FOSSEE Autumn Internship Report

on

System Administration

Submitted By:

Srijan Maurya

VIT Bhopal University

Under the Guidance of:

Thomas Stephen Lee

and

Raghavjit Rana

January 2026

Contents

Acknowledgment **4**

Declaration **5**

Introduction **6**

Development Environment Note **7**

1 Pre-requisite Task: Setting Up Host-Only Adapter **8**
1.1 Network Architecture 9
1.2 Configuration Steps 9
1.2.1 Step 1: VirtualBox Network Manager 9
1.2.2 Step 2: VM Adapter Assignment 9
1.2.3 Step 3: In-Guest Static IP Configuration 9
1.3 Security & Firewall 10
1.4 Validation Results 10

2 Drupal Installation & Keycloak SSO Integration (Automated Setup) **11**
2.1 Objective 11
2.2 Infrastructure Setup 11
2.3 Phase 1: Drupal and Keycloak Installation 11
2.3.1 Server A – Keycloak Setup 11
2.3.2 Server B – Drupal Setup 12
2.4 Phase 2: Automated Keycloak Client Setup (Server A) 12
2.4.1 Directory Setup 12
2.4.2 Required Files 12
2.4.3 Script Setup and Execution 13
2.5 Phase 3: Automated Drupal OIDC Configuration (Server B) 13
2.5.1 Access Container 13
2.5.2 Run Script 13
2.6 Drupal createClient() Configuration Explanation 14
2.7 Additional Automated Steps Performed 14
2.8 Validation and Testing 14
2.9 Conclusion 15

3 Keycloak Version Upgrade and Data Migration **16**
3.1 Objective 16
3.2 Part A: Backup Strategy for Version Upgrade and Data Migration 17

3.2.1 Backup Script Development . . . 17
3.2.2 Migration Validation on a Fresh Environment . . . 17
3.3 Part B: Version Management & Upgrade . . . 18
3.3.1 Installation of Legacy Version (v21.0.0) . . . 18
3.3.2 In-Place Upgrade to Stable (v26.4.0) . . . 18
3.3.3 Challenges & Resolutions . . . 18
3.4 Conclusion . . . 19

4 Mailman 3 Deployment & Data Migration 20
4.1 Objective . . . 20
4.2 Infrastructure & Prerequisites . . . 21
4.3 Implementation Details . . . 21
4.3.1 System Preparation . . . 21
4.3.2 Component Deployment . . . 22
4.4 Data Migration (Mailman 2 to 3) . . . 22
4.4.1 Artifact Extraction . . . 22
4.4.2 Migration Execution . . . 22
4.5 Challenges & Resolutions . . . 23
4.6 Conclusion . . . 23

5 Postfix, Dovecot & Roundcube Deployment 24
5.1 Objective . . . 24
5.2 Infrastructure . . . 24
5.3 Implementation Details . . . 24
5.3.1 Phase 1: Mail Server Configuration (Postfix & Dovecot) . . . 24
5.3.2 Phase 2: Roundcube Webmail Installation . . . 25
5.3.3 Phase 3: Security & System Integration . . . 25
5.4 Testing & Validation . . . 26
5.5 Conclusion . . . 27

6 Automated CI/CD Pipeline for Astro Web Deployment 28
6.1 Objective . . . 28
6.2 Architecture Overview . . . 28
6.3 Environment Configuration . . . 29
6.3.1 Jenkins Server Setup (Server A) . . . 29
6.3.2 Web Server Setup (Server B) . . . 29
6.4 Security and Credential Management . . . 30
6.5 The Pipeline Definition (Jenkinsfile) . . . 30
6.6 Automation and Testing . . . 31
6.7 Conclusion . . . 31

7 Intelligent Mail Filtering: The AI-Driven "Intelligent Funnel" 32
7.1 Objective . . . 32
7.2 Architectural Design: The Intelligent Funnel . . . 32
7.3 Phase 1: Mail Transfer Agent (Postfix) Configuration . . . 33
7.4 Phase 2: Containerized Heuristic Engine (Rspamd) . . . 33
7.5 Phase 3: The AI Oracle (Python & Hugging Face) . . . 33
7.5.1 Microservice Implementation . . . 33
7.6 Phase 4: Integration and Cross-Network Communication . . . 34

7.7 Testing and Performance Results 34
7.8 Rationale for Modern Tooling 34
7.9 Conclusion 35

8 Kanboard Deployment & Zulip Integration **36**
8.1 Objective 36
8.2 Prerequisites 36
8.3 Phase 1: Kanboard Installation (Docker Compose) 36
8.4 Phase 2: Zulip Webhook Configuration 37
8.5 Phase 3: Integration and Plugin Setup 37
8.6 Testing and Validation 38
8.7 Conclusion 39

Acknowledgment

I want to take this opportunity to express my heartfelt gratitude to everyone who played a crucial role in making my autumn internship with FOSSEE - System Administration a valuable and rewarding experience. First and foremost, I extend my deepest thanks to Mr. Thomas Stephen Lee for believing in my abilities and selecting me for this project. Their unwavering support and encouragement have been instrumental in my progress.

I am deeply grateful to my mentor Raghavjit Rana, whose guidance and wisdom have been indispensable throughout this journey. His advice and timely assistance helped me complete my tasks and enhanced my technical and non-technical skills.

Additionally, I want to express my appreciation to my colleague, Kolli Jyothi Swaruph who is working on various projects. His constructive feedback and insightful discussions have significantly contributed to my learning and growth.

I extend my sincerest gratitude to everyone who supported me during this internship for their contributions to this unforgettable experience.

Declaration

I hereby declare that this written submission represents my ideas in my own words. Whenever the ideas or words of others have been included, I have appropriately cited and referenced the sources.

I have accurately acknowledged all sources used in producing this report. Furthermore, I confirm that I have adhered to all academic honesty and integrity principles. I have not misrepresented, fabricated, or falsified any idea, data, fact, or source in this submission.

I understand that any violation of the above principles will lead to disciplinary action by the Institute and may also result in legal consequences from sources that have not been appropriately cited or from those from whom proper permission was not obtained where required.

Srijan Maurya

Introduction

During the tenure of my internship, I had the opportunity to work on several tasks recommended by Mr. Thomas Stephen Lee. These tasks primarily required maintaining a structured daily progress record while performing tasks related to software development, debugging, documentation, and project collaboration. This report presents a consolidated overview of the work carried out throughout the internship, as documented in the accompanying Git repository. The primary objective of the repository was to track day-to-day activities, challenges, learning outcomes, and deliverables in a transparent and organized manner.

The project involved consistent engagement with assigned tasks, understanding technical workflows, resolving issues, and implementing improvements wherever necessary. Each entry in the repository reflects incremental progress — from initial setup and orientation to executing responsibilities, refining solutions, and documenting outcomes. Maintaining this log ensured clarity in task management, accountability in workflow, and a systematic record of the internship journey.

This final report compiles these daily progress notes into a cohesive narrative, highlighting key contributions, technical insights gained, and the overall growth achieved during the internship period. It serves both as a reflection of the work completed and a demonstration of disciplined project tracking aligned with FOSSEE's standards.

Development Environment Note

All configurations, deployments, and experiments described in this report were performed in a controlled development environment using virtual machines. The objective of this setup was to safely test system configurations, integrations, and automation workflows without affecting production infrastructure.

For real-world production deployment, additional steps such as security hardening, domain configuration, SSL certificate management, load balancing, monitoring, and backup strategies would need to be implemented.

Choice of Operating Systems

Throughout the internship tasks, different RHEL-compatible Linux distributions were used, primarily Rocky Linux 10 and AlmaLinux 10. Both distributions are community-supported downstream rebuilds of Red Hat Enterprise Linux (RHEL) and provide nearly identical package compatibility, security updates, and enterprise-grade stability.

Rocky Linux was initially used for tasks involving Identity and Access Management systems such as Keycloak and Drupal because the initial development environment had already been provisioned using this distribution.

For later tasks involving mail servers, CI/CD pipelines, and containerized applications, AlmaLinux was used on newly provisioned virtual machines. Since AlmaLinux maintains full binary compatibility with RHEL, switching between these distributions did not affect software compatibility or configuration procedures.

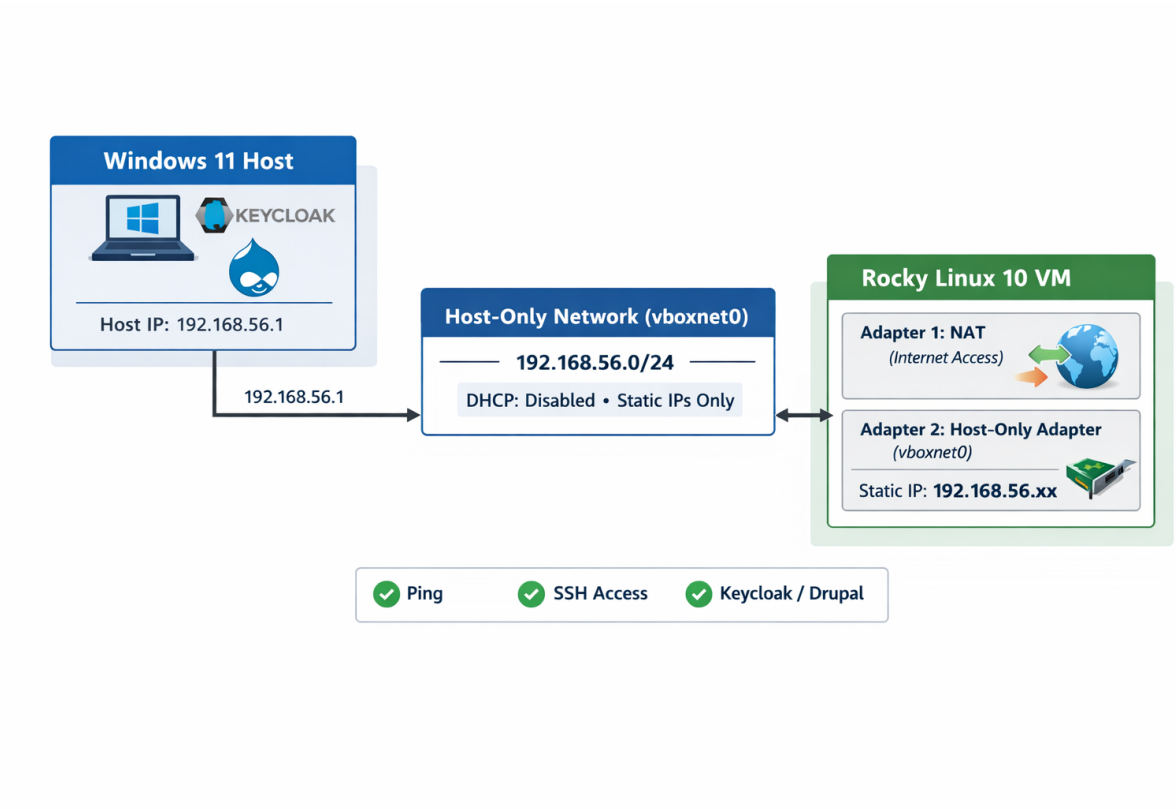
Therefore, the use of both Rocky Linux and AlmaLinux in this report reflects the availability of different development environments during the internship while maintaining compatibility within the same RHEL-based ecosystem.

Chapter 1

Pre-requisite Task: Setting Up Host-Only Adapter

Objective

To establish a private, static network communication channel between the Host machine (in my case Windows 11) and the Guest Virtual Machine (Rocky Linux 10) to facilitate consistent access for services like SSH, Keycloak, and Drupal.



1.1 Network Architecture

This configuration utilizes a **Host-Only Adapter**, which creates a private internal network (`vboxnet0`) visible only to the host and the VM. This prevents IP address changes (common with DHCP/NAT) that would break Keycloak redirection and SSO configurations.

1.2 Configuration Steps

1.2.1 Step 1: VirtualBox Network Manager

- **Action:** A new Host-Only network was created via the VirtualBox Host Network Manager.
- **Settings:**
  - **Adapter Name:** `vboxnet0`
  - **Host IP:** `192.168.56.1`
  - **Subnet Mask:** `255.255.255.0`
  - **DHCP Server: Disabled** (Crucial for manual static IP assignment).

1.2.2 Step 2: VM Adapter Assignment

- **Action:** The Rocky Linux 10 VM settings were updated to include a second network interface.
- **Interface Map:**
  - **Adapter 1:** NAT (Used for Internet access/Updates).
  - **Adapter 2:** Host-Only Adapter (Attached to `vboxnet0` for internal static communication).

1.2.3 Step 3: In-Guest Static IP Configuration

Inside the Rocky Linux VM, the new interface (identified as `enp0s8`) was configured with a permanent static IP using `nmcli`.

Command Execution:

```
sudo nmcli connection add \
  type ethernet \
  con-name HostOnly \
  ifname enp0s8 \
  ipv4.addresses 192.168.56.xx/24 \
  ipv4.method manual \
  connection.autoconnect yes
```

Persistence Check: To ensure the configuration survives reboots, the connection file at `/etc/NetworkManager/system-connections/HostOnly.nmconnection` was verified to contain `autoconnect=true` and `method=manual`.

1.3 Security & Firewall

To allow traffic (Ping, SSH, HTTP) from the host to the VM, the interface was added to the **trusted** zone in the firewall.

```
sudo firewall-cmd --permanent --zone=trusted --change-interface=enp0s8
sudo firewall-cmd --reload
```

1.4 Validation Results

- **Host to VM:** `ping 192.168.56.xx` — **Successful**.
- **VM to Host:** `ping 192.168.56.1` — **Successful**.
- **SSH Access:** Login via `ssh user@192.168.56.xx` confirmed.

Outcome: The Virtual Machine now has a stable, static IP address (192.168.56.xx), fulfilling the network requirements for the subsequent Keycloak and Drupal installation tasks.

Chapter 2

Drupal Installation & Keycloak SSO Integration (Automated Setup)

2.1 Objective

To establish a decoupled architecture consisting of a Drupal-based web application and a centralized Keycloak Identity Provider (IdP) on separate Virtual Machines (Rocky Linux 10), and to automate OpenID Connect (OIDC) authentication using CLI-based scripts.

2.2 Infrastructure Setup

Two separate Virtual Machines were provisioned running **Rocky Linux 10**.

- **Network Configuration:** A Host-Only Adapter was configured on both VMs to ensure static IP communication between the host and guest machines. This is essential for reliable SSO redirection.
- **Server A:** Keycloak Identity Provider
- **Server B:** Drupal (OpenPLC website)

Reference: Separated VM Setup documentation (Host-Only Adapter configuration).

2.3 Phase 1: Drupal and Keycloak Installation

2.3.1 Server A – Keycloak Setup

Keycloak was installed on a dedicated VM with:

- OpenJDK 21
- MySQL 8.4
- Keycloak configured to bind to Host-Only static IP

2.3.2 Server B – Drupal Setup

Drupal (OpenPLC website) was deployed using Podman containerization.

```
./init_sites
```

Detailed documentation for the `init_sites` script and server setup procedure is available at: OpenPLC Website Setup Documentation (Server B)

The SQL dump (`openplc.sql`) was imported automatically, and the container runs on port **8081**.

2.4 Phase 2: Automated Keycloak Client Setup (Server A)

Instead of manually creating clients from the Admin Console, an automated CLI-based script was used.

2.4.1 Directory Setup

```
mkdir ~/keycloak
cd keycloak
```

2.4.2 Required Files

config.json

```
[
  {
    "keycloak_realm": "",
    "keycloak_base": "",
    "keycloak_user": "",
    "keycloak_password": "",
    "drupal_base": "",
    "keycloak_client_id": ""
  }
]
```

client.json

```
{
  "clientId": "",
  "enabled": true,
  "publicClient": false,
  "protocol": "openid-connect",
  "clientAuthenticatorType": "client-secret",
  "standardFlowEnabled": true,
  "redirectUris": [""]
}
```

2.4.3 Script Setup and Execution

Add the script:

```
wget https://github.com/CoolSrj06/Fossee-Daily-Progress/blob/master/
   Automated%20SSO%20Setup/keycloak_user_config_script.md
```

Fix permissions and run the script:

```
sudo chown -R keycloak:$USER .
sudo chmod -R g+w .
chmod +x keycloak
./keycloak
```

The script performs:

1. Validates input (lowercase realm, valid URL, no duplicate client).
2. Logs into Keycloak using `kcadm.sh`.
3. Creates a new OpenID Connect client.
4. Automatically generates and retrieves the Client Secret.
5. Stores client credentials inside `client.json`.

A **secret.json** file is generated, containing the Client Secret required for Drupal configuration.

2.5 Phase 3: Automated Drupal OIDC Configuration (Server B)

Drupal configuration was automated using the `openplc_config_script` inside the running Podman container.

2.5.1 Access Container

```
podman exec -it <container-name> bash
```

2.5.2 Run Script

```
chmod +x openplc_config_script

./openplc_config_script \
--client-id openplc \
--client-secret <secret> \
--base http://<keycloak-ip>:8080 \
--realm master \
--email admin@example.com \
--drupal-user admin \
--drupal-password password \
--user <shell-user>
```

2.6 Drupal createClient() Configuration Explanation

Configurations applied inside createClient() function of Drupal Script

2.7 Additional Automated Steps Performed

The Drupal script also:

- Installs required modules:

```
    composer require drupal/openid_connect drupal/keycloak
    drush en openid_connect keycloak -y

```

- Creates a Drupal administrator user.
- Assigns administrator role.
- Enables `connect_existing_users` option.
- Clears cache after all configurations.

2.8 Validation and Testing

1. Open Drupal site: `http://<Drupal-IP>:8081`
2. Click login.
3. Browser redirects to Keycloak.
4. Enter Keycloak credentials.
5. Successful authentication redirects back to Drupal dashboard.

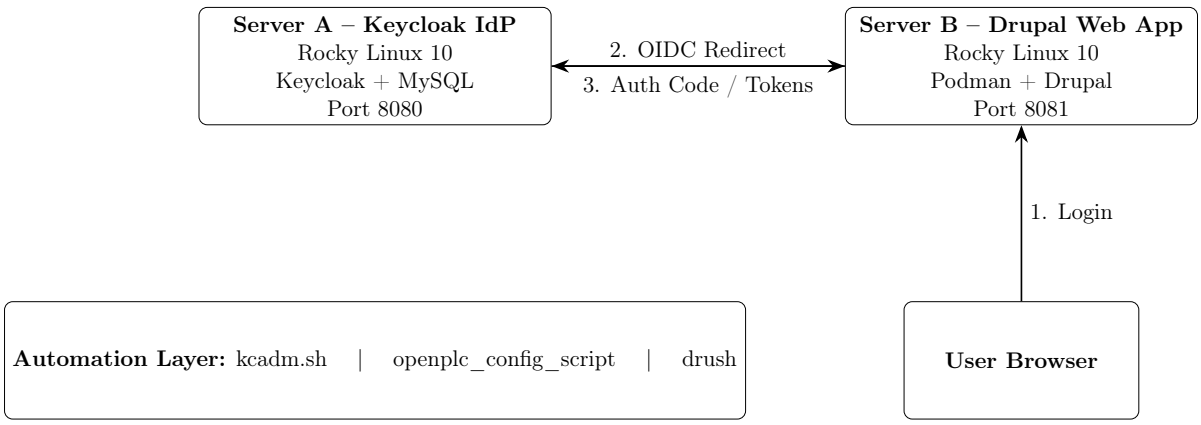


Figure 2.1: Architecture Diagram: Automated Drupal–Keycloak SSO Setup

2.9 Conclusion

The Drupal–Keycloak integration was successfully implemented using fully automated CLI-based scripts.

Key achievements:

- Separate VM architecture with static IP communication.
- Automated Keycloak client creation using `kcadm.sh`.
- Automated Drupal OpenID Connect configuration using `drush`.
- Cache handling and validation checks integrated into scripts.
- Fully reproducible SSO setup without manual UI configuration.

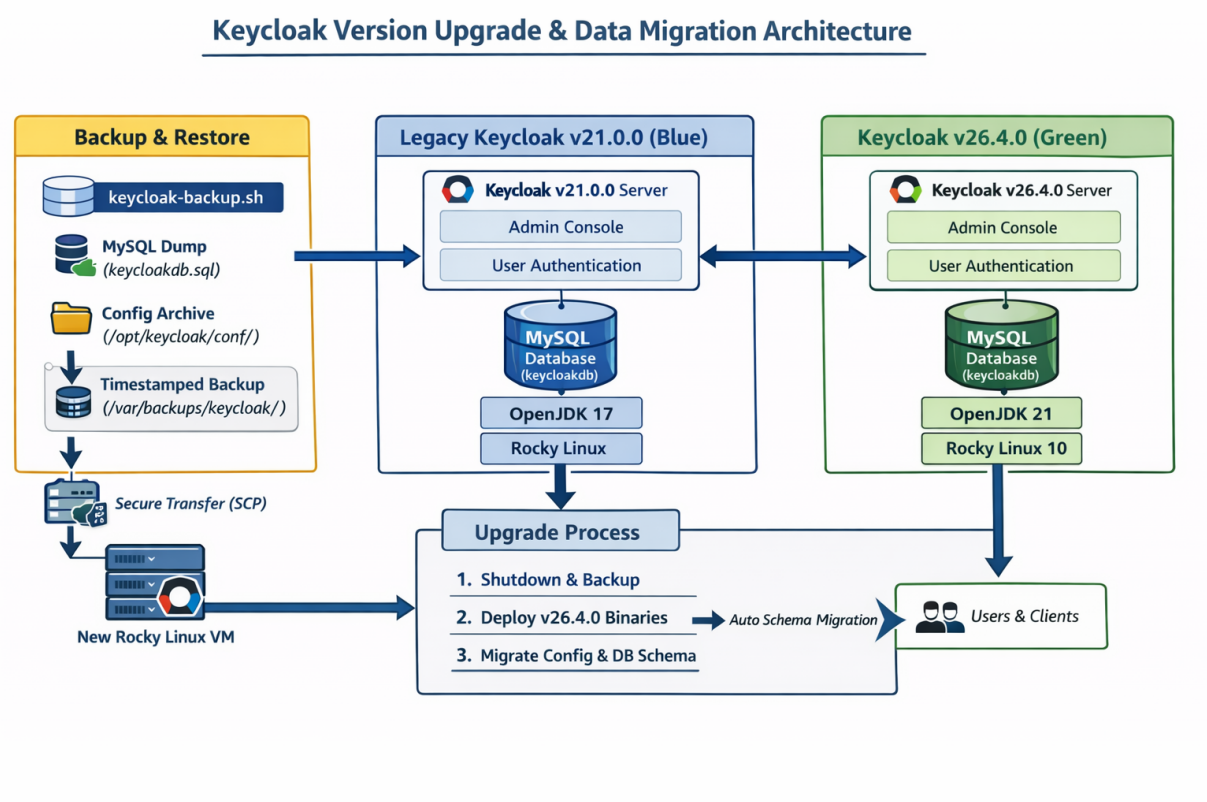
The authentication pipeline is now production-ready, reproducible, and fully script-driven.

Chapter 3

Keycloak Version Upgrade and Data Migration

3.1 Objective

To demonstrate secure version upgrade and data migration of the Identity Access Management (IAM) infrastructure by implementing a structured backup and restore workflow, and upgrading an outdated Keycloak instance (v21.0.0) to the current stable release (v26.4.0) while preserving data integrity.



3.2 Part A: Backup Strategy for Version Upgrade and Data Migration

3.2.1 Backup Script Development

A robust Bash script (`keycloak-backup.sh`) was developed to capture the entire state of the Keycloak server. This includes the persistent data layer (MySQL) and the configuration layer.

- **Script Logic:**
  1. **Database Dump:** Uses `mysqldump` to export the `keycloakdb` into a SQL file.
  2. **Config Archive:** Uses `tar` to compress the `/opt/keycloak/conf/` directory.
  3. **Timestamping:** All artifacts are saved in a time-stamped directory (e.g., `/var/backups/keycloak/2025-11-11_16-02-11/`) for easy versioning.
- **Automation:** A cron job was configured to execute this script automatically every day at 02:00 AM.

```
0 2 * * * /usr/local/bin/keycloak-backup.sh >> /var/log/keycloak-backup.log 2>&1
```

- **Manual Execution:** The script is fully executable on-demand for ad-hoc backups before critical maintenance (e.g., upgrades).

3.2.2 Migration Validation on a Fresh Environment

To validate the migration process, Keycloak was installed and restored on a fresh Rocky Linux 10 VM to simulate environment portability.

- **Restoration Steps:**
  1. **Environment Prep:** Installed Java 21, MySQL 8.4, and the Keycloak binaries on the new VM.
  2. **Artifact Transfer:** Securely copied (scp) the latest backup archive from the primary server to the new VM.
  3. **Database Restoration:** Imported the SQL dump into the fresh MySQL instance.

```
mysql -u keycloakuser -p keycloakdb < keycloak-db.sql

```

  4. **Config Restoration:** Extracted the backed-up configuration files to `/opt/keycloak/conf/`, ensuring hostname and database credentials matched the new environment.
- **Validation:** The Keycloak service was started, and a successful login was performed using the **old administrator credentials**, confirming that all user data and realm configurations were successfully restored.

3.3 Part B: Version Management & Upgrade

3.3.1 Installation of Legacy Version (v21.0.0)

To simulate a real-world migration scenario, an older version of Keycloak (v21.0.0, released approx. early 2023) was first installed.

- **Prerequisites:** Configured **OpenJDK 17** (required for v21.x) and MySQL.
- **Configuration:**
  - Created a specific `keycloakdb` database.
  - Populated the instance with sample users and clients to serve as test data for the upgrade validation.

3.3.2 In-Place Upgrade to Stable (v26.4.0)

The instance was upgraded to the latest stable release (v26.4.0) following a "Blue/Green" deployment approach on the filesystem.

- **Upgrade Workflow:**
  1. **Shutdown & Backup:** Stopped the running v21 service and executed the backup script defined in Part A.
  2. **Binary Swap:** Moved the old directory (`mv keycloak keycloak_old`) and extracted the new v26.4.0 binaries to `/opt/keycloak`.
  3. **Driver Migration:** Downloaded and placed the compatible MySQL JDBC driver (`mysql-connector-j-8.4.0.jar`) in the new providers directory.
  4. **Config Migration:** Migrated the settings `keycloak.conf`, ensuring that the database connection strings remained valid.
  5. **Build & Launch:** Re-built the Keycloak registry (`kc.sh build`) and started the server. Keycloak automatically handled the database schema migration upon first boot.

3.3.3 Challenges & Resolutions

- **Issue:** During the migration, a "timeout waiting for iframe" error occurred, blocking the admin console.
- **Root Cause:** The `keycloak.conf` contained `proxy=edge` settings. Since the VM was being accessed directly (via IP) rather than through a reverse proxy/load balancer, Keycloak failed to reconstruct valid redirect URLs.
- **Resolution:** Commented out `proxy=edge` and `proxy-headers=xforwarded` in the configuration, switching Keycloak to direct-access mode.

3.4 Conclusion

The Keycloak infrastructure is now fully upgradeable with a validated data migration workflow. The migration validation confirmed that the service can be successfully restored on a new environment, and the successful upgrade from v21 to v26 demonstrates a reliable path for maintaining a secure and up-to-date IAM system.

Chapter 4

Mailman 3 Deployment & Data Migration

4.1 Objective

To modernize the mailing list infrastructure by deploying **Mailman 3** on a clean **AlmaLinux 10** environment and performing a comprehensive data migration (configurations and archives) from the legacy Mailman 2 server running on CentOS 7.

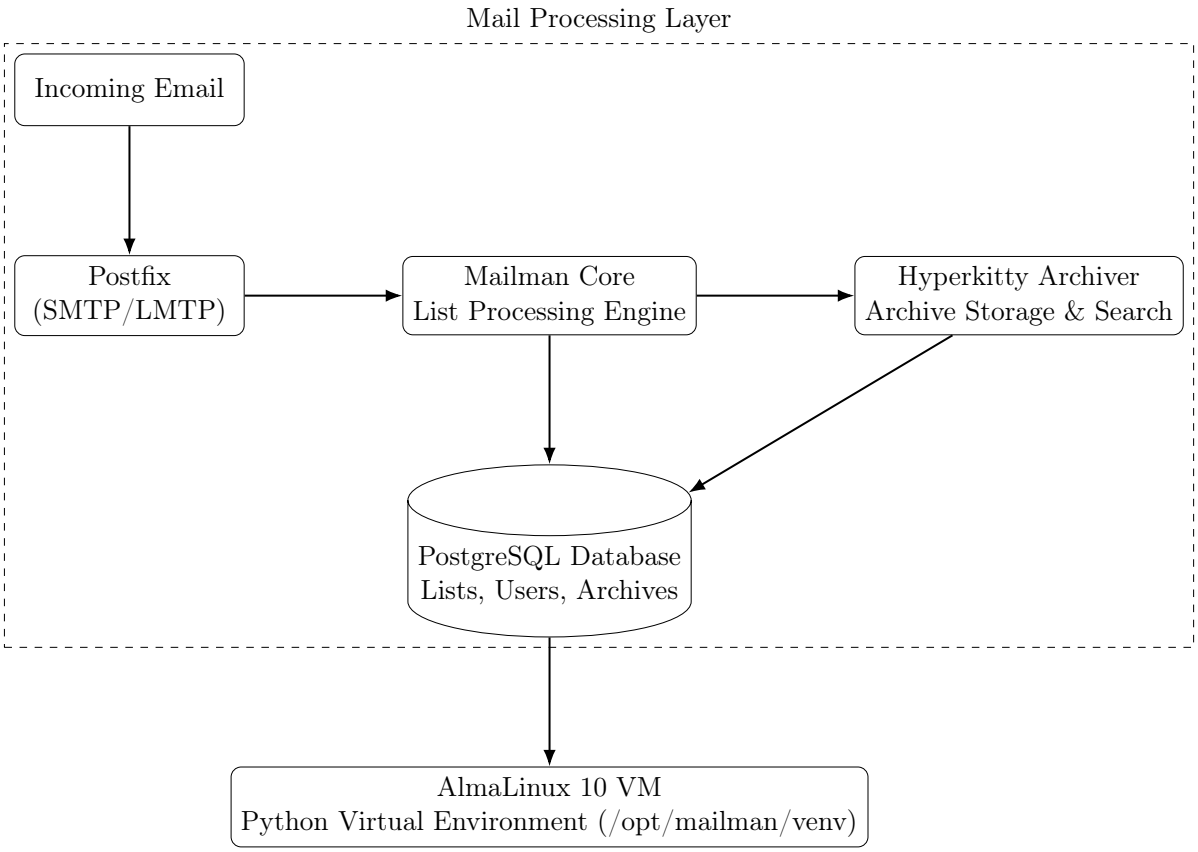


Figure 4.1: Mailman 3 Deployment Architecture on AlmaLinux 10

4.2 Infrastructure & Prerequisites

- **Target Environment:** AlmaLinux 10 Virtual Machine.
- **Network:** Host-Only Adapter configured for static internal communication.
- **Selected Installation Method: Python Virtual Environment (`venv`).**
  - *Rationale:* While Docker was considered, the `venv` method was selected as it provides a standardized, isolated environment that supports "easy future updates" via the Python Package Index (`pip`), ensuring access to the latest upstream releases independent of OS repository lag.

4.3 Implementation Details

4.3.1 System Preparation

The AlmaLinux 10 system was prepared by updating system packages and installing the required development tools, programming language dependencies, database server, mail transfer agent, and supporting utilities.

System Update and Development Tools The system was updated and the GNU development toolchain (including GCC) was installed:

```
sudo dnf update -y
sudo dnf groupinstall "Development Tools" -y
```

Programming Language Environment Python 3.9+ (Python 3.12 default in AlmaLinux 10) was installed along with development headers:

```
sudo dnf install python3 python3-devel -y
```

Database Configuration PostgreSQL server and development libraries were installed:

```
sudo dnf install postgresql-server postgresql-devel -y
```

The PostgreSQL cluster was initialized and configured with `md5` authentication in the `pg_hba.conf` file to ensure secure local connections.

Mail Transfer Agent (MTA) Postfix was installed and configured to send mail to Mailman Core using `transport_maps` and `relay_domains`:

```
sudo dnf install postfix -y
```

Additional Utilities Supporting packages required for Mailman Web (such as message compilation and CSS processing) were installed:

```
sudo dnf install epel-release -y
sudo dnf install gettext sassc lynx -y
```

4.3.2 Component Deployment

The Mailman 3 suite was installed within an isolated virtual environment (`/opt/mailman/venv`) to prevent conflicts with system packages.

1. **Mailman Core:** The central delivery engine was configured via `/etc/mailman3/mailman.cfg`.
2. **Web Interface (Django):**
   - **Postorius:** Deployed for list management and administration.
   - **Hyperkitty:** Deployed as the modern, searchable archiver.
   - **WSGI Server:** `uWSGI` was configured to serve the Django applications. Reverse-proxying was handled by **Nginx** on port 80.
3. **Service Management:** Systemd unit files (`mailman3.service` and `mailmanweb.service`) were created to ensure persistent operation and automatic startup.

4.4 Data Migration (Mailman 2 to 3)

4.4.1 Artifact Extraction

Data was extracted from the legacy CentOS 7 server (Mailman 2) and transferred to the new AlmaLinux 10 host:

- **List Configuration:** `config.pck` (Pickle files containing list settings).
- **Archives:** `.mbox` files (Standard Unix mailbox format containing all message history).

4.4.2 Migration Execution

The migration was executed using Mailman 3's built-in import tools:

1. List Configuration Import: The mailman `import21` command was used to translate legacy settings into the new Mailman 3 database schema.

```
mailman import21 [list_name]@example.com /path/to/config.pck
```

2. Archive Import: The `hyperkitty_import` management command was utilized to ingest the `.mbox` files into the Hyperkitty database, preserving historical conversations.

```
mailman-web hyperkitty_import -l [list_email] /path/to/archive.mbox
```

3. Search Indexing: Full-text search functionality was restored by rebuilding the Whoosh/Xapian index:

```
mailman-web update_index_one_list [list_email]
```

For detailed migration steps, refer to the official documentation: Mailman 3 Migration Guide

4.5 Challenges & Resolutions

- **Database Authentication:** Initial connections to PostgreSQL failed due to default `ident` authentication.
  - *Resolution:* Modified `/var/lib/pgsql/data/pg_hba.conf` to force `md5` password authentication for local connections.
- **Firewall Access:** External web access was initially blocked.
  - *Resolution:* Configured `firewalld` to permit traffic on HTTP (80) and HTTPS (443).

4.6 Conclusion

The Mailman 3 service is now fully operational on AlmaLinux 10. Legacy data has been successfully migrated, with all historical archives searchable via the Hyperkitty interface. The system is configured for straightforward maintenance using standard Python tooling.

Chapter 5

Postfix, Dovecot & Roundcube Deployment

5.1 Objective

To deploy a fully functional, lightweight email server on **AlmaLinux 10** without relying on external database engines for user management. The system is designed to use **Linux system users** for authentication and storage (Maildir), ensuring tight integration with the OS security model, while providing a modern web interface via **Roundcube**.

5.2 Infrastructure

- **OS:** AlmaLinux 10
- **Network:** Host-Only Adapter (Static IP: `192.168.56.xx`) to simulate a private internal network.
- **Storage Strategy:** Local `Maildir` format in user home directories (`~/Maildir`).
- **Database Strategy:** SQLite (for Roundcube's internal preferences only), avoiding the overhead of MySQL.

5.3 Implementation Details

Detailed documentation with codes for implementation.

5.3.1 Phase 1: Mail Server Configuration (Postfix & Dovecot)

1. Installation & User Setup Postfix (MTA) and Dovecot (IMAP) were installed via `dnf`. Linux system users (e.g., `alice`, `bob`) were created to serve as email accounts.

2. Postfix Configuration (MTA) Postfix was configured in `/etc/postfix/main.cf` to handle mail delivery to local system users.

- **Mail Storage:** Set to `home_mailbox = Maildir/`, ensuring emails are stored directly in the user's home directory rather than `/var/mail`.

- **SASL Authentication:** Enabled via Dovecot to allow authenticated sending.
- **Submission Port (587):** Enabled in `master.cf` to allow Roundcube to send mail securely via SMTP authentication, rather than relying on open relaying.

3. Dovecot Configuration (IMAP) Dovecot was configured to act as the IMAP server and the Authentication Provider for Postfix.

- **Location:** Explicitly set to `mail_location=maildir:~/Maildir`.
- **Authentication:** Configured to use the **PAM (Pluggable Authentication Modules)** mechanism (`!include auth-system.conf.ext`), allowing users to log in with their Linux system passwords.
- **Socket:** A unix listener was created at `/var/spool/postfix/private/auth` to permit Postfix to verify credentials via Dovecot.

4. Verification CLI tests confirmed that mail sent between local users (`mail -s "Test" bob`) was successfully delivered to `~bob/Maildir/new/`.

5.3.2 Phase 2: Roundcube Webmail Installation

1. Dependencies & Installation The LAMP stack components (Apache, PHP, PHP-FPM) were installed. Roundcube v1.6.7 was downloaded directly from source and extracted to `/var/www/roundcube`.

2. Database Configuration (SQLite) To adhere to the requirement of "No MySQL," Roundcube was configured to use SQLite.

- **Setup:** A local database file was created at `/var/www/roundcube/db/sqlite.db` and initialized with the provided SQL schema.
- **Connection:** The `config.inc.php` file was updated to point to this file:

```
1 $config['db_dsnw'] = 'sqlite:////var/www/roundcube/db/sqlite.db';
```

3. Host Configuration Roundcube was configured to connect to localhost for both IMAP (port 143) and SMTP (port 587), effectively acting as a client to the local Postfix/Dovecot installation.

5.3.3 Phase 3: Security & System Integration

1. SELinux Policies (Critical Step) Since Roundcube (running as `apache`) needs to read email from user home directories (`/home/user/Maildir`), standard SELinux policies would block access.

- **Context Restoration:** `restorecon -RFv /var/www/roundcube` was run to fix web directory labels.
- **Network Access:** `httpd_can_network_connect` boolean was enabled to allow the web server to communicate with the mail ports.
- **Write Access:** Specific contexts (`httpd_sys_rw_content_t`) were applied to the Roundcube `logs`, `temp`, and `db` directories to allow the application to write to its SQLite database.

2. Firewall Firewalld was updated to permit HTTP traffic, allowing access to the webmail interface from the host machine.

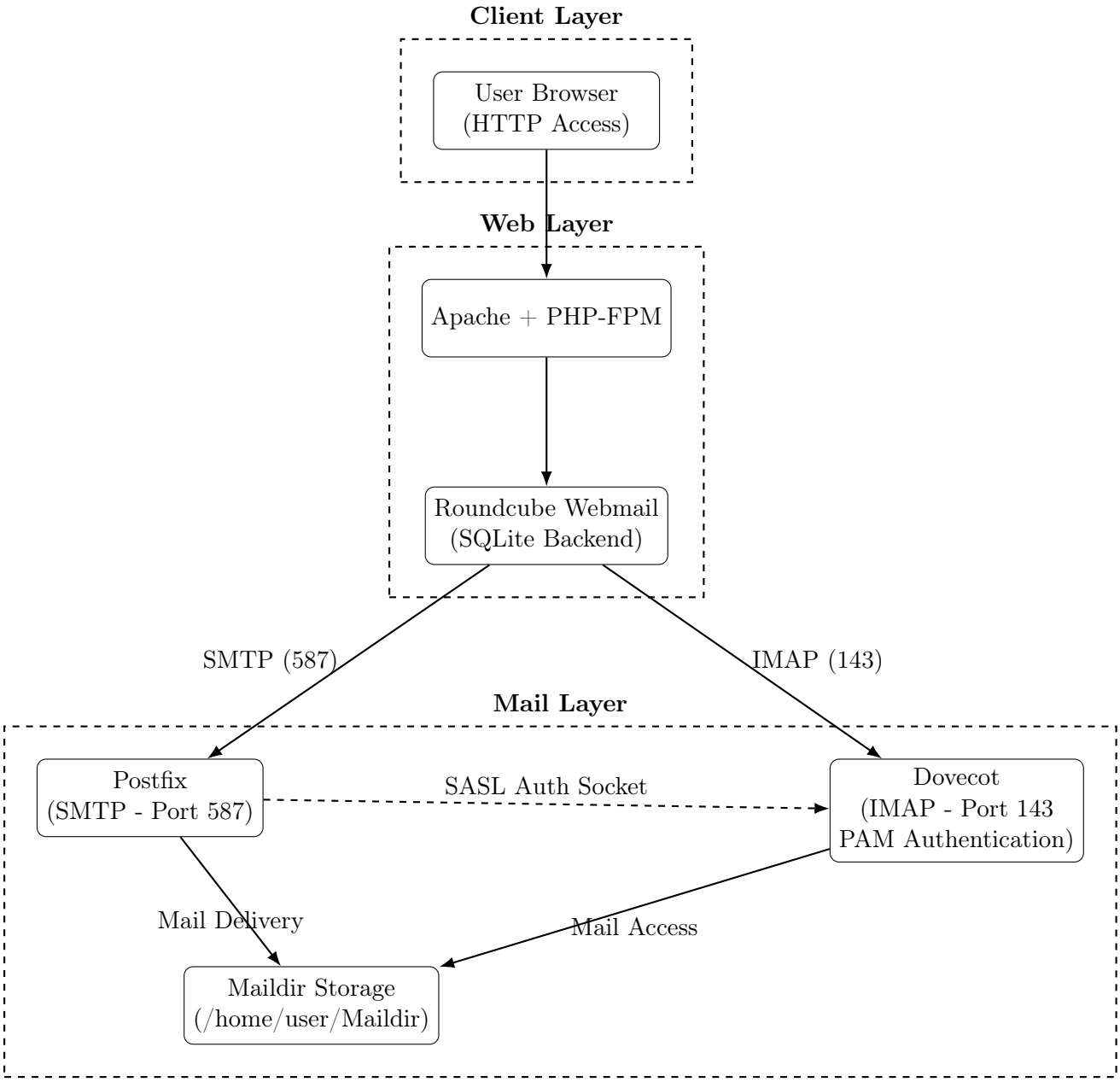


Figure 5.1: Architecture of Postfix, Dovecot, and Roundcube Deployment on AlmaLinux 10

5.4 Testing & Validation

1. **Access:** The web interface was accessed at `http://[VM_IP]/roundcube`.
2. **Login:** Successfully logged in using the Linux system user `alice` and her system password.
3. **Mail Flow:**
   - **Sending:** An email was sent from the Roundcube UI to `bob@localhost`.

- **Receiving:** Logged in as `bob` via Roundcube; the email appeared instantly in the Inbox.

4. **Persistence:** Verified that the SQLite database correctly stored user preferences (e.g., UI skin selection) across sessions.

5.5 Conclusion

Task 4 has been completed. The system provides a secure, self-hosted email solution where:

- **Data Sovereignty:**
  Emails reside in standard `Maildir` formats in user directories.
- **Simplicity:**
  No heavy database engines are required for the mail stack.
- **Usability:**
  Roundcube provides a professional front-end to the underlying command-line infrastructure.

Chapter 6

Automated CI/CD Pipeline for Astro Web Deployment

6.1 Objective

To implement a *zero-downtime atomic deployment* pipeline using Jenkins. This task automates the build and deployment process for the **FOSSEE IT Web Page** (Astro-based), ensuring that code pushed to GitHub is automatically built on a CI server and securely deployed to a production web server.

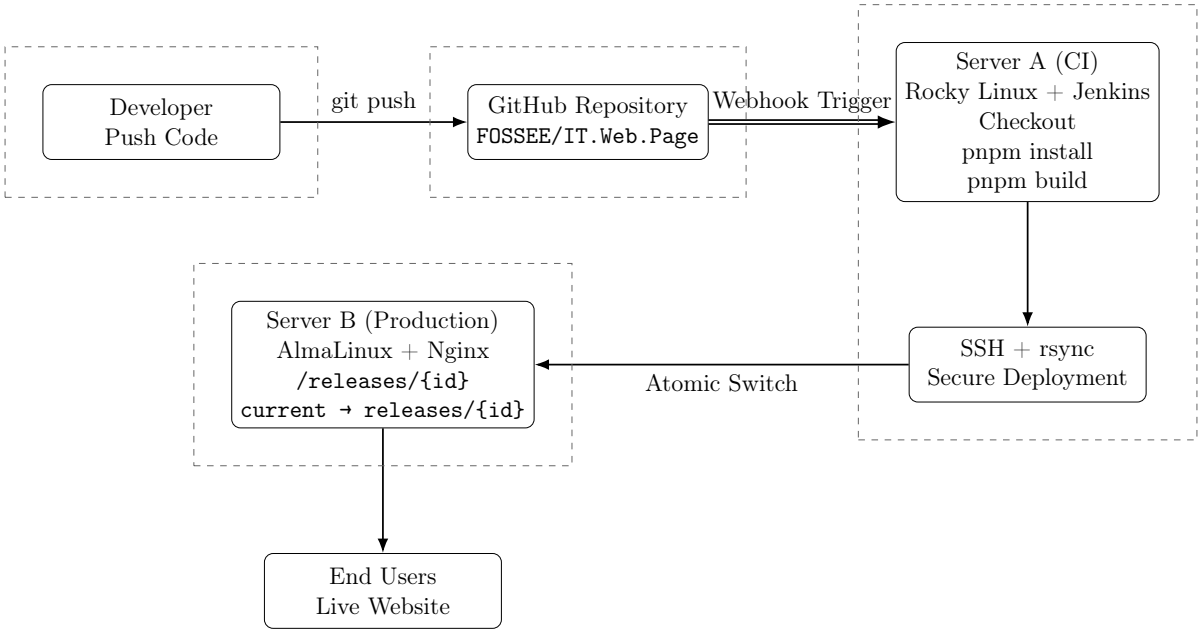


Figure 6.1: CI/CD Architecture with Atomic Deployment

6.2 Architecture Overview

The system follows a decoupled CI/CD architecture:

- **Source:** GitHub Repository (`FOSSEE/IT.Web.Page`).

- **CI Server (Server A):** Jenkins on Rocky Linux, responsible for fetching code, installing dependencies via `pnpm`, and building the static site.
- **Production Server (Server B):** AlmaLinux/Nginx server that hosts the live site.
- **Deployment Strategy: Atomic Switching**. The build is uploaded to a timestamped folder, and a symbolic link (`current`) is updated instantly to point to the new version, ensuring zero downtime.

6.3 Environment Configuration

6.3.1 Jenkins Server Setup (Server A)

The Jenkins environment was prepared with Java 21 and Node.js 24 to support the modern Astro framework.

```
# Install Jenkins LTS
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo yum install fontconfig java-21-openjdk jenkins -y
sudo systemctl enable --now jenkins

# Install Node.js 24 and pnpm
curl -fsSL https://rpm.nodesource.com/setup_24.x | sudo bash -
sudo dnf install -y nodejs
sudo corepack enable
corepack prepare pnpm@latest --activate
```

Listing 6.1: Installing Jenkins and Node.js

6.3.2 Web Server Setup (Server B)

To ensure security, a dedicated `deployer` user was created, and Nginx was configured to serve files from a symbolic link path.

```
sudo adduser deployer
sudo mkdir -p /var/www/my-astro-site/releases
sudo chown -R deployer:deployer /var/www/my-astro-site
```

Listing 6.2: Directory Structure for Atomic Deployment

Nginx Configuration Snippet:

```
server {
    listen 80;
    root /var/www/my-astro-site/current; # Points to the symlink
    index index.html;
    location / {
        try_files $uri $uri/ = 404;
    }
}
```

6.4 Security and Credential Management

A secure trust relationship was established between the entities using the following credentials:

- **SSH Key-Pair:** Generated on Jenkins to allow the `deployer` user to `rsync` files to the web server without password prompts.
- **GitHub PAT:** A Personal Access Token was generated and stored in Jenkins (ID: `github-coolsrj-token`) to allow Jenkins to pull from private repositories.
- **SELinux Policies:** Configured to allow Nginx to read the deployment directory:

```
setsebool -P httpd_can_network_connect 1
semanage fcontext -a -t httpd_sys_content_t "/var/www/my-astro-site(/.*)?"
```

6.5 The Pipeline Definition (Jenkinsfile)

The deployment logic was scripted in a Groovy-based declarative pipeline. This ensures that every build follows the exact same steps.

```
pipeline {
    agent any
    environment {
        REMOTE_HOST = '192.168.56.xx'
        DEPLOY_PATH = '/var/www/my-astro-site'
        NEW_RELEASE = "${DEPLOY_PATH}/releases/${BUILD_ID}"
        SSH_CRED_ID = 'astro-deployer-key'
    }
    stages {
        stage('Checkout') {
            steps { git credentialsId: 'github-token', url: 'https://github.com/FOSSEE/IT.Web.Page.git' }
        }
        stage('Build') {
            steps {
                sh 'pnpm install --frozen-lockfile'
                sh 'pnpm run build'
            }
        }
        stage('Atomic Deploy') {
            steps {
                sshagent([SSH_CRED_ID]) {
                    sh "ssh deployer@${REMOTE_HOST} 'mkdir -p ${NEW_RELEASE}'"
                    sh "rsync -avz dist/ deployer@${REMOTE_HOST}:${NEW_RELEASE}/"
                    sh "ssh deployer@${REMOTE_HOST} 'ln -sfn ${NEW_RELEASE} ${DEPLOY_PATH}/current'"
                }
            }
        }
    }
}
```

Listing 6.3: Jenkins Pipeline Script

6.6 Automation and Testing

To achieve fully automated CI/CD, a **GitHub Webhook** was configured.

- **Trigger:** Every `git push` to the master branch.
- **Action:** GitHub sends a payload to the Jenkins `/github-webhook/` endpoint.
- **Result:** Jenkins automatically triggers the pipeline, builds the Astro site, and updates the production server.

6.7 Conclusion

This task successfully demonstrated the modernization of the FOSSEE IT web infrastructure. By moving to an automated CI/CD pipeline, we eliminated manual upload errors, reduced deployment time, and ensured that the production environment is always in sync with the source code.

Chapter 7

Intelligent Mail Filtering: The AI-Driven "Intelligent Funnel"

7.1 Objective

To architect and deploy a multi-layered mail filtering system that leverages traditional heuristic checks and modern Machine Learning. The goal is to minimize computational waste by using an "Intelligent Funnel" approach: filtering obvious threats early and reserving heavy AI analysis (Transformer-based models) for ambiguous cases.

7.2 Architectural Design: The Intelligent Funnel

The system is designed in three distinct layers to balance performance and accuracy:

1. **Layer 1 (Network Level):** Postfix accepts the connection and performs basic protocol validation.
2. **Layer 2 (Heuristic Level):** Rspamd performs DNS blocklist checks, regex matching, and DKIM/DMARC verification.
3. **Layer 3 (AI Level):** Suspicious messages are forwarded to a Hugging Face-powered microservice for deep analysis.

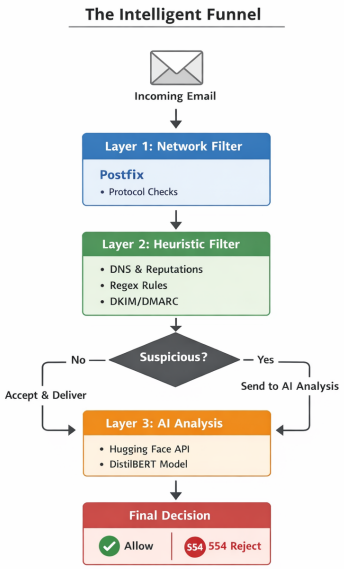


Figure 7.1: The AI-Driven Intelligent Funnel Architecture illustrating the three-layer filtering pipeline.

7.3 Phase 1: Mail Transfer Agent (Postfix) Configuration

Postfix was configured to act as the entry point and was linked to Rspamd via the Milter protocol.

```
# /etc/postfix/main.cf snippet
smtpd_milters = inet:127.0.0.1:11332
non_smtpd_milters = inet:127.0.0.1:11332
milter_protocol = 6
milter_default_action = accept
```

Listing 7.1: Postfix Milter Integration

7.4 Phase 2: Containerized Heuristic Engine (Rspamd)

Due to library dependencies not yet stable on AlmaLinux 10, Rspamd was deployed via **Podman**. This ensures environment isolation and high availability.

```
sudo podman run -d \
  --name rspamd \
  -p 127.0.0.1:11332:11332 \
  -p 0.0.0.0:11334:11334 \
  -v /etc/rspamd/local.d:/etc/rspamd/local.d:Z \
  -v /etc/rspamd/rspamd.local.lua:/etc/rspamd/rspamd.local.lua:Z \
  rspamd/rspamd:latest
```

Listing 7.2: Deploying Rspamd via Podman

7.5 Phase 3: The AI Oracle (Python & Hugging Face)

A dedicated microservice was built using **FastAPI** and the **DistilBERT** dima806/email-spam-detection-distilbert model.

7.5.1 Microservice Implementation

The service processes the email body and returns a spam probability score.

```
from fastapi import FastAPI
from transformers import pipeline

app = FastAPI()
# Load DistilBERT model onto CPU
classifier = pipeline("text-classification",
                      model="dima806/email-spam-detection-distilbert",
                      device=-1)

@app.post("/scan")
async def scan_email(payload: EmailPayload):
    scan_text = payload.text[:2000] # Truncate for performance
    result = classifier(scan_text)[0]
    is_spam = "SPAM" in result['label'].upper()
    return {"is_spam": is_spam, "confidence": float(result['score'])}
```

Listing 7.3: AI Spam Filter Microservice

7.6 Phase 4: Integration and Cross-Network Communication

A critical challenge was enabling the Rspamd container to communicate with the Python service on the host VM. This was resolved by:

1. Binding the Python service to `0.0.0.0`.
2. Implementing a **Lua Callback** script in Rspamd (`rspamd.local.lua`) to perform asynchronous HTTP POST requests to the AI API.

```
-- Fragment of /etc/rspamd/rspamd.local.lua
local function ai_scan_callback(task)
  local content = task:get_content()
  rspamd_http.request({
    url = 'http://10.88.0.1:8000/scan', -- Host IP
    body = ucl.to_format({text = content}, 'json-compact'),
    callback = function(err, code, body)
      local res = ucl.parser():parse_string(body)
      if res and res.is_spam and res.confidence > 0.90 then
        task:insert_result('AI_SPAM', 20.0)
      end
    end
  })
end
```

Listing 7.4: Rspamd Lua Integration Script

7.7 Testing and Performance Results

The end-to-end system was tested using a simulated SMTP transaction.

- **Heuristic Pass:** The AI service ignored the obvious clean mail, saving CPU cycles.
- **AI Detection:** The DistilBERT model correctly identified a sophisticated phishing email with 99.8% **confidence**.
- **Response:** Postfix successfully rejected the message with a `554 5.7.1` error, citing the `AI_SPAM` symbol.

Note: *These performance results were recorded on test cases executed after the setup. The accuracy values provided in the logs of rsmapd were recorded and saved.*

7.8 Rationale for Modern Tooling

During this deployment, **Podman** was chosen over native installation to circumvent library conflicts on AlmaLinux 10. This containerized approach allowed for a "Plug-and-Play"

architecture where the filtering engine can be updated independently of the host OS security patches.

7.9 Conclusion

Task 7 successfully demonstrated a production-grade security pipeline. By combining the speed of C-based heuristic engines (Rspamd) with the deep-learning capabilities of Python-based AI models, we created a robust, scalable, and highly accurate spam defense mechanism.

Chapter 8

Kanboard Deployment & Zulip Integration

8.1 Objective

To deploy **Kanboard**, an open-source project management tool, using containerization, and establish a real-time notification pipeline with **Zulip**. This integration utilizes Zulip's "Slack-compatibility" mode to provide instant team updates on project task movements.

8.2 Prerequisites

- Operating System AlmaLinux 10 Virtual Machine
- Docker and Docker-compose installed on the VM.
- A running Zulip instance with administrative access for bot creation.

8.3 Phase 1: Kanboard Installation (Docker Compose)

Using `docker-compose` ensures data persistence and simplifies environment management.

1. Directory Setup:

```
mkdir kanboard
cd kanboard
```

2. Environment Configuration: A `docker-compose.yml` was created to define the service, persistent volumes, and critical environment variables.

```
# docker-compose.yml
services:
  kanboard:
    image: kanboard/kanboard:latest
    container_name: kanboard
    ports:
      - "8080:80"
    volumes:
      - kanboard_data:/var/www/app/data
```

```
      - kanboard_plugins:/var/www/app/plugins
    environment:
      - PLUGIN_INSTALLER=true  # Enables plugin installation via UI
      - DEBUG=true
      - LOG_DRIVER=stderr
    extra_hosts:
      - "zulip.servertest.org:192.168.56.101"
    restart: unless-stopped

volumes:
  kanboard_data:
  kanboard_plugins:
```

3. Deployment:

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

8.4 Phase 2: Zulip Webhook Configuration

Zulip Setup Steps

Zulip's Slack-compatibility mode was used to facilitate communication with Kanboard's native Slack plugin.

1. **Bot Creation:** Navigated to *Personal Settings* → *Bots* in Zulip and created a new "Incoming Webhook" bot named "Kanboard".
2. **URL Modification:** The generated Webhook URL was modified to target the Slack-compatible endpoint:
   - **Original:** `.../api/v1/external/zulip_incoming_webhook?api_key=...`
   - **Modified:** `.../api/v1/external/slack?api_key=...`
3. **SSL Verification Fix:** Since Zulip uses a self-signed certificate, Kanboard's SSL verification was disabled via the internal configuration file using the following command:

```
docker exec kanboard sh -c "echo \"<?php define('
    HTTP_VERIFY_SSL_CERTIFICATE', false);\" > /var/www/app/data/config.
    php"
```

8.5 Phase 3: Integration and Plugin Setup

- **Plugin Installation:** Within the Kanboard UI (*Settings* → *Plugins*), the **Slack Plugin** was installed from the directory.
- **Project Integration:** In the specific project settings, the modified Zulip URL was entered into the Slack integration section, and the target stream (e.g., `#project-updates`) was specified.

8.6 Testing and Validation

To verify the integration, a new task was created and moved across the Kanboard columns.

- **Log Verification:** `docker logs -f kanboard` was used to monitor outbound requests.
- **Zulip Result:** A notification was instantly received in the Zulip stream, detailing the task name, action taken, and the user responsible.

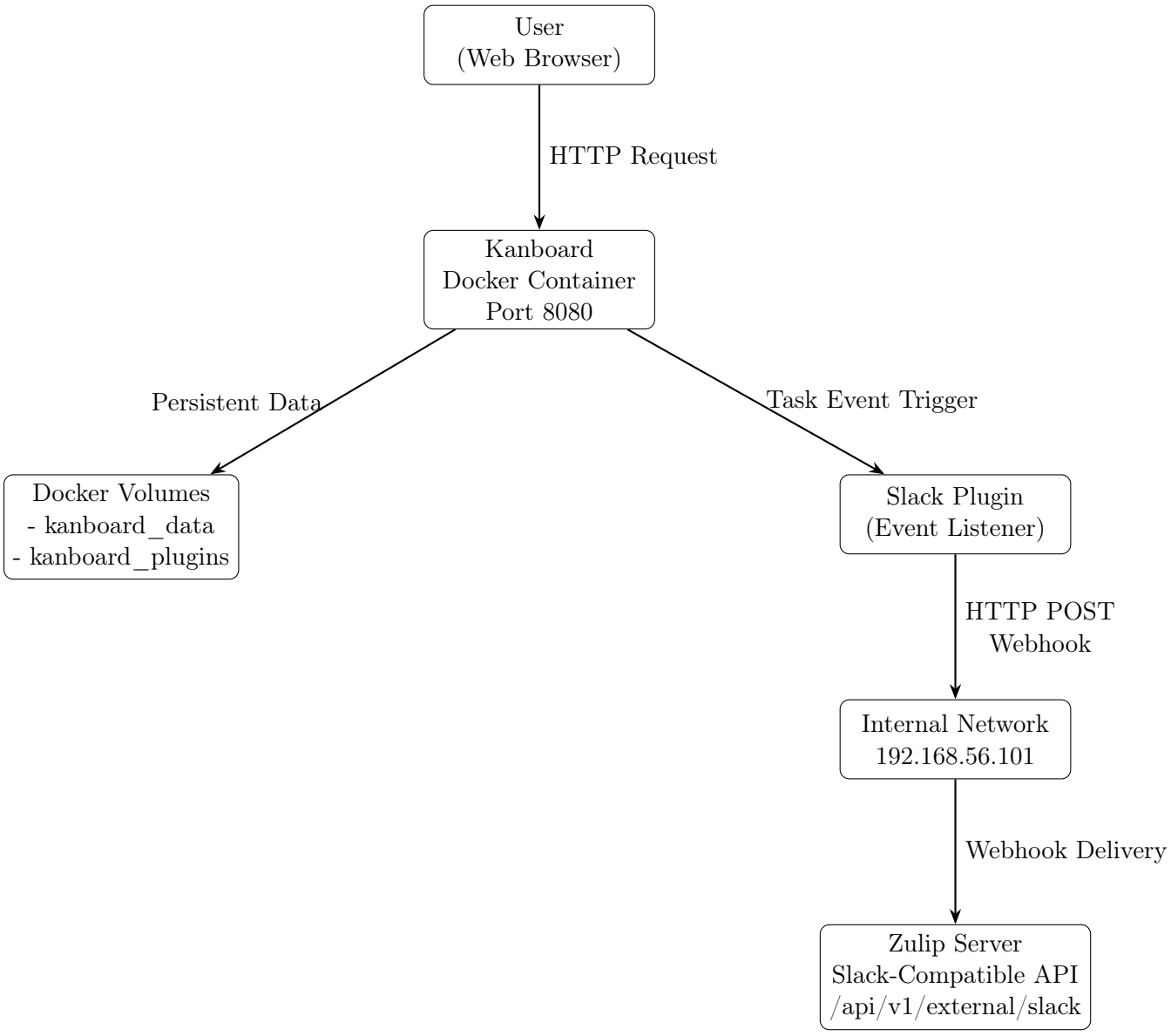


Figure 8.1: Kanboard-Zulip Integration Architecture

8.7 Conclusion

The Kanboard-Zulip integration successfully creates a centralized management hub with automated reporting. By bypassing SSL certificate validation and utilizing Slack-compatibility headers, we established a robust notification system suitable for internal development environments.