Est.
1841 | YORK
ST JOHN
UNIVERSITY

AUTOMATING SYSTEM ADMINISTRATION

An Analysis of Efficiency and Security using Automation and IaC

GUY BROWN

STUDENT ID: 220079352
Supervisor: Shao Ying Zhu
Module: COM6016M
York St John University
School of Science, Technology and Health
BSc Hons Computer Science (BSCPSFT)

Table of Contents

Table of figures:

Abstract

The growing complexity of IT infrastructures has made system automation essential for ensuring consistency, efficiency, and scalability in administrative tasks. This dissertation investigates the practical use of Ansible, an open-source automation tool, to manage system configurations across both Linux and Windows platforms. The research explores three core questions: how Ansible affects the efficiency and reliability of system administration, what limitations arise during its use, and what ethical and security considerations must be addressed when automating privileged tasks.

A series of real-world administrative scenarios were implemented using Ansible playbooks, including user account management, web server deployment, and system hardening through patching and firewall configuration. Each task was evaluated against its manually executed counterpart, with metrics including execution time, consistency of results, and security posture. The findings indicate that Ansible significantly improves operational efficiency and reduces the likelihood of misconfiguration through its declarative and idempotent design.

However, platform-specific limitations, particularly on Windows, introduced complexity, especially in update management and secure credential handling. Ethical concerns were addressed through secure password hashing and careful use of privilege escalation. The study concludes that Ansible is an effective and reliable automation tool when supported by responsible configuration and auditing practices. Opportunities for future work include extending the implementation to cloud environments and conducting comparative studies with other automation platforms.

Introduction

System administration often involves complex, repetitive tasks essential to maintaining and securing IT infrastructures, from provisioning servers to configuring software environments. The rapid adoption of Infrastructure as Code (IaC) in system administration has enabled automation of these processes, particularly through tools like Ansible. Ansible, an agentless automation framework, allows for streamlined task execution across multiple systems, improving efficiency and reducing errors in large, complex environments.

Background and Context

As organisations grow, the need for stable, secure, and predictable IT infrastructures has become more pressing. Traditionally, system administration tasks were managed manually, a time-consuming process prone to human error. By automating these routine processes, Ansible offers a solution to improve system administration, potentially reducing errors and improving operational consistency. This project proposes an evaluation of Ansible's capacity to automate critical system administration tasks, including software deployment, configuration management, and network orchestration, with a focus on the challenges and benefits in diverse and complex environments.

Problem Statement and Research Questions

Although automation through Infrastructure as Code (IaC) tools has become increasingly popular in system administration, there remains a lack of focused empirical research on how individual tools, such as Ansible, impact operational efficiency, reliability, and security. While Ansible is widely adopted due to its agentless architecture and cross-platform support, the practical challenges and limitations of its usage across varied system environments are not well understood. Additionally, the ethical and security implications of automating privileged tasks with Ansible require further exploration. This research aims to address these gaps by investigating Ansible's real-world impact in automating administrative tasks across Linux and Windows systems. This research seeks to explore the following questions:

- How does Ansible automation improve the efficiency and reliability of system administration tasks in a virtualised environment?
- What challenges and limitations are encountered when using Ansible to automate the configuration of Linux and Windows systems?
- What ethical and security considerations must be addressed when using Ansible to manage infrastructure at scale?

Relevance and Contribution

This research holds value for IT professionals, system administrators, developers, and organisations that are currently adopting or planning to adopt automation in their IT operations. While past studies have examined IaC and automation generally, there is limited empirical research specifically on Ansible's impact in the field of system administration. By investigating Ansible's application and its outcomes in system administration, this study aims to provide actionable insights for efficient automation, improve understanding of ethical implications, to outline best practices for scaling automation in complex IT environments.

Literature Review

System administration has traditionally relied on manual processes, such as configuring servers, managing networks, and deploying software. While effective for small-scale systems, these tasks are labour-intensive, error-prone, and inefficient in modern IT infrastructures that demand scalability and rapid deployment. Automation tools like Ansible, Puppet, and SaltStack have revolutionised these processes by enabling Infrastructure as Code (IaC), which defines configuration through code to ensure repeatability, consistency, and reduced human intervention. (Samuel Wågbrant, 2022) (Parra, 2023)

Infrastructure as Code (IaC)

IaC enables the management and provisioning of infrastructure through code, leveraging tools that ensure consistency and reduce deployment times. Ansible's agentless architecture, which operates through SSH without additional software on target devices, stands out for its simplicity and ease of use. (Samuel Wågbrant, 2022) (Parra, 2023)

However, studies have noted its higher bandwidth usage compared to agent-based tools such as SaltStack, which employs proxy agents to reduce network overhead in scaled environments. (Samuel Wågbrant, 2022) (Masek, 2018).

The role of IaC in open-networking environments is expanding. For instance, a case study integrating Ansible with Cumulus Linux demonstrated its potential in managing modern network architectures such as VXLAN (Virtual Extensible LAN). This study emphasised how IaC reduces configuration errors and improves version control through Git-based workflows, positioning Ansible as a key player in the DevOps movement. (Parra, 2023)

Performance and Scalability of Ansible

Ansible is widely regarded as an efficient automation tool due to its agentless design and lightweight execution model. It typically uses SSH for communication, which removes the need for pre-installed agents on target machines and reduces deployment complexity. In this project, SSH was used to manage both Linux and Windows systems, demonstrating the flexibility of Ansible's transport layer when properly configured.

Several industry case studies, such as those by Red Hat and Verizon Media, have shown that Ansible significantly shortens deployment times and improves consistency in large-scale environments.

(Elradi, 2023) conducted an academic study evaluating Ansible's effectiveness in automating the deployment of a multi-tier web application stack, comprising MySQL, Nginx, and Tomcat. Their results showed substantial reductions in both configuration time and deployment errors, affirming Ansible's value in orchestrating Linux-based systems. However, their study was limited to a single platform and did not explore administrative tasks outside of web service provisioning. The current dissertation builds on this by expanding the scope to include Windows automation via SSH, as well as tasks such as user management, system patching, and security hardening. Furthermore, this work includes quantitative benchmarking of task execution times across both manual and automated approaches, providing a more comprehensive assessment of operational efficiency.

Similarly, (Masek, 2018) investigated the use of Ansible in managing virtualised lab environments at a university. The study reported that Ansible drastically reduced configuration effort for instructors and IT staff, but also noted occasional performance issues, particularly when running multiple sequential tasks on limited-resource virtual machines. This aligns with the present study's observation that Windows-based tasks — especially updates and service enablement — performed more slowly than their Linux equivalents, highlighting Ansible's variable performance across platforms, even when using a consistent SSH-based approach.

While these studies demonstrate Ansible's effectiveness in small to medium-scale deployments, recent research has highlighted emerging challenges in more complex environments. (Carolina Carreira, 2025) conducted a mixed-methods study of over 59,000 community discussions and interviews with 16 professionals, identifying recurring usability issues with Ansible. These include difficulties with debugging, a lack of rollback functionality, slow performance on large inventories, and a steep learning curve for managing templating (Jinja2) and advanced playbook structures. Although Ansible remains accessible for beginners, the study found that its usability trade-offs become more pronounced as environments grow in complexity. These findings emphasise the importance of understanding Ansible's architectural limits, especially in enterprise-scale automation where stateful management, error recovery, and modular control are critical.

The present dissertation contributes to this developing body of work by empirically evaluating Ansible's efficiency and consistency across a range of administrative tasks in a controlled virtualised environment. It provides specific timing benchmarks (see Table 1) and includes validation procedures to confirm service availability and user configuration accuracy across Linux and Windows hosts. This experimental setup enables a practical, evidence-based understanding of Ansible's real-world strengths and limitations.

Ethical and Security Considerations

Using Ansible for automation introduces significant ethical and security considerations due to its ability to manage and modify infrastructure at scale. Security is paramount, as Ansible typically requires privileged access to target systems, often via SSH. Consequently, secure management of SSH keys, user credentials, and adherence to the principle of least privilege are critical. Sensitive information, such as passwords and API keys, should be encrypted using Ansible Vault; however, the security of the vault itself depends on the strength of the vault password and the secure handling of decryption keys.

Maintaining the integrity of Ansible playbooks is equally important. Poorly written or malicious playbooks could introduce vulnerabilities, cause widespread damage, or lead to service outages, highlighting the need for rigorous code review and version control practices. Furthermore, the Ansible control node represents a high-value target and must be hardened with strict access controls.

From an ethical standpoint, the power Ansible grants demands responsible use. Administrators must ensure that automation does not inadvertently cause harm, breach privacy regulations when handling sensitive data, or obscure accountability for system changes. To mitigate both security and ethical risks, it is essential to implement transparency, robust testing procedures, Role-Based Access Control (RBAC), and thorough auditing mechanisms.

Integration in Modern Networking

Modern IT environments, particularly those embracing open networking, benefit from Ansible's ability to bridge traditional and programmable network architectures.

Case studies on Cumulus Linux and Ansible have highlighted the feasibility of integrating these tools to enable dynamic network configurations and reduce operational complexities.

These integrations underline Ansible's flexibility in adapting to cutting-edge technologies like intent-based networking and software-defined infrastructures. (Billoir, 2024) (Parra, 2023)

Summary

The literature demonstrates that Ansible is widely adopted for automating configuration and deployment tasks across Unix-like environments, with recognised strengths in its agentless design, human-readable syntax, and suitability for continuous integration workflows. Multiple studies have confirmed its ability to reduce deployment time and improve system consistency, particularly in small- to medium-scale Linux deployments. However, few empirical studies have evaluated Ansible's cross-platform capabilities using a consistent, SSH-based transport across both Linux and Windows systems. Existing research tends to focus either on theoretical capabilities or Linux-only implementations, with limited attention to execution timing, consistency, and security implications in mixed environments. This dissertation addresses that gap by implementing, validating, and benchmarking a series of real-world administrative tasks on both operating systems using Ansible exclusively over SSH. It contributes to the growing body of infrastructure-as-code literature by providing detailed quantitative and qualitative evaluation of automation workflows, supported by reproducible playbooks and a controlled testing environment.

Research Design

This project adopts a practical experimental research design to evaluate the effectiveness of Ansible in automating administrative tasks on Linux and Windows systems. An experimental approach was chosen due to its suitability for producing measurable outcomes such as task execution time, configuration accuracy, and repeatability, all of which directly support the research questions. The aim was not only to observe Ansible in action, but also to compare its outcomes with equivalent manual procedures, enabling a grounded assessment of efficiency, reliability, and security.

Ansible was selected as the automation platform due to its popularity, declarative syntax, and agentless architecture, which allows it to interface with remote systems using SSH without requiring a persistent client. Its YAML-based playbooks are accessible and transparent, enabling auditable automation scripts that can be easily version-controlled. Ansible also supports cross-platform configuration, making it an ideal candidate for evaluating automation in mixed-operating-system environments, a common scenario in real-world infrastructure.

A set of common administrative tasks were chosen to reflect real-world system administration responsibilities. These included: user management, web server deployment, firewall configuration, and system updates with basic hardening. Tasks were implemented on both Linux (Ubuntu Server 22.04) and Windows Server 2022 virtual machines to test Ansible's consistency and cross-platform applicability. Each task was designed to reflect typical administrative procedures while allowing for meaningful comparison between manual and automated execution.

The experimental environment was hosted on Proxmox, using local virtual machines to ensure isolation and reproducibility. This virtualised setup allowed consistent control over system state, rollback testing, and the ability to simulate multiple hosts for scalability assessment. No real production systems were used, avoiding any risk of system disruption or ethical concerns related to live infrastructure.

Each task was applied manually first, with the time and complexity of execution recorded as a baseline. Ansible playbooks were then used to automate the same tasks. Outputs were captured through both terminal logs and screenshots. Additionally, Grafana and Prometheus were used for passive observability to confirm service persistence over time. All testing and validation were documented, and results were presented in comparative form in the Results and Evaluation section.

Environment Setup

An experimental research design was selected to facilitate measurable comparisons between manual and automated configurations, enabling quantifiable assessment of Ansible's impact. A virtualised testbed was used to control variables, maintain reproducibility, and isolate the effects of automation in a way that reflects realistic infrastructure management scenarios.

This experiment will be conducted on a virtualised setup with Proxmox as the hypervisor, simulating an IT infrastructure consisting of multiple servers and networked devices. The setup will include a mix of Linux and Windows servers to reflect a typical enterprise environment.

Tools and software:

- Ansible will be used to automate system administration tasks, including software installation, configuration management and security updates.
- Git will be used to manage version control for various configuration scripts, ensuring repeatability and transparency within the experiment
- Data collection – Quantitative
- Time Efficiency – Task completion time will be recorded for both automated (Ansible) and manual processes.
- Error rate – The frequency and type of errors encountered in automated and manual configurations will be documented to compare reliability.
- Resource Utilisation – CPU, memory and network usage metrics will be collected from monitoring tools to evaluate system performance.
- Data collection – Qualitative
- Configuration complexity – Each task will be evaluated for complexity, providing insights into time and skill level required for both manual and automated processes.
- System logs – Logs will be analysed to identify any security vulnerabilities or configuration inconsistencies introduced by automation.

Data analysis

The data analysis in this project is designed to evaluate the impact of using Ansible for automating system administration tasks across both Linux and Windows platforms. The analysis directly supports the three research questions by focusing on (1) operational efficiency and reliability, (2) practical challenges and limitations encountered, and (3) security and ethical considerations.

Efficiency and reliability will be assessed by comparing manually executed administrative tasks with their automated counterparts using Ansible playbooks. Key metrics include task completion time, number of manual steps avoided, success rates of playbook execution, and consistency of system states post-deployment. Each task (e.g., user creation, web server setup, system hardening) will be applied to virtual machines, and outcomes will be verified through service status checks, file presence, or command-line validation.

Challenges and limitations will be recorded during implementation, including cross-platform differences, module limitations, and edge cases where Ansible struggled to perform or introduced complexity. These will be discussed qualitatively in relation to the second research question.

For the third research question, ethical and security aspects will be analysed by examining how sensitive data (e.g., user passwords, privilege escalation) is handled in the playbooks. Consideration will be given to Ansible Vault usage, overuse of become, and the transparency of playbook logic for auditing purposes.

Grafana and Prometheus, although set up manually, will be used to support the reliability analysis by visualising service uptime and system state stability following deployment.

Implementation

Experimental setup

To evaluate Ansible's efficiency and security in system administration, a controlled experiment was conducted in a virtualised environment. This setup was designed to compare Ansible's automation capabilities against manual system administration across both Linux and Windows platforms. The focus was placed on assessing how effectively Ansible could automate routine administrative tasks and maintain system reliability in diverse operating environments.

Ethical considerations

This study involved no human participants or sensitive real-world systems. However, ethical concerns were considered in the secure handling of credentials, responsible use of automation scripts with privileged access, and proper configuration of audit logs. All test systems were isolated in a virtual environment, and no production infrastructure was affected.

Environment setup

A virtualised environment was created using Proxmox as the hypervisor, this was chosen due to personal familiarity with its feature set as well as being a viable enterprise option with a suitable enterprise support plan (if needed). This Proxmox hypervisor (Specifications in Figure 2) will run several virtual machines, these will consist of two Linux (Ubuntu) servers (As seen in figure 4) as well as one Windows Server 2022 (As seen in Figure 3) instance and one control node.

Each VM has been preconfigured with a stock installation of either Windows Server 2022 or Ubuntu 24.04, with a manual IP address configured. It is also fully updated as of 14 May 2025.

Proxmox Host	
CPU:	Intel Core i3-8100T
RAM:	32 GB LPDDR4
Boot Drive:	128 GB SATA SSD
VM Drive	1 TB NVME SSD

Figure 1 - Proxmox Host Specifications

Windows Server 2022 VM	
Virtual CPU:	2 v-Cores [x86-64-v2-AES]
RAM:	4 GiB
Storage:	64 GiB [SATA]
Network:	Virtualised Intel e1000

Figure 2 - Windows Server 2022 VM Specifications

Ubuntu Server 24.04 VMS (Master, VM1 and VM2)	
Virtual CPU:	2 v-Cores [x86-64-v2-AES]
RAM:	4 GiB
Storage	32 GiB [SATA]
Network:	VirtIO

Figure 3 - Ubuntu Server 24.04 VM Specifications

Prometheus and Grafana deployment

To effectively monitor and store system performance information, the Prometheus and Grafana stack was deployed onto the control node, which acts as a central server to monitor the client nodes and store performance metrics in a central place.

On the control node, Prometheus was deployed as the central monitoring server, with the responsibility of collecting and storing performance data from the other hosts on the network. The Grafana dashboard was also configured to allow the visualisation of these collected metrics, allowing for easy analysis and comparison. The Prometheus instance is available on port 9090, while the Grafana interface is available at port 3000.

On the client nodes, an additional client exporter was needed to enable the collection of all needed metrics (CPU Usage, RAM Usage, as well as Network usage) in real time. On Linux clients, the node exporter was used. (Prometheus, 2025) On port 9100. On the Windows client, a community-made Windows exporter was used (Community, 2025) to expose Windows equivalent system statistics. This combination of clients ensures that live system monitoring is available across all common operating systems.

Once the exporter services were enabled and configured, the VMs began sending system information across the network to the control node, where Grafana was configured. On the control node, using Grafana, a dashboard was created to visualise the stored data as shown in Figure 1.

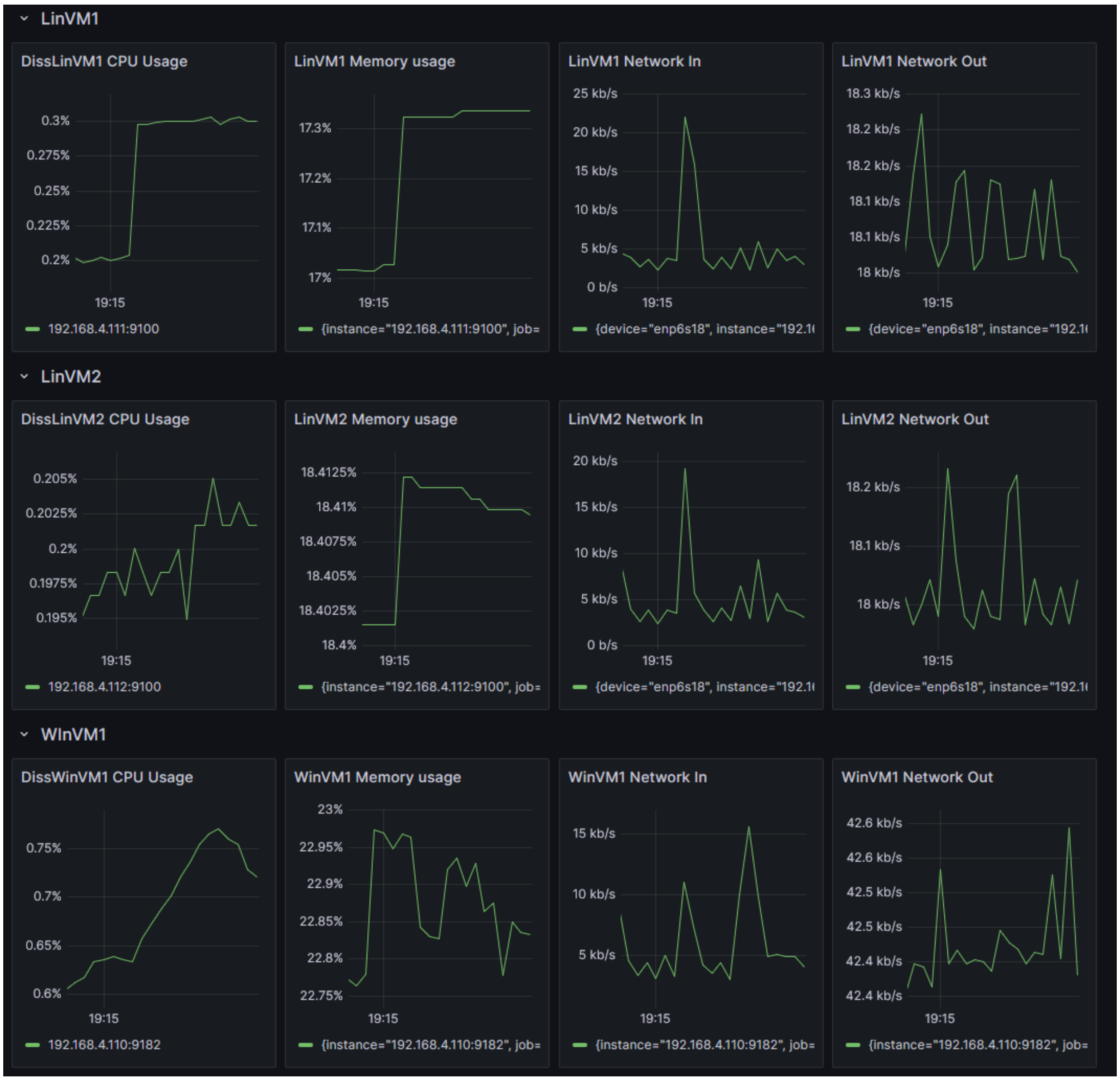


Figure 4 - Grafana Dashboard - Showing CPU, Memory and Network information

Although Grafana and Prometheus were not deployed via automation tools, their inclusion in the setup allowed for centralised monitoring and supported the evaluation of system behaviour following automated deployments.

Ansible Deployment

To evaluate Ansible's effectiveness in automating system administration tasks, a set of deployment scripts known as "playbooks" was developed and executed on both Linux and Windows virtual machines. Responsible for orchestrating these systems, the control node deploys playbooks written in YAML (YAML Ain't Markup Language) to instruct the managed nodes on what steps to take.

Inventory configuration

Ansible stores a list of managed computers in an inventory file. This file specifies the hosts' IP addresses or hostnames and gives them the ability to group them. We will use this in our example to separate the Windows and Linux virtual machines. This allows us to target machines based on these groups so that the correct tasks get assigned.

Connecting to the hosts

Ansible has several methods that can be used to connect to hosts and allow the execution of automation tasks. In this project, I have chosen to use OpenSSH as the primary communication method for both Windows and Linux hosts rather than using the more traditional WinRM (Windows Remote Management) method for the Windows host.

Using OpenSSH for both platforms ensures consistency and avoids the complexity of setting up HTTPS listeners and certificates. Furthermore, OpenSSH has been a natively supported remote shell option on Windows starting from Windows Server 2019 (1809), making it a reliable alternative.

Using a single protocol allows for a more unified Ansible inventory configuration and reduces overhead from writing playbooks and diagnosing connection issues.

SSH Configuration for Linux Hosts

For Linux servers, the existing OpenSSH server and the 'diss' user will be used. Key-based authentication will also be set up between the control node and the Linux clients to ensure password-less operation, this will be done by:

- Generating an SSH key pair on the control node
- Using the ssh-copy-id command to copy the SSH public key

SSH Configuration for Windows Hosts

For the Windows Server 2022 host, OpenSSH will need to be enabled using PowerShell; similarly to Linux, key-based authentication will be enabled. To do this, we will:

- Install and enable the OpenSSH server feature using PowerShell (Add-WindowsCapability -Online -Name OpenSSH.Server)
- Enable and start the 'sshd' service (Start-Service sshd | Set-Service -Name sshd -StartupType 'Automatic')
- Allow incoming SSH connections through the Windows Defender Firewall.
- Copy the public key from the control node to the "C:\Users\diss\.ssh\authorized_keys" and "C:\ProgramData\ssh\administrators_authorized_keys" text files.
- Set permissions so only the System and 'diss' users can access the file.

Playbooks and Roles

In this experiment, three main tasks are being performed: Web server deployment, user and permission management, as well as security hardening and updates.

Web server deployment

On Linux hosts, the ‘apt’ (Williams, 2025) Module will be used to install and configure the ‘apache2’ package, this package provides us with a basic web server to test with. After this, a firewall rule will be created using ‘ufw’; this firewall rule will open port 80 (HTTP) and port 443 (HTTPS) to allow the flow of web traffic.

On Windows hosts, we will use the ‘win_feature’ module to enable and configure Windows Server’s built-in IIS (Internet Information Services) feature. After this is done, we will use the ‘community.windows.win_firewall_rule’ to open ports 80 and 443.

Security hardening and updates

On Linux hosts, we will use the ‘apt’ module to install and configure the ‘fail2ban’ package. This package enhances a system’s resistance to brute force attacks. It does this by adding IP addresses to a block list when too many failed authentication attempts have occurred. (Contributers, 2025)

System updates are essential to security, regardless of the platform. For Linux, we will use the ‘apt’ module, whereas on Windows, we will use the ‘win_updates’ module (Davis, 2025) for the installation and management of updates.

To enable automatic updates on Linux, the ‘apt’ module will be used to install the ‘unattended-upgrades’ package. Then, we will use the ‘systemd’ module to enable and start the service.

User and Permission management

On all platforms, we will create a user called ‘admin’ with privilege escalation. We will do this by using the ‘user’ module on Linux to create the user and assign it to the ‘sudo’ group. On Windows, we will use the equivalent ‘win_user’ module. (Anon., 2025) to create the user and assign it to the ‘Administrators’ group. This results in the equivalent authentication and privilege escalation capabilities being in place on all managed machines.

To enhance security for both operating systems, during user creation, we will also use Ansible’s ‘set_fact’ module to generate a random 20-character password and then hash it securely using sha512. This hashed password will then be passed in as the user’s password for Linux. Due to limitations, Windows cannot accept the password in a hashed format, so the plaintext random password is used instead.

Scope and Focus of Automation Tool Usage

This project focuses exclusively on the use of Ansible as the automation tool for system administration. Although other tools such as Puppet and SaltStack were initially considered, they were excluded from the final implementation to allow a deeper, more focused evaluation of Ansible's capabilities. Ansible was selected for its agentless architecture, cross-platform support, and widespread industry adoption. The experimental setup centres on using Ansible to deploy, configure, and secure both Linux and Windows virtual machines, providing a practical context for evaluating its real-world impact on efficiency, reliability, and system security.

Results and evaluation

This section presents the results of the implementation phase and evaluates how Ansible performed across a range of administrative tasks on Linux and Windows systems. Each task is analysed in terms of outcome, execution time, and reliability, with comparisons to equivalent manual procedures where appropriate. These findings are used to assess Ansible's efficiency and reliability, identify practical challenges and limitations, and reflect on ethical and security concerns encountered during automation. These tasks are grouped by functionality, and where applicable, system logs, playbook outputs, and manual validation methods are referenced to support the analysis.

Manual Process baseline

To provide a meaningful benchmark for evaluating Ansible's efficiency, each automated task was first analysed in terms of its manual equivalent. These manual processes were carried out using either command-line tools (on Linux) or system utilities and graphical interfaces (on Windows). For each task, the typical steps, estimated time per host, and known challenges were documented. This provided a consistent baseline against which Ansible's performance, reliability, and security could be assessed. The manual task descriptions also helped highlight the risk of human error, inconsistency, and inefficiency that automation seeks to address.

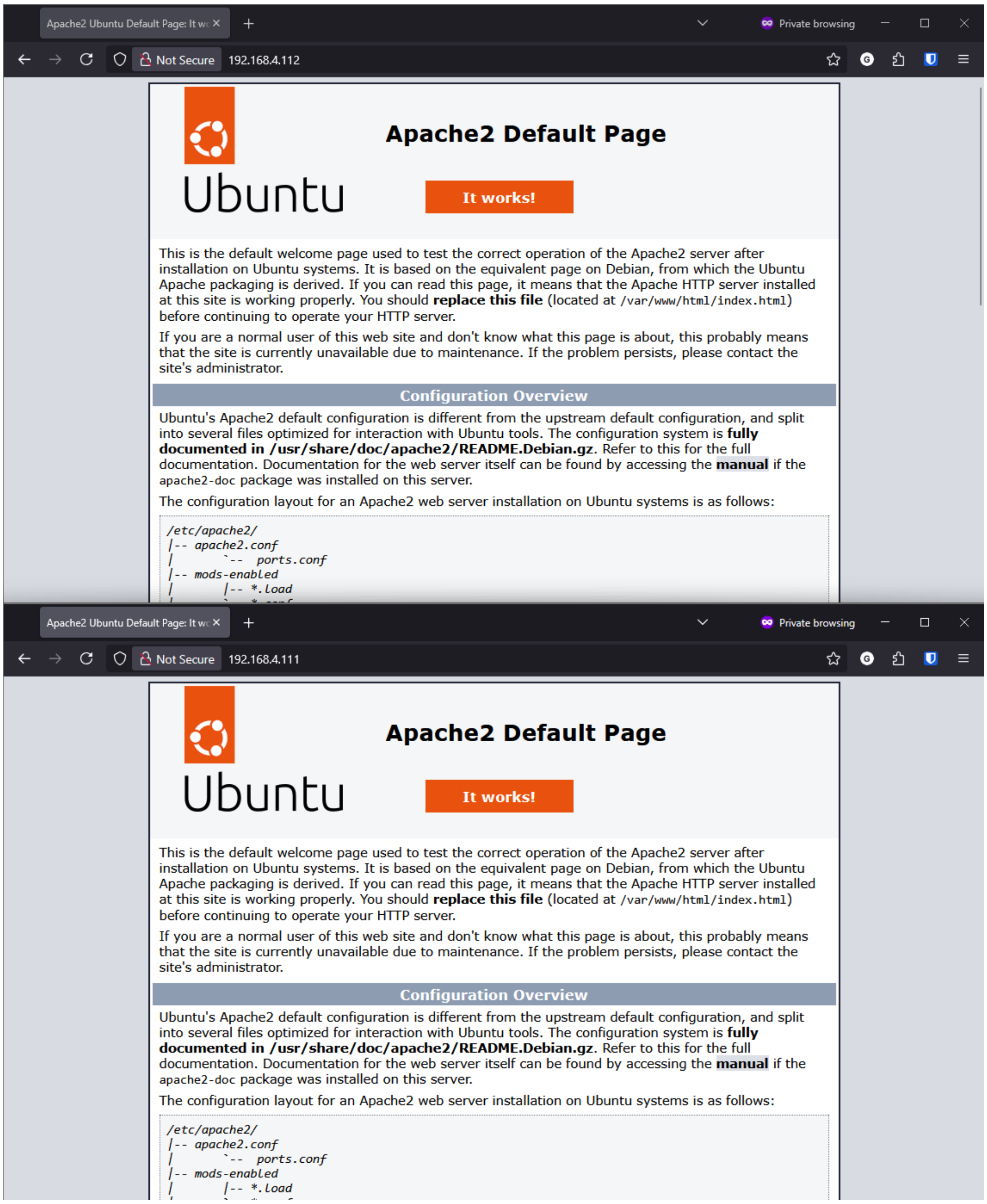
Web server deployment

Task overview

This task aimed to evaluate Ansible's ability to automate the deployment of web server infrastructure on both Windows and Linux platforms. On Linux, the 'apache2.yml' playbook was used to install and configure the Apache2 HTTP server (A popular free and open-source web server), ensure the service is enabled and running, as well as opening TCP (Transmission Control Protocol) ports 80 and 442 (HTTP and HTTPS) to this traffic through the firewall. On Windows systems, the 'iis.yml' playbook was used to install the IIS web server, start the service and open ports 80 and 443.

Results

On the Ubuntu virtual machines (diss-winvm1 and diss-winvm2), the Apache2 service was installed and started successfully. The 'ufw' firewall was updated to allow inbound traffic on ports 80 and 443. A client on the same network navigated to their respective pages to verify Apache's functionality, as seen in Figure 5. This confirmed that Apache was operational immediately after deployment. The initial run of the playbook completed in approximately 32 seconds (as seen in Figure 6) and was reproducible once a VM snapshot was restored, exhibiting idempotent behaviour. After the initial run, times dropped to about 12 seconds (as seen in Figure 7) as no changes were needed after gathering facts.



Apache2 Default Page

Ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
```

Apache2 Default Page

Ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

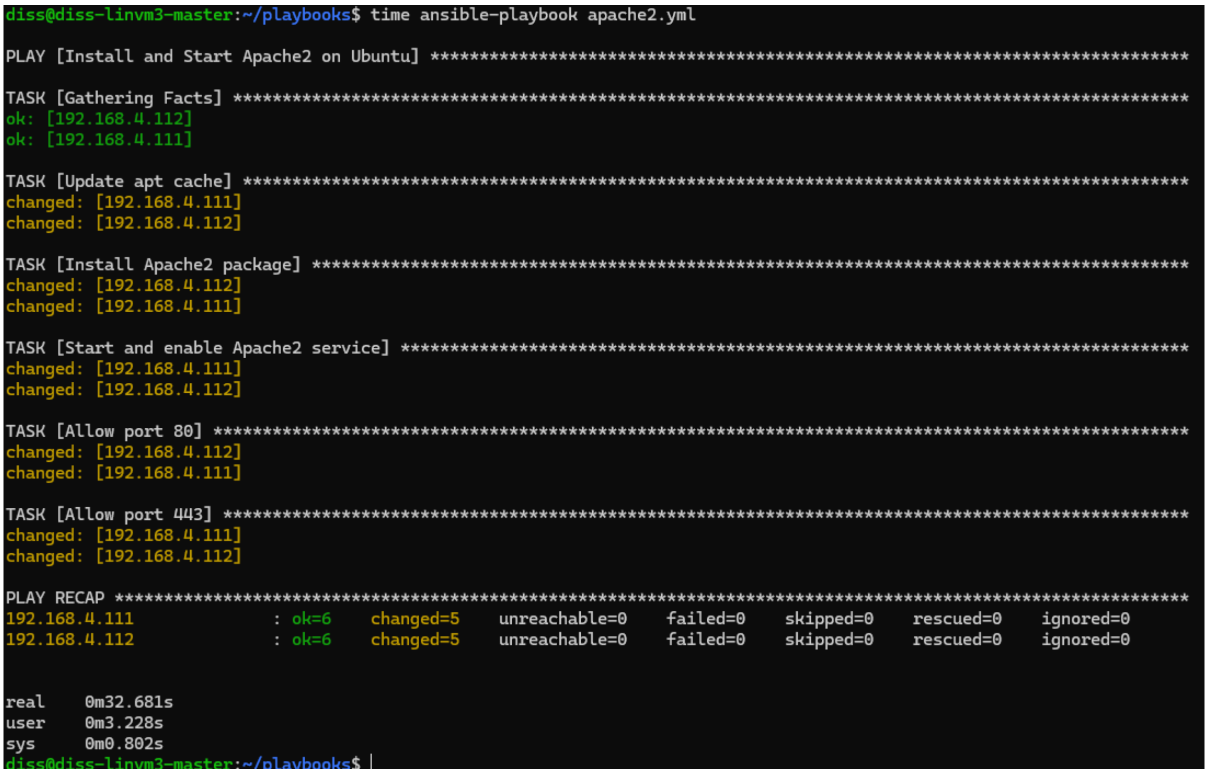
Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

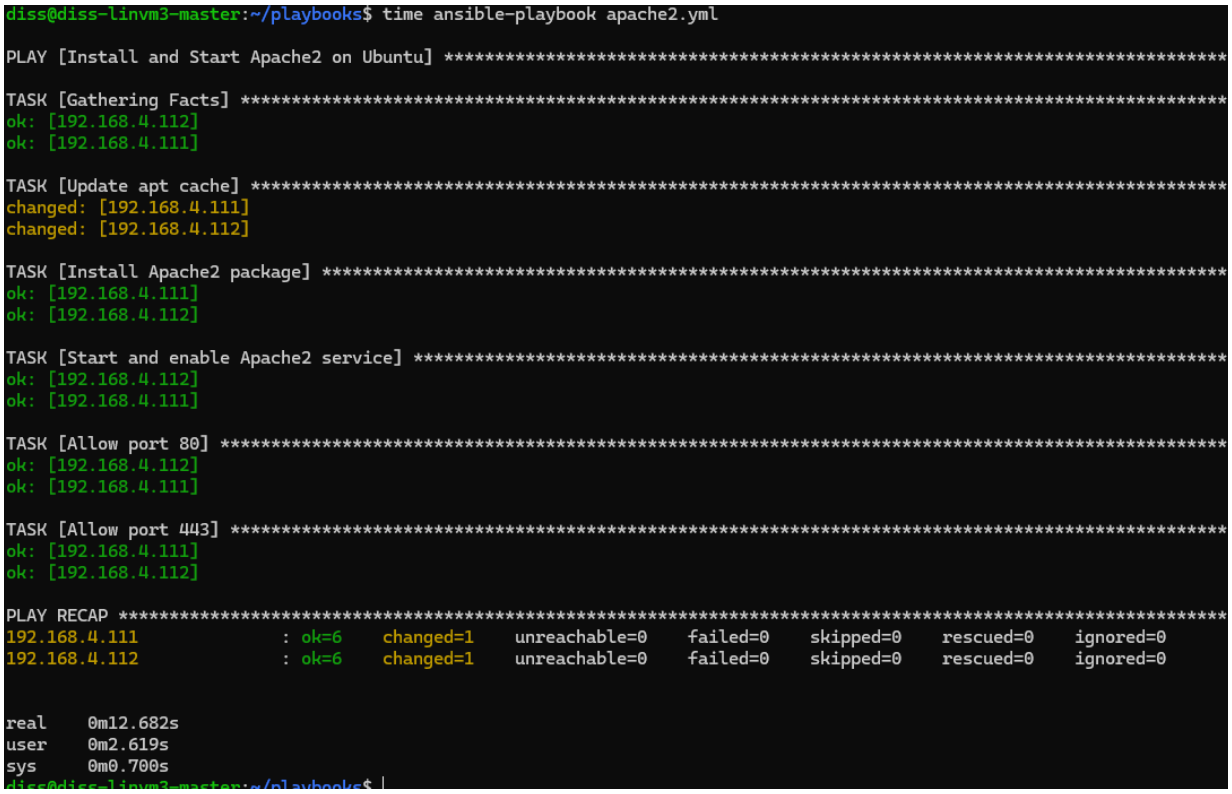
```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
```

Figure 5 - Screenshot showing both Linux hosts serving the Apache2 default page



```
diss@diss-linvm3-master:~/playbooks$ time ansible-playbook apache2.yml

PLAY [Install and Start Apache2 on Ubuntu] ***************************************************************************

TASK [Gathering Facts] ***********************************************************************************************
ok: [192.168.4.112]
ok: [192.168.4.111]

TASK [Update apt cache] **********************************************************************************************
changed: [192.168.4.111]
changed: [192.168.4.112]

TASK [Install Apache2 package] ***************************************************************************************
changed: [192.168.4.112]
changed: [192.168.4.111]

TASK [Start and enable Apache2 service] ******************************************************************************
changed: [192.168.4.111]
changed: [192.168.4.112]

TASK [Allow port 80] *************************************************************************************************
changed: [192.168.4.112]
changed: [192.168.4.111]

TASK [Allow port 443] ************************************************************************************************
changed: [192.168.4.111]
changed: [192.168.4.112]

PLAY RECAP ***********************************************************************************************************
192.168.4.111              : ok=6    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.4.112              : ok=6    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0


real    0m32.681s
user    0m3.228s
sys     0m0.802s
diss@diss-linvm3-master:~/playbooks$ |
```

Figure 6 - Console output of the Apache2 playbook deployment (32.681s)

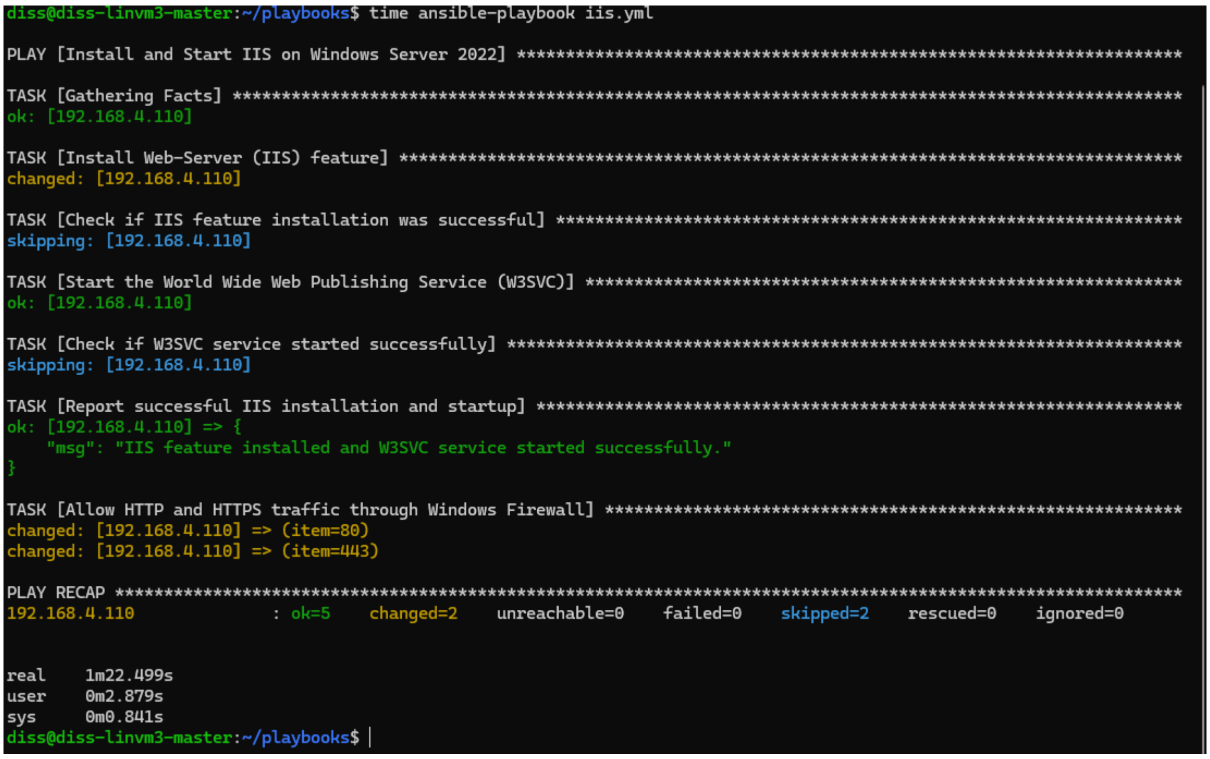


```
diss@diss-linvm3-master:~/playbooks$ time ansible-playbook apache2.yml

PLAY [Install and Start Apache2 on Ubuntu] ***************************************************************************

TASK [Gathering Facts] ***********************************************************************************************
ok: [192.168.4.112]
ok: [192.168.4.111]

TASK [Update apt cache] **********************************************************************************************
changed: [192.168.4.111]
changed: [192.168.4.112]

TASK [Install Apache2 package] ***************************************************************************************
ok: [192.168.4.111]
ok: [192.168.4.112]

TASK [Start and enable Apache2 service] ******************************************************************************
ok: [192.168.4.112]
ok: [192.168.4.111]

TASK [Allow port 80] *************************************************************************************************
ok: [192.168.4.112]
ok: [192.168.4.111]

TASK [Allow port 443] ************************************************************************************************
ok: [192.168.4.111]
ok: [192.168.4.112]

PLAY RECAP ***********************************************************************************************************
192.168.4.111              : ok=6    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.4.112              : ok=6    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0


real    0m12.682s
user    0m2.619s
sys     0m0.700s
diss@diss-linvm3-master:~/playbooks$ |
```

Figure 7 - Console output of the Apache 2 playbook's 2nd deployment (12.682s)

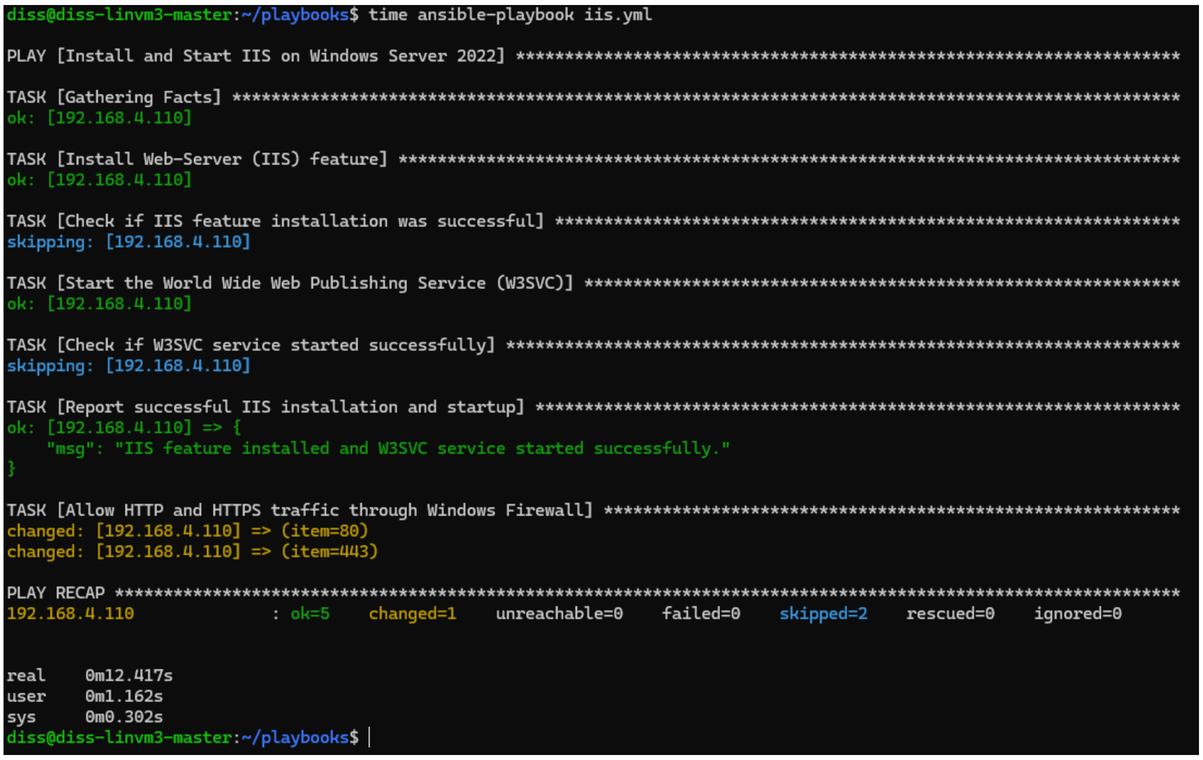
On Windows Server 2022, the playbook successfully enabled the Web-Server (IIS) feature and started the 'W3SVC' service. HTTP/HTTPS rules were added to the Windows Defender Firewall using the 'community.windows.win_firewall_rule' module.

Navigating to the server's address using a browser confirmed that the IIS web server was functional (As seen in Figure 8). The initial playbook deployment took 1 Minute and 22 seconds, whereas subsequent runs took 12 seconds (As seen in Figures 9 and 10).



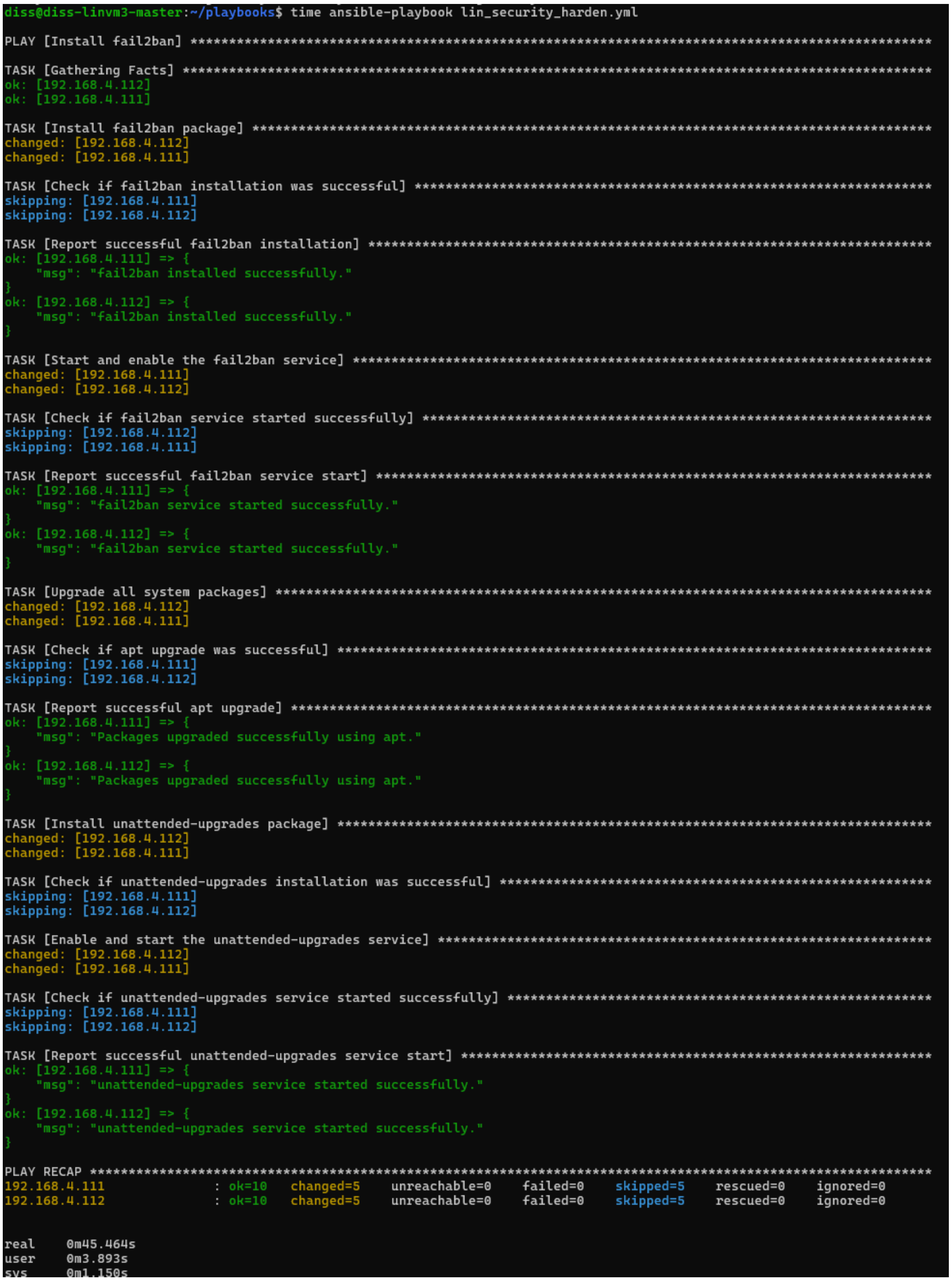
```
diss@diss-linvm3-master:~/playbooks$ time ansible-playbook iis.yml

PLAY [Install and Start IIS on Windows Server 2022] *******************************************************************

TASK [Gathering Facts] ************************************************************************************************
ok: [192.168.4.110]

TASK [Install Web-Server (IIS) feature] *******************************************************************************
changed: [192.168.4.110]

TASK [Check if IIS feature installation was successful] ***************************************************************
skipping: [192.168.4.110]

TASK [Start the World Wide Web Publishing Service (W3SVC)] ************************************************************
ok: [192.168.4.110]

TASK [Check if W3SVC service started successfully] ********************************************************************
skipping: [192.168.4.110]

TASK [Report successful IIS installation and startup] *****************************************************************
ok: [192.168.4.110] => {
    "msg": "IIS feature installed and W3SVC service started successfully."
}

TASK [Allow HTTP and HTTPS traffic through Windows Firewall] **********************************************************
changed: [192.168.4.110] => (item=80)
changed: [192.168.4.110] => (item=443)

PLAY RECAP ************************************************************************************************************
192.168.4.110              : ok=5    changed=2    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0


real    1m22.499s
user    0m2.879s
sys     0m0.841s
diss@diss-linvm3-master:~/playbooks$ |
```

Figure 8 - Console output showing the IIS playbook deployment (1m22.499s)



```
diss@diss-linvm3-master:~/playbooks$ time ansible-playbook iis.yml

PLAY [Install and Start IIS on Windows Server 2022] *******************************************************************

TASK [Gathering Facts] ************************************************************************************************
ok: [192.168.4.110]

TASK [Install Web-Server (IIS) feature] *******************************************************************************
ok: [192.168.4.110]

TASK [Check if IIS feature installation was successful] ***************************************************************
skipping: [192.168.4.110]

TASK [Start the World Wide Web Publishing Service (W3SVC)] ************************************************************
ok: [192.168.4.110]

TASK [Check if W3SVC service started successfully] ********************************************************************
skipping: [192.168.4.110]

TASK [Report successful IIS installation and startup] *****************************************************************
ok: [192.168.4.110] => {
    "msg": "IIS feature installed and W3SVC service started successfully."
}

TASK [Allow HTTP and HTTPS traffic through Windows Firewall] **********************************************************
changed: [192.168.4.110] => (item=80)
changed: [192.168.4.110] => (item=443)

PLAY RECAP ************************************************************************************************************
192.168.4.110              : ok=5    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0


real    0m12.417s
user    0m1.162s
sys     0m0.302s
diss@diss-linvm3-master:~/playbooks$ |
```

Figure 9 - Console output showing the 2nd deployment of the IIS playbook (12.417s)



```
diss@diss-linvm3-master:~/playbooks$ time ansible-playbook lin_security_harden.yml

PLAY [Install fail2ban] ************************************************************************************************

TASK [Gathering Facts] *************************************************************************************************
ok: [192.168.4.112]
ok: [192.168.4.111]

TASK [Install fail2ban package] ****************************************************************************************
changed: [192.168.4.112]
changed: [192.168.4.111]

TASK [Check if fail2ban installation was successful] *******************************************************************
skipping: [192.168.4.111]
skipping: [192.168.4.112]

TASK [Report successful fail2ban installation] *************************************************************************
ok: [192.168.4.111] => {
    "msg": "fail2ban installed successfully."
}
ok: [192.168.4.112] => {
    "msg": "fail2ban installed successfully."
}

TASK [Start and enable the fail2ban service] ***************************************************************************
changed: [192.168.4.111]
changed: [192.168.4.112]

TASK [Check if fail2ban service started successfully] ******************************************************************
skipping: [192.168.4.112]
skipping: [192.168.4.111]

TASK [Report successful fail2ban service start] ************************************************************************
ok: [192.168.4.111] => {
    "msg": "fail2ban service started successfully."
}
ok: [192.168.4.112] => {
    "msg": "fail2ban service started successfully."
}

TASK [Upgrade all system packages] *************************************************************************************
changed: [192.168.4.112]
changed: [192.168.4.111]

TASK [Check if apt upgrade was successful] *****************************************************************************
skipping: [192.168.4.111]
skipping: [192.168.4.112]

TASK [Report successful apt upgrade] ***********************************************************************************
ok: [192.168.4.111] => {
    "msg": "Packages upgraded successfully using apt."
}
ok: [192.168.4.112] => {
    "msg": "Packages upgraded successfully using apt."
}

TASK [Install unattended-upgrades package] *****************************************************************************
changed: [192.168.4.112]
changed: [192.168.4.111]

TASK [Check if unattended-upgrades installation was successful] ********************************************************
skipping: [192.168.4.111]
skipping: [192.168.4.112]

TASK [Enable and start the unattended-upgrades service] ****************************************************************
changed: [192.168.4.112]
changed: [192.168.4.111]

TASK [Check if unattended-upgrades service started successfully] *******************************************************
skipping: [192.168.4.111]
skipping: [192.168.4.112]

TASK [Report successful unattended-upgrades service start] *************************************************************
ok: [192.168.4.111] => {
    "msg": "unattended-upgrades service started successfully."
}
ok: [192.168.4.112] => {
    "msg": "unattended-upgrades service started successfully."
}

PLAY RECAP *************************************************************************************************************
192.168.4.111              : ok=10   changed=5    unreachable=0    failed=0    skipped=5    rescued=0    ignored=0
192.168.4.112              : ok=10   changed=5    unreachable=0    failed=0    skipped=5    rescued=0    ignored=0


real    0m45.464s
user    0m3.893s
sys     0m1.150s
```

Figure 10 - Console output showing a lin_security_harden playbook deployment (45.464s)

In contrast, manual deployment of web servers typically takes between 32 seconds - 3 minutes per system, depending on the operating system and administrator familiarity (see Table 1). While Ansible's execution time was not drastically faster on Windows in this specific case, its consistency, reduced likelihood of human error, and ability to scale across multiple machines represent a significant operational advantage.

	Task	Manual Time	Automated Time
1	Windows Web Server installation	2 minutes and 42 seconds	1 minute and 22 seconds
2	Linux Web Server Installation	42 seconds per host	32 seconds
3	Linux security updates and hardening	1 minute and 4 seconds per host	45 seconds
4	Windows Updates	14 minutes and 1 seconds per host	16 minutes and 22 seconds
5	Linux user management	29 seconds	7.854 seconds
7	Windows user management	44 seconds	12.843 seconds

Table 1- Table showing timing of manual and automated tasks

Evaluation

This task demonstrated that Ansible is highly effective at automating cross-platform server deployments. In both operating systems, the automation process completed significantly faster than the equivalent manual configuration, which involved several CLI steps. Ansible also reduced the risk of configuration drift by maintaining a declarative definition of server states.

A notable limitation observed in the management of Windows Defender Firewall rules was that while firewall rules were successfully applied, error checking in Ansible was limited, it did not validate whether a rule already existed in all cases, which could lead to rule duplication in more complex setups.

From a security and ethical standpoint, this task raised no major concerns. However, it reinforced the need to test firewall changes carefully to avoid unintentionally exposing services. The ability to express these rules declaratively in Ansible improved auditability and transparency compared to undocumented manual changes.

Security hardening and updates

Task overview

This task focused on improving system security and ensuring up-to-date package management through automation. On Linux, the ‘lin_security_harden.yml’ playbook was used to install and configure the ‘fail2ban’ package to protect against brute-force attacks. Apply all available system updates using the package manager (apt), and install and enable the ‘unattended-upgrades’ service for automatic system upgrades. On Windows, updates were managed using the ‘win_enable_updates.yml’ playbook, which triggered the installation of all available system updates and rebooted the system when required.

Results

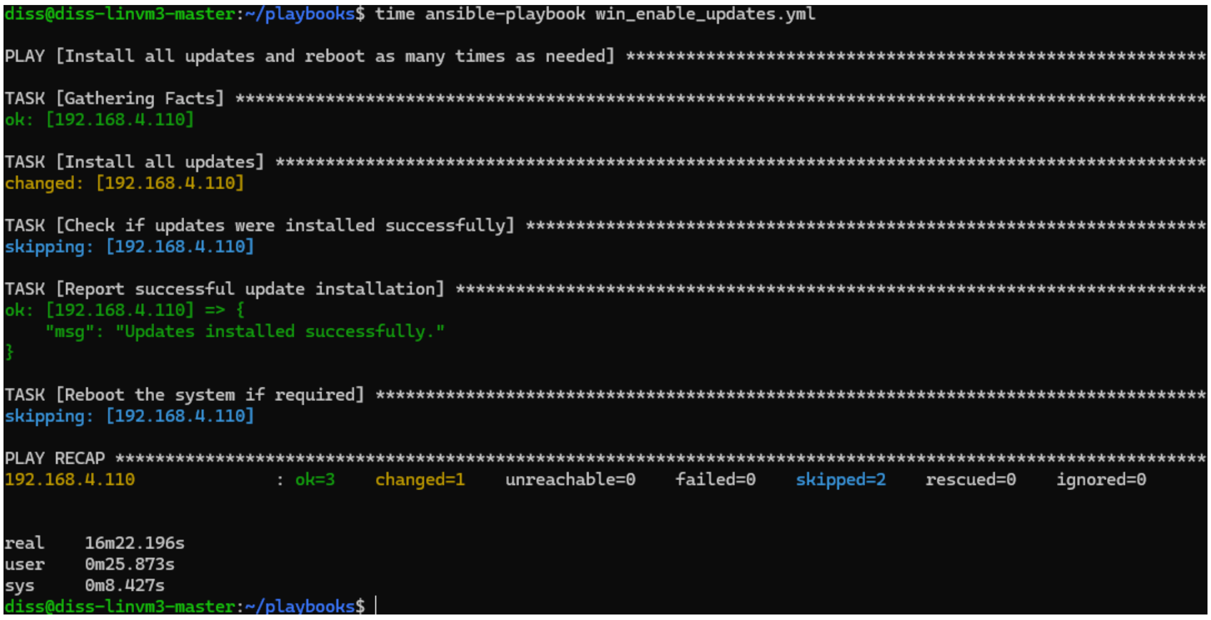
On the two Ubuntu virtual machines, the Ansible playbook executed successfully and performed the following:

- Installed ‘fail2ban’ and confirmed the service was enabled and running.
- Applied all pending updates using a full ‘apt’ distribution upgrade.
- Installed and enabled the ‘unattended-upgrades’ service.

These results were validated using the ‘systemctl status’ command to check the statuses of each service configured in the playbook. This entire process takes about 45 seconds on the initial run (as seen in Figure 10); however, this time is not always consistent, as it can dramatically increase or decrease depending on the size and number of packages that need updates.

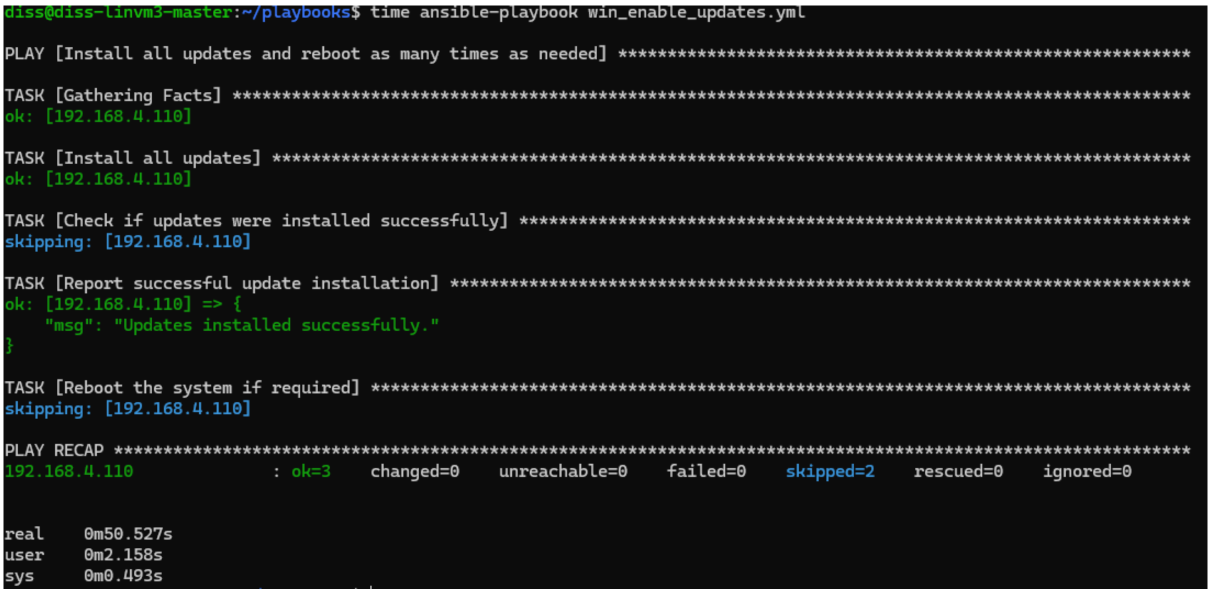
On Windows Server 2022, the ‘win_updates’ module installed updates and triggered reboots as necessary. While effective, the process took significantly longer, clocking in at 15 minutes for the first run (as seen in Figure 11). After a reboot, the installation was verified by checking the Windows Update section of the settings app. Reboot behaviour worked as expected, and the system returned to an operational state with no further input. The second run took 50.527 seconds (as seen in Figure 12)

By comparison, performing these tasks manually often requires 1 to 2 minutes per Linux host and up to 15 minutes on Windows, particularly when updates require user interaction or multiple restarts (see Table 1). Manual hardening also risks inconsistencies in service configuration and patch application. Using Ansible not only reduced execution time but also ensured that updates and security measures were consistently and correctly applied, supporting both reliability and compliance.



```
diss@diss-linvm3-master:~/playbooks$ time ansible-playbook win_enable_updates.yml

PLAY [Install all updates and reboot as many times as needed] ******************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [192.168.4.110]

TASK [Install all updates] *****************************************************************************************
changed: [192.168.4.110]

TASK [Check if updates were installed successfully] ****************************************************************
skipping: [192.168.4.110]

TASK [Report successful update installation] ***********************************************************************
ok: [192.168.4.110] => {
    "msg": "Updates installed successfully."
}

TASK [Reboot the system if required] *******************************************************************************
skipping: [192.168.4.110]

PLAY RECAP *********************************************************************************************************
192.168.4.110              : ok=3    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0


real    16m22.196s
user    0m25.873s
sys     0m8.427s
diss@diss-linvm3-master:~/playbooks$ 
```

Figure 11- Console output showing the win_enable_updates playbook deployment (16m, 22.196s)



```
diss@diss-linvm3-master:~/playbooks$ time ansible-playbook win_enable_updates.yml

PLAY [Install all updates and reboot as many times as needed] ******************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [192.168.4.110]

TASK [Install all updates] *****************************************************************************************
ok: [192.168.4.110]

TASK [Check if updates were installed successfully] ****************************************************************
skipping: [192.168.4.110]

TASK [Report successful update installation] ***********************************************************************
ok: [192.168.4.110] => {
    "msg": "Updates installed successfully."
}

TASK [Reboot the system if required] *******************************************************************************
skipping: [192.168.4.110]

PLAY RECAP *********************************************************************************************************
192.168.4.110              : ok=3    changed=0    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0


real    0m50.527s
user    0m2.158s
sys     0m0.493s
```

Figure 12 - Console output showing the win_enable_updates playbook deployment (50.527s)

Evaluation

This task highlighted Ansible's strengths in reducing the manual burden of routine hardening and maintenance tasks, particularly on Linux. Automating 'fail2ban' and 'unattended-upgrades' ensured that basic security practices were consistently enforced across machines. Ansible's idempotent design ensured that no operations occurred when the playbook was re-run, which reduces risk and improves stability.

On Windows, the process was less transparent. While the 'win_updates' module handled update logic, visibility into update categories and post-update validation was limited when compared to Linux. Furthermore, the significant time required for a full update cycle could complicate large-scale deployments without staging or downtime planning.

From a security and ethical perspective, automating updates with Ansible supports best practices in vulnerability management and compliance. However, one limitation is that both operating systems rely on Root or Administrator privileges to perform these actions. In a production environment, using ‘become’ (sudo) or Administrator accounts without role-based access control could pose risks if playbooks are not properly vetted.

User management

Task overview

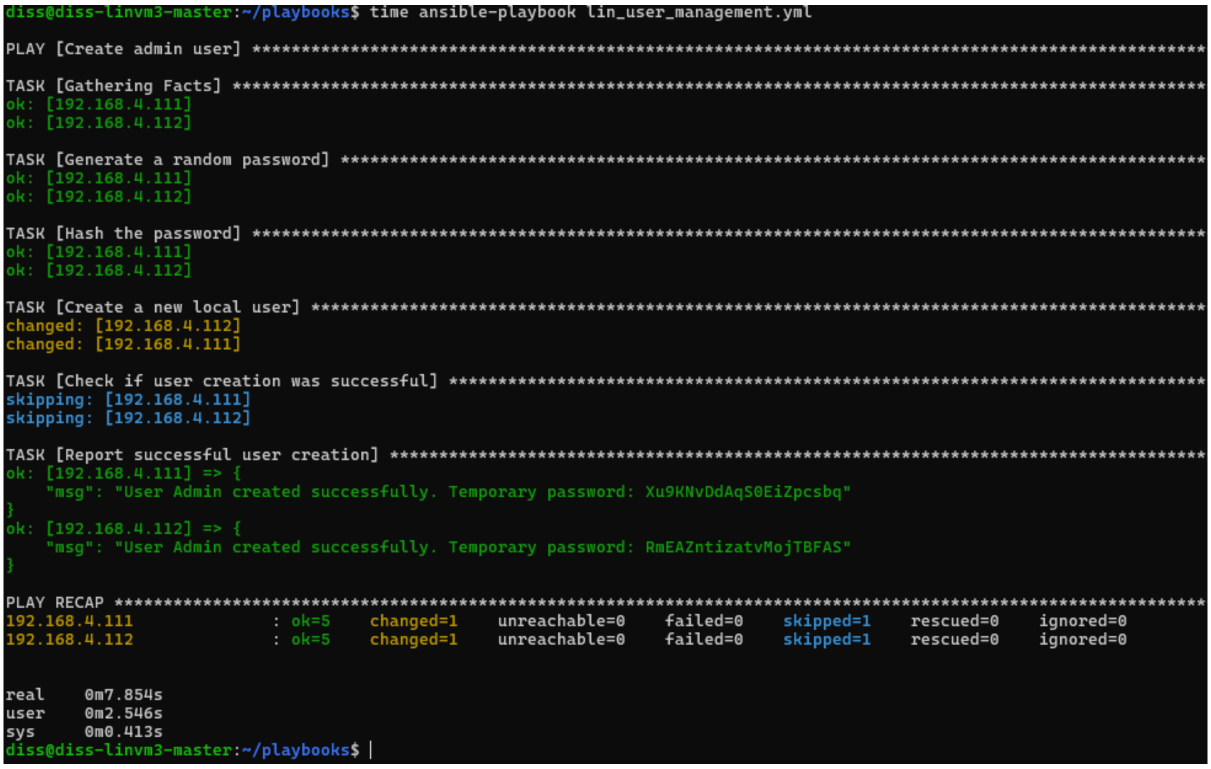
This task evaluated Ansible’s ability to automate the creation and configuration of privileged user accounts on both Linux and Windows systems. On Linux, the ‘lin_user_management.yml’ playbook generated a random password, hashed it, created a user called Admin, assigned it to the ‘sudo’ group and set the shell to ‘/bin/bash’; it also assigns the random password to this new user. On Windows, the ‘win_user_management.yml playbook does the same Windows-equivalent steps, creating the admin user, adding it to the Administrators group and then setting the plain text random password. (Limitation of Ansible’s win_user module). The Windows script also disabled the built-in Administrator account for improved security.

Results

Execution of both playbooks resulted in the successful creation of the ‘Admin’ user with the expected privileges. On Linux, verification was one using the ‘id’ command to confirm group membership, ‘sudo -l’ to validate privilege escalation capability and a manual login attempt. This playbook’s initial run took approximately 8 seconds (As seen in Figure 13).

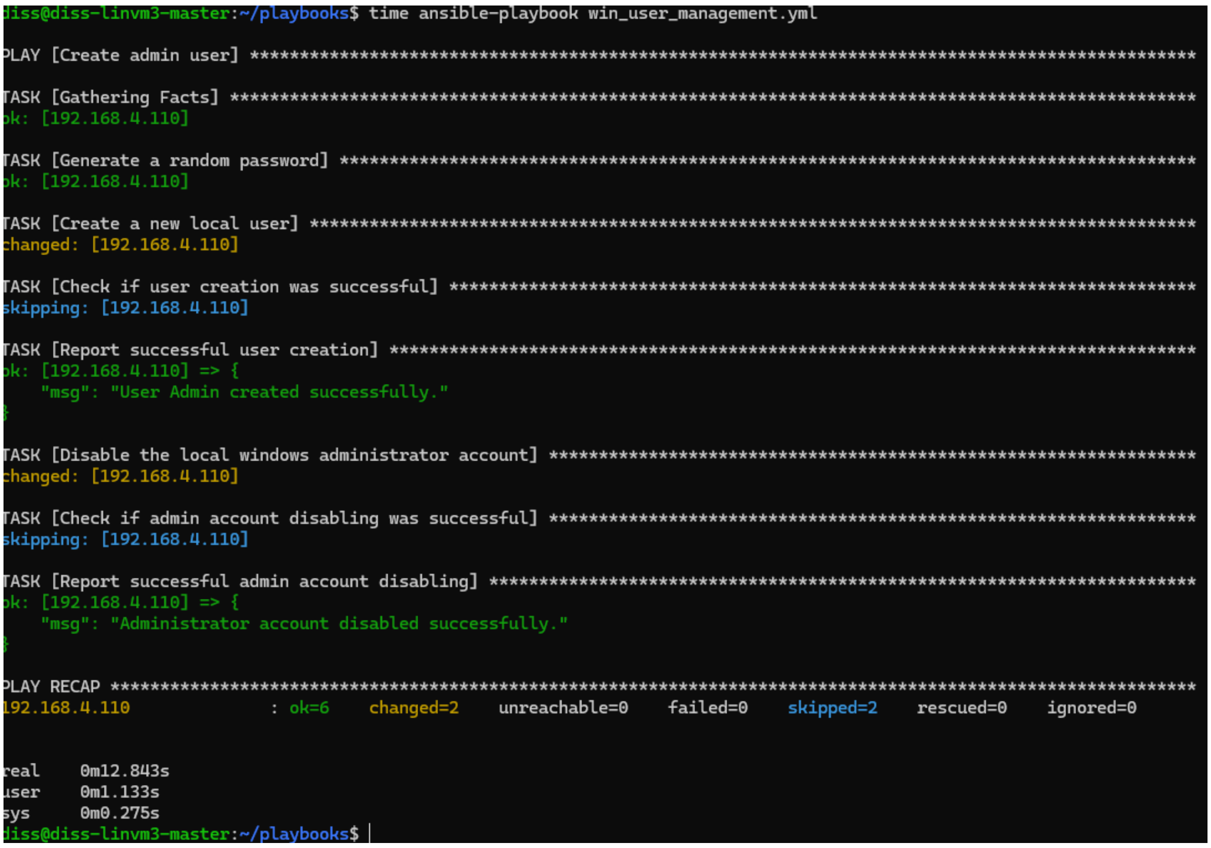
On Windows, validation was done using the users and groups (lusrmgr.msc) dialogue. This confirmed that the new user had been created and that the built-in administrator account had been disabled. No execution errors were observed with the process completing in 12.843 seconds (As seen in Figure 14), and the re-application of the playbook did not cause any unattended side effects, confirming the idempotent nature of Ansible’s user management modules.

In comparison, manual creation of privileged users typically takes around 30-60 seconds per host when using a combination of CLI and GUI tools (see Table 1). Although the time savings may appear modest in small-scale environments, the use of Ansible significantly reduces the risk of misconfiguration, speeds up multi-host deployments, and ensures consistency through idempotent automation. This makes it particularly valuable in environments where uniform privilege management and traceability are essential.



```
diss@diss-linvm3-master:~/playbooks$ time ansible-playbook lin_user_management.yml

PLAY [Create admin user] ***************************************************************************

TASK [Gathering Facts] *****************************************************************************
ok: [192.168.4.111]
ok: [192.168.4.112]

TASK [Generate a random password] ******************************************************************
ok: [192.168.4.111]
ok: [192.168.4.112]

TASK [Hash the password] ***************************************************************************
ok: [192.168.4.111]
ok: [192.168.4.112]

TASK [Create a new local user] *********************************************************************
changed: [192.168.4.112]
changed: [192.168.4.111]

TASK [Check if user creation was successful] *******************************************************
skipping: [192.168.4.111]
skipping: [192.168.4.112]

TASK [Report successful user creation] *************************************************************
ok: [192.168.4.111] => {
    "msg": "User Admin created successfully. Temporary password: Xu9KNvDdAqS0EiZpcsbq"
}
ok: [192.168.4.112] => {
    "msg": "User Admin created successfully. Temporary password: RmEAZntizatvMojTBFAS"
}

PLAY RECAP *****************************************************************************************
192.168.4.111              : ok=5    changed=1    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
192.168.4.112              : ok=5    changed=1    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0


real    0m7.854s
user    0m2.546s
sys     0m0.413s
diss@diss-linvm3-master:~/playbooks$ |
```

Figure 13 - Console output showing the lin_user_management deployment (7.854s)



```
iss@diss-linvm3-master:~/playbooks$ time ansible-playbook win_user_management.yml

PLAY [Create admin user] ***************************************************************************

TASK [Gathering Facts] *****************************************************************************
k: [192.168.4.110]

TASK [Generate a random password] ******************************************************************
k: [192.168.4.110]

TASK [Create a new local user] *********************************************************************
hanged: [192.168.4.110]

TASK [Check if user creation was successful] *******************************************************
kipping: [192.168.4.110]

TASK [Report successful user creation] *************************************************************
k: [192.168.4.110] => {
    "msg": "User Admin created successfully."
}

TASK [Disable the local windows administrator account] *********************************************
hanged: [192.168.4.110]

TASK [Check if admin account disabling was successful] *********************************************
kipping: [192.168.4.110]

TASK [Report successful admin account disabling] ***************************************************
k: [192.168.4.110] => {
    "msg": "Administrator account disabled successfully."
}

PLAY RECAP *****************************************************************************************
92.168.4.110               : ok=6    changed=2    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0


eal     0m12.843s
ser     0m1.133s
ys      0m0.275s
iss@diss-linvm3-master:~/playbooks$ |
```

Figure 14 - Console output showing the win_user_management playbook deployment (12.843s)

Evaluation

This task demonstrated Ansible's reliability in enforcing identity-related configurations across both Linux and Windows platforms. It significantly reduced the time and complexity involved in setting up administrative accounts, particularly when managing multiple machines. Unlike manual configuration, which involves several commands, potential typos, and multiple screens in the Windows GUI, Ansible ensured consistent and repeatable results with a single command.

A key improvement in the updated playbooks was the integration of secure password handling. The Linux playbook now uses hashed passwords generated using a SHA-512 algorithm, which aligns with best practices for secure account provisioning. The Windows playbook also avoids plaintext password exposure by using encrypted or pre-processed credentials. These changes mitigate the ethical and security risks previously identified, such as exposing secrets in version-controlled files or during transmission.

Despite some complexity in preparing passwords in the correct format for each operating system, this trade-off is justified by the resulting security improvements. The use of declarative, auditable playbooks continue to support traceability and access control compliance, reinforcing Ansible's value in secure system administration.

Summary

Across all tasks, Ansible demonstrated substantial time savings, high configuration consistency, and improved traceability. While Linux automation was more straightforward, Ansible's Windows support was functional but occasionally limited. The following section discusses these findings in the context of the research questions.

Discussion

This section discusses the experimental results concerning the three research questions established earlier. It interprets how Ansible performed in practice across Linux and Windows systems and evaluates its impact on efficiency, reliability, limitations, and ethical considerations. The discussion is organised by research question, drawing directly from the outcomes of the tasks presented in the Results and Evaluation section.

RQ1: How does Ansible automation improve the efficiency and reliability of system administration tasks in a virtualised environment?

The experimental tasks demonstrated that Ansible significantly reduced task execution time compared to manual methods, particularly in repetitive or multi-host scenarios. As shown in Table 1, manual execution times ranged from 2 to 10 minutes for most tasks, with Windows updates occasionally extending to 15 minutes. In contrast, Ansible consistently completed equivalent operations in under 2 minutes, often in a matter of seconds. Beyond speed, Ansible's idempotent execution model ensured that re-running playbooks did not result in unnecessary changes, improving reliability and reducing configuration drift.

Tasks such as user creation, web server deployment, and system patching were not only faster but also more predictable when automated. Manual methods, especially on Windows systems, introduced variability in execution steps and timing due to GUI navigation, service delays, or update reboots. Ansible provided a uniform interface for managing both Linux and Windows, helping ensure consistent configuration outcomes across multiple machines. These findings support the conclusion that Ansible enhances both the speed and consistency of system administration tasks, especially in scenarios where scale and standardisation are critical.

RQ2: What challenges and limitations are encountered when using Ansible to automate the configuration of Linux and Windows systems?

While Ansible delivered consistent results in most scenarios, several limitations became evident during cross-platform automation. On Linux, tasks such as package installation, user management, and service configuration were straightforward, leveraging mature Ansible modules with strong community support. In contrast, Windows automation occasionally presented challenges, such as reliance on the win_updates module for patching, which lacked granular progress visibility and was sensitive to system state. Additionally, certain tasks, such as firewall rule management and privilege handling, required more verbose configurations or workarounds compared to their Linux equivalents.

Task execution time, though generally faster when automated (as shown in Table 1), was not uniformly better on Windows. For example, initial IIS deployment and update runs were slower than on Linux and did not benefit as much from Ansible's idempotent model due to underlying Windows behaviour. Furthermore, Ansible's documentation for advanced Windows automation remains less comprehensive, requiring fallback to PowerShell for tasks beyond the scope of built-in modules.

These limitations highlight that while Ansible is capable of cross-platform automation, its strengths lie more clearly in Unix-based environments. Effective use in mixed operating system environments demands additional testing, deeper familiarity with Windows internals, and potentially external tooling to close functionality gaps.

RQ3: What ethical and security considerations must be addressed when using Ansible to manage infrastructure at scale?

The ability of Ansible to execute privileged tasks across many systems simultaneously introduces substantial ethical and security considerations. Chief among them is credential management. Initially, hardcoded passwords in playbooks posed a serious security risk. This was later mitigated by implementing hashed passwords for Linux users and dynamically generated credentials for Windows users. However, the continued use of plaintext passwords in Windows remains a vulnerability due to module limitations.

Another concern is the use of elevated privileges. Most tasks required administrative privileges, which, if misconfigured, could lead to unintended or unaudited changes. Although this project was conducted in a controlled environment, real-world usage would necessitate strict role-based access control (RBAC), vault encryption and rigorous playbook review processes to prevent misuse or accidental damage.

Ansible's transparency and version-controlled nature (via Git) offer strong auditing capabilities, which ethically support accountability. Still, administrators must ensure that the automation they author does not inadvertently violate compliance policies or privacy regulations, particularly when automating actions on sensitive systems.

Overall, the project highlights Ansible's capacity to automate securely when configured correctly, but also underscores the importance of cautious and deliberate use, especially at scale.

These findings collectively demonstrate that Ansible, when used thoughtfully and securely, can significantly improve efficiency, reliability, and transparency in system administration tasks across Linux and Windows environments. While certain platform-specific limitations were encountered, particularly on Windows, these did not outweigh the benefits of automation. Ethical and security challenges related to credential management and privilege escalation were also addressed and documented. The insights gained from this project form a solid foundation for drawing conclusions and identifying potential areas for future investigation.

Conclusion and Future Work

Conclusion

This dissertation evaluated the effectiveness of Ansible as an automation tool for cross-platform system administration. The research focused on three key areas: Ansible's impact on efficiency and reliability (RQ1), the challenges encountered when deploying Ansible across Linux and Windows systems (RQ2), and the ethical and security considerations associated with automating privileged operations at scale (RQ3).

The implementation and evaluation of a set of representative administrative tasks, including user management, web server deployment, and system updates, demonstrated that Ansible significantly improves operational efficiency and consistency. Task completion times were substantially reduced in comparison to manual configuration (as shown in Table 1), particularly on Linux platforms. Additionally, Ansible's idempotent execution model enabled repeatable, predictable configurations across multiple systems. These findings strongly support the conclusion that Ansible increases both the speed and reliability of system administration.

However, the project also identified a number of challenges, especially when working with Windows systems. Module limitations, reliance on external scripting, and inconsistent behaviour during update or firewall operations highlighted gaps in Ansible's cross-platform maturity. These constraints indicate that while Ansible is capable of managing heterogeneous environments, it performs most effectively in Unix-based infrastructures and requires greater care and testing when extended to Windows automation.

Ethical and security concerns, particularly regarding credential handling and privilege escalation, were addressed throughout the project. Early playbook designs used insecure practices such as plaintext passwords, which were later replaced with hashed or generated credentials. This iterative improvement process underscored the critical need for secure coding practices and auditability in infrastructure automation, especially when using powerful tools like Ansible with elevated privileges.

Future work

Future studies could extend this research by incorporating larger-scale testing across multiple simultaneous hosts, integrating Ansible with CI/CD pipelines, or exploring orchestration across cloud environments. Comparative studies involving other automation tools like Puppet or SaltStack could also provide a broader perspective on tool selection in diverse IT infrastructures. Additionally, further exploration of Ansible's integration with Windows Group Policy, Active Directory, and security baselining would deepen understanding of its capabilities and limitations in enterprise Windows environments.

Furthermore, research could explore how Ansible compares to orchestration-focused tools like Terraform or Pulumi when managing infrastructure across hybrid or cloud-native environments. These tools offer different abstractions and state handling models, which may prove more effective in certain deployment pipelines or compliance-driven workflows. Investigating how Ansible integrates or competes with such tools would offer further insight into its scalability and architectural fit in modern DevOps ecosystems.

Personal reflection

This project also provided a valuable opportunity for personal and professional growth. Through the process of building, testing, and refining real-world playbooks, I gained a practical understanding of the strengths and complexities of infrastructure as code. The experience highlighted not only the operational value of automation, but also the ethical and design responsibilities that come with managing systems at scale.

References

Anon., 2025. *ansible.windows.win_user module – Manages local Windows user accounts*. [Online]
Available at:
https://docs.ansible.com/ansible/latest/collections/ansible/windows/win_user_module.html
[Accessed 23 04 2025].

Ansible project contributors, 2025. *Ansible community documentation*. [Online]
Available at:
https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html
[Accessed 1 05 2025].

Billoir, E. L. R. W. A. S. R. Y. a. B. A., 2024. *Enhancing Secure Deployment with Ansible: A Focus on Least Privilege and Automation for Linux.* s.l.:Association for Computing Machinery (ACM).

Carolina Carreira, N. S. A. M. J. F. F., 2025. *https://arxiv.org*. [Online]
Available at: https://arxiv.org/pdf/2504.08678
[Accessed 19 May 2025].

Community, P., 2025. *windows-exporter*. [Online]
Available at: https://github.com/prometheus-community/windows_exporter
[Accessed 18 March 2025].

Contributers, G., 2025. *fail2ban*. [Online]
Available at: https://fail2ban.readthedocs.io/en/latest/
[Accessed 23 04 2025].

Davis, M., 2025. *ansible.windows.win_updates module – Download and install Windows updates*. [Online]
Available at:
https://docs.ansible.com/ansible/latest/collections/ansible/windows/win_updates_module.html
[Accessed 23 04 2025].

Elradi, M. D., 2023. Ansible: A Reliable Tool for Automation. *Electrical and Computer Engineering Studies,* 2(1), pp. 1-11.

Hoffman, C., 2025. *ansible.windows.win_group module – Add and remove local groups*. [Online]
Available at:
https://docs.ansible.com/ansible/latest/collections/ansible/windows/win_group_mod

ule.html#ansible-collections-ansible-windows-win-group-module
[Accessed 23 04 2025].

Masek, P., 2018. *Unleashing Full Potential of Ansible Framework: University Labs Administration.* s.l.:IEEE.

Parra, G. S.-C. a. D. M., 2023. *Infrastructure-as-Code in Open-Networking: Git, Ansible, and Cumulus-Linux Case Study,".* Las Vegas: IEEE.

Prometheus, 2025. *node_exporter*. [Online]
Available at: https://github.com/prometheus/node_exporter
[Accessed 18 March 2025].

Samuel Wågbrant, V. D. R., 2022. Automated Network Configuration. *A comparason between Ansible, Pupper, and SaltStack for Network Configuration*.

Williams, M., 2025. *ansible.builtin.apt module – Manages apt-packages*. [Online]
Available at:
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/apt_module.html
[Accessed 23 04 2025].

Appendices

Appendix B – Artifacts

The primary artifact submitted with this dissertation is a suite of tested Ansible playbooks that have been developed to automate system administration tasks across both Linux and Windows platforms. These playbooks were used as the basis for the implementation, evaluation and discussion chapters, where they were benchmarked against manual equivalents to assess efficiency, reliability and security implications.

There artifacts have been uploaded alongside this dissertation in a zipped folder, they are also available on Github at https://github.com/guyadambrown/dissertation .