Journal of Artificial Intelligence and Engineering Applications

Website: https://ioinformatic.org/

June 2026. Vol. 5. No. 3; e-ISSN: 2808-4519

Implementing Infrastructure as Code Using Ansible on Debian Server Administration

Elsahday Tambunan[1]*, Anastasya Jesica Sidauruk[2], Lotar Mateus Sinaga[3]

[1,2,3]*Universitas Katolik Santo Thomas*

elsahdaytambunan@gmail.com[1*], anastasyasidauruk1877@gmail.com[2] , lotarmateus88@gmail.com[3]

Abstract

The growth of cloud computing and virtualization has changed how IT infrastructure is managed, moving from doing things by hand to using Infrastructure as Code (IaC) methods. This study uses Ansible as a tool for managing server setup to automate tasks on Debian 12 servers within a virtualized environment. The setup includes four main areas of administration:managing users and masing SSH more secure, setting up a firewall using UFW,insatlling the Ngnix web server with Jinja2 templates, and setting up automatic security uodates through unattended upgrades. The method used is Design Science Research (DSR), which involves an experimental approach that uses two connected virtual machines. The results show that Ansible was able to fully aautomate the setup process, taking a total of 54.121. seconds to complete. It also demonstrated that all the roles worked correctly even when run more than once, with no changes needed on re-run. This setup helps cut down on mistakes people might make, makes sure settings are the same every time, and allows for creating the same infrastructure again and again.

Keywords: *Infrastructure as Code, Ansible, Debian 12, Server Administration, Automation, DevOps, Idempotency.*

1. Introduction

Developments in the field of information technology are encouraging various organizations to manage increasingly complex server infrastructures, both in data center environments, cloud environments, and hybrid systems [1]. Server management that is still done manually has shortcomings, such as high human errors, misalignment of configurations between servers, deployment processes that take a long time, and difficulties in replicating configurations when the number of servers increases [2]. These issues can lead to low operational efficiency and increased infrastructure maintenance costs. Therefore, organizations need an approach that can automate server management processes in a consistent, documented, and easily reproducible manner [3].

In the implementation of Infrastructure as Code, Ansible is one of the automation platforms that is widely used because it has an agentless architecture, uses the Secure Shell (SSH) protocol, and utilizes YAML-based playbooks that are easy to understand and maintain [4]. These characteristics make Ansibel capable of automating various server administration activities, such as user management, service configuration, firewall settings, and strengthening system security (*securuty herdening)* [5].

One frequently used strategy is Infrastructure as Code (IaC). Infrastructure as Code is the management of infrastructure through code that can be stored in a version control system, tested, and operated automatically, resulting in a consistent configuration for each deployment. This approach transforms infrastructure management from manual to automated and well-documented. In addition to improving configuration consistency, IaC also supports DevOps principles through automation, reproducibility, and ease of infrastructure maintenance [6].

The purpose of this research is to implement Infrastructure as Code using Ansible in Debian 12 server administration, evaluate the success of server configuration automation through functional and idempotency testing, and analyze the effectiveness of the implementation based on playbook execution time and the consistency of the resulting configuration. The results of this research are expected to be a reference for system administrators and researchers in implementing Infrastructure as Code-based server administration automation in Linux environments, especially Debian 12.

2. Theoretical Study

2.1. Infrastructure as Code (AaC)

Infrastructure as Code (AaC) is an approach to infrastructure administration that allows server configurations to be defined in code, allowing for automated execution, documentation, and easy replication [7]. IaC makes infrastructure provisioning and configuration more consistent than manual configuration.

2.2 Ansible

Ansible is open-source software for server administration automation that utilizes an agentless architecture and utilizes the SSH protocol for communication with target servers. All configurations are written in playbooks using YAML syntax, making them easy to understand and maintain. Ansible is widely used in configuration management, application deployment, and service orchestration because it reduces configuration errors and speeds up server administration.

2.3 Debian

Debian 12 is a Linux distribution widely used as a server operating system due to its stability, security, and comprehensive software package support. This operating system supports various network services such as SSH, Nginx, and UFW, making it suitable for implementing server administration automation using Ansible.

2.4 Role and Playbook Ansible

A playbook is a collection of automation instructions written in YAML format, while roles are used to group configurations based on specific functions, making them more structured and reusable. Using roles simplifies configuration management, such as user creation, SSH configuration, firewall configuration, web server installation, and system updates, within a single automation project.

2.5 Idempotency

One of Ansible's key characteristics is idempotency, the ability to run playbooks repeatedly without causing configuration changes if server conditions meet the desired state. This concept ensures configuration consistency, allowing administrators to safely deploy and update configurations without worrying about unnecessary changes. Idempotency testing is one indicator of the success of Infrastructure as Code implementation in this study.

3. Research Methods

3.1 Types of Research

This research uses an experimental research method with a quantitative approach. The experimental method was used to implement server administration automation using Ansible on the Debian 12 operating system, then evaluate the configuration results based on deployment success, idempotency, playbook execution time, and the success of the service functions executed. This approach allows researchers to obtain data directly from the results of the system implementation in a prepared test environment [7].

3.2 Research Environment

The research was conducted in a virtual environment using VirtualBox with two virtual machines: one computer as the Ansible Control Node and one computer as the Managed Node, running the Debian 12 operating system. The two machines were connected via an internal network, enabling communication using the Secure Shell (SSH) protocol. Ansible was used to automate server configuration without requiring an agent on the target server. These steps align with the implementation performed during the practicum [8].

3.3 Research Stages

The research phase began with installing Ansible on the control node and configuring SSH authentication using a public key to allow password-free communication with the managed node. After a successful connection, a host inventory and an Ansible playbook were created as the basis for automating server configuration [9].
Next, we implemented four Ansible roles: user_ssh_hardening, firewall, webserver, and auto_updates. The user_ssh_hardening role is used to create user deployments and improve SSH access security. The firewall role is used to configure UFW so that only required ports are accessible. The webserver role is used to install and configure the Nginx service using a Jinja2 template, while the auto_updates role is used to enable automatic security updates on Debian systems.
After all roles are created, the playbook is run using the ansible-playbook command to apply the entire configuration to the managed node [10]. Testing is then performed by rerunning the playbook to ensure idempotency, meaning that no configuration changes occur if the server conditions are in accordance with the desired configuration [11].

3.4 Data Collection Techniques

Research data was obtained directly from the system implementation results. The collected data included playbook execution results, configuration change status (changed), execution times using the time command, Nginx service verification results, UFW firewall status, automatic security update configuration, and SSH security test results. All of this data was used to evaluate the success of server administration automation using Ansible.

3.5 Data Analysis Techniques

Data analysis was performed descriptively by comparing configuration results before and after Ansible playbook implementation. Parameters analyzed included the success of each role, the number of configuration changes, service deployment success, playbook

execution time, and idempotency testing success. The analysis results were used to assess the effectiveness of server administration automation in improving configuration consistency, deployment efficiency, and system security.

4. Results and Discussion

4.1 Infrastructure and Connectivity Preparation

The first phase of the implementation involved setting up two Debian 12 VMs, installing Ansible version 2.14.x on the control node, and configuring SSH key-based authentication. Connectivity between the nodes was verified using Ansible's built-in ping module.

```
ansible@ansible-control:~/ansible-project$ ansible -i inventory/hosts.ini target -m pin
g
ansible-target | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
ansible@ansible-control:~/ansible-project$
```

Figure 1: Ansible Connection Verification Output

A SUCCESS response with ping:pong confirms that the control node successfully communicated with the managed node via passwordless SSH, proving the infrastructure is ready to run the playbook.

Role 1: User Management and SSH Hardening
This role implements five fundamental security configuration tasks, including: creating a non-root user (deploy) with sudo group membership; setting up the .ssh directory with 0700 permissions; registering the node's public control key to authorized_keys; disabling root login via SSH; and disabling SSH password authentication. Here's an example YAML task for disabling root login:

```
- name: Disable SSH root login
  lineinfile:
    path: /etc/ssh/sshd_config
    regexp: '^#?PermitRootLogin'
    line: 'PermitRootLogin no'
  notify: Restart SSH
```

Figure 2: Task YAML SSH Hardening on Role user_ssh_hardening

Using the lineinfile module ensures that only the relevant configuration lines are modified, without disrupting other configurations in the sshd_config file. The SSH restart handler will be called automatically only when a configuration change occurs, minimizing unnecessary restarts.

Role 2: Configuration-Based Firewall (UFW)
The firewall role implements the principle of least privilege by defining a default deny rule and allowing only absolutely necessary ports:

```
- name: Install UFW
  apt:
    name: ufw
    state: present
    update_cache: yes

- name: Allow SSH (port 22)
  ufw:
    rule: allow
    portL '22'
    proto: tcp

- name: Allow HTTP (port 80)
  ufw:
    rule: allow
    port: '80'
    proto: tcp

- name: Enable UFW
  ufw:

- name: Enable UFW
  ufw:
    state: enabled
    policy: deny
```

Figure 3: UFW Firewall Role Tasks

Verifying the firewall status on the managed node after execution shows:

```
ansible@ansible-target:~$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW IN    Anywhere
80/tcp                     ALLOW IN    Anywhere
22/tcp (v6)                ALLOW IN    Anywhere (v6)
80/tcp (v6)                ALLOW IN    Anywhere (v6)

ansible@ansible-target:~$
```

Figure 4: UFW Status Output After Configuration

Role 3: Nginx Web Server with Jinja2 Templating
The webserver role demonstrates Ansible's templating capabilities using the Jinja2 templating engine. Deployed web pages automatically populate dynamic variables such as hostnames and IP addresses from data collected by Ansible.

```
<!DOCTYPE html>
<html>
<head>
        <title>Server Provisioned by Ansible</title>
</head>
<body>
        <h1>Selamat Datang!</h1>
        <p>Server ini berhasil di konfigurasi otomatis menggunakan Ansible.</p>
        <p>Hostname: {{ ansible_hostname }}</p>
        <p>IP Address: {{ ansible_default_ipv4.address }}</p>
</body>
</html>
```

Figure 5: Jinja2 Template for Nginx Index Page

Verification using curl from the node instance confirms that the web server is successfully serving the confirmed content.

```
ansible@ansible-control:~$ curl http://10.0.2.5
<!DOCTYPE html>
<html>
<head>
        <title>Server Provisioned by Ansible</title>
</head>
<body>
        <h1>Selamat Datang!</h1>
        <p>Server ini berhasil di konfigurasi otomatis menggunakan Ansible.</p>
        <p>Hostname: ansible-target</p>
        <p>IP Address: 10.0.2.5</p>
</body>
</html>
```

Figure 6: Web Server curl output

Role 4: Automatic Security Updates
The auto_updates role configures the unattended-upgrades package to automate the installation of security updates. The applied configuration ensures that only security category updates are automatically installed, with no updates requiring an automatic reboot:

```
 GNU nano 7.2      roles/auto_updates/templates50unattended-upgrades.j2 *
Unattended-Upgrade::Origins-pattern{
        "origini=Debian,codename=${distro_codename},label=Debian-security";
};

Unattended-Upgrade::Remove-Unused-Dependencies "true";
Unattended-Upgrade::Automatic-Reboot "false";
```

Figure 7: Unattended-upgrades configuration

```
ASK [webserver : Install Nginx] ****************************************************
k: [ansible-target]

ASK [webserver : Deploy custom index page] *****************************************
k: [ansible-target]

ASK [webserver : Ensure Nginx is running and enabled] ******************************
k: [ansible-target]

ASK [auto_updates : Install unattended-upgrades package] ***************************
k: [ansible-target]

ASK [auto_updates : Deploy unattended-upgrades configuration] **********************
k: [ansible-target]

ASK [auto_updates : Enable automatic updates] **************************************
hanged: [ansible-target]

LAY RECAP **************************************************************************
nsible-target              : ok=16   changed=1    unreachable=0    failed=0    skipped=
    rescued=0    ignored=0

nsible@ansible-control:~/ansible-project$
```

Figure 8: Ansible-Playbook Execution

4.2 Analysis of Execution Results and Idempotency

```
real    0m54.121s
user    0m7.934s
sys     0m5.072s
ansible@ansible-control:~/ansible-project$
```

Figure 9: Execution time results

A comprehensive evaluation was performed by executing the playbook twice for each role—the first execution (initial deployment) and the second execution (idempotency check). The collected data is presented in the following table:

Table 1: Playbook Execution Results: Comparison of Run 1 vs Run 2

Role	Run 1 (initial)	Run 2 (idempotency)
User & SSH Hardening	ok=7, changed=6	ok=6, changed=0
Firewall (UFW)	ok=10, changed=4	ok=10, changed=0
Web Server (Nginx)	ok=14, changed=3	ok=14, changed=0
Auto Updates	ok=16, changed=1	—
Full playbook (semua role)	—	ok=16, changed=0, **waktu: 54.121s**

All roles successfully verified the idempotency property with changed=0 on the second execution. The full playbook execution time under idempotent conditions (all configurations met the target) was 54.121 seconds, with a breakdown of 7.934 seconds of user time (Ansible processes) and 5.072 seconds of system time (kernel overhead).

4.3 Integrated Functional Verification

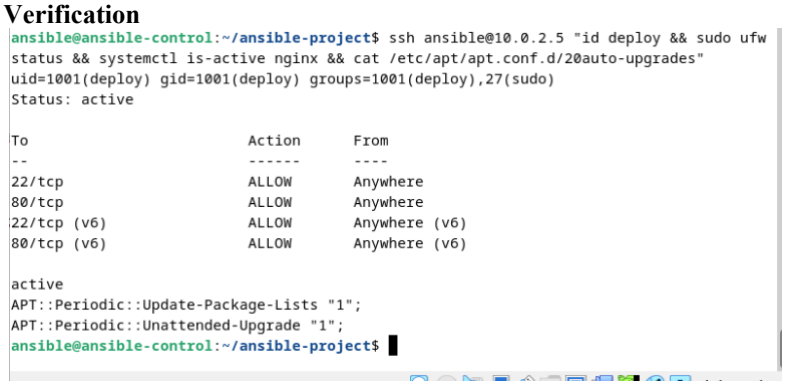

Figure 10: Integrated Functional Verification Output

The verification results confirm that all four aspects of the configuration are working correctly simultaneously: the deploy user is registered with the sudo group, the UFW firewall is active, Nginx is running (active), and the auto-update configuration is saved correctly.

5. Conclusion

This research successfully implemented Infrastructure as Code using Ansible to comprehensively automate Debian 12 server administration. Ansible proved effective as an IaC tool with the ability to automate all four domain configurations—user management and SSH hardening, UFW firewall, Nginx web server, and automatic security updates—in a single integrated ecosystem. Ansible's idempotency property proved consistency across roles, making it ideal for routine compliance checks and automated configuration drift remediation. The execution time of 54.121 seconds for the full playbook demonstrated significant efficiency compared to manual configurations that take hours with a higher risk of errors. The role-based architecture successfully improved the modularity and maintainability of the infrastructure code, aligning with sound software engineering principles.

Acknowledgement

The author expresses his gratitude to God Almighty for His grace and blessings, enabling the successful completion of this research. He also expresses his gratitude to the lecturers in charge of the course who provided guidance, direction, and practical materials, enabling the successful implementation of this research on server administration automation using Ansible on the Debian 12 operating system.
He also expresses his gratitude to the Informatics Engineering Study Program and all parties who provided laboratory facilities, equipment, and a supportive learning environment for the practical work and the preparation of this scientific article. He also extends his gratitude to his family and friends who provided support, motivation, and assistance throughout the research process and the completion of this article. He acknowledges that this research has limitations. Therefore, constructive criticism and suggestions are highly appreciated for the development of further research.

References

[1] I. Kumara *et al.*, "The do ' s and don ' ts of infrastructure code : A systematic gray literature review," *Inf. Softw. Technol.*, vol. 137, no. March, hal. 106593, 2021, doi: 10.1016/j.infsof.2021.106593.
[2] G. Rathor, "INTELLIGENT SYSTEMS AND APPLICATIONS IN Infrastructure as Code for Performance Engineering : Automating Deployment and Testing with Terraform and Ansible," vol. 11, hal. 420–427, 2023.
[3] I. Byzov dan P. Student, "Terraform vs Ansible : When and how to use infrastructure tools as code," vol. 25, no. 6, hal. 11–17, 2024, doi: 10.30857/2786-5371.2024.6.1.
[4] R. Jurnal, T. Informatika, A. Fadhudin, dan D. Ramayanti, "Implementasi Sistem Otomatisasi Konfigurasi Server Berbasis Ansible dan Semaphore pada Infrastruktur TI Perusahaan Ritel," vol. 5, no. 1, hal. 1–13, 2026.
[5] D. Regvart, J. Red, A. Bubnjek, dan R. Petruni, "Security hardening using infrastructure as code," vol. 9, no. 2, hal. 1147–1155, 2025, doi: 10.55214/25768484.v9i2.4697.
[6] E. S. Engineering, A. Rahman, E. Farhana, dan L. Williams, "for Infrastructure as Code".
[7] W. Riski, A. Putra, A. Reza, A. Nurwa, D. F. Priambodo, dan M. Hasbi, "Infrastructure as Code for Security Automation and Network

Infrastructure Monitoring," vol. 22, no. 1, hal. 201–214, 2022, doi: 10.30812/matrik.v22i1.2471.
[8] M. R. Afandi, P. Hatta, dan A. Efendi, "Otomatisasi Perangkat Jaringan Komputer Menggunakan Ansible Pada Laboratorium Komputer," vol. 6, no. 2, hal. 48–53, 2020.
[9] T. Online, K. H. Hutapea, M. Arif, dan F. Ridha, "Jurnal Politeknik Caltex Riau Implementasi Continuous Delivery dengan Zero – Downtime Rolling Update Menggunakan Ansible," vol. 8, no. 2, hal. 316–323, 2022.
[10] S. Dalla, D. Di, dan D. A. Tamburri, "SoftwareX AnsibleMetrics : A Python library for measuring Infrastructure-as-Code blueprints in Ansible," *SoftwareX*, vol. 12, hal. 100633, 2020, doi: 10.1016/j.softx.2020.100633.
[11] A. N. Hidayat dan A. Wre, "Automating Server Deployment with Ansible to Improve Performance , Virtualization , and Network Security," vol. 5, no. 8, hal. 10618–10626, 2025.