Politehnica University of Bucharest

RabbitMQ: Technical Report 1

Name: Alexandru – Florin Ionescu

Date: 06/04/2025

Table of Contents

1. Introduction

Chosen Framework/Platform: RabbitMQ

Rationale for Selection:

I chose RabbitMQ as the distributed messaging platform for this project because it's deployed in products of many large companies, such as Reddit or Slack. Moreover, it's flexible, working with different protocols and languages, like HTTP/AMQP/MQTT. Finally, its fault tolerance means that, if a server were to crash, the messages would be saved.

Intended Use Case:

I'm building a system where different services need to talk to each other without being tightly connected. RabbitMQ will be our middleman, making sure messages get where they need to go, even if some parts of the system are busy or down.

Specific Application Scenario:

In this project, RabbitMQ will be utilized to build a notification system for a distributed microservices architecture. Each microservice will publish and subscribe to event notifications, ensuring real-time updates and asynchronous communication. This design aims to improve system reliability and scalability.

2. Environment Setup

2.1. Virtualization Technology

- **Host Machine:** Windows 11 Pro
- **Hypervisor:** Oracle VirtualBox 7.1.6
- **Virtualization Type:** Full virtualization with bridged networking for external communication

2.2. Operating System Configuration

- **Base Image:** Ubuntu Server 24.04.2 LTS (ubuntu-24.04.2-live-server-amd64.iso)
- **Nodes:** 3 identical VMs (rabbitmq-node1, rabbitmq-node2, rabbitmq-node3)
- **Common Configuration:**
    - Minimal installation (OpenSSH server only)
    - Unified user account (rabbitmq) across all nodes
    - Identical network and security policies

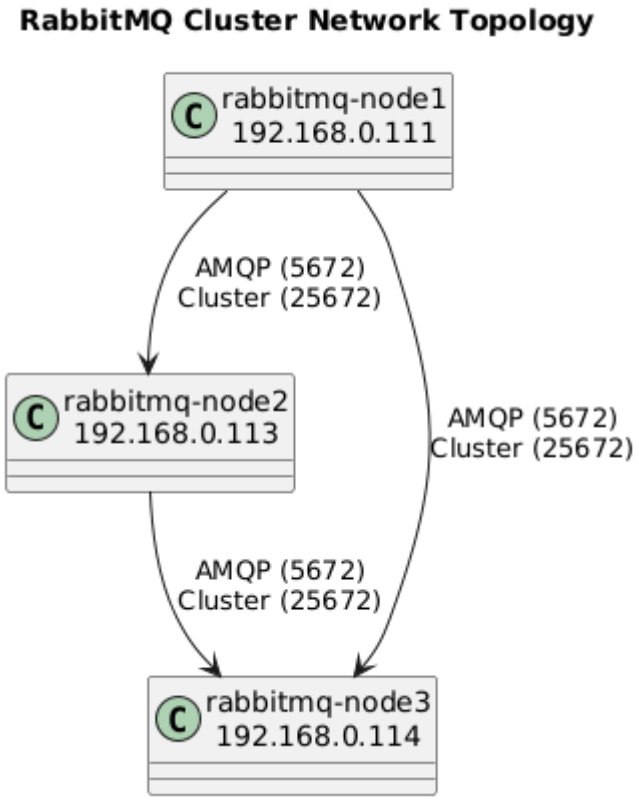
2.3. Network Configuration

- **Network Mode:** Bridged networking
- **IP Allocation:**
    - **rabbitmq-node1:** 192.168.0.111
    - **rabbitmq-node2:** 192.168.0.113
    - **rabbitmq-node3:** 192.168.0.114
- **Firewall Rules**
    - **5672/tcp** (AMQP)
    - **15672/tcp** (Management UI)
    - **25672/tcp** (Cluster communication)

2.4. Virtual Machine Specifications:

Node	vCPUs	RAM	Storage	Network Adapter
rabbitmq-node1	2	4GB	40GB	Bridged (virtio-net)
rabbitmq-node2	2	4GB	40GB	Bridged (virtio-net)
rabbitmq-node3	2	4GB	40GB	Bridged (virtio-net)

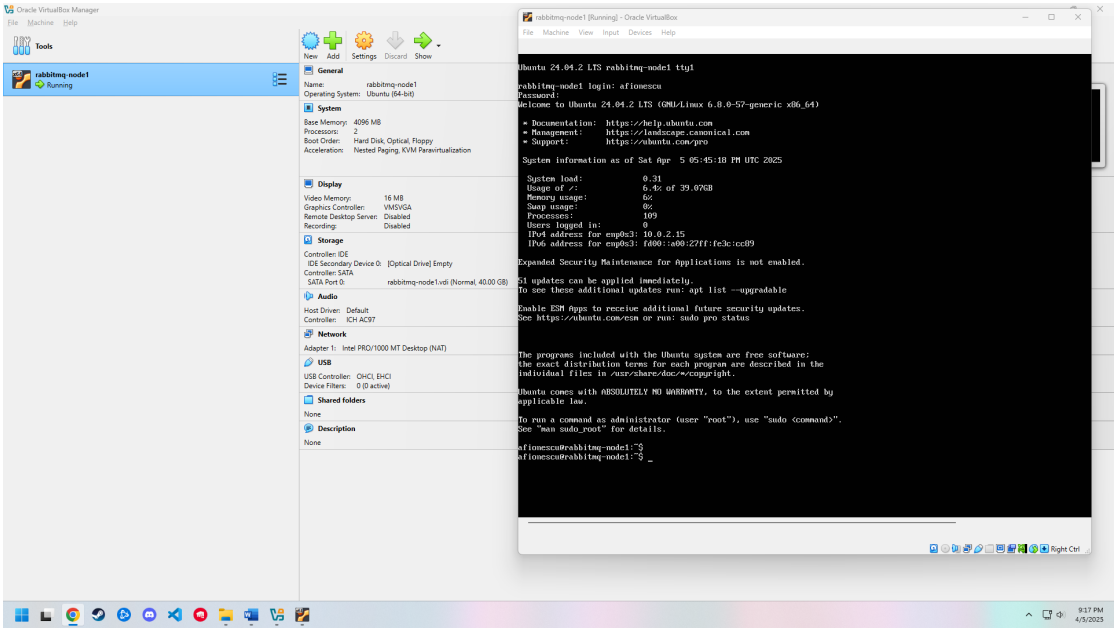
2.5. Network Topology Diagram



3. Installation and Functional Verification

3.1. Setting Up VirtualBox

- Downloaded **VirtualBox 7.1.6** from the official website and installed it on my Windows 11 Pro machine.
- I installed **OpenSSH** to connect to all VMs from my machine
- Created the first VM, **rabbitmq-node1**, with:
    - **Ubuntu Server 24.04 LTS ISO** (attached via *Settings* > *Storage*)
    - **4GB RAM, 2 vCPUs, 40GB** dynamically allocated storage
    - **Bridged Networking** (critical for inter-VM communication)



3.2. Cloning Additional Nodes

- Cloned **rabbitmq-node1** twice to create:
    - **rabbitmq-node2**
    - **rabbitmq-node3**
- Key Clone Settings Used:
    - **Clone Type:** *Full Clone* (to ensure independence)

- **MAC Address Policy:** *Generate new MAC addresses for all network adapters* (avoids IP conflicts)

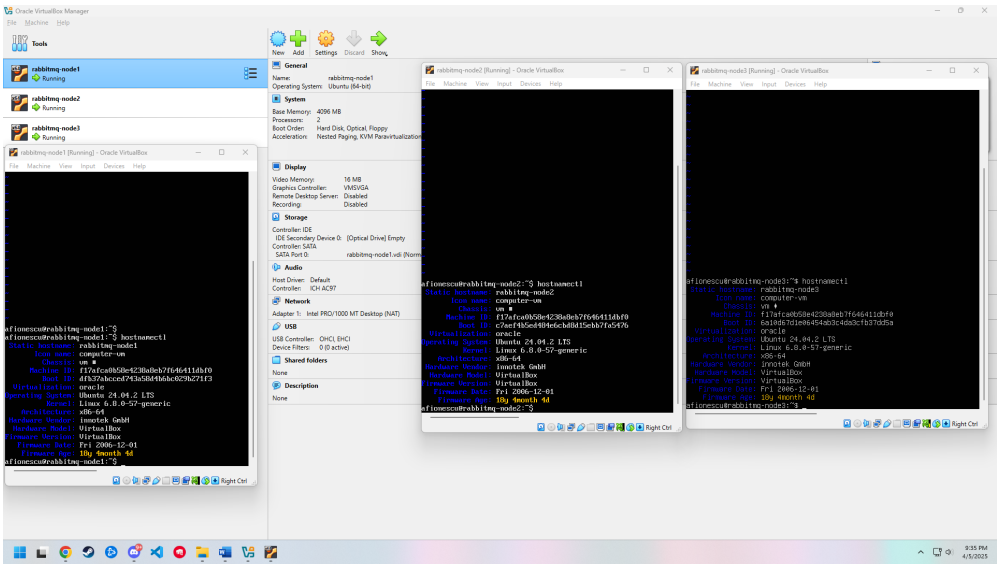
- Change hostnames inside each VM:

rabbitmq-node2:

sudo hostnamectl set-hostname rabbitmq-node2

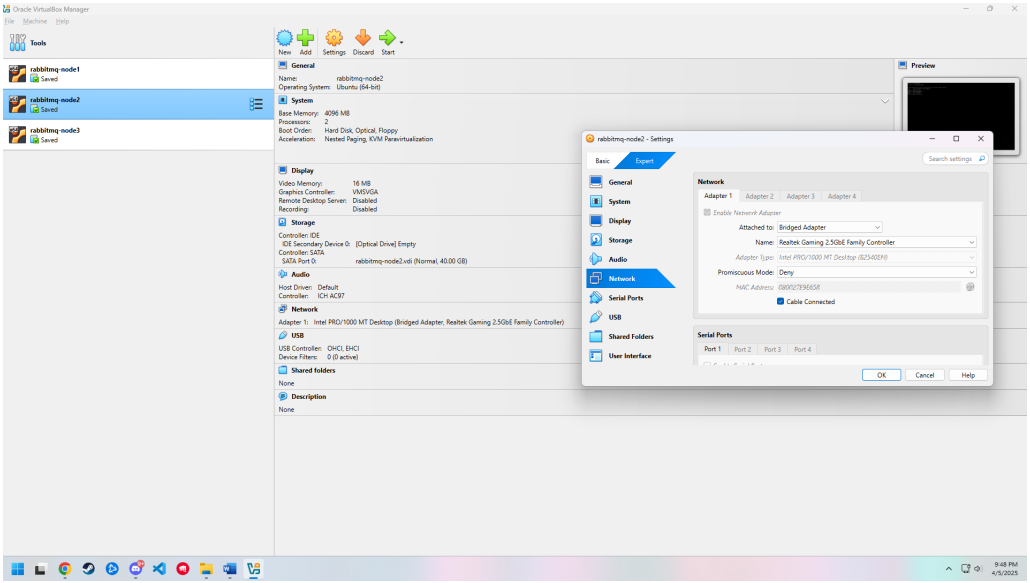
rabbitmq-node3:

sudo hostnamectl set-hostname rabbitmq-node3

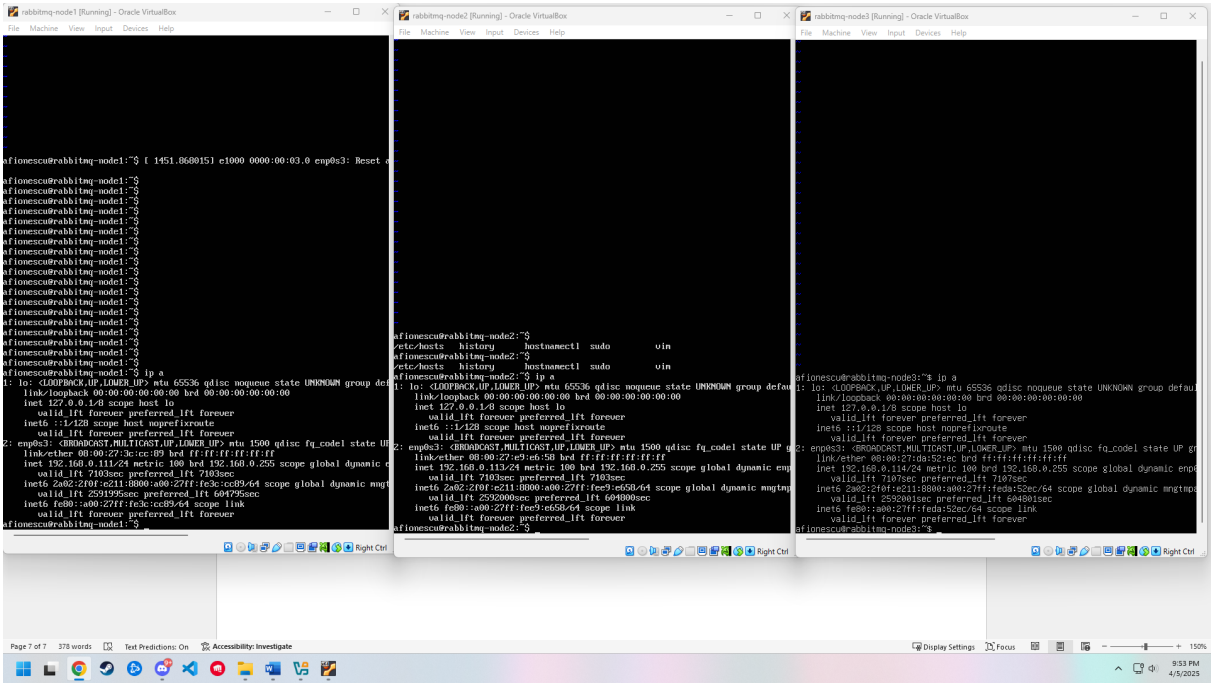


3.3. Fixing the Network

- Switched to Bridged Networking so that the VMs act like real computers



To find the IP Address I ran into each VM “*ip a*”

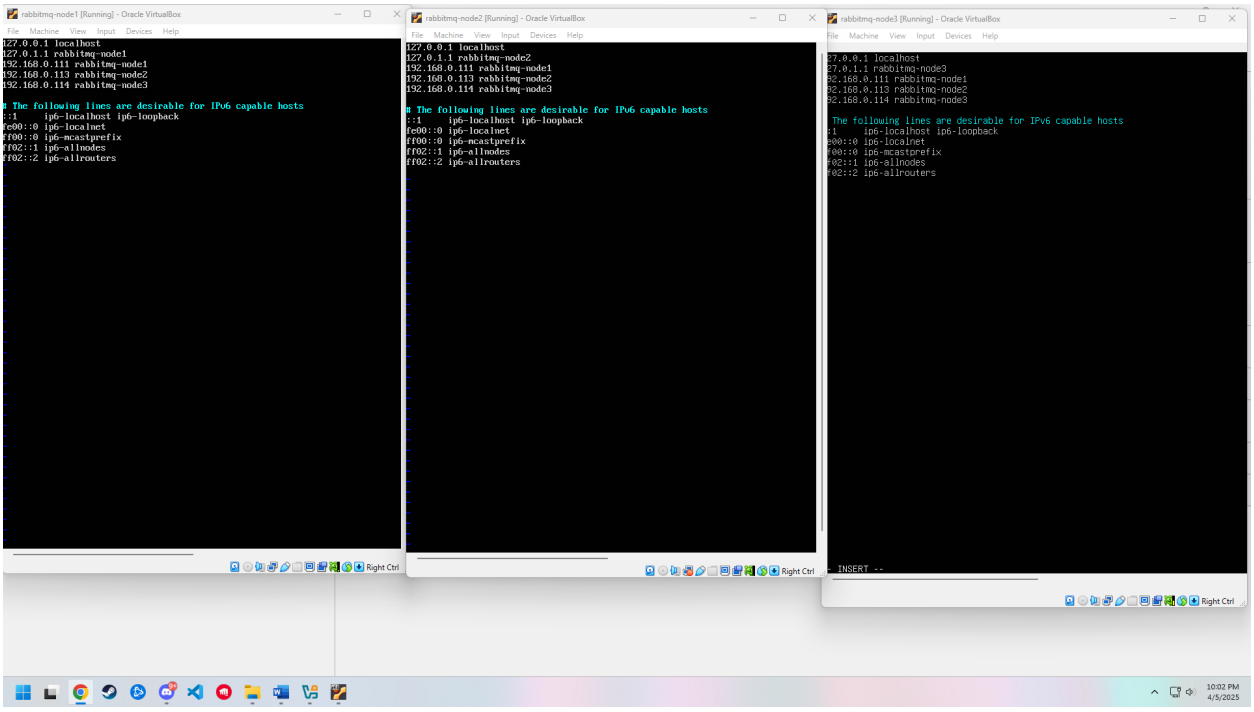


- I updated */etc/hosts* on all VMs with the ips for the nodes:

192.168.0.111 rabbitmq-node1

192.168.0.113 rabbitmq-node2

192.168.0.114 rabbitmq-node3



3.4. Installing RabbitMQ

- First I added RabbitMQ signing key and official repository :

sudo apt-get update && sudo apt-get install -y curl gnupg apt-transport-https

curl -1sLf "https://keys.openpgp.org/vks/v1/by-fingerprint/0A9AF2115F4687BD29803A206B73A36E6026DFCA" | sudo gpg --dearmor | sudo tee /usr/share/keyrings/rabbitmq.gpg > /dev/null

echo "deb [signed-by=/usr/share/keyrings/rabbitmq.gpg] https://packagecloud.io/rabbitmq/rabbitmq-server/ubuntu noble main" | sudo tee /etc/apt/sources.list.d/rabbitmq.list

To install RabbitMQ I ran:

sudo apt update && sudo apt install -y rabbitmq-server

- I started it and I enabled auto-start on boot :

sudo systemctl enable --now rabbitmq-server

3.5. Initial Verification Steps

- I checked the service status:

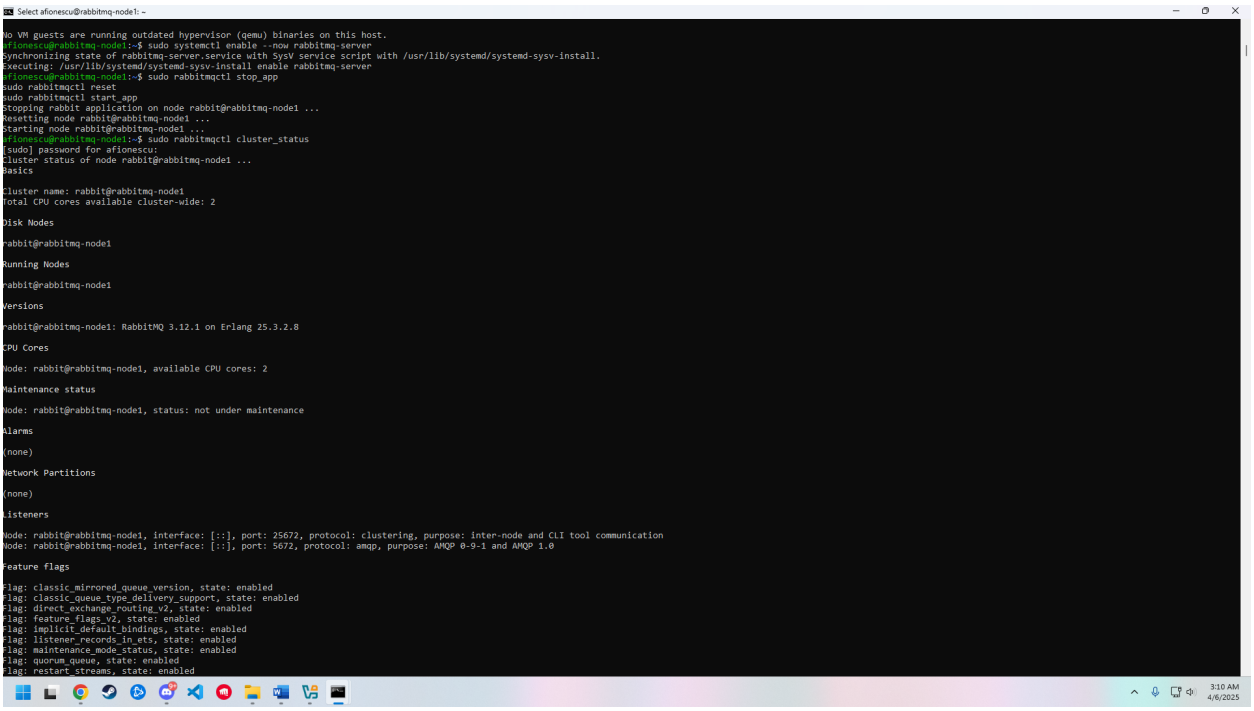
sudo systemctl status rabbitmq-server

- I reset the application state:

sudo rabbitmqctl stop_app

sudo rabbitmqctl reset

sudo rabbitmqctl start_app



3.6. Cluster Configuration Process

- I synchronized the cookie for them to communicate with each other (I copied cookie value from rabbitmq-node1 to to the other nodes) and I had to add writing mode to the cookie file

sudo cat /var/lib/rabbitmq/.erlang.cookie

sudo chown rabbitmq:rabbitmq /var/lib/rabbitmq/.erlang.cookie

sudo chmod 400 /var/lib/rabbitmq/.erlang.cookie

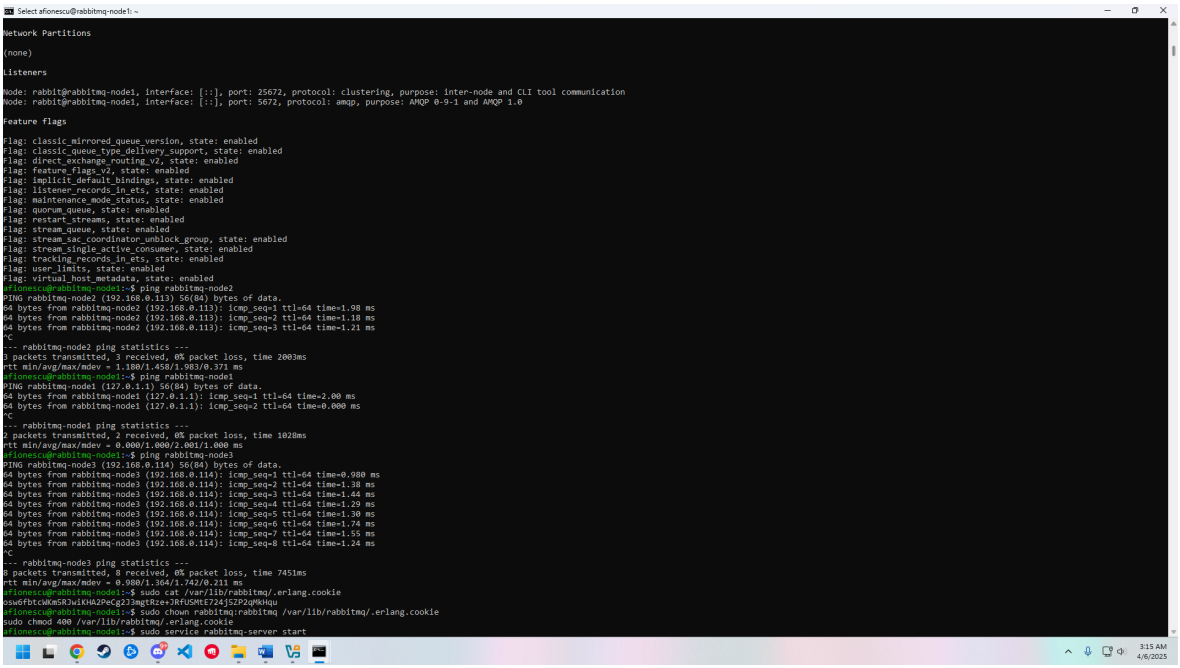
- I tested the connectivity between nodes:

ping rabbitmq-node2 # Successful

ping rabbitmq-node3 # Successful

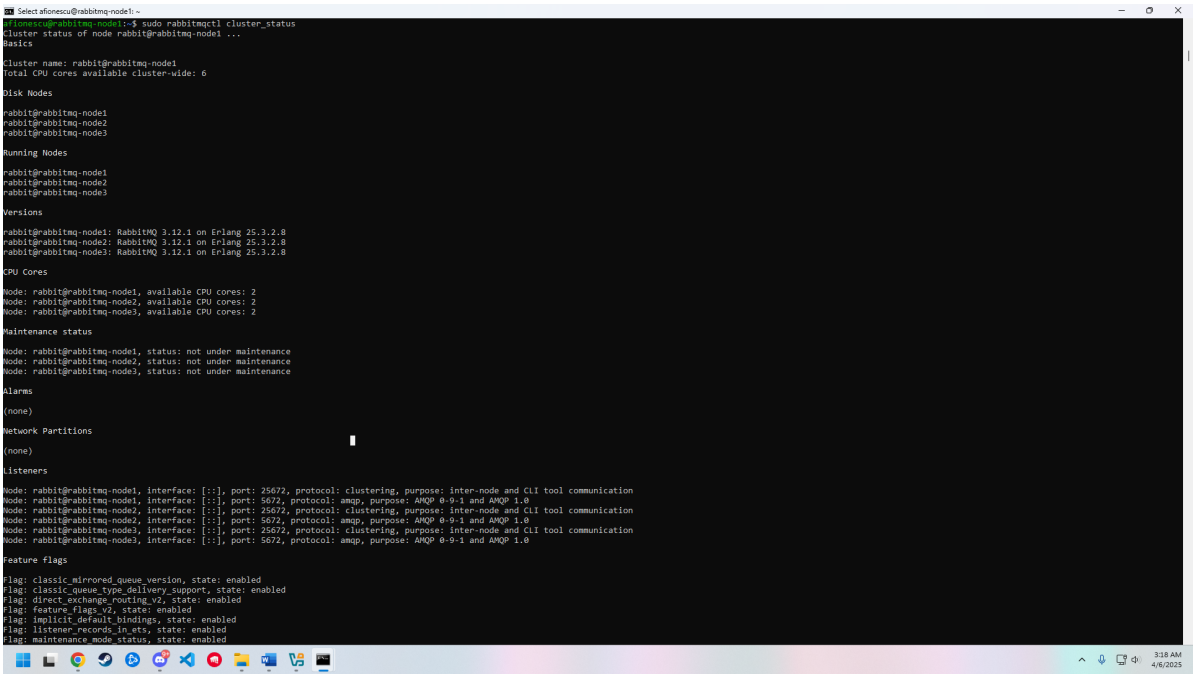
telnet 192.168.0.113 5672 # Connected

telnet 192.168.0.114 5672 # Connected



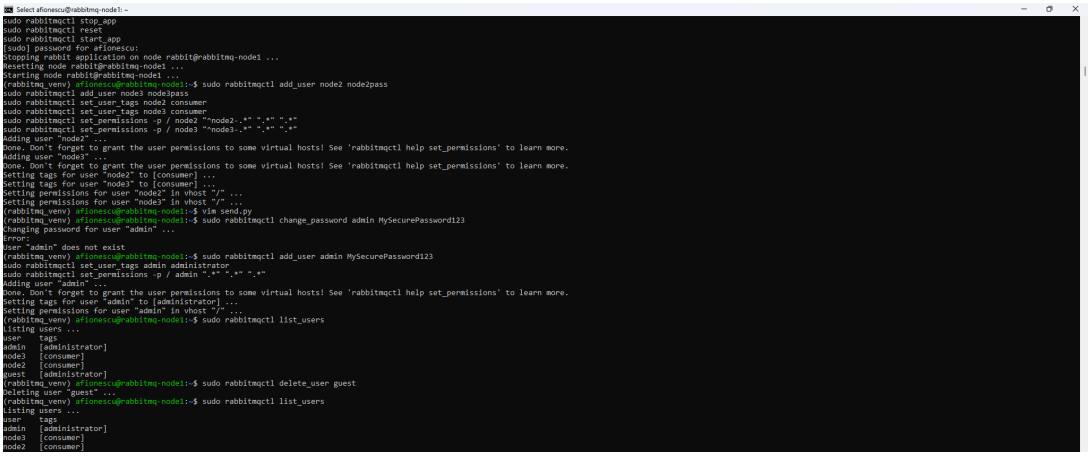
- I checked the cluster status and the output shows that all three nodes joined together:

sudo rabbitmqctl cluster_status



3.7. User Management Configuration

- I created and set up admin user and node-specific users for **rabbitmq-node2** and **rabbitmq-node3** and I've deleted the **guest** user



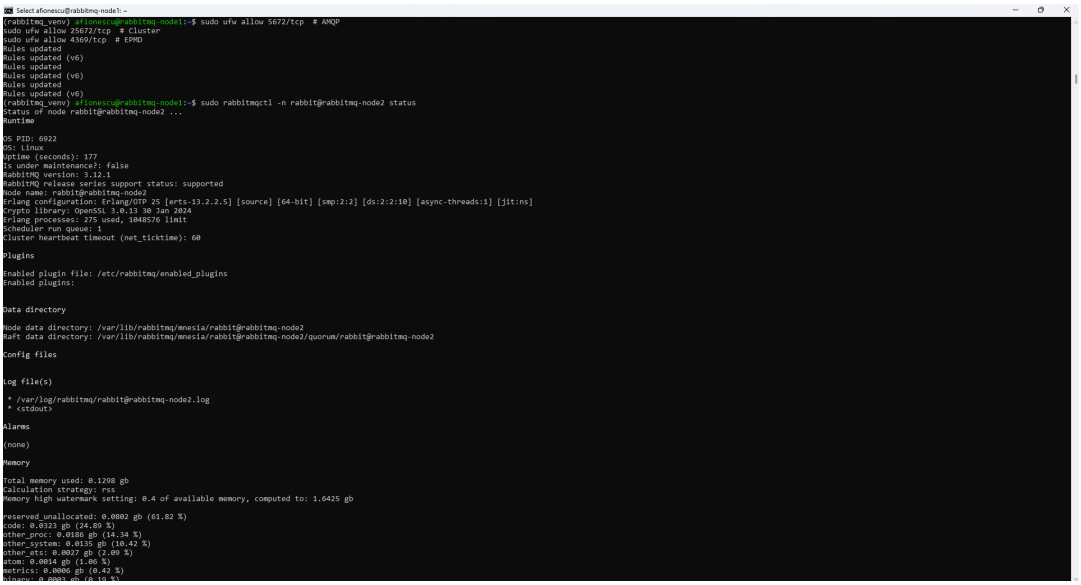
3.8. Firewall Configuration

- I've done the necessary firewall configuration

sudo ufw allow 5672/tcp #AMQP

sudo ufw allow 25672/tcp #Cluster

sudo ufw allow 4369/tcp #EPMD

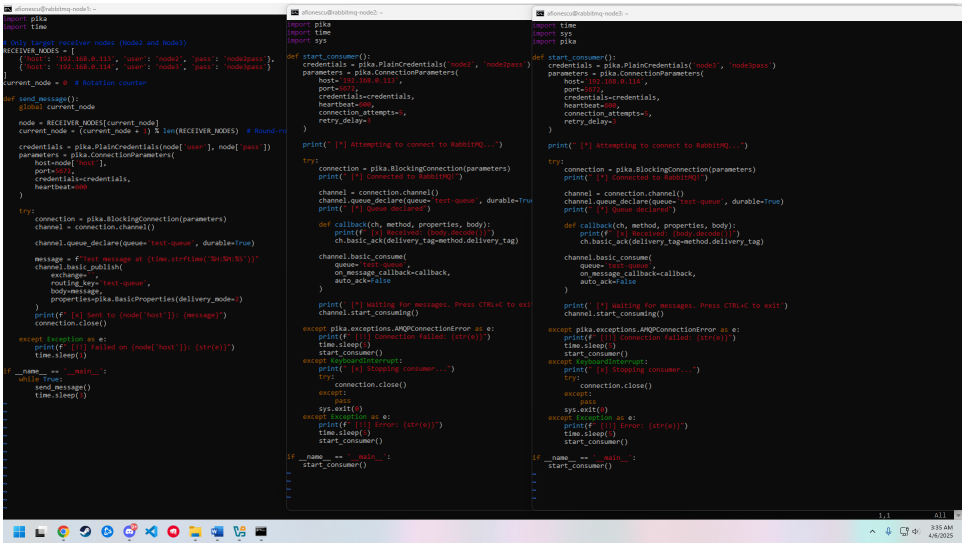


3.9. Testing and Troubleshooting

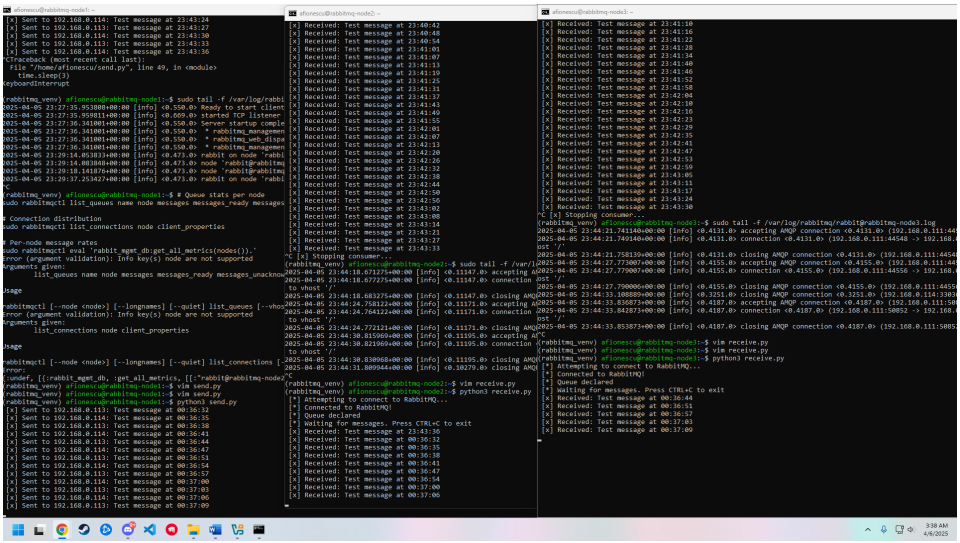
- I’ve had some Initial Testing issues:

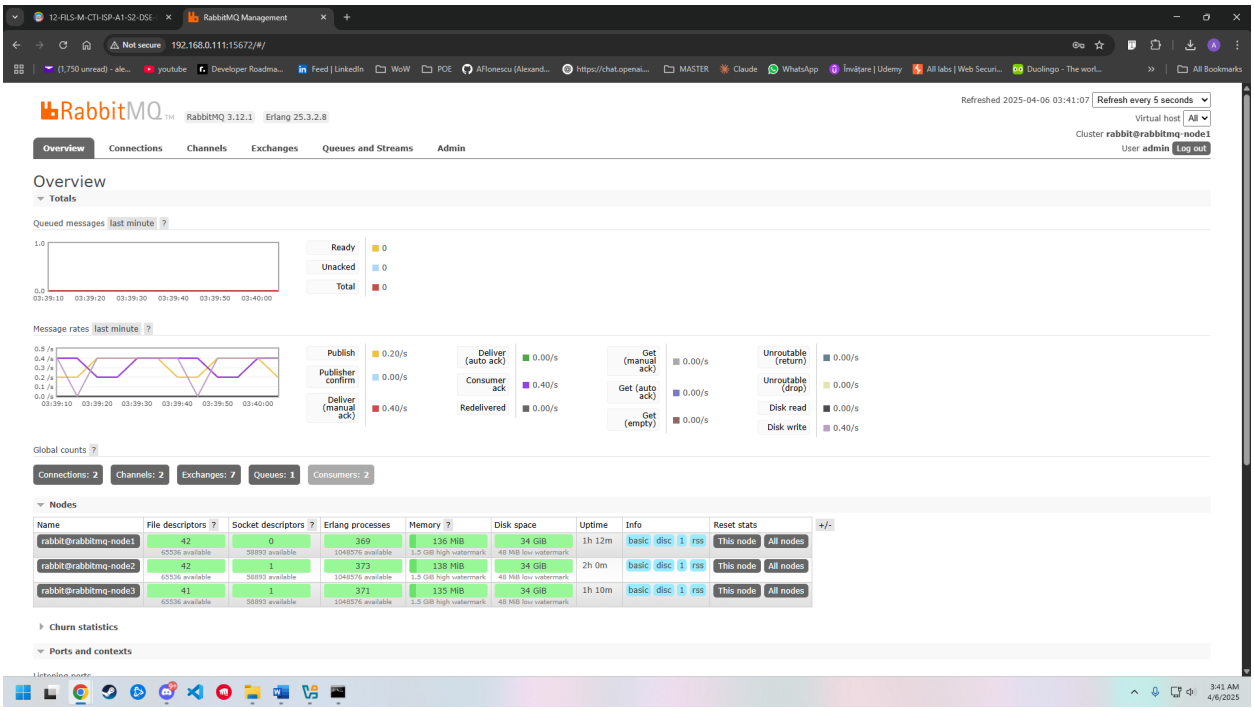
I encountered *ACCESS_REFUSED* initially and I fixed them by adjusting permissions and user tags

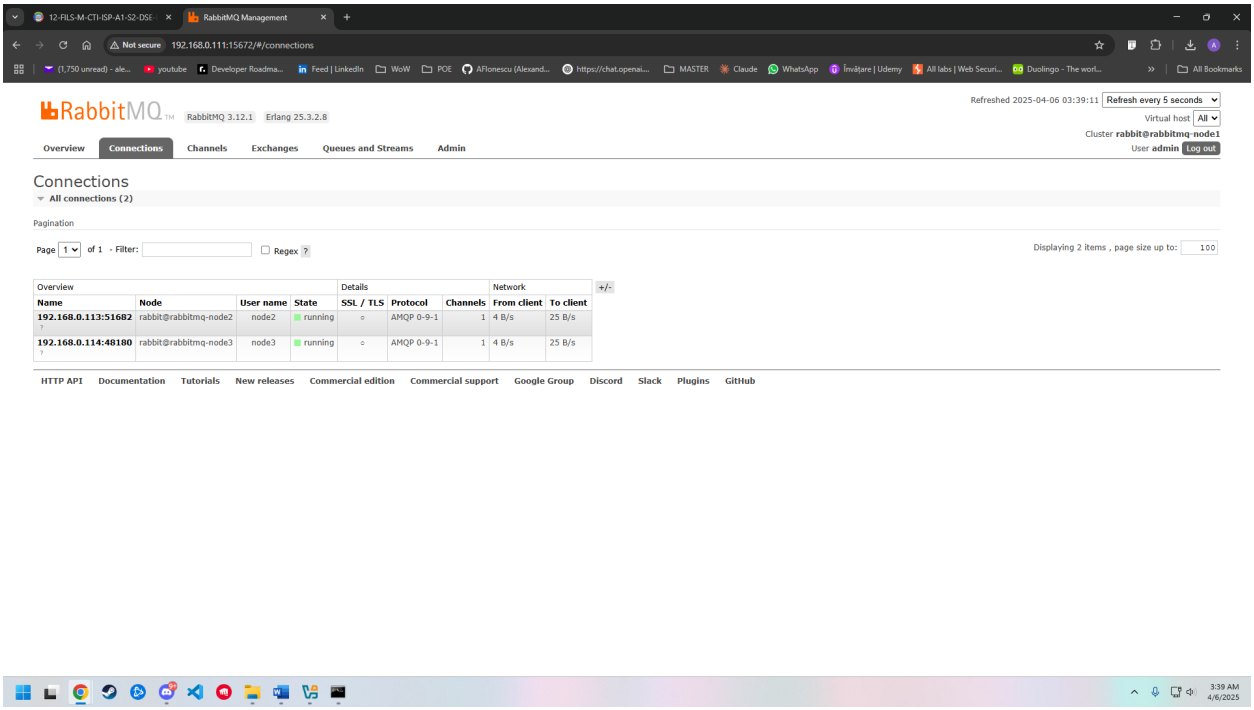
- For testing I’ve decided to test it by creating python scripts to send messages to both other nodes from rabbitmq-node1 (sender.py for rabbitmq-node1 & receive.py for rabbitmq-node2 and rabbitmq-node3)

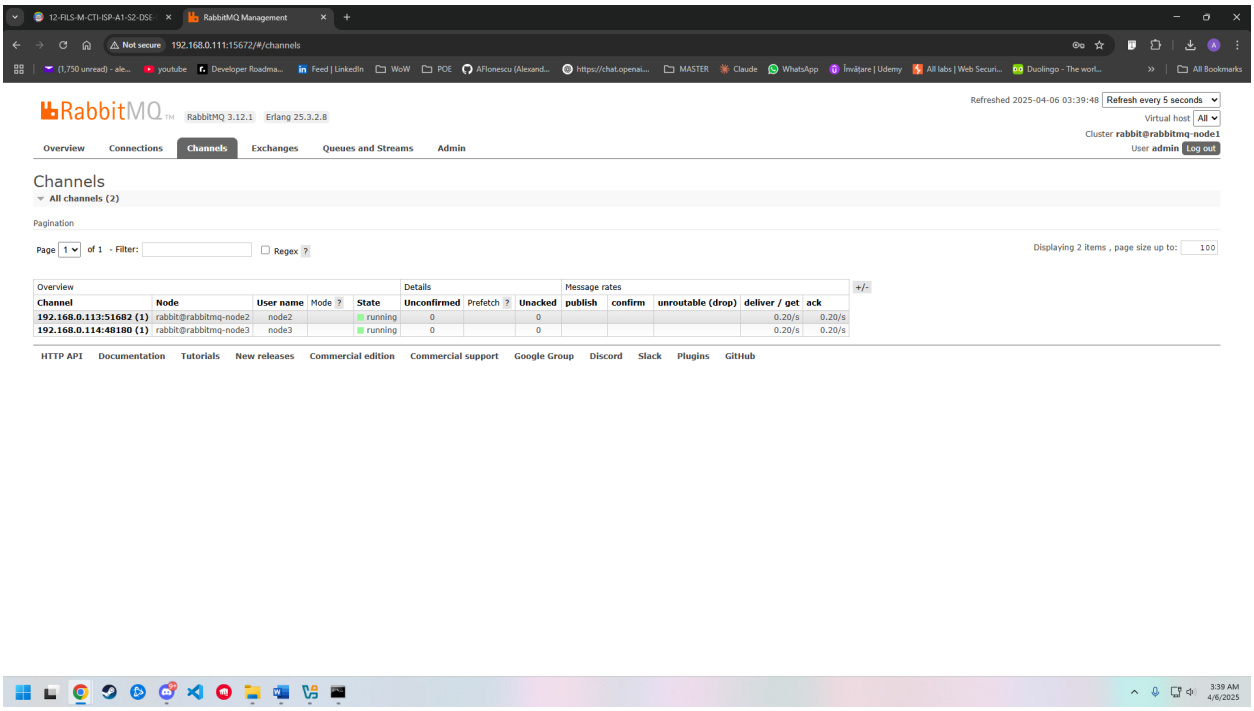


- I’ve ran the scripts and I’ve observed the output from the terminals, as well as the one from interface in the browser and I checked the logs for errors and the results were successful




RabbitMQ Management
Not secure 192.168.0.111:15672/#/
RabbitMQ 3.12.1 Erlang 25.3.2.8
Refreshed 2025-04-06 03:41:07 Refresh every 5 seconds
Virtual host All
Cluster rabbit@rabbitmq-node1
User admin Log out
Overview Connections Channels Exchanges Queues and Streams Admin
Overview
Totals
Queued messages last minute
Ready 0
Unacked 0
Total 0
Message rates last minute
Publish 0.20/s
Publisher confirm 0.00/s
Deliver (manual ack) 0.40/s
Deliver (auto ack) 0.00/s
Consumer ack 0.40/s
Redelivered 0.00/s
Get (manual ack) 0.00/s
Get (auto ack) 0.00/s
Get (empty) 0.00/s
Unroutable (return) 0.00/s
Unroutable (drop) 0.00/s
Disk read 0.00/s
Disk write 0.40/s
Global counts
Connections: 2
Channels: 2
Exchanges: 7
Queues: 1
Consumers: 2
Nodes
Name File descriptors Socket descriptors Erlang processes Memory Disk space Uptime Info Reset stats
rabbit@rabbitmq-node1 42 0 369 136 MiB 34 GiB 1h 12m basic disc 1 rss This node All nodes
rabbit@rabbitmq-node2 42 1 373 138 MiB 34 GiB 2h 0m basic disc 1 rss This node All nodes
rabbit@rabbitmq-node3 41 1 371 135 MiB 34 GiB 1h 10m basic disc 1 rss This node All nodes
Churn statistics
Ports and contexts


RabbitMQ Management
Not secure 192.168.0.111:15672/#/connections
RabbitMQ 3.12.1 Erlang 25.3.2.8
Refreshed 2025-04-06 03:39:11 Refresh every 5 seconds
Virtual host All
Cluster rabbit@rabbitmq-node1
User admin Log out
Overview Connections Channels Exchanges Queues and Streams Admin
Connections
All connections (2)
Pagination
Page 1 of 1 - Filter: Regex
Displaying 2 items , page size up to: 100
Overview Details Network
Name Node User name State SSL / TLS Protocol Channels From client To client
192.168.0.113:51682 rabbit@rabbitmq-node2 node2 running AMQP 0-9-1 1 4 B/s 25 B/s
192.168.0.114:48180 rabbit@rabbitmq-node3 node3 running AMQP 0-9-1 1 4 B/s 25 B/s
HTTP API Documentation Tutorials New releases Commercial edition Commercial support Google Group Discord Slack Plugins GitHub



3.10. Corrected Observations and Personal Experience

During my RabbitMQ cluster setup, I encountered several challenges that required iterative problem-solving to achieve a fully functional system.

3.10.1. Repository Setup Challenges:

I initially struggled with the repository configuration when I accidentally saved the GPG key under two different filenames (rabbitmq.gpg and com.rabbitmq.team.gpg) across multiple attempts. This inconsistency caused some initial confusion, but I resolved it by:

- Consistently using rabbitmq.gpg as the key filename
- Verifying the repository worked with sudo apt update before proceeding
- Double-checking the distribution codename (noble for Ubuntu 24.04)

3.10.2. Cluster Formation Difficulties

The cluster setup required multiple attempts before succeeding. My breakthrough came when I:

- Manually verified the identical .erlang.cookie across all nodes using:

sudo cat /var/lib/rabbitmq/.erlang.cookie

- Ensured proper permissions with:

sudo chown rabbitmq:rabbitmq /var/lib/rabbitmq/.erlang.cookie

sudo chmod 400 /var/lib/rabbitmq/.erlang.cookie

- Systematically tested connectivity between nodes using both ping and telnet on critical ports (5672, 25672, 4369)

3.10.3. Firewall Configuration

The most persistent issues came from blocked inter-node communication. My solution involved:

- Implementing a structured firewall approach:

sudo ufw allow 5672/tcp # AMQP

sudo ufw allow 25672/tcp # Cluster communication

sudo ufw allow 4369/tcp   # EPMD

- Verifying each rule worked by testing connectivity after each change
- Using telnet to confirm ports were truly accessible between nodes

3.10.4. User Permission Evolution

The permission system required several iterations to get right. My successful approach was:

1. Starting with broad permissions for testing:

sudo rabbitmqctl set_permissions -p / node2 "." ".*" ".*"*

2. Gradually refining to more specific permissions once basic functionality worked
3. Creating dedicated users for each node with appropriate tags:

sudo rabbitmqctl set_user_tags node2 monitoring policymaker

3.10.5. High Availability Implementation

After the basic cluster worked, I enhanced reliability with:

sudo rabbitmqctl set_policy ha-all "^" '{"ha-mode":"all", "ha-sync-mode":"automatic"}'

This policy ensured messages would be mirrored across all nodes automatically.

Challenge	Solution
Repository Key Filename Inconsistency	Standardized to use *rabbitmq.gpg* and verified with sudo apt update
Cluster Formation Failures	Verified *identical .erlang.cookie* on all nodes and ensured proper file permissions
Firewall Blocking Inter-Node Communication	Configured UFW rules and confirmed connectivity using telnet
ACCESS_REFUSED Errors	Adjusted permissions and refined user tags with *sudo rabbitmqctl set_permissions*
High Availability Setup Issues	Configured HA policy using *sudo rabbitmqctl set_policy ha-all "^" '{"ha-mode":"all", "ha-sync-mode":"automatic"}*

4. Conclusion

Setting up a **highly available RabbitMQ** cluster across three nodes was a challenging but rewarding experience that taught me valuable lessons in distributed systems, fault tolerance, and troubleshooting.

4.1. Key Achievements

- **Successful Cluster Deployment** – Configured a **3-node RabbitMQ cluster** with synchronized .erlang.cookie files, ensuring secure inter-node communication
- .**High Availability (HA) Policy** – Implemented **mirrored queues** with ha-mode: all to ensure message redundancy across all nodes.
- **Secure User Management** – Removed the default guest user and created **dedicated users** with fine-grained permissions.
- **Firewall & Network Optimization** – Resolved connectivity issues by properly configuring **UFW rules** for AMQP (5672), clustering (25672), and EPMD (4369).
- **Automated Testing** – Validated the setup using a **Python script** that successfully published messages across all nodes.

4.2. Key Lessons Learned:

1. **Persistence Pays Off**: The cluster didn't work on the first try, but methodical troubleshooting succeeded
2. **Verification at Each Step**: Checking connectivity, permissions, and configurations after each change prevented compound errors
3. **Documentation is Crucial**: Keeping notes of changes helped backtrack when something went wrong

4. **Security Matters**: Taking time to properly configure users and permissions prevented future access issues

Final validation was confirmed via Python test scripts, error-free logs, and the management UI.

4.3. Next Steps for the Project

- **Performance Benchmarking:** Test message throughput and latency under different loads to evaluate scalability
- **Fault Tolerance Testing:** Simulate node failures and network issues to verify recovery and data consistency
- **Integration with Applications:** Build a basic messaging application (like a notification system) to demonstrate real-world use
- **Security Enhancements:** Implement message encryption and properly configure user access controls