BACHELORTHESIS

Sylvester Ofulue

Design and Implementation of a Centralized Network Switch Management System

FACULTY OF ENGINEERING AND COMPUTER SCIENCE
Department of Information and Electrical Engineering

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Sylvester Ofulue

Design and Implementation of a Centralized Network Switch Management System

Bachelor Thesis based on the examination and study regulations for the
Bachelor of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner: Prof. Dr. Pawel Buczek
Second examiner: Mr. Sasikumar Subbarayan

Day of delivery: 11. February 2025

Sylvester Ofulue

Title of the paper
Design and Implementation of a Centralized Network Switch Management System

Keywords
Centralized Network Management, Switch Configuration, Ansible automation, Flask Framework, SQLite Database, Ansible Playbook, SSH communication.

Abstract
Efficient network switch management is vital for stability, security, and scalability. This thesis introduces a *Centralized Network Switch Management System* to prevent configuration drift, human errors, and inconsistent practices. Using **Flask** for the backend, **Ansible** for automation, and **SQLite** for storage, the system offers reliability, monitoring, and secure SSH-based management for **Cisco switches**. While suitable for ongoing management, it excludes initial switch deployment due to its reliance on pre-configured SSH access. The system lays a robust foundation for future advancements like analytics and broader vendor integration.

End of text

Sylvester Ofulue

Thema der Bachelorthesis
Entwurf und Implementierung eines zentralisierten Netzwerk-Switch-Managementsystems

Stichworte
Zentralisiertes Netzwerkmanagement, Switch-Konfiguration, Ansible-Automatisierung, Flask Framework, SQLite-Datenbank, Ansible Playbook, SSH-Kommunikation.

Kurzzusammenfassung

Eine effiziente Verwaltung von Netzwerk-Switches ist für Stabilität, Sicherheit und Skalierbarkeit unerlässlich. Diese Arbeit stellt ein zentralisiertes Netzwerk-Switch-Verwaltungssystem vor, um Konfigurationsabweichungen, menschliche Fehler und inkonsistente Vorgehensweisen zu verhindern. Das System verwendet Flask für das Backend, Ansible für die Automatisierung und SQLite für die Speicherung und bietet Zuverlässigkeit, Überwachung und sichere SSH-basierte Verwaltung für Cisco-Switches. Es eignet sich zwar für die laufende Verwaltung, schließt jedoch die anfängliche Bereitstellung von Switches aus, da es auf vorkonfigurierten SSH-Zugriff angewiesen ist. Das System legt eine solide Grundlage für zukünftige Weiterentwicklungen wie Analysen und eine breitere Anbieterintegration.

Ende des Textes

Acknowledgement

I would like to express my sincere gratitude to my supervisors, Prof. Dr. Pawel Buczek and Mr. Sasikumar Subbarayan, for their invaluable guidance and support throughout this project. I also thank q.beyond Logineer GmbH for providing the necessary resources for this research. I am deeply grateful to my colleagues, Timo Luttmann, Philipp Kaufmann, Christian Wolken, Stefan Riese, Franziska Mohr, and Seyi Ewegbemi, for their assistance and collaboration throughout this process. Additionally, I would like to acknowledge the use of OpenAI's ChatGPT for assisting with initial code implementation and Grammarly for proofreading and grammar corrections. Finally, I am truly thankful to my family and friends for their encouragement and patience during this journey.

Table of Contents

Table 1: List of Abbreviations

Abbreviation	Definition
ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
CLI	Command-Line Interface
CRUD	Create, Read, Update, Delete
CSS	Cascaded Style Sheets
DBMS	Database Management System
DDoS	Distributed Denial-of-Service
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP(S)	HyperText Transfer Protocol (Secure)
IaC	Infrastructure as Code
INI	Initialization
IOS	Internetwork Operating System
IP	Internet Protocol
MFA	Multi-Factor Authentication
NSG	Network Security Group
OSI	Open Systems Interconnections
RBAC	Role-Based Access Control
RESTful	Representational State Transfer
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TLS	Transport Layer Security
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
vCPU	virtual Central Processing Unit
VM	Virtual Machine
VPN	Virtual Private Network
WSGI	Web Server Gateway Interface
YAML	Yet Another Markup Language

Chapter 1: Introduction

1.1 Background and Motivation

Maintaining consistent and secure switch configurations in modern network environments is essential for ensuring network stability, security, and performance. Many organizations, including q.beyond Logineer GmbH, rely on predefined templates to configure network switches. These templates aim to standardize device configurations, reduce variability, and simplify troubleshooting. However, despite these efforts, human errors during manual configuration or troubleshooting are inevitable [1] [2] [3].
For instance, engineers may inadvertently misconfigure switches or forget to revert temporary changes made during troubleshooting. These errors can result in configuration drift, where a switch's running configuration deviates from the intended baseline. Such deviations may go unnoticed for extended periods, leading to performance degradation, potential security vulnerabilities, and operational inefficiencies [1].
Additionally, the lack of a centralized management platform often means network administrators must rely on fragmented tools or direct access to individual switches, which increases the likelihood of errors and complicates oversight [1] [3] [4].
This project proposes designing and implementing a **Centralized Network Switch Management System** to address these challenges. The system aims to minimize human error, ensure consistent configuration standards, and provide a clear overview of switch status and reachability while adhering to zero-cost principles and maintaining robust security measures.

1.2 Problem Statement

The current approach to switch configuration management within q.beyond Logineer GmbH relies heavily on manual processes and adherence to standard configuration templates. However:

- Human errors during manual configurations may result in configuration drift.
- Temporary troubleshooting changes may not always be reverted, leading to inconsistencies.

- Detecting and identifying misconfigurations manually can be time-consuming and error-prone.
- The lack of a centralized platform complicates monitoring, auditing, and enforcing compliance.
- Existing proprietary solutions are often expensive and may not align with cost-saving objectives.

This project seeks to address these problems by developing a centralized, automated, and secure network switch management system that will simplify configuration tasks, reduce errors, and ensure consistent oversight.

1.3 Company Overview

Logineer is a specialized IT service provider for the international logistics sector. It offers a comprehensive range of IT services tailored to the unique needs of logistics companies, particularly those involved in sea and air freight. These services encompass IT infrastructure, digital workplace solutions, and essential logistics applications, all delivered globally within a few days [5].

Logineer offers IT consulting, system integration, implementation, and operation. It strongly emphasizes cyber security, ensuring maximum protection for data and processes through extensive vulnerability analysis and redundant infrastructure [5]. Additionally, Logineer is a platinum-certified CargoWise service partner, helping companies automate and standardize their logistics processes using the CargoWise Transport Management System [5].

With a dedicated team of 200 IT and logistics experts, Logineer supports its clients' digital transformation efforts, making them faster, more efficient, and more competitive in the digital age [6]. Their 24/7 help desk ensures continuous support and promptly addresses IT-related issues [5].

1.4 Objectives

The primary objectives of this project are:

1. **Centralized Management:** Develop a unified web-based interface to manage multiple network switches from a single platform efficiently.

2. **Error Reduction:** Minimize manual configuration errors by leveraging automation through Ansible playbooks.
3. **Configuration Consistency:** Standardize switch configurations using predefined templates, ensuring device uniformity.
4. **Monitoring and Visibility:** Provide visibility of switch availability and operational status through an interactive dashboard.
5. **Zero-Cost Implementation:** Build a cost-effective solution by utilizing open-source tools and platforms.
6. **Security:** Implement robust security measures, including VPN connectivity, user authentication, and secure communication protocols like HTTPS and SSH.
7. **Flexibility:** Ensure interoperability with switches from any vendor, focusing on Cisco Catalyst switches with SSH access.
8. **Deployment Options:** Offer flexible hosting options, including Azure cloud infrastructure or on-premises deployment.

1.5 Scope of the Project

This project focuses on automating switch configurations using Ansible playbooks, tailored explicitly for Cisco switches with SSH enabled. It provides a centralized platform for management and monitoring through a Flask-based web interface. Data persistence is achieved through an SQLite database, storing essential information such as switch details, user credentials, and group associations.

The system incorporates ping-based monitoring to detect switch reachability and display results on an interactive dashboard for better visualization and to maintain network stability. Secure remote access is facilitated using OpenfortiVPN, ensuring safe communication between the management system and network devices. Furthermore, the system supports flexible deployment options, functioning effectively in both on-premises environments and Azure cloud infrastructure.

However, the project does not include advanced analytics, proprietary switch management tools integration, or direct hardware-level control over switches. It is also important to note that the system is not designed for initial switch deployment, as it relies on pre-existing SSH configurations to establish communication with devices. Despite these limitations, the project provides a robust and scalable centralized network switch management foundation.

1.6 Methodology Overview

The project utilizes the following methodologies and technologies to achieve the defined objectives:

- **The Flask Framework** develops the backend Application Programming Interface (API) to manage application logic and server-side operations [7].
- **HTML, JavaScript, and CSS** are utilized to create the web interface, ensuring usability, responsiveness, and user interaction.
- **Ansible** automates configuration tasks and standardizes deployments across network devices [8].
- **SQLite Database** stores persistent data, including user credentials and switch details [9].
- **OpenfortiVPN** ensures secure communication between the management system and remote network switches.
- **Azure Infrastructure** offers cloud-based deployment, supporting scalability and availability. [10]
- **Nginx** is used to serve the Flask application securely and efficiently [11].

The system integrates these components to establish a robust, scalable, and secure switch management platform. It’s important to highlight that the initial code development and brainstorming were conducted with the support of OpenAI’s ChatGPT.

1.7 Thesis Structure

This thesis is organized into six chapters:

- **Chapter 1: Introduction** — Introduces the background, problem statement, objectives, scope, and methodology.
- **Chapter 2: Literature Review** — Analyzes current network switch management systems, highlighting their shortcomings and identifying opportunities for enhancement.

- **Chapter 3: System Design and Architecture** covers the system requirements, including functional and non-functional specifications, as well as the architectural design.
- **Chapter 4: Implementation** — Outlines the technical implementation, covering the backend, frontend, database schema, Ansible playbooks, and security measures.
- **Chapter 5: Testing and Evaluation** examines the testing methodologies, results, and assessment of system performance.
- **Chapter 6: Conclusion and Future Work** — Summarizes the key contributions, lessons learned, and possible future improvements.

Chapter 2: Literature Review

This session analyzes existing research, tools, and technologies related to centralized network switch management systems. This chapter explores the challenges associated with manual switch configuration, the benefits of automation and centralization, and the tools and frameworks available for building such systems. Furthermore, it evaluates industry best practices, existing solutions, and the technological landscape to establish a foundation for the current project.

2.1 Challenges in Network Switch Management

Network administrators face numerous challenges when managing network switches, including:

- **Configuration Drift:** Manual configurations often lead to inconsistencies, especially when temporary troubleshooting changes are not reverted [2].
- **Human Errors:** Misconfigurations during deployment or troubleshooting may result in network outages or vulnerabilities [4].
- **Scalability Issues:** Managing an increasing number of switches without centralized tools becomes inefficient [3].
- **Lack of Centralized Oversight:** Monitoring individual switches without a unified platform increases operational complexity [3].
- **Security Risks:** Unauthorized access, misconfigurations, or outdated firmware may expose switches to security threats [2].

Addressing these challenges requires centralized management, automation, and standardized configuration practices.

2.2 Existing Network Switch Management Solutions

This section reviews commonly used tools and platforms for network switch management.

2.2.1 Cisco Catalyst Center

Cisco Catalyst Center is a management platform designed to enhance network operations through automation, security policies, and analytics. It simplifies managing Cisco network infrastructure while ensuring consistent performance across wired and wireless environments [12].

Benefits

This system has numerous benefits. Automation streamlines network operations and reduces operational costs. Additionally, it provides valuable insights into application and client performance, enhancing overall efficiency. Integrating with third-party tools significantly improves business agility, allowing for more flexible and responsive operations. Automated compliance checks strengthen network security, ensuring security protocols are consistently met [12].

License Cost

Cisco Catalyst Center operates under a subscription-based licensing model, typically dependent on the number of managed devices. Pricing details vary based on deployment specifics and required features, and it is recommended that you consult with Cisco sales representatives [13].

Supported Cisco Switch Models

Cisco Catalyst Center supports a wide range of Cisco switch models, including but not limited to the Catalyst 2960, 3560-CX, 3650, 3850, 4500, 6500, 9000 Series, and Nexus 9000 Series. Compatibility with advanced features may depend on hardware and software versions [14].

2.2.2 SolarWinds Network Management Software

SolarWinds offers network management tools tailored for monitoring, analyzing, and optimizing network performance. Known for its scalability and user-friendly interface, SolarWinds supports both small and large-scale networks [15].

Benefits

The system offers centralized monitoring of devices, servers, and applications, providing a detailed overview of the network. It includes advanced analytics for identifying performance bottlenecks and ensuring optimal operation. The scalable architecture accommodates network growth, allowing for seamless expansion. Enhanced security and compliance monitoring are integral features, ensuring the network stays secure and adheres to regulations. Additionally, the system provides customizable dashboards and reporting tools, enabling users to tailor the interface to their specific needs [15].

License Cost

SolarWinds licensing depends on the number of monitored elements, such as devices, interfaces, and nodes. Pricing varies based on selected modules, deployment size, and support requirements [15].

Supported Network Devices and Switch Models

SolarWinds supports various network devices, including switches, routers, firewalls, and wireless access points from multiple vendors. Commonly supported switch models include the Cisco Catalyst Series, HP ProCurve Series, Juniper EX Series, Aruba Switches, and Dell EMC Networking Series.
The software's device compatibility ensures seamless integration with diverse network hardware, allowing IT teams to monitor multi-vendor environments effectively. Specific features and monitoring capabilities may vary based on device firmware and software versions [15].

2.2.3 Ansible Automation

Ansible is a free, open-source automation platform developed by Red Hat that manages configurations, deploys applications, and efficiently orchestrates tasks. It simplifies complex IT tasks by automating workflows and managing infrastructure as code (IaC). Known for its agentless architecture, Ansible leverages SSH for Linux systems and WinRM for Windows systems, eliminating the need for client-side agents. This lightweight design reduces resource overhead and simplifies deployment [8].

Key Features of Ansible [8]:

- **Agentless Architecture:** Uses SSH and WinRM to connect to target devices without requiring additional software installations.
- **Simplicity:** Configurations are defined in YAML files (playbooks), which are human-readable and easy to understand.
- **Scalable Automation:** Supports large-scale deployments, simultaneously managing thousands of devices and systems.

- **Modular Design:** It offers a vast library of pre-built modules covering networking, cloud services, application management, and more.
- **Idempotency:** Ensures repeated tasks will not produce unintended changes to the system state.
- **Extensibility:** Supports custom modules and plugins to extend functionality as per organizational requirements.

Benefits of Ansible:

Managing configurations is simplified, allowing for consistent results across multiple devices. Deployment is rapid, efficiently automating software installations, updates, and configuration changes. IaC enables standardized infrastructure management and version control through playbooks. The system supports multiple platforms, including Linux, Windows, cloud environments, and network devices. Innovation is driven by a robust open-source community that frequently contributes modules, roles, and best practices [8].

Use Cases in Network Management:

Ansible plays a crucial role in modern network management by simplifying and automating complex tasks across diverse network environments. It is extensively used for automating network switch configurations, particularly on Cisco Catalyst devices, ensuring consistency and reducing the risk of human errors. Additionally, Ansible enforces security policies uniformly across network nodes, ensuring compliance with organizational standards and industry regulations. The tool is also employed for continuous network health monitoring, enabling administrators to detect and address issues proactively while generating detailed compliance reports. Furthermore, Ansible facilitates the seamless deployment of software patches and updates across multiple devices, maintaining uniform configurations and minimizing downtime [8].

Ansible Concepts

1. Building an Inventory:

An inventory in Ansible organizes managed nodes, providing system details and network locations in INI (default) or YAML formats. Inventories define groups of hosts, enabling centralized management and simplified task execution [16].

The example below shows an inventory file named *switches* with no specified extension. In such cases, the default .ini format is assumed:

```
#inventory/switches inventory file
[all]
HAW-SWT ansible_host=172.16.1.100 ansible_user=sylvester ansible_password=mypass
ansible_network_os=ios ansible_connection=network_cli
LAB-SWT ansible_host=172.16.2.100 ansible_user=sylvester ansible_password=mypass
ansible_network_os=ios ansible_connection=network_cli
SMS-SWT ansible_host=172.16.3.100 ansible_user=sylvester ansible_password=mypass
ansible_network_os=ios ansible_connection=network_cli
HOME-SWT ansible_host=192.168.2.240 ansible_user=admin ansible_password=my_pass
ansible_network_os=ios ansible_connection=network_cli

[HOME]
HOME-SWT ansible_host=192.168.2.240 ansible_user=admin ansible_password=my_pass
ansible_network_os=ios ansible_connection=network_cli

[LAB]
HAW-SWT ansible_host=172.16.1.100 ansible_user=sylvester ansible_password=mypass
ansible_network_os=ios ansible_connection=network_cli
LAB-SWT ansible_host=172.16.2.100 ansible_user=sylvester ansible_password=mypass
ansible_network_os=ios ansible_connection=network_cli
SMS-SWT ansible_host=172.16.3.100 ansible_user=sylvester ansible_password=mypass
ansible_network_os=ios ansible_connection=network_cli
```

In the example, any of the groups in square brackets ([all], [HOME], [LAB]) can be specified as a target. Individual hosts, such as LAB-SWT, can also be targeted.

Verification Commands:

- ***List inventory***: *ansible-inventory -i switches --list*

```
(myenv) sylvesterofulue@Sylvesters-iMac switch_management % ansible-
inventory -i inventory/switches --list
{
    "HOME": {
        "hosts": [
            "HOME-SWT"
        ]
    },
    "LAB": {
        "hosts": [
            "HAW-SWT",
            "LAB-SWT",
            "SMS-SWT"
        ]
    },
    "_meta": {
        "hostvars": {
            "HAW-SWT": {
                "ansible_connection": "network_cli",
                "ansible_host": "172.16.1.100",
                "ansible_network_os": "ios",
                "ansible_password": "mypass",
                "ansible_user": "sylvester"
            },
            "HOME-SWT": {
                "ansible_connection": "network_cli",
                "ansible_host": "192.168.2.240",
                "ansible_network_os": "ios",
                "ansible_password": "my_pass",
                "ansible_user": "admin"
            },
            "LAB-SWT": {
                "ansible_connection": "network_cli",
                "ansible_host": "172.16.2.100",
                "ansible_network_os": "ios",
                "ansible_password": "mypass",
                "ansible_user": "sylvester"
            },
            "SMS-SWT": {
                "ansible_connection": "network_cli",
                "ansible_host": "172.16.3.100",
                "ansible_network_os": "ios",
                "ansible_password": "mypass",
                "ansible_user": "sylvester"
            }
        }
    },
    "all": {
        "children": [
            "ungrouped",
            "HOME",
            "LAB"
        ]
    }
}
```

- ***Test connectivity***: ansible [target-group] -m ping -i switches

```
(myenv) sylvesterofulue@Sylvesters-iMac switch_management % ansible HOME -m
ping -i inventory/switches
HOME-SWT | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

2. Creating a Playbook

Playbooks are YAML-based automation blueprints Ansible uses to deploy and configure managed nodes. A playbook comprises a sequence of plays, where each play contains an ordered set of **tasks** assigned to managed nodes listed in an inventory file. These plays dictate Ansible's sequence of operations to achieve a specific objective [17].

An essential aspect of the playbook is its use of Cisco IOS collection to facilitate configuration management, state validation, and information retrieval from IOS-based network infrastructure [18].

Key Cisco IOS Modules [18]:

- **ios_facts:** Collects device-specific information (e.g., version, model, serial number).
- **ios_config:** Applies configuration commands to Cisco IOS devices.
- **ios_command:** Executes arbitrary commands on Cisco IOS devices.
- **ios_interfaces:** Manages interface configuration, including enabling/disabling interfaces.
- **ios_system:** Manages global configurations, including hostname and domain settings.

The **ios_facts** module, as shown in the example below, efficiently gathers device-specific information from Cisco IOS devices.

Example playbook: Collect Device Information from Cisco IOS Devices

```
# playbook_collect_device_info.yml
# This playbook collects and displays device information from Cisco IOS devices.

- name: Collect device information
  hosts: all  # Target all hosts. This can be passed dynamically.
  gather_facts: no  # Disable fact gathering.
  tasks:
    - name: Gather device facts
      cisco.ios.ios_facts:  # Ansible module to collect facts from Cisco IOS
devices.

    - name: Display device information  # Task name: Displaying collected device
information
      debug:  # Debug module to show the gathered information
        msg:  # Displayed messages
          - "Model: {{ ansible_net_model | default('N/A') }}"  # Show device model,
default to 'N/A' if not available.
          - "Serial: {{ ansible_net_serialnum | default('N/A') }}"  # Show device
serial number, default to 'N/A' if not available.
          - "Software Version: {{ ansible_net_version | default('N/A') }}"  # Show
software version, default to 'N/A' if not available.
          - "Hardware: {{ ansible_net_hardware | default('N/A') }}"  # Show
hardware type, default to 'N/A' if not available.
```

Playbooks execution syntax:

ansible-playbook playbooks/device_info.yml -i inventory/switches -l HOME

Ansible will default to the value passed to the hosts variable within the playbook if no target is specified. In the example above, the target would be set to *all*.

```
(myenv) sylvesterofulue@Sylvesters-iMac switch_management % ansible-
playbook playbooks/device_info.yml -i inventory/switches -l HOME

PLAY [Collect device information]
************************************************************************
******************************************

TASK [Gather device facts]
************************************************************************
*************************************************
ok: [HOME-SWT]

TASK [Display device information]
************************************************************************
******************************************
ok: [HOME-SWT] => {
    "msg": [
        "Model: WS-C3560-24TS",
        "Serial: FDO1313X3U6",
        "Software Version: 15.0(2)SE11",
        "Hardware: N/A"
    ]
}

PLAY RECAP
************************************************************************
****************************************************************
HOME-SWT                   : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

If parameters, especially SSH connection parameters, are not correctly configured, playbook execution will fail precisely when the error occurs. In the example below, an SSH key misconfiguration resulted in a fatal error during execution.

```
(myenv) sylvesterofulue@Sylvesters-iMac switch_management % ansible-
playbook playbooks/device_info.yml -i inventory/switches -l SMS-SWT

PLAY [Collect device information]
************************************************************************
******************************************

TASK [Gather device facts]
************************************************************************
*************************************************
```

```
fatal: [SMS-SWT]: FAILED! => {"changed": false, "msg": "ssh connection
failed: ssh connect failed: kex error : no match for method server host key
algo:
server [ssh-rsa], client [ssh-ed25519,ecdsa-sha2-nistp521,ecdsa-sha2-
nistp384,ecdsa-sha2-nistp256,sk-ssh-ed25519@openssh.com,sk-ecdsa-sha2-
nistp256@opens
sh.com,rsa-sha2-512,rsa-sha2-256]"}

PLAY RECAP
****************************************************************************
********************************************************************
SMS-SWT                    : ok=0    changed=0    unreachable=0    failed=1
skipped=0    rescued=0    ignored=0
```

As previously mentioned, one significant advantage of using playbooks is their simultaneous ability to execute commands across thousands of devices [17]. A significant disadvantage is that Ansible is primarily designed as a command-based tool, which might be very difficult to memorize, especially when it is rarely used.

The centralized network switch management system project offers a GUI from which these Ansible engines will be executed.

2.3 Key Technologies and Frameworks

2.3.1 Flask Framework

Flask is a lightweight Python web framework designed to build web applications quickly and efficiently. Known for its simplicity and extensibility, Flask is ideal for small-scale projects and large, complex systems. It follows the WSGI (Web Server Gateway Interface) standard and provides developers with the essential tools to create web services, APIs, and dynamic web applications with minimal boilerplate code [7].

Key Features of Flask

Flask is designed as a microframework, offering core essentials while allowing developers to add additional components. It includes built-in support for Jinja2, enabling developers to create dynamic HTML templates efficiently. Flask's architecture is highly extensible, supporting many extensions for adding functionality such as authentication, database integration, and API management. It also features a built-in development server and debugger, facilitating efficient development and testing. Flask simplifies URL handling with clean and flexible routing mechanisms and is well-suited for building RESTful APIs with precise and structured request handling. Additionally, it provides integrated support for secure cookies, enabling session management to maintain user state [7].

Advantages of Flask

Flask has several advantages. It is easy to learn due to its minimalistic approach, making it beginner-friendly. It is highly customizable, allowing developers to tailor applications to specific requirements without unnecessary overhead. Flask benefits from an active community, which provides extensive documentation and third-party libraries. Additionally, it offers compatibility, seamlessly integrating with popular Python libraries and tools [7].

2.3.2 Ansible Automation

For an in-depth discussion on Ansible Automation, please refer to session 2.2.3 Ansible Automation.

2.3.3 SQLite Database

SQLite is a lightweight, serverless, self-contained SQL database engine widely recognized for its simplicity, reliability, and cross-platform compatibility. It is an open-source database management system suitable for embedded systems and lightweight applications [9].

Appropriate Uses for SQLite

SQLite is particularly suitable for applications with low to medium traffic, embedded systems, and situations where simplicity and ease of use are paramount. SQLite is applicable when a full-fledged client-server database system would be more than required [9].

Distinctive Features

SQLite's distinctive features include its zero-configuration setup, cross-platform compatibility, and public domain licensing. These features make it widely acceptable and integrated into various projects [9].

Testing and Reliability

Rigid testing, including extensive automated tests and real-world usage scenarios, ensures SQLite's reliability and helps maintain the database engine's robustness and stability [9].

2.3.4 Nginx Web Server

Nginx is a high-performance, open-source reverse proxy server and web server that handles many concurrent connections with low memory usage and high efficiency. Originally developed to address the C10K problem, it has become one of the most widely used web servers globally. It offers capabilities beyond traditional web serving, including reverse proxying, load balancing, caching, and efficient static content serving [11].

Key Features of Nginx

Nginx acts as a reverse proxy, serving as an intermediary between clients and backend servers, which improves performance and enhances security. It enables load balancing by evenly spreading incoming requests among multiple servers, ensuring efficient use of resources and avoiding overloading any single server. Caching mechanisms store frequently accessed data locally, reducing backend load and improving response times. With its scalable architecture, Nginx efficiently handles a high volume of concurrent connections, which is suitable for high-traffic websites. Additionally, it integrates robust security features, such as Secure Socket Layer/Transport Layer Security (SSL/TLS) termination, access controls, and Distributed Denial-of-Service (DDoS) mitigation. The modular design further allows Nginx to support extended functionalities without compromising performance [11].

Deployment Scenarios

Nginx is widely used as a reverse proxy to handle incoming traffic and enhance scalability. Additionally, it acts as a load balancer, evenly spreading requests across backend servers to maintain optimal performance. Caching static content reduces latency and minimizes server load, while SSL/TLS termination offloads cryptographic tasks from backend servers, enhancing overall efficiency [11].

Integration with Flask Applications

In web application development, Nginx is often used as a frontend reverse proxy with Flask. Nginx manages client requests, serves static files, and forwards dynamic content requests to the Flask application server (e.g., Gunicorn or uWSGI). This integration significantly improves the performance, scalability, and security of Flask-based applications.
By incorporating Nginx into the deployment architecture, organizations can build an optimized and resilient web infrastructure that meets modern application demands [11].

2.4 Best Practices in Switch Configuration Management [19]

Standardized Configuration Templates: Reducing variability by enforcing uniform configurations.

Regular Auditing and Compliance Checks: Ensuring switches comply with security and operational policies.

Regular Backups: Regularly backing up configurations to prevent data loss.

Maintain Documentation: Up-to-date records of configurations, inventories, and topologies simplify troubleshooting, streamline changes, and ensure clarity in network modifications.

These practices are essential for minimizing configuration errors and ensuring long-term network stability.

2.5 Gaps in Existing Solutions

While existing network management tools offer robust features, they often exhibit key limitations that impact usability, flexibility, and cost efficiency. The primary gaps include:

- **High Licensing Costs:** Proprietary tools usually have substantial licensing fees, which conflict with zero-cost objectives. Open-source solutions like Ansible effectively address this concern.
- **Vendor-Specific Limitations:** Many tools are restricted to specific hardware vendors, limiting their versatility in multi-vendor environments. Ansible mitigates this with its vendor-agnostic approach.
- **Complex Command-Line Dependency:** Despite Ansible's flexibility, running playbooks traditionally relies on memorizing and executing command-line instructions. This dependency creates a steep learning curve for users without extensive technical expertise.

This project addresses the command-line dependency gap by introducing a user-friendly Graphical User Interface (GUI). Instead of requiring users to memorize commands or interact with the terminal, the GUI simplifies playbook execution, host management, and configuration tasks. While Ansible remains the backend engine, the GUI empowers users with an intuitive and accessible interface, enhancing overall efficiency and reducing the barriers to effective network management.

By bridging this gap, the project combines Ansible's powerful automation capabilities with a streamlined GUI, offering an accessible, zero-cost, and scalable solution for switch configuration management.

Chapter 3: System Design and Architecture

This chapter outlines the foundational aspects of the system's design and architecture. It includes an overview of the system requirements, functional and non-functional specifications, the architectural structure, and the rationale for chosen technologies. Additionally, it explores alternatives considered during the design phase and integrates system diagrams for clarity.

3.1 System Requirements

The system requirements define the functionality, performance, and environmental constraints the centralized switch management system must meet. These requirements are categorized as functional and non-functional.

3.1.1 Functional Requirements

- **Switch Configuration Management:** Enable the execution of Ansible playbooks to configure and manage Cisco Catalyst switches seamlessly.
- **User Management:** Provide secure user authentication and access control mechanisms to prevent unauthorized access.
- **Device Monitoring:** Display the status of switches (online or offline) upon user login. Device status updates are available through manual dashboard refreshes.
- **Configuration Backup:** Provide on-demand backups of switch configurations to prevent data loss.
- **Group Operations:** Allow users to group switches logically for efficient batch operations, such as running playbooks on multiple devices simultaneously.

3.1.2 Non-Functional Requirements

- **Scalability:** The system can support many switches without compromising performance, with a target capacity of at least 500 devices.
- **Usability:** Provide an intuitive web interface to simplify user interaction.
- **Security:** Ensure secure access with robust authentication mechanisms and adhere to organizational cloud security policies.
- **Cost-Effectiveness:** Use open-source tools to ensure a zero-cost solution.

- **Compatibility:** Ensure compatibility with existing infrastructure, specifically Cisco Catalyst switches.

3.2 Framework Conditions

The framework conditions set the stage for system implementation and highlight constraints based on hardware, software, and organizational policies.

3.2.1 Hardware

Switches: Cisco Catalyst switches form the core network infrastructure.

Server Environment: The system is hosted on an Azure virtual machine or an on-premises server.

Minimum Server Specifications:

CPU: Dual-core 2 GHz or higher

RAM: 4 GB (8 GB recommended for scalability)

Storage: 10 GB (minimum) with sufficient space for backups

Network: Gigabit Ethernet interface

TFTP Server: Tftpd64 by Ph. Jounin is used for configuration backup and restore operations.

3.2.2 Software

Operating System: Ubuntu 20.04 LTS for the server.

Automation Tool: Ansible is used to manage switch configurations.

Web Framework: Flask is the Python framework for the web application.

Database: SQLite stores user, switch, and group data.

3.2.3 Organizational Constraints

- The system must comply with the company’s security policies and existing IT infrastructure.
- Development is restricted to open-source tools to adhere to the zero-cost requirement.

3.3 Architectural Design

The system architecture follows a modular design, integrating various components to ensure scalability, maintainability, and performance.

3.3.1 High-Level Architecture

The system comprises the following key components:

User Interface (UI): A web-based interface built using HTML and Bootstrap for managing switches, users, and groups. UI operates at the application layer of the OSI model, ensuring user-friendly access to system features.

Application Layer: The Flask framework is the middleware that handles user requests and executes Ansible playbooks. This component aligns with the session and presentation layers by bridging the user interface and underlying automation, enabling structured data exchange and communication flow management.

Database: SQLite stores all persistent data, including user credentials, switch information, and group associations.

Automation Engine: Ansible automates configuration management tasks, interacting directly with Cisco Catalyst switches via SSH. These interactions leverage the transport layer (e.g., TCP) to ensure reliable data exchange and the network layer (e.g., IP) for routing between nodes [20].

Monitoring Dashboard: Displays the current reachability status of switches upon user login and updates upon manual refresh.

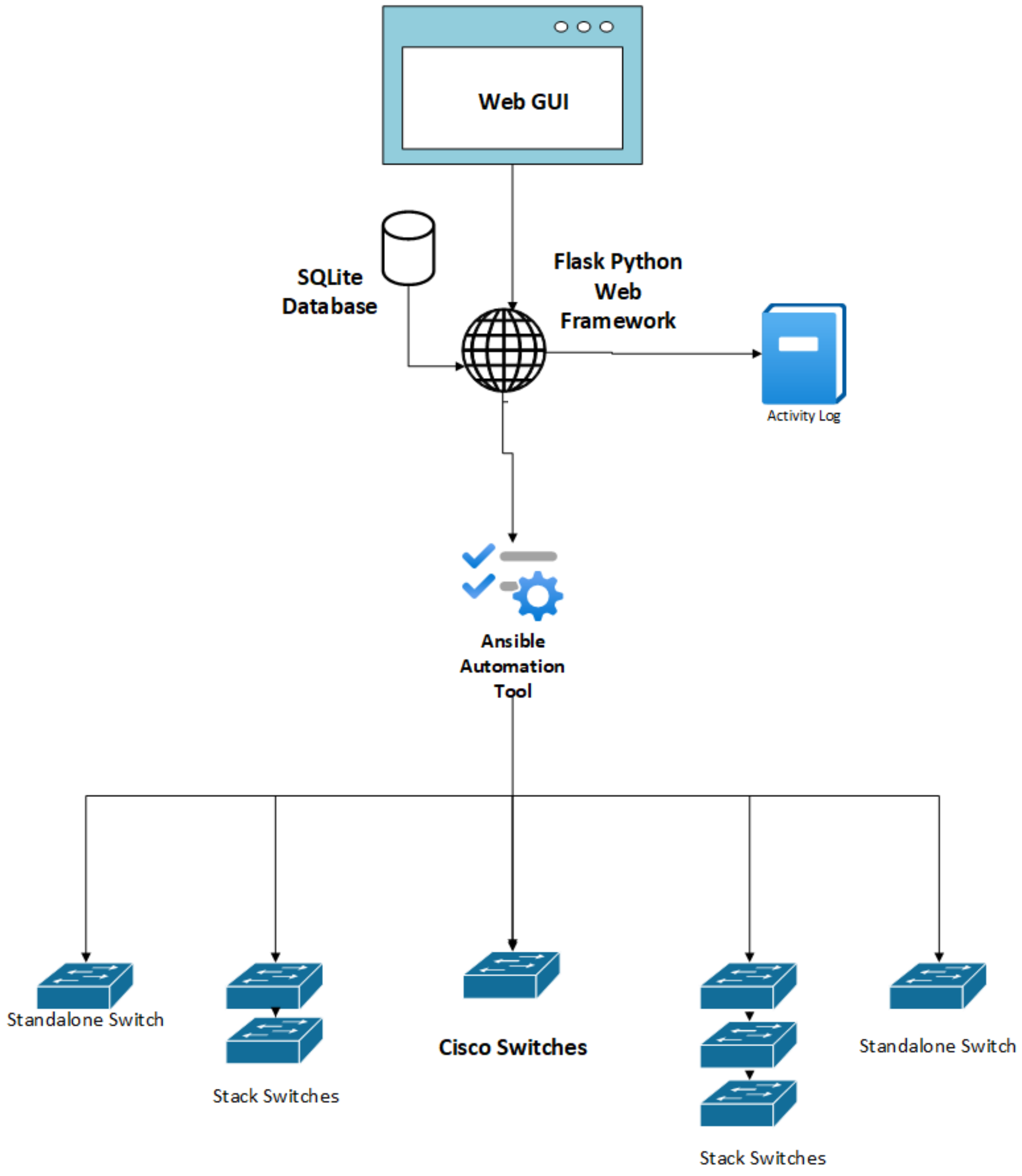


Figure 1: High-Level System Architecture

3.3.2 Modular Components

Authentication Module: Manages user login/logout and enforces multi-layered access control.

Switch Management Module: Handles add, edit, delete, and group operations for switches.
Monitoring Module: Displays device reachability and health using charts upon user request.
Playbook Execution Module: Executes playbook configuration on-demand.

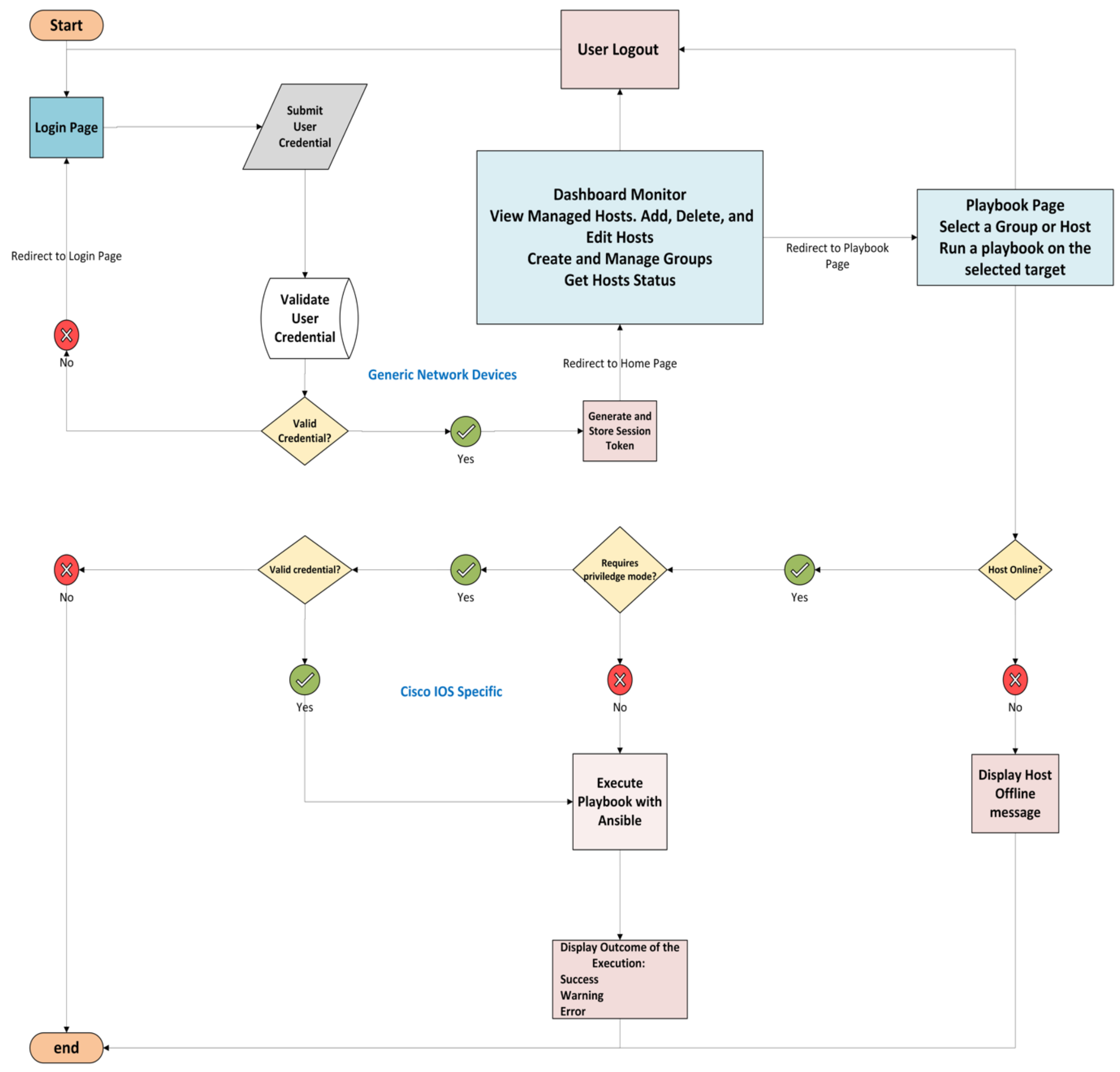


Figure 2: Component Interaction Flowchart

3.4 Technology Alternatives and Evaluation

During the design phase, alternative technologies were evaluated to determine the most appropriate solution. The following sections summarize the alternatives and the rationale for the chosen solutions.

3.4.1 Automation Tools: Ansible vs. Puppet, Chef

Alternatives Considered:

- **Puppet** is a configuration management tool designed to automate the deployment and configuration of software across multiple servers. It uses an agent-based architecture, with a Puppet master server controlling deployment and Puppet agents executing tasks on nodes based on *catalogs* generated from manifests [21].
- **Chef** is a configuration management tool designed for server management. It relies on a central Chef Server and distributed Chef Clients to handle deployments. While powerful, a human administrator must select the system and use *cookbooks* to configure tasks [21].

Chosen Solution:

Ansible was selected due to its simplicity, agentless architecture, and minimal setup requirements. These features align with the project's goals of reducing complexity, ensuring ease of use, and maintaining robust functionality.

3.4.2 Backend Framework: Flask vs. Django

Alternatives Considered:

- **Django** is a full-stack framework recognized for its extensive built-in features like authentication, URL routing, a template engine, database migrations, and an ORM. These features make it ideal for building robust and scalable web applications [22].

Chosen Solution:

Flask's lightweight and modular design offered the project the necessary flexibility. Its minimalist nature made integrating with the system's specific requirements easier, avoiding the overhead associated with Django's more extensive feature set.

3.4.3 Database: SQLite vs. MySQL, PostgreSQL

Alternatives Considered:

- **MySQL** is a widely used relational database management system known for its speed, reliability, and scalability. It employs a structured data model with tables, rows, and columns and leverages SQL for database access. Open-source and governed by the GNU General Public License (GPL), MySQL is flexible and user-friendly and supports clustering for seamless scalability [23].

- **PostgreSQL** is a powerful, open-source object-relational database system designed for complex workloads. Developed for over 35 years by the POSTGRES project, it offers ACID compliance, rich data types, extensibility, and advanced features like geospatial support with PostGIS. Renowned for its reliability and versatility, it is compatible with major operating systems and ideal for robust applications [23].

Chosen Solution:

SQLite was sufficient for the system's scale. It offered simplicity and avoided the operational overhead of managing a full-fledged RDBMS. Its lightweight nature and self-contained setup made it ideal for a standalone, resource-efficient application.

3.4.4 Front-End Framework: HTML/Bootstrap vs. React, Angular

Alternatives Considered:

- **React** is a declarative, efficient, and flexible JavaScript library for building user interfaces. It allows developers to create complex UIs from small, isolated code called components [24].
- **Angular** is a web framework maintained by Google that enables developers to build fast, reliable applications. It offers a comprehensive suite of tools, APIs, and libraries to streamline development workflows and support projects as they scale in team size and codebase complexity [25].

Chosen Solution:

Bootstrap’s simplicity and responsiveness made it the optimal choice for the project. It facilitated the creation of a clean, minimalistic UI that aligns with the system’s focus on usability and efficiency without introducing unnecessary complexity. Additionally, the learning curve for HTML and Bootstrap is significantly faster than that of modern frameworks like React or Angular, making it a practical solution for the project's needs.

3.5 Key Design Decisions

Summary of the decisions made during the system design phase are as follows:

Agentless Automation: Ansible was chosen for its agentless architecture, which simplifies deployment and reduces overhead.

Lightweight Framework: Flask was preferred over Django because of its minimalistic approach, which aligns with the project’s simplicity requirements.
SQLite Database: Selected for its serverless nature, meeting the project's scale and zero-cost objectives.
Bootstrap for UI: Ensured responsiveness and simplicity in the user interface design.
On-Demand Operations: All playbook executions, including backups and monitoring, are initiated by user actions to avoid unnecessary load on devices.

Chapter 4: Implementation

This chapter provides an in-depth explanation of the implementation phase of the Centralized Network Switch Management System. It covers integrating various technologies, including Flask, Ansible, SQLite, and Nginx, into a cohesive platform for managing network switches. The chapter also describes the development of backend services, interfaces, database management, and automation tool integration. Challenges encountered during this phase and their solutions are discussed to highlight the evolution of the implementation.
The complete source code for the Centralized Network Switch Management System is included in the submitted CD.

4.1 Overview of the Technology Stack

The system's functionality relies on a carefully designed technology stack, with each component fulfilling a critical role:

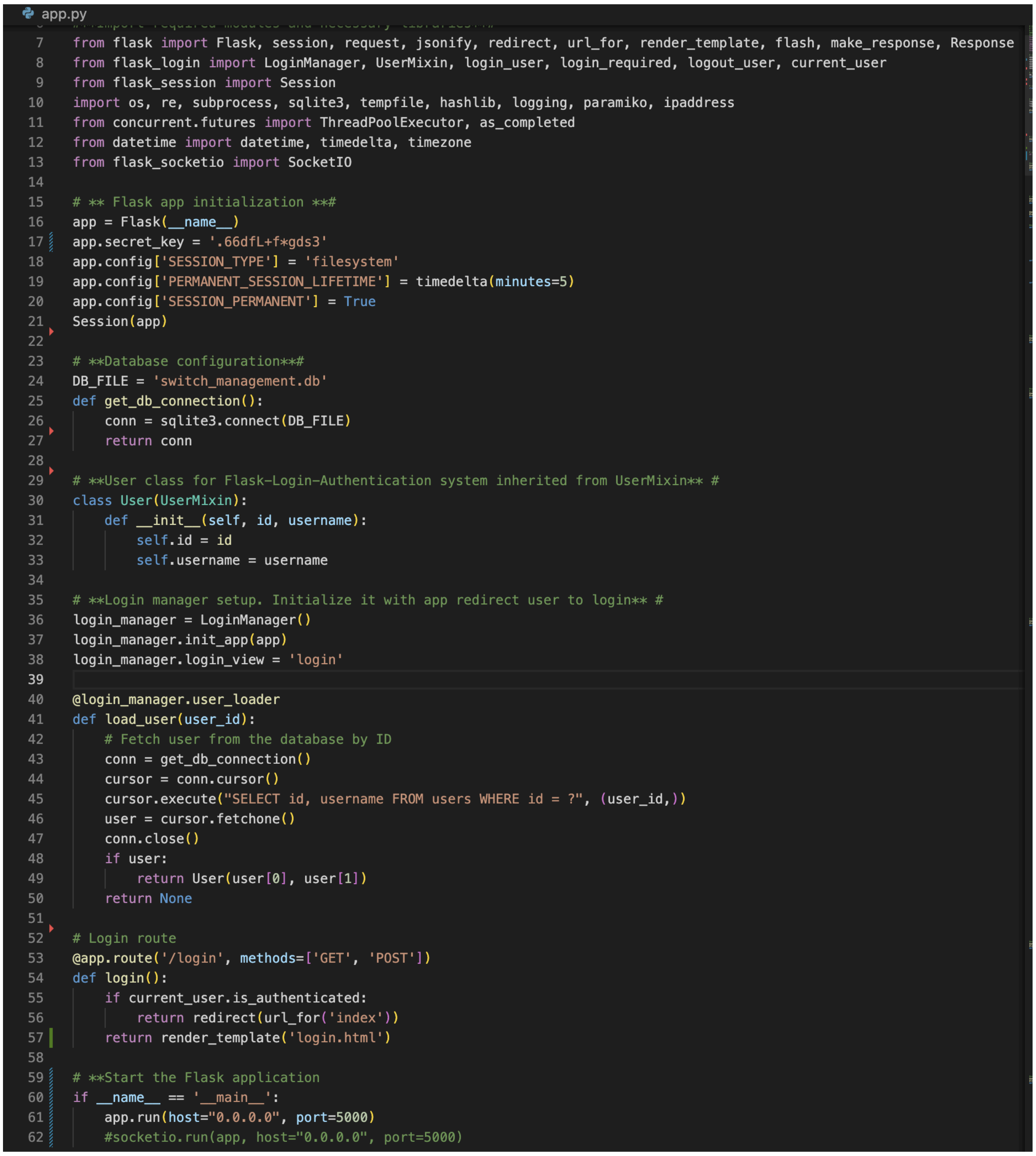
- **Flask** handles backend logic, routing, and API endpoints.
- **Gunicorn** serves as the WSGI server, enabling efficient handling of concurrent requests.
- **SQLite** provides persistent storage for user accounts, device information, and logs.
- **Ansible** automates key operations such as configuration management and backups.
- **Nginx** acts as a reverse proxy, providing SSL/TLS encryption and static content caching.
- **HTML, CSS, and JavaScript (Bootstrap 4.5.2, Chart.js)** power the frontend interface, ensuring usability and real-time visualization.

This technology stack enables seamless interaction between layers, adhering to the multi-tier architecture described in Chapter 3.

4.2 Backend Implementation

The switch management system's backend is implemented using the Flask framework, which was chosen for its simplicity, flexibility, and robust support for web application development. The backend handles user requests, manages database operations, executes Ansible playbooks, and facilitates communication between the user interface and the underlying system components.

Figure 3 shows code snippets of app.py, the main application file responsible for routing and initializing the application. For the complete implementation, see Appendix A.1: app.py.



```
app.py
from flask import Flask, session, request, jsonify, redirect, url_for, render_template, flash, make_response, Response
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user
from flask_session import Session
import os, re, subprocess, sqlite3, tempfile, hashlib, logging, paramiko, ipaddress
from concurrent.futures import ThreadPoolExecutor, as_completed
from datetime import datetime, timedelta, timezone
from flask_socketio import SocketIO

# ** Flask app initialization **#
app = Flask(__name__)
app.secret_key = '.66dfL+f*gds3'
app.config['SESSION_TYPE'] = 'filesystem'
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(minutes=5)
app.config['SESSION_PERMANENT'] = True
Session(app)

# **Database configuration**#
DB_FILE = 'switch_management.db'
def get_db_connection():
    conn = sqlite3.connect(DB_FILE)
    return conn

# **User class for Flask-Login-Authentication system inherited from UserMixin** #
class User(UserMixin):
    def __init__(self, id, username):
        self.id = id
        self.username = username

# **Login manager setup. Initialize it with app redirect user to login** #
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'

@login_manager.user_loader
def load_user(user_id):
    # Fetch user from the database by ID
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT id, username FROM users WHERE id = ?", (user_id,))
    user = cursor.fetchone()
    conn.close()
    if user:
        return User(user[0], user[1])
    return None

# Login route
@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    return render_template('login.html')

# **Start the Flask application
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5000)
    #socketio.run(app, host="0.0.0.0", port=5000)
```

Figure 3: App.py Code Snippets

4.2.1 Integration with Ansible

The switch management system integrates seamlessly with Ansible to execute network automation tasks, such as backing up configurations, retrieving device information, and ensuring security compliance. This functionality is achieved through a Flask API endpoint (*/run_playbook*) that dynamically processes user requests to run specified playbooks on targeted hosts or groups.

Workflow and Dynamic Inventory Management

The integration uses a dynamic inventory system, allowing hosts or groups to be selected programmatically based on database queries. The inventory is generated for a specific host or all reachable hosts within a specified group. This dynamic approach ensures flexibility and scalability:

1. **Host-Based Execution**:
   - The user specifies a host by name.
   - The system fetches the host's details (e.g., IP address, Ansible credentials) from the SQLite database.
   - Before executing the playbook, the system uses a ping utility to check the host's reachability.
   - If the host is reachable, it is added to the dynamic inventory, and the playbook is executed.
2. **Group-Based Execution**:
   - The user specifies a group name.
   - The system fetches all hosts belonging to the group from the database.
   - Each host's reachability is checked; only reachable hosts are included in the dynamic inventory. This ensures that unreachable hosts do not cause playbook execution failures.

Validating Playbook Parameters

The system includes a validation mechanism to ensure that only authorized playbooks are executed. Each playbook is mapped to a predefined list of required variables. When a playbook is selected, the system verifies whether the necessary variables are provided in the request. If required variables are missing, an appropriate error message is returned to the user.

For instance:

- The restore_config.yml playbook requires extra parameters, such as the tftp_server and filename variables. The *vlan_config.yml* playbook requires variables such as the *vlan_id, vlan_name, interface, description,* and *mode.*

Playbook Execution

Once the dynamic inventory is created and the variables are validated, the system executes the playbook using the subprocess module. The command dynamically includes inventory details, optional extra variables, and target specifications:

```
result = subprocess.run(
            ['ansible-playbook', playbook, '-i', inventory_file, '-l', target,
                *extra_vars_args],
            capture_output=True, text=True
        )
```

Error Handling and Logging

Comprehensive error-handling mechanisms are implemented to manage potential issues during playbook execution:

- Invalid playbook names are logged, and a 400 error is returned.
- Missing required variables prompt detailed error messages, enabling users to correct their input.
- Unreachable hosts are identified and logged, and the user is informed about which hosts cannot be accessed.
- Playbook execution failures are captured, and the system returns both the standard error output and the return code for debugging.

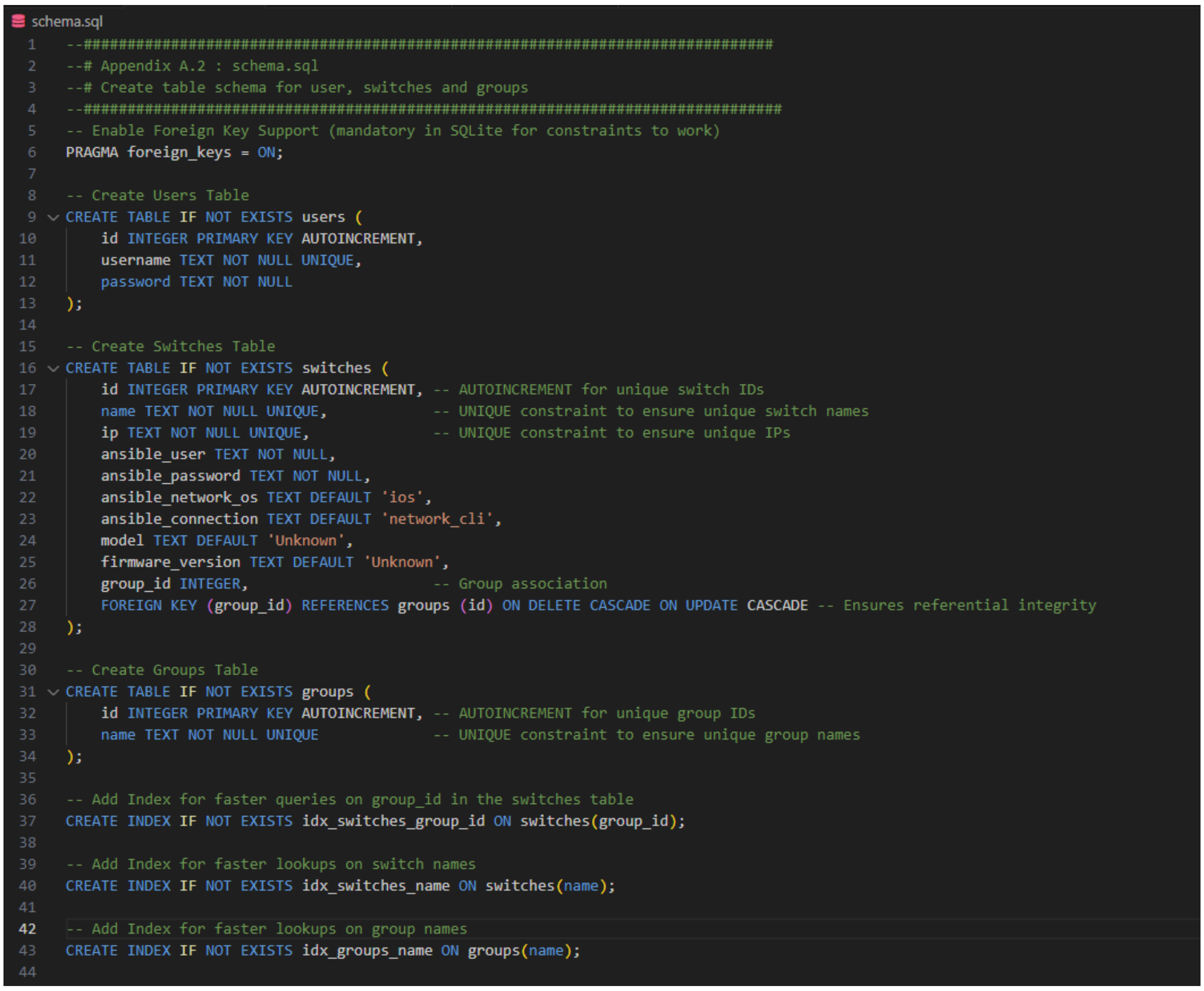
Additionally, the system logs every playbook request, including the logged-in user's username, the playbook executed, and the targets involved. To maintain security, sensitive information such as Ansible passwords is excluded from the logs.

4.3 Database Schema

The system uses SQLite as the database engine to store persistent information about users, switches, and groups. The schema consists of three primary tables: users, switches, and groups tables. On Linux OS, the schema.sql is read to *switch_management.db* using the following commands:

$sqlite3 switch_management.db

>. read schema.sql



```sql
--################################################################################
--# Appendix A.2 : schema.sql
--# Create table schema for user, switches and groups
--#################################################################################
-- Enable Foreign Key Support (mandatory in SQLite for constraints to work)
PRAGMA foreign_keys = ON;

-- Create Users Table
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL
);

-- Create Switches Table
CREATE TABLE IF NOT EXISTS switches (
    id INTEGER PRIMARY KEY AUTOINCREMENT, -- AUTOINCREMENT for unique switch IDs
    name TEXT NOT NULL UNIQUE,            -- UNIQUE constraint to ensure unique switch names
    ip TEXT NOT NULL UNIQUE,              -- UNIQUE constraint to ensure unique IPs
    ansible_user TEXT NOT NULL,
    ansible_password TEXT NOT NULL,
    ansible_network_os TEXT DEFAULT 'ios',
    ansible_connection TEXT DEFAULT 'network_cli',
    model TEXT DEFAULT 'Unknown',
    firmware_version TEXT DEFAULT 'Unknown',
    group_id INTEGER,                     -- Group association
    FOREIGN KEY (group_id) REFERENCES groups (id) ON DELETE CASCADE ON UPDATE CASCADE -- Ensures referential integrity
);

-- Create Groups Table
CREATE TABLE IF NOT EXISTS groups (
    id INTEGER PRIMARY KEY AUTOINCREMENT, -- AUTOINCREMENT for unique group IDs
    name TEXT NOT NULL UNIQUE             -- UNIQUE constraint to ensure unique group names
);

-- Add Index for faster queries on group_id in the switches table
CREATE INDEX IF NOT EXISTS idx_switches_group_id ON switches(group_id);

-- Add Index for faster lookups on switch names
CREATE INDEX IF NOT EXISTS idx_switches_name ON switches(name);

-- Add Index for faster lookups on group names
CREATE INDEX IF NOT EXISTS idx_groups_name ON groups(name);

```

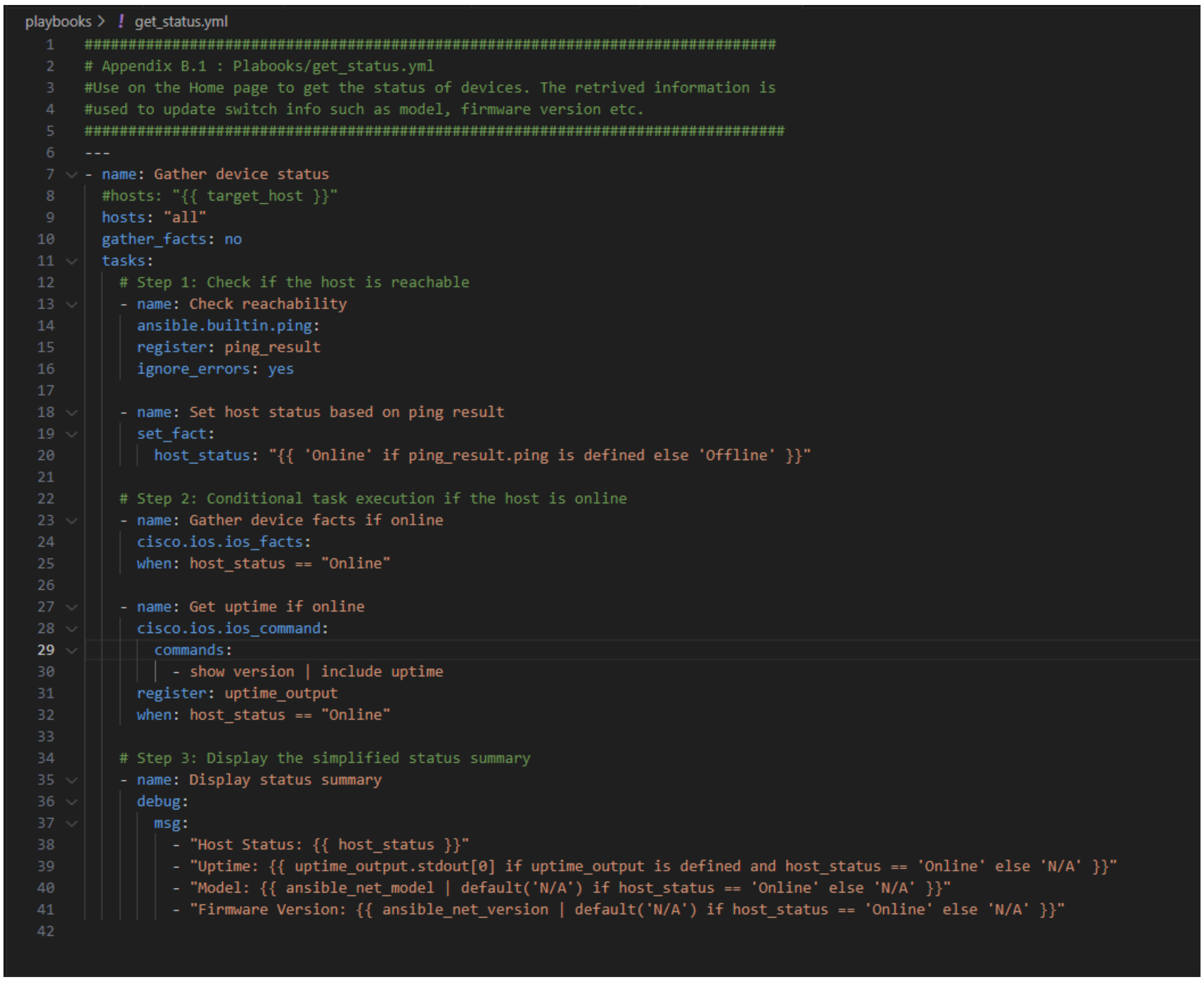
Figure 4: Schema.sql

The relationships between tables ensure that switches are associated with groups and that user credentials are stored securely. The implementation code location on the CD is in Appendix A.2: schema.sql.

4.4 Ansible Playbooks

Ansible playbooks automate tasks such as getting or setting device configurations. Each playbook is tailored for Cisco Catalyst switches and uses SSH (Secure Shell) for secure communication.

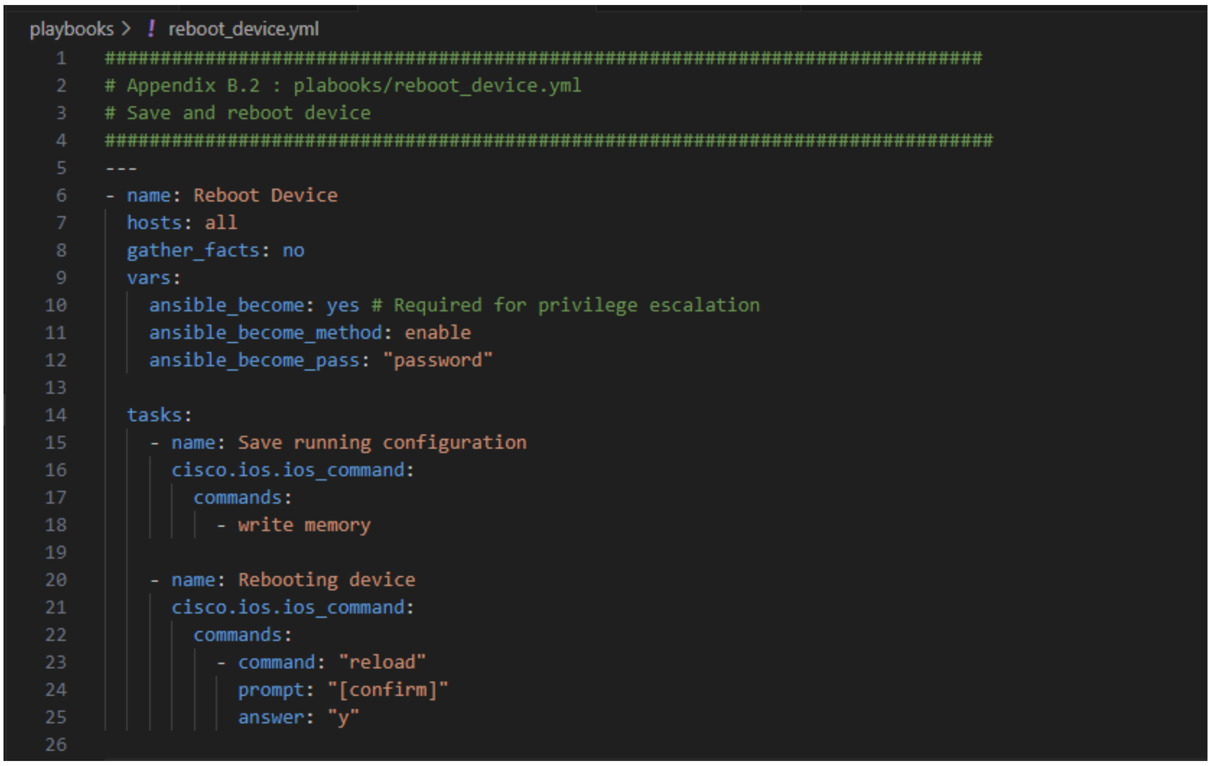
Figure 5 shows the *get_status.yml* playbook, which retrieves device status information from Cisco Catalyst switches.


playbooks > ! get_status.yml

```yaml
###############################################################################
# Appendix B.1 : Plabooks/get_status.yml
#Use on the Home page to get the status of devices. The retrived information is
#used to update switch info such as model, firmware version etc.
################################################################################
---
- name: Gather device status
  #hosts: "{{ target_host }}"
  hosts: "all"
  gather_facts: no
  tasks:
    # Step 1: Check if the host is reachable
    - name: Check reachability
      ansible.builtin.ping:
      register: ping_result
      ignore_errors: yes

    - name: Set host status based on ping result
      set_fact:
        host_status: "{{ 'Online' if ping_result.ping is defined else 'Offline' }}"

    # Step 2: Conditional task execution if the host is online
    - name: Gather device facts if online
      cisco.ios.ios_facts:
      when: host_status == "Online"

    - name: Get uptime if online
      cisco.ios.ios_command:
        commands:
          - show version | include uptime
      register: uptime_output
      when: host_status == "Online"

    # Step 3: Display the simplified status summary
    - name: Display status summary
      debug:
        msg:
          - "Host Status: {{ host_status }}"
          - "Uptime: {{ uptime_output.stdout[0] if uptime_output is defined and host_status == 'Online' else 'N/A' }}"
          - "Model: {{ ansible_net_model | default('N/A') if host_status == 'Online' else 'N/A' }}"
          - "Firmware Version: {{ ansible_net_version | default('N/A') if host_status == 'Online' else 'N/A' }}"

```

Figure 5: Get Status Playbook

Figure 6 illustrates the *reboot_device.yml* playbook, designed to reboot Cisco Catalyst switches. This playbook ensures a secure SSH connection and executes the necessary commands to safely reboot the devices.


playbooks > ! reboot_device.yml

```
#################################################################################
# Appendix B.2 : plabooks/reboot_device.yml
# Save and reboot device
##################################################################################
---
- name: Reboot Device
  hosts: all
  gather_facts: no
  vars:
    ansible_become: yes # Required for privilege escalation
    ansible_become_method: enable
    ansible_become_pass: "password"

  tasks:
    - name: Save running configuration
      cisco.ios.ios_command:
        commands:
          - write memory

    - name: Rebooting device
      cisco.ios.ios_command:
        commands:
          - command: "reload"
            prompt: "[confirm]"
            answer: "y"

```

Figure 6: Reboot Device Playbook

Appendix B: Playbook Source Code Files on the CD contains the complete implementation of these playbooks and additional playbooks.

4.5 Frontend Implementation

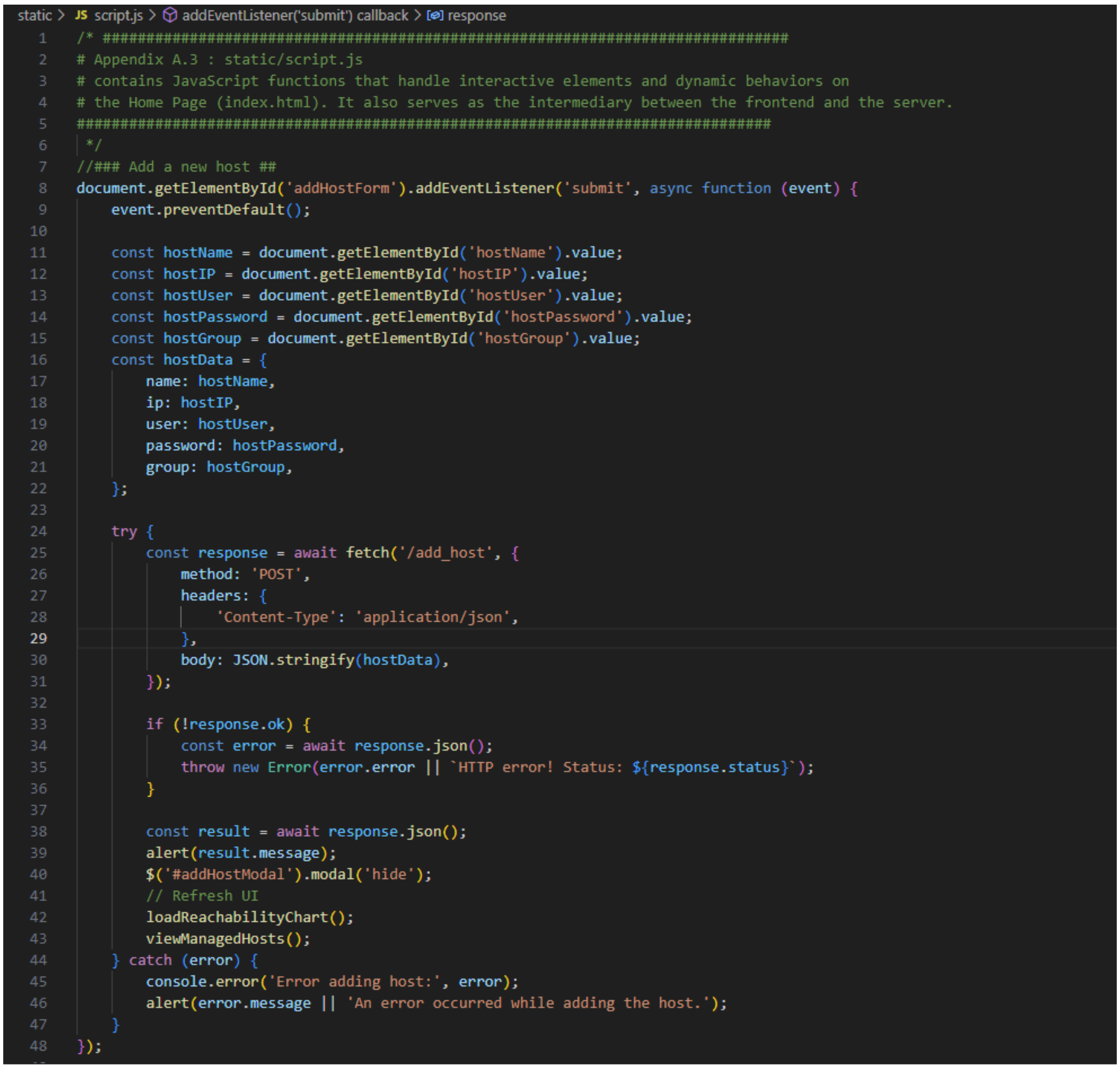
The frontend offers an intuitive interface for managing devices, monitoring their status, and accessing reports. It leverages HTML, CSS, JavaScript, and Bootstrap for seamless interactivity and responsiveness.

4.5.1 Dashboard Functionality and Behavior

The dashboard allows users to perform tasks such as:

- Adding new hosts and managing configurations.
- Viewing the status of devices through monitoring tools.
- Accessing logs and historical data for troubleshooting.

A sample code snippet for adding a new host is shown in Figure 7



static > script.js > addEventListener('submit') callback > response

```javascript
/* ###############################################################################
# Appendix A.3 : static/script.js
# contains JavaScript functions that handle interactive elements and dynamic behaviors on
# the Home Page (index.html). It also serves as the intermediary between the frontend and the server.
################################################################################
 */
//### Add a new host ##
document.getElementById('addHostForm').addEventListener('submit', async function (event) {
    event.preventDefault();

    const hostName = document.getElementById('hostName').value;
    const hostIP = document.getElementById('hostIP').value;
    const hostUser = document.getElementById('hostUser').value;
    const hostPassword = document.getElementById('hostPassword').value;
    const hostGroup = document.getElementById('hostGroup').value;
    const hostData = {
        name: hostName,
        ip: hostIP,
        user: hostUser,
        password: hostPassword,
        group: hostGroup,
    };

    try {
        const response = await fetch('/add_host', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(hostData),
        });

        if (!response.ok) {
            const error = await response.json();
            throw new Error(error.error || `HTTP error! Status: ${response.status}`);
        }

        const result = await response.json();
        alert(result.message);
        $('#addHostModal').modal('hide');
        // Refresh UI
        loadReachabilityChart();
        viewManagedHosts();
    } catch (error) {
        console.error('Error adding host:', error);
        alert(error.message || 'An error occurred while adding the host.');
    }
});
```

Figure 7. JavaScript Add Host Code Snippet

This snippet illustrates how the system processes user input to add a new host and fetches updated information. It highlights the on-demand interaction model**,** where updates to the interface occur after user-triggered actions, such as form submissions or button clicks.

4.5.2 Error Handling and Validation

The system employs:

- **Frontend Validation:** JavaScript ensures user input is accurate before submitting data to the server.
- **Backend Validation:** As mentioned, Flask performs additional checks to maintain data integrity and security.

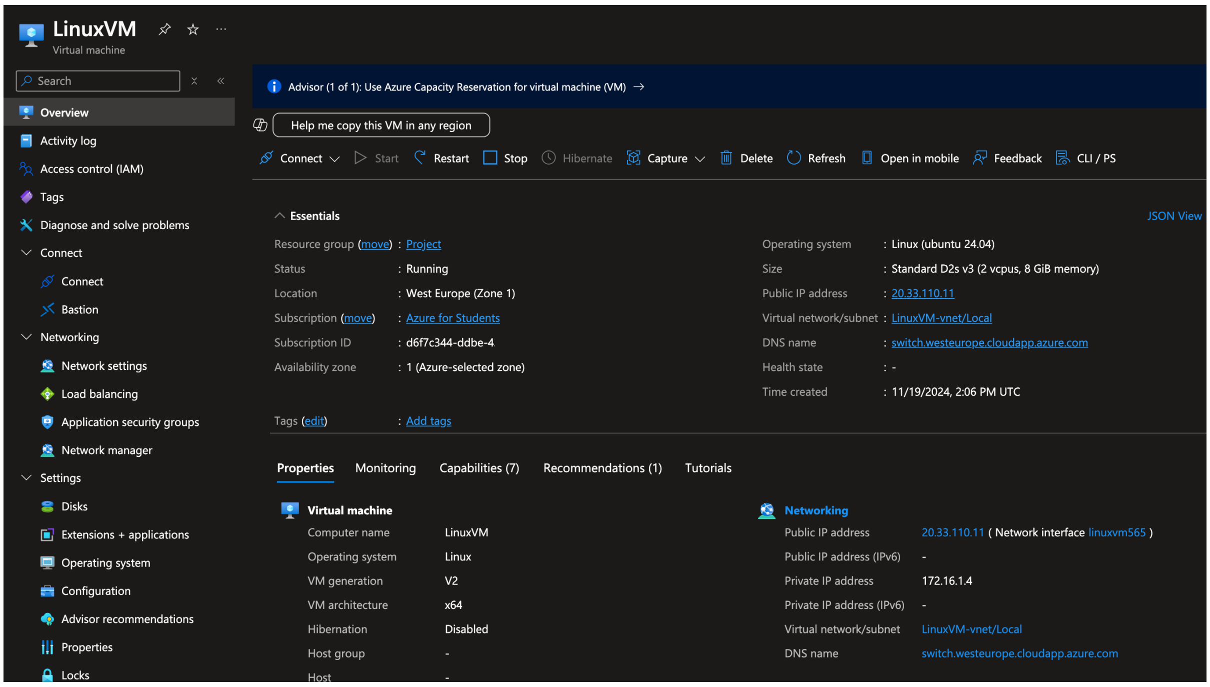
This layered approach minimizes errors and guides users with meaningful feedback. For further details on the frontend code, refer to Appendix A.3: script.js.

4.6 Deployment to Azure

The application is hosted on Microsoft Azure, leveraging its scalability, reliability, and secure infrastructure. The deployment process included the following key steps:

4.6.1 Creating an Azure Virtual Machine

An Ubuntu VM was provisioned to host the application. It was configured with 8GB RAM and two virtual CPUs to support concurrent user operations and Ansible tasks efficiently. Figure 8 provides an overview of the created VM.



Resource group (move) : Project
Status : Running
Location : West Europe (Zone 1)
Subscription (move) : Azure for Students
Subscription ID : d6f7c344-ddbe-4
Availability zone : 1 (Azure-selected zone)
Operating system : Linux (ubuntu 24.04)
Size : Standard D2s v3 (2 vcpus, 8 GiB memory)
Public IP address : 20.33.110.11
Virtual network/subnet : LinuxVM-vnet/Local
DNS name : switch.westeurope.cloudapp.azure.com
Health state : -
Time created : 11/19/2024, 2:06 PM UTC
Tags (edit) : Add tags

Virtual machine

Computer name	LinuxVM
Operating system	Linux
VM generation	V2
VM architecture	x64
Hibernation	Disabled
Host group	-
Host	-

Networking

Public IP address	20.33.110.11 (Network interface linuxvm565)
Public IP address (IPv6)	-
Private IP address	172.16.1.4
Private IP address (IPv6)	-
Virtual network/subnet	LinuxVM-vnet/Local
DNS name	switch.westeurope.cloudapp.azure.com

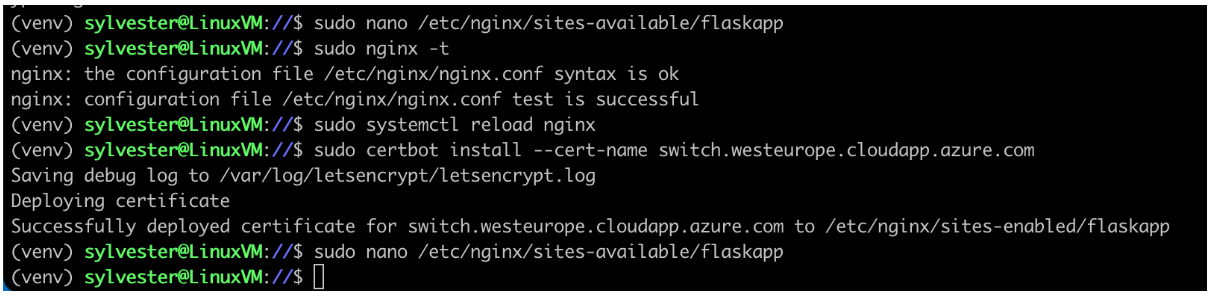
Figure 8: Ubuntu Linux VM Overview

4.6.2 Setting Up the Application

The application code was managed using GitHub, which enabled seamless version control and collaboration. The code and required dependencies were then directly deployed to the Azure VM by cloning the repository.

4.6.3 Configuring Nginx as a Reverse Proxy

Nginx was installed and configured to act as a reverse proxy for the Flask application. SSL/TLS certificates were configured using Let's Encrypt to enable secure communication. This setup aligns with modern security practices for web applications.

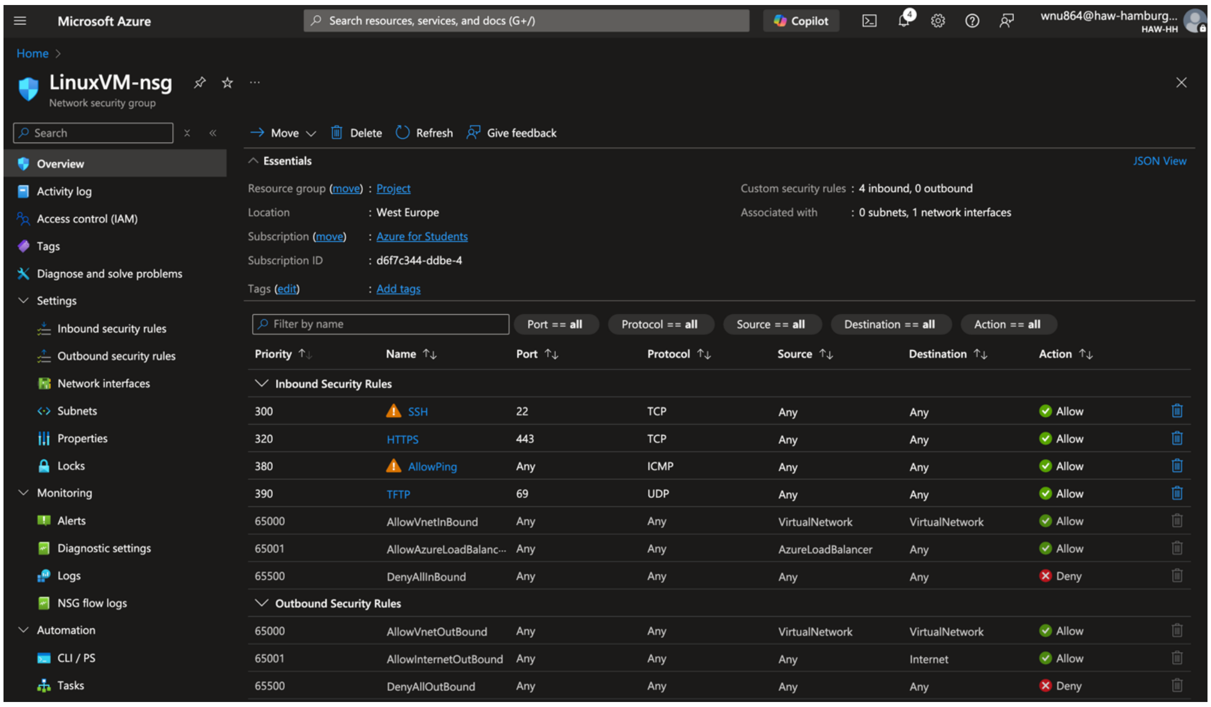


```
(venv) sylvester@LinuxVM://$ sudo nano /etc/nginx/sites-available/flaskapp
(venv) sylvester@LinuxVM://$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
(venv) sylvester@LinuxVM://$ sudo systemctl reload nginx
(venv) sylvester@LinuxVM://$ sudo certbot install --cert-name switch.westeurope.cloudapp.azure.com
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Deploying certificate
Successfully deployed certificate for switch.westeurope.cloudapp.azure.com to /etc/nginx/sites-enabled/flaskapp
(venv) sylvester@LinuxVM://$ sudo nano /etc/nginx/sites-available/flaskapp
(venv) sylvester@LinuxVM://$ 
```

Figure 9: Generating Server Certificate using Let's Encrypt

4.6.4 Establishing Secure Access

Azure Network Security Groups (NSGs) were configured to allow access only to essential ports, such as port 443 for HTTPS. OpenfortiVPN was used to establish a secure connection to the lab's SSL VPN, enabling remote access to switches from the Azure-hosted application. As shown in Figure 10, ports 22, 69, and 443 were enabled for external access and TFTP file transfer. Access will be restricted to the company's management network in the production environment, ensuring only authorized personnel can connect.



LinuxVM-nsg
Network security group

Essentials

Resource group (move) : Project
Location : West Europe
Subscription (move) : Azure for Students
Subscription ID : d6f7c344-ddbe-4
Tags (edit) : Add tags
Custom security rules : 4 inbound, 0 outbound
Associated with : 0 subnets, 1 network interfaces

Priority	Name	Port	Protocol	Source	Destination	Action
Inbound Security Rules						
300	SSH	22	TCP	Any	Any	Allow
320	HTTPS	443	TCP	Any	Any	Allow
380	AllowPing	Any	ICMP	Any	Any	Allow
390	TFTP	69	UDP	Any	Any	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalanc...	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny
Outbound Security Rules						
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
65500	DenyAllOutBound	Any	Any	Any	Any	Deny

Figure 10: NSG Configuration

4.7 Security Measures

Security was a critical consideration during the system's implementation, and several measures were adopted to protect data and ensure secure operations. For user authentication, passwords were securely hashed using SHA-256 before being stored in the database, eliminating the risk of saving plaintext passwords. Flask-Login managed user sessions securely, providing mechanisms for session handling, login tracking, and authentication.

Communication between the system and managed switches was conducted exclusively using SSH, which ensured data encryption during transmission and protected against interception. To secure user interactions with the application, Nginx was configured as a reverse proxy with enforced HTTPS, using SSL/TLS certificates to enable end-to-end encryption.

A robust logging mechanism was also implemented to monitor critical system activities. User logins were tracked to identify unauthorized access, while every playbook execution request was logged to create a traceable audit trail. Additionally, warnings and errors were logged to monitor potential issues and anomalies, aiding in troubleshooting. Logging was managed carefully to avoid excessive information at the INFO level, prioritizing relevant details for security and operational monitoring.

These combined measures significantly enhanced the system's security posture, ensuring the integrity, confidentiality, and accountability of data and operations.

Chapter 5: Testing and Evaluation

This chapter evaluates the functionality, performance, and reliability of the Centralized Network Switch Management System through a series of structured test cases. Testing ensures the system meets the design objectives and performs as expected under various scenarios. Before initiating the tests, a few steps were required to prepare the environment. These steps are outlined in Appendix A.4: README.md.

5.1 Application Initialization

The database was preloaded with a default *admin* user to allow immediate access to the application upon startup.

Starting the Application: To start the application, follow these steps:

- SSH to the Linux VM

  ssh username@switch.westeurope.cloudapp.azure.com or IP address

- Create a virtual environment (if not already created) using:

  python3 -m venv venv

- Activate the virtual environment:

  source venv/bin/activate

- Install all necessary dependencies listed in the requirements.txt file (see Appendix A.5: requirements.txt):
  pip install -r requirements.txt

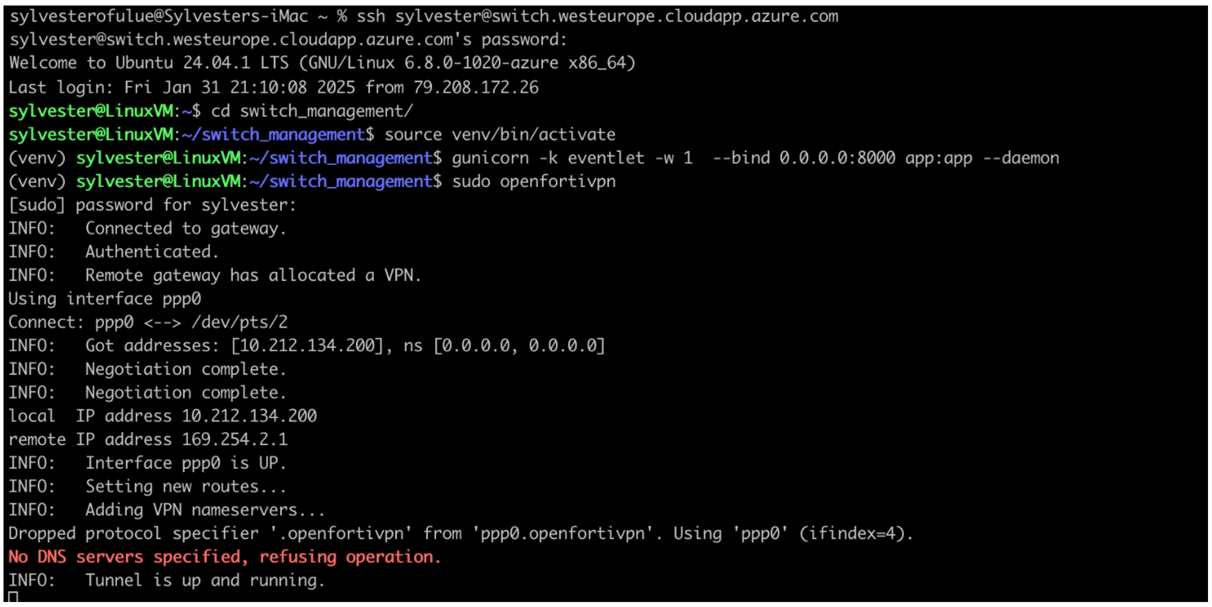
- Launch the Flask application with Gunicorn:

- *gunicorn -k eventlet -w 1 --bind 0.0.0.0:8000 app:app*

- (Optional) Establish a VPN connection for secure access to a network

  sudo openfortivpn

Once these steps are complete, the application can be tested through a web browser.



```
sylvesterofulue@Sylvesters-iMac ~ % ssh sylvester@switch.westeurope.cloudapp.azure.com
sylvester@switch.westeurope.cloudapp.azure.com's password:
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1020-azure x86_64)
Last login: Fri Jan 31 21:10:08 2025 from 79.208.172.26
sylvester@LinuxVM:~$ cd switch_management/
sylvester@LinuxVM:~/switch_management$ source venv/bin/activate
(venv) sylvester@LinuxVM:~/switch_management$ gunicorn -k eventlet -w 1  --bind 0.0.0.0:8000 app:app --daemon
(venv) sylvester@LinuxVM:~/switch_management$ sudo openfortivpn
[sudo] password for sylvester:
INFO:   Connected to gateway.
INFO:   Authenticated.
INFO:   Remote gateway has allocated a VPN.
Using interface ppp0
Connect: ppp0 <--> /dev/pts/2
INFO:   Got addresses: [10.212.134.200], ns [0.0.0.0, 0.0.0.0]
INFO:   Negotiation complete.
INFO:   Negotiation complete.
local  IP address 10.212.134.200
remote IP address 169.254.2.1
INFO:   Interface ppp0 is UP.
INFO:   Setting new routes...
INFO:   Adding VPN nameservers...
Dropped protocol specifier '.openfortivpn' from 'ppp0.openfortivpn'. Using 'ppp0' (ifindex=4).
No DNS servers specified, refusing operation.
INFO:   Tunnel is up and running.
```

Figure 11: Starting the Application from a Remote Terminal

5.2 Test Cases

The following test cases were designed to validate key functionalities and measure system performance:

5.2.1 Login Functionality

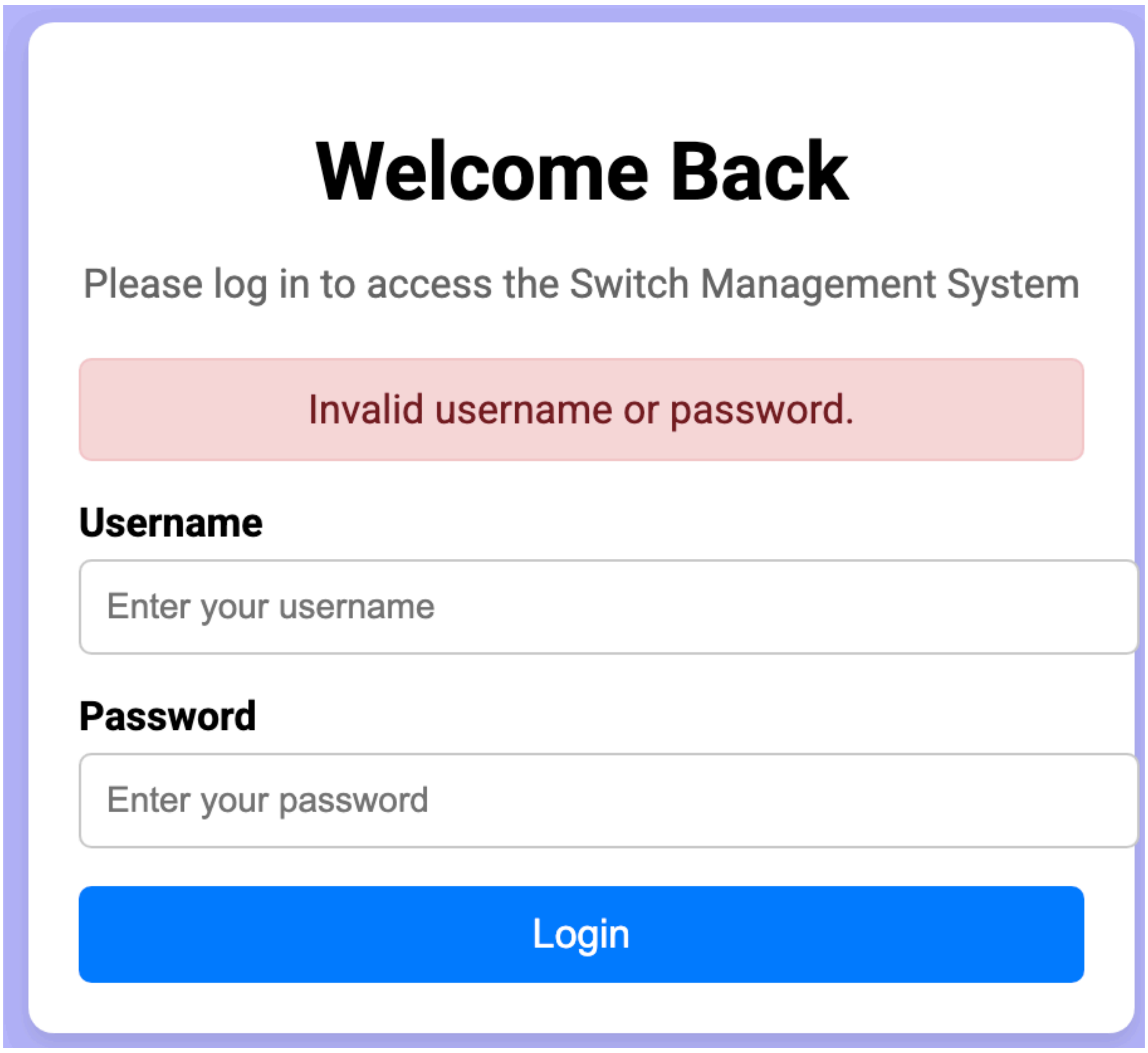
Objective: To ensure the system correctly authenticates users.

Test: Login attempts were made using incorrect and correct credentials.

Expected Outcome:

- Incorrect credentials: Display an error message without granting access.
- Correct credentials: Redirect to the dashboard with appropriate session management.

Result: Authentication worked as expected, preventing unauthorized access and correctly handling valid logins.


Welcome Back

Please log in to access the Switch Management System

Invalid username or password.

Username

Enter your username

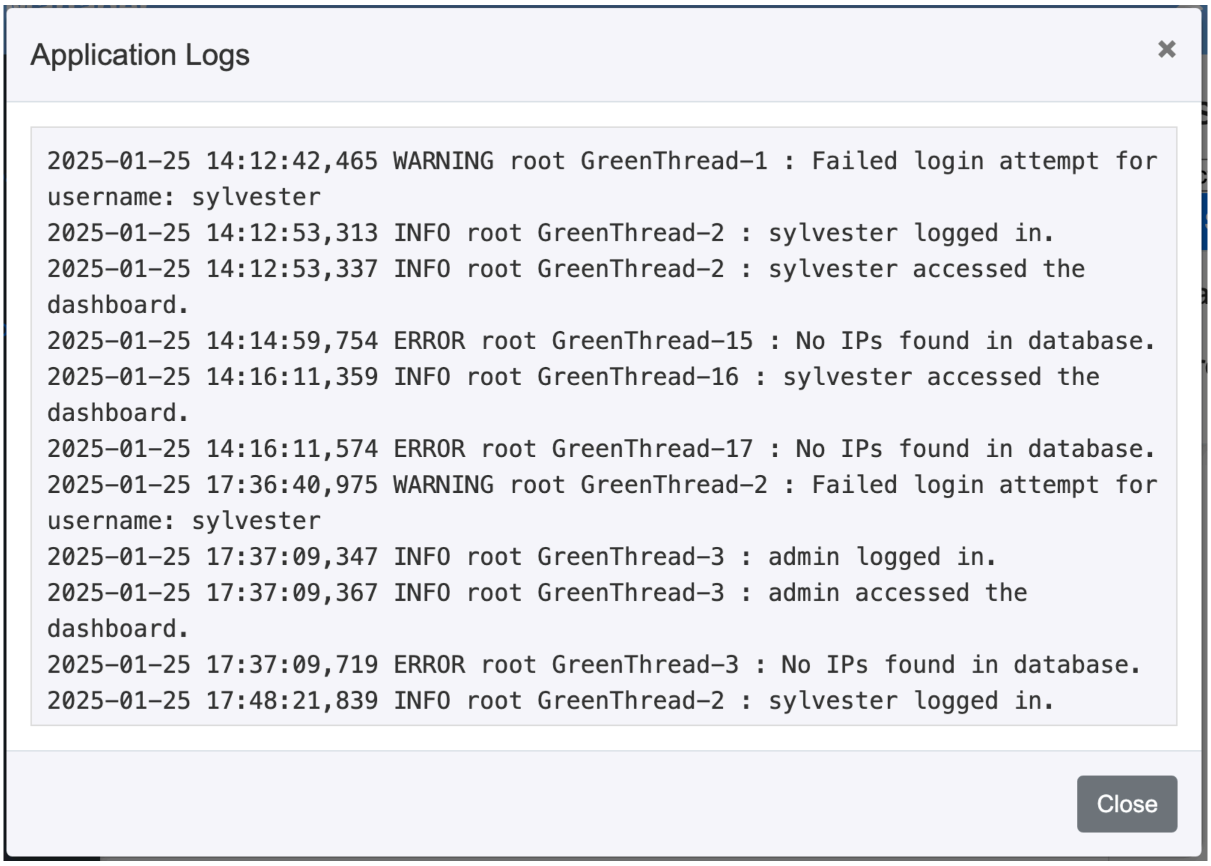
Password

Enter your password

Login

Figure 12: Failed Login Attempt

Figure 13 below shows a screenshot of the application log. The failed and successful logins were correctly logged, which can be helpful in cases of security breaches.



```
2025-01-25 14:12:42,465 WARNING root GreenThread-1 : Failed login attempt for
username: sylvester
2025-01-25 14:12:53,313 INFO root GreenThread-2 : sylvester logged in.
2025-01-25 14:12:53,337 INFO root GreenThread-2 : sylvester accessed the
dashboard.
2025-01-25 14:14:59,754 ERROR root GreenThread-15 : No IPs found in database.
2025-01-25 14:16:11,359 INFO root GreenThread-16 : sylvester accessed the
dashboard.
2025-01-25 14:16:11,574 ERROR root GreenThread-17 : No IPs found in database.
2025-01-25 17:36:40,975 WARNING root GreenThread-2 : Failed login attempt for
username: sylvester
2025-01-25 17:37:09,347 INFO root GreenThread-3 : admin logged in.
2025-01-25 17:37:09,367 INFO root GreenThread-3 : admin accessed the
dashboard.
2025-01-25 17:37:09,719 ERROR root GreenThread-3 : No IPs found in database.
2025-01-25 17:48:21,839 INFO root GreenThread-2 : sylvester logged in.
```

Figure 13: Application Log Showing the Failed and Successful Login Attempts

5.2.2 Adding Switches

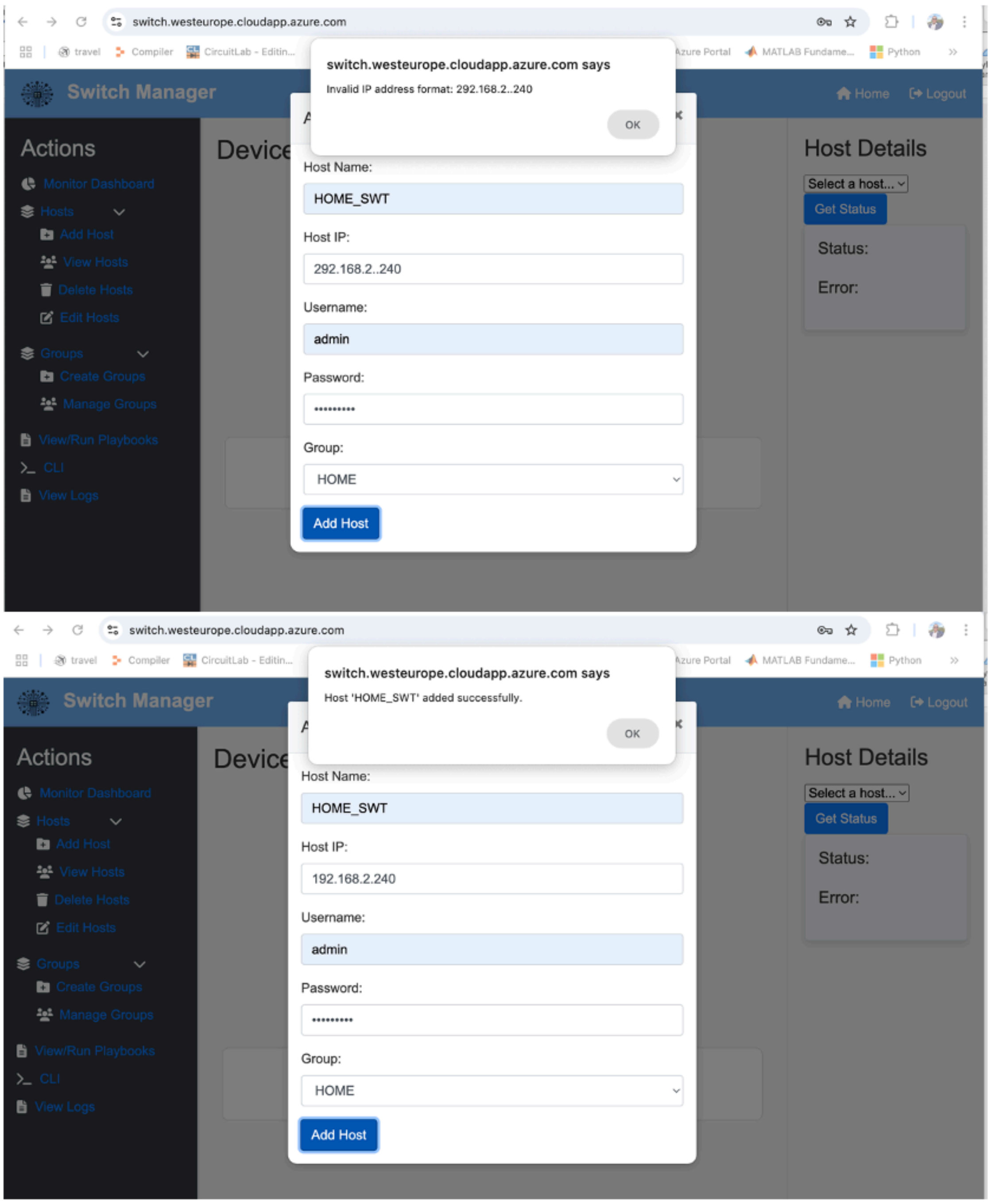
Objective: Verify the system's ability to add switches with valid IPs.

Test: New switches with invalid and valid IPs were added to the system.

Expected Outcome:

- Invalid entries: Display an error message, and no database changes occur.
- Valid entries: Switches are added to the database and displayed in the managed switches list.

Result: Switch addition was successful, with appropriate validation handling. In Figure 14 below, an attempt was made to add a switch with an IP address of 292.168.2..240. Since it failed validation, it was rejected with an error, allowing the user to add a switch with the acceptable IP address format. Other checks were implemented to avoid duplicate hostnames and IP addresses.


switch.westeurope.cloudapp.azure.com says

Invalid IP address format: 292.168.2..240

Host Name: HOME_SWT

Host IP: 292.168.2..240

Username: admin

Password: •••••••••

Group: HOME

switch.westeurope.cloudapp.azure.com says

Host 'HOME_SWT' added successfully.

Host Name: HOME_SWT

Host IP: 192.168.2.240

Username: admin

Password: •••••••••

Group: HOME

Figure 14: Adding Switch with an invalid and a Valid IP Address

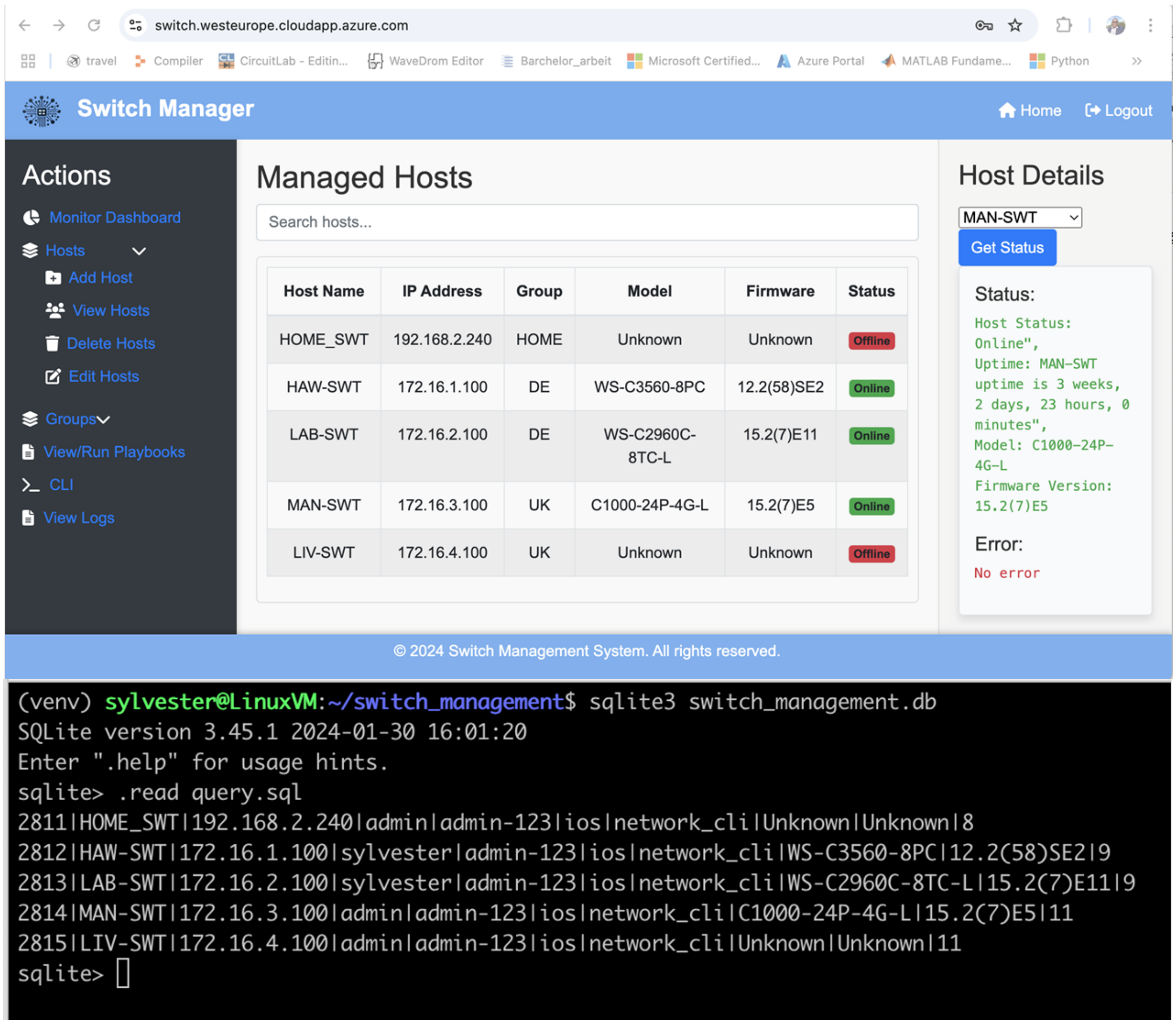
5.2.3 Database Operations

Objective: To validate CRUD operations for switches.

Test Procedure: Switches were added, edited, and deleted from the GUI.

Expected Outcome: The database reflects all changes accurately.

Result: All operations were successful, and updates were immediately reflected in the GUI and Database. Figure 15 to Figure 17 compare the GUI displayed with the database to verify that changes were correctly reflected.



Host Name	IP Address	Group	Model	Firmware	Status
HOME_SWT	192.168.2.240	HOME	Unknown	Unknown	Offline
HAW-SWT	172.16.1.100	DE	WS-C3560-8PC	12.2(58)SE2	Online
LAB-SWT	172.16.2.100	DE	WS-C2960C-8TC-L	15.2(7)E11	Online
MAN-SWT	172.16.3.100	UK	C1000-24P-4G-L	15.2(7)E5	Online
LIV-SWT	172.16.4.100	UK	Unknown	Unknown	Offline

Status:

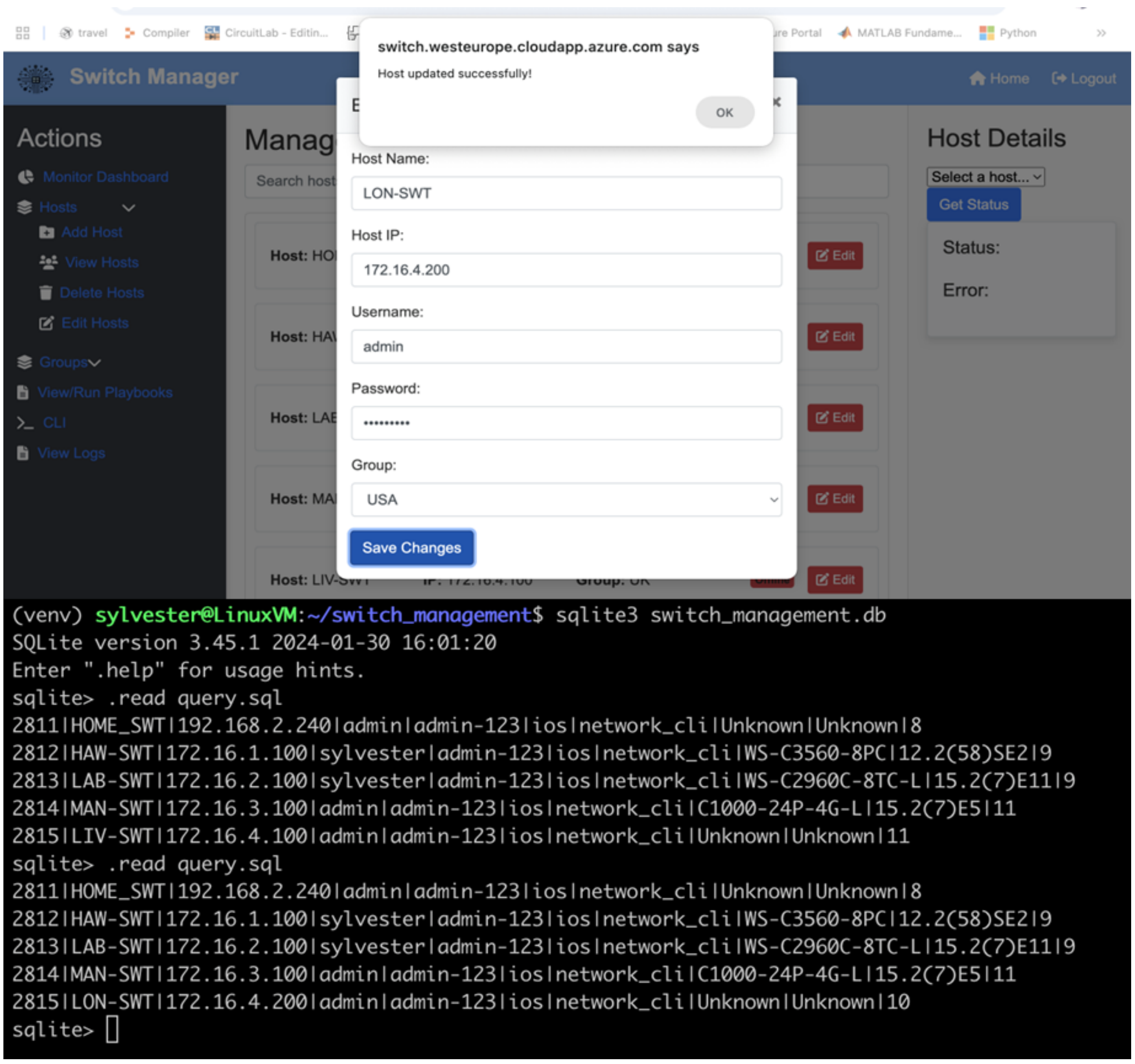
```
Host Status:
Online",
Uptime: MAN-SWT
uptime is 3 weeks,
2 days, 23 hours, 0
minutes",
Model: C1000-24P-
4G-L
Firmware Version:
15.2(7)E5
```

Error:

```
No error
```

```
(venv) sylvester@LinuxVM:~/switch_management$ sqlite3 switch_management.db
SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
sqlite> .read query.sql
2811|HOME_SWT|192.168.2.240|admin|admin-123|ios|network_cli|Unknown|Unknown|8
2812|HAW-SWT|172.16.1.100|sylvester|admin-123|ios|network_cli|WS-C3560-8PC|12.2(58)SE2|9
2813|LAB-SWT|172.16.2.100|sylvester|admin-123|ios|network_cli|WS-C2960C-8TC-L|15.2(7)E11|9
2814|MAN-SWT|172.16.3.100|admin|admin-123|ios|network_cli|C1000-24P-4G-L|15.2(7)E5|11
2815|LIV-SWT|172.16.4.100|admin|admin-123|ios|network_cli|Unknown|Unknown|11
sqlite> 
```

Figure 15: Added Switches verified on the Database



switch.westeurope.cloudapp.azure.com says

Host updated successfully!

OK

Host Name: LON-SWT

Host IP: 172.16.4.200

Username: admin

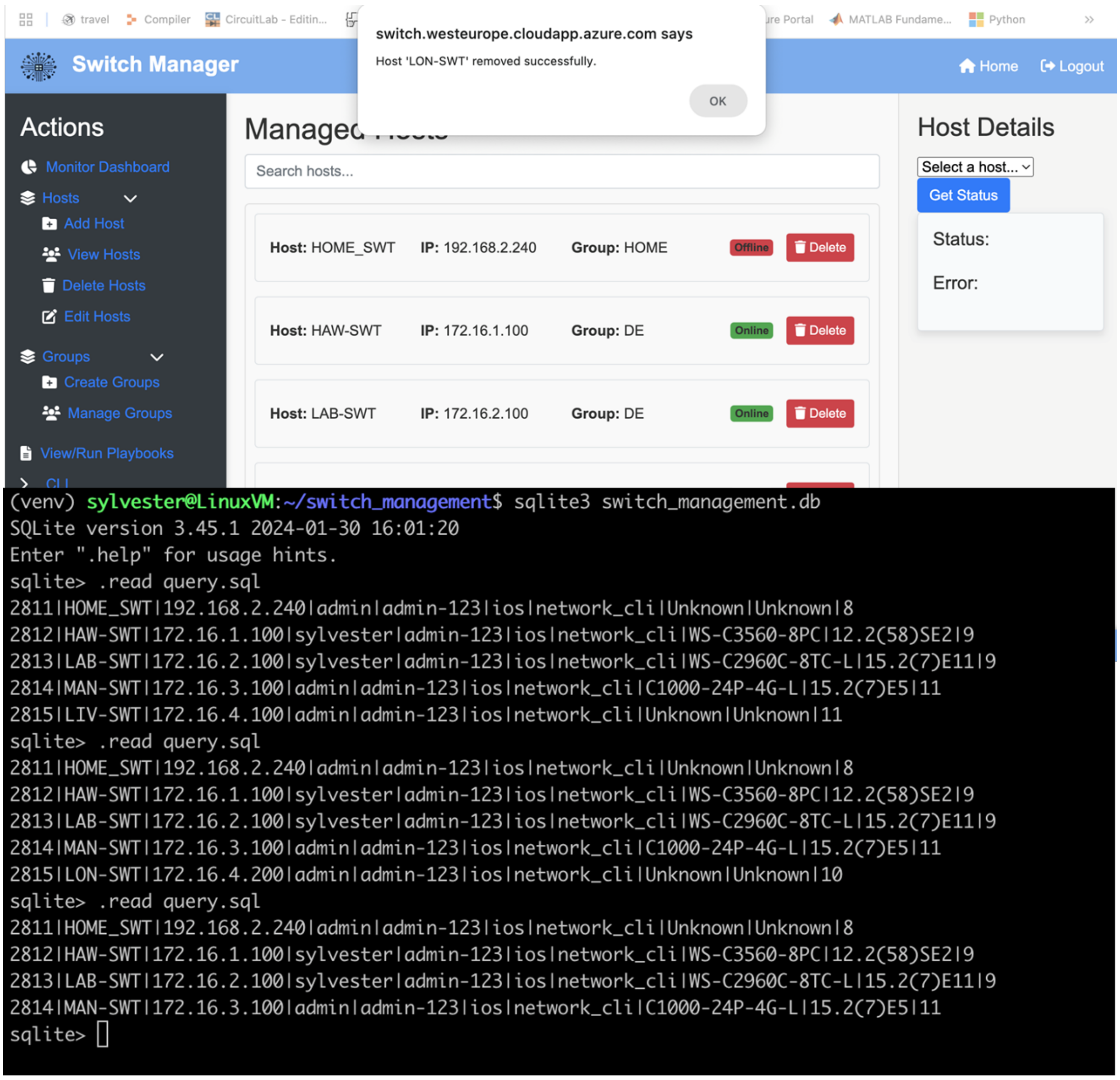
Password: •••••••••

Group: USA

Save Changes

```
(venv) sylvester@LinuxVM:~/switch_management$ sqlite3 switch_management.db
SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
sqlite> .read query.sql
2811|HOME_SWT|192.168.2.240|admin|admin-123|ios|network_cli|Unknown|Unknown|8
2812|HAW-SWT|172.16.1.100|sylvester|admin-123|ios|network_cli|WS-C3560-8PC|12.2(58)SE2|9
2813|LAB-SWT|172.16.2.100|sylvester|admin-123|ios|network_cli|WS-C2960C-8TC-L|15.2(7)E11|9
2814|MAN-SWT|172.16.3.100|admin|admin-123|ios|network_cli|C1000-24P-4G-L|15.2(7)E5|11
2815|LIV-SWT|172.16.4.100|admin|admin-123|ios|network_cli|Unknown|Unknown|11
sqlite> .read query.sql
2811|HOME_SWT|192.168.2.240|admin|admin-123|ios|network_cli|Unknown|Unknown|8
2812|HAW-SWT|172.16.1.100|sylvester|admin-123|ios|network_cli|WS-C3560-8PC|12.2(58)SE2|9
2813|LAB-SWT|172.16.2.100|sylvester|admin-123|ios|network_cli|WS-C2960C-8TC-L|15.2(7)E11|9
2814|MAN-SWT|172.16.3.100|admin|admin-123|ios|network_cli|C1000-24P-4G-L|15.2(7)E5|11
2815|LON-SWT|172.16.4.200|admin|admin-123|ios|network_cli|Unknown|Unknown|10
sqlite> 
```

Figure 16: Updating a Host



switch.westeurope.cloudapp.azure.com says

Host 'LON-SWT' removed successfully.

Host: HOME_SWT	**IP:** 192.168.2.240	**Group:** HOME	Offline	Delete
Host: HAW-SWT	**IP:** 172.16.1.100	**Group:** DE	Online	Delete
Host: LAB-SWT	**IP:** 172.16.2.100	**Group:** DE	Online	Delete

```
(venv) sylvester@LinuxVM:~/switch_management$ sqlite3 switch_management.db
SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
sqlite> .read query.sql
2811|HOME_SWT|192.168.2.240|admin|admin-123|ios|network_cli|Unknown|Unknown|8
2812|HAW-SWT|172.16.1.100|sylvester|admin-123|ios|network_cli|WS-C3560-8PC|12.2(58)SE2|9
2813|LAB-SWT|172.16.2.100|sylvester|admin-123|ios|network_cli|WS-C2960C-8TC-L|15.2(7)E11|9
2814|MAN-SWT|172.16.3.100|admin|admin-123|ios|network_cli|C1000-24P-4G-L|15.2(7)E5|11
2815|LIV-SWT|172.16.4.100|admin|admin-123|ios|network_cli|Unknown|Unknown|11
sqlite> .read query.sql
2811|HOME_SWT|192.168.2.240|admin|admin-123|ios|network_cli|Unknown|Unknown|8
2812|HAW-SWT|172.16.1.100|sylvester|admin-123|ios|network_cli|WS-C3560-8PC|12.2(58)SE2|9
2813|LAB-SWT|172.16.2.100|sylvester|admin-123|ios|network_cli|WS-C2960C-8TC-L|15.2(7)E11|9
2814|MAN-SWT|172.16.3.100|admin|admin-123|ios|network_cli|C1000-24P-4G-L|15.2(7)E5|11
2815|LON-SWT|172.16.4.200|admin|admin-123|ios|network_cli|Unknown|Unknown|10
sqlite> .read query.sql
2811|HOME_SWT|192.168.2.240|admin|admin-123|ios|network_cli|Unknown|Unknown|8
2812|HAW-SWT|172.16.1.100|sylvester|admin-123|ios|network_cli|WS-C3560-8PC|12.2(58)SE2|9
2813|LAB-SWT|172.16.2.100|sylvester|admin-123|ios|network_cli|WS-C2960C-8TC-L|15.2(7)E11|9
2814|MAN-SWT|172.16.3.100|admin|admin-123|ios|network_cli|C1000-24P-4G-L|15.2(7)E5|11
sqlite> 
```

Figure 17: Deleting a Host

5.2.4 Device Dashboard Monitoring

Objective: Verify whether the dashboard displays the number of managed hosts and the count of the online and offline hosts.

Test Procedure: Log in to the switch management system or refresh the current session session.

Expected Outcome: A doughnut chart should show online hosts in green and offline hosts in red. Additionally, it should display the total number of managed hosts.

Result: In Figure 18, the device monitoring dashboard accurately displays the number of managed hosts. The green section of the chart represents online hosts, and the red section represents offline hosts.

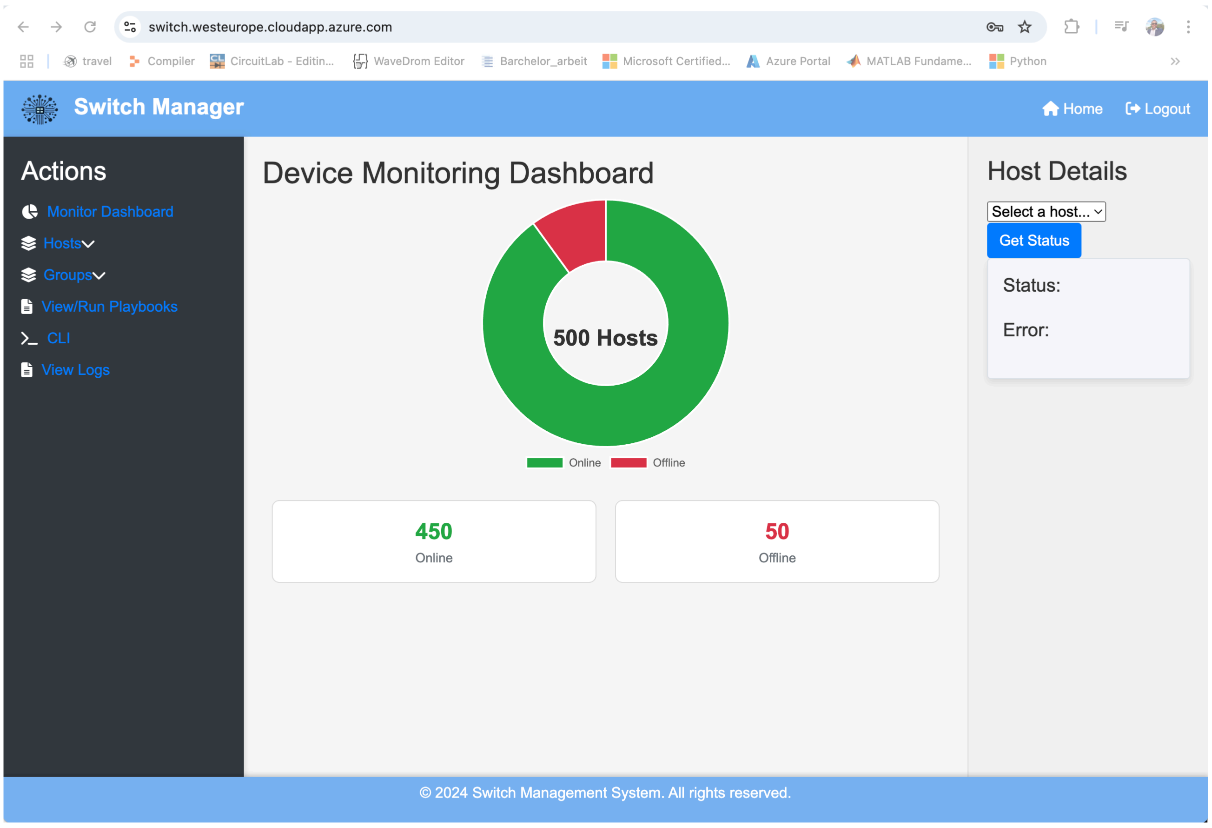


Figure 18: Device Monitoring Dashboard

5.2.5 Scalability Testing

To ensure the system could handle growing demands, scalability testing was conducted to measure response times and performance as the number of managed devices increased.

Test Environment:

Server - Ubuntu Virtual Machine on Azure: 2 vCPUs, 8 GB RAM.

Client – Google Chrome on Ubuntu Machine: 4 CPUs, 4 GB RAM.

The application was configured with 50 threads and two ping retries.

Postman was used to simulate and analyze API requests.

5.2.5.1 Load Test with Online Hosts

This test evaluated the system's ability to manage requests for online and reachable devices.

Test Procedure: Three Cisco switches were provided in the lab. Additional online switches were simulated with the hosts' loopback addresses. Five reachability (load) tests were performed on up to 1,000 hosts, and their average was recorded. The screenshots of the measurements are provided in Appendix C.1: Load Test for Online Switches.

Expected Outcome:

- The system should remain responsive without crashing.
- Loading times should be within 1000ms.

Results: The system performed as expected, maintaining an acceptable average response time of 720ms for 500 online devices.

Figure 19 shows the system response time when all the added hosts are online.

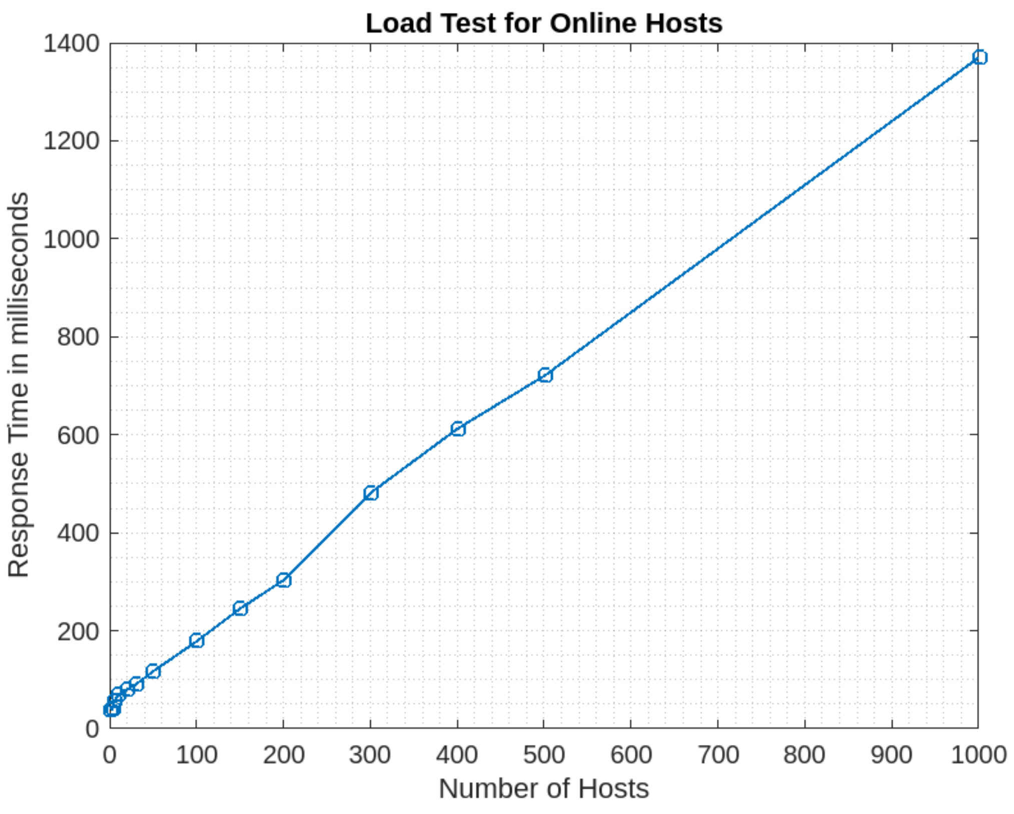


Figure 19: Plot of System Response to Online Host

5.2.5.2 Load Test with Offline Hosts

This test evaluated the system's ability to manage requests for offline and unreachable devices.

Test Procedure: Three Cisco switches were provided in the lab. Additional offline switches were simulated with an off-net of 10.0.0.0/8.

Five reachability (load) tests were performed on up to 1,000 hosts, and their average was recorded. The screenshots of the measurements are provided in Appendix C.2: Load Test for Offline Switches.

Expected Outcome:

- The system should remain responsive without crashing.
- Loading times should be within 1000ms.

Results: The system experienced slower response times for offline hosts, with a response time of 10 seconds for 500 offline hosts.

Figure 20 shows the system response time when all hosts are offline.

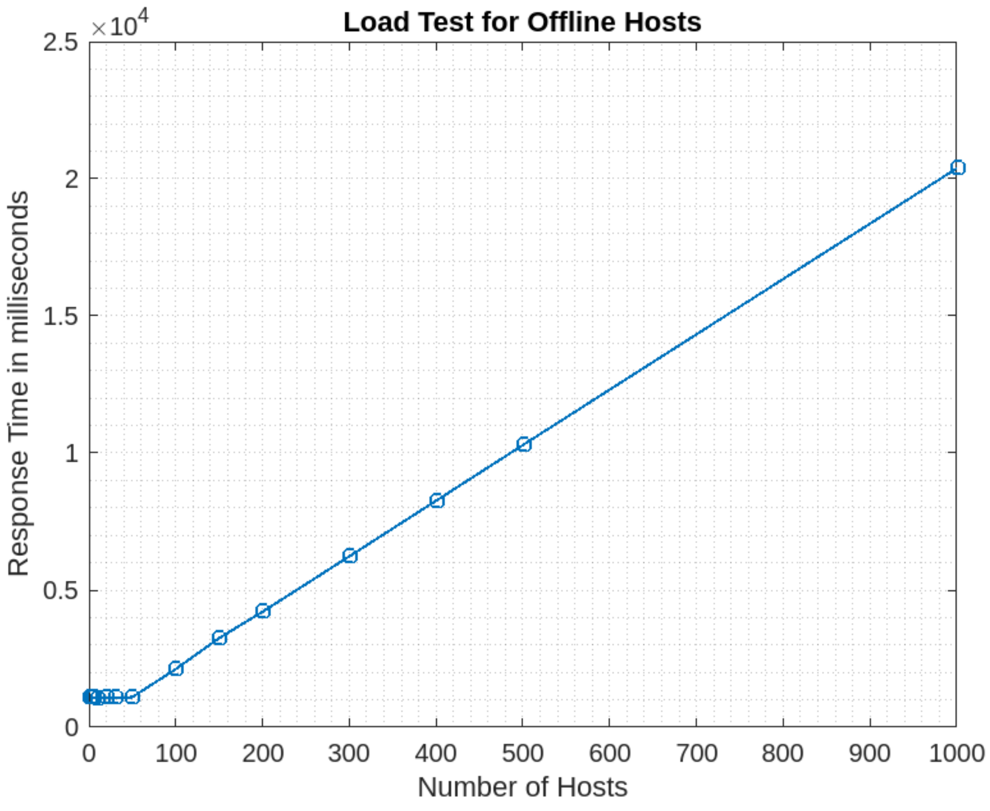


Figure 20: Plot of System Response to Offline Hosts

5.2.6 Threading Optimization

Objective: Investigate the impact of threading on system performance when managing offline devices.

Test Procedure: The threading parameter (max_threads) was adjusted incrementally (e.g., 2, 4, 8, 16…) to evaluate its effect on response times.

Observation:

Response times improved proportionally as the number of threads increased.

However, increasing the thread count beyond 50 was not recommended due to resource constraints. Also, the system is not designed to manage offline hosts. If all hosts are offline, it is likely because the system is offline or off-net with the managed network environment.

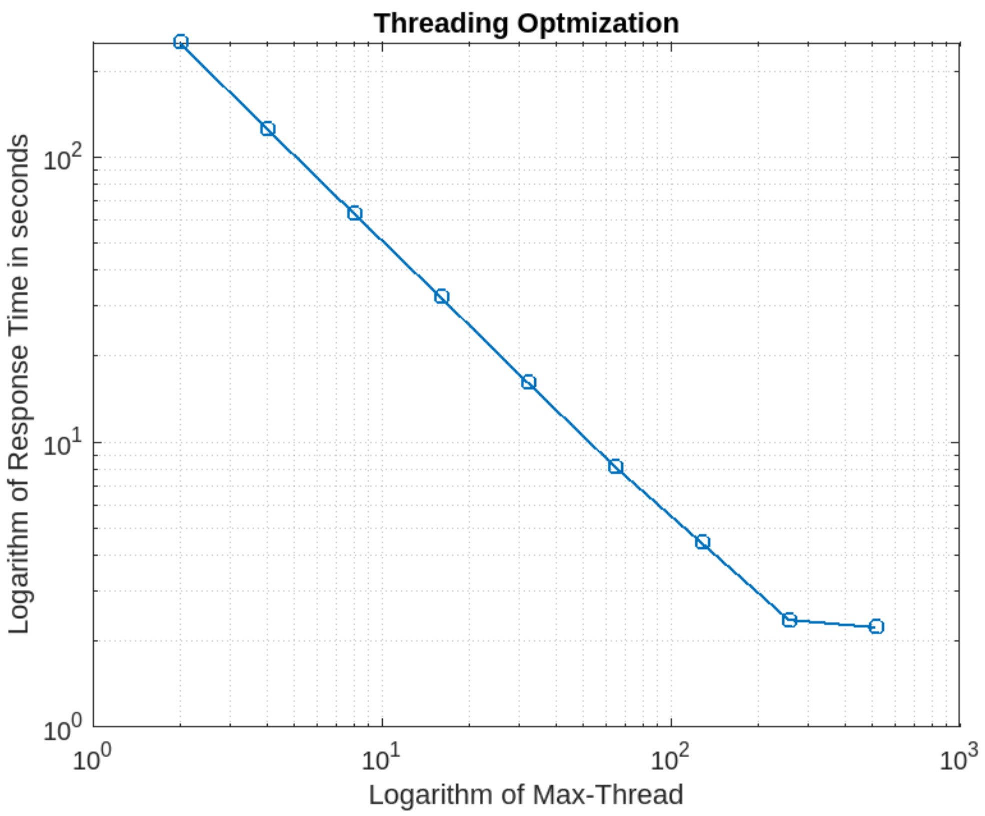


Figure 21: Threading Optimization

5.2.7 Playbook Execution (Single Host)

Objective: To verify playbook execution for individual hosts.

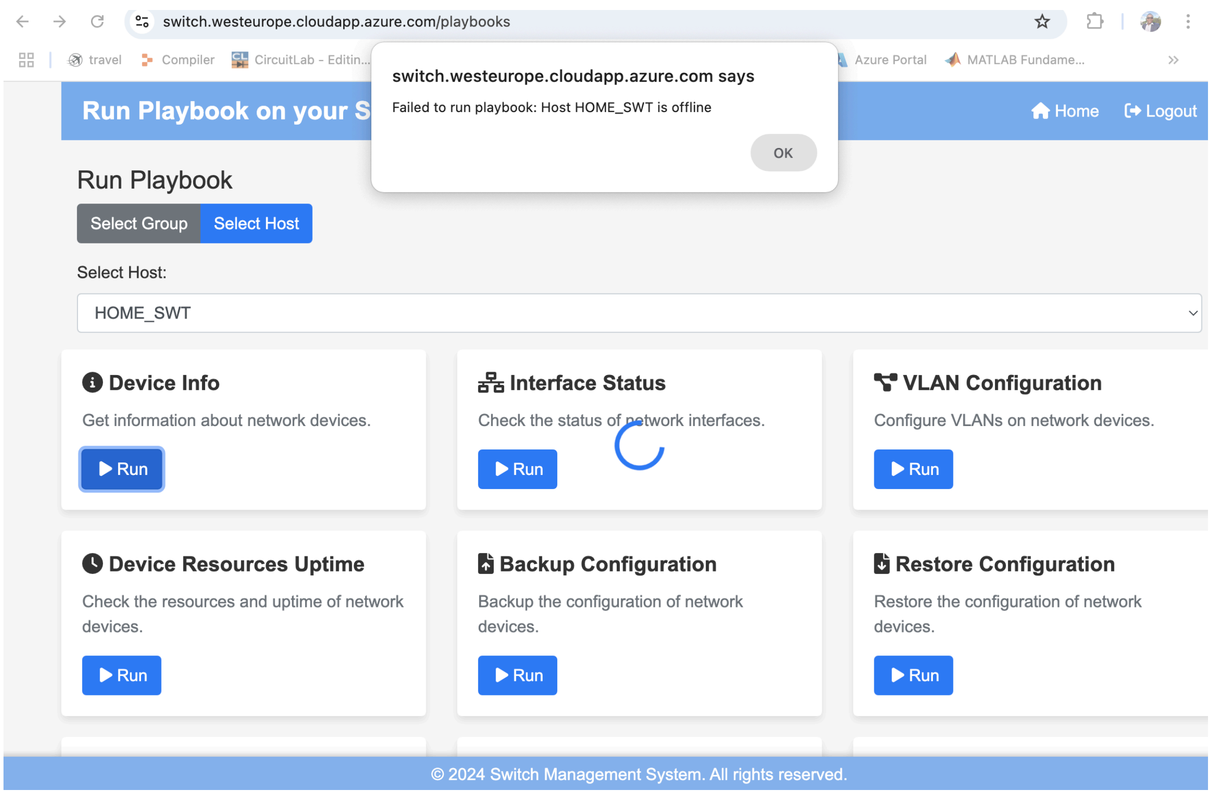
Test: Playbooks were run on:

1. An offline switch.
2. An online switch.

Expected Outcome:

- Offline: The system halts execution and displays an error message.
- Online: Playbook executes successfully.

Result: The system behaved as expected, ensuring pre-checks for connectivity. When a host is offline, playbook execution fails but runs on an online host.



switch.westeurope.cloudapp.azure.com/playbooks

switch.westeurope.cloudapp.azure.com says

Failed to run playbook: Host HOME_SWT is offline

Run Playbook

Select Host:

HOME_SWT

Device Info

Get information about network devices.

Interface Status

Check the status of network interfaces.

VLAN Configuration

Configure VLANs on network devices.

Device Resources Uptime

Check the resources and uptime of network devices.

Backup Configuration

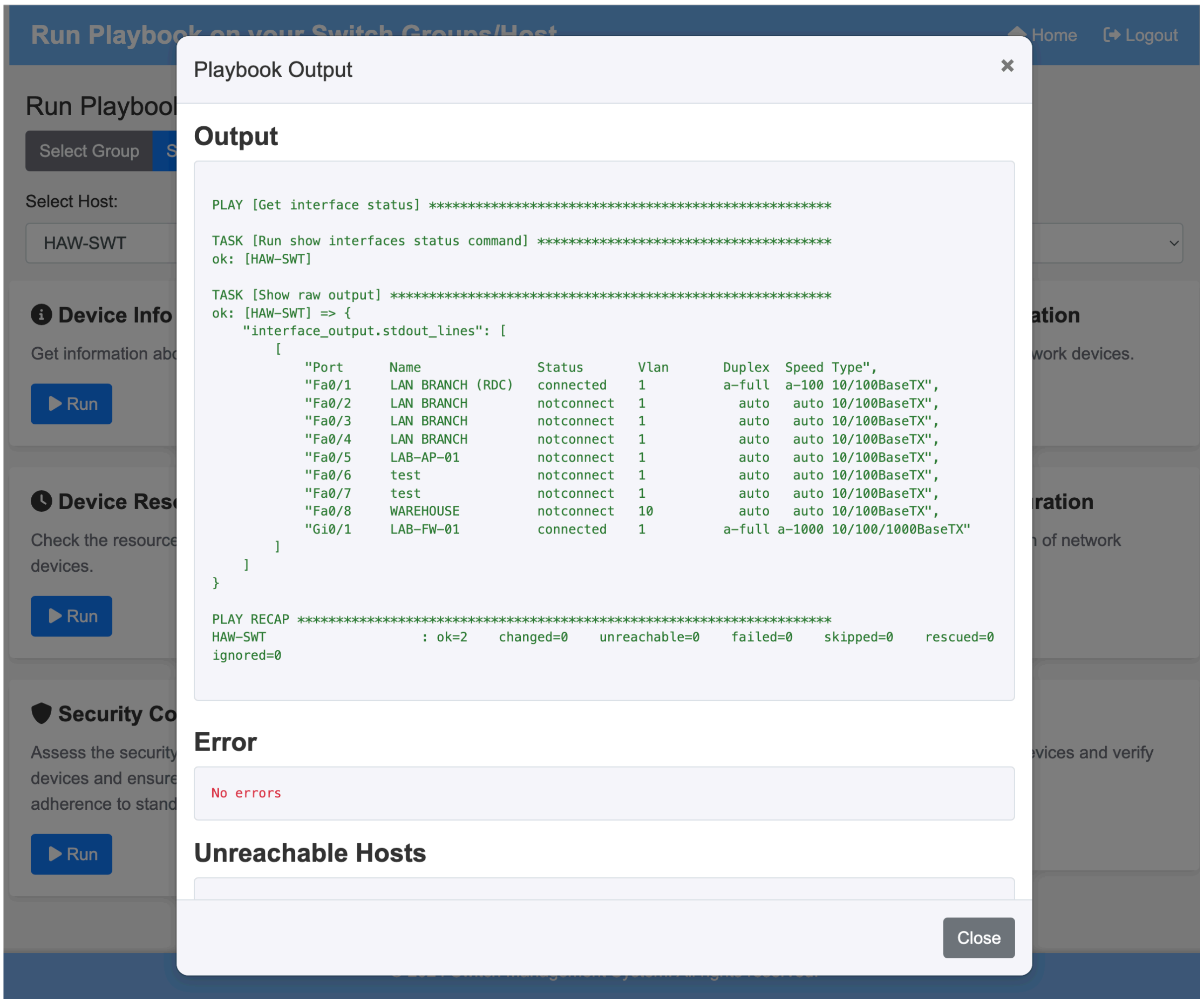
Backup the configuration of network devices.

Restore Configuration

Restore the configuration of network devices.

© 2024 Switch Management System. All rights reserved.

Figure 22: Executing Playbook on an Offline Host



Playbook Output

Output

```
PLAY [Get interface status] ****************************************************

TASK [Run show interfaces status command] **************************************
ok: [HAW-SWT]

TASK [Show raw output] *********************************************************
ok: [HAW-SWT] => {
    "interface_output.stdout_lines": [
        [
            "Port      Name               Status       Vlan       Duplex  Speed Type",
            "Fa0/1     LAN BRANCH (RDC)   connected    1          a-full  a-100 10/100BaseTX",
            "Fa0/2     LAN BRANCH         notconnect   1            auto   auto 10/100BaseTX",
            "Fa0/3     LAN BRANCH         notconnect   1            auto   auto 10/100BaseTX",
            "Fa0/4     LAN BRANCH         notconnect   1            auto   auto 10/100BaseTX",
            "Fa0/5     LAB-AP-01          notconnect   1            auto   auto 10/100BaseTX",
            "Fa0/6     test               notconnect   1            auto   auto 10/100BaseTX",
            "Fa0/7     test               notconnect   1            auto   auto 10/100BaseTX",
            "Fa0/8     WAREHOUSE          notconnect   10           auto   auto 10/100BaseTX",
            "Gi0/1     LAB-FW-01          connected    1          a-full a-1000 10/100/1000BaseTX"
        ]
    ]
}

PLAY RECAP *********************************************************************
HAW-SWT                    : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

Error

```
No errors
```

Unreachable Hosts

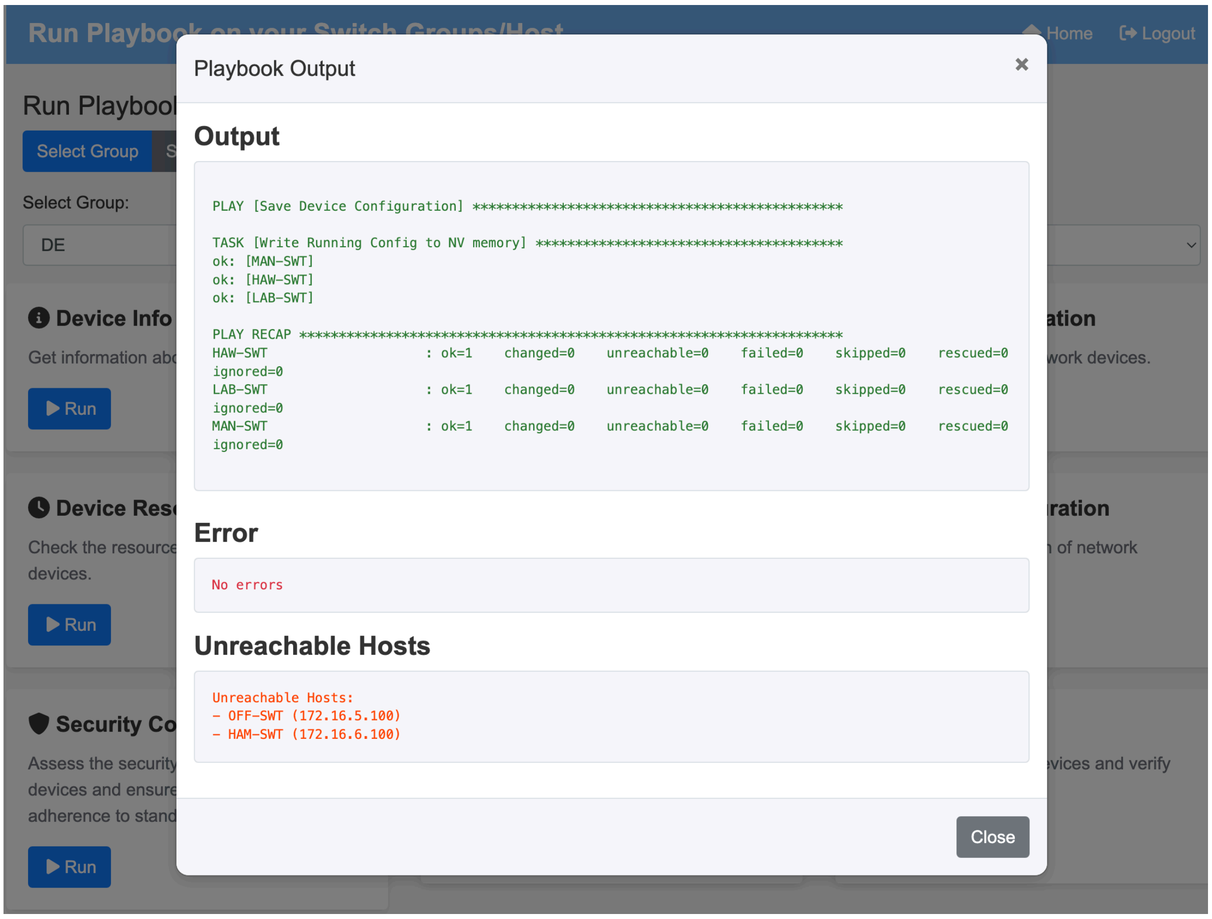
Figure 23: Executing Playbook on an Online Host

5.2.8 Playbook Execution (Multiple Hosts)

Objective: To verify playbook execution for multiple hosts.

Test Procedure: Execute playbook on a group with a mix of online and offline hosts

Expected Outcome: The playbook should successfully run on online hosts and display the list of unreachable hosts.



```
PLAY [Save Device Configuration] ************************************************

TASK [Write Running Config to NV memory] ****************************************
ok: [MAN-SWT]
ok: [HAW-SWT]
ok: [LAB-SWT]

PLAY RECAP **********************************************************************
HAW-SWT                    : ok=1    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
LAB-SWT                    : ok=1    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
MAN-SWT                    : ok=1    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

```
No errors
```

```
Unreachable Hosts:
- OFF-SWT (172.16.5.100)
- HAM-SWT (172.16.6.100)
```

Figure 24: Executing Playbook on a Group with Online and Offline Hosts

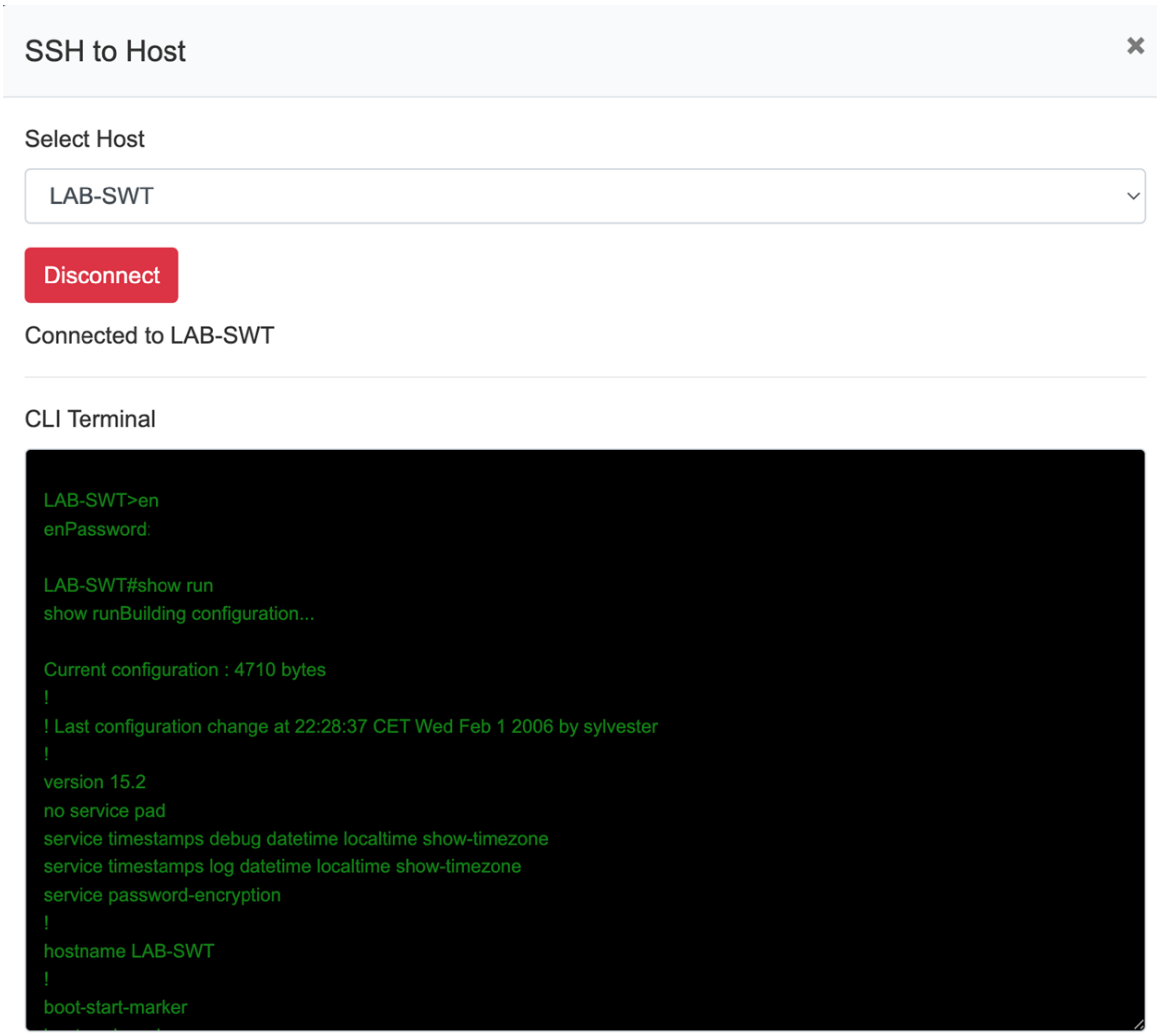
Result: The system correctly differentiated between online and offline hosts, ensuring reliable execution.

5.2.9 Command-Line Interface (CLI) Access

Objective: To enable direct access to individual switches for custom configurations.

Test: CLI access was tested for an online switch.

Expected Outcome: Administrators can successfully establish an SSH session.



```
LAB-SWT>en
enPassword:

LAB-SWT#show run
show runBuilding configuration...

Current configuration : 4710 bytes
!
! Last configuration change at 22:28:37 CET Wed Feb 1 2006 by sylvester
!
version 15.2
no service pad
service timestamps debug datetime localtime show-timezone
service timestamps log datetime localtime show-timezone
service password-encryption
!
hostname LAB-SWT
!
boot-start-marker
```

Figure 25: CLI Access to Online Switch

Result: CLI access did not function properly, though the admin could establish a connection and log in. Since input and output occur on the same interface, commands and responses overlap.

5.2.10 Secure Communication via HTTPS

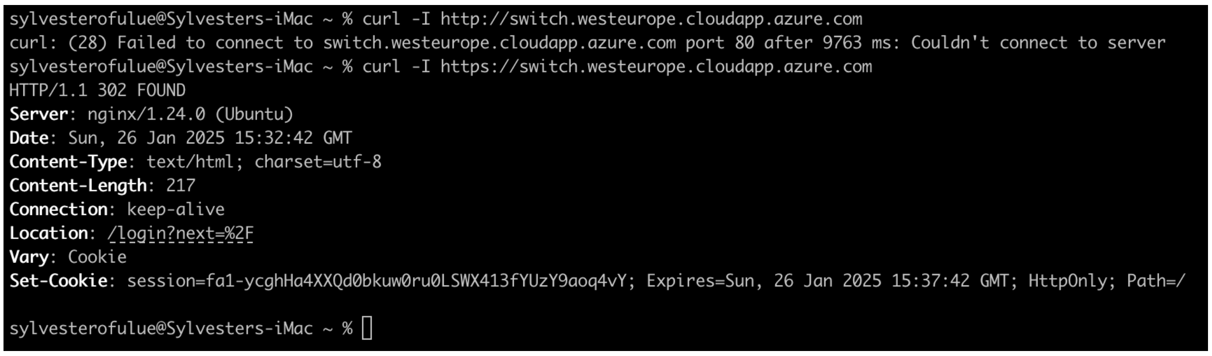
Objective: To ensure secure communication through HTTPS.

Test: The application was accessed using both HTTP and HTTPS.

Expected Outcome:

HTTP: The system should redirect users to HTTPS or block insecure access.

HTTPS: The system should establish a secure connection.



```
sylvesterofulue@Sylvesters-iMac ~ % curl -I http://switch.westeurope.cloudapp.azure.com
curl: (28) Failed to connect to switch.westeurope.cloudapp.azure.com port 80 after 9763 ms: Couldn't connect to server
sylvesterofulue@Sylvesters-iMac ~ % curl -I https://switch.westeurope.cloudapp.azure.com
HTTP/1.1 302 FOUND
Server: nginx/1.24.0 (Ubuntu)
Date: Sun, 26 Jan 2025 15:32:42 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 217
Connection: keep-alive
Location: /login?next=%2F
Vary: Cookie
Set-Cookie: session=fa1-ycghHa4XXQd0bkuw0ru0LSWX413fYUzY9aoq4vY; Expires=Sun, 26 Jan 2025 15:37:42 GMT; HttpOnly; Path=/

sylvesterofulue@Sylvesters-iMac ~ %
```

Figure 26: Secure Communication Protocol

Result:

Access via HTTP (*http://switch.westeurope.cloudapp.azure.com*) failed as expected, indicating that the server does not respond on port 80. This prevents insecure access. Access via HTTPS (*https://switch.westeurope.cloudapp.azure.com*) succeeded, with the server responding with an HTTP 302 status code, indicating a redirect or a successful connection to the secure endpoint. The server configuration ensures that port 80 (HTTP) does not serve requests, effectively enforcing HTTPS-only communication. However, HTTP requests are redirected to HTTPS on the browser.

Chapter 6: Conclusion and Future Work

This chapter presents the key contributions and potential future improvements to the switch management system.

6.1 Conclusion

The Centralized Network Switch Management System has successfully achieved its objectives by providing a unified platform for managing network switches, ensuring security and reliability. Key accomplishments include:

- **User Authentication**: The login functionality works as designed, ensuring only authorized users can access the system.
- **Database CRUD Operations**: The system reliably handles creating, reading, updating, and deleting records for switches, users, and groups, performing consistently during testing.
- **Monitoring Dashboard**: Displays accurately as intended.
- **Logging**: Critical events—such as user logins, warnings, and playbook executions—are logged, enhancing accountability and simplifying troubleshooting.
- **Playbook Execution**: The system efficiently executes Ansible playbooks on single and multiple hosts, enabling streamlined switch configuration and management.
- **CLI Access**: Secure CLI access allows customized configurations for individual switches.
- **HTTPS Communication**: Secure communication is ensured through HTTPS, with successful redirection from HTTP to HTTPS verified during testing.

These achievements demonstrate the system's ability to monitor and manage network switches from a single pane of view. While limited to Cisco switches as it relies heavily on Ansible libraries to set or retrieve switch configurations, the system's ping-base capability is a milestone in displaying the status of multi-vendor switches on the dashboard. However, some areas need further improvement, such as optimizing the offline switch load time and ensuring a smooth user experience when communicating with the switches via CLI.

6.2 Future Work

Several areas have been identified for future development to build on the current implementation. First, performance optimization efforts should focus on reducing latency in playbook execution to improve system responsiveness and enhance offline host handling by exploring asynchronous methods or parallel processing to overcome current threading limitations.

In terms of monitoring and reporting, introducing real-time monitoring dashboards would provide instant insights into switch availability and system performance, while developing comprehensive reporting features would enable better tracking of system activities and performance metrics.

The system's capacity should be expanded for scalability to manage a more significant number of switches without compromising performance. Exploring distributed architectures or cloud-native solutions could further improve scalability.

Enhanced security measures should include implementing advanced authentication mechanisms like multi-factor authentication (MFA), conducting regular audits and updates of dependencies to address vulnerabilities, and introducing role-based access control (RBAC) for finer control over administrator privileges.

Feature expansion could involve supporting non-Cisco devices to increase flexibility and integrating scheduling capabilities to automate playbook execution at predefined intervals.

Finally, user interface improvements should focus on refining the web interface to enhance usability and accessibility and incorporating visual indicators for real-time updates on switch statuses and playbook progress.

6.3 Final Remarks

This project demonstrates the viability of a cost-effective, centralized network switch management solution built using open-source tools. While the current implementation meets core requirements, load testing and performance analysis insights provide a roadmap for future enhancements. By addressing the identified areas for improvement, the system can evolve into a more versatile and efficient tool, empowering network administrators with greater control and flexibility.

Bibliography

[1] Huawei Technologies Co., Ltd, Data Communications and Network Technologies, Singapore: Springer Nature, 2023, pp. 537-538.

[2] L. Jia, S. Wenyan, W. Qiang, Y. Ren and H. Mingyi, "An Efficient Configuration Management Framework of Data Collection System in Power Dispatching Automation," in *2023 Power Electronics and Power System Conference (PEPSC)*, Hangzhou, China, 2023.

[3] G. Elena, V. Elena and S. Sergey, "Automated Service Configuration Management in IP/MPLS Networks," in *2022 International Conference on Modern Network Technologies (MoNeTec)*, Moscow, 2022.

[4] W. Rui, L. Yan, W. Zepeng and Z. Limei, "Research on Application Configuration Management Technology for Cloud Platform," in *2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, Chongqing, China, 2022.

[5] q.beyond Logineer GmbH, "Logineer IT Services for International Logistics," 2025. [Online]. Available: https://www.logineer.com/. [Accessed 1 January 2025].

[6] q.beyond Logineer GmbH, "About Us: logineer," 2025. [Online]. Available: https://www.logineer.com/en/about-us/. [Accessed 1 January 2025].

[7] Pallets Projects, "Flask Documentation (3.1.x)," 2024. [Online]. Available: https://flask.palletsprojects.com/en/stable/. [Accessed 5 January 2024].

[8] Ansible Community, "Introduction to Ansible," 2024. [Online]. Available: https://docs.ansible.com/ansible/latest/getting_started/introduction.html. [Accessed 3 January 2025].

[9] SQLite Documentation Team, "SQLite Documentation," 2024. [Online]. Available: https://www.sqlite.org/docs.html. [Accessed 5 January 2025].

[10] Microsoft Corporation, "What is Azure," 2025. [Online]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure.

[11] NGINX, Inc., "NGINX Documentation," 2024. [Online]. Available: https://nginx.org/en/docs/. [Accessed 5 January 2025].

[12] Cisco Systems, Inc., "Cisco DNA Center At-A-Glance," 2024. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/dna-center/nb-06-cisco-dna-center-aag-cte-en.html. [Accessed 2 January 2025].

[13] Cisco Systems, Inc., "Cisco Catalyst Software Subscription for Switching," December 2024. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/software/one-wireless-subscription/nb-06-dna-acces-wl-sw-faq-ctp-en.html?oid=faqswt027039#CiscoCatalystsoftwaresubscriptionforSwitching . [Accessed 2 January 2025].

[14] Cisco Systems, Inc., "Cisco Catalyst Center Compatibility Matrix," October 2024. [Online]. Available: https://www.cisco.com/c/dam/en/us/td/docs/Website/enterprise/catalyst_center_compatibility_matrix/index.html. [Accessed 2 January 2025].

[15] SolarWinds Corporation, "Systems Management Software," 2024. [Online]. Available: https://www.solarwinds.com/system-management-software. [Accessed 2 January 2025].

[16] Ansible Community, "Getting Started with Ansible Inventory," 2024. [Online]. Available: https://docs.ansible.com/ansible/latest/getting_started/get_started_inventory.html. [Accessed 3 January 2025].

[17] Ansible Community, "Getting Started with Ansible Playbook," 2024. [Online]. Available: https://docs.ansible.com/ansible/latest/getting_started/get_started_playbook.html. [Accessed 3 January 2025].

[18] Ansible Community, "Cisco.Ios — Ansible Community Documentation," 2024. [Online]. Available: https://docs.ansible.com/ansible/latest/collections/cisco/ios/index.html. [Accessed 3 January 2025].

[19] Cisco Systems, Inc., "Configuration Management: Best Practices White Paper," 2006. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/availability/high-availability/15111-configmgmt.html. [Accessed 7 January 2025].

[20] P. Buczek, *Lecture Notes on Bus Systems and Sensors,* vol. 002_OSI Model, Hamburg: Hochschule für Angewandte Wissenschaften Hamburg, SoSe 2024, pp. 28-29.

[21] H. Nico, B. Geoffrey and V. Holger, "A Reference Architecture for Deploying Component-Based Robot Software and Comparison with Existing Tools," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*, Laguna Hills, CA, USA, 2018.

[22] T. Pooja and J. Prashant, "Django: Developing web using Python," in *2023 3rd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, Greater Noida, India, 2023.

[23] A. Dany, L. Maria, S. Luzia, A. Maryam, M. Pedro and S. José, "Performance Comparison of Redis, Memcached, MySQL, and PostgreSQL: A Study on Key-Value and Relational Databases," in *2023 Second International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, Singapore, 2023.

[24] React Team, "React Tutorial," 2025. [Online]. Available: https://legacy.reactjs.org/tutorial/tutorial.html. [Accessed January 2025].

[25] Angular Team, "Overview," 2025. [Online]. Available: https://angular.dev/overview. [Accessed January 2025].

Appendix A: Source Code Overview

A.1: app.py

- **Description:** Main Flask application file containing API routes, database integration, and playbook execution logic. ChatGPT generated the initial structure of this code while modifications and expansions were made.
- **Location:** See CD: ~/switch_management/app.py

A.2: schema.sql

- **Description:** SQL schema file defining the database structure.
- **Location:** See CD: ~/switch_management/schema.sql

A.3: script.js

- **Description:** JavaScript functions that handle the home page's interactive elements and dynamic behaviors (index.html). It also serves as the intermediary between the frontend and the server. ChatGPT generated the initial structure of this code while modifications and expansions were made.
- **Location:** See CD: ~/switch_management/static/script.js

A.4: README.md

- **Description:** README.md provides a high-level overview of the Centralized Switch Management System and instructions for quickly starting the application.
- **Location:** See CD: ~/switch_management/README.md

A.5: requirements.txt

- **Description:** This contains the necessary dependencies needed to run the application.
- **Location:** See CD: ~/switch_management/requirements.txt

Appendix B: Playbook Source Code Files on the CD

B.1: get_status.yml

- **Description:** This function verifies whether hosts are reachable. It retrieves and displays the device's uptime, model, and firmware version if they are.
- **Location:** See CD: ~/switch_management/playbooks/get_status.yml

B.2: reboot_device.yml

- Description: Save the device configuration and perform a reboot operation.
- **Location:** See CD: ~/switch_management/playbooks/reboot_device.yml

B.3: backup_config.yml

- **Description:** This program creates a backup of the current device configuration and stores it locally on the host computer for recovery or auditing purposes.
- **Location:** CD Path: ~/switch_management/playbooks/backup_config.yml
-

While not directly referenced in the main document, the following playbooks are part of the overall system and serve various purposes for managing network switches:

B.4: device_info.yml

- **Description:** Retrieves detailed device information, including hostname, management IP, and hardware specifications.
- **Location:** CD Path: ~/switch_management/playbooks/device_info.yml

B.5: device_resources_uptime.yml

- **Description:** This program collects data about device resources, such as CPU and memory usage and uptime. It helps monitor device health and performance.
- **Location:** CD Path: ~/switch_management/playbooks/device_resources_uptime.yml

B.6: interface_status.yml

- **Description:** This function checks the status of all device interfaces and provides information on operational status, errors, and bandwidth usage.

- **Location:** CD Path: ~/switch_management/playbooks/interface_status.yml

B.7: restore_config.yml

- **Description:** This function restores a previously backed-up configuration to the device, ensuring consistent settings in case of failure or configuration drift.
- **Location:** CD Path: ~/switch_management/playbooks/restore_config.yml

B.8: save_config.yml

- **Description:** Saves the device's running configuration to persistent storage, ensuring changes are retained after a reboot.
- **Location:** CD Path: ~/switch_management/playbooks/save_config.yml

B.9: security_compliance.yml

- **Description:** Verifies device security compliance policies, such as ensuring access lists and secure protocols are configured.
- **Location:** CD Path: ~/switch_management/playbooks/security_compliance.yml

B.10: vlan_config.yml

- **Description:** Configures VLANs on network devices, enabling segmentation and isolation of network traffic for better management and security.
- **Location:** CD Path: ~/switch_management/playbooks/vlan_config.yml

Appendix C: Time Response Measurement

This MATLAB script file can generate the plots used in the corresponding sessions.

C.1: Load Test for Online Switches

```
clf;
Number_of_Hosts = [1,3, 5, 10, 20, 30, 50, 100, 150, 200, 300, 400,
500, 1000];
Response_time = [38,40,55,69,80,90,117,178,245,303,481,612,720,1370];


plot(Number_of_Hosts,Response_time, '-o', 'MarkerSize', 5, 'LineWidth',
1)
title('Load Test for Online Hosts')
grid minor
ylabel('Response Time in milliseconds')
xlabel('Number of Hosts')
```

Screenshots Location: CD Path: ~/Measurements/Online_Hosts

C.2: Load Test for Offline Switches

```
clf;
Number_of_Hosts = [1, 3, 5, 10, 20, 30, 50, 100, 150, 200, 300, 400,
500, 1000];
Response_time = [1080, 1080, 1080, 1070, 1080, 1080,
1120,2120,3250,4210,6230,8260,10290,20380];


plot(Number_of_Hosts,Response_time, '-o', 'MarkerSize', 5, 'LineWidth',
1)
title('Load Test for Offline Hosts')
grid minor
ylabel('Response Time in milliseconds')
xlabel('Number of Hosts')
```

Screenshots Location: CD Path: ~/Measurements/Offline_Hosts

C.3: Optimization with Threading

```
clf;
Number_of_Thread = [2, 4, 8, 16, 32, 64, 128, 256, 512];
Response_time = [251.22, 125.70, 63.42, 32.19, 16.16, 8.21, 4.43, 2.38,
2.24];
```

```
loglog(Number_of_Thread,Response_time, '-o', 'MarkerSize', 5,
'LineWidth', 1)
title('Threading Optmization')
grid minor
ylabel('Logarithm of Response Time in seconds')
xlabel('Logarithm of Max-Thread')
```

Screenshots Location: CD Path: ~/Measurements/Optimization