Henry Jaas

AUTOMATED SERVER INFRASTRUCTURE DEPLOYMENT

Technology
2025

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Henry Jaas
Opinnäytetyön nimi	Automatisoitu palvelininfrastruktuurin käyttöönotto
Vuosi	2025
Kieli	englanti
Sivumäärä	44
Ohjaaja	Mikael Jakas

Nykyisessä toimintamallissa Wärtsilän henkilöstön on matkustettava asiakkaiden luo asentamaan ja konfiguroimaan virtuaaliympäristöt manuaalisesti. Tämä prosessi on kallis ja aikaa vievä. Tämän työn tavoitteena on kehittää automatisoitu käyttöönottojärjestelmä, joka voidaan suorittaa etänä ilman fyysistä läsnäoloa. Tämän avulla voidaan vähentää kustannuksia ja nopeuttaa käyttöönottoa.

Työn teoreettinen viitekehys perustuu infrastruktuurin automaation ja konfiguraationhallintaan hyödyntäen Debiania, Windows Remote Managementia, Ansiblea, Semaphore UI:ta ja Microsoft Deployment Toolkitiä. Keskeisiä käsitteitä ovat infrastruktuurin automaatio, konfiguraationhallinta, pilviympäristöjen provisiointi ja Windows-virtuaalikoneiden käyttöönotto. Tutkimusmenetelmänä käytettiin toimintatutkimusta. Tutkimusaineisto koostuu järjestelmän lokitiedostoista, käyttöönottotuloksista ja asiantuntijoiden palautteesta.

Työ osoittaa, että automatisoitu käyttöönotto voi merkittävästi vähentää manuaalisen työn tarvetta ja virheiden määrää. Opinnäytetyö mahdollistaa järjestelmien nopeamman ja selkeämmän provisioinnin sekä helpottaa ylläpitoa. Johtopäätöksenä voidaan todeta, että infrastruktuurin automaatio parantaa käyttöönoton tehokkuutta ja vähentää operatiivisia kustannuksia merkittävästi

Avainsanat provisiointi, konfiguraationhallinta, Ansible, virtuaaliympäristö, Semaphore UI, infrastruktuuri

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Henry Jaas
Title	Automated Server Infrastructure Deployment
Year	2025
Language	English
Pages	44
Name of Supervisor	Mikael Jakas

In the current operational model, Wärtsilä personnel must travel to customer sites to install and configure virtual environments manually. This process is costly and time-consuming. The aim of this thesis is to develop an automated deployment process that can be executed remotely without physical presence, reducing costs and accelerating deployment.

The theoretical framework of the study is based on infrastructure automation and configuration management, utilizing Debian, WinRM, Ansible, Semaphore UI, and Microsoft Deployment Toolkit. Key concepts include infrastructure automation, configuration management, cloud environment provisioning, and the Windows virtual machine deployment. The research method applied is action research. The research data consists of system log files, deployment results, and expert feedback.

The study demonstrates that automated deployment can significantly reduce the need for manual work and the number of errors. The thesis enables faster and more structured provisioning of systems and facilitates maintenance. The conclusion is that infrastructure automation improves deployment efficiency and significantly reduces operational costs.

Keywords	provisioning, configuration management, Ansible, virtual environment, Semaphore UI, infrastructure

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

TERMS AND ABBREVIATIONS

AD: Active Directory. A directory service in Windows environments used to manage users, groups, and devices.

API: Application Programming Interface. It is a programming interface.

Ansible Playbook: A script that contains processes related to setting up a computer.

Bitbucket: A version control software where virtual machine playbooks, scripts, and configurations are stored.

FAT: Factory Acceptance Test. A factory test used to verify that the tested product meets the documented requirements and specifications.

GPO: Group Policy Object. A group policy used in Windows environments.

Hardening: The process of securing a system by reducing vulnerabilities, implementing security configurations, and enforcing strict access controls.

ISO: An ISO file that contains the installation files for an operating system.

MDT: Microsoft Deployment Toolkit, a deployment tool for Windows machines.

Provisioning: The process of setting up a system or virtual machine with necessary configurations before deployment

Repository: An archive in a version control system

Rest API: A specific API model. It includes standard HTTP requests such as GET, POST, PUT, and DELETE.

UI: User Interface.

VM: Virtual Machine. An isolated operating system that can be run locally or in a cloud environment.

WinRM: Windows Remote Management. A tool that allows remote commands to be executed on a Windows machine.

1 INTRODUCTION

Wärtsilä has teams that prepare the sWOIS application for power plants on the customer's premises. The application is developed by Wärtsilä and includes, among other things, a management system, device integration, data storage, remote access to the power plant, and reporting. sWOIS is a Windows-based application that runs in a complex environment, requiring specific system configurations for deployment. Setting up this environment has traditionally been automated using Microsoft Deployment Toolkit (MDT). However, this method requires expert supervision and can take several days to establish a fully functional deployment.

The purpose of this thesis is to build a Debian-based virtual machine that hosts Ansible and Semaphore UI to improve and automate the deployment process. This system will be used to remotely configure and prepare existing Windows machines for sWOIS installation, reducing manual work and speeding up deployment. While Ansible provides an efficient way to manage configurations remotely, MDT will still be used for the initial steps, such as operating system installation and base configuration, since Ansible requires an existing OS to operate. Once the base setup is completed, all additional configurations, role assignments, and deployments can be fully automated using Ansible and Semaphore UI.

With Semaphore UI, experts can manage servers through a graphical interface, simplifying system maintenance and making the onboarding of new machines significantly easier. The automation framework ensures that deployments are consistent, repeatable, and less prone to human error. This thesis is particularly important for Wärtsilä's energy business, as it optimizes expert time and reduces the need for on-site installations at customer locations. The automation process leads to cost savings and increased efficiency, benefiting both Wärtsilä and its customer.

2 CURRENT STATE AND DEVELOPMENT REQUIREMENTS

2.1 Current Operating Model and Its Challenges

In the current operating model, it is necessary to travel to client sites to install and configure virtual environments manually. This process is time-consuming and costly, as the installations require physical presence. This results in travel expenses and a loss of productive work hours

The installation process involves transferring .iso files created by MDT from Azure to the client's environment. This means that each .iso file must be copied individually whenever new updates are released. Since the installation must be performed by one machine at a time, this significantly increases the workload and makes system updates and maintenance cumbersome and time-consuming.

The lack of remote management means that potential issues often require physical presence, which can delay troubleshooting and corrective actions. This may cause production interruptions and increase operational costs. These challenges highlight the need to develop a comprehensive automated deployment system that enables remote management and reduces manual work.

2.2 Automation Objectives and Requirement Specification

For the project, a requirement specification was carefully defined in collaboration with the team. The requirement specification and priorities have been documented accordingly (Table 1).

Priority Levels:

1 = Mandatory

2 = Essential but not critical

3= Optional

Table 1. Requirement Specification

Requirement	Priority	Details
User Interface	1	SemaphoreUI (open source)
Windows ja Linux Playbooks	1	Playbook versions for existing virtual machines
Documentation	1	User manuals
Version Control	2	A ready version control environment for the project
Logging and Reporting	3	Semaphore UI includes its own logging, so no separate logging is required

3 SOFTWARE AND AUTOMATION TOOLS USED

3.1 Ansible

Ansible is an open-source automation and configuration management tool used for infrastructure management, software installation, and system configuration automation (Red Hat, 2025). Ansible was acquired by Red Hat in 2015 and has been maintained by Red Hat ever since (Wikipedia, n.d.). It enables system management without requiring additional agent software, as it uses an agentless architecture and connects Linux systems through SSH and to Windows systems through WinRM (Red Hat, 2025). This means that target machines do not need separate installation software; it is sufficient that Ansible is installed on the host machine.

Its key components include the inventory, modules, and the control machine, from which commands are executed. The inventory defines the target devices to be managed, modules perform individual tasks, and the control machine is the system where Ansible is installed and from which commands are executed (Red Hat, 2025).

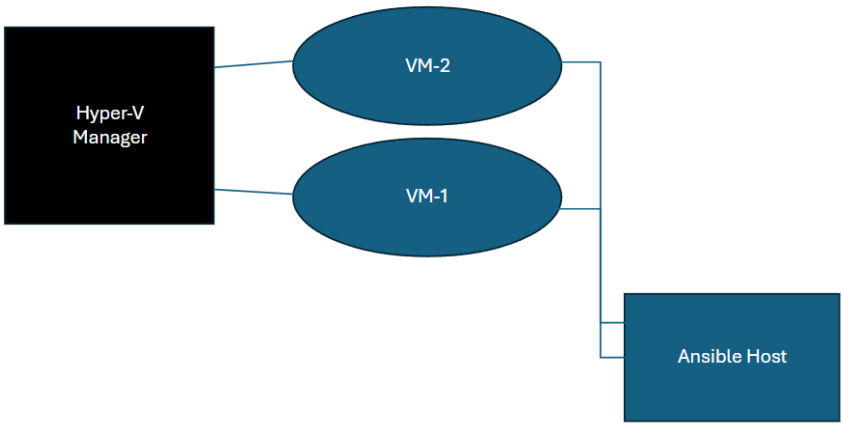

Figure 1. Ansible Example

Ansible uses YAML-formatted Playbooks to define tasks and system configurations. A single playbook can contain multiple tasks or even all configurations for a server. However, for clarity and easier updates, it is better to create multiple smaller playbooks.

```yaml
- name: Copy Specific Folders to Target Machine
  hosts: windows
  tasks:
    - name: Copy Scripts folder
      ansible.windows.win_copy:
        src: /tmp/repo/Scripts
        dest: C:\Scripts\
        remote_src: no

    - name: Copy Files folder
      ansible.windows.win_copy:
        src: /tmp/repo/Files
        dest: C:\Files\
        remote_src: no
```

Figure 2. Playbook Example

In the playbook example, the roles are as follows:

- **hosts** specifies the target where the tasks will be executed. In this case, the machines in the "windows" group.
- **tasks** contains a list of tasks to be executed sequentially. In the example, there are two tasks.
- **modules** define what the task does. In the example, the win_copy module is used to copy files.

Ansible is designed to be idempotent, which means it does not make unnecessary changes if the system is already in the desired state (Red Hat, 2025). This feature distinguishes Ansible from many other automation tools and makes it a particularly reliable and efficient solution for infrastructure management. Idempotency means that the same command can be executed multiple times without changing the system unnecessarily if it is already in the correct state.

When this playbook is executed for the first time, Ansible creates the directory. If the playbook is executed again, Ansible first checks if the directory already exists, and if it does, nothing is changed. The execution has three possible statuses: **ok**, **changed**, and **failed**. **Ok** means the system was already in the desired state, and no changes were made. **Changed** indicates that Ansible made a change, and **Failed** means that something went wrong.

Another essential feature of Ansible is the inventory, which defines the devices managed by Ansible. It can include IP addresses, hostnames, group classifications, and other settings, such as login credentials and ports.

```
[windows]
192.168.1.100
    ansible_user=user
    ansible_password=password
    ansible_connection=winrm
    ansible_winrm_transport=basic
    ansible_port=5985
```

Figure 3. Ansible Inventory Example

In the inventory example, roles are as follows:

- **[windows]** is the group name under which machines belonging to this group are defined.
- **192.168.1.100** is the IP address of the managed target machine.
- **ansible_user** is the username used by Ansible to log in.
- **ansible_password** is the password for the username.
- **ansible_connection** specifies the method used to establish the connection; in this case, WinRM, because the target machine is Windows.
- **ansible_winrm_transport** specifies which authentication method is used to establish a WinRM connection.
- **ansible_port** specifies the port Ansible uses to connect to the target machine.

3.2 Semaphore UI

Semaphore UI is an open-source user interface designed to simplify the management of DevOps tools such as Ansible, Terraform, and OpenTofu (Semaphore UI, 2025a). It allows users to execute Ansible playbooks, Terraform configurations, and Bash or PowerShell scripts through a web-based interface without needing to use the command line.

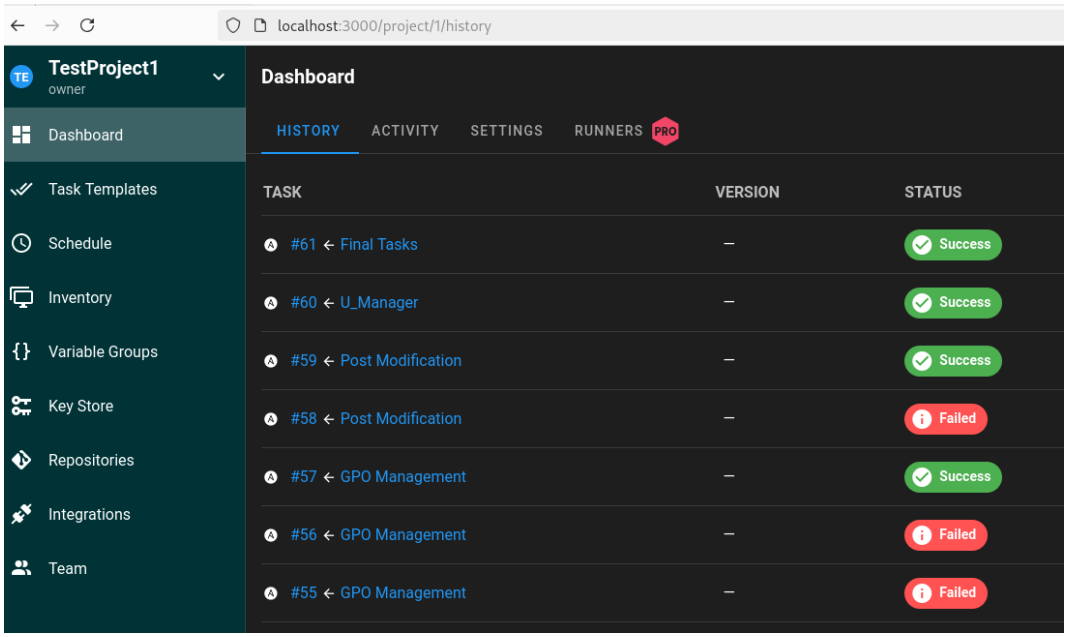

Figure 4. Semaphore UI Dashboard

One of Semaphore UI's greatest advantages is its ease of use compared to other CI/CD solutions like Jenkins or GitLab CI/CD. Unlike these tools, Semaphore UI is primarily designed for infrastructure automation and does not require complex configurations. This makes it a lighter and faster-to-deploy alternative (Semaphore UI, 2025a)

Semaphore UI operates as a server-based system with a REST API, enabling users to execute commands and manage various automation processes. While the interface runs in a web browser, all background operations are based on API calls (Semaphore UI, 2025b). The architecture of Semaphore UI consists of the following components:

- **Frontend:** A user interface that allows users to launch tasks, review executed processes and monitor the server's status.
- **Backend:** An API-based system that processes commands received from the frontend. It also integrates with tools such as Ansible, Terraform, and others.
- **Database:** Semaphore UI supports PostgreSQL, MySQL, and DoltDB databases.

Through API calls, users can add and manage users, initiate and monitor playbook executions, view task logs, configure new target systems, and define connection details. Semaphore's API library provides a wide range of functionalities and enables integrations with other systems without requiring manual use of the UI (Semaphore UI, 2025b).

3.3 MDT

Microsoft Deployment Toolkit (MDT) is a deployment tool developed by Microsoft that enables automated provisioning of Windows operating systems and applications (Microsoft, 2025a). It allows the controlled and efficient deployment of workstations and servers, reducing the need for manual work and speeding up the process. MDT supports installations in both physical and virtual environments and can be used independently or in conjunction with other Microsoft management tools (Wikipedia, n.d.).

MDT utilizes so-called task sequences, which systematically define and execute the various steps of the deployment process. These can include disk partitioning, operating system installation, adding drivers and applications, and final system configuration (Microsoft, 2025a).

With MDT, the deployment process can be automated so that no manual intervention is required during the installation. This is particularly useful in large environments where multiple servers are deployed simultaneously.

MDT consists of several key components, listed below, that enable the automated deployment of operating systems and applications.

- **Deployment Workbench** serves as MDT's graphical interface, where configurations are managed and the deployment process is controlled.
- **Task Sequences** define the various steps of the deployment process and automate tasks, enabling systematic and managed installations.
- **Boot Images** include Windows PE boot images, which allow computers to boot and begin the operating system installation without requiring physical installation media.
- **Application Deployment** allows applications to be installed alongside the operating system installation, enabling automatic inclusion of company-required software without manual effort.
- **Driver Management** ensures the automatic handling of drivers, adding the necessary hardware-specific drivers during installation. This simplifies system deployment across different hardware configurations (Microsoft, 2025b).

3.4 WinRM

Windows Remote Management (WinRM) is a protocol developed by Microsoft that enables the remote management and configuration of Windows-based systems (Microsoft, 2025b). It is based on the WS-Management protocol, which adheres to the Web Services for Management standard. WinRM allows commands to be executed remotely, enabling system administrators to manage servers without physical access to the target systems. This makes it a crucial technology in automated system management, especially in environments that utilize PowerShell-based administration and configuration (Ansible Documentation, 2025).

WinRM facilitates the sending and receiving of management commands, which can be used for tasks such as running PowerShell scripts, modifying system settings, and performing remote administration. The protocol allows for system configuration, application installation, and checking the system's status without requiring local login to the managed server. This makes it an effective tool for managing large Windows environments where automated configuration and remote management are essential (Microsoft, 2025b).

WinRM supports multiple authentication methods, such as Kerberos, NTLM, and Basic Authentication, ensuring secure system access and control (Ansible Documentation, 2025). Communication can be conducted over HTTP or HTTPS, with enhanced security provided by Transport Layer Security (TLS) encryption to prevent unauthorized data interception. This makes WinRM a secure method for remotely managing Windows environments, particularly when connections traverse public networks (Microsoft. 2025b).

4 PROJECT IMPLEMENTATION

The implementation of the project was divided into five main phases: Initial Setup, Tool Selection, Environment Preparation, Playbook Configuration, and Testing. Each phase represents a key area of the project and ensures its systematic progression without significant setbacks. Figure *5* illustrates the content and main tasks of each phase, which supported the achievement of the project goals.

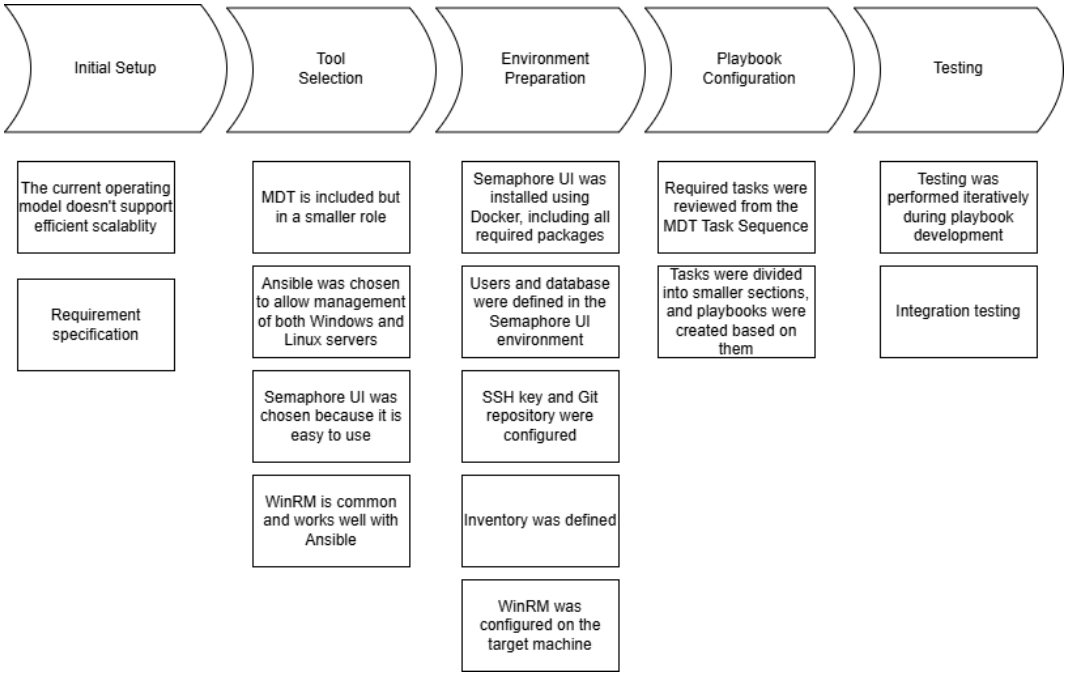

Figure 5. Project Stages

4.1 Environment Setup and Requirements

The implementation of the project began with defining the environment requirements and preparing the necessary technical resources. The current operating model had several challenges, such as the need for physical presence, time-consuming manual setup processes, and complex maintenance procedures. To address these issues, an automated system was developed to enable more efficient deployment and remote management.

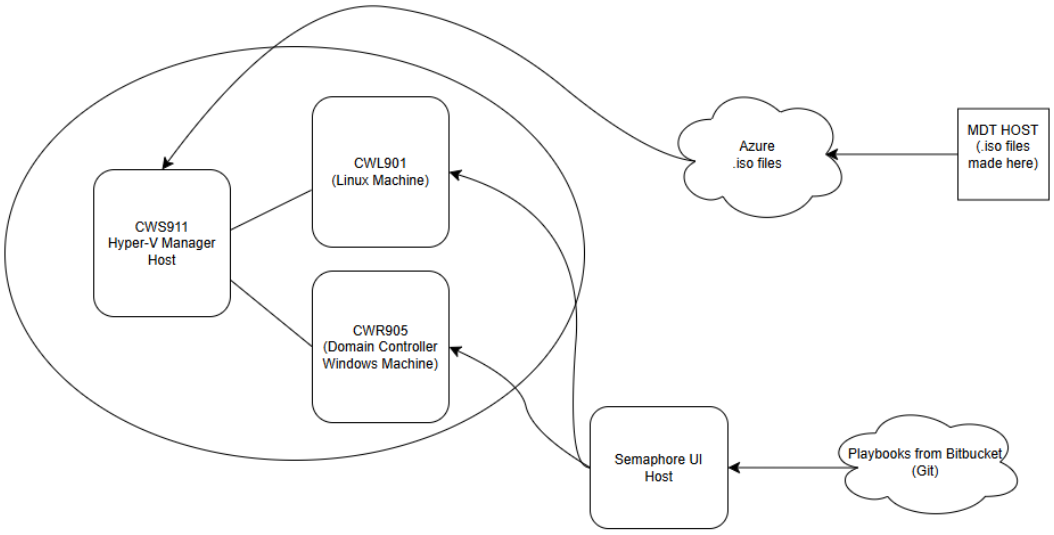

Figure 6. Simple figure of the environment

Figure 6 illustrates a simplified structure of the environment. In a real environment, there could be additional virtual machines, but this project focuses on the following four components: CWS911, CWL901, CWR905, and Semaphore UI host.

- CWS911 acts as the platform where all virtual machines are located and managed. This virtual machine is not part of the project's modifications but serves as the foundation for the environment.
- CWR905 is a Windows-based Domain Controller responsible for server management. Its configuration is automated using Semaphore UI and Ansible playbooks.
- CWL901 is a Linux-based (Debian) virtual machine. The configuration of CWL901 is also managed using Semaphore UI and playbooks.
- Semaphore UI Host is a Linux-based (Debian) virtual machine. This environment is the foundation of the project. It provides a graphical interface for managing and executing playbooks.
- MDT Host was used to create the entire virtual machine. Now, it is responsible only for creating the base environment, which is copied to Azure and used for initializing virtual machines.

- Bitbucket (Git) serves as the version control system where playbooks and other configuration files are stored. This enables collaboration within the team and ensures that all changes are documented and easily recoverable.

4.2 Deployment of Semaphore UI

The deployment process began with setting up the Semaphore UI host machine. Semaphore UI was chosen for its ability to manage Ansible playbooks through a graphical interface, simplifying server management tasks. The installation of Semaphore UI was performed using Docker Compose, ensuring a consistent and straightforward deployment process.

The following steps were taken to set up Semaphore UI:

1. **Preparing the Host Environment**

   The host machine, running a Debian-based operating system, was updated, and Docker Compose was installed to enable containerized deployment. This ensured that Semaphore and its dependencies could be managed easily. The installation was completed using the command:

   ```
   sudo apt install -y docker-compose
   ```

   Figure 7. Installing Docker Compose

2. **Creating the Docker Compose Configuration**

   A docker-compose.yml file was created to define the configuration for Semaphore and its required database (MySQL). The playbook for this installation could be found on the semaphore UI installation site (Semaphore Installation, 2025). The file includes details such as environment variables, database credentials, and exposed ports.

```yaml
version: '3.7'
services:
  mysql:
    image: mysql:8.0
    restart: unless-stopped
    hostname: mysql
    volumes:
      - semaphore-mysql:/var/lib/mysql
    environment:
      MYSQL_RANDOM_ROOT_PASSWORD: 'yes'
      MYSQL_DATABASE: semaphore
      MYSQL_USER: semaphore
      MYSQL_PASSWORD: semaphore_password
  semaphore:
    image: semaphoreui/semaphore:latest
    restart: unless-stopped
    ports:
      - "3000:3000"
    environment:
      SEMAPHORE_DB_USER: semaphore
      SEMAPHORE_DB_PASS: semaphore_password
      SEMAPHORE_DB_HOST: mysql
      SEMAPHORE_DB_PORT: 3306
      SEMAPHORE_DB_DIALECT: mysql
      SEMAPHORE_DB: semaphore
      SEMAPHORE_ADMIN_NAME: admin
      SEMAPHORE_ADMIN_EMAIL: admin@localhost
      SEMAPHORE_ADMIN_PASSWORD: changeme
    depends_on:
      - mysql
volumes:
  semaphore-mysql:
```

Figure 8. docker-compose.yml file

3. Launching the Semaphore Environment

Once the docker-compose.yml file was in place, Docker Compose was used to launch both Semaphore and MySQL services. This was achieved by navigating to the directory where the docker-compose.yml file was stored and running the following command:

```
sudo docker-compose up -d
```

Figure 9. Command for launching the UI and database services

This command started the services in the background, exposing Semaphore UI on port 3000 for access through a web browser.

4. Accessing and Configuring Semaphore

After launching the services, Semaphore UI was accessed via a browser at “http://host-machine-ip:3000” or “http://localhost:3000. During the initial setup, an admin account was created using the credentials specified in the docker-compose.yml file. So, the credentials for logging in here are: “admin@localhost” and “changeme”. The database was also configured at this stage to store project data, playbooks, and inventory details.

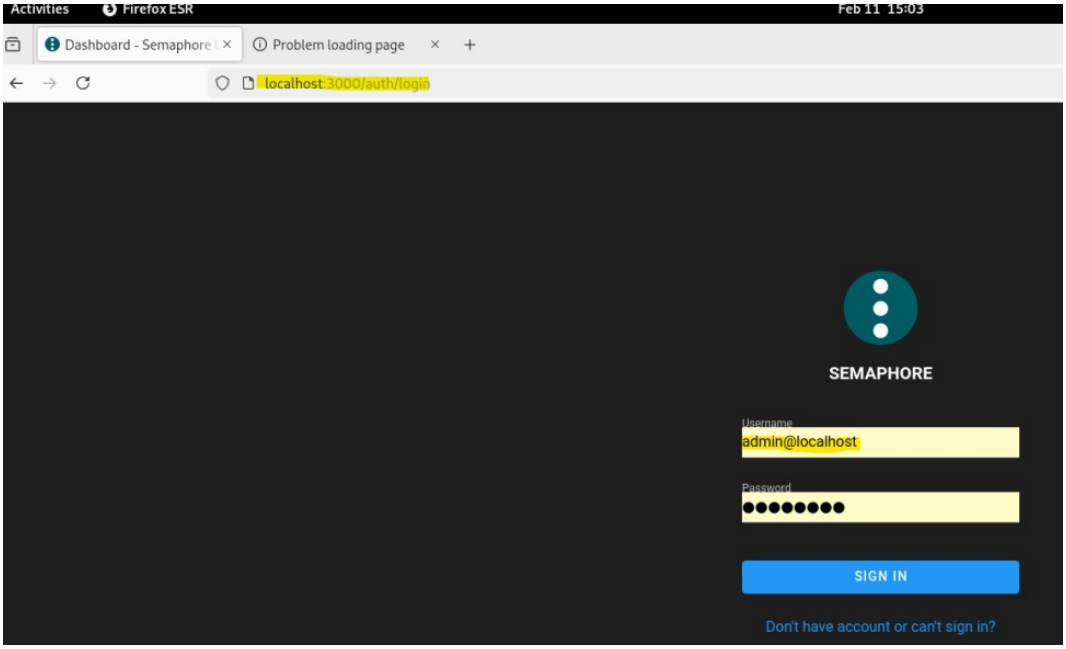

Figure 10. Semaphore Login page

4.3 WinRM Connectivity

To enable Ansible to communicate with the CWR905 target machine, WinRM (Windows Remote Management) was configured on the target machine. This setup involved enabling the WinRM service, modifying connection settings and validating the configuration through Ansible in Figure 23.

The WinRM service was first enabled on the Windows target machine using Powershell. The command in figure 11 ensured that the service was active and listening. This command automatically created a listener and adjusted the necessary settings. The "-force" flag bypassed confirmation prompts to streamline the process.

```
winrm quickconfig -force
```

Figure 11. Enabling the WinRM service

To support Ansible's WinRM connection requirements, the following configurations were applied:

- Basic Authentication enabled to allow Ansible to authenticate with the target machine.
- Unencrypted Traffic is permitted for testing purposes, as HTTPS was not configured.

```
Set-Item -Path WSMan:\localhost\Service\Auth\Basic -Value $true
```

Figure 12. Enable Basic Authentication

```
Set-Item -Path WSMan:\localhost\Service\AllowUnencrypted -Value $true
```

Figure 13. Enable Unencrypted traffic

To ensure that the target machine could accept incoming requests on port 5985, a new firewall rule was added as shown in Figure 14. This rule allowed HTTP traffic for WinRM across all network profiles: public, private and domain.

```
New-NetFirewallRule -Name "WinRM HTTP" -DisplayName "WinRM HTTP" -Protocol TCP -LocalPort 5985 -Action Allow -Profile Any
```

Figure 14. Configuring firewall

4.4 Configuring Semaphore UI

After deploying Semaphore UI, the next phase involved configuring the environment to enable effective management of automation tasks. This configuration focused on defining the inventory of target machines, integrating SSH keys for secure communication, and linking a Git repository for managing playbooks and configuration files. These steps were essential to prepare Semaphore UI for its role as the central automation tool.

To enable secure repository access, an SSH key was added to Semaphore UI. The process included: Generating an SSH key pair on the host machine, adding the private key to Semaphore UI under the Access Key section and copying the public key to the authorized SSH keys of the remote Git repository under the Access Management SSH Keys section.

```
ssh-keygen -t rsa -b 2048 -f ~/.ssh/semaphore_id_rsa
```

Figure 15. Command to make the SSH key pair

As soon as the key pair is made, we can copy the public to the target machine. This enables us later to run playbooks on the CWL901.

```
ssh-copy-id -i ~/.ssh/semaphore_repo_rsa.pub HJA@192.168.10.101
```

Figure 16. Copying the public to the CWL901

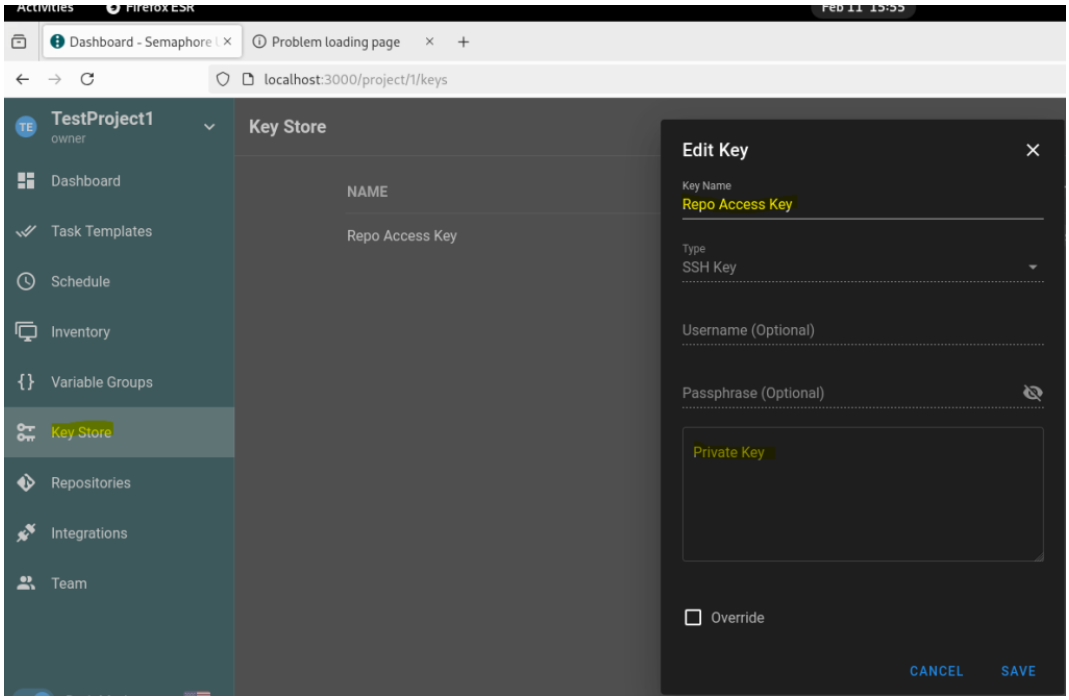

Figure 17. Figure of adding the private SSH key

Under the Key Store tab, it is required to add a key name and to copy the private SSH key to the text box.

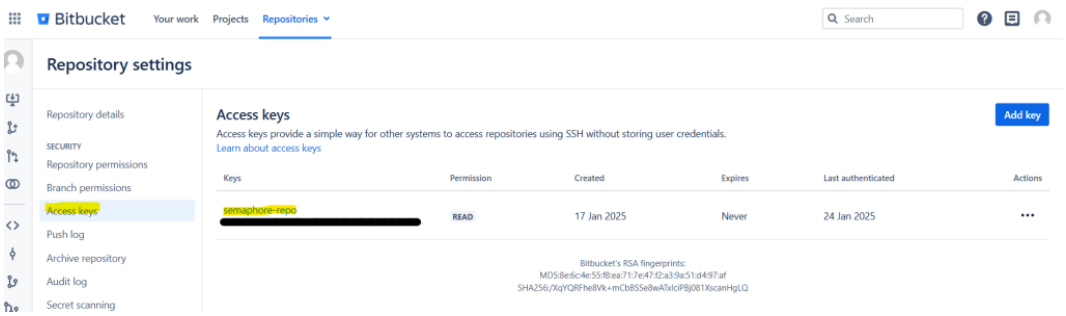

Figure 18. Linking the public SSH key to the Bitbucket repository

For Linux servers such as CWL901, SSH was used as the communication protocol. The configuration steps are like those of Windows. The Windows inventory was configured separately to include Windows servers, such as CWR905. These servers use WinRM for communication. The inventory specified: IP address or hostname, WinRM usernames and passwords and connection details for WinRM.

```
[windows]
192.168.10.100
ansible_user=HJA
ansible_password=pswd
ansible_connection=winrm
ansible_winrm_transport=basic
ansible_port=5985
```

Figure 19. Inventory for CWR905

```
[linux]
192.168.10.101
ansible_user=wois
ansible_become_password=pswd
ansible_ssh_private_key_file=/home/wois/.ssh/semaphore_repo_rsa
```

Figure 20. Inventory for CWL901

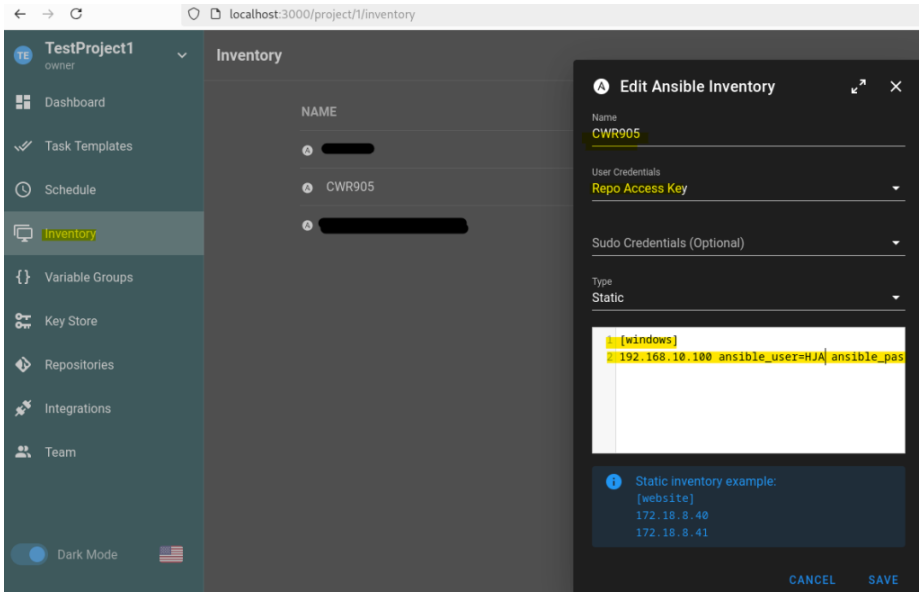

Figure 21: Adding the CWR905 Inventory

To centralize playbooks and configuration files, a Git repository was linked to Semaphore UI under the Repositories section. The required information is the branch where the playbooks are located and the Access key. The same access key is used in this example. The repository link can be obtained from the Git.

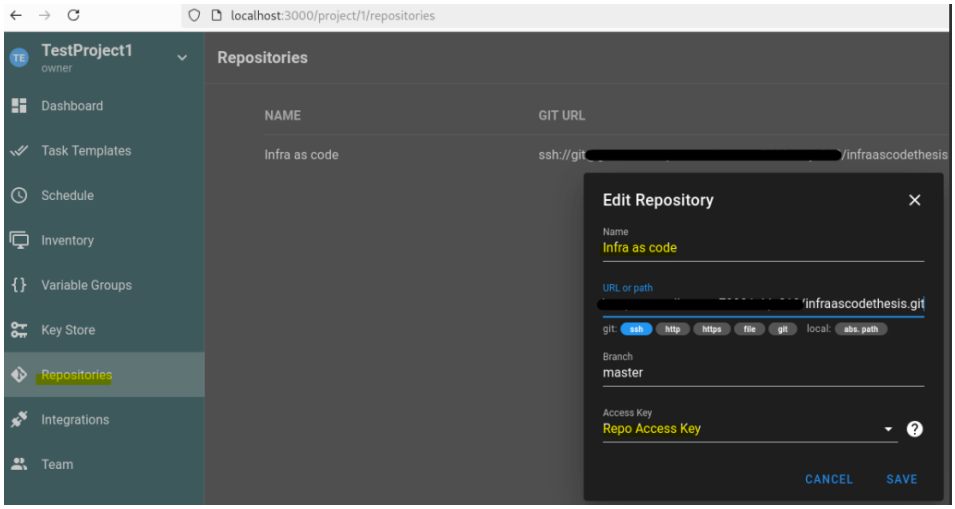

Figure 22. Adding the git repository

4.5 Playbook Development and Testing

The playbook development process focused on creating modular, efficient, and reusable Ansible playbooks. These playbooks were primarily designed to automate system configurations and script execution on both Windows and Linux machines. The playbooks were developed iteratively to ensure accuracy and reliability.

To manage the complexity of automation tasks, the playbooks were divided into smaller, logical chunks. Each playbook focused on a specific task or group of tasks, such as system validation, script execution, or software installation. This modular approach provided the following advantages: Easier debugging and testing, reusability across different machines and environments, reduced risk of errors by isolating tasks in separate files. For example, in this environment there's a validation playbook that checks the system compatibility such as memory and processor speed requirements. This can be used multiple times on different machines.

To test connectivity and basic functionality, the following simple playbook was created and tested in Figure 23 on the CWR905 Windows-virtual machine.

```
- name: Test Windows Playbook
  hosts: windows
  tasks:
    - name: Check connectivity
      win_ping:
    - name: Create a test directory on the C drive
      win_file:
        path: C:\TestFolder
        state: directory
    - name: Create a test file in the folder
      win_shell: echo "Hello, World!" > C:\TestFolder\hello.txt
    - name: Display contents of the file
      win_shell: type C:\TestFolder\hello.txt
      register: file_contents
    - name: Print file contents
      debug:
        msg: "{{ file_contents.stdout }}"
```

Figure 23. Test playbook

The playbook in Figure 23 verifies that WinRM connections are configured correctly and that the basic commands could be executed remotely on the target machine. The most important part about the playbooks was making sure they are correctly spaced. For that use online yaml-file validators. They check that the yaml-files are correctly written (YAML Checker, 2025).

After adding the test playbook into the repository, we must make a new task template in Semaphore UI. It requires a name for the template, the path in the repository to the playbook and its name, the inventory we created earlier and the repository as well.

After creating the task template, we can run the task. Semaphore UI will display what is happening during the run step-by-step. In Figure 25 it shows a successful “Test playbook Windows” run. On the bottom there is a recap which will indicate what it did during the run.

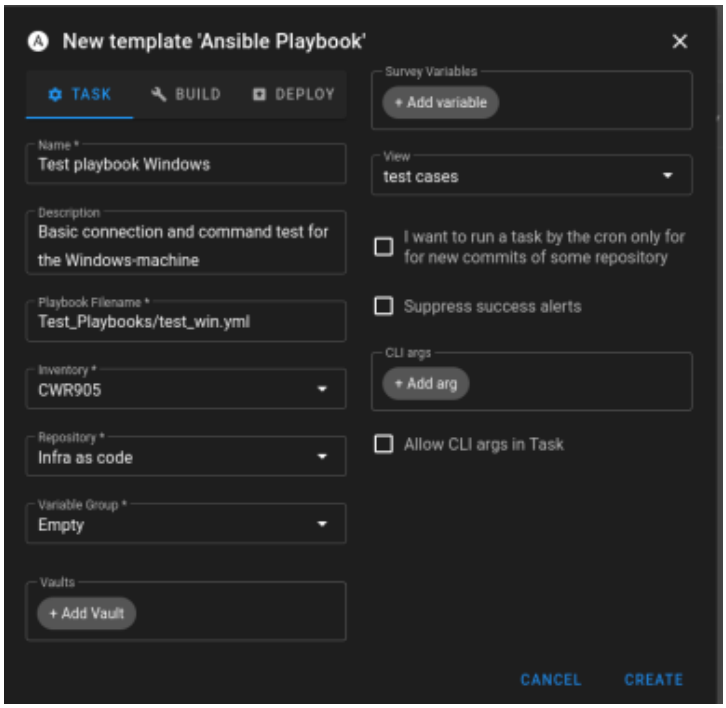

Figure 24. Making a task template for the test playbook

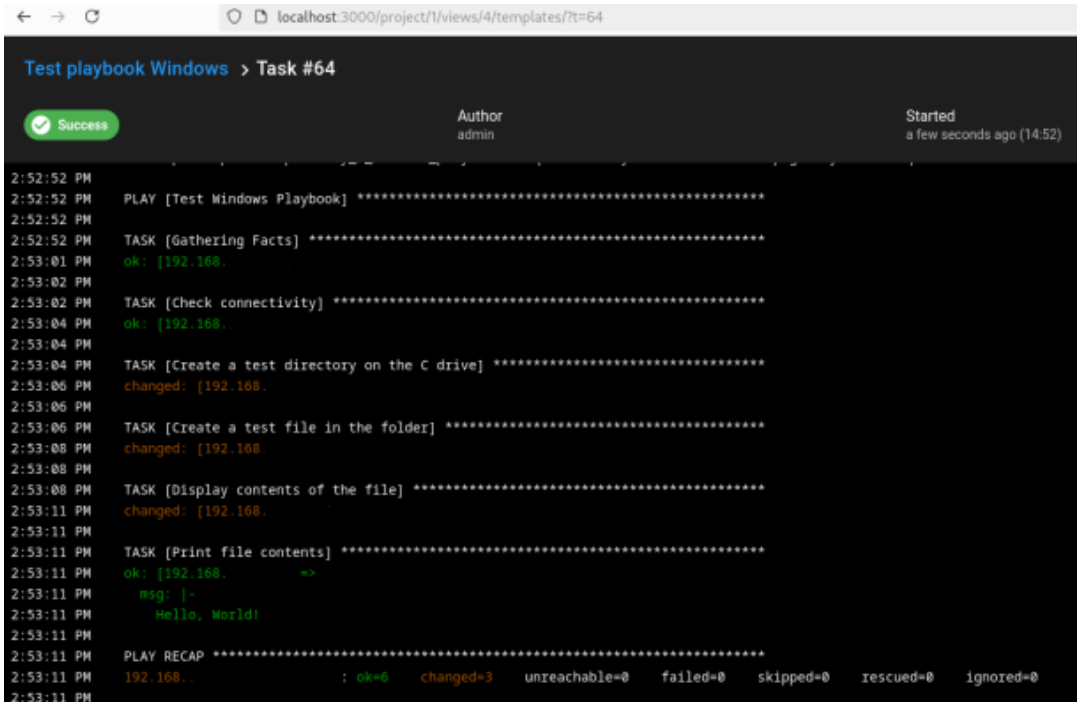

Figure 25. Running the test playbook

Semaphore UI provides an intuitive way to manage tasks and track playbook executions. Task runs are displayed sequentially, and views can be created to organize tasks by computer or function. For example, a "Test Cases" view was created to group all test-related tasks. Under it the test playbook was run twice. The playbook path was wrong the first time which caused the failed run, but after resolving the issue, the second run was successful as shown in Figure 26.

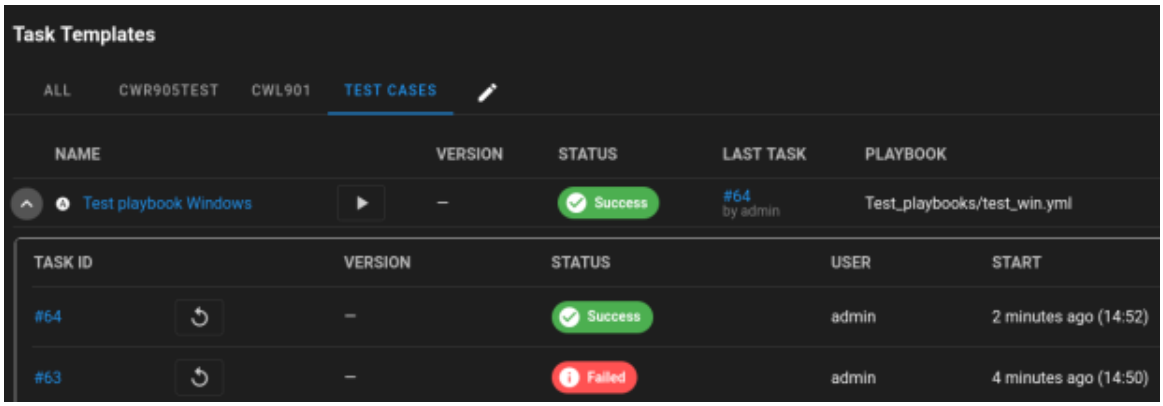

Figure 26. Semaphore Test Cases view

The full list of the playbooks that are going to be deployed to the CWR905 are shown below in Figure 27. The files are in a bitbucket repository that is linked to the Semaphore UI. These playbooks cover key automation tasks, such as system validation, application setup and configuration management, ensuring the CWR905 virtual machine is fully prepared for production. The playbooks for CWR905 are in the CWR905playbooks/ directory, while the playbooks for CWL901 are stored in the CWL901playbooks/ directory.

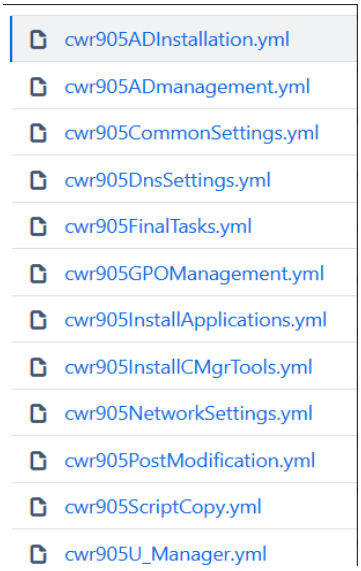

Figure 27. CWR905 Playbooks

The playbooks developed for the CWR905 machine utilized both Ansible modules and raw PowerShell commands to achieve automation. Tasks requiring a system reboot, such as Active Directory installation, leveraged Ansible's win_reboot module. This module ensured that reboots were handled cleanly, allowing playbooks to pause and resume automatically after the system came back online. An example of its usage is shown below in Figure 28.

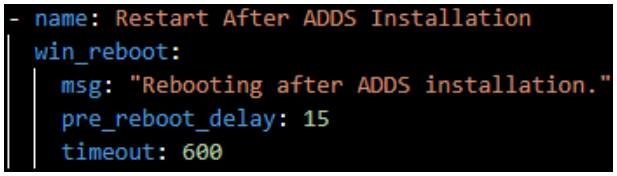

Figure 28. Example of the win_reboot module

For other configurations, the win_shell module was used to execute PowerShell scripts for tasks not supported by Ansible's built-in modules.

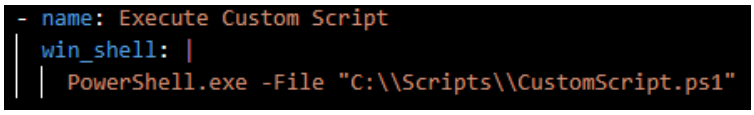

Figure 29. Example of win_shell module usage

5 RESULTS AND ANALYSIS

5.1 Success of Deployment

The deployment process was successfully completed, achieving the primary goal of automating the configuration of the CWR905 and CWL901 machines. The playbooks developed for this project, executed through Semaphore UI, performed all tasks accurately and consistently, reducing the need for manual intervention.

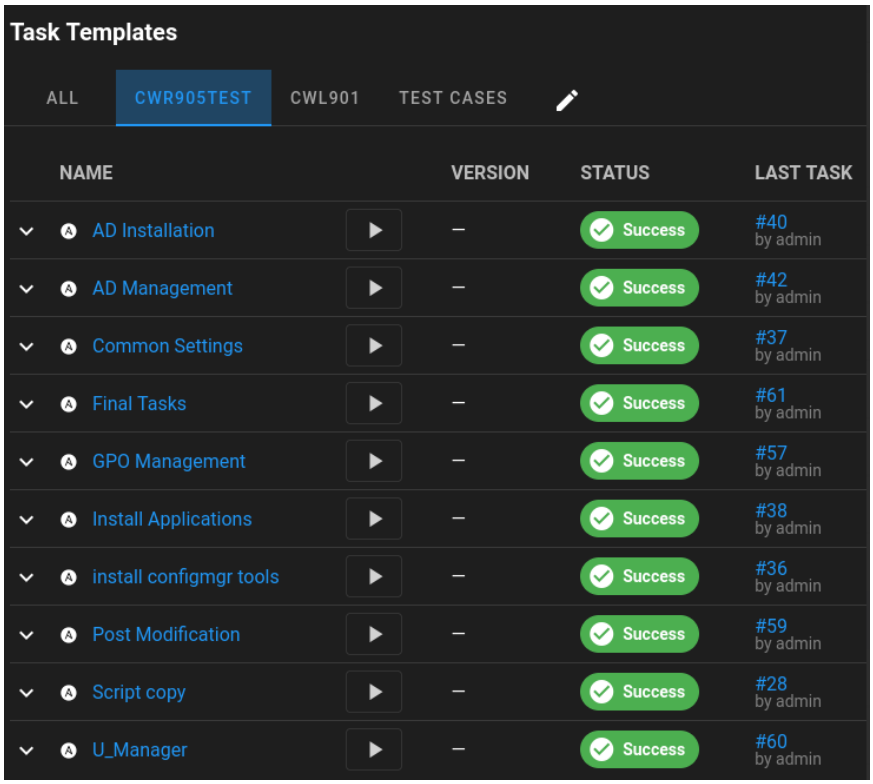

Figure 30. All playbooks run successfully for CWR905

Figure 30 shows the successful execution of all playbooks for the CWR905 machine in Semaphore UI. The logs confirm that each task was completed as expected. The playbooks were rigorously tested to ensure that each task was executed as intended with logs confirming successful completion.

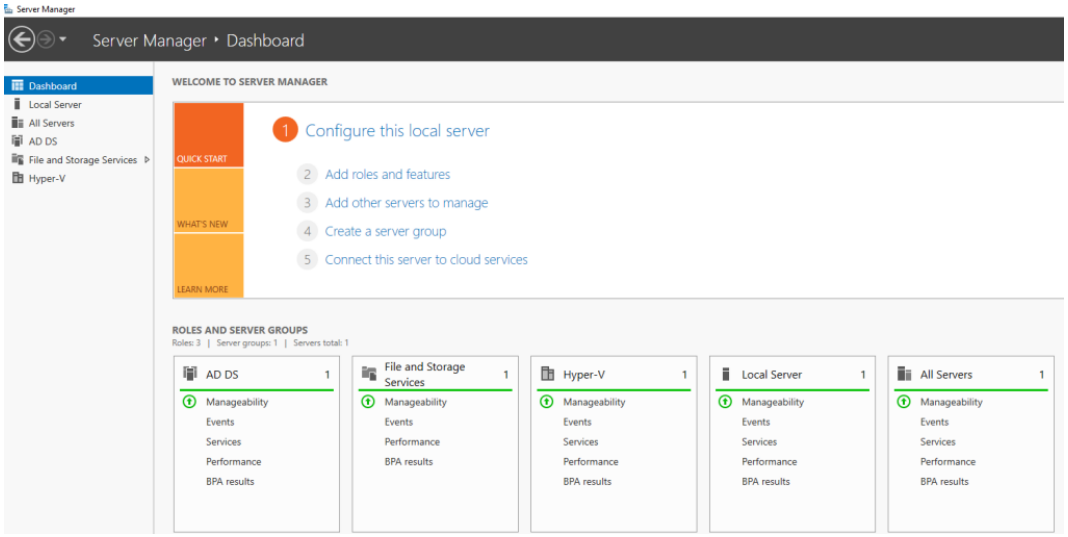

Figure 31. CWR905 Server Manager changes

As part of the deployment process, the CWR905 machine was configured to include roles such as Active Directory Domain Services (AD DS), File and Storage Services, and Hyper-V, as shown in Figure 31. These roles were installed and configured through the developed playbooks, which ensured consistency and accuracy without requiring manual intervention. In addition to the roles shown in Figure 30, critical tasks such as configuring group policies, setting up user accounts and ensuring network configurations were automated through playbooks. These steps further validated the robustness of the deployment process.

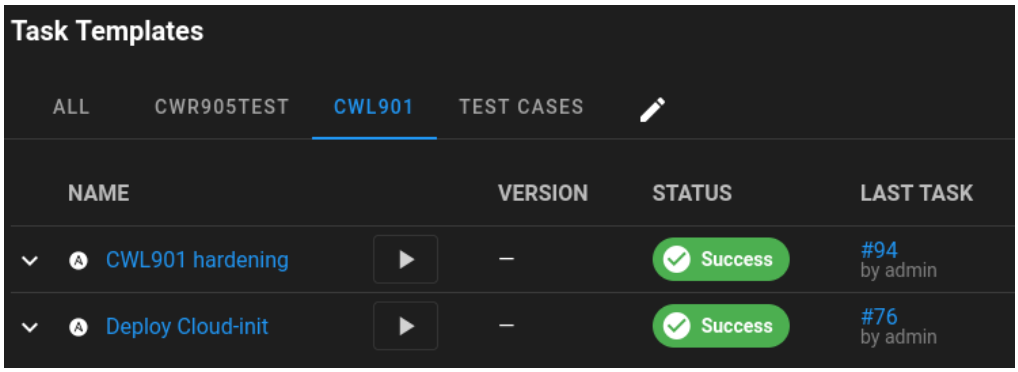

Figure 32. All playbooks run successfully for CWL901

Figure 32 shows that Semaphore UI successfully executed both playbooks. The Deploy Cloud-Init playbook made the user on CWL901 a sudoer and retrieved the user-data.yml file from Bitbucket and applied it to the target machine. The Hardening playbooks added security features such as audit rules and system configurations.

```
wois@cwl901:~$ sudo auditctl -l
-w /etc/passwd -p wa -k identity
-w /etc/shadow -p wa -k identity
-w /etc/group -p wa -k identity
-w /etc/gshadow -p wa -k identity
-w /var/log/auth.log -p wa -k auth
-w /var/log/syslog -p wa -k syslog
-w /var/log/wtmp -p wa -k logins
-w /var/log/btmp -p wa -k logins
-a always,exit -F arch=b64 -S execve -F key=exec_log
-a always,exit -F arch=b32 -S execve -F key=exec_log
-w /etc/sudoers -p wa -k sudo
-w /etc/sudoers.d -p wa -k sudo
-w /sbin/insmod -p x -k module-load
-w /sbin/rmmod -p x -k module-load
-w /sbin/modprobe -p x -k module-load
-a always,exit -F arch=b64 -S init_module,delete_module -F key=module-load
-a always,exit -F arch=b32 -S init_module,delete_module -F key=module-load
-w /etc/security/opasswd -p wa -k opasswd
-w /var/log/faillog -p wa -k faillog
-w /var/log/tallylog -p wa -k logins
-w /var/run/utmp -p wa -k logins
```

Figure 33. Successful hardening on CWL901

In Figure 33 we use the command “sudo auditctl -l” to check the rules that we added for the hardening. The playbook managed to add the security features needed to the target machine.

5.2 Impact of Automation on Efficiency

The automation process significantly improved the efficiency of system configurations by streamlining and accelerating tasks that were previously manual. Playbooks ensured consistency and minimized variability, resulting in faster deployments and more reliable configurations.

Tasks were completed in under 30 minutes per machine using automation. This marked a significant improvement over the one to two hours typically required for manual configurations. By automating tasks, multiple machines could be configured in parallel, further reducing total deployment time.

Automation ensured that all configurations followed predefined workflows, eliminating inconsistencies caused by human error. The modular structure of playbooks ensured repeatability, allowing identical configurations to be applied to different machines without requiring additional adjustments.

By automating critical tasks, such as testing connectivity, applying security configurations, and managing reboots, the automation framework minimizes manual errors. Logs generated by Semaphore UI provided real-time feedback, allowing immediate detection and resolution of issues during playbook execution.

5.3 Challenges and Issues in Developing Solutions

The implementation of automation using Ansible, Semaphore UI, and WinRM introduced several technical challenges that required extensive troubleshooting and iterative refinements. These challenges ranged from connectivity issues and configuration inconsistencies to software installation constraints and YAML syntax errors.

One of the most persistent challenges encountered during the deployment process was network connectivity between the Semaphore host and the target Windows machine (CWR905). Multiple instances of connection failures were observed, often caused by incorrect routing table configurations and firewall restrictions. After machine reboots, static routes would frequently be lost, requiring reconfiguration. Additionally, Windows Defender Firewall and UFW on Debian occasionally blocked communication over WinRM ports 5985 and 5986, necessitating repeated rule adjustments to ensure uninterrupted connections.

Another major challenge was configuring WinRM to allow Ansible to communicate with the target system reliably. While WinRM is a fundamental requirement for managing Windows hosts with Ansible, several issues arose throughout the implementation. The listener configurations would often reset after reboots, requiring re-execution of the winrm quickconfig command to restore connectivity. Furthermore, enabling Basic Authentication and unencrypted traffic was necessary for the connection to function correctly. Connection timeouts were another recurring issue, requiring fine-tuning of Ansible's default timeout settings to accommodate the WinRM response delays.

In addition to connectivity and configuration challenges, YAML formatting errors caused frequent disruptions in playbook execution. YAML's strict indentation rules led to syntax errors that prevented certain tasks from being executed correctly. Furthermore, misconfigured inventory definitions occasionally caused playbooks to fail due to incorrect target group mappings. The task execution order also posed a challenge, as some configurations relied on dependencies that were not yet installed. Without proper sequencing, some playbooks executed tasks prematurely, leading to failures that required manual intervention.

Software installation constraints also played a significant role in shaping the automation process. The target machines lacked direct internet access, which meant that all necessary software, scripts, and configuration files had to be pre-downloaded and staged in external repositories, such as Azure Blob Storage, Bitbucket, or MDT. This limitation was one of the main reasons why MDT was retained in the automation workflow instead of being fully replaced. Preloading dependencies and ensuring that installation scripts were executed in the correct sequence was essential to avoid incomplete or failed deployments.

Another issue encountered during implementation was the intermittent failure of the Semaphore UI service. When the host machine entered the sleep mode or was powered off, Semaphore UI sometimes appeared to be running but was unresponsive. In these cases, verifying its actual status using "sudo docker ps" confirmed

that the service was inactive despite showing as active, which required restarting the service. First, it should be shut down with: “sudo docker-compose down” and after it is shut down, it must be started again with: “sudo docker-compose up -d”. Once restarted, logging back into the UI via port 3000 restored normal operation.

Throughout the development process, these challenges required continuous troubleshooting, iterative playbook refinements, and improved system documentation. Network reliability, secure WinRM configurations, YAML validation, and software installation sequencing were all key focus areas that needed repeated testing and adjustments. Despite these challenges, automation ultimately provided a scalable, repeatable, and efficient deployment framework, significantly reducing manual intervention while maintaining system consistency.

6 CONCLUSIONS AND FUTURE DEVELOPMENT

6.1 Key Findings

The automation framework developed in this project successfully streamlined the configuration of both Windows and Linux-based machines, reducing deployment time and ensuring repeatability. The integration of Ansible playbooks with Semaphore UI provided a structured and scalable solution for managing deployments across different operating systems. The results demonstrated that manual configuration processes can be largely replaced by automation, significantly improving efficiency, consistency, and long-term maintainability.

A key finding was that a modular approach to automation significantly improves adaptability across different system types. By structuring playbooks into distinct tasks, configurations could be applied consistently to both Windows (CWR905) and Linux (CWL901) machines. This approach not only minimized human error but also improved the adaptability of playbooks, making them easier to troubleshoot and reuse in different environments.

The project also highlighted the differences in automating Windows vs. Linux environments. WinRM posed a significant challenge for Windows automation, requiring additional configuration steps such as enabling Basic Authentication and adjusting firewall rules. In contrast, Linux-based automation via SSH was more straightforward, requiring fewer connectivity adjustments. This comparison reinforced the importance of understanding the nuances of each platform when designing automation workflows.

Another key takeaway was that MDT remains necessary for initial system provisioning in Windows environments. While Ansible and Semaphore UI effectively automated post-installation tasks, MDT played a critical role in preparing Windows machines before Ansible could take over. In contrast, Linux deployments benefited from a more direct Ansible approach, as SSH-based automation allowed for end-to-end configuration without additional provisioning tools.

6.2 System Expansion Possibilities

The automation framework developed in this project can be expanded in multiple ways to further improve system management, scalability, and efficiency. While the current implementation covers the configuration of Windows (CWR905) and Linux (CWL901) machines, several opportunities exist to enhance automation coverage and performance.

One key area for expansion is extending automation to additional machines and roles. The current setup is designed for configuring a limited number of servers, but the same framework could be applied to other infrastructure components. The sWOIS environment consists of multiple interconnected machines, yet automation has so far only been implemented for the domain controller (CWR905) and a logging machine (CWL901). Expanding automation to application servers and backup systems would enhance deployment consistency, ensuring a more unified and scalable infrastructure management approach.

Another potential improvement involves enhancing the complexity of the playbooks to include additional automation tasks. The current playbooks primarily focus on system configuration and role installation but expanding them to automate software deployment and patch management would ensure that all systems remain updated without manual intervention. Additionally, integrating advanced security configurations, such as compliance hardening and access control policies,

would strengthen the security of the infrastructure. Real-time monitoring and logging could also be incorporated to provide continuous visibility into system health and performance, reducing the need for manual diagnostics.

To further improve the security of automation, integrating additional tools and technologies should be considered. One key enhancement would be enabling HTTPS for WinRM instead of HTTP, which would increase security while maintaining remote management capabilities. This adjustment would mitigate potential vulnerabilities associated with unencrypted communication while ensuring that automation remains effective.

As the automation framework scales, performance considerations must also be addressed. Automating a larger number of machines simultaneously may introduce challenges related to execution time delays, increased resource allocation in Semaphore UI when running multiple concurrent tasks, and higher network traffic due to parallel playbook executions. Optimizing Semaphore UI to handle a higher volume of simultaneous executions or implementing a dedicated Ansible control node would significantly enhance both performance and scalability. Ensuring that automation workflows remain efficient under heavier loads will be crucial as the framework expands.

6.3 Recommendations for Future Development

Although the automation framework developed in this project successfully streamlined system configuration and deployment, several areas can be improved to enhance security, efficiency, and scalability. Future development should focus on strengthening security measures and expanding automation to cover the entire sWOIS environment, ensuring that all necessary components are included in the deployment process.

Beyond connection security, access control within Semaphore UI also requires improvement. At present, the Semaphore UI environment has only a single Admin

user, which presents a security risk if credentials are compromised. Implementing role-based access control would allow for more granular permission management, restricting automation execution to specific users or teams based on their access level. Further security hardening, such as firewall rule adjustments and automated auditing of executed tasks, could also enhance the resilience of the automation framework.

Lastly, future development should include comprehensive documentation and training to ensure that the automation framework can be easily maintained and expanded. While the current implementation provides a strong foundation, additional documentation detailing troubleshooting steps, configuration best practices, and security guidelines would improve long-term usability. Providing training sessions for system administrators would also help ensure the efficient adoption of automation workflows across different teams.

7 SUMMARY

This thesis explored the implementation of an automation framework using Ansible, Semaphore UI, and WinRM to streamline system deployment. The automation process significantly reduced configuration time, ensured consistency, and minimized manual effort across both Windows and Linux environments.

The project demonstrated that modular automation improves flexibility and reliability. Ansible playbooks provided reusable configurations, while Semaphore UI enabled centralized management of automation tasks. Although Linux automation via SSH was straightforward, Windows automation required additional configurations, particularly for WinRM authentication and firewall adjustments. Despite these challenges, automation was a success and reduced human intervention.

Security remains an area for improvement. Future iterations should implement HTTPS-based WinRM and role-based access control in Semaphore UI to enhance security. Additionally, expanding automation to application servers and other infrastructure components would further improve efficiency.

Overall, this project confirmed that automation enhances IT infrastructure management by reducing errors, improving scalability, and ensuring repeatability. With continued improvements, the framework can evolve into a fully secure and comprehensive automation solution.

REFERENCES

Ansible Documentation. "Using WinRM with Ansible." Retrieved February 14, 2025, from

https://docs.ansible.com/ansible/latest/os_guide/windows_winrm.html

Microsoft. (2025a) Use the Microsoft Deployment Toolkit. Retrieved February 14, 2025, from https://learn.microsoft.com/en-us/mem/configmgr/mdt/use-the-mdt

Microsoft. (2025b) Windows Remote Management portal. Retrieved February 14, 2025, from https://learn.microsoft.com/en-us/windows/win32/winrm/portal

Red Hat. How Ansible works. Retrieved February 14, 2025, from

https://www.redhat.com/en/ansible-collaborative/how-ansible-works

Semaphore UI. Semaphore installation: Docker (2.12). Retrieved February 14, 2025, from https://semaphoreui.com/install/docker/2_12

Semaphore UI. Semaphore API documentation. Retrieved February 14, 2025, from https://semaphoreui.com/api-docs/

Semaphore UI. Semaphore UI. Retrieved February 14, 2025, from https://semaphoreui.com/

Ansible (software). Wikipedia(n.d.). Retrieved February 14,2025, from https://en.wikipedia.org/wiki/Ansible_(software)

Microsoft Deployment Toolkit. Wikipedia (n.d.). Retrieved February 14, 2025, from https://en.wikipedia.org/wiki/Microsoft_Deployment_Toolkit

YAML Checker. Free YAML syntax validator. Retrieved February 14, 2025, from https://yamlchecker.com/