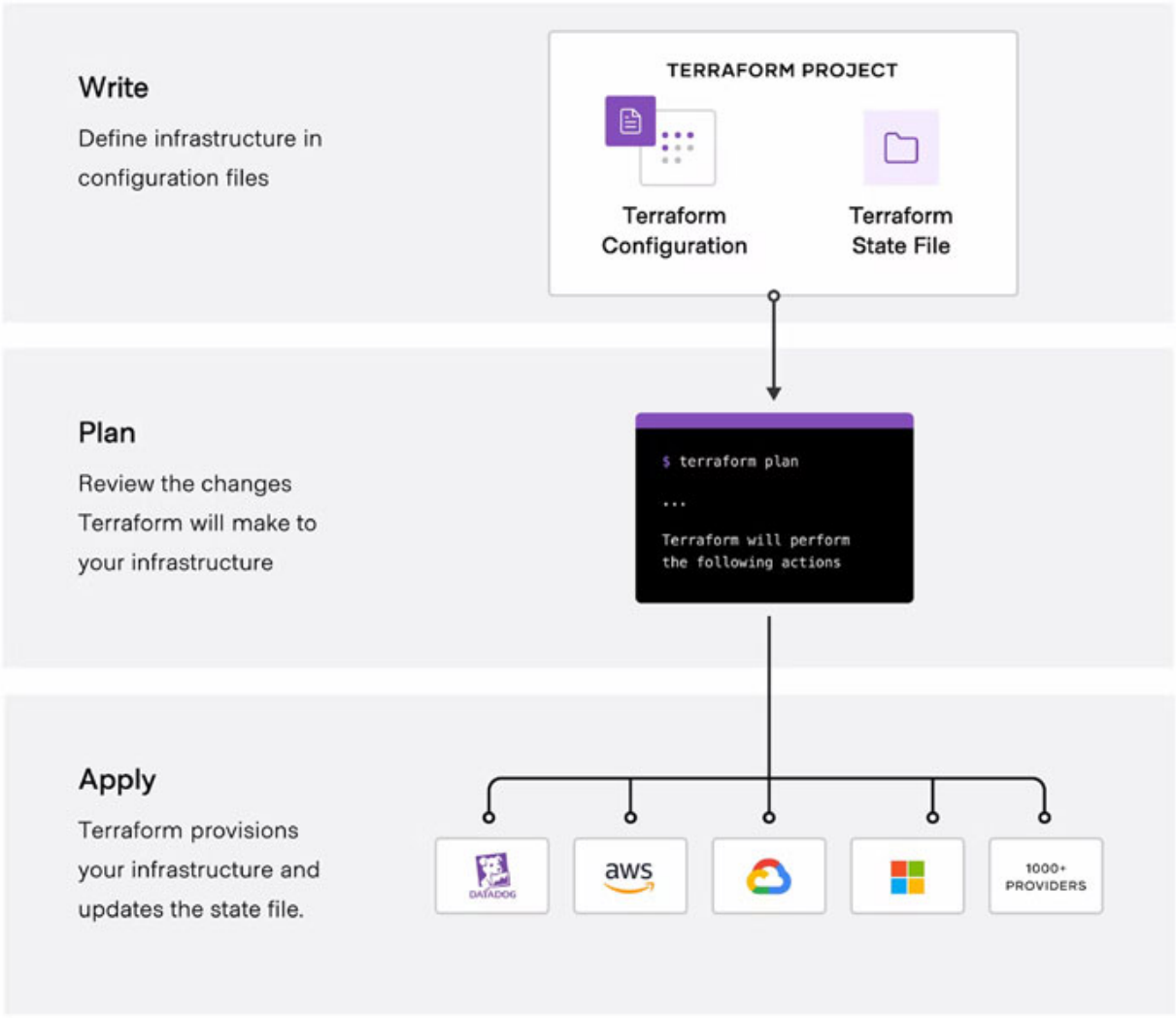


Figure 15.5: *Terraform workflow. Source: Terraform*

HarshiCorp offers a solution called **Terraform Cloud**, an environment to be used for automation instead of using a local system. The **Terraform Cloud UI** provides a detailed view of the resources managed by a Terraform project and gives enhanced visibility into each Terraform operation.

Installation

The installation of **Terraform CLI** is a simple task. The following steps define the installation in an **Ubuntu** system:

```
wget -O- https://apt.releases.hashicorp.com/gpg | gpg --dearmor
| sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg
```

```
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
sudo apt update && sudo apt install terraform
```

Running a simple Web server on AWS

The definition on **Terraform** is plain text files with an ending *.tf*. For the simplicity of this example, all the definitions are in the same file instead of being distributed in several ones (the recommended way). An existing *VPC* named **iac-vpc** is already configured on *AWS*. The content of the file **main.tf** is as follows:

```
data "aws_ami" "ubuntu" {
  most_recent = true
  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"]
  }
  owners = ["099720109477"] # Canonical
}
data "aws_vpc" "iac-vpc" {
  cidr_block       = "10.0.0.0/16"
  tags = {
    Name = "iac-vpc"
  }
}
data "aws_subnet" "iac-subnet" {
  vpc_id     = data.aws_vpc.iac-vpc.id
  cidr_block = "10.0.0.0/20"
}
data "aws_security_group" "allow_http" {
  vpc_id     = data.aws_vpc.iac-vpc.id
  tags = {
    Name = "allow_http"
  }
}
```

```
resource "aws_instance" "web01" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"
  subnet_id        = data.aws_subnet.iac-subnet.id
  associate_public_ip_address = true
  vpc_security_group_ids  =
  [data.aws_security_group.allow_http.id]
  key_name = "UserKey"
  user_data = <<-EOF
  #!/bin/bash
  sudo apt update -y
  sudo apt install apache2 -y
  sudo echo "Hello from AWS" | sudo tee /var/www/html/index.html
  EOF
  tags = {
   Name = "web01"
  }
}
output "instance_ip_addr" {
  value = aws_instance.web01.public_ip
}
```

The keyword **data** is used by **Terraform** to query data from the provider, and the keyword **`resource`** will create the resource defined. The **`output`** keyword is used to print the values required. Having a directory with the required definition and files for **Terraform**, it is required to execute **`terraform init`** to install the providers and other resources required to perform the tasks. The following output shows the installation of the *AWS* provider.

```
Initializing the backend…
Initializing provider plugins…
- Finding latest version of hashicorp/aws…
- Installing hashicorp/aws v4.41.0…
- Installed hashicorp/aws v4.41.0 (signed by HashiCorp)
```

Terraform has created a lock file **`.terraform.lock.hcl`** to record the provider selections it made previously. Include this file in your version control repository so that Terraform can guarantee to make the same selections by default when you run “terraform init” in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running the “terraform plan” to see any changes that are required for your infrastructure. All Terraform commands should now work.

If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.

After the initialization of the directory where the definition is located, the next optional step is to execute the **terraform plan** to check the syntax and the dependencies and show the tasks which will be performed. The following output shows an example:

```
data.aws_vpc.iac-vpc: Reading…
data.aws_ami.ubuntu: Reading…
data.aws_vpc.iac-vpc: Still reading… [10s elapsed]
data.aws_ami.ubuntu: Still reading… [10s elapsed]
data.aws_ami.ubuntu: Read complete after 11s [id=ami-
0a3e4ec6adf694c3f]
data.aws_vpc.iac-vpc: Still reading… [20s elapsed]
data.aws_vpc.iac-vpc: Read complete after 23s [id=vpc-
065e086222c706994]
data.aws_security_group.allow_http: Reading…
data.aws_subnet.iac-subnet: Reading…
data.aws_subnet.iac-subnet: Read complete after 0s [id=subnet-
0faf005c6d517ae9b]
data.aws_security_group.allow_http: Read complete after 0s
[id=sg-04313711a4beec2f7]
```

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:

```
+ create
```

Terraform will perform the following actions:

```
# aws_instance.web01 will be created
+ resource "aws_instance" "web01" {
    + ami                                  = "ami-
    0a3e4ec6adf694c3f"
```

```
+ arn                                    = (known after
apply)
+ associate_public_ip_address           = true
+ availability_zone                      = (known after
apply)
+ cpu_core_count                         = (known after
apply)
+ cpu_threads_per_core                   = (known after
apply)
+ disable_api_stop                       = (known after
apply)
+ disable_api_termination                = (known after
apply)
+ ebs_optimized                          = (known after
apply)
+ get_password_data                      = false
+ host_id                                = (known after
apply)
+ host_resource_group_arn                = (known after
apply)
+ id                                     = (known after
apply)
+ instance_initiated_shutdown_behavior = (known after
apply)
+ instance_state                         = (known after
apply)
+ instance_type                          = "t3.micro"
+ ipv6_address_count                     = (known after
apply)
+ ipv6_addresses                         = (known after
apply)
+ key_name                               = "UserKey"
+ monitoring                             = (known after
apply)
+ outpost_arn                            = (known after
apply)
```

```
    + password_data                          = (known after
    apply)
    + placement_group                        = (known after
    apply)
    + placement_partition_number             = (known after
    apply)
    + primary_network_interface_id           = (known after
    apply)
    + private_dns                            = (known after
    apply)
    + private_ip                             = (known after
    apply)
    + public_dns                             = (known after
    apply)
    + public_ip                              = (known after
    apply)
    + secondary_private_ips                  = (known after
    apply)
    + security_groups                        = (known after
    apply)
    + source_dest_check                      = true
    + subnet_id                              = "subnet-
    0faf005c6d517ae9b"
    + tags                                   = {
    + "Name" = "web01"
  }
    + tags_all                               = {
    + "Name" = "web01"
  }
    <<OMITTED>>
  }
```

Plan: 1 to add, 0 to change, 0 to destroy.

```
Changes to Outputs:
  + instance_ip_addr = (known after apply)
```

Note: **You did not use the -out option to save this plan, so Terraform cannot guarantee to take exactly these actions if you run "terraform apply" now.**

If no errors were found in the definition provided, it is possible to execute **terraform apply** to the provision of the definition to the target environment. The following output shows the example output for the previous example:

```
<<OMITTED>>
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.
  Enter a value: yes
aws_instance.web01: Creating…
aws_instance.web01: Still creating… [10s elapsed]
aws_instance.web01: Still creating… [20s elapsed]
aws_instance.web01: Creation complete after 28s [id=i-
0c5d6e9700ea03a06]
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
Outputs:
instance_ip_addr = "18.159.48.0"
```

Figure 15.6 shows the output connecting to port 80 of the external IP shown in the previous example:

```
root@ubuntu:~/ec2instance# curl 18.159.48.0
Hello from AWS
```

Figure 15.6: *Example accessing to Web server deployed on AWS*

Running a simple Web server on the Google Cloud Platform

In the following example using **Terraform**, the provider to be used is *GCP,* and the same approach is used. A *Virtual Machine* is created, and it will run a Web server. The following code shows the **Terraform** definition.

```
provider "google" {
  project     = "linuxserver-9mtgt"
  region      = "europe-southwest1"
}
```

```
resource "google_compute_firewall" "allow-http" {
 name    = "allow-http"
 network = "iac-net"
 allow {
  protocol = "tcp"
  ports    = ["80"]
 }
 source_ranges = ["0.0.0.0/0"]
 target_tags = ["http"]
}
resource "google_compute_instance" "ubuntu02" {
 name         = "ubuntu02"
 machine_type = "e2-medium"
 zone         = "europe-southwest1-a"
 tags         = ["http"]
 boot_disk {
  initialize_params {
   image = "ubuntu-os-cloud/ubuntu-2204-lts"
  }
 }
 network_interface {
  network = "iac-net"
  access_config {}
 }
 metadata_startup_script = <<-EOF
 #!/bin/bash
 sudo apt update -y
 sudo apt install apache2 -y
 sudo echo "Hello from GCP" | sudo tee /var/www/html/index.html
 EOF
}
output "instance_ip_addr" {
 value =
 google_compute_instance.ubuntu02.network_interface.0.access_co
 nfig.0.nat_ip
}
```

Figure 15.7 shows the output connecting to the external IP associated with the virtual machine.

```
Outputs:

instance_ip_addr = "34.175.75.227"
root@ubuntu:~/gcpinstance# curl 34.175.75.227
Hello from GCP
```

Figure 15.7: *Example accessing to Web server deployed on GCP*

Running a simple Web server on Microsoft Azure

We have to follow the same approach for the **Microsoft Provider**. The following example code performs the same operations as observed for the other providers:

```
data "azurerm_resource_group" "iac" {
  name     = "openenv-45lhc"
}
data "azurerm_subnet" "iac-subnet" {
  name                  = "iac-subnet"
  resource_group_name = data.azurerm_resource_group.iac.name
  virtual_network_name = "iac-net"
}
resource "azurerm_public_ip" "ubuntu03-fip" {
  name                = "ubuntu03-ip"
  location            = data.azurerm_resource_group.iac.location
  resource_group_name = data.azurerm_resource_group.iac.name
  allocation_method   = "Static"
}
resource "azurerm_network_security_group" "iac-sg" {
  name                = "iac-sg"
  location            = data.azurerm_resource_group.iac.location
  resource_group_name = data.azurerm_resource_group.iac.name
  security_rule {
    name                       = "SSH_HTTP"
    priority                   = 100
```

```
   direction                  = "Inbound"
   access                     = "Allow"
   protocol                   = "Tcp"
   source_port_range          = "*"
   destination_port_ranges     = ["22","80"]
   source_address_prefix      = "*"
   destination_address_prefix = "*"
  }
}
resource "azurerm_network_interface" "main" {
 name                = "iac-nic"
 location            = data.azurerm_resource_group.iac.location
 resource_group_name = data.azurerm_resource_group.iac.name
 ip_configuration {
  name                          = "testconfiguration1"
  subnet_id                     = data.azurerm_subnet.iac-
  subnet.id
  private_ip_address_allocation = "Dynamic"
  public_ip_address_id          = azurerm_public_ip.ubuntu03-
  fip.id
 }
}
resource "azurerm_virtual_machine" "main" {
 name                  = "ubuntu03"
 location              =
 data.azurerm_resource_group.iac.location
 resource_group_name   = data.azurerm_resource_group.iac.name
network_interface_ids = [azurerm_network_interface.main.id]
 vm_size               = "Standard_DS1_v2"
 delete_os_disk_on_termination = true
 storage_image_reference {
  publisher = "Canonical"
  offer     = "0001-com-ubuntu-server-jammy"
  sku       = "22_04-lts"
  version   = "latest"
 }
 os_profile_linux_config {
```

```
    disable_password_authentication = false
  }
  os_profile {
    computer_name  = "ubuntu03"
    admin_username = "ubuntu"
    admin_password = "Password1234!"
    custom_data = <<-EOF
    #!/bin/bash
    sudo apt update -y
    sudo apt install apache2 -y
    sudo echo "Hello from Azure" | sudo tee
    /var/www/html/index.html
    EOF
  }
  tags = {
    Name = "web01"
  }
  storage_os_disk {
    name              = "rootdisk"
    caching           = "ReadWrite"
    create_option     = "FromImage"
    managed_disk_type = "Standard_LRS"
  }
}
output "instance_ip_addr" {
  value = azurerm_public_ip.ubuntu03-fip.ip_address
}
```

Figure 15.8 shows the external IP, for instance, and the access to it.

```
Outputs:

instance_ip_addr = "172.173.177.139"
root@ubuntu:~/azureinstance# curl 172.173.177.139
Hello from Azure
root@ubuntu:~/azureinstance#
```

Figure 15.8: *Example accessing to Web server deployed on Azure*

Configuring DNS with Cloudflare

In the following example, the provider *Cloudflare* is used to configure a *Round-Robin DNS* record to point a subdomain to the three endpoints created previously.

```
resource "cloudflare_record" "www-aws" {
  zone_id = "7b02797059c3d78208a18ba3210e36e7"
  name    = "iac"
  value   = aws_instance.web01.public_ip
  type    = "A"
}
resource "cloudflare_record" "www-gcp" {
  zone_id = "7b02797059c3d78208a18ba3210e36e7"
  name    = "iac"
  value   =
  google_compute_instance.ubuntu02.network_interface.0.access_co
  nfig.0.nat_ip
  type    = "A"
}
resource "cloudflare_record" "www-azure" {
  zone_id = "7b02797059c3d78208a18ba3210e36e7"
  name    = "iac"
  value = azurerm_public_ip.ubuntu03-fip.ip_address
  type    = "A"
}
```

Figure 15.9 shows the resolution for the subdomain configured.

```
root@ubuntu:~/cloudfare# dig +short iac.mucloma.com
40.112.55.95
52.59.223.51
34.175.55.219
root@ubuntu:~/cloudfare# curl iac.mucloma.com
Hello from AWS
root@ubuntu:~/cloudfare# curl iac.mucloma.com
Hello from GCP
root@ubuntu:~/cloudfare# curl iac.mucloma.com
Hello from Azure
```

Figure 15.9: *Example accessing to Cloudflare subdomain*

Conclusion

The adoption of the **Cloud** brought several advantages and challenges related to it. This chapter covered the popular providers and the common services offered by them. Working in a specific **Cloud** brings with it the limitation of having a vendor lock-in. Automation tools and management systems allow for managing multiple clouds and distributing services between them, saving costs and reducing the dependency on only one provider. This chapter covered how **Ansible** can be used to deploy an infrastructure and described how to use **Terraform** for a multi cloud application deployment.

Key facts

- **AWS** is the biggest **Cloud Provider** as well as the pioneer.
- **Microsoft Azure** and **Google Cloud Platform** are growing and obtaining a market share in the last few years.
- **Alibaba Cloud** is one of the biggest providers based in Asia.
- **Ansible** can be used as an *Infrastructure-as-Code* tool, defining step by step the tasks to perform in the target provider.
- **Terraform** is a specific *Infrastructure-as-Code* tool with hundreds of providers to operate with public and private clouds**.**

Questions

1. What is the name for the *Object Storage Service* on **AWS**?
    a. S3
    b. EC2
    c. RDS
2. What is the name for the NoSQL database on **Microsoft Azure**?
    a. Azure MongoDB
    b. Azure Cosmos DB
    c. Azure NoSQL DB
3. What is the name of the service provided by **Google Cloud** for functions-as-a-service?
    a. Cloud Functions
    b. GCP Functions
    c. Google Functions
4. What is the name of the component on **OpenStack** for networking?
    a. nova
    b. cinder
    c. neutron
5. Executing **terraform** to apply a definition is a declarative or imperative approach?
    a. declarative
    b. imperative

Answers

1. a
2. b
3. a
4. c

5. b

CHAPTER 16

Infrastructure as a Service

Introduction

This chapter will detail what **Infrastructure as a Service (IaaS***)* is and its different usages. This type of *Cloud* offers compute, storage, and network resources (among other resources) on demand. Users will access a platform to request the virtual resources needed.

The previous chapter explained the common services offered by the popular *Public Clouds* and introduced **OpenStack** as a *Private Cloud* solution. This chapter will deepen on the options available for the *Private Cloud* and the term *Hybrid Cloud*.

This chapter will describe how to operate an **OpenStack** platform, explaining all the elements required to run *Virtual Machines* and access them.

Structure

In this chapter, we will discuss the following topics:

- Infrastructure as a Service
- Private Cloud
- Hybrid Cloud
- OpenStack as an IaaS
- Running virtual machines on OpenStack

Infrastructure as a Service

The concept of *Infrastructure as a Service* consists of converting the resources on a traditional on-premise data center to an infrastructure based on the *Cloud* model. In this model, the resources are virtualized and offered to the users to be consumed. Some of the important characteristics offered by this service are as follows:

- A self-service API to perform new requests, query, or manipulate resources.

- A Web-based self-service portal to ease operations.
- High availability services to provide the best uptime possible.
- Replication of the data stored to avoid data loss.
- Physical resource optimization to provide the best performance on virtual resources offered.
- Security of the data with secret management and encryption of the data.

The primary components virtualized to be offered to the consumers are as follows:

- **Computing resources**: Run *Virtual Machines* in a controlled environment.
- **Networking resources**: Networks, subnets, switches, firewall, and routers can be used as virtual elements.
- **Storage resources**: Disks, file systems, or object containers can be consumed.

In an *Infrastructure as a Service*, the provider is the one responsible for managing the underlay physical resources, and the user is only responsible for managing the virtual components and the configuration for the virtual resources consumed, such as the operating system for a virtual machine created or the application running on it. *Figure 16.1* features an Infrastructure as a Service diagram:

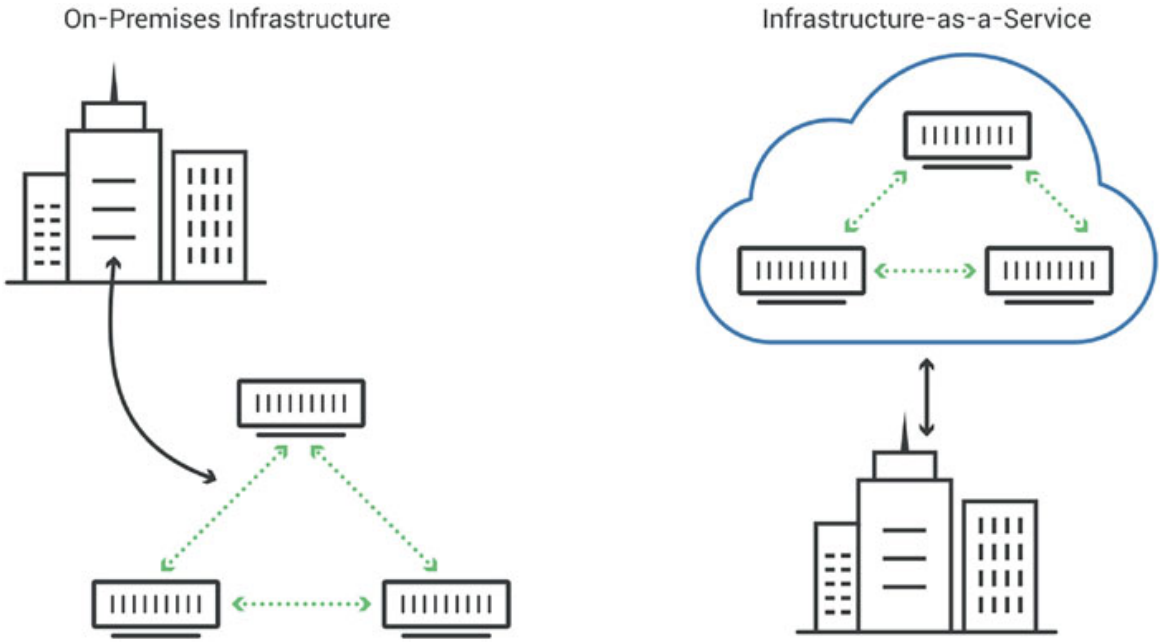


Figure 16.1: *Infrastructure as a Service diagram. Source: Cloudflare*

Private Cloud

In a *Private Cloud*, the resources are only shared via the internet or private network to a specific user or organizations. Companies can decide on a *Private Cloud* for one of the following reasons:

- **Privacy:** Some enterprises require the data and the resources to be under their own responsibility.
- **Cost control:** The cost of a *Public Cloud* can be increased if it is not controlled or estimated properly. Having a *Private Cloud,* the costs are fixed and not related to the usage.
- **Multicloud:** As described in the previous chapter, it is possible to have a *Private Cloud* to perform development and test and, for production, to use a *Public* one.

A *Private Cloud* requires a dedicated team or teams responsible for managing the infrastructure (such as servers, physical network devices, data center, and so on), the platform providing the services, and the support for the services offered. The team or teams are also responsible for scaling up the infrastructure when needed to provide more resources to end users.

The underlay infrastructure in a *Private Cloud* has to ensure the following considerations:

- The system resources, such as memory and CPU, should be sufficient for the requirements of the tenants using the platform.
    - *Overcommitment* is a technique used to provide more resources (memory or CPU) than the available ones. In most of the cases, a *Virtual Machine* is not using all the resources dedicated to them.
- The traffic between the different networks should be isolated for security while avoiding bottleneck. For example, the storage network should not affect the *Virtual Machines* traffic.
- Infrastructure should operate in a *High Availability* mode. This reduces the **single point of failure** (**SPoF**) having redundant devices, such as power supply, Ethernet cards, switch, and so on.

A *Private Cloud* is a complex environment that requires a good architecture design and a maintenance plan to avoid unexpected downtime and degradation of the services offered. Different Open-Source enterprise-ready projects are available for building a *Private Cloud*; some popular examples are as follows:

- **OpenStack**: A *Cloud Computing* solution for providing multiple services to end users.
- **Ceph**: A software-defined storage solution to provide block, file, and object storage.
- **Open Virtual Network (OVN)**: A software-defined network solution on top of *Open vSwitch (*a distributed virtual multilayer switch implementation).

Public Clouds offers the possibility of having a *Private Cloud* on-premise integrated with the *Public Cloud*. The options and descriptions are as follows:

- **Azure Stack**: A portfolio of products that extend *Azure services* and capabilities to customer's environments of choice—from the *datacenter* to *edge locations*. The hardware needs to be certified by *Azure*.
- **AWS Outposts**: It is possible to run *AWS services* locally and connect to a broad range of services available in the local *AWS Region*. The hardware needs to be bought directly from *AWS*.
- **Google Anthos**: It allows building and managing modern applications on *Google Cloud*, existing on-premises environments, or public cloud environments. It enables consistency between on-premises and cloud environments.
- **Alibaba Cloud Apsara Stack**: A full-stack cloud solution designed for enterprise-level customers that require a highly capable and flexible *Hybrid cloud* solution. Hardware needs to be certified by *Alibaba Cloud.*

Hybrid Cloud

The term *Hybrid Cloud* refers to the combination of computing, storage, networking, and services between different environments. It is usually the combination between a *Private Cloud* and a *Public Cloud.* As described in the previous chapter, one of the reasons to work with multiple *Clouds* is for avoiding vendor lock-in as well as reducing costs. Modern applications require an environment to be developed and tested because it is difficult for a developer to use a local system.

Private Cloud in a *Hybrid* environment can be used for that purpose: for developing new applications and testing them. **Continuous Integration** (**CI**) and **Continuous Delivery** (**CD**) can be used for the development process. The release of the applications can be performed on the *Public Cloud* when the tests and the required approvals are performed.

The *Public Cloud Providers* offer the possibility of communicating between both environments through a secure and private connection. *Figure 16.2* features a Hybrid Cloud diagram:

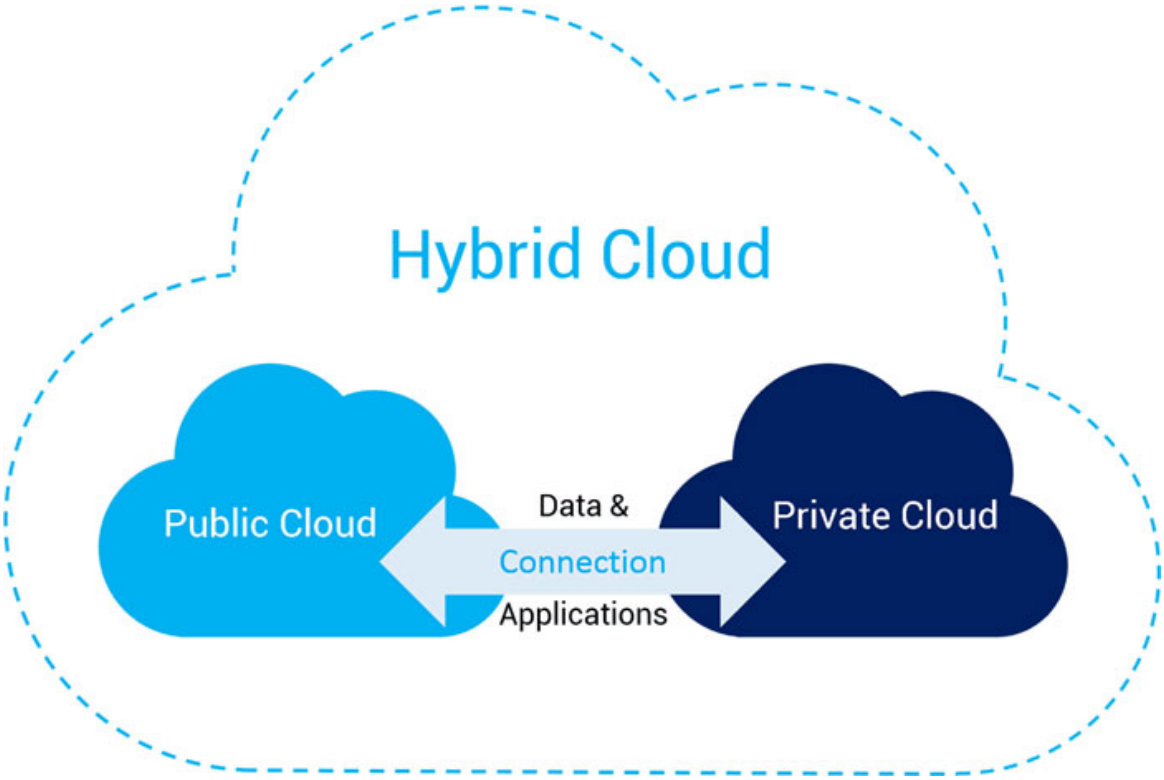


Figure 16.2: *Hybrid Cloud Diagram. Source: Alibaba Cloud*

OpenStack as an IaaS

In the previous chapter, an introduction to **OpenStack** was described, including a list of the core components and other useful ones. This section will cover the architecture of **OpenStack**, its main use cases, and its advantages compared to a *Public Cloud*.

In an **OpenStack** cluster, there are two main types of nodes providing computing services:

- **Controller nodes**: They are servers running the database, queue services, load balancers, and **OpenStack** services accessible via *API*. These nodes include networking and storage services.
- **Compute nodes**: They run *Virtual Machines,* and they are responsible for the correct functioning.

Other types can be part of the cluster depending on the use case and the needs, for example:

- **Block storage nodes**: Nodes responsible for the storage, called *Volumes,* to be used by the *Virtual Machines*.
- **Object storage nodes**: Nodes responsible for the objects; it stores the *images* to be used by *Virtual Machines* and other unstructured data.

Figure 16.3 represents an architecture example for **OpenStack**. It includes the core (controllers and compute) nodes and optional ones (Block and Object storage). Please refer to the following figure:

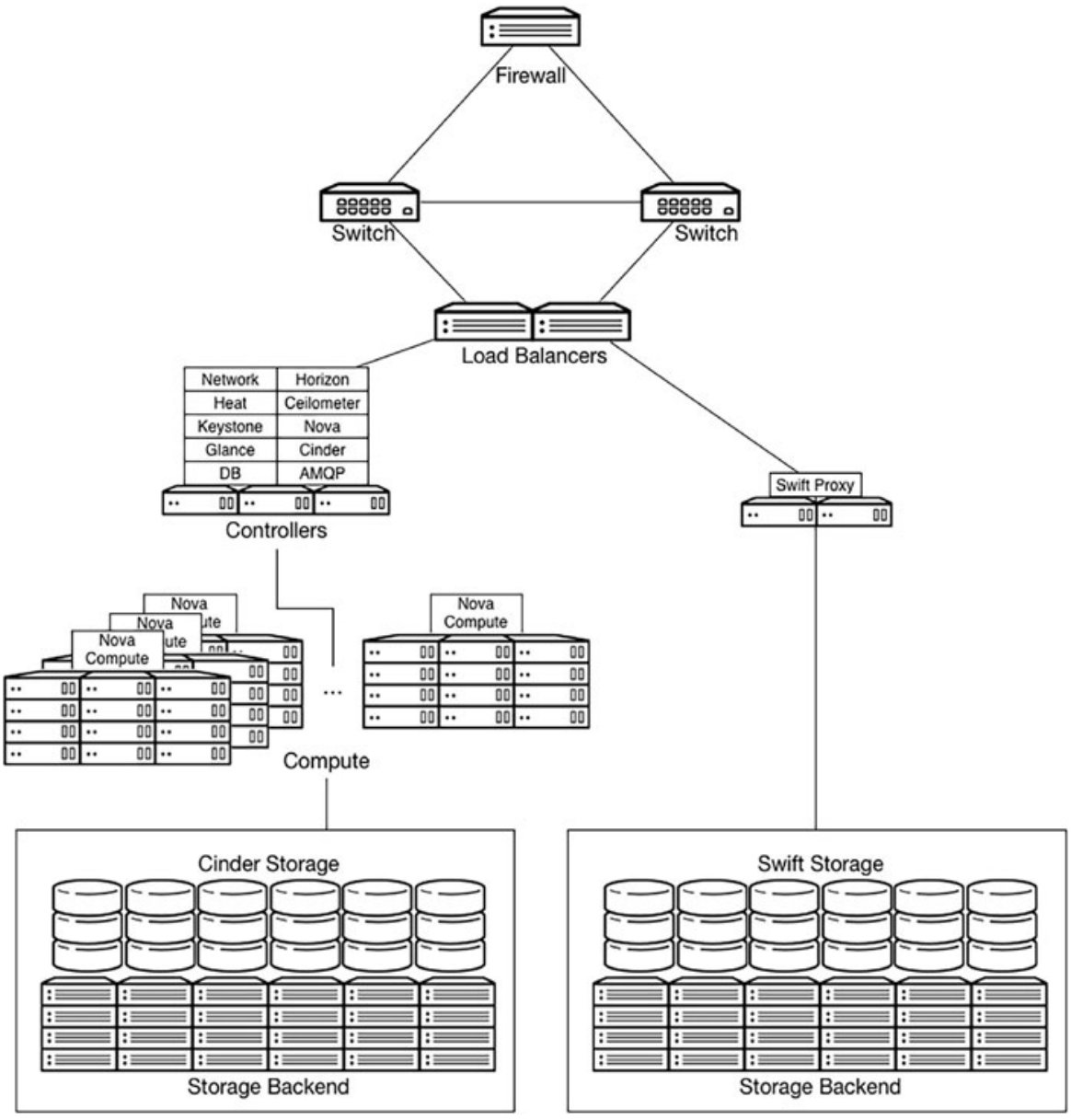


Figure 16.3: *OpenStack architecture*

The *Networking* configuration on **OpenStack** can be simple for development or proof of concept clusters or complex when the cluster is for production purposes. There are the following six main networks required in a production-ready installation when network isolation is used:

- **Control Plane** (also known as *Provisioning*): Used in development or installations without network isolation as the only one network. In a production-ready environment, this network is used for provisioning and management.
- **Internal API**: Communication to the *Services API* between the services. For example, the **nova** service (responsible for computing) communicates with the **neutron** service (responsible for networking) using this network.
- **Tenant**: Used by the *Virtual Machines* in the east-west traffic (communication between *Virtual Machines*) and the north-south traffic (from or to outside the *cluster*).
- **Storage**: Traffic generated from and to the systems providing the storage to **OpenStack**.
- **Storage Management**: Traffic used internally between the systems involved in the *Storage systems*.
- **External**: Used to provide external access from outside the cluster. The *Network* is used to provide an external address to be used for *API*, *Web access*, and access to the instances running on **OpenStack**.

An additional network called **Management** can be configured to access the cluster nodes to be administrated, usually only accessible by the teams who administrate the cluster. *Figure 16.4* shows a network layout for **OpenStack**:

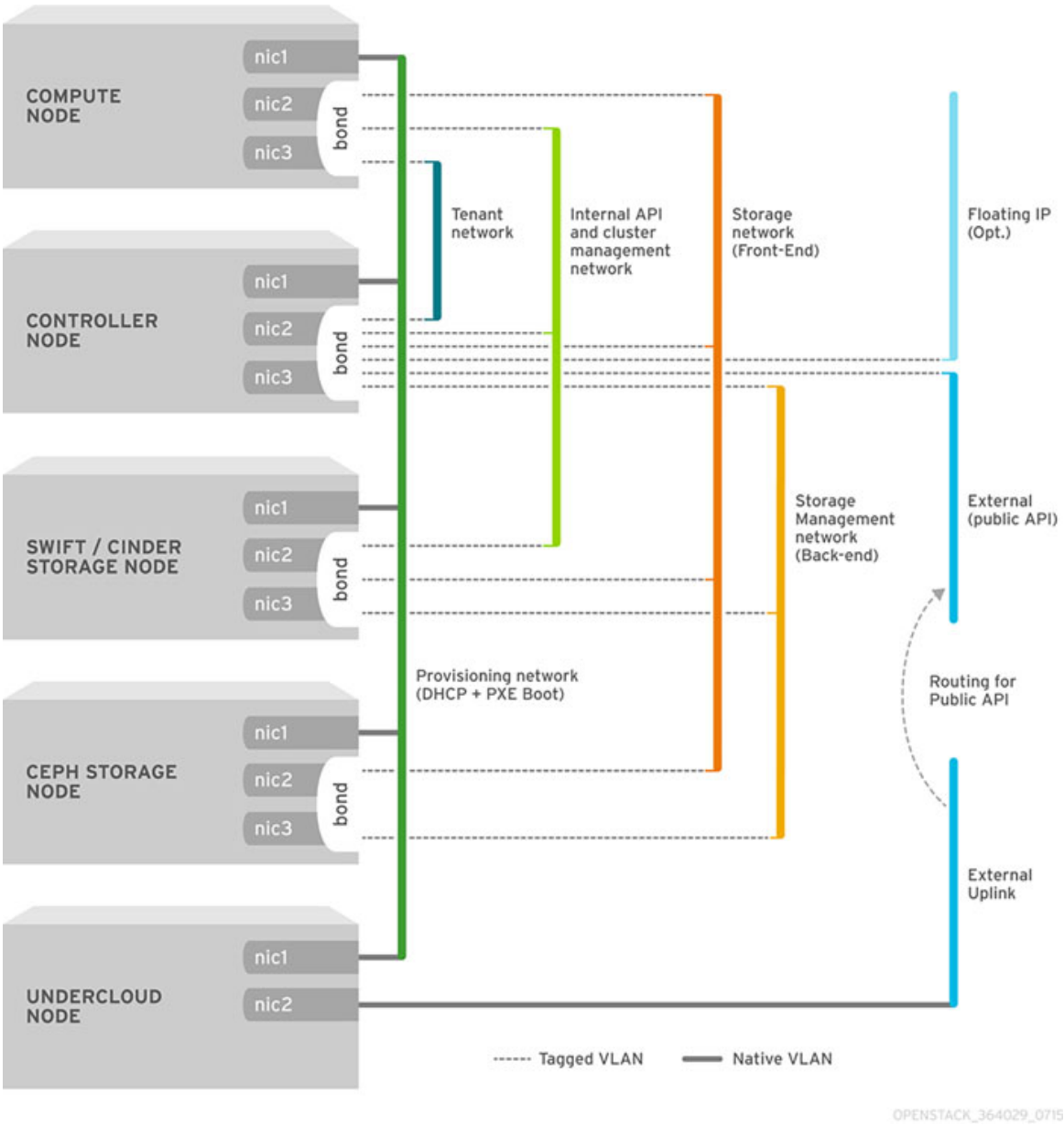


Figure 16.4: *OpenStack Network Layout. Source: Red Hat*

Installation of **OpenStack** is a complex process, but there are available installation tools available, making the process easier. The two popular main options are as follows:

- **TripleO** (*OpenStack On OpenStack*): This project aims at the installation, upgrading, scaling, and operating **OpenStack**. The installation consists of a node called *undercloud,* which is an installation of **OpenStack** in a single node, which further helps the installation of an **OpenStack** cluster called *overcloud*. *Figure 16.5* features a TripleO diagram:

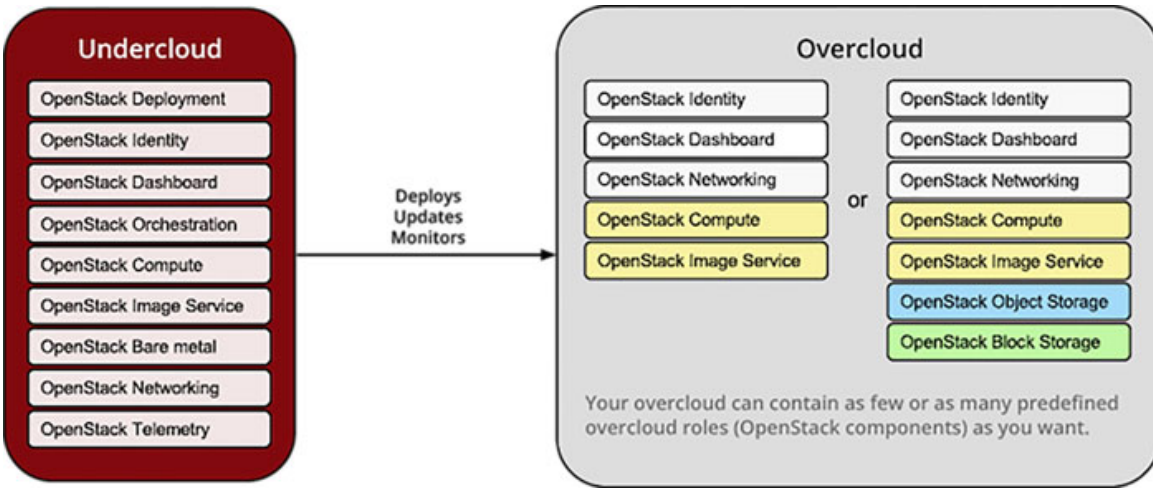


Figure 16.5: *TripleO diagram. Source: OpenStack*

- **OpenStack Charms**: This option uses **Metal-As-A-Service** (**MAAS**) and *Juju* projects. The project MAAS is used for provisioning a bare metal system, and *Juju* to deploy **OpenStack** services on the nodes.

OpenStack uses open-source projects to provide high availability of the services offered. The projects used to provide a reliable service are as follows:

- **Pacemaker**: A popular *High Availability* resource manager for *Linux*. It is responsible for ensuring that some of the services configured are available on the controller nodes. The services can be configured in active-passive or active-active. If it detects that some of the applications or nodes are not responding, it is responsible for promoting a node as active. The resources managed by *Pacemaker* are as follows:
    - **Galera**: A *Multi-Master* cluster for **MySQL/MariaDB** using synchronous replication.
    - **Haproxy**: It is responsible for performing load balancing for the services offered by **OpenStack**.
    - **RabbitMQ**, **Qpid,** or **ZeroMQ**: The message queue service to coordinate operations and status information among services.
    - **Virtual IPs**: IP addresses associated with an application and move with the application when the application is moved from one server to another server. It allows a client to always call the same IP address instead of guessing where the application is running.
- **Memcached**: Used to cache tokens for the *Identity Service*.
- **Redis** or **MongoDB**: These *NoSQL* are used for the *Telemetry Service*.

The *Controller* nodes are usually three to guarantee the correct functionality of the system and provide high availability to the services. The number of *Compute* nodes depends on the resources for each of them and the needs of the end users.

There are the following five main use cases for **OpenStack** these days:

- **Telco Companies**: These companies require a big performance and high bandwidth to offer telecommunication services to customers. **OpenStack** provides different options to increase performance, such as *Real Time Kernel* and *Huge Pages*. For data path networking, it is possible to use *SRIOV* or *DPKD*, among other solutions.
- **Service Providers**: These kinds of companies need the flexibility to create, scale, and delete services. **OpenStack** is a good solution to reduce operational costs.
- **Development**: Having a *Private Cloud* helps developers speed up the process and perform the testing.
- **Artificial Intelligence and Machine Learning (AI/ML)**: These technologies require access to special hardware to perform operations, such as a **Graphical Process Unit** *(***GPU***)*. **OpenStack** is a perfect platform to run *Virtual Machines* to use **virtual GPUs** *(***vGPUS**).
- **Edge Computing**: The current trend is to decentralize the infrastructure to have nodes to perform tasks in a different location. **OpenStack** allows it to be installed and contain *Edge* nodes.

Running virtual machines on OpenStack

To operate with **OpenStack**, it is required to have the address to perform the authentication, a valid username, the password, and the project to be used. The authentication is performed against **Keystone**, which uses port *5000/tcp* or *1300/tcp* (TLS protected) to accept the requests. It is possible to specify the authentication data to the **OpenStack** command line client in the following three different ways:

- Using the file `clouds.yaml`, which can include different configurations for multiple installations. The environment variable OS_*CLOUD* or the option *–os-cloud* is used to specify the authentication to be used. This file can be located in the following directories, with the first having more priority:
    - Inside the current directory.
    - Inside the directory `~/.config/openstack`
    - Inside the directory `/etc/openstack`

- Using environment variables in the terminal to specify the values. Some variable examples are as follows:
  - **`OS_AUTH_URL`**: The *endpoint* URL against the client will perform the authentication.
  - **`OS_PASSWORD`**: The password for the specified username.
  - **`OS_PROJECT_NAME`**: The project name to be used during authentication.
  - **`OS_USERNAME`**: To specify the username to perform the authentication.
- Using the argument options for the client, some of them are as follows:
  - **`--os-auth-url`**: The *endpoint* URL against the client will perform the authentication.
  - **`--os-password`**: The password for the specified username.
  - **`--os-project-name`**: The project name to be used during authentication.
  - **`--os-username`**: To specify the username to be performed for the authentication.

The following block shows the example content of file **`clouds.yaml`** with two clusters configured:

```
clouds:
  production:
    auth:
      auth_url: "https://production.osp.example.com:13000"
      username: "regularuser"
      project_name: "webproject"
      user_domain_name: "Default"
      password: "complexpassword
    region_name: "regionOne"
    interface: "public"
    identity_api_version: 3:
  developemnt:
    auth:
      auth_url: "http://development.osp.example.com:5000"
      username: "admin"
      project_name: "devapps"
      user_domain_name: "Default"
      password: "supersecretpassword
    region_name: "regionOne"
```

```
interface: "public"
identity_api_version: 3
```

The client **OpenStack** has multiple arguments for the different services available in the cluster where the tasks are going to be performed. To list the available services, it is possible to use the arguments **service list**. *Figure 16.6* shows an example:

```
[stack@undercloud ~]$ openstack --os-cloud development service list
+----------------------------------+-----------+----------------+
| ID                               | Name      | Type           |
+----------------------------------+-----------+----------------+
| 00360677d68c424480d7cd8a5713ac43 | cinderv2  | volumev2       |
| 017f36b8b36f4cb8a64d4cb4d4099f7a | swift     | object-store   |
| 13c5a514ed134bb88d86fef0dbf11151 | octavia   | load-balancer  |
| 21a6de95593f4b00b807d1cb37d92516 | keystone  | identity       |
| 311ef0d661fc436c88d1fe5fd3e41633 | placement | placement      |
| 885d2f1720dd4adc85a4d964e665b2b5 | neutron   | network        |
| 9267e15a501f43c0895385830626082d | heat      | orchestration  |
| a95e9fea9beb40ebbc3848992ecd080a | nova      | compute        |
| afed1bfc0d8549e2bb8dd4ce8c709b0d | glance    | image          |
| d5cab5fc41ab4aad91f797257b8d1e2c | manilav2  | sharev2        |
| d97093c4a4094115bc2b8e5b74b64722 | cinderv3  | volumev3       |
| dc05703398584e6bb7a1bf2fe9978d61 | manila    | share          |
| ec46c86136bb4f2e8600183ec029affc | heat-cfn  | cloudformation |
| f79308b62f9547e9b0d306d24b6097c5 | barbican  | key-manager    |
+----------------------------------+-----------+----------------+
```

Figure 16.6: *Output example listing services available*

With the argument **`catalog`**, it is possible to obtain the services and *endpoint URL* using a **catalog list** and obtain information about a specific one using a **catalog show**. *Figure 16.7* shows an example of the service *nova*.

```
[stack@undercloud ~]$ openstack --os-cloud development catalog show nova
+-----------+-----------------------------------------+
| Field     | Value                                   |
+-----------+-----------------------------------------+
| endpoints | regionOne                               |
|           |   public: http://10.0.0.150:8774/v2.1   |
|           | regionOne                               |
|           |   internal: http://172.17.0.150:8774/v2.1 |
|           | regionOne                               |
|           |   admin: http://172.17.0.150:8774/v2.1  |
|           |                                         |
| id        | a95e9fea9beb40ebbc3848992ecd080a        |
| name      | nova                                    |
| type      | compute                                 |
+-----------+-----------------------------------------+
```

Figure 16.7: *Output example describing a service*

The clients access the cluster using the *public* endpoint, the services use the *internal* endpoint (located in the *internal API* network*),* and some administrative tasks are performed using the *admin* endpoint.

Images

To run a *Virtual Machine*, it is required to use an *image* containing the operating system desired. The project to upload, manipulate or delete *images* is called **Glance**. The popular disk formats are as follows:

- ***QCOW2* (*QEMU copy-on-write version 2*)**: is the main format used with the *KVM* hypervisor, the one used on **OpenStack**. The size is compressed, can be dynamically resized, and it is possible to run several *Virtual Machines* from a base image as well as only store the differences.
- ***RAW***: This is an unstructured disk image; it contains the full size of the disk. This format is needed when *Ceph* is used as the backend for the storage.
- ***VMDK* (*Virtual Machine Disk*)**: This format is supported by many hypervisors, including *Vmware ESXi*.
- ***VDI* (*Virtual Disk Image*)**: Format supported by *VirtualBox* and *QEMU*.

Creating an *Image* requires specifying the disk format and the container format. When the image has no metadata, the container format is *bare*. *Figure 16.8* shows the creation of an image for *Debian 10*.

```
[stack@undercloud ~]$ export OS_CLOUD=development
[stack@undercloud ~]$ openstack image create --container-format bare \
>         --disk-format qcow2 --file debian-10-openstack-amd64.qcow2 debian10
```

Figure 16.8: *Output example creating an image*

The output for the command includes useful information, including the *id* of the image created, the *size*, and the *status,* among other properties. *Table 16.1* shows (truncated) an example of the output:

Field	Value
checksum	b2e5db6e58c16546a4b561eeaf2d5274
container_format	bare
created_at	2022-12-04T19:04:57Z
disk_format	qcow2
file	/v2/images/01805448-4401-47a3-930b(..)

id	01805448-4401-47a3-930b-1d38f2324514
min_disk	0
min_ram	0
name	debian10
owner	7690147846bb4f24b724a940673d6ce8
properties	direct_url='swift+config://01805448(..)
protected	False
schema	/v2/schemas/image
size	665426944
status	active
tags	
updated_at	2022-12-04T19:05:03Z
virtual_size	None
visibility	shared

Table 16.1: *Output is shown after creating an image. caption*

Table 16.2 shows the typical tasks performed related to *images*.

Arguments	Description
image list	List the existing *images* defined.
image delete	Delete an existing *image*.
image show	Shows all the information about the specified *image*.
image set	Updates parameters related to the specified *image*.

Table 16.2: *Typical tasks related to flavors*

Flavors

To run a *Virtual Machine*, it is required to specify the resources required. For that, a *flavor* is specified during the creation. A *flavor* contains how many *virtual CPUs*, memory, and disk the *Virtual Machine* will have associated. *Figure 16.9* shows the creation of a flavor named `m1.small`:

```
[stack@client ~]$ openstack flavor create m1.small --ram 2048 --disk 20 --vcpus 1

+----------------------------+-------------------------------------+
| Field                      | Value                               |
+----------------------------+-------------------------------------+
| OS-FLV-DISABLED:disabled   | False                               |
| OS-FLV-EXT-DATA:ephemeral  | 0                                   |
| disk                       | 20                                  |
| id                         | 639b81ea-07ee-4fd3-a934-cde57870ff58 |
| name                       | m1.small                            |
| os-flavor-access:is_public | True                                |
| properties                 |                                     |
| ram                        | 2048                                |
| rxtx_factor                | 1.0                                 |
| swap                       |                                     |
| vcpus                      | 1                                   |
+----------------------------+-------------------------------------+
```

Figure 16.9: *Output example creating a flavor*

The data is located on the compute service named **nova**. It is possible to resize an existing *Virtual Machine* using the arguments **server resize**. *Table 16.3* shows the typical tasks performed related to *flavors*.

Arguments	Description
flavor list	List the existing *flavors* defined, including the resource values.
flavor delete	Delete an existing *flavor*.
flavor show	Shows all the information about the specified *flavor*.
flavor set	It is not possible to change the resources defined, only other projects such as the description.

Table 16.3: *Typical tasks related to flavors*

Networking architecture

The *Networking* part of **OpenStack** is a complex element. The service offering the networking functionality is called **neutron**. This is due to the need for the isolation of the projects and the networks for the tenants. There are the following two types of networks:

- **Overlay Networks**: These virtual networks are internal to **OpenStack**, and the traffic is encapsulated through the *tenant* network. The two most common encapsulation types are as follows:
    - **Virtual Extensible LAN (VXLAN):** An encapsulation protocol that provides data center connectivity using tunneling, stretching Layer 2 connections over an underlying Layer 3 network.

    - **Generic Network Virtualization Encapsulation** (**GENEVE**): The encapsulation protocol used on OVN. It promises to address the perceived limitations of the earlier specifications and support all of the capabilities of VXLAN.

  To access to an *Overlay Network*, it is required to perform a **Network Address Translation** (**NAT**) from a physical network (the *external* or *public* network) to this network. On **OpenStack**, the *IP addresses* used for this purpose are called **Floating IPs**.
- **Provider Networks**: These are physical networks attached to the nodes in the cluster. *Virtual Machines* can be attached directly to these networks, similar to other *Virtualization Platforms*, allowing access directly to them.

Figure 16.10 shows the complexity, and in this case, using *VXLAN* tunnels, of the *Network* architecture on **OpenStack**.

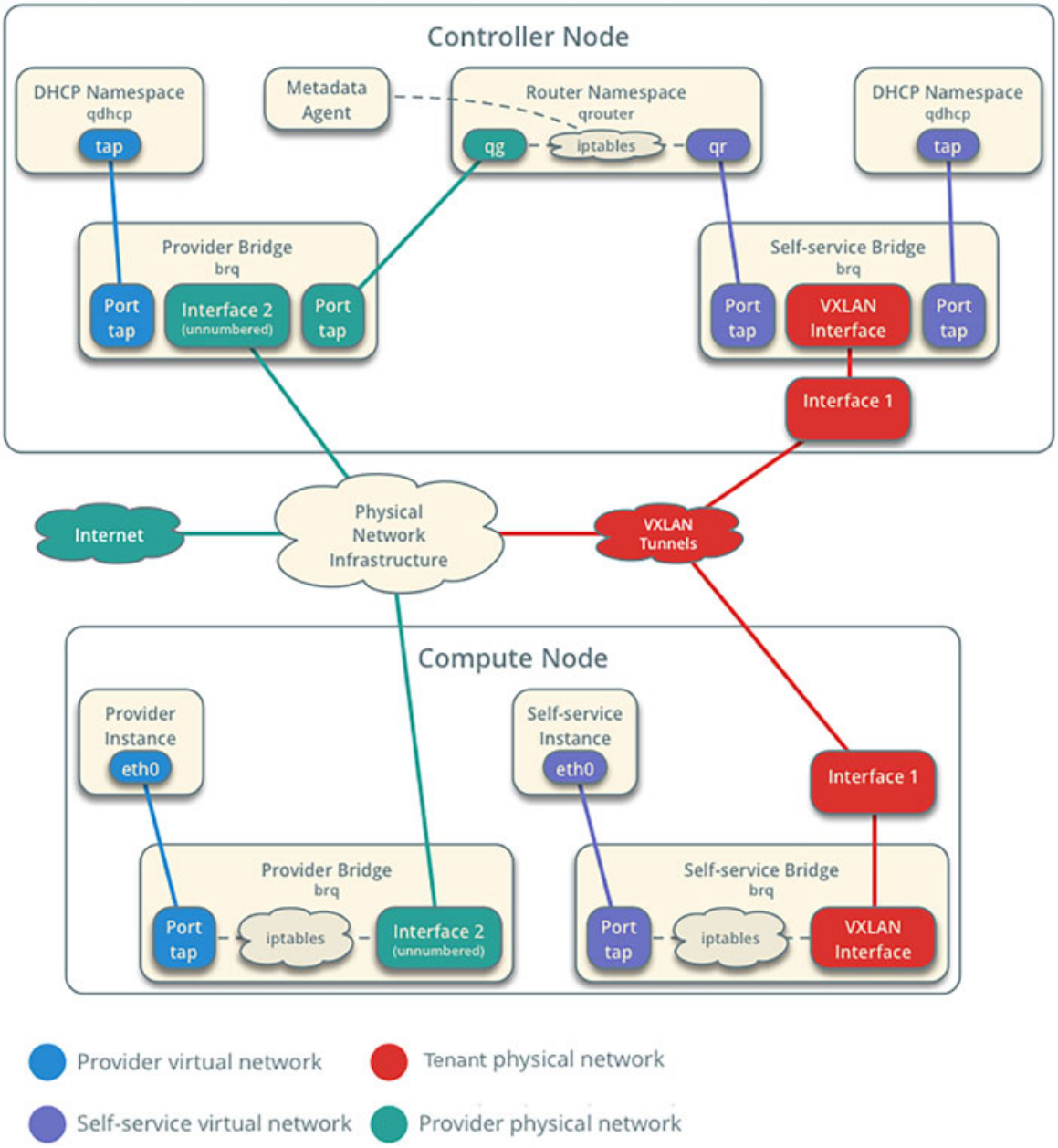


Figure 16.10: *Network architecture on OpenStack using VXLAN. Source: OpenStack*

The adoption of OVN simplifies the infrastructure but requires more knowledge about how a **Software Defined Network** (**SDN**) works to separate the logic part from the data path. *Figure 16.11* shows the components involved:

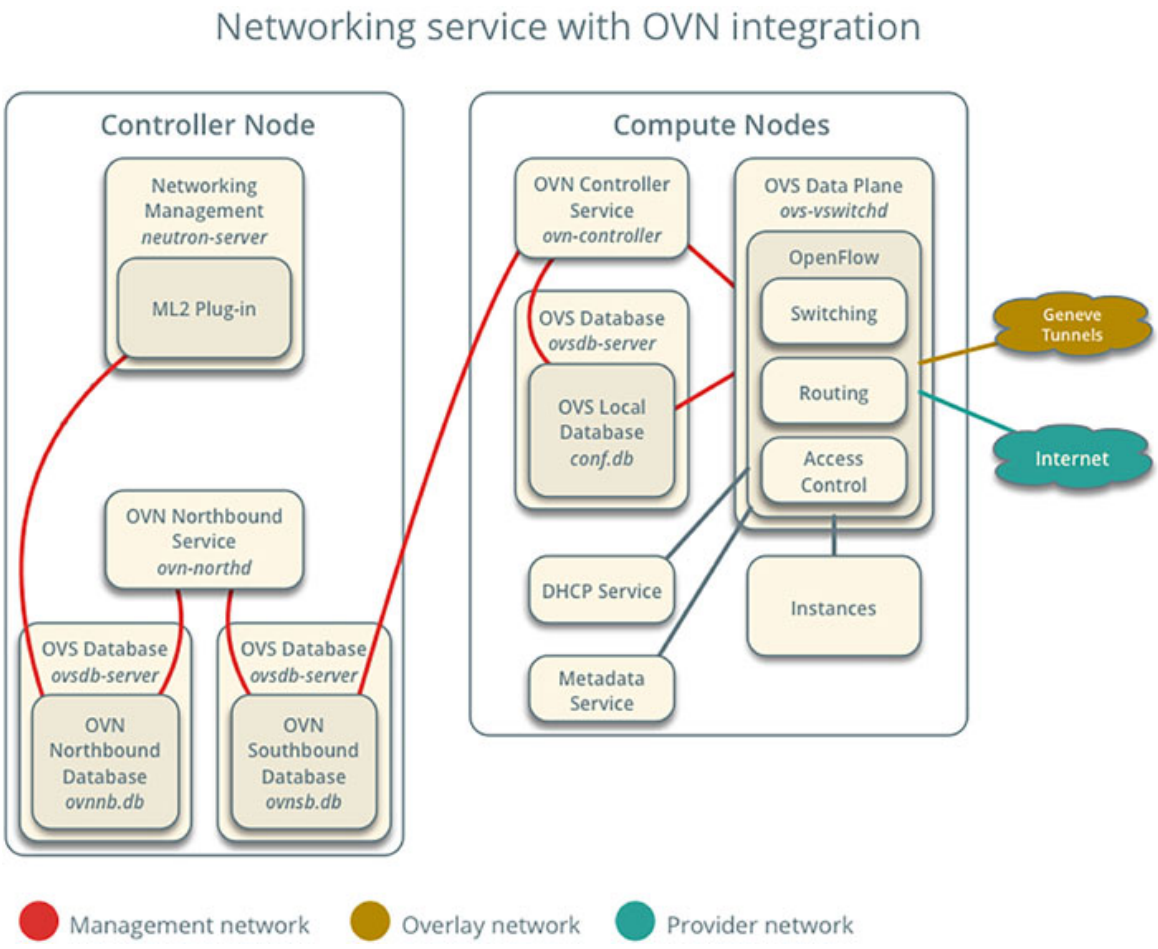


Figure 16.11: *Network architecture on OpenStack using VXLAN. Source: OpenStack*

Networking for Virtual Machines

To run a *Virtual Machine* on OpenStack, it should be connected to a *network*. This *network* should have at least one *subnet* associated. The *subnet* can be configured to provide DHCP addresses to the *Virtual Machines* connected to it. The argument **`network create`** is used to create the network, which can be an overlay or a provider network. The argument **`subnet create`** is used to create *subnets* inside a *network*. *Figure 16.12* creates a *provider network* to be used as an external network for **Floating IPs.**

```
[stack@client ~]$ openstack network create external  --share --external \
>    --provider-physical-network datacentre \
>    --provider-network-type flat
```

Figure 16.12: *Example command creating an external network*

The output of the following command includes the *id*, the *status*, and information relational to the network created. Refer to *table 16.4*:

Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2022-12-04T19:44:24Z
description	
dns_domain	
id	8229579d-1427-4b74-bb89-1d7a16cfac5f
location	cloud='development', project.domain_id=, project.domain_name='Default', project.id='7690147846bb4f24b724a940673d6ce8', project.name='admin', region_name='regionOne', zone=
mtu	1500
name	external
port_security_enabled	True
project_id	7690147846bb4f24b724a940673d6ce8
provider:network_type	flat
provider:physical_network	datacentre
provider:segmentation_id	None
qos_policy_id	None
revision_number	1
router:external	External
segments	None
shared	True
status	ACTIVE
subnets	
tags	
updated_at	2022-12-04T19:44:24Z

Table 16.4: *Output example creating a network.*

The physical network is connected to a subnet with the address *10.0.0.0/24*. *Figure 16.13* shows an example of the creation of the subnet attached to the previous network created.

```
[stack@client ~]$ openstack subnet create subexternal --no-dhcp \
>   --network external --subnet-range 10.0.0.0/24 \
>   --allocation-pool start=10.0.0.220,end=10.0.0.230 \
>   --gateway 10.0.0.1 --dns-nameserver 8.8.8.8
```

Figure 16.13: *Example command creating a subnet*

A subnet by default is created by having *DHCP* enabled; the option **`--no-dhcp`** is used to disable it. The output of the previous command is shown in *table 16.5*:

Field	Value
allocation_pools	10.0.0.220-10.0.0.230
cidr	10.0.0.0/24
created_at	2022-12-04T20:10:50Z
description	
dns_nameservers	8.8.8.8
enable_dhcp	False
gateway_ip	10.0.0.1
host_routes	
id	229b4e9e-35aa-47db-8f30-8ad1dcd50608
ip_version	4
location	cloud='development', project.domain_id=, project.domain_name='Default', project.id='7690147846bb4f24b724a940673d6ce8', project.name='admin', region_name='regionOne', zone=
name	subexternal
network_id	8229579d-1427-4b74-bb89-1d7a16cfac5f
prefix_length	None
project_id	7690147846bb4f24b724a940673d6ce8
tags	
updated_at	2022-12-04T20:10:50Z

Table 16.5: *Output example creating a subnet*

Figure 16.14 shows the creation of an overlay network to be used by the *Virtual Machines*.

```
[stack@client ~]$ openstack network create overlay_network
[stack@client ~]$ openstack subnet create sub_overlay_network \
>  --network overlay_network --subnet-range 172.16.0.0/24
```

Figure 16.14: *Example command creating a network and a subnet*

From the network creation, it is important to value that when an *overlay* network is created, the *MTU* is less than 1500 because some bytes are reserved for the encapsulation, and the encapsulation type in the previous example is *GENEVE*. Refer to *table 16.6*:

Field	Value
mtu	1442
provider:network_type	geneve

Table 16.6: *Fields for the subnet related to the overlay network.*

Table 16.7 shows the typical tasks performed related to *networks* and *subnets*.

Arguments	Description
network list	List the existing *networks* created, including the *subnets* configured for them.
network delete	Delete a not-in-use *network*.
network show	Shows all the information about the specified *network*.
network set	Sets different parameters of the *subnet*, such as the name or description.
subnet list	List the existing *subnets* created.
subnet delete	Delete a not-in-use *subnet*.
subnet show	Shows all the information about the specified *subnet*.
subnet set	Sets different parameters of the *subnet*

Table 16.7: *Typical tasks related to networks and subnets*

Routing

After the networks are created, a *Virtual Machine* can be connected to the *overlay network,* but it will be isolated from outside of the cluster. To connect the external network with the private network, it is required to create a *Virtual Router*. The argument **router create** is used to create a *virtual router*. Refer to *figure 16.15*:

```
[stack@client ~]$ openstack router create router_private -c id -c status
+--------+--------------------------------------+
| Field  | Value                                |
+--------+--------------------------------------+
| id     | 51ac2c9c-54d6-4eab-9572-c1dcfd3e1d84 |
| status | ACTIVE                               |
+--------+--------------------------------------+
```

Figure 16.15: *Example command creating a virtual router*

To specify the external network connected to the *virtual router*, the argument **set** with the option *–external gateway.* Using the argument **`router add subnet`**, it is possible to add a subnet to a router. *Figure 16.16* shows the actions using the previous router created:

```
[stack@client ~]$ openstack router set --external-gateway external router_private
[stack@client ~]$ openstack router add subnet router_private sub_overlay_network
```

Figure 16.16: *Example command configuring a router*

Table 16.8 shows the typical tasks performed related to *routers*

Arguments	Description
router remove subnet	Remove an existing *subnet* from the *router*.
router list	List the existing *routers* created.
router delete	Delete an existing *router* if it is not in use.
router show	Shows all the information about the specified *router*.
router set	Sets different parameters of the *router*, such as the name or description.
router unset	Unset some parameters configured, such as the external gateway.

Table 16.8: *Typical tasks related to routers*

Security Groups

A *Security Group* is a set of firewall rules applied to *Virtual Machines*. By default, **OpenStack** only allows egress traffic. When a project is created, a *security group* is created and associated as a default. Administrators and users can add rules to the *security group* or create a new one and add the rules to them. *Figure 16.17* shows an example to allow connecting to port 22 (*SSH*):

```
[stack@client ~]$ openstack security group create allowssh -c id
+-------+--------------------------------------+
| Field | Value                                |
+-------+--------------------------------------+
| id    | 7af16f2c-b257-4c78-8d56-cd7397843976 |
+-------+--------------------------------------+
[stack@client ~]$ openstack security group rule create  --ingress \
>   --ethertype IPv4  --protocol tcp  --dst-port 22 allowssh -c id
+-------+--------------------------------------+
| Field | Value                                |
+-------+--------------------------------------+
| id    | 97a95b9d-1946-4c21-9d96-96108d521994 |
+-------+--------------------------------------+
```

Figure 16.17: *Example of creating a security group and allowing SSH*

Table 16.9 shows the typical tasks performed related to *security groups*.

Arguments	Description
security group list	List the existing *security groups* created.
security group delete	Deletes an existing *security group.*
security group show	Shows all the information about the specified *security group*, including the rules assigned.
security group set	Sets different parameters of the *security group*, such as the name or description.
security group rule delete	Deletes an existing *security group rule* from a *security group*.

Table 16.9: *Typical tasks related to the security group*

Public key

To access the *Virtual Machines,* it is required to use the *SSH key* authentication method. Cloud images available for the *Linux* distributions do not have a default password. **OpenStack** allows to upload a *Public Key* to be injected into the *Virtual Machine* to allow remote access. Refer to *figure 16.18*:

```
[stack@client ~]$ openstack keypair create --public-key ~/.ssh/id_rsa.pub mykey
+-------------+-------------------------------------------------+
| Field       | Value                                           |
+-------------+-------------------------------------------------+
| fingerprint | f1:1a:36:55:5c:7f:38:5f:ef:2b:5d:85:61:1b:d8:e6 |
| name        | mykey                                           |
| user_id     | 6c1f154be7c14eb3b137e73910fad113                |
+-------------+-------------------------------------------------+
```

Figure 16.18: *Example of creating a security group and allowing SSH*

Virtual Machines

By having a network ready to be used, a *flavor* defined, and an *image* with the desired operating system, it is possible to run a *Virtual Machine*. The computing service is called **nova**. The security group can be the *default* one in the project where it is going to be created or specify a custom one created in the cluster. Optionally, it is possible to specify the public key to be used. The arguments **server create** are used for the creation. *Figure 16.19* shows an example of the creation:

```
[stack@client ~]$ openstack server create myfirstvm  --network overlay_network \
>   --flavor m1.small --image debian10 \
>   --key-name mykey --security-group allowssh --wait
```

Figure 16.19: *Example of creating a new Virtual Machine*

Using the option **`--wait`**, the command will create the *Virtual Machine* or show the reason why it was not possible to be created. The output includes the *IP address* assigned and other useful information. *Table 16.10* shows the output example for the previous command:

Field	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-SRV-ATTR:instance_name	instance-00000004
OS-EXT-STS:power_state	Running
OS-EXT-STS:task_state	None
OS-EXT-STS:vm_state	active
OS-SRV-USG:launched_at	2022-12-05T18:49:38.000000
addresses	overlay_network=172.16.0.64
adminPass	f8P3jTtgBffZ
config_drive	
created	2022-12-05T18:49:21Z
flavor	m1.small
hostId	ce58aff5b3d82cc6e61b(..)
id	ab12fad7-5dca-(..)
image	debian10 (..)
key_name	mykey
name	myfirstvm

security_groups	name=’allowssh’
status	ACTIVE
updated	2022-12-05T18:49:38Z
user_id	6c1f154be7c14eb3b137e73910fad113
volumes_attached	

Table 16.10: *Output fields example creating a server*

It is possible to run the *Virtual Machines* running with the arguments **server list** as observed in *figure 16.20*:

```
[stack@client ~]$ openstack server list -c Name -c Status -c Networks -c Image
+-----------+--------+-----------------------------+----------+
| Name      | Status | Networks                    | Image    |
+-----------+--------+-----------------------------+----------+
| myfirstvm | ACTIVE | overlay_network=172.16.0.64 | debian10 |
+-----------+--------+-----------------------------+----------+
```

Figure 16.20: *Example listing Virtual Machines*

Table 16.11 shows the typical tasks performed related to *security groups*.

Arguments	Description
server delete	Deletes an existing *Virtual Machine.*
server show	Shows all the information about the specified *Virtual Machine*.
server set	Sets different parameters of the *Virtual Machine*, such as the name or description.
server console log show	Shows the log related to the *Virtual Machine* during the *boot* process.
server stop	Stops the *Virtual Machine*.
server reboot	Reboots the *Virtual Machine*.

Table 16.11: *Typical tasks related to Virtual Machines*

Floating IPs

The *Virtual Machines* attached to an *Overlay Network* are not accessible from outside of the cluster. To have connectivity to the external world, a *Floating IP* in the external network should be created and attached to the *Virtual Machine.* The arguments **`floating ip create`** is used to create a new *Floating IP* to be used later on a *Virtual Machine*. The argument **`server add floating ip`** is used to attach the *IP* to the *Virtual Machine*. *Figure 16.21* shows the process:

```
[stack@client ~]$ openstack floating ip create external -c floating_ip_address
+---------------------+------------+
| Field               | Value      |
+---------------------+------------+
| floating_ip_address | 10.0.0.225 |
+---------------------+------------+
[stack@client ~]$ openstack server add floating ip myfirstvm 10.0.0.225
[stack@client ~]$ openstack server list -c Name -c Status -c Networks
+-----------+--------+---------------------------------------+
| Name      | Status | Networks                              |
+-----------+--------+---------------------------------------+
| myfirstvm | ACTIVE | overlay_network=172.16.0.64, 10.0.0.225 |
+-----------+--------+---------------------------------------+
```

Figure 16.21: *Floating IP creating and assignment*

The *Virtual Machine* is now accessible using the *IP address* assigned from the external network. *Figure 16.22* shows the connection using the default username *debian,* and the proper *SSH* key.

```
[stack@client ~]$ ssh -i ~/.ssh/id_rsa debian@10.0.0.225
The authenticity of host '10.0.0.225 (10.0.0.225)' can't be established.
ECDSA key fingerprint is SHA256:EV5U2j3bR/G0Vigqm/8ws9SM5BnOu03bZ052vjoTB/8.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.0.225' (ECDSA) to the list of known hosts.
Linux myfirstvm 4.19.0-22-cloud-amd64 #1 SMP Debian 4.19.260-1 (2022-09-29) x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
debian@myfirstvm:~$
```

Figure 16.22: *Connecting using a Floating IP*

Inside of the *Virtual Machine*, the only IP available is the one assigned in the *Overlay Network* because the *Floating IP* is a *NAT address* managed by **OpenStack**. *Figure 16.23* shows the output of the command **`ip`**.

```
debian@myfirstvm:~$ ip a s dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1442 qdisc pfifo_fast state UP grou
p default qlen 1000
    link/ether fa:16:3e:7f:8e:18 brd ff:ff:ff:ff:ff:ff
    inet 172.16.0.64/24 brd 172.16.0.255 scope global dynamic eth0
       valid_lft 41997sec preferred_lft 41997sec
    inet6 fe80::f816:3eff:fe7f:8e18/64 scope link
       valid_lft forever preferred_lft forever
```

Figure 16.23: *IP available inside the Virtual Machine*

Table 16.12 shows the typical tasks performed related to *Floating IPs.*

Arguments	Description

floating ip list	Lists the existing *Floating IPs*.
floating ip delete	Deletes an existing F*loating IP.*
floating ip show	Shows all the information about the specified *Floating IP*.
floating ip set	Sets different parameters of the *Floating IP*, such as the port associated.
server remove floating ip	Dissociate a *Floating IP* from a *Virtual Machine*.

Table 16.12: *Typical tasks related to Floating IPs*

Persistent storage

By default, the *Virtual Machines* have ephemeral storage for the root disk. This is a common behavior working in a **Cloud**, and the persistent data are usually in a separate disk. The project **`cinder`** allows for creating volumes and attaching them to the *Virtual Machine*. The argument **`volume create`** is used to create a *volume* to be used later with the argument **`server volume add`**. *Figure 16.24* shows an example:

```
[stack@client ~]$ openstack volume create --size 10 firstvolume
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| consistencygroup_id | None                                 |
| created_at          | 2022-12-05T19:20:33.000000           |
| description         | None                                 |
| encrypted           | False                                |
| id                  | 1d0a2bf9-c10d-4b17-8c85-fff9f9da5cee |
| migration_status    | None                                 |
| multiattach         | False                                |
| name                | firstvolume                          |
| properties          |                                      |
| replication_status  | None                                 |
| size                | 10                                   |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | tripleo                              |
| updated_at          | None                                 |
| user_id             | 6c1f154be7c14eb3b137e73910fad113     |
+---------------------+--------------------------------------+
[stack@client ~]$ openstack server add volume myfirstvm firstvolume
```

Figure 16.24: *Create and attach a volume to a Virtual Machine*

Inside the *Virtual Machine*, without the need to be restarted, the disk will be available for usage. *Figure 16.25* shows the output of the command **`lsblk`** inside of the instance showing the new disk (**`vdb`**):

```
debian@myfirstvm:~$ lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda    254:0    0  20G  0 disk
└─vda1 254:1    0  20G  0 part /
vdb    254:16   0  10G  0 disk
```

Figure 16.25: *Output example for the command lsblk inside the instance*

Table 16.13 shows the typical tasks performed related to *Volumes*.

Arguments	Description
volume list	Lists the existing *Volumes*.
volume delete	Deletes an existing *Volume* which is not in use.
volume show	Shows all the information about the specified *Volume*.
volume set	Sets different parameters of the *Volume*, such as the name and description.
server remove volume	De-attach a *Volume* from a *Virtual Machine*.

Table 16.13: *Typical tasks related to Volumes*

Orchestration

As described in the previous chapter, related to the *Orchestration* in the **Public Clouds**, it is possible to write a template on **OpenStack**, to perform the creation of the resources. The orchestration server is called **Heat**. **The** following template creates a new *Virtual Machine* and assigns a *Floating IP* and a *Volume*:

```
heat_template_version: rocky
description: Provision a instance with a floating ip
parameters:
  KeyName:
    description: Name of an existing SSH keypair
    type: string
  InstanceName:
    description: Name of the instance
    type: string
```

```
  FlavorSize:
    description: The flavor required for the instance
    type: string
    default: "m1.small"
  ImageName:
    description: The name of an image to deploy
    type: string
    default: "debian10"
  PrivateNet:
    type: string
    description: Private Network
  ExternalNet:
    type: string
    description: External Network
  SecurityGroup:
    type: string
    description: Security group
    default: "default"
  VolumeSize:
    type: number
    description: Size of the volume to be created.
    default: 1
resources:
  instance:
    type: OS::Nova::Server
    properties:
      flavor: { get_param: FlavorSize }
      image: { get_param: ImageName }
      key_name: { get_param: KeyName }
      name: { get_param: InstanceName }
      networks:
    - port: { get_resource: instance_port }
  instance_port:
    type: OS::Neutron::Port
    properties:
      network: { get_param: PrivateNet }
      security_groups:
    - { get_param: SecurityGroup }
  instance_external:
    type: OS::Neutron::FloatingIP
```

```
    properties:
      floating_network: { get_param: ExternalNet }
      port_id: { get_resource: instance_port }
  volume:
    type: OS::Cinder::Volume
    properties:
      size: { get_param: VolumeSize }
      name: datavolume
      description: Cinder Volume Test
  instance_volume:
      type: OS::Cinder::VolumeAttachment
      properties:
    instance_uuid: { get_resource: instance }
    volume_id: { get_resource: volume }
    mountpoint: /dev/vdb
outputs:
  instance_external_ip:
    description: Floating IP address of instance in External network
    value: { get_attr: [ instance_external, floating_ip_address ] }
```

The arguments **stack create** are used to process the template and create the resources. The option **`--parameter (-p)`** is used to pass the parameter in the format **`key=value`**. The following *Figure 16.26* shows an example of creating a *Virtual Machine* using the previous template:

```
[stack@client ~]$ openstack stack create firststack -t example.yaml \
> --parameter InstanceName=newvm --parameter KeyName=mykey \
> --parameter ExternalNet=external --parameter PrivateNet=overlay_network \
> --parameter FlavorSize=m1.small --parameter SecurityGroup=allowssh \
> --parameter ImageName=debian10 --parameter VolumeSize=5 --wait
2022-12-05 20:34:24Z [firststack]: CREATE_IN_PROGRESS  Stack CREATE started
2022-12-05 20:34:24Z [firststack.instance_port]: CREATE_IN_PROGRESS  state changed
2022-12-05 20:34:25Z [firststack.volume]: CREATE_IN_PROGRESS  state changed
2022-12-05 20:34:25Z [firststack.instance_port]: CREATE_COMPLETE  state changed
2022-12-05 20:34:26Z [firststack.instance]: CREATE_IN_PROGRESS  state changed
2022-12-05 20:34:26Z [firststack.instance_external]: CREATE_IN_PROGRESS  state changed
2022-12-05 20:34:27Z [firststack.volume]: CREATE_COMPLETE  state changed
2022-12-05 20:34:29Z [firststack.instance_external]: CREATE_COMPLETE  state changed
2022-12-05 20:34:38Z [firststack.instance]: CREATE_COMPLETE  state changed
2022-12-05 20:34:39Z [firststack.instance_volume]: CREATE_IN_PROGRESS  state changed
2022-12-05 20:34:43Z [firststack.instance_volume]: CREATE_COMPLETE  state changed
2022-12-05 20:34:43Z [firststack]: CREATE_COMPLETE  Stack CREATE completed successfully
+--------------------+----------------------------------------+
| Field              | Value                                  |
+--------------------+----------------------------------------+
| id                 | 569b3ded-c7d9-4ac4-a1a9-df3603d7c010   |
| stack_name         | firststack                             |
| description        | Provision a instance with a floating ip |
| creation_time      | 2022-12-05T20:34:23Z                   |
| updated_time       | None                                   |
| stack_status       | CREATE_COMPLETE                        |
| stack_status_reason | Stack CREATE completed successfully    |
+--------------------+----------------------------------------+
```

Figure 16.26: *Output example creating a stack*

With the argument **`stack output list`**, it is possible to obtain a list in the *outputs* section from the template, and with the argument **`stack output show`**, the value can be visualized. In the previous example, the assigned *Floating IP* is configured to be shown. *Figure 16.27* shows the *output* of the previous *stack* created:

```
[stack@client ~]$ openstack stack output list firststack
+---------------------+-----------------------------------------------------+
| output_key          | description                                         |
+---------------------+-----------------------------------------------------+
| instance_external_ip | Floating IP address of instance in External network |
+---------------------+-----------------------------------------------------+
[stack@client ~]$ openstack stack output show firststack instance_external_ip
+-------------+-----------------------------------------------------+
| Field       | Value                                               |
+-------------+-----------------------------------------------------+
| description | Floating IP address of instance in External network |
| output_key  | instance_external_ip                                |
| output_value | 10.0.0.228                                         |
+-------------+-----------------------------------------------------+
[stack@client ~]$ ssh -i ~/.ssh/id_rsa debian@10.0.0.228 lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda    254:0    0  20G  0 disk
└─vda1 254:1    0  20G  0 part /
vdb    254:16   0   5G  0 disk
```

Figure 16.27: *Output value examples for a stack*

Table 16.14 shows the typical tasks performed related to *Stacks*.

Arguments	Description
stack list	Lists the existing *Stacks*.
stack delete	Deletes an existing *Stack*, and the resources created
stack show	Shows all the information about the specified *Stack*.
stack update	Updates an existing *Stack* with a new version of the template.

Table 16.14: *Typical tasks related to Volumes*

Dashboard

The component **Horizon** is responsible for offering a Web dashboard to users for self-service operations. *Figure 16.28* shows an example of the aspect:

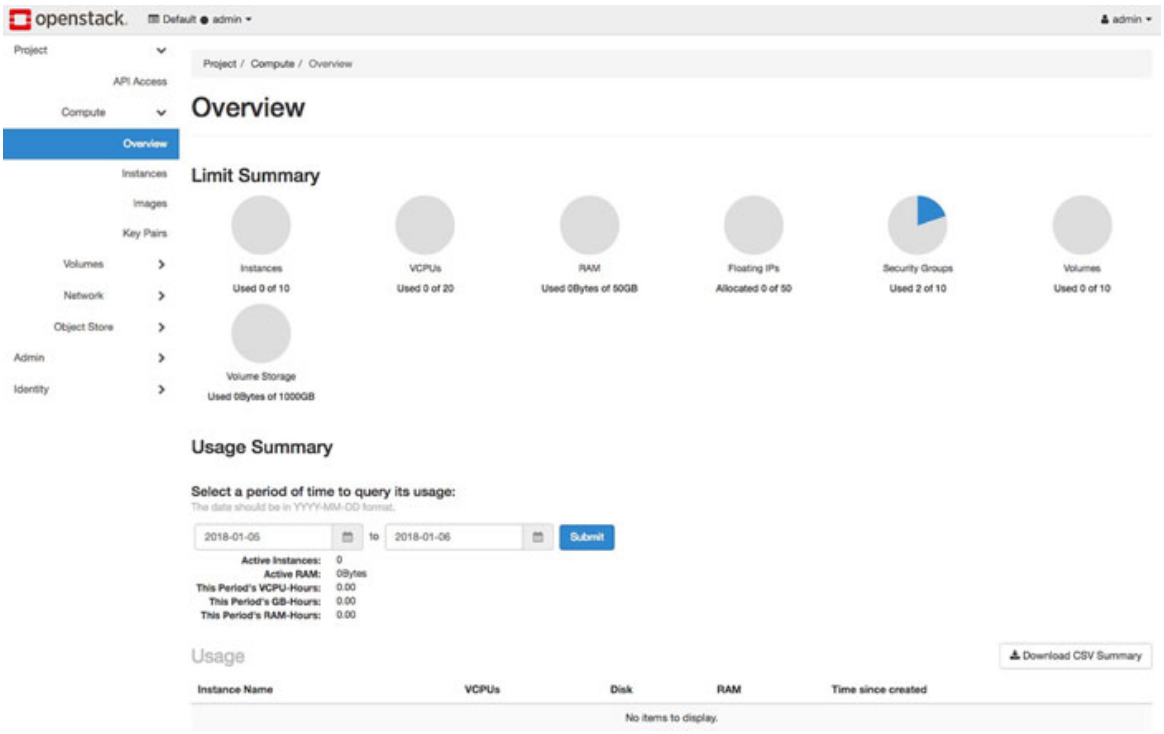


Figure 16.28: *OpenStack dashboard. Source: openstack*

Conclusion

Hybrid Cloud is a trending architecture to combine a *Private Cloud* with a *Public Cloud*. Many reasons are behind this, such as reducing costs and flexibility. This

chapter covered what is *Infrastructure as Service* is and detailed **OpenStack** as a *Cloud* solution.

Key facts

- **Infrastructure as a Service** allows converting on-premise infrastructure to a **Cloud**-ready infrastructure.
- **Private Cloud** can be combined with a **Public Cloud** to operate together. This combination is called **Hybrid Cloud**.
- **OpenStack** is the most important open-source project to build a **Cloud**.

Questions

1. What is the name for computing on **OpenStack**?

    a. glance
    b. nova
    c. neutron

2. Which is one of the most popular open-source solutions for storage?

    a. cifs
    b. ceph
    c. nfs

3. It is possible directly connect to the IP of a *Virtual Machine* attached to an overlay network. True or False?

    a. true
    b. false

4. Which project is used to create persistent volumes?

    a. glance
    b. swift
    c. cinder

5. What arguments are used to create a new stack on **OpenStack**?

    a. stack create
    b. heat create

c. template create

Answers

1. b
2. b
3. b
4. c
5. b

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z