People's Democratic Republic of Algeria
Ministry of higher education and scientific research
University of Kasdi Merbah Ouargla

Faculty of New Technologies of Information and Communication
Department of Computer Science and Information Technology

Master Thesis

Domain: Mathematics and Computer Science

Sector: Computer Science

Specialty: Industrial Computing

Theme

Configuration Management Automation:

A Comparative Study

Presented by:

Babai Manar Elwoudjoud & Noui Imane

Publicly discussed: 06/12/2025

Jury Members:

Dr. Khadidja Ameur	President	University of Ouargla
Dr. Maroua Meissa	Examiner	University of Ouargla
Dr. Fares Kahlessenane	Supervisor	University of Ouargla

Academic year: 2024/2025

Acknowledgments

I would like to express my sincere gratitude to my esteemed professor, **Kahlessenane Fares**, for his continuous support, insightful guidance, and valuable feedback, which had a significant impact on the development of this work. Thanks to his encouragement and close follow-up, I was able to overcome many challenges and move forward in completing this academic project.

I would also like to extend my deep appreciation to the members of the jury for taking the time to read and evaluate this work, and for their constructive comments, which will undoubtedly enrich this project and contribute to improving its academic quality.

Dedication

To my dearest parents,

Thank you for your boundless love and unwavering support. You have always been my source of strength and inspiration. Everything I've achieved is thanks to your prayers and kindness.

To my beloved siblings — Khaoula, Dhoha, Salaheddine, Anase, and Hudhaifa, the last-born,

Thank you for standing by my side through every challenge and every joy. I pray that our bond remains strong and everlasting.

To my precious friends — Islam, Inas, Noussaiba, Amira, and Jihane,

You are the light of my days and the reason behind my smile. I'm truly grateful for every beautiful moment we've shared and for your genuine support.

To a soul close to my heart,

Your silent support has meant the world to me. I pray our hearts always remain united in goodness and love.

And finally, to myself,

Keep moving forward with faith and hope, trusting that the best is yet to come. May God always bless you with happiness and guidance.

Noui Imane

Dedication

To my beloved parents,

Thank you for your endless love and steadfast support, and for every moment you spent by my side.

You are my inspiration and strength.

To my loyal brothers – Akram, Khairallah, Najmeddine, AbdelSabour, Abrar, AbdelIlah,

You are my lifelong companions and helping hands in hardship and joy.

I truly appreciate your presence in my life, and I pray that God will always keep the affection and harmony between us.

To my dear friends,

– Houria, Siham, Rym Dalal, Malak, Chahd, Manal, and to my best friend Arwa and to my little sister Riham, you are the light in my days and the reason behind my smile.

Thank you both for every moment we shared, and I pray that God will always bring us together in goodness and joy.

Finally, for myself,

I encourage her to continue striving and achieving, and I pray to God to guide me to the right path and give me lasting happiness and contentment.

BABAI MANAR ELWOUDJOUD

ملخص

شهد مجال تكنولوجيا المعلومات تطوراً سريعاً فرض تحديات متزايدة على المتخصصين في إدارة البنية التحتية الرقمية. وقد جعل ذلك من الأتمتة خياراً استراتيجياً لتعزيز الكفاءة وتبسيط العمليات التقنية. تفحص هذه الاطروحة دور أداة Ansible في أتمتة الشبكة، مع التركيز على تشغيلها بدون وكيل وسهولة التكامل، استنادًا إلى تجارب عملية في بيئة محاكاة باستخدام GNS3 و VMware. بالإضافة إلى ذلك، تم استخدام Python لتوسيع قدرات التحكم، بالاعتماد على مكتبات مثل Netmiko و Paramiko التي توفر وظائف قوية لإدارة الشبكة. أظهرت النتائج أن أنسيبل يتفوق على بايثون من حيث الأداء وسهولة الاستخدام، مما يجعله الخيار الأمثل لأتمتة مهام الشبكة بفعالية ومرونة. وتسلط هذه المذكرة الضوء على أهمية اعتماد أدوات الأتمتة لمواجهة تحديات العصر الرقمي وتسريع إدارة البنية التحتية الرقمية.

الكلمات المفتاحية: أتمتة الشبكات، أنسيبل، بايثون، إدارة التهيئة، YAML، Playbook.

Abstract

The field of information technology has experienced rapid development, which has imposed increasing challenges on specialists managing digital infrastructure. This has made automation a strategic choice for enhancing efficiency and streamlining technical processes. This thesis explores examine the role of the Ansible tool in network automation, emphasizing its agentless operation and ease of integration, based on practical experiences in a simulated environment using GNS3 and VMware. Additionally, Python was utilized to extend control capabilities, relying on libraries such as Netmiko and Paramiko that provide robust functions for network management. The results demonstrated that Ansible outperforms Python in terms of performance and ease of use, making it the optimal choice for effectively and flexibly automating network tasks. This thesis highlights the importance of adopting automation tools to address the challenges of the digital age and accelerate the management of digital infrastructure.

Keywords: network automation, Ansible, Python, Configuration management, YAML, Playbook.

Résumé

Le domaine des technologies de l'information a connu un développement rapide, ce qui a imposé des défis croissants aux spécialistes qui gèrent l'infrastructure numérique. L'automatisation est donc devenue un choix stratégique pour améliorer l'efficacité et rationaliser les processus techniques. Cette thèse examine le rôle de l'outil Ansible dans l'automatisation du réseau, en mettant l'accent sur son fonctionnement sans agent et sa facilité d'intégration, sur la base d'expériences pratiques dans un environnement simulé utilisant GNS3 et VMware. En outre, Python a été utilisé pour étendre les capacités de contrôle, en s'appuyant sur des bibliothèques telles que Netmiko et Paramiko qui fournissent des fonctions robustes pour la gestion du réseau. Les résultats ont démontré qu'Ansible surpasse Python en termes de performance et de facilité d'utilisation, ce qui en fait le choix optimal pour automatiser de manière efficace et flexible les tâches liées au réseau. Cette thèse souligne l'importance d'adopter des outils d'automatisation pour relever les défis de l'ère numérique et accélérer la gestion de l'infrastructure numérique.

Mots-clés : Automatisation réseau, Ansible, Python, gestion de la configuration, YAML, Playbook.

Table of Contents

List of figures

List of tables

List of Acronyms

AI	Artificial Intelligence
ACL	Access Control List
API	Application Programming Interface
ARP	Address Resolution Protocol
ARPANET	Advanced Research Projects Agency Network
CLI	Command-Line Interface
CMDB	Configuration Management Database
CPU	Central Processing Unit
DevOps	Development and Operations
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
FTP	File Transfer Protocol
GPLv3	GNU General Public License version 3
GNS3	Graphical Network Simulator-3
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
IAC	Infrastructure as Code
IBM	International Business Machines

ICMP	Internet Control Message Protocol
IMAP	Internet Message Access Protocol
INI	Initialization File
IOS	Internetwork Operating System
IP	Internet Protocol
IPX	Internetwork Packet Exchange
IRC	Internet Relay Chat
IT	Information Technology
JSON	JavaScript Object Notation
LAN	Local Area Network
MAC	Medium Access Control
MPLS	Multi-Protocol Label Switching
ML	Machine Learning
NFV	Network Functions Virtualization
OSI	Open Systems Interconnection
OS	Operating System
OSPF	Open Shortest Path First
POP3	Post Office Protocol 3
PPP	Point-to-Point Protocol
RSA	Rivest–Shamir–Adleman
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SPX	Sequenced Packet Exchange

SSH	Secure Shell
SUDO	Superuser Do
SU	Superuser
TCP/IP	Transmission Control Protocol / Internet Protocol
Telnet	Telecommunication Network
VTY	Virtual Teletype
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VM	Virtual Machine
VPN	Virtual Private Network
WAN	Wide Area Network
WinRM	Windows Remote Management
YAML	YAML Ain't Markup Language
ZTS	Zero Trust Security

General Introduction

As information technology continues to evolve at an unprecedented pace, modern organizations are increasingly confronted with the growing complexity of digital infrastructures. Networks have become larger, more dynamic, and more heterogeneous, making their management both technically and operationally challenging. Traditional manual configuration methods are no longer adequate, as they are prone to human error, delay deployments, and hinder scalability in environments where the volume of data and the number of connected devices is constantly increasing.

In this context, the adoption of configuration management automation tools has become essential. These tools minimize manual intervention, improve consistency and reliability, accelerate operational workflows, and reduce the overhead costs associated with managing complex infrastructures. Among the most prominent solutions in this domain, Ansible stands out as a flexible and user-friendly open-source automation platform. It enables agentless network management through simple YAML-based playbooks, making task automation across diverse environments intuitive and efficient.

In parallel, scripting languages such as Python play a pivotal role in enabling personalized automation. With libraries like Netmiko and Paramiko, Python provides a high degree of control and customization, allowing network engineers and system administrators to develop scripts personalized to specific needs, thereby enhancing the flexibility and adaptability of infrastructure management processes.

This study aims to analyze and compare the use of Ansible and Python scripting in a simulated environment, in order to evaluate their effectiveness, ease of use, integration capabilities, and suitability for different operational contexts. The goal is to identify the strengths and limitations of each approach, and to provide justifiable recommendations for selecting the most appropriate tool depending on the organization's technical requirements and environment constraints.

To achieve this goal, the study begins with a comprehensive theoretical overview of computer networks, network automation concepts and tools. This is followed by a practical phase in which a network topology is implemented using GNS3 emulator, allowing each tool to be tested independently. The efficiency of each tool is evaluated based on execution style, ease of use and verification Method.

The thesis is organized into three chapters:

Chapter 1 introduces fundamental concepts related to computer networks and network automation concept.

Chapter 2 presents Ansible-based and python-based automation tools, highlighting their principles and architectures.

Chapter 3 is dedicated to the applied study, in which simulations are performed and analyzed in order to assess the operational efficiency of each tool.

Chapter

1 Computer Networks

1.1. Introduction

In recent decades, computer networks have emerged as the foundation of the digital infrastructure for organizations, businesses, and government entities. Communication has evolved beyond mere data exchange between two devices; it now encompasses extensive systems that link thousands of devices over vast geographic regions, employing advanced technologies to guarantee speed, security, and reliability. According to Cisco's Annual Internet Report (2023), the global number of connected devices is projected to surpass 29 billion by 2025, highlighting the essential role of strong and automated networks in facilitating digital transformation. As networks grow, new challenges in their management and maintenance have arisen, prompting engineers to seek intelligent solutions for automating network operations to enhance efficiency and reduce human error. Following an incremental approach to network design, it has become crucial to comprehend how applications and services interact with the network infrastructure to ensure optimal performance. [1] **Network automation** the application of software and programmable tools to streamline network management tasks—has emerged as one of the most significant recent trends, utilizing software and technical tools to ease the configuration, management, testing, and operation of both physical and virtual devices. By minimizing manual intervention, organizations can realize substantial operational cost reductions, decrease errors, and expedite service delivery. Although the idea of network automation is not novel, its significance has surged dramatically due to the demand for flexibility and swift responses to growth. An incremental approach to network design aids in understanding the impact of applications and services on network performance, thereby making automation more straightforward and efficient. [2]

1.2. Computer network

A computer network is composed of various types of **nodes**, which include network devices (such as switches and routers), servers, personal computers, and other general-purpose or specialized hosts. A **node** refers to any device that is connected to the network and has the capability to send, receive, or forward information. Each node in the network is identified by either a host name or a network address, which enables it to be recognized and accessed within the system.

Essentially, a **network** is a collection of devices that can communicate with one another through communication links. Every device that is connected is regarded as a host, and each host has a unique address that differentiates it from others, thereby ensuring precise routing and data exchange within the network.[3]

1.2.1. Definition of computer networks

A **computer network** is a cohesive system that connects two or more devices, facilitating the sharing of information and resources (such as files, printers, and databases) through wired or wireless connections. Communication within the network is regulated by standardized protocols like TCP/IP, which guarantee interoperability and dependable data exchange between devices. The network comprises various nodes—including network devices (switches, routers), servers, personal computers, and specialized hosts—each assigned a distinct network address for identification and communication. The history of networking traces back to the late 1960s with the establishment of ARPANET by the U.S. Department of Defense. ARPANET established the groundwork for the modern Internet, which now enables global business, entertainment, and communication across continents.[4]

1.2.2. Types of computer networks

Computer networks are commonly classified by their geographic coverage into three main types: Local Area Network (LAN), Metropolitan Area Network (MAN), and Wide Area Network (WAN). [5]

Table 1.1 – Comparative Table of Network Types

Criterion	**LAN**	**MAN**	**WAN**
Geographic Scope	Building, office, school	City, campus	Countries, continents
Speed	High	Medium to high	Medium to low
Cost	Low	Moderate	High
Ownership	Individual/organization	Government/large org.	Multiple providers
Ease of Management	Easy	Moderate	Complex
Usage	Local resource sharing	Linking city institutions	Connecting distant branches
Technologies Used	Ethernet, Wi-Fi	Fiber Optic, MPLS	MPLS, VPN, Satellite
Practical Examples	Network within a small company	Network connecting universities in a city	Internet, global banking network

a. Local Area Network (LAN)

A Local Area Network (LAN) links computers and peripheral devices within a limited area, such as a building, school, or office, thereby facilitating communication, data sharing, and access to shared resources like printers and files through Ethernet cables or wireless technologies like Wi-Fi. LANs are recognized for their high speed, robust security, and comparatively low cost when contrasted with larger network types. For instance, in a standard office setting, all workstations and devices are interconnected through a LAN, which allows employees to seamlessly share documents, utilize shared printers, and maintain uninterrupted Internet connectivity.

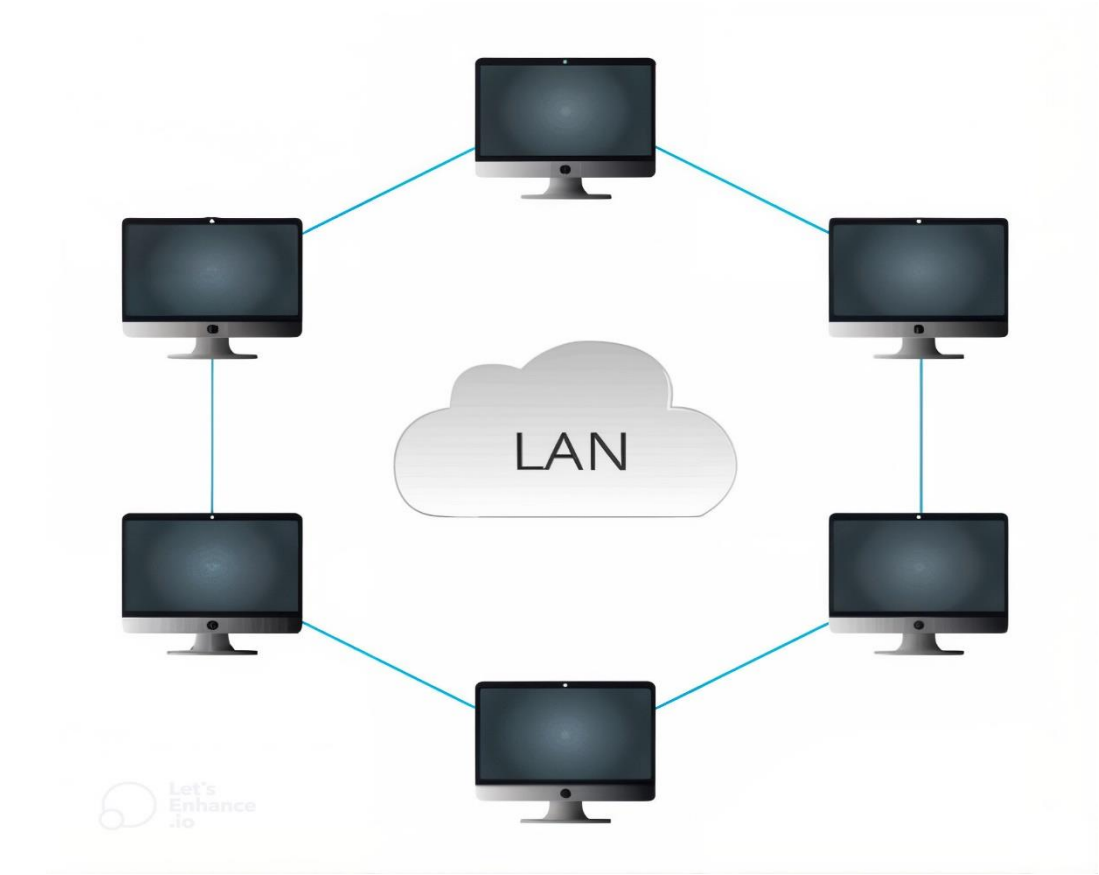

Figure 1.1 – Local area network.[6]

b. Metropolitan Area Network (MAN)

A Metropolitan Area Network (MAN) encompasses a broader area than a Local Area Network (LAN) yet is more limited in scope than a Wide Area Network (WAN), generally covering a city or a substantial campus. It interconnects several LANs utilizing high-speed fiber optic cables or wireless technologies and is typically overseen by municipalities or large organizations. For instance, a university might connect all its colleges and facilities via a MAN, allowing students and researchers to effortlessly access digital libraries and databases throughout the entire campus.

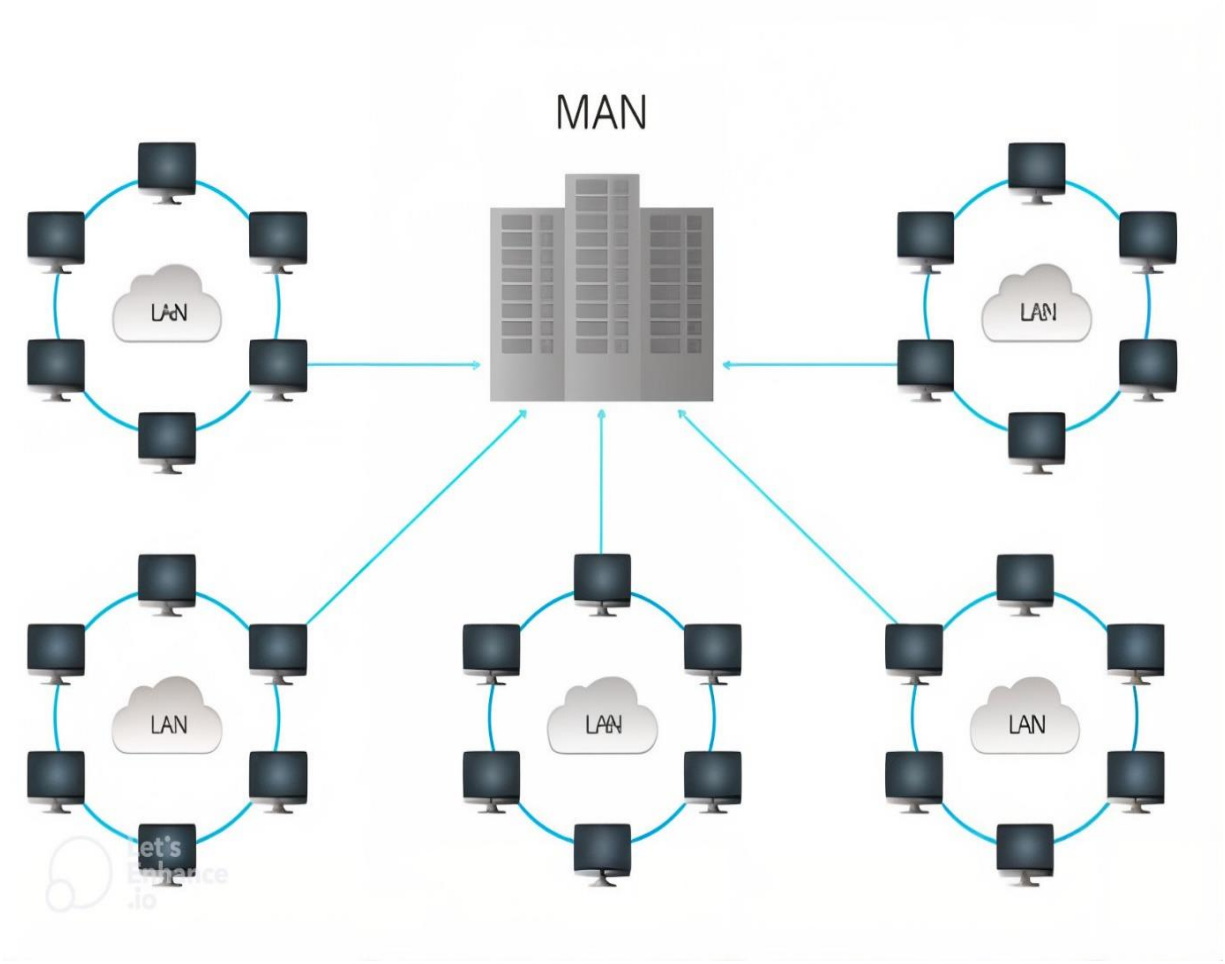

Figure 1.2 – Metropolitan area network.[6]

c. Wide Area Network (WAN)

A Wide Area Network (WAN) links various Local Area Networks (LANs) over extensive geographic regions, including cities, nations, or continents, by employing technologies such as leased lines, fiber optics, satellites, or the public Internet. Importantly, the Internet serves as the most significant example of a WAN. For example, a financial organization with branches in multiple countries depends on a WAN to connect all its branches to a central data center, thus facilitating secure and immediate information transfer.[7]

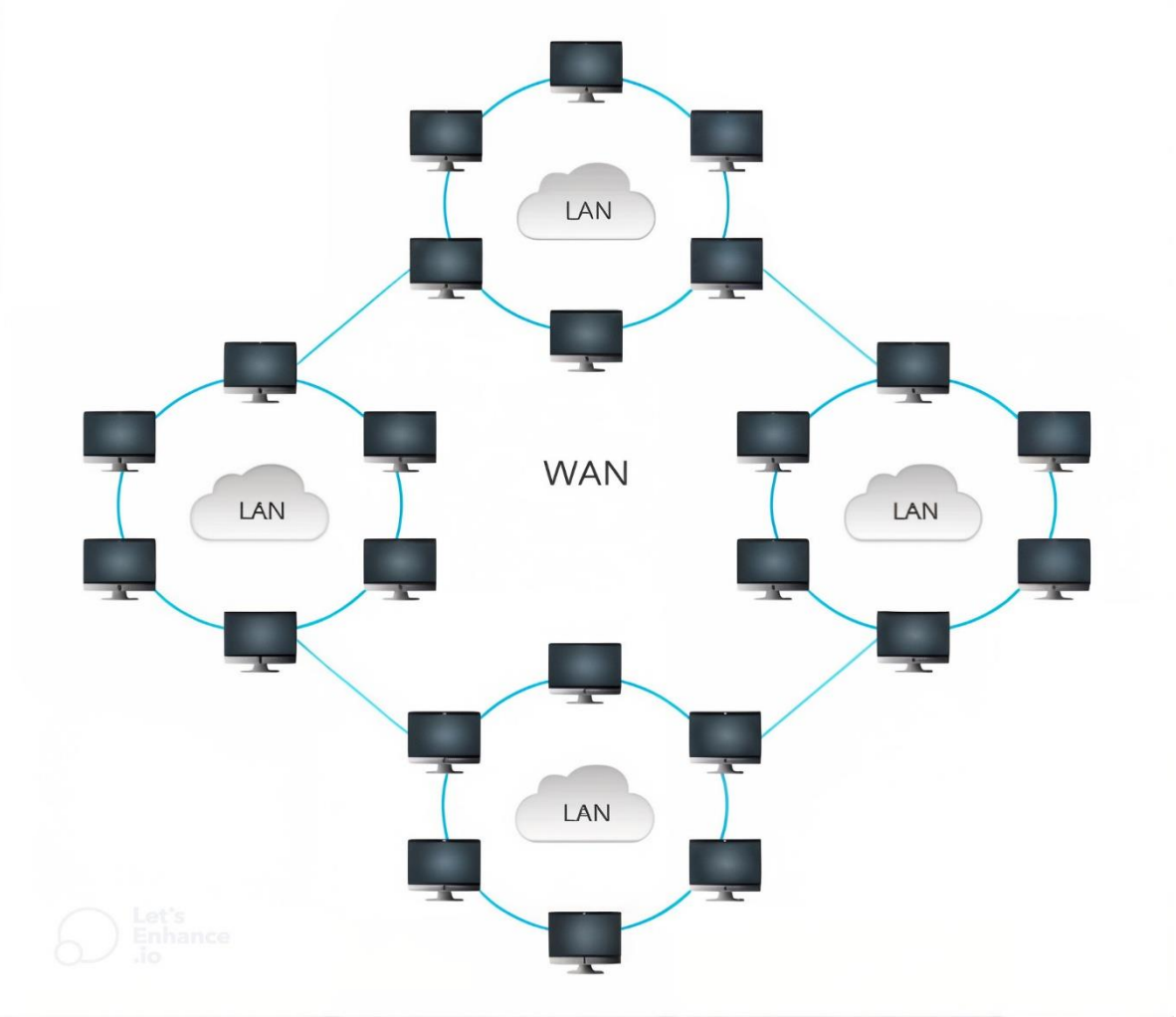

Figure 1.3 – Wide area network[6]

1.2.3. Network Architectures

Network architecture defines how devices interact and how roles are distributed. The two primary models are:

- **Peer-to-Peer (P2P)**
- **Client/Server**

a. Peer-to-Peer (P2P)

A Peer-to-Peer (P2P) network represents a decentralized framework where users can share resources and files directly among their computers, bypassing the need for a central server. In this setup, every device functions both as a client and a server, which allows for efficient and scalable sharing of resources. P2P networks are easy to establish, economical, and are frequently utilized in small to medium-sized settings. Most contemporary operating systems, such as Windows, Linux, and macOS, are compatible with this model. For instance, in a small office environment, employees can directly exchange files between their computers without relying on a central server, thereby promoting seamless collaboration and resource sharing.[8]

b. Client/Server

The Client/Server architecture consolidates functions within specialized servers that oversee resources and security for client devices. This methodology is frequently utilized in larger enterprises due to its provision of superior control and heightened security. Within this framework, servers take on the roles of data storage, security enforcement, and resource distribution, thereby guaranteeing centralized management and scalability. For instance, a medium-sized business might employ a central server to store files and regulate access permissions for all staff members, thus optimizing operations and protecting sensitive data.[9]

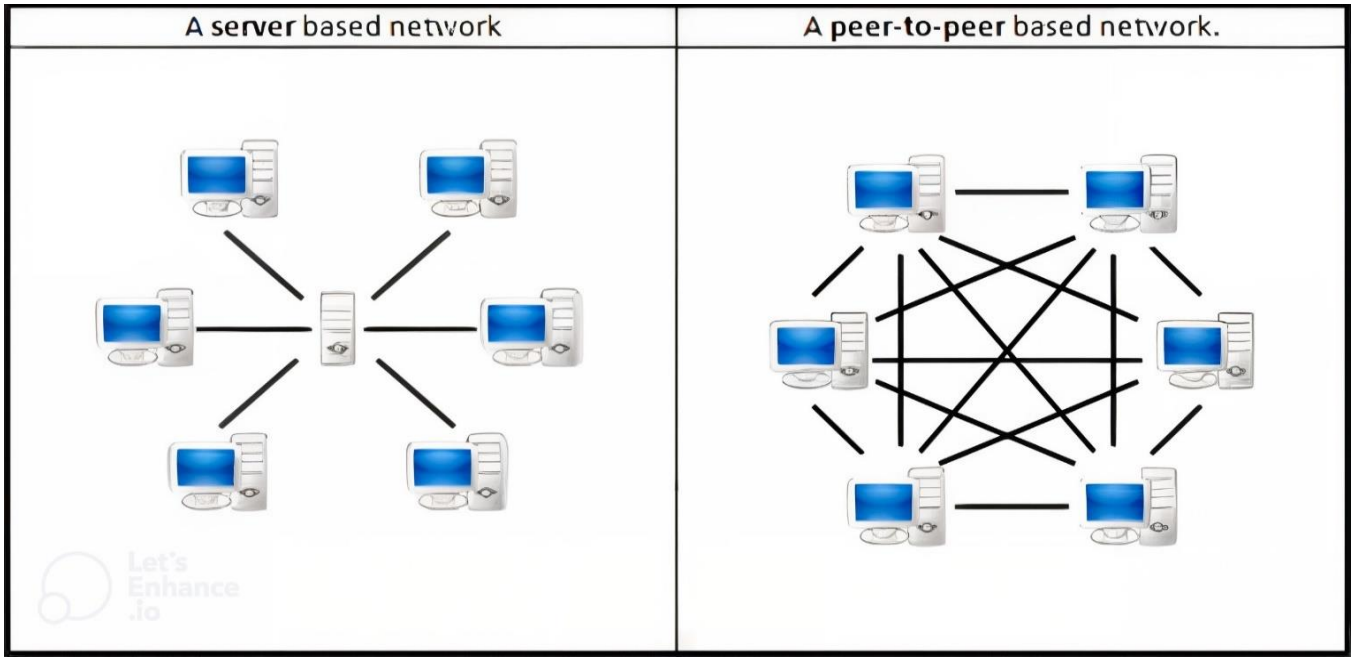

Figure 1.4 – Client/server network

Figure 1.5 – Peer-to-peer network

1.2.4. Network Protocol

A protocol refers to a collection of rules that regulates communication between computers within a network. For two computers to communicate effectively, they must utilize the same language. A variety of network protocols and standards are necessary to guarantee that your computer—regardless of the operating system, network card, or application in use—can connect with another computer, whether it is situated on the adjacent desk or halfway across the globe. The OSI model organizes network communication into seven separate layers, each designated for specific functions.

Table 1.2 – OSI model related to common network protocols.[10]

OSI Layer	Name	Common Protocol
7	Application	HTTP | FTP | SMTP | DNS | Telnet
6	Presentation	/
5	Session	/
4	Transport	TCP | SPX
3	Network	IP | IPX
2	Data Link	Ethernet
1	Physical	Ethernet

Most modern networks rely on the TCP/IP model, which consists of four main layers, each with a specific role in data transmission:

- **Application Layer:** Provides an interface for applications and network services like email and web browsing, including protocols such as HTTP, SMTP, and FTP.
- **Transport Layer:** Ensures reliable data transfer between devices, managing flow control and error correction with protocols like TCP and UDP.
- **Internet Layer:** Responsible for addressing, routing, and forwarding packets across different networks using protocols such as IP and ICMP.
- **Network Access Layer:** Handles physical connection and data transfer between devices within the same network using protocols like Ethernet and ARP.

These layers work together to ensure efficient and reliable data transfer across networks. Several protocols overlap the session, presentation, and application layers of networks. There protocols listed below are a few of the more well-known:

- **DNS** - Domain Name System - translates network address (such as IP addresses) into terms understood by humans (such as Domain Names) and vice-versa
- **DHCP** - Dynamic Host Configuration Protocol - can automatically assign Internet addresses to computers and users
- **FTP** - File Transfer Protocol - a protocol that is used to transfer and manipulate files on the Internet
- **HTTP** – Hyper Text Transfer Protocol - An Internet-based protocol for sending a receiving webpage.
- **IMAP** - Internet Message Access Protocol - A Protocol for e-mail messages on the Internet.
- **IRC** - Internet Relay Chat - a protocol used for Internet chat and other communications.
- **POP3** - Post Office protocol Version 3 - a protocol used by e-mail clients to retrieve messages from remote servers.
- **SMTP** - Simple Mail Transfer Protocol - A protocol for e-mail messages on the Internet**.**

1.2.5. Network Topology

The physical topologies commonly used in networks include:

- **Linear Bus**
- **Star**
- **Ring**
- **Tree (Expanded Star)** [11]

a. Linear Bus

All nodes (file server, workstations, and peripherals) are connected to the linear cable. (See fig.1.6) [12]

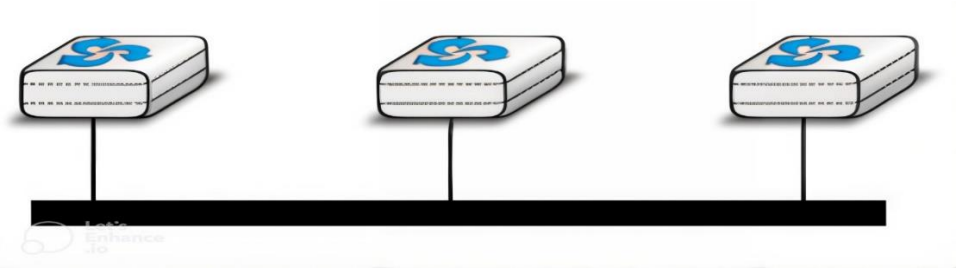

Figure 1.6 – Linear Bus topology.

b. Star

A star topology is designed with each node (file server, workstations, and peripherals) connected directly to a central network hub, switch, or concentrator (See fig. 1.7).

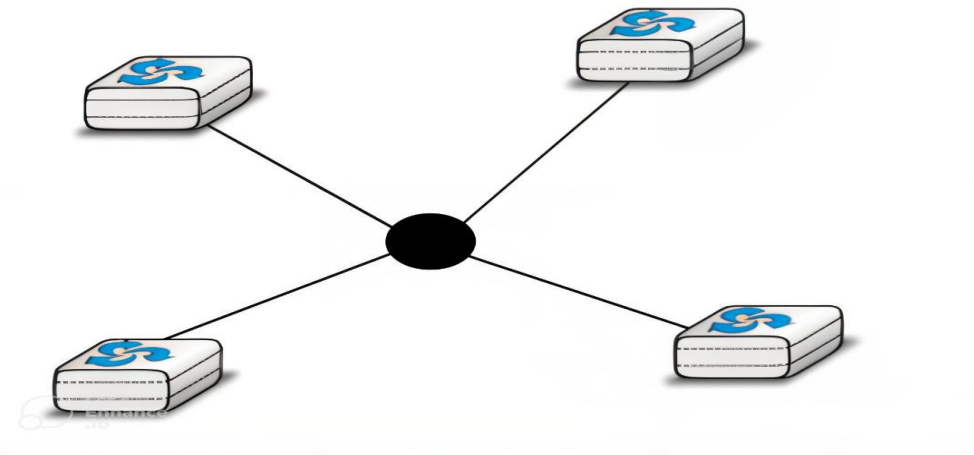

Figure 1.7 – Star topology.

c. Ring

A ring network is a configuration in which every node is connected to two others in a continuous loop. Data travels in each node that processes every packet in the path. (See fig. 1.8)

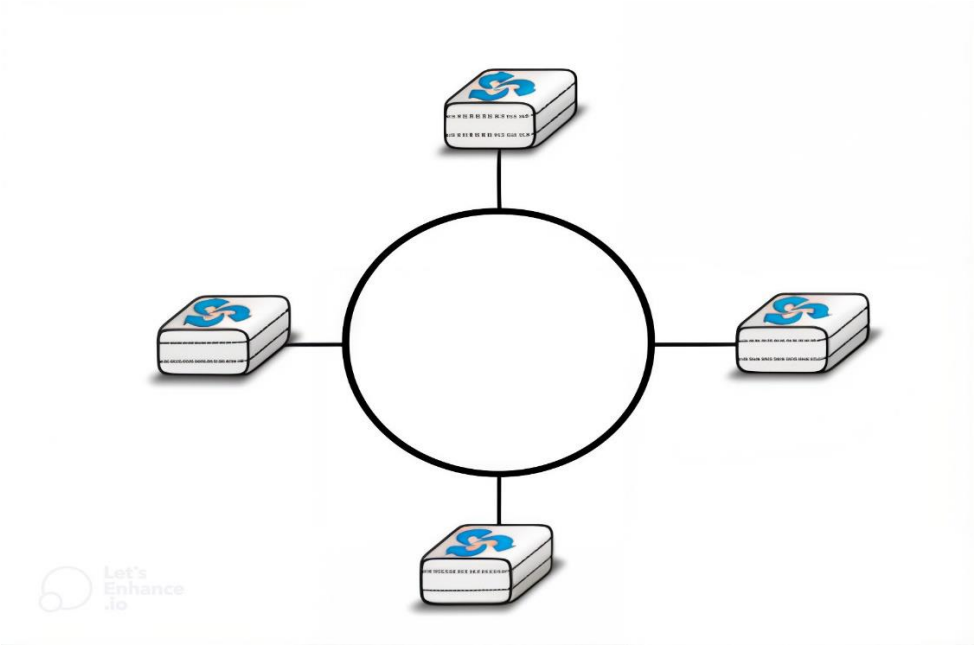

Figure 1.8 – Ring topology.

d. Tree or Expanded Star

A star topology is designed with each node (file server, workstations, and peripherals) connected directly to a central network hub, switch, or concentrator (See fig. 1.9).

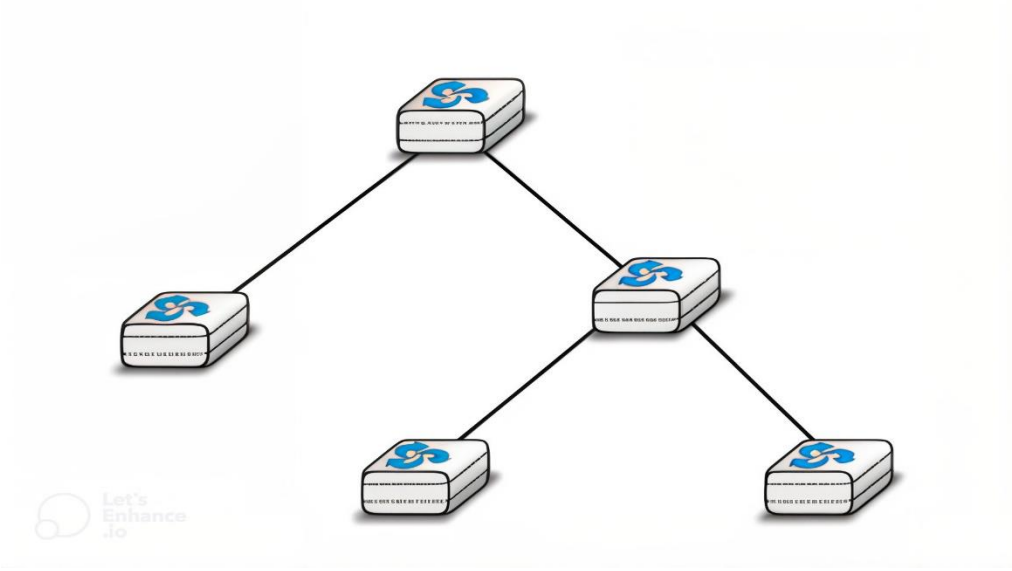

Figure 1.9 – Tree topology.

- **Considerations When Choosing a Topology:**
  - **Cost:** Linear bus network is likely to be the least expensive to implement a network; you do not have to purchase concentrators.
  - **Amount of cable required:** Less length of cable is required for the linear bus network.
  - **Expansion:** Expanding the network in star topology is a simple task of adding a different concentrator.
  - **Type of cable:** Most commonly used cable in schools is unshielded twisted pair, and this cable is used most often with star topologies.

1.3. Network Automation

Network automation pertains to the method of automating the configuration, management, and functioning of computer networks. Tasks that were previously executed manually by network or system administrators can now be automated through various tools and technologies. This automation is crucial as human errors are a significant contributor to problems such as service outages, downtime, and security breaches within network environments. By adopting effective automation, organizations can greatly diminish these errors, expedite network operations, and reduce both time and costs. The implementation of network automation depends on hardware and software solutions that automatically perform routine tasks within the network. Network and system administrators frequently utilize scripting languages to automate these processes, which conserves time, lessens effort, and decreases human errors. Among the most recognized automation tools are Python and Ansible. Moreover, with the emergence of Software-Defined Networking (SDN), expertise in programming languages like these has become increasingly vital, influencing the future of network and system administration.[7]

1.3.1. Definition

Network automation involves utilizing software tools to facilitate the automation of configuration, management, and operation of both physical and virtual network devices. This process enhances the efficiency of routine tasks, minimizes human error, and reduces operational expenses. Automation is extensively applied in various areas, ranging from device discovery to intricate workflows such as configuration management and virtual resource provisioning. It plays a vital role in software-defined networking (SDN), network virtualization, and orchestration. For instance, In a technology firm, a network engineer used to manually update router configurations, a process

that was both time-consuming and prone to errors. However, with the implementation of Ansible, updates are now automated across all devices within minutes, accompanied by immediate reporting.[13]

1.3.2. Types of Network Automation

Network automation can be applied to various types of networks, including LANs, WANs, data centers, cloud, and wireless networks, provided they are accessible through CLI or APIs.[14] The following is a summary of the main types of network automation:

- **Script-based Automation:** Uses scripting languages like Python, Ansible, Ruby, and legacy tools like Perl and Tcl to automate repetitive tasks by scripting. Ideal for well-defined procedural environments.
- **Software-based Automation:** Relies on platforms with graphical interfaces and templates to execute tasks without manual scripting, making automation more user-friendly and easier.
- **Intent-based Automation:** Includes AI and machine learning to align network behavior with business objectives. Dynamically adjusts policies for assurance of service levels and application performance.
- **Network Orchestration:** Automates the deployment and management of network services such as virtual networks, firewalls, and load balancers. Terraform and OpenStack are commonly utilized tools.
- **API-based Automation:** Utilizes APIs to control network devices and integrate automation with broader workflows. Allows for tailored, large-scale automation methodologies.
- **Security Automation:** Automates security tasks like patching, policy enforcement, and threat detection to help improve compliance and reduce risk.
- **Cloud Network Automation:** Manages and automates cloud network resources, delivering efficient provisioning, configuration, and monitoring in cloud environments.
- **Data Center Automation:** Is focused on automating data center operations like resource allocation and performance monitoring, which makes them more efficient and scalable.
- **Monitoring Tools (Surveillance):** Monitoring solutions provide real-time insight into the performance of networks and allow the detection of faults before they affect users. Such critical tools include Nagios, Zabbix, SolarWinds, and Prometheus with Grafana, all needed to guarantee network reliability and transparency.[15]

1.3.3. Network automation tools and languages

a. **Orchestration Tools**

- **Ansible:** Ansible is an orchestration tool by which users can utilize YAML-based playbooks to declare infrastructure as code. It is primarily used for configuration management, application deployment, and network automation orchestration.
- **Cisco DNA Center:** Cisco DNA Center is a centralized network automation and management platform with AI-powered capabilities for automated provisioning, policy-based enforcement, and performance assurance for Cisco devices.
- **Juniper Networks Junos Automation:** Junos Automation tools offered by Juniper Networks include Junos OS scripts, automation scripts, and Junos PyEZ (Python library) for managing Junos devices.

b. **Monitoring Tools**

- **Python:** Python is widely used network automation for monitoring due to its ease of use and extensive libraries, facilitating network data collection and analysis.

c. **Security Tools**

- **JavaScript:** JavaScript is applied primarily to web-based network automation programs and custom scripts, talking to APIs and web interfaces to impose network security.[8]

1.3.4. Benefits and challenge of network automation

a. **Benefits:**

- **Improved Efficiency:** Automates routine tasks, saving time and optimizing performance.
- **Reduced Human Error:** Minimizes mistakes common in manual configurations.
- **Enhanced Security:** Ensures consistent policy enforcement and rapid vulnerability response.[16]

b. **Challenges:**

- **Device Incompatibility:** Older devices may lack API support.
- **Tool Complexity:** Advanced tools (especially those using AI/ML) can be complex to implement.
- **Skill Requirements:** Effective automation requires specialized knowledge.[17]

1.3.5. Future of Network Automation

The future of network automation is evolving rapidly, driven by technological advancements. Key trends include:

- **AI and ML:** These technologies enhance predictive analytics, fault detection, and decision-making, improving performance and reducing errors.
- **Intent-Based Networking (IBN):** IBN translates business objectives into automated network configurations, reducing errors and improving efficiency.
- **Zero Trust Security:** With rising cyber threats, Zero Trust principles ensure secure access through strict controls and encryption.
- **SDN and NFV:** These technologies enable flexible, software-based network management, reducing reliance on traditional hardware.[18]

1.4. Conclusion

In conclusion, network automation is a rapidly evolving field that offers significant benefits, including improved efficiency and reduced errors. It enables organizations to enhance network performance and reduce human error by automating repetitive tasks, allowing network administrators to focus on more strategic responsibilities. Automation tools come in various forms, including scripting, APIs, orchestration, and Software-Defined Networking (SDN). However, network automation also presents several challenges such as complexity, integration issues, cost, security concerns, maintenance, and organizational resistance to change. Therefore, successful implementation requires proper planning, adequate training, and the adoption of best practices such as continuous monitoring.[11]

Chapter

2

Automation with Ansible and Python

2.1. Introduction

The swift proliferation and escalating diversity of network apparatus have rendered conventional manual network management techniques inefficient, prone to errors, and demanding of substantial resources. Contemporary organizations increasingly depend on network automation to optimize configuration, deployment, and maintenance procedures, thereby substantially diminishing human errors and operational expenditures. Instruments such as **Ansible** and **Python** have emerged as fundamental components in this paradigm shift, empowering administrators to automate repetitive functions, guarantee uniformity across systems, and expedite service delivery. The incorporation of automation into DevOps methodologies further narrows the divide between development and operations teams, fostering collaboration and enhancing agility. Furthermore, virtualization technologies enable the emulation and assessment of automated scenarios, thereby simplifying the management of intricate infrastructures. Consequently, network automation has become an indispensable element for organizations striving to attain scalability, reliability, and operational efficiency within their IT ecosystems.

2.2. Ansible

Ansible constitutes an open-source automation framework meticulously crafted to enhance the management of information technology infrastructure. It facilitates the automation of various tasks, including provisioning, configuration management, application deployment, and orchestration across both Unix-like and Windows operating systems. A distinguishing characteristic of Ansible is its agentless architecture, which indicates that it does not necessitate the installation of supplementary software on the managed nodes. Rather, it establishes a remote connection via SSH for Linux/Unix systems or through Windows Remote Management (WinRM) for Windows hosts to execute the prescribed tasks.[19]

Ansible employs a straightforward, human-readable declarative language grounded in YAML to articulate automation tasks, rendering it accessible even to individuals possessing minimal programming skills. Its modular design enables users to augment its functionality through the incorporation of custom modules and plugins. Owing to its inherent simplicity, robust security, and formidable capabilities, Ansible has emerged as a preferred option for network and system automation within contemporary IT environments.[20]

2.2.1. Ansible History

The term "Ansible" is derived from the realm of science fiction, initially introduced by the esteemed author Ursula K. Le Guin in 1966, to denote a device that facilitates instantaneous communication over extensive distances. Motivated by this notion, Michael DeHaan developed the Ansible automation tool in 2012, with the objective of delivering a straightforward, agentless solution for information technology automation. Following its introduction, Ansible rapidly attained prominence owing to its user-friendliness and adaptability. In 2015, Ansible was acquired by Red Hat, a prominent provider of open-source solutions. Subsequently, in 2018, Red Hat itself became integrated into IBM. Presently, Ansible continues to progress as an open-source initiative under the aegis of Red Hat (an IBM subsidiary), and it is extensively utilized by organizations globally for the automation of IT operations, encompassing network, cloud, and server management.[19]

2.2.2. Ansible Architecture

Ansible's architecture revolves around two core components:

- **The Ansible server:** commonly referred to as the Node control machine, it is the Ansible machine.
- **Hosts:** also called managed nodes are the machines on which Ansible will perform tasks.

Ansible establishes connections to devices through SSH, runs designated Modules within a Playbook, and depends on an Inventory File to specify target hosts. Additionally, it can integrate with Cloud Services and CMDB when utilizing Dynamic Inventory.[21](See figure 2.1)

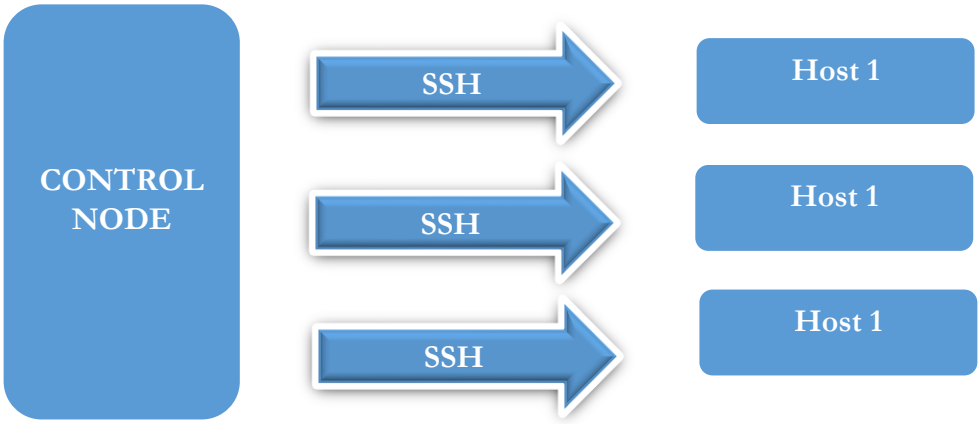

Figure 2.1 – SSH Connection Diagram in Ansible.

a. Users

Represents the control machine. This is the machine where Ansible is installed and on which is created the inventory and the Playbook, to push tasks to hosts via an SSH connection. The control node requires Linux and Python 3+, while managed nodes need only SSH and Python

b. Hosts

Also called managed nodes are the machines on which Ansible will push automation tasks. It is not necessary to install Ansible on managed nodes the only prerequisite is to install SSH on the hardware as well as the python 3 language.

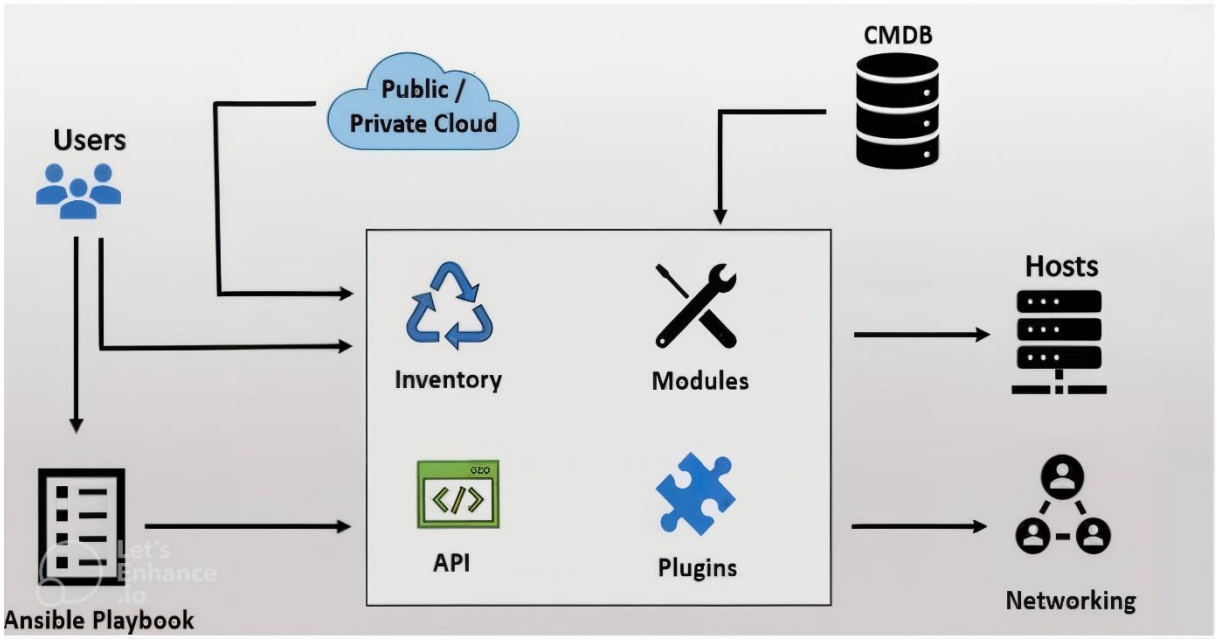

Figure 2.2 – Ansible Operating Principle.

c. Networking

Ansible can also be used to automate different networks. Ansible uses the same simple, powerful and agent-free automation framework that IT operations and development are already using. It uses a separate data model (playbook) from the Ansible automation engine that easily covers different network hardware.

d. Cloud

It is a network of remote servers hosted on the Internet to store, manage and process data, rather than a local server. It is possible to launch the resources and instances on the cloud to connect to servers.

e. Inventory

This is a default file written in INI format but it can also be written in YAML format. It mainly contains the names of hosts and their IP addresses in order to identify them for use in a playbook or ad-hoc commands.[22]

An inventory can be created statically with a collection of defined hosts as explained above, or create dynamically through a dynamic script querying the CMDB. The inventory file is located in/etc/ansible/hosts by default.

Figure 2.3 – Creating an inventory with IP address.

f. CMDB

Is a repository that acts as a data warehouse for IT facilities. It contains data about a collection of computer assets (commonly referred to as CI configuration items), and describes the relationships between these assets.

g. Module

Modules in Ansible are programs that are executed on Managed Nodes or locally on the Controller for the purpose of network management. They can be run from the Command Line Interface (CLI) without the need for a Playbook by utilizing Ad-hoc Commands, which operate through SSH using the JSON Protocol, and are removed after their execution.

h. Tasks

It is a written instruction in YAML that uses an Ansible module.

i. Roles

A role is a tree structure made up of directories and YAML configuration files, whose function is the installation of a system. Roles can be nested and interrelated. They contain multiple tasks (Tasks) and allow you to organize them.

j. Plugins

Plugins allow you to run Ansible tasks as a job generation step. Plugins are pieces of code that enhance the basic functionality of Ansible.[19]

There are several types of plugins, which are specialized kinds of "modules". Some of them are:

- connections: ssh, winrm, ...
- inventory: ini, yaml, scripts, list, ...
- become: su, sudo, enable, runes

k. Playbook

A playbook delineates a sequence of Tasks or Roles encapsulated within a YAML file. It represents the intersection of hosts and tasks. Indeed, this is the locus where we delineate the operations that must be executed utilizing Ansible. The document primarily encompasses the host group in question by referencing the inventory, in addition to a role play (a sequence of tasks or roles) that incorporates Ansible modules. The execution of the tasks outlined in a playbook transpires in accordance with their specified reporting sequences.

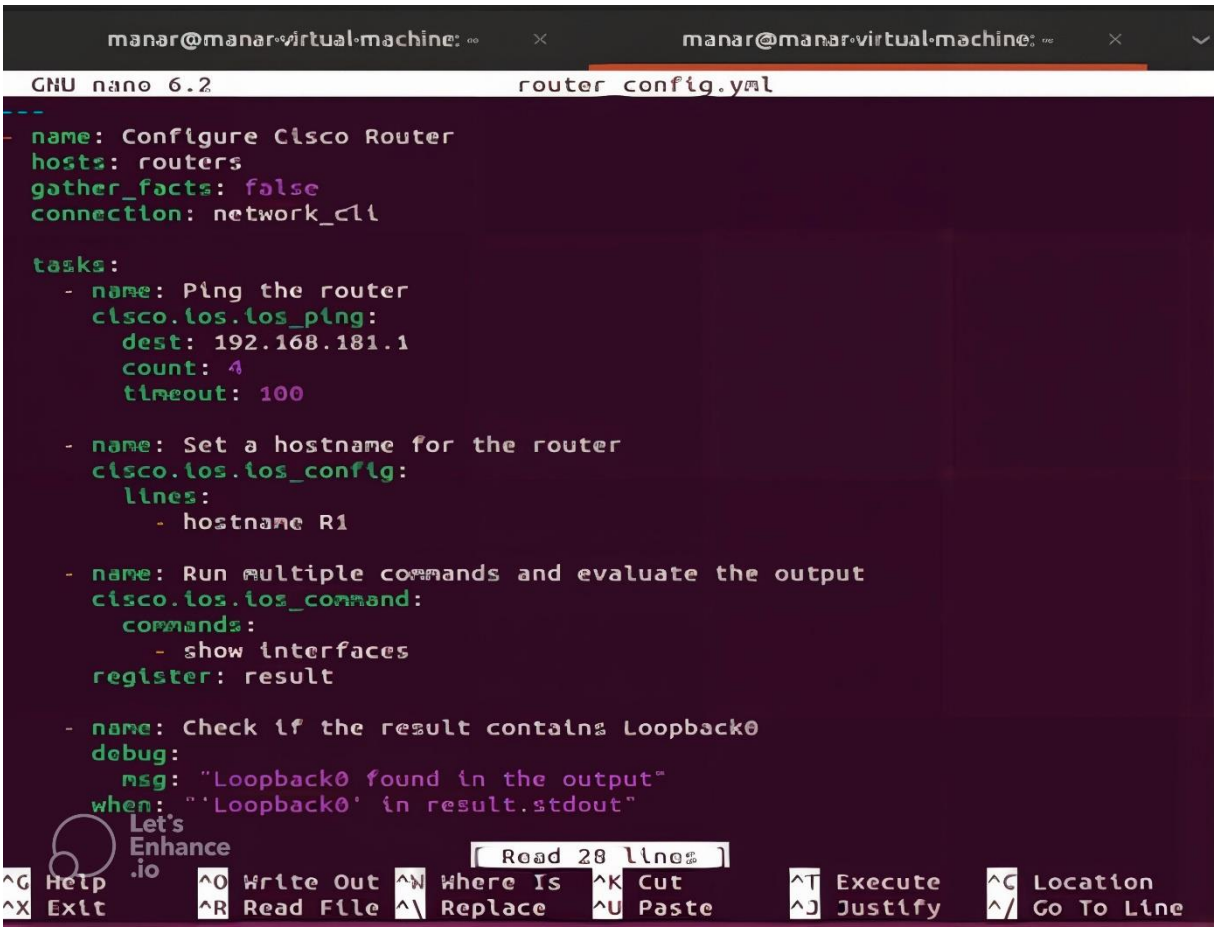

Figure 2.4 – Representation of an example playbook.

l. API

The application programming interface (API) is used in Ansible as a communication solution with cloud services, public or private.[23]

m. Variable

The variables in Ansible are exactly the same as in other languages. A variable is considered to be a 'name' attached to a specific 'object'. You can define these variables in playbooks, in an inventory, in reusable files or roles, or on the command line. To create a variable, simply assign it a value, for example syslog_ip=10.10.5.20 and then refer to an IP address anywhere in Ansible using the syslog_ip key instead of the IP address.

n. Collection

"Collections" is a distribution format for Ansible content that can include playbooks, roles, modules and plugins. Collections are distributed by Ansible Galaxy via ansible-Galaxy collection Install <namespace.collection>*.[24]

2.2.3. Language and Protocol

Technically, Ansible is based on the following languages or protocols:

- Platform and modules developed in Python
- The YAML (Yet Another Markup Language) is a human-readable data serialization format used for playbooks and inventories. Which means it is used to represent data rather than documents. It is used for the declarative part including the playbook and inventory.
- The SSH (secure shell) protocol is finally used for communication with target machines.

2.2.4. Area of application

Ansible is capable of automating various IT processes[25], such as:

- **Configuration management**

Configuration management is a process that keeps IT systems, servers and software in the desired state and maintains consistency. This is a way to ensure that a system works as intended as changes are made. For example, it can set up a server and then create its systems and ensure their smooth operation.

Ansible allows us to accelerate change and deployments, avoid the risk of human error, and make systems management more predictable and scalable. In addition, it allows you to track the status of resources and avoid repeating tasks, such as installing the same package twice.

- **Orchestration**

Orchestration describes how to automate a process consisting of many steps performed on several different systems. Ansible allows us to orchestrate the deployment by executing the tasks in the playbook in the order they were written, while ensuring that the process is going as planned.

- **The application deployment**

Automation of deployment allows software to move between test and production environments using automated processes. Thus, it ensures reproducibility and reliability of deployments throughout the distribution cycle.

- **Provisioning**

The first step in automating the application lifecycle is to automate infrastructure provisioning. Ansible provisions bare-metal servers, cloud instances, network devices, and virtualized infrastructure.

2.2.5. Basic option for using Ansible

When a command line instruction is launched, certain keyword are used:[26]

Table 2.1 – Core Ansible CLI Options.

-u	Remote User used
-b	Pass commands at privilege elevation (sudo)
-k/ --ask-pass	Password SSH
-K/ --ask-become-pass	Password for privilege escalation
-C/ --check	Run a dry run
-D/-diff	Have an outpout of the difference
--key-file	Direct link to private key
-e/ -extra-vars	Define variables
-ask-vault-pass	Decrypt a secret vault
-vault-password-file	File to decrypt

-m	Module
-i	Inventory
-f	Parallelize
-vvv	verbose

2.2.6. Benefits and challenge

Automation tools have become essential in modern IT environments, and Ansible stands out for its unique approach and capabilities. Below is an overview of its main benefits and some challenges users might face.[27]

a. Benefits

Among the strengths of Ansible are the following:

- **Open-source:** Ansible is freely available under the GPLv3 license, with enterprise support via Red Hat
- **Very easy to set up and use:** No coding skills are required to use the Ansible playbooks.
- **Powerful:** Ansible allows you to model even very complex IT workflows.
- **Flexible:** it has the ability to orchestrate the entire application environment, regardless of where it is deployed. As it is possible to customize according to the needs.
- **No agent:** it is not necessary to install other software or firewall ports on the client systems to be automated and to set up a separate management structure.
- **Language:** the python language is very easy to learn and understand by humans, as well as the use of the YAML format in inventory and playbook.

b. Challenge

- **Limited support for Windows:** In the case of Windows, Ansible uses native PowerShell remote communication rather than SSH. Therefore, a Linux control machine is required for the management of Windows hosts.
- **Relative maturity:** While Ansible is newer than competitors like Puppet or Chef, its community and Red Hat backing mitigate this limitation.

2.3. Python in Network Automation

Python is a versatile and widely-used programming language that has become a cornerstone in the field of network automation. Thanks to its simple syntax, extensive standard libraries, and strong community support, Python enables network engineers to automate repetitive tasks, manage configurations, and interact with network devices from various vendors. Python scripts can be used

for device configuration, monitoring, data collection, and troubleshooting, making it a preferred tool for both beginners and experienced professionals in network engineering. One of the key strengths of Python in network automation is the availability of specialized libraries such as **Netmiko**, **NAPALM**, and **Paramiko**. These libraries provide ready-to-use functions for connecting to network devices via SSH or API, retrieving device information, and applying configuration changes programmatically. With Python, engineers can automate complex workflows, integrate with other automation tools like Ansible, and build custom solutions tailored to their specific network environments.

Furthermore, Python's compatibility with DevOps practices and its ability to integrate with APIs and cloud platforms have made it a fundamental component in modern network automation strategies. As network infrastructures become more dynamic and software-defined, Python continues to play a vital role in enabling scalable, reliable, and efficient network management.[28]

2.3.1. Introduction to Python in Network Automation

Python has become an essential programming language in the field of network automation due to its simplicity, readability, and extensive ecosystem of libraries. Its clear syntax makes it accessible for both beginner and advanced network engineers, allowing them to automate repetitive tasks, manage device configurations, and collect network data efficiently. Python's versatility enables it to interact with a wide variety of network devices and platforms through APIs, SSH, and other protocols.

In recent years, the adoption of Python for network automation has grown rapidly, driven by the increasing complexity of network infrastructures and the need for scalable, reliable, and programmable solutions. Python scripts are now commonly used to automate configuration changes, monitor network health, and integrate with other automation tools, making Python a cornerstone of modern network engineering practices.[21]

2.3.2. Benefits of Python

- Easy-to-read and simple syntax, making it accessible for beginners (as noted in "Foundations of Python Network Programming").
- Rich set of libraries for network automation and communication, such as Paramiko and the built-in socket module.
- Supports rapid prototyping and quick development, helping engineers automate tasks efficiently.[19]

2.3.3. Challenge of using automation python

- **Dependency Management:** Overseeing Python environments and package versions can pose challenges, necessitating the use of tools such as Docker and Virtualenv to maintain consistency.
- **Device Compatibility:** Certain network devices do not support modern APIs, requiring tailored solutions like Netmiko or Scrapli.
- **Error Handling:** It is crucial to ensure robustness against timeouts, failures, and network interruptions, employing retries and exception handling techniques.
- **Security Risks:** Credentials must be securely managed using Ansible Vault or HashiCorp Vault to avert unauthorized access.[29]

2.3.4. Integrating Python with Ansible

Python's versatility makes it an ideal companion for other automation tools like Ansible. Python can be used to develop custom Ansible modules and plugins, extending Ansible's capabilities to support additional devices or workflows. Many Ansible modules for network automation are written in Python, leveraging its libraries for device communication and data processing. Integration between Python scripts and Ansible playbooks allows for advanced automation scenarios, such as dynamic inventory generation, complex data manipulation, and orchestration of multi-step processes. This synergy enables organizations to build end-to-end automation pipelines that are both powerful and adaptable.[30]

2.3.5. Comparison Between Ansible and Python

Both Ansible and Python are powerful tools for network automation, but they serve different purposes and excel in different scenarios. **Ansible** is ideal for configuration management, orchestration, and repetitive tasks across multiple devices, thanks to its agentless architecture and declarative playbooks. It is user-friendly and requires minimal programming knowledge, making it suitable for standard automation workflows. **Python**, on the other hand, offers greater flexibility and control for custom automation tasks, complex logic, and integration with diverse APIs. Python is preferred when building tailored solutions, performing data analysis, or automating non-standard processes. In practice, many organizations use both tools together-Ansible for broad orchestration and Python for specialized tasks-achieving a balance between simplicity and power.[31]

Table 2.2 - Comparison Between Ansible and Python.

Criterion	**Ansible**	**Python**
Ease of Use	Easy, does not require programming	Requires programming knowledge
Scalability	High	High with programming effort
Network Support	Strong via ready-made modules	Strong via multiple libraries
Customization	Relatively limited	Very high
Performance	Good for repetitive tasks	Faster for complex tasks

2.4. Conclusion

Network automation has become indispensable for modern IT environments, enabling organizations to manage complex infrastructures efficiently and reliably. Both Ansible and Python play crucial roles in this transformation: Ansible excels at orchestrating and standardizing repetitive tasks, while Python offers the flexibility needed for custom automation and integration. By leveraging these tools together, network engineers can build robust, scalable, and adaptable automation solutions that meet the demands of today's dynamic networks. The next chapter will focus on practical implementation, demonstrating how to apply these concepts in real-world scenarios.[32]

Chapter

3 Applied Study and Comparative Analysis of Ansible and Python for Network Automation

3.1 Introduction

In this chapter, we present the practical setup used to implement our network automation project. The focus is on the key tools, platforms, and procedures applied to build and manage an integrated network using automation techniques. The environment was based on the Ubuntu operating system, chosen for its robustness and widespread use in programming and network management. A virtualized lab was created using VMware Workstation Pro, and the network topology was simulated using GNS3 with Cisco devices. Once the environment was ready, the Ansible tool was installed as the primary automation solution, alongside Python and the Netmiko library, which were used for comparison. This chapter explains how each of these tools was configured and applied to automate different network tasks.[33]

3.2 Experiment Setup

The project was implemented using a personal computer equipped with an Intel Core i7 or AMD Ryzen processor, 16GB of RAM, at least 512GB of SSD storage, and a Full HD or higher resolution display. These specifications were chosen to ensure smooth performance when running virtualization tools and network simulations.

3.3 Installation of APT Packages

When we start on UBUNTU it is important to update and install the APT packages that will be useful in our future operations. Advanced Packaging Tool is a comprehensive and advanced package management system, allowing easy search and efficient, simple installation and clean uninstallation of software and utilities. It also makes it easier to update the Ubuntu distribution with packages in latest versions and to upgrade to a newer version of Ubuntu, when it is available.[34]

The commands to implement to load APT are as follows:

```
manar@manar-virtual-machine:~$ sudo apt-get install
```

Figure 3.1 – Installation of the APT package

```
manar@manar-virtual-machine:~$ sudo apt-get upgrade
```

Figure 3.2 – Update packages

```
manar@manar-virtual-machine:~$ sudo apt update
```

Figure 3.3 – system update

3.4 Installation de VMWare Workstation Pro

To install VMware Workstation Pro on Windows, you must first download the executable installation file from the official VMware website, such as version: VMware Workstation Pro 17.6.2. build 24409262 ,which is one of the latest versions in the 16 series.

Figure 3.4 – Installation de VMWare Workstation Pro

Once the installation is done, VMWare will be on the list of activities. Virtual machines can now be created in that environment. It is advisable to follow the standard setup to avoid making the installation more complicated.

To run our project, we will need to install a virtual machine: Ansible admin machine: as ansible is designed for Unix-like systems, we chose to operate under Ubuntu.

It is crucial to give the device a standardized user name for establishing connectivity.

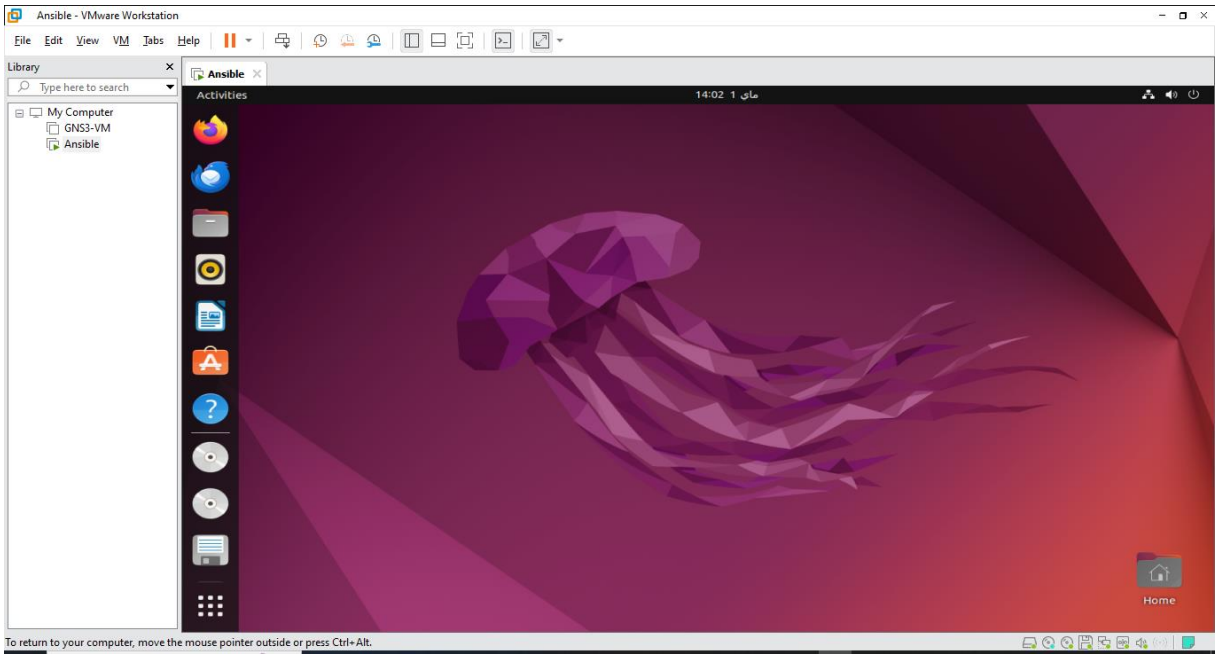

Figure 3.5 – Liste des VMs installées.

3.5 GNS3

GNS3 (Graphical Network Simulator-3) is a robust network simulator for creating networks without the need for actual hardware, and training for Cisco, MikroTik and other environments. [35]

3.5.1 Installation of GNS3

To install GNS3 on Windows:

- Go to the website: https://www.gns3.com
- Sign up or log in.
- After logging in, download the appropriate version for your operating system (typically GNS3 All-in-One for Windows).
- Download the GNS3-2.xx-all-in-one.exe setup file.

- After uploading, run the file and proceed with the installation process that includes installing the tools included.

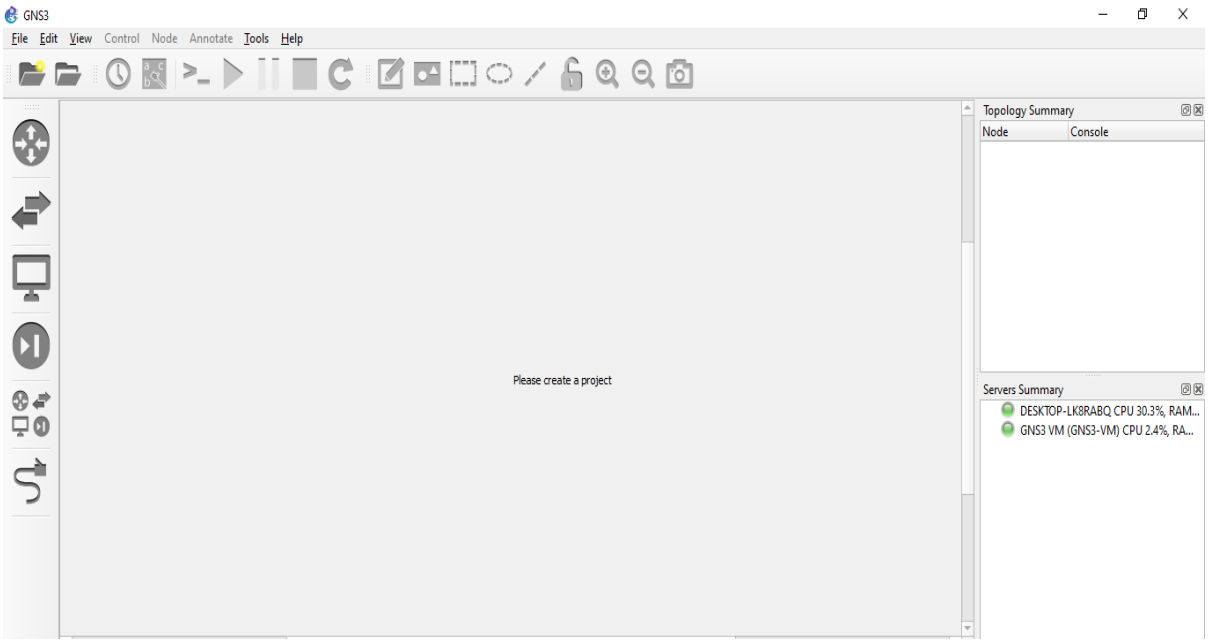

Figure 3.6 – Installation of GNS3

Once the installation is complete, we run GNS3 on our local machine to configure it and add the equipment we will need to realize our model; Cisco routers as well as ansible virtual machine

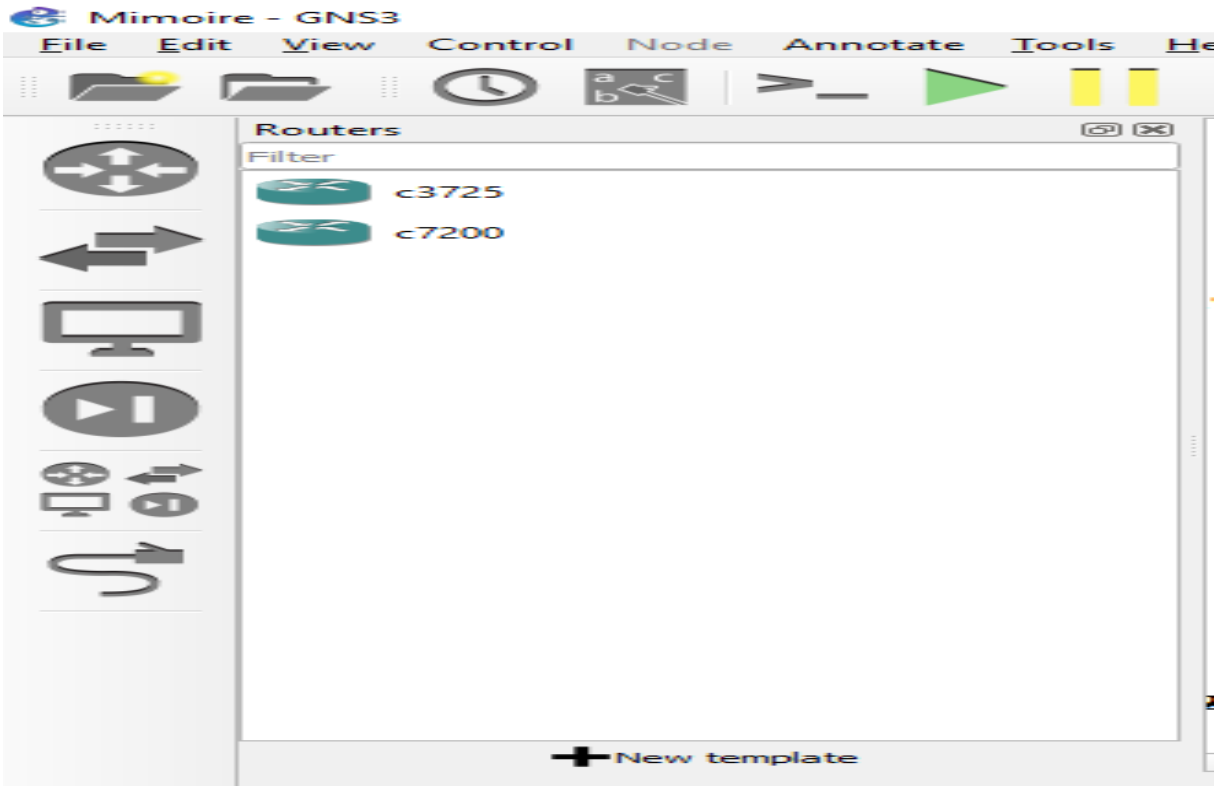

Figure 3.7 – List of downloaded routers.

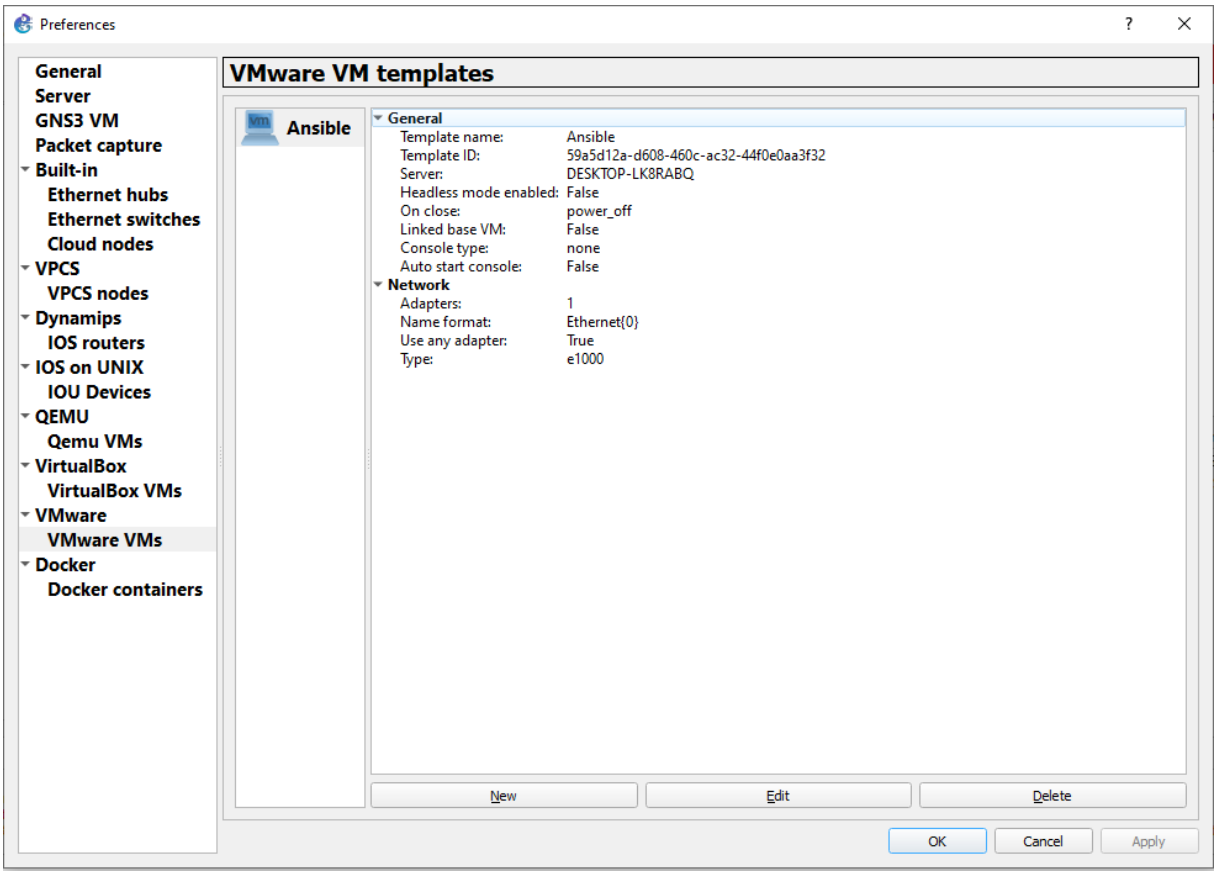

Figure 3.8 – List of virtual machines created in GNS3.

3.5.2. Implementation of the network architecture

Once our equipment is ready, we will implement this network architecture on our LAB, as shown in Fig 3.9

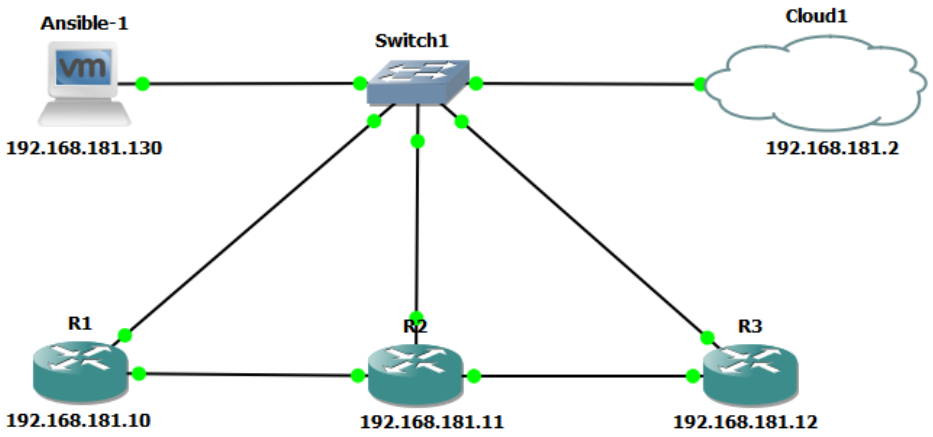

Figure 3.9 – topology of the network.

The presented topology illustrates a simulated network environment used for network automation tasks with Ansible. It includes a central switch (Switch1) that connects multiple devices. The Ansible control node, named Ansible-1, is a virtual machine assigned the IP address 192.168.181.130, and it is responsible for managing and automating network configurations. Three routers (R1, R2, and R3) are connected to the switch, each with the IP addresses 192.168.181.10, 192.168.181.11, and 192.168.181.12 respectively. Additionally, a Cloud1 node represents external network access with the IP 192.168.181.2. This setup allows the Ansible controller to communicate with the routers over SSH and perform configuration management tasks efficiently within a simulated lab environment.

3.5.3. Assigning IP addresses to equipment

Table 3.1 – IP addressing.

Name of the machine	Interface	IP address	Under mask network	Gateway
Ansible	/	192.168.181.130	255.255.255.0	192.168.181.2
R1	F0/0	192.168.181.10	255.255.255.0	/
R2	F0/0	192.168.181.11	255.255.255.0	/
R3	F0/0	192.168.181.12	255.255.255.0	/

3.5.4. Secure Shell

- **SSH on Cisco Equipment**
  - **Enable SSH on Cisco Routers**

Here we will configure SSH on our routers, for that we will follow the steps following:

- Rename our equipment
- Create user account with password
- Define a domain name
- Generate the «RSA» keys and choose the number of bits a 1024 (key length
- Define the SSH version we will use
- Configure VTY virtual line
- Use the user account we created
- Disable the Telnet protocol by enabling SSH

Configurations are shown in Fig 3.10 and must be performed on all network routers.

```
R1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#hostname R1
R1(config)#ip domain-name manar.local
R1(config)#crypto key generate rsa
The name for the keys will be: R1.manar.local
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
% Generating 1024 bit RSA keys, keys will be non-exportable...
*May  3 13:15:08.171: %SYS-3-CPUHOG: Task is running for (2020)msecs, more than (2000)msecs (0/0),process = crypto sw pk pro
c.
-Traceback= 0x63086DDCz 0x6392B88Cz 0x6395449Cz 0x63954B04z 0x63951FE0z 0x63952404z 0x63952640z 0x639536F0z 0x6307E640z 0x63
07E624z
*May  3 13:15:08.223: %SYS-3-CPUYLD: Task ran for (2072)msecs, more than (2000)msecs (0/0),process = crypto sw pk proc[OK]

R1(config)#
*May  3 13:15:16.843: %SSH-5-ENABLED: SSH 1.99 has been enabled
R1(config)#username ansible password ansible
R1(config)#line vty 0 4
R1(config-line)#login local
R1(config-line)#transport input ssh
R1(config-line)#exit
R1(config)#ip ssh version 2
R1(config)#exit
```

Figure 3.10 – Configuring the SSH protocol on routers.

3.6 Ansible installation

In order to perform an Ansible installation on our responsible VM, we will run The commands shown in Fig 3.11, 3.12 and 3.13.

- **Software Properties Common**

This software provides an abstraction layer over APT repositories, making it easier to manage software sources and vendor-specific packages. It simplifies the addition and removal of Personal Package Archives (PPAs).

```
manar@manar-virtual-machine:~$ sudo apt install software-properties-common
```

Figure 3.11 – Installation des propriétés du logiciel Commun.

- **Ansible tool installation**

```
manar@manar-virtual-machine:~$ sudo apt install ansible
```

Figure 3.12 – Complete installation of ansible.

– **Paramiko Installation**

Paramiko is a python interface (2.7et plus) that implements the SSH version2 protocol, providing both client and server functionality.

```
manar@manar-virtual-machine:~/ansible-project/playbook$ pip install paramiko
```

Figure 3.13 – Installation of paramiko.

– **Netmiko Installation**

A library built on Paramiko, but focused solely on making it easier to connect to network devices via SSH as a client only. It does not support acting as an SSH server, and is specifically designed to send commands to routers and switches and return results.

```
manar@manar-virtual-machine:~/python-projects$ pip install netmiko
```

Figure 3.14 – Installation of netmiko.

– **Verification of installation**

To check the installation of Ansible we execute the command illustrated by the figure III.27

```
manar@manar-virtual-machine:~$ ansible --version
ansible [core 2.17.11]
  config file = None
  configured module search path = ['/home/manar/.ansible/plugins/modules', '/usr
/share/ansible/plugins/modules']
  ansible python module location = /home/manar/.local/lib/python3.10/site-packag
es/ansible
  ansible collection location = /home/manar/.ansible/collections:/usr/share/ansi
ble/collections
  executable location = /home/manar/.local/bin/ansible
  python version = 3.10.12 (main, Feb  4 2025, 14:57:36) [GCC 11.4.0] (/usr/bin/
python3)
  jinja version = 3.0.3
  libyaml = True
```

Figure 3.15 – Verification of the Ansible installation

3.7 Base configuration with Ansible

– Ansible inventory file created

In the "ansible" directory, a "Playbook" subdirectory must be created, which will contain the file «hosts.yml» or our nodes will be registered as shown in Fig 3.14.

```
manar@manar-virtual-machine:~/ansible-project$ sudo mkdir playbook
mkdir: cannot create directory 'playbook': File exists
manar@manar-virtual-machine:~/ansible-project$ cd playbook
manar@manar-virtual-machine:~/ansible-project/playbook$ ls
acl.yaml  hosts.ini  ospf.yaml  rout1.yaml  rout2.yaml  router.yaml
manar@manar-virtual-machine:~/ansible-project/playbook$ sudo nano hosts.ini
manar@manar-virtual-machine:~/ansible-project/playbook$ cat hosts.ini
[routers]
192.168.181.10 ansible_user=ansible ansible_password=ansible ansible_network_os=ios ansible_connection=network_cli
192.168.181.11 ansible_user=ansible ansible_password=ansible ansible_network_os=ios ansible_connection=network_cli
192.168.181.12 ansible_user=ansible ansible_password=ansible ansible_network_os=ios ansible_connection=network_cli
manar@manar-virtual-machine:~/ansible-project/playbook$
```

Figure 3.16 – Inventory.

- The «mkdir» command is used to create a directory.
- The «ls» command is used to list the contents of the directory.
- The «nano» command is used to create or modify the contents of a file (. txt, .ini, .yml...)
- The «cat» command is used to display the contents of the file in the terminal.

3.7.1 Injection of configurations on network equipment

– Ansible Playbook Structure

A Playbook is structured as follows : A YAML file always starts with three dashes.

- **The first part of the Playbook contains:**

  - Name**:** The name of the Playbook.

  - Hosts: The target machines.

  - Gather_facts: The "facts' module is called by the Playbook to gather the useful variables on remote nodes. The boolean value «true» is assigned when we want Ansible to automatically execute tasks. long as the boolean value «false» is used when we want to add the modules manually.

- **The second part deals with "to-dos". Indicates various tasks**

  We are working on for the contract. Each task includes:

  - Name: This is the name of the task

  - The unit in question, for example "ios_config"

3.7.2 Creation of the playbook

a. Playbook has several spots

– Creation of the playbook

As shown in Fig 3.17 we have run three tasks in a single Playbook. Here is the meaning of each task:

- Task 1: Test the reachability of a host using the Ansible module "ios_ping".
- Task 2: Configuring the machine names on using Ansible module "ios_config".
- Task 3: Execute a command to display interfaces and their evaluations in using an Algorithm. The Ansible module used is "ios_command".

```
GNU nano 6.2                                    router.yaml
---
- name: Configure Routers R1, R2, and R3
  hosts: all
  gather_facts: no
  tasks:
    - name: Configure Router Interfaces
      ios_config:
        lines:
          - ip address {{ ip_address }} 255.255.255.0
          - no shutdown
        parents: interface FastEthernet0/0
      vars:
        ip_address: "{{ hostvars[inventory_hostname]['ansible_host'] }}"
    - name: Run multiple commands and evaluate the output
      ios_command:
        commands:
          - show ip interface brief
          - show running-config
      register: command_output_raw

    - name: Store command output in a fact
      set_fact:
        command_output: "{{ command_output_raw.stdout_lines }}"
```

Figure 3.17 – Playbook with multiple tasks

– Playbook execution and output

We inject the "router.yaml" playbook into a group of nodes located in the "hosts.ini" inventory file using the "ansible-playbook" command as shown in Fig 3.18.

```
manar@manar-virtual-machine:~/ansible-project/playbook$ ansible-playbook router.yaml -i hosts.ini
```

Figure 3.18 – Playbook run command

The results of this automation appear a few seconds after running the command in the Ansible machine terminal, as shown in the figure. The tasks were executed on all routers (R1, R2, and R3). The first task configures the router interfaces using the `ios_config` module, assigning an IP address dynamically using the value defined in the inventory file (`ansible_host`), with a subnet mask of `255.255.255.0`, and enabling the interface `FastEthernet0/0`.

The second task uses the `ios_command` module to run multiple commands on the routers—specifically `show ip interface brief` and `show running-config`. These outputs are stored temporarily in a variable called `command_output_raw`.

Finally, the third task stores the output in a fact variable called `command_output` using the `set_fact` module, allowing the output to be reused or processed later in the playbook. the playbook focuses on interface configuration and collecting basic command outputs.

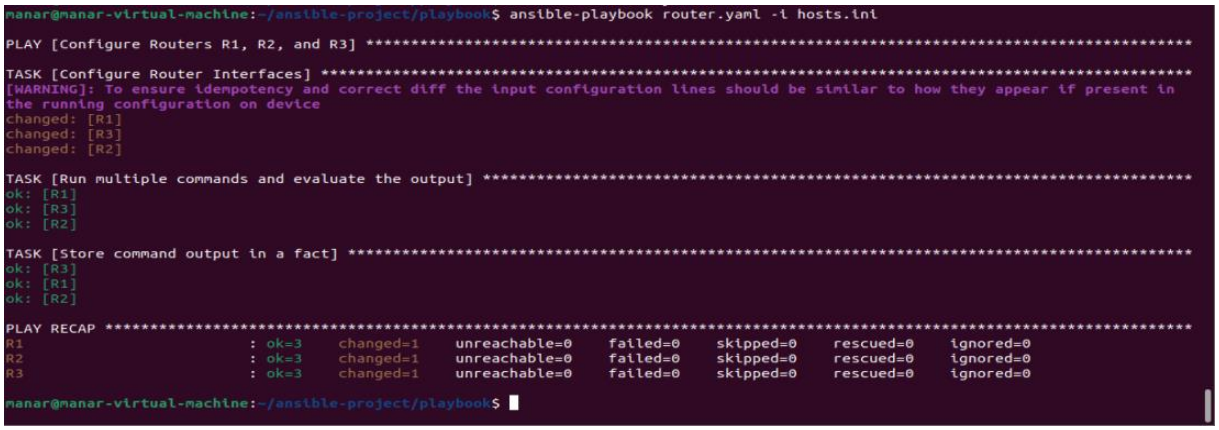

Figure 3.19 – Playbook result.

At the end of the Ansible Playbook displays a "Play Summary" which shows the number of tasks that were executed, the number of tasks that made a change as well as the number of tasks that returned one or more errors.

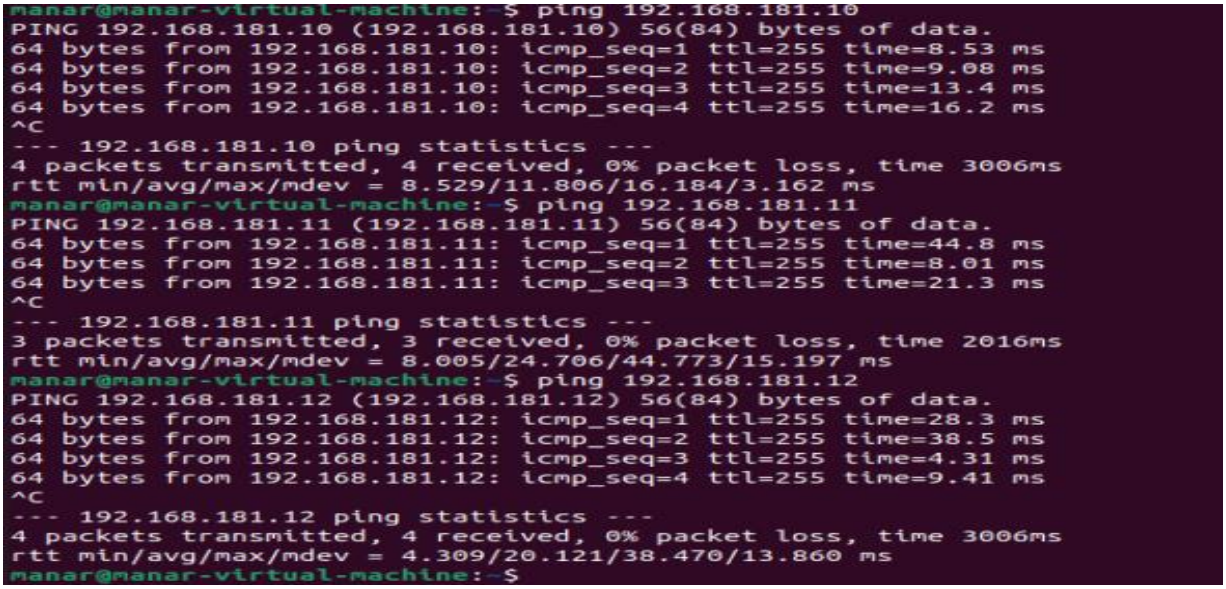

Figure 3.20 – Result of configurations on routers.

3.7.3 Static routing playbook

a. Playbook creation

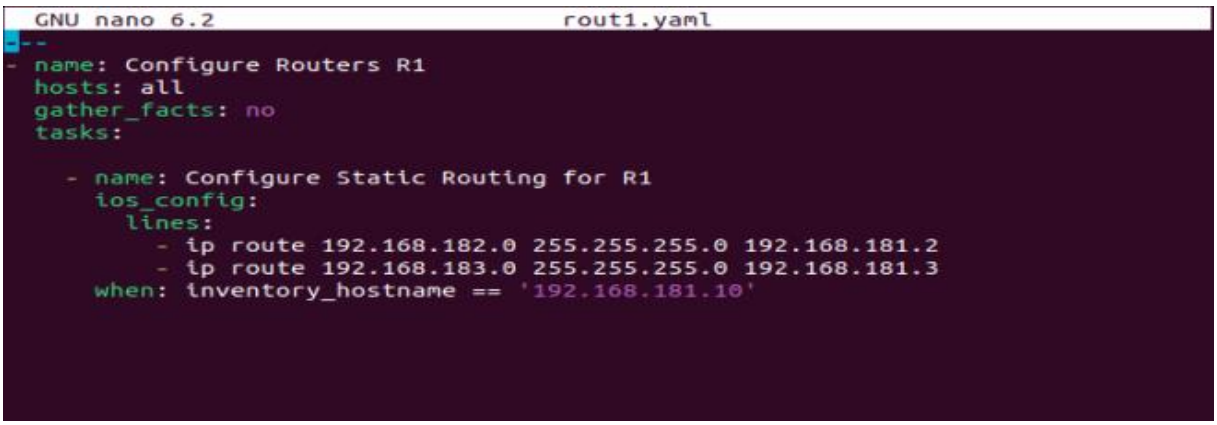

Figure 3.21 – Static routing playbook for R1.

- To configure the Static Routing on routers using Ansible, the module «ios_config» is used. This creates a playbook for each router.
- Fig 3.21 shows the «rout1.yaml» playbook that was injected into the R1 router to assign it a static routing.
- Fig 3.22 shows the "rout2.yaml" playbook that was injected on the R2 router:

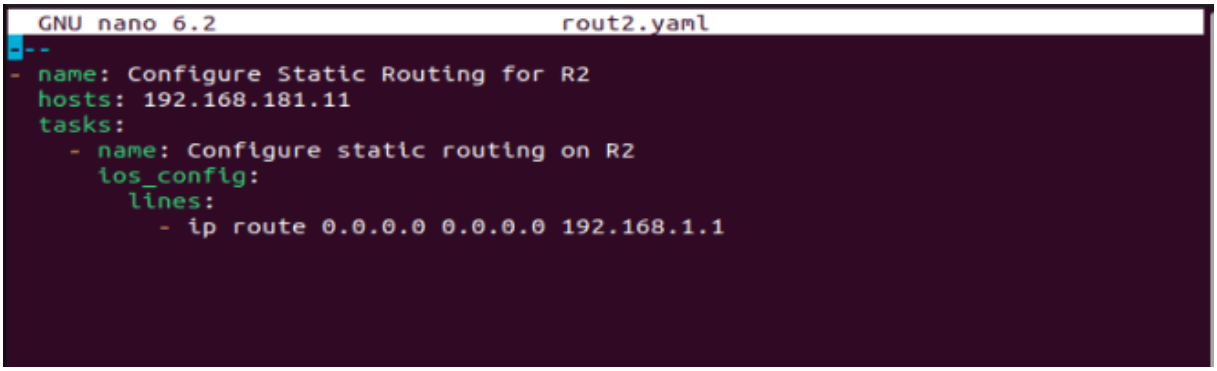

Figure 3.22 – Static routing playbook for R2.

b. Playbook execution and output

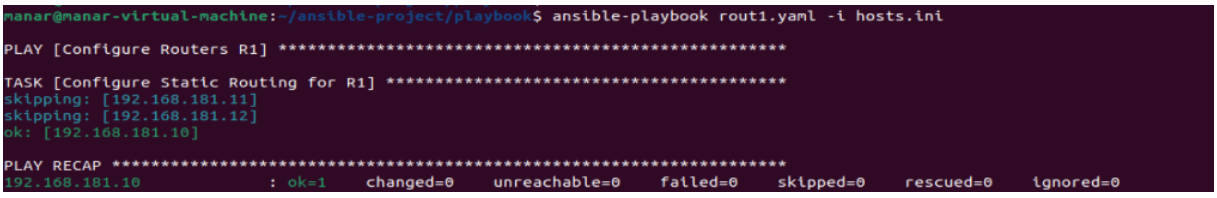

Figure 3.23 – Playbook result on R1.

Fig 3.23 shows us the execution of the playbook «rout1.yml» on our router 1 and the result that indicates that it was executed successfully and that Ansible made a configuration change for our router (changed =1).

Fig 3.24 shows the result of running the playbook on R2. Static routes have been configured successfully:

```
manar@manar-virtual-machine:~/ansible-project/playbook$ ansible-playbook rout2.yaml -i hosts.ini

PLAY [Configure Static Routing for R2] *****************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************
ok: [192.168.181.11]

TASK [Configure static routing on R2] ******************************************************************************************
[WARNING]: To ensure idempotency and correct diff the input configuration lines should be similar to how they appear if present in
the running configuration on device
changed: [192.168.181.11]

PLAY RECAP *********************************************************************************************************************
192.168.181.11             : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Figure 3.24 – Playbook result on R2.

3.7.4 Playbook ACL

a. Creation du playbook

```
 GNU nano 6.2                                    acl.yaml *
---
- name: Configure ACL on Routers
  hosts: routers
  gather_facts: no
  tasks:
    - name: Apply ACL Configuration
      cisco.ios.ios_config:
        lines:
          - ip access-list standard MY_ACL
          - permit 192.168.181.0 0.0.0.255
          - deny any
        parents:
          - interface FastEthernet0/0
          - ip access-group MY_ACL in
      register: acl_output

    - name: Display ACL Configuration Result
      debug:
        var: acl_output
```

Figure 3.25 – ACL Playbook.

To configure ACL on routers using Ansible We created a standard ACL named MY_ACL that allows traffic coming from network 192.168.181.0/24 only, and denies any other traffic (deny any). This ACL is then connected to interface FastEthernet0/0 to apply filtering to the incoming data. The result of the configuration is then logged and displayed using the debug task.

b. Playbook execution and result

```
manar@manar-virtual-machine:~/ansible-project/playbook$ ansible-playbook -i hosts.ini acl.yaml

PLAY [Configure ACL on Routers] ***********************************************

TASK [Apply ACL Configuration] ************************************************
[WARNING]: To ensure idempotency and correct diff the input configuration lines
should be similar to how they appear if present in the running configuration on
device
changed: [192.168.181.12]
changed: [192.168.181.11]
changed: [192.168.181.10]

TASK [Display ACL Configuration Result] ***************************************
ok: [192.168.181.10] => {
    "acl_output": {
        "banners": {},
        "changed": true,
        "commands": [
            "interface FastEthernet0/0",
            "ip access-group MY_ACL in",
            "ip access-list standard MY_ACL",
            "permit 192.168.181.0 0.0.0.255",
            "deny any"
        ],
        "failed": false,
        "updates": [
            "interface FastEthernet0/0",
            "ip access-group MY_ACL in",
            "ip access-list standard MY_ACL",
            "permit 192.168.181.0 0.0.0.255",
            "deny any"
        ],
        "warnings": [
            "To ensure idempotency and correct diff the input configuration lines should be similar to how they appe
ar if present in the running configuration on device"
        ]
    }
}
ok: [192.168.181.11] => {
    "acl_output": {
        "banners": {},
        "changed": true,
        "commands": [
            "interface FastEthernet0/0",
            "ip access-group MY_ACL in",
            "ip access-list standard MY_ACL",
            "permit 192.168.181.0 0.0.0.255",
            "deny any"
        ],
        "failed": false,
        "updates": [
            "interface FastEthernet0/0",
            "ip access-group MY_ACL in",
            "ip access-list standard MY_ACL",
            "permit 192.168.181.0 0.0.0.255",
            "deny any"
        ],
        "warnings": [
            "To ensure idempotency and correct diff the input configuration lines should be similar to how
they appear if present in the running configuration on device"
        ]
    }
}
ok: [192.168.181.12] => {
    "acl_output": {
        "banners": {},
        "changed": true,
        "commands": [
            "interface FastEthernet0/0",
            "ip access-group MY_ACL in",
            "ip access-list standard MY_ACL",
            "permit 192.168.181.0 0.0.0.255",
            "deny any"
        ],
        "failed": false,
        "updates": [
            "interface FastEthernet0/0",
            "ip access-group MY_ACL in",
            "ip access-list standard MY_ACL",
            "permit 192.168.181.0 0.0.0.255",
            "deny any"
        ],
        "warnings": [
            "To ensure idempotency and correct diff the input configuration lines should be similar to
how they appear if present in the running configuration on device"
        ]
    }
}

PLAY RECAP ********************************************************************
192.168.181.10             : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0
  ignored=0
192.168.181.11             : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0
  ignored=0
192.168.181.12             : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0
  ignored=0
```

Figure 3.26 – Result of ACL playbook

Figure 3.26 shows the Ansible command and the result obtained where two Tasks: Apply ACL Configuration and Display ACL Configuration Result were executed and the ACL was successfully configured on all routers.

3.7.5 Playbook OSPF

a. Creation du playbook

```
  GNU nano 6.2                                          ospf.yaml
- name: Configure OSPF on Routers
  hosts: routers
  gather_facts: no
  tasks:
    - name: Configure OSPF
      cisco.ios.ios_config:
        lines:
          - router ospf 1
          - network 192.168.181.0 0.0.0.255 area 0
        parents:
          - router ospf 1
```

Figure 3.27 – OSPF Playbook.

To configure OSPF on routers using Ansible, we activate process 1, then add network 192.168.181.0/24 to zone 0. This configuration is performed using the cisco.ios_config module, enabling the automation of dynamic routing setup between Cisco devices.

b. Playbook execution and result

```
manar@manar-virtual-machine:~/ansible-project/playbook$ ansible-playbook -i hosts.inl ospf.yaml

PLAY [Configure OSPF on Routers] ***********************************************************

TASK [Configure OSPF] **********************************************************************
[WARNING]: To ensure idempotency and correct diff the input configuration lines should be similar to
how they appear if present in the running configuration on device
changed: [192.168.181.11]
changed: [192.168.181.12]
changed: [192.168.181.10]

PLAY RECAP *********************************************************************************
192.168.181.10             : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0
  ignored=0
192.168.181.11             : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0
  ignored=0
192.168.181.12             : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0
  ignored=0
```

Figure 3.28 – Result of OSPF playbook

Figure 3.28 shows the playbook for configuring OSPF using Ansible running successfully on all three routers, with the "Configure OSPF" task successfully executed on each machine, and one change logged for each (changed=1)

3.8 Base configuration with python

3.8.1 Creation of the playbook

a. Playbook creation

```
  GNU nano 6.2                                    router.py *
from netmiko import ConnectHandler
routers = [
    {"device_type": "cisco_ios", "ip": "192.168.181.10", "username": "ansible", "password": "ansible"},
    {"device_type": "cisco_ios", "ip": "192.168.181.11", "username": "ansible", "password": "ansible"},
    {"device_type": "cisco_ios", "ip": "192.168.181.12", "username": "ansible", "password": "ansible"},
]
def summarize_router(router):
    print(f"\nConnecting to {router['ip']}...")
    conn = ConnectHandler(**router)

    config_cmds = [
        "interface FastEthernet0/0",
        f"ip address {router['ip']} 255.255.255.0",
        "no shutdown"
    ]
    conn.send_config_set(config_cmds)

    hostname = conn.send_command("show run | include hostname").strip()

    iface_output = conn.send_command("show ip interface brief")
    fa0 = next((line for line in iface_output.splitlines() if "FastEthernet0/0" in line), "Not found")

    running_config = conn.send_command("show run").splitlines()
    config_preview = "\n".join(running_config[:5])

    print(f"\nRouter {hostname} ({router['ip']}):")
    print(f"  Interface Fa0/0: {fa0}")
    print("  Config Preview:")
    print("  " + "\n  ".join(config_preview.splitlines()))

    conn.disconnect()

for router in routers:
    summarize_router(router)
```

Figure 3.29 – Playbook

Figure 3.29 we use the Netmiko library to connect to the three Cisco IOS routers via SSH. For each router, we configure an IP address on interface FastEthernet0/0, display brief information about the interface using the show ip interface brief command, and display a portion of the running-config with the hostname.

b. Playbook execution

```
manar@manar-virtual-machine:~/python-projects$ python3 router.py

Connecting to 192.168.181.10...

Router hostname R1 (192.168.181.10):
  Interface Fa0/0: FastEthernet0/0       192.168.181.10  YES NVRAM  up
           up
  Config Preview:
  Building configuration...

  Current configuration : 1157 bytes
  !
  ! Last configuration change at 14:48:32 UTC Sun May 25 2025 by ansible

Connecting to 192.168.181.11...

Router hostname R2 (192.168.181.11):
  Interface Fa0/0: FastEthernet0/0       192.168.181.11  YES NVRAM  up
           up
  Config Preview:
  Building configuration...

  Current configuration : 1121 bytes
  !
  ! Last configuration change at 14:48:36 UTC Sun May 25 2025 by ansible

Connecting to 192.168.181.12...

Router hostname R3 (192.168.181.12):
  Interface Fa0/0: FastEthernet0/0       192.168.181.12  YES NVRAM  up
           up
  Config Preview:
  Building configuration...

  Current configuration : 1055 bytes
  !
  ! Last configuration change at 14:48:40 UTC Sun May 25 2025 by ansible
```

Figure 3.30 – Playbook result

Figure 3.30 shows the successful execution of the commands by ansible, and the successful connection to three routers (R1, R2, and R3) via SSH using Netmiko. For each router, its hostname was displayed, and it was confirmed that the FastEthernet0/0 interface is up/up and has the correct IP address. Next, the running-config pane was displayed, showing the size of the settings and the date of the last modification.

3.8.2 Static routing playbook

a. Playbook creation

```
 GNU nano 6.2                                    static-route.py
from netmiko import ConnectHandler, NetmikoTimeoutException, NetmikoAuthenticationException

routers = [
    {'ip': '192.168.181.10', 'device_type': 'cisco_ios', 'username': 'ansible', 'password': 'ansible'},
    {'ip': '192.168.181.11', 'device_type': 'cisco_ios', 'username': 'ansible', 'password': 'ansible'},
    {'ip': '192.168.181.12', 'device_type': 'cisco_ios', 'username': 'ansible', 'password': 'ansible'}
]

static_routes = [
    {'network': '192.168.182.0', 'mask': '255.255.255.0', 'next_hop': '192.168.181.1'},
    {'network': '192.168.183.0', 'mask': '255.255.255.0', 'next_hop': '192.168.181.2'}
]

for router in routers:
    print(f"🚀 Connecting to {router['ip']}...")
    try:
        connection = ConnectHandler(**router)
        print("✅ Connected successfully!")

        print("✅ Configuring Static Routes...")
        commands = [f"ip route {route['network']} {route['mask']} {route['next_hop']}" for route in static_routes]
        connection.send_config_set(commands)

        print("✅ Verifying Static Routes...")
        output = connection.send_command("show ip route static")
        print(f"Verification Output:\n{output}")

        connection.disconnect()
        print("✅ Connection closed.\n")

    except (NetmikoTimeoutException, NetmikoAuthenticationException) as e:
        print(f"❌ Failed to connect to {router['ip']}. Error: {e}")
```

Figure 3.31 – Static routing playbook

In Figure 3.31, each router is connected using SSH, and then sends commands to add two fixed paths to the networks 192.168.182.0/24 and 192.168.183.0/24 via the appropriate gateways. After configuration, the script executes the show ip route static command to verify that the routing was successful, and displays the results.

b. Playbook execution and result

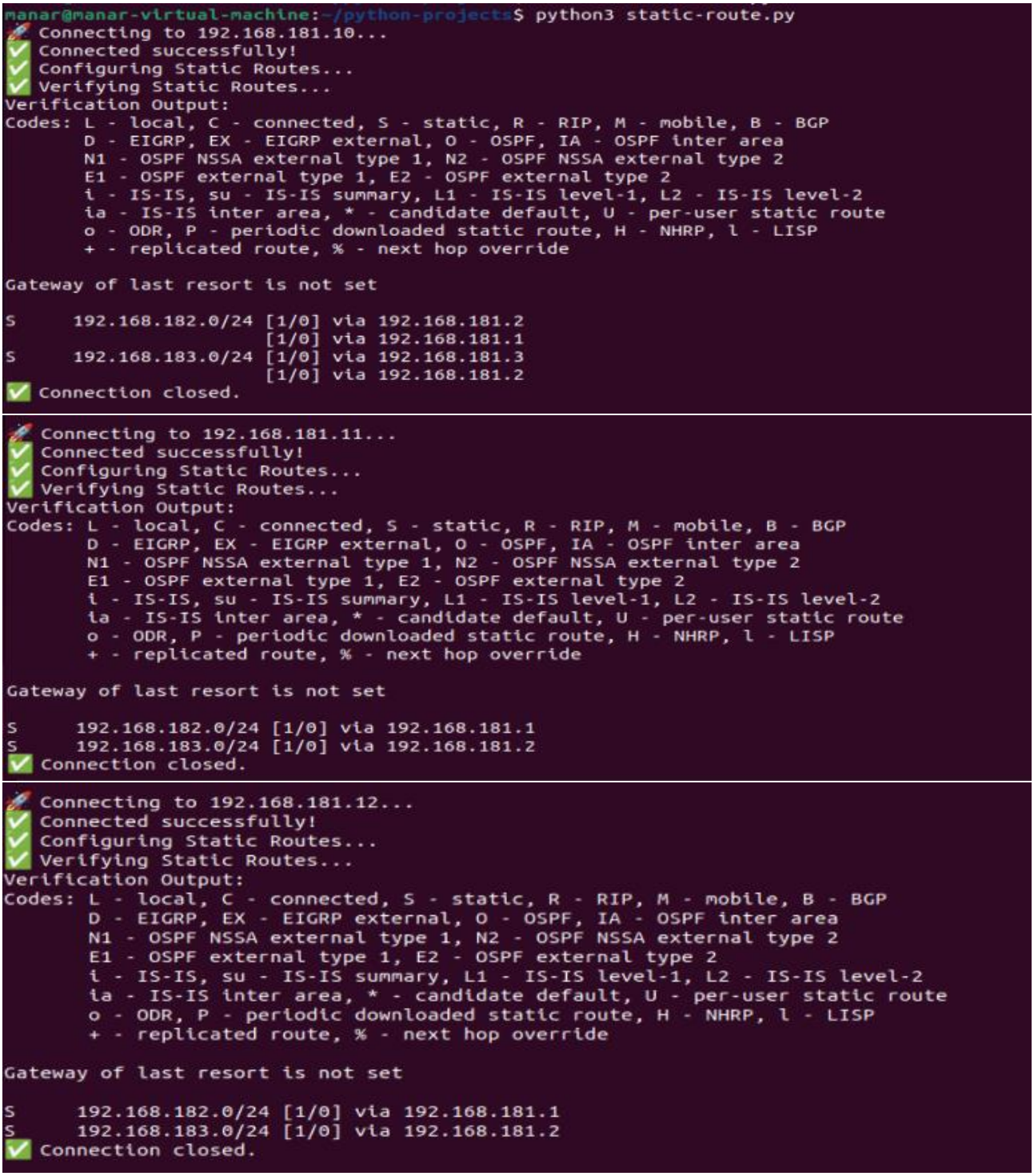

Figure 3.32 – Playbook result

Figure 3.32 shows in the results that the static-route.py setting was successfully executed on the three routers, where each router was connected via SSH, and the fixed paths were successfully configured.

3.8.3 Playbook ACL

a. Playbook creation

```
  GNU nano 6.2                                    acl.py
from netmiko import ConnectHandler, NetmikoTimeoutException, NetmikoAuthenticationException

routers = [
    {'ip': '192.168.181.10', 'device_type': 'cisco_ios', 'username': 'ansible', 'password': 'ansible'},
    {'ip': '192.168.181.11', 'device_type': 'cisco_ios', 'username': 'ansible', 'password': 'ansible'},
    {'ip': '192.168.181.12', 'device_type': 'cisco_ios', 'username': 'ansible', 'password': 'ansible'}
]

acl_number = 100
acl_rules = [
    'permit ip any any',
    'deny icmp any any'
]

for router in routers:
    print(f"🚀 Connecting to {router['ip']}...")
    try:
        connection = ConnectHandler(**router)
        print("✅ Connected successfully!")

        print("✅ Configuring ACL...")
        commands = [f"access-list {acl_number} {rule}" for rule in acl_rules]
        connection.send_config_set(commands)

        print("✅ Verifying ACL configuration...")
        output = connection.send_command(f"show access-lists {acl_number}")
        print(f"Verification Output:\n{output}")

        connection.disconnect()
        print("✅ Connection closed.\n")

    except (NetmikoTimeoutException, NetmikoAuthenticationException) as e:
        print(f"❌ Failed to connect to {router['ip']}. Error: {e}")
```

Figure 3.33 – ACL Playbook.

In Figure 3.33 an access control list (ACL) 100 is set up. First, we try to connect to each router, and then we send commands to apply the ACL rules: Allow all IP traffic, and block ICMP. Second, a command is executed to check the ACL implementation using show access-lists. Finally, if the connection is successful, a confirmation message is printed, and if the connection fails for any reason, the appropriate error is displayed.

b. Playbook execution and result

```
manar@manar-virtual-machine:~/python-projects$ python3 acl.py
🚀 Connecting to 192.168.181.10...
✅ Connected successfully!
✅ Configuring ACL...
✅ Verifying ACL configuration...
Verification Output:
Extended IP access list 100
    10 permit ip any any
    20 deny icmp any any
✅ Connection closed.

🚀 Connecting to 192.168.181.11...
✅ Connected successfully!
✅ Configuring ACL...
✅ Verifying ACL configuration...
Verification Output:
Extended IP access list 100
    10 permit ip any any
    20 deny icmp any any
✅ Connection closed.

🚀 Connecting to 192.168.181.12...
✅ Connected successfully!
✅ Configuring ACL...
✅ Verifying ACL configuration...
Verification Output:
Extended IP access list 100
    10 permit ip any any
    20 deny icmp any any
✅ Connection closed.
```

Figure 3.34 – Playbook result

Figure 3.33 Each router was successfully connected, and ACL 100 was applied, allowing all IP traffic (permit ip any any) and blocking ICMP (deny icmp any any). The output of the show access-lists 100 command was displayed.

3.8.4 Playbook OSPF

a. Playbook creation

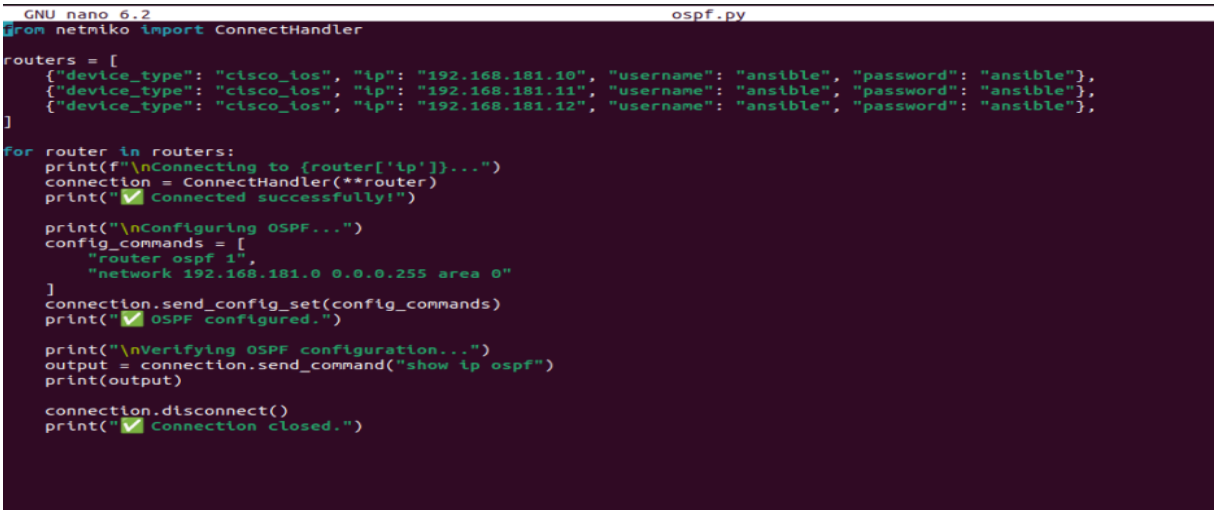

Figure 3.35 – OSPF Playbook.

To configure OSPF on routers using Ansible, we connect to three Cisco routers using the Netmiko library, configure the OSPF routing protocol on each router under process 1, and include network 192.168.181.0/24 under area 0. Next, we verify that OSPF is enabled using the show ip ospf command and it displays the result. The connection is disconnected after each router is finished.

b. Playbook execution and result

```
manar@manar-virtual-machine:~/python-projects$ python3 rip.py
🚀 Connecting to 192.168.181.10...
✅ Connected successfully!
✅ RIP configured.
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#router rip
R1(config-router)#version 2
R1(config-router)#network 192.168.181.0
R1(config-router)#no auto-summary
R1(config-router)#end
R1#
Verification Output:
*** IP Routing is NSF aware ***

Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 192.168.181.10
  Number of areas in this router is 1. 1 normal 0 stub 0 nssa
  Maximum path: 4
  Routing for Networks:
    192.168.181.0 0.0.0.255 area 0
  Routing Information Sources:
    Gateway         Distance      Last Update
  Distance: (default is 110)

Routing Protocol is "rip"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Sending updates every 30 seconds, next due in 0 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Redistributing: rip
  Default version control: send version 2, receive version 2
    Interface             Send  Recv  Triggered RIP  Key-chain
    FastEthernet0/0       2     2
  Automatic network summarization is not in effect
  Maximum path: 4
  Routing for Networks:
    192.168.181.0
  Routing Information Sources:
    Gateway         Distance      Last Update
  Distance: (default is 120)

✅ Connection closed.

🚀 Connecting to 192.168.181.11...
✅ Connected successfully!
✅ RIP configured.
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#router rip
R2(config-router)#version 2
R2(config-router)#network 192.168.181.0
R2(config-router)#no auto-summary
R2(config-router)#end
R2#
Verification Output:
*** IP Routing is NSF aware ***

Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 192.168.181.11
  Number of areas in this router is 1. 1 normal 0 stub 0 nssa
  Maximum path: 4
  Routing for Networks:
    192.168.181.0 0.0.0.255 area 0
  Routing Information Sources:
    Gateway         Distance      Last Update
  Distance: (default is 110)

Routing Protocol is "rip"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Sending updates every 30 seconds, next due in 0 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
```

```
  Redistributing: rip
  Default version control: send version 2, receive version 2
    Interface             Send  Recv  Triggered RIP  Key-chain
    FastEthernet0/0       2     2
  Automatic network summarization is not in effect
  Maximum path: 4
  Routing for Networks:
    192.168.181.0
  Routing Information Sources:
    Gateway         Distance      Last Update
  Distance: (default is 120)

✅ Connection closed.

🚀 Connecting to 192.168.181.12...
✅ Connected successfully!
✅ RIP configured.
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#router rip
R3(config-router)#version 2
R3(config-router)#network 192.168.181.0
R3(config-router)#no auto-summary
R3(config-router)#end
R3#
Verification Output:
*** IP Routing is NSF aware ***

Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 192.168.181.12
  Number of areas in this router is 1. 1 normal 0 stub 0 nssa
  Maximum path: 4
  Routing for Networks:
    192.168.181.0 0.0.0.255 area 0
  Routing Information Sources:
    Gateway         Distance      Last Update
  Distance: (default is 110)
```

```
✅ Connection closed.

🚀 Connecting to 192.168.181.11...
✅ Connected successfully!
✅ RIP configured.
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#router rip
R2(config-router)#version 2
R2(config-router)#network 192.168.181.0
R2(config-router)#no auto-summary
R2(config-router)#end
R2#
Verification Output:
*** IP Routing is NSF aware ***

Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 192.168.181.11
  Number of areas in this router is 1. 1 normal 0 stub 0 nssa
  Maximum path: 4
  Routing for Networks:
    192.168.181.0 0.0.0.255 area 0
  Routing Information Sources:
    Gateway         Distance      Last Update
  Distance: (default is 110)

Routing Protocol is "rip"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Sending updates every 30 seconds, next due in 0 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Redistributing: rip
  Default version control: send version 2, receive version 2
    Interface             Send  Recv  Triggered RIP  Key-chain
    FastEthernet0/0       2     2
  Automatic network summarization is not in effect
  Maximum path: 4
```

```
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#router rip
R3(config-router)#version 2
R3(config-router)#network 192.168.181.0
R3(config-router)#no auto-summary
R3(config-router)#end
R3#
Verification Output:
*** IP Routing is NSF aware ***

Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 192.168.181.12
  Number of areas in this router is 1. 1 normal 0 stub 0 nssa
  Maximum path: 4
  Routing for Networks:
    192.168.181.0 0.0.0.255 area 0
  Routing Information Sources:
    Gateway         Distance      Last Update
  Distance: (default is 110)

Routing Protocol is "rip"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Sending updates every 30 seconds, next due in 0 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Redistributing: rip
  Default version control: send version 2, receive version 2
    Interface             Send  Recv  Triggered RIP  Key-chain
    FastEthernet0/0       2     2
  Automatic network summarization is not in effect
  Maximum path: 4
  Routing for Networks:
    192.168.181.0
  Routing Information Sources:
    Gateway         Distance      Last Update
  Distance: (default is 120)

✅ Connection closed.
```

Figure 3.36 – Playbook result

Figure 3.36 shows the successful connection to each router, confirming that the OSPF protocol has been successfully configured. After configuration, the output of the show ip ospf command is displayed, which includes information about the OSPF process such as the process number, Router ID, OSPF status, and the number of participating interfaces.

3.9 The difference between Ansible and Python network setup

3.9.1 Execution style

Table 3.2 – Comparison of Execution Styles between Ansible and Python.

The characteristic	Ansible	Python
The language of execution	YAML + SSH Automation	Python Code
How to connect	Via SSH using Playbooks Direct connection	via Netmiko library
No coding required	Yes, uses YAML files	No, relies on explicit Python programming

The table 3.2 shows that Ansible relies on YAML files to automatically execute tasks over SSH using Playbooks, making it easy to use and not requiring programming expertise. In contrast, Python relies on explicit programming using the Netmiko library, which provides greater flexibility but requires writing the code manually.

3.9.2 Easy to use and maintain

Table 3.3 – Ease of Use and Maintenance: Ansible vs Python.

The characteristic	Ansible	Python
Ease of typing commands	Simpler using YAML files	requires a good knowledge of Python
Manage multiple devices	Very easy using inventory files	must iterate or use Python loops
Scalability	Excellent (works easily with hundreds of devices)	More complex if the number of devices increases

The table 3.3 shows that Ansible is simple to write commands through YAML files, a large number of devices are easily managed using inventory files, and Ansible has excellent scalability even with hundreds of devices. In contrast, Python requires good programming knowledge, and needs to use loops to manage multiple devices.

3.9.3 Verification and output

Table 3.4 – Verification Methods and Output: Ansible vs Python.

The characteristic	Ansible	Python
Check Settings	Using modules such as ios_ping or ios_command	Via send_command and manually analyse the output
Reports and output	Organised and easy to save as an HTML file	Requires the results to be formatted programmatically

The table 3.4 shows that Ansible provides ready-made modules like ios_ping and ios_command to easily check the status of devices, and its results are organized and saveable in formats like HTML.In contrast, Python relies on executing commands using send_command, and requires analyzing the results manually or programmatically, and outputting data requires custom formatting within the code.

3.9.4 Static Routing setup

Table 3.5 – Static Routing Configuration: Ansible vs Python.

The characteristic	Ansible	Python
Command example	ip route 192.168.168.2.0 255.255.255.255.0 ...	send_config_set([...])
Configuration method	Easy using loop in playbook	Requires a loop in a Python script
Verify	using show ip route	using send_command('show ip route')

Static Routing setup with Ansible is based on YAML files in which networks and destinations are precisely defined, making deployment automated and easy to replicate on multiple routers, minimising human error. In contrast, using Python with Netmiko, routing commands are sent manually via a script, providing greater flexibility in customising commands and handling special cases during execution.

3.9.5 ACL setup

Table 3.6 – ACL Configuration: Ansible vs Python.

The characteristic	Ansible	Python
command	ios_config: With lines:	send_config_set([...])
How it works	Written in a YAML file and sent in one go	It is executed as a scripted command string
Verify	using show access-lists	using send_command('show access-lists')

Setting up Access Control Lists (ACLs) with Ansible is done in a structured way through Playbook files, making it easy to apply the same ACL rules to multiple machines quickly and efficiently, with automatic verification of the results. Using Python with Netmiko, setup is done via a script that includes ACL commands, providing granular and interactive control when special cases need to be handled or custom verification needs to be performed.

3.9.6 OSPF setup

Table 3.7 – OSPF Configuration: Ansible vs Python.

The characteristic	Ansible	Python
command	router ospf 1, network ...	send_config_set([...])
Ease	Very easy when using Jinja2 variables	Requires a thorough knowledge of the software syntax
Verify	show ip ospf using the ios_command module	send_command('show ip ospf')

Setting up OSPF using Ansible is easier and more organised, as YAML files are used to define settings in a clear and reusable format, and the setup can be applied to multiple devices at once without the need for detailed programming. Using Python with the Netmiko library requires writing a script that contains the configuration commands for each router individually, providing greater flexibility but with increased complexity and execution time.

3.10 Conclusion

In conclusion, the main goal of this chapter has been successfully achieved, demonstrating that it is possible to perform configurations and manage the network using Ansible and Python automation tools. We have detailed and explained the different instruction sets, as well as the two methods that Ansible uses to execute tasks: Playbooks and Ad-hoc Commands, and we used Python's Netmiko library to accomplish the same tasks The comparative analysis indicates that Ansible is more user-friendly and better suited for managing large-scale infrastructures due to its YAML-based declarative syntax and inventory system. Python, while offering greater flexibility and customization through scripting, requires more advanced programming skills and manual handling. Both tools are powerful and complementary, and their use depends on the complexity and requirements of the network environment.

General Conclusion

This thesis has aimed to demonstrate the growing importance of automation in modern network environments by designing and implementing a simulated infrastructure managed through a centralized control system. The proposed model allowed for an in-depth exploration of automation tools, particularly Ansible and Python, and their impact on the simplification of complex configuration tasks and the overall enhancement of network management efficiency.

Ansible's declarative approach, based on intuitive YAML syntax, enables even non-expert users to execute robust automation procedures while maintaining strong capabilities in configuration management, system orchestration, and task distribution. Its agentless nature and ease of integration into diverse network environments make it a highly accessible yet powerful solution.

Through experimental simulations using platforms such as GNS3 and VMware, we have demonstrated that automation using Ansible can significantly reduce time and effort required for routine administrative operations. The results highlight improved accuracy in task execution and a substantial decrease in configuration errors—factors that contribute to increased network reliability and maintainability.

Moreover, we have shown that network automation enables the implementation of large-scale changes across infrastructure components with minimal manual intervention. This directly translates into improved operational performance and more efficient use of human and technical resources. Consequently, various perspectives have emerged regarding this approach. Some professionals consider automation a major advancement in network management, as it reduces human error and accelerates incident response. On the other hand, some express concerns about the gradual loss of traditional technical skills due to increasing reliance on automated tools. Additionally, several researchers emphasize that the effectiveness of automation heavily depends on well-prepared planning and accurate scripting, which requires continuous training and ongoing adaptation to technological advancements.

Despite certain constraints in time and computing power, the work conducted in this thesis privides a solid foundation for future research in the field of network automation.

We hope that this project will serve as a useful reference and source of inspiration for students, researchers, and practitioners interested in exploring the vast possibilities that automation offers in building smarter, more agile, and more secure network systems.

We believe that this work constitutes a good starting point, but it opens the door to many future improvements and expansions.There are many aspects that were not addressed in this project, such as integration with monitoring and alert systems, automation of cybersecurity tasks, and application of artificial intelligence techniques in network analysis.Therefore, we believe that experimenting with other tools such as Puppet, Chef, SaltStack, as well as Terraform to manage infrastructure as code (IaC) will give a broader vision of the strengths and weaknesses of each tool, and help choose the most appropriate solution according to each case.

Bibliography

[1] M. El Rajab, L. Yang, and A. Shami, "Zero-Touch Networks: Towards Next-Generation Network Automation," Dec. 2023, [Online]. Available: http://arxiv.org/abs/2312.04159

[2] W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, "The Road Towards 6G: A Comprehensive Survey," Feb. 2021, doi: 10.1109/OJCOMS.2021.3057679.

[3] "ITU-T TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU Driving forces and vision towards Network 2030," 2020.

[4] J. F. Kurose and K. W. Ross, "A Top-Down Approach."

[5] A. I. Salameh and M. El Tarhuni, "From 5G to 6G—Challenges, Technologies, and Applications," Apr. 01, 2022, *MDPI*. doi: 10.3390/fi14040117.

[6] Olivier Bonaventure, "Computer Networking : Principles, Protocols and Practice," 2010, [Online]. Available: http://inl.info.ucl.ac.be/CNP3

[7] "Network Automation." Accessed: May 28, 2025. [Online]. Available: https://www.networkcomputing.com/network-management/network-automation

[8] J. Xu and G. Russello, "Automated Security-focused Network Configuration Management: State of the Art, Challenges, and Future Directions," in *Proceedings - 2022 9th International Conference on Dependable Systems and Their Applications, DSA 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 409–420. doi: 10.1109/DSA56465.2022.00061.

[9] A. Elezi and D. Karras, "On automating network systems configuration management," *CRJ*, pp. 18–31, Feb. 2023, doi: 10.59380/crj.v1i1.639.

[10] "What is the OSI Model? | Cloudflare." Accessed: May 28, 2025. [Online]. Available: https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/

[11] Darius, "End-to-End Network Automation: Why and How To Do It?"

[12] R. Rojas-Cessa, "Experiments on Computer Networks: Quickly Knowing the Protocols in the TCP/IP Suite."

[13] C. Este *et al.*, "Towards the Automation of Data Networks Hacia la automatización de las redes de datos," *Rev. ITECKNE-Universidad*, vol. 20, no. 1, pp. 25–33, doi: 10.15332/iteckne.

[14] "Automated Network Configuration Management," 2023. [Online]. Available: www.jetir.org

[15] F. Li, H. Lang, J. Zhang, J. Shen, and X. Wang, "PreConfig: A Pretrained Model for Automating Network Configuration," Mar. 2024, [Online]. Available: http://arxiv.org/abs/2403.09369

[16] O. G. Lira, O. M. Caicedo, and N. L. S. da Fonseca, "Large Language Models for Zero Touch Network Configuration Management," Aug. 2024, [Online]. Available: http://arxiv.org/abs/2408.13298

[17] B. Sunaryo, M. I. Rusydi, A. Hazmi, and M. Sasaki, "A Systematic Literature Review of Automation Quality of Service in Computer Networks: Research Trends, Datasets, and Methods," *J. RESTI (Rekayasa Sist. dan Teknol. Informasi)*, vol. 7, no. 2, pp. 353–366, Mar. 2023, doi: 10.29207/resti.v7i2.4810.

[18] P. R. R. Konala, V. Kumar, D. Bainbridge, and J. Haseeb, "A Framework for Measuring the Quality of Infrastructure-as-Code Scripts," Feb. 2025, [Online]. Available: http://arxiv.org/abs/2502.03127

[19] B. Meijer, L. Hochstein, and R. Moser, "Ansible: Up and Running Automating Configuration Management and Deployment the Easy Way THIRD EDITION," 2022. [Online]. Available: http://oreilly.com

[20] "Join the Ansible community | Ansible Documentation." Accessed: May 28, 2025. [Online]. Available: https://docs.ansible.com/community.html

[21] Pramod Muppala Kumar, "Leveraging Ansible for Middleware Scalability in Financial Services: A Technical Deep Dive," *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, vol. 11, no. 1, pp. 1800–1807, Feb. 2025, doi: 10.32628/CSEIT251112175.

[22] W. Irtaza, *IT Infrastructure Automation Using Ansible : Guidelines to Automate the Network, Windows, Linux, and Cloud Administration*. BPB Publications, 2021.

[23] "Cisco DevNet: APIs, SDKs, Sandbox, and Community for software developers and network engineers." Accessed: May 28, 2025. [Online]. Available:

https://developer.cisco.com/

[24] F. S. Bruschetti, J. Guevara, M. C. Abeledo, and D. A. Priano, "An Empirical Evaluation of Automated Configuration Tools for Software-Defined Networking: A Usability and Performance Perspective," *Ing. des Syst. d'Information*, vol. 28, no. 5, pp. 1127–1134, 2023, doi: 10.18280/isi.280502.

[25] N. Saavedra and J. F. Ferreira, "GLITCH: Automated Polyglot Security Smell Detection in Infrastructure as Code," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Sep. 2022. doi: 10.1145/3551349.3556945.

[26] "Ansible Documentation." Accessed: May 28, 2025. [Online]. Available: https://docs.ansible.com/

[27] E. Billoir, R. Laborde, A. S. Wazan, Y. Rutschle, and A. Benzekri, "Enhancing Secure Deployment with Ansible: A Focus on Least Privilege and Automation for Linux," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Jul. 2024. doi: 10.1145/3664476.3670929.

[28] "Python Tutorials – Real Python." Accessed: May 28, 2025. [Online]. Available: https://realpython.com/

[29] C. Carreira, N. Saavedra, A. Mendes, and J. F. Ferreira, "From 'Worse is Better' to Better: Lessons from a Mixed Methods Study of Ansible's Challenges," Apr. 2025, [Online]. Available: http://arxiv.org/abs/2504.08678

[30] P. K. Thopalle, "Enhancing Ansible Playbooks with Large Language Models: Revolutionizing Automation," *J. Artif. Intell. Cloud Comput.*, pp. 1–2, Jun. 2022, doi: 10.47363/JAICC/2022(1)E188.

[31] S. Wågbrant, V. D. Radic, V. Struhár, C. Möllberg, C. Karlsson, and V. Dahlén Radic, "AUTOMATED NETWORK CONFIGURATION A COMPARISON BETWEEN ANSIBLE, PUPPET, AND SALTSTACK FOR NETWORK CONFIGURATION Examiner: Thomas Nolte."

[32] T. Com and M. T. Munir, "Network Automation 1* Tayyab Muhammad," 2023. [Online]. Available: www.ajpojournals.org

[33] "Ansible Network Automation Introduction to Ansible for network engineers and operators."

[34] "AptGet/Howto - Community Help Wiki." Accessed: May 28, 2025. [Online]. Available: https://help.ubuntu.com/community/AptGet/Howto

[35] "GNS3 Full Pack installation." Accessed: May 28, 2025. [Online]. Available: https://dynamips.io/gns3v/