

**Politehnica University of Bucharest**

**RabbitMQ: Technical Report 1**

**Name:** Alexandru – Florin Ionescu

**Date:** 06/04/2025

# Table of Contents

TABLE OF CONTENTS .....	2
1. INTRODUCTION .....	3
2. ENVIRONMENT SETUP .....	4
2.1. Virtualization Technology.....	4
2.2. Operating System Configuration .....	4
2.3. Network Configuration .....	4
2.4. Virtual Machine Specifications:.....	5
2.5. Network Topology Diagram .....	5
3. INSTALLATION AND FUNCTIONAL VERIFICATION .....	6
3.1. Setting Up VirtualBox .....	6
3.2. Cloning Additional Nodes .....	6
3.3. Fixing the Network .....	7
3.4. Installing RabbitMQ.....	9
3.5. Initial Verification Steps .....	9
3.6. Cluster Configuration Process.....	10
3.7. User Management Configuration .....	12
3.8. Firewall Configuration .....	12
3.9. Testing and Troubleshooting .....	13
3.10. Corrected Observations and Personal Experience .....	15
3.10.1. Repository Setup Challenges: .....	15
3.10.2. Cluster Formation Difficulties .....	15
3.10.3. Firewall Configuration .....	16
3.10.4. User Permission Evolution.....	16
3.10.5. High Availability Implementation .....	16
4. CONCLUSION .....	17
4.1. Key Achievements.....	17
4.2. Key Lessons Learned: .....	17
4.3. Next Steps for the Project .....	18

# 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

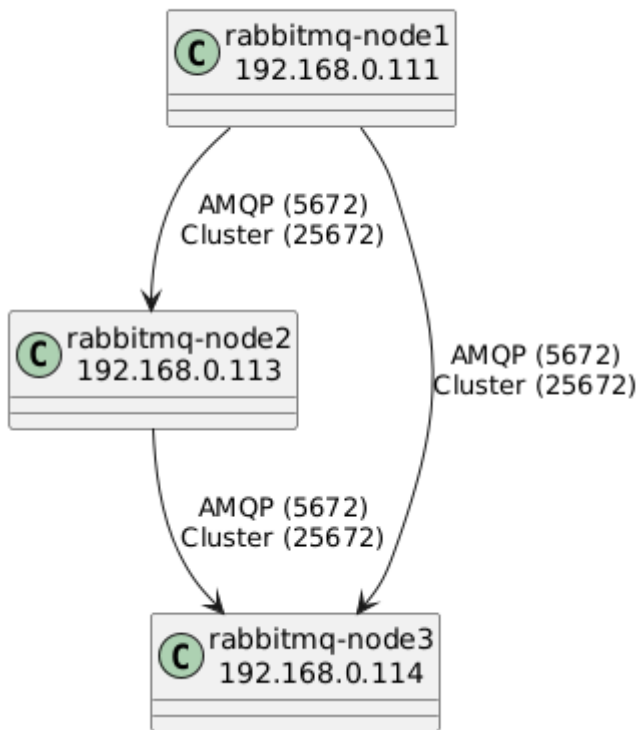
- **Network Mode:** Bridged networking
- **IP Allocation:**
  - **rabbitmq-node1:** 192.168.0.111
  - **rabbitmq-node2:** 192.168.0.113
  - **rabbitmq-node3:** 192.168.0.114
- **Firewall Rules**
  - **5672/tcp** (AMQP)
  - **15672/tcp** (Management UI)
  - **25672/tcp** (Cluster communication)

## 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

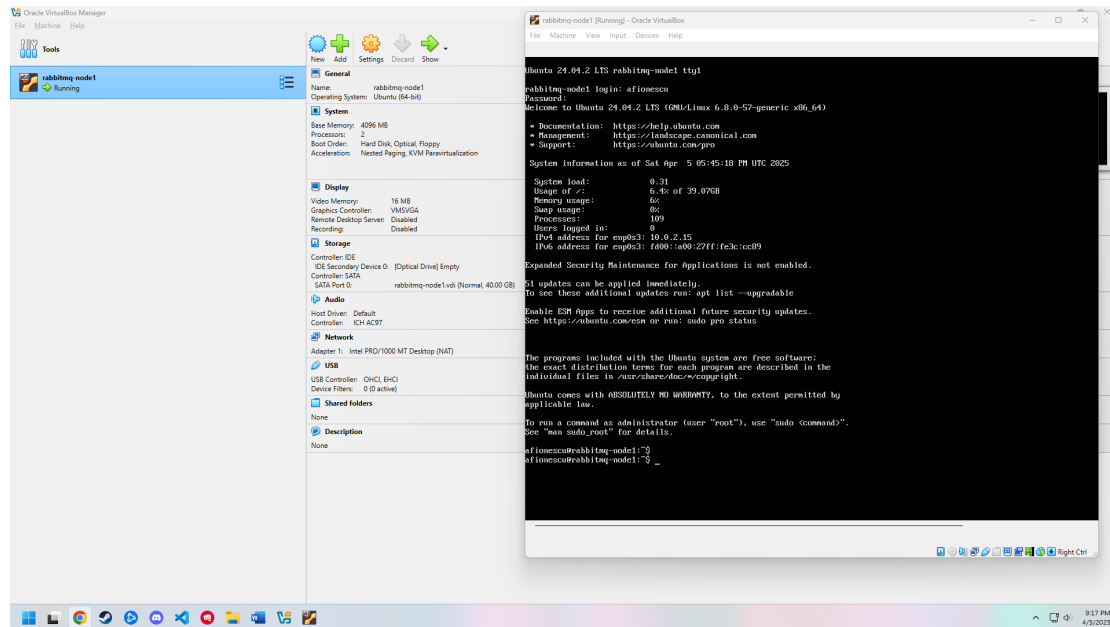
### RabbitMQ Cluster Network Topology



## 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)

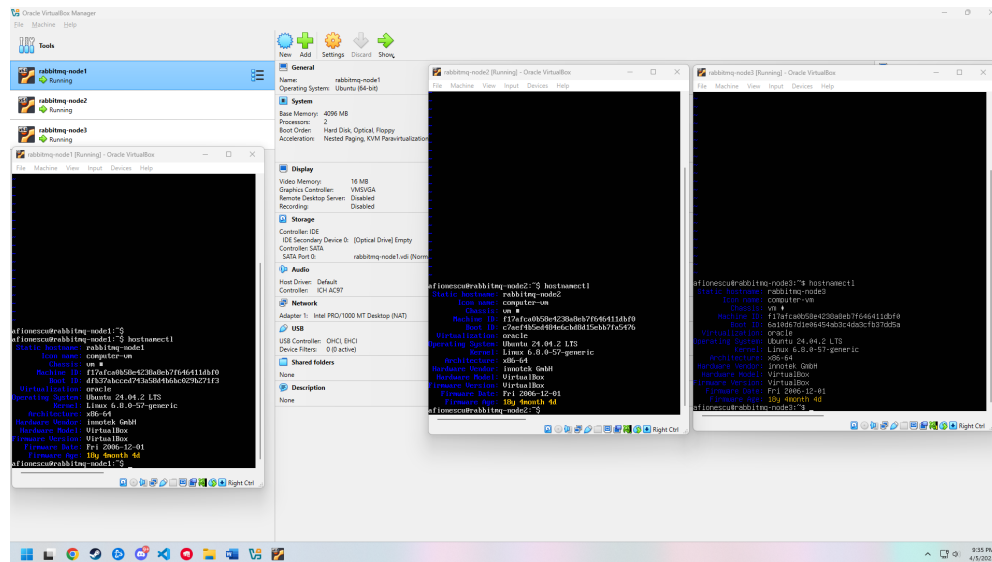
- **MAC Address Policy:** *Generate new MAC addresses for all network adapters* (avoids IP conflicts)
- Change hostnames inside each VM:

### rabbitmq-node2:

```
sudo hostnamectl set-hostname rabbitmq-node2
```

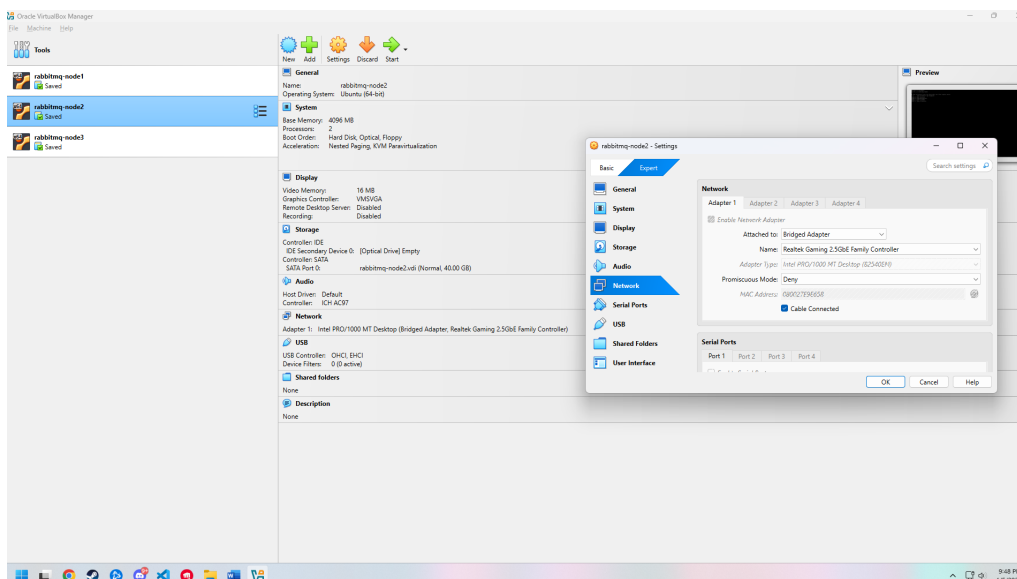
### rabbitmq-node3:

```
sudo hostnamectl set-hostname rabbitmq-node3
```

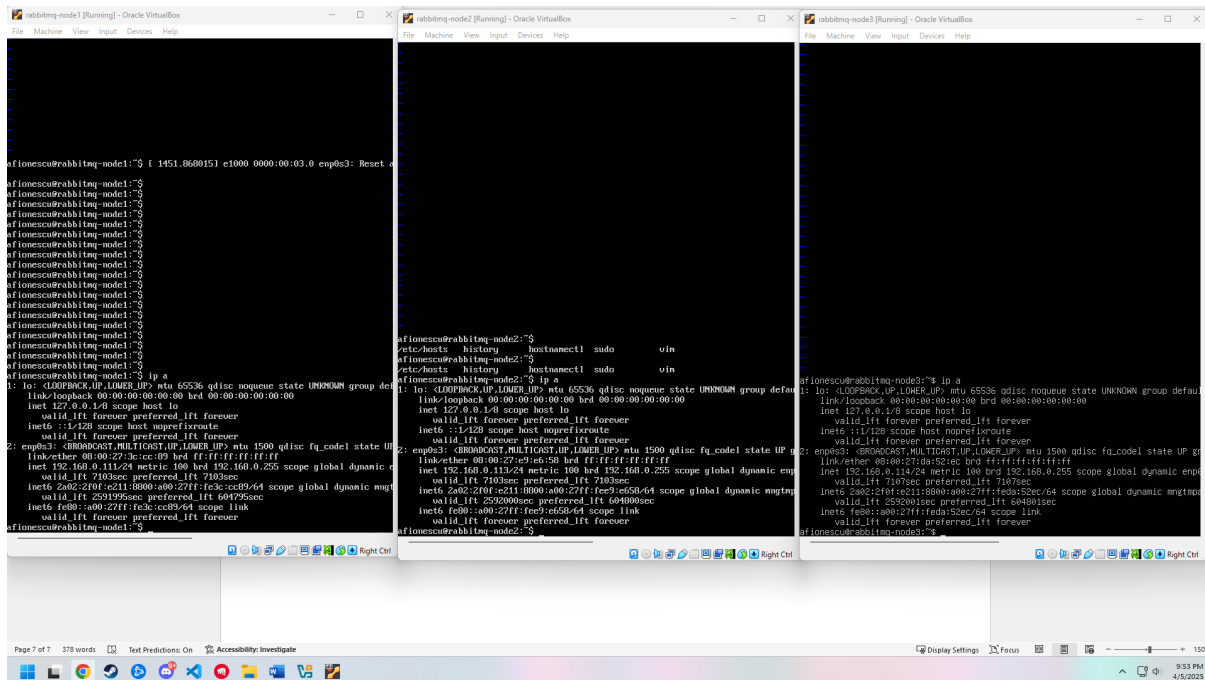


## 3.3. Fixing the Network

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



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

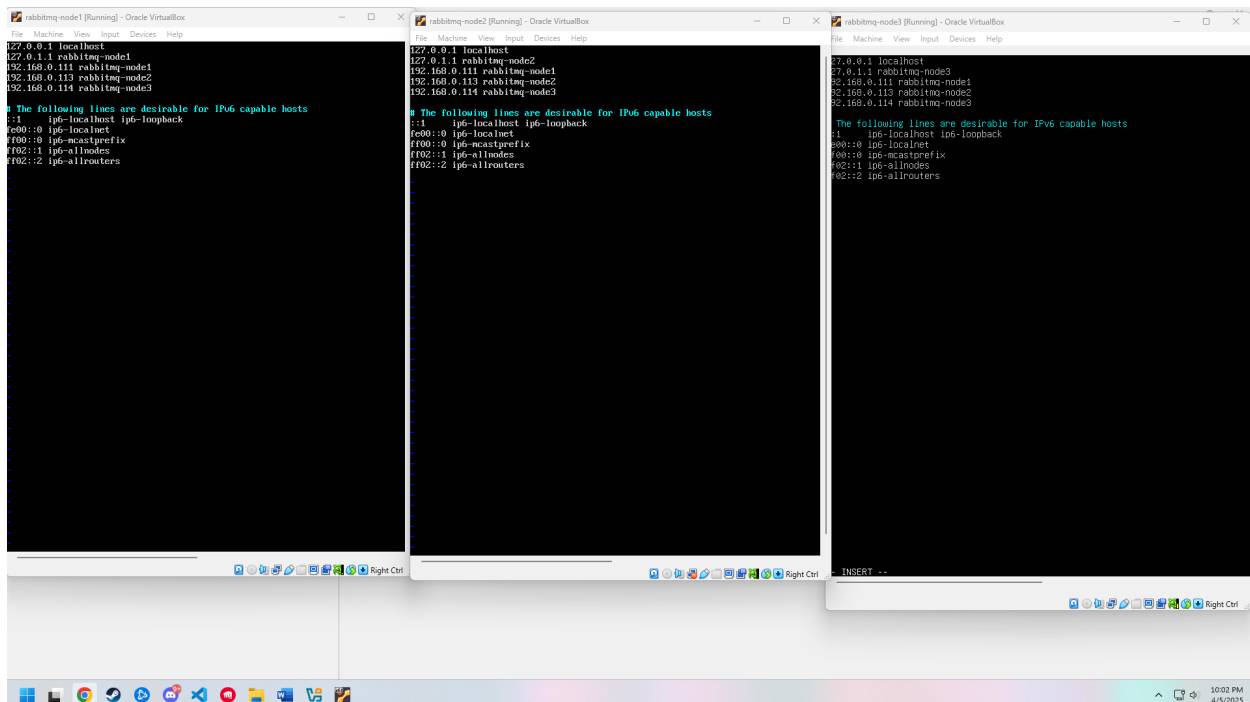


- I updated `/etc/hosts` on all VMs with the ips for the nodes:

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
```

```
Select afionescu@rabbitmq-node1: ~
No VM guests are running outdated hypervisor (qemu) binaries on this host.
afionescu@rabbitmq-node1:~$ sudo systemctl enable --now rabbitmq-server
Synchronizing state of rabbitmq-server.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable rabbitmq-server
afionescu@rabbitmq-node1:~$ sudo rabbitmqctl stop_app
afionescu@rabbitmq-node1:~$ sudo rabbitmqctl reset
afionescu@rabbitmq-node1:~$ sudo rabbitmqctl start_app
Stopping rabbit application on node rabbit@rabbitmq-node1 ...
Resetting node rabbit@rabbitmq-node1 ...
Starting node rabbit@rabbitmq-node1 ...
afionescu@rabbitmq-node1:~$ sudo rabbitmqctl cluster_status
[sudo] password for afionescu:
Cluster status of node rabbit@rabbitmq-node1 ...
Basics
Cluster name: rabbit@rabbitmq-node1
Total CPU cores available cluster-wide: 2
Disk Nodes
rabbit@rabbitmq-node1
Running Nodes
rabbit@rabbitmq-node1
Versions
rabbit@rabbitmq-node1: RabbitMQ 3.12.1 on Erlang 25.3.2.8
CPU Cores
Node: rabbit@rabbitmq-node1, available CPU cores: 2
Maintenance status
Node: rabbit@rabbitmq-node1, status: not under maintenance
Alarms
(none)
Network Partitions
(none)
Listeners
Node: rabbit@rabbitmq-node1, interface: [::], port: 25672, protocol: clustering, purpose: Inter-node and CLI tool communication
Node: rabbit@rabbitmq-node1, interface: [::], port: 5672, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0
Feature flags
flag: classic_mirrored_queue_version, state: enabled
flag: classic_queue_type_delivery_support, state: enabled
flag: direct_exchange_routing_v2, state: enabled
flag: feature_flags_v2, state: enabled
flag: implicit_default_bindings, state: enabled
flag: listener_records_in_ets, state: enabled
flag: maintenance_mode_status, state: enabled
flag: quorum_queue, state: enabled
flag: restart_streams, state: enabled
```

### 3.6. Cluster Configuration Process

- I synchronized the cookie for them to communicate with each other (I copied cookie value from rabbitmq-node1 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

```
Select afionescu@rabbitmq-node1:~
Network Partitions
(none)

Listeners
Node: rabbitmq-node1, interface: [::], port: 25672, protocol: clustering, purpose: inter-node and CLI tool communication
Node: rabbitmq-node1, interface: [::], port: 5672, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0

Feature flags
Flag: classic_mirrored_queue_version, state: enabled
Flag: classic_queue_type_delivery_support, state: enabled
Flag: direct_exchange_routing_v2, state: enabled
Flag: feature_flags_v2, state: enabled
Flag: implicit_default_bindings, state: enabled
Flag: listener_records_in_ets, state: enabled
Flag: maintenance_mode_status, state: enabled
Flag: quorum_queue, state: enabled
Flag: restart_streams, state: enabled
Flag: stream_queue, state: enabled
Flag: stream_sac_coordinator_unblocks_group, state: enabled
Flag: stream_single_active_consumer, state: enabled
Flag: tracking_records_in_ets, state: enabled
Flag: user_limits, state: enabled
Flag: virtual_host_metadata, state: enabled
afionescu@rabbitmq-node1:~$ ping rabbitmq-node2
PING rabbitmq-node2 (192.168.0.113) 56(84) bytes of data:
64 bytes from rabbitmq-node2 (192.168.0.113): icmp_seq=1 ttl=64 time=1.98 ms
64 bytes from rabbitmq-node2 (192.168.0.113): icmp_seq=2 ttl=64 time=1.18 ms
64 bytes from rabbitmq-node2 (192.168.0.113): icmp_seq=3 ttl=64 time=1.21 ms
^C
--- rabbitmq-node2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.180/1.452/1.911/0.372 ms
afionescu@rabbitmq-node1:~$ ping rabbitmq-node1
PING rabbitmq-node1 (127.0.0.1) 56(84) bytes of data:
64 bytes from rabbitmq-node1 (127.0.0.1): icmp_seq=1 ttl=64 time=2.00 ms
64 bytes from rabbitmq-node1 (127.0.0.1): icmp_seq=2 ttl=64 time=0.000 ms
^C
--- rabbitmq-node1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1828ms
rtt min/avg/max/mdev = 0.880/1.080/2.081/1.080 ms
afionescu@rabbitmq-node1:~$ ping rabbitmq-node3
PING rabbitmq-node3 (192.168.0.114) 56(84) bytes of data:
64 bytes from rabbitmq-node3 (192.168.0.114): icmp_seq=1 ttl=64 time=0.980 ms
64 bytes from rabbitmq-node3 (192.168.0.114): icmp_seq=2 ttl=64 time=1.38 ms
64 bytes from rabbitmq-node3 (192.168.0.114): icmp_seq=3 ttl=64 time=1.44 ms
64 bytes from rabbitmq-node3 (192.168.0.114): icmp_seq=4 ttl=64 time=1.29 ms
64 bytes from rabbitmq-node3 (192.168.0.114): icmp_seq=5 ttl=64 time=1.30 ms
64 bytes from rabbitmq-node3 (192.168.0.114): icmp_seq=6 ttl=64 time=1.74 ms
64 bytes from rabbitmq-node3 (192.168.0.114): icmp_seq=7 ttl=64 time=1.55 ms
64 bytes from rabbitmq-node3 (192.168.0.114): icmp_seq=8 ttl=64 time=1.24 ms
^C
--- rabbitmq-node3 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7451ms
rtt min/avg/max/mdev = 0.980/1.364/2.742/0.211 ms
afionescu@rabbitmq-node1:~$ sudo cat /var/lib/rabbitmq/.erlang.cookie
osuefbcwms8RuiKHA2PeG2JmgT8ze+JRFUSMe724j52P2qWkqu
afionescu@rabbitmq-node1:~$ sudo chown rabbitmq:rabbitmq /var/lib/rabbitmq/.erlang.cookie
sudo chmod 400 /var/lib/rabbitmq/.erlang.cookie
afionescu@rabbitmq-node1:~$ sudo service rabbitmq-server start
```

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

sudo rabbitmqctl cluster\_status

```
Select afionescu@rabbitmq-node1:~
afionescu@rabbitmq-node1:~$ sudo rabbitmqctl cluster_status
Cluster status of node rabbitmq-node1 ...
Basics
Cluster name: rabbitmq-node1
Total CPU cores available cluster-wide: 6

Disk Nodes
rabbitmq-node1
rabbitmq-node2
rabbitmq-node3

Running Nodes
rabbitmq-node1
rabbitmq-node2
rabbitmq-node3

Versions
rabbitmq-node1: RabbitMQ 3.12.1 on Erlang 25.3.2.8
rabbitmq-node2: RabbitMQ 3.12.1 on Erlang 25.3.2.8
rabbitmq-node3: RabbitMQ 3.12.1 on Erlang 25.3.2.8

CPU Cores
Node: rabbitmq-node1, available CPU cores: 2
Node: rabbitmq-node2, available CPU cores: 2
Node: rabbitmq-node3, available CPU cores: 2

Maintenance status
Node: rabbitmq-node1, status: not under maintenance
Node: rabbitmq-node2, status: not under maintenance
Node: rabbitmq-node3, status: not under maintenance

Alarms
(none)

Network Partitions
(none)

Listeners
Node: rabbitmq-node1, interface: [::], port: 25672, protocol: clustering, purpose: inter-node and CLI tool communication
Node: rabbitmq-node1, interface: [::], port: 5672, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0
Node: rabbitmq-node2, interface: [::], port: 25672, protocol: clustering, purpose: inter-node and CLI tool communication
Node: rabbitmq-node2, interface: [::], port: 5672, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0
Node: rabbitmq-node3, interface: [::], port: 25672, protocol: clustering, purpose: inter-node and CLI tool communication
Node: rabbitmq-node3, interface: [::], port: 5672, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0

Feature flags
Flag: classic_mirrored_queue_version, state: enabled
Flag: classic_queue_type_delivery_support, state: enabled
Flag: direct_exchange_routing_v2, state: enabled
Flag: feature_flags_v2, state: enabled
Flag: implicit_default_bindings, state: enabled
Flag: listener_records_in_ets, state: enabled
Flag: maintenance_mode_status, state: enabled
```

## 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

```
Select afonsec@rabbitmq-node1:~$ sudo rabbitmqctl stop_app
Select afonsec@rabbitmq-node1:~$ sudo rabbitmqctl reset
Select afonsec@rabbitmq-node1:~$ sudo rabbitmqctl start_app
[sudo] password for afonsec:
Stopping rabbit application on node rabbit@rabbitmq-node1 ...
Resetting node rabbit@rabbitmq-node1 ...
Starting node rabbit@rabbitmq-node1 ...
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo rabbitmqctl add_user node2 node2pass
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo rabbitmqctl add_user node3 node3pass
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo rabbitmqctl set_permissions -p / node2 ".*" ".*" ".*"
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo rabbitmqctl set_permissions -p / node3 ".*" ".*" ".*"
Adding user "node2" ...
Done. Don't forget to grant the user permissions to some virtual hosts! See 'rabbitmqctl help set_permissions' to learn more.
Adding user "node3" ...
Done. Don't forget to grant the user permissions to some virtual hosts! See 'rabbitmqctl help set_permissions' to learn more.
Setting tags for user "node2" to [consumer] ...
Setting tags for user "node3" to [administrator] ...
Setting permissions for user "node2" in vhost "/" ...
Setting permissions for user "node3" in vhost "/" ...
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo rabbitmqctl change_password admin MySecurePassword123
Changing password for user "admin" ...
error:
user "admin" does not exist
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo rabbitmqctl add_user admin MySecurePassword123
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo rabbitmqctl set_user_tags admin administrator
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo rabbitmqctl set_permissions -p / admin ".*" ".*" ".*"
Adding user "admin" ...
Done. Don't forget to grant the user permissions to some virtual hosts! See 'rabbitmqctl help set_permissions' to learn more.
Setting tags for user "admin" to [administrator] ...
Setting permissions for user "admin" in vhost "/" ...
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo rabbitmqctl list_users
Listing users ...
user      tags
admin     [administrator]
node2     [consumer]
node3     [consumer]
guest     [administrator]
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo rabbitmqctl delete_user guest
Deleting user "guest" ...
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo rabbitmqctl list_users
Listing users ...
user      tags
admin     [administrator]
node2     [consumer]
node3     [consumer]
```

## 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
```

```
Select afonsec@rabbitmq-node1:~$ sudo ufw allow 5672/tcp # AMQP
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo ufw allow 25672/tcp # Cluster
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo ufw allow 4369/tcp # EPMD
Rules updated
Rules updated (v6)
Rules updated
Rules updated (v6)
Rules updated
Rules updated (v6)
(rabbitmq_venv) afonsec@rabbitmq-node1:~$ sudo rabbitmqctl -n rabbit@rabbitmq-node2 status
Status of node rabbit@rabbitmq-node2 ...
Runtime
OS PID: 6922
OS: Linux
Uptime (seconds): 177
Is under maintenance: false
RabbitMQ version: 3.12.1
RabbitMQ release series support status: supported
Node name: rabbit@rabbitmq-node2
Erlang configuration: [rlang/GDP 25 [erts-13.2.2.5] [source] [64-bit] [smp:2:2] [ds:2:2:10] [async-threads:1] [:jit:ms]]
Crypto library: OpenSSL 1.0.1k 30 Jan 2024
Erlang processes: 275 used, 1048576 limit
Scheduler run queue: 1
Cluster heartbeat timeout (net_ticktime): 60
Plugins
RabbitMQ plugin file: /etc/rabbitmq/enabled_plugins
Enabled plugins:

Data directory
Node data directory: /var/lib/rabbitmq/mnesia/rabbit@rabbitmq-node2
Raft data directory: /var/lib/rabbitmq/mnesia/rabbit@rabbitmq-node2/quorum/rabbit@rabbitmq-node2
Config files

Log file(s)
* /var/log/rabbitmq/rabbit@rabbitmq-node2.log
* stdout

Alarms
(none)

Memory
Total memory used: 0.1298 gb
Calculation strategy: rss
Memory high watermark setting: 0.4 of available memory, computed to: 1.6425 gb
reserved_unallocated: 0.0002 gb (0.02 %)
code: 0.0323 gb (2.62 %)
other_proc: 0.0180 gb (1.44 %)
other_system: 0.0033 gb (0.26 %)
other_ets: 0.0027 gb (0.21 %)
atom: 0.0014 gb (0.11 %)
metrics: 0.0006 gb (0.05 %)
langsup: 0.0003 gb (0.02 %)
```

## 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 )

```
#!/usr/bin/env python3
import pika
import time

# Only target receiver nodes (Node2 and Node3)
RECEIVER_NODES = [
    ('192.168.0.111', 'user1', 'node2', 'pass1', 'nodepass'),
    ('192.168.0.114', 'user1', 'node3', 'pass1', 'nodepass')
]

current_node = 0 # Rotation counter

def send_message():
    global current_node
    node = RECEIVER_NODES[current_node]
    current_node = (current_node + 1) % len(RECEIVER_NODES) # Round-robin

    credentials = pika.PlainCredentials(node['user'], node['pass'])
    parameters = pika.ConnectionParameters(
        host=node['host'],
        port=node['port'],
        credentials=credentials,
        heartbeat=60,
        connection_attempts=5,
        retry_delay=5
    )

    connection = pika.BlockingConnection(parameters)
    channel = connection.channel()

    channel.queue_declare(queue='test_queue', durable=True)

    message = f"Test message at {time.strftime('%Y-%m-%d %H:%M:%S')}"
    channel.basic_publish(
        exchange='',
        routing_key='test_queue',
        body=message,
        properties=pika.BasicProperties(delivery_mode=2)
    )

    print(f"[*] Sent to {node['host']} : {message}")
    connection.close()

except KeyboardInterrupt as e:
    print(f"[*] Based on {node['host']} : {str(e)}")
    time.sleep(1)

if __name__ == '__main__':
    send_message()
    time.sleep(1)
```

```
#!/usr/bin/env python3
import pika
import sys

def start_consumer():
    credentials = pika.PlainCredentials('user1', 'nodepass')
    parameters = pika.ConnectionParameters(
        host='192.168.0.111',
        port=5672,
        credentials=credentials,
        heartbeat=60,
        connection_attempts=5,
        retry_delay=5
    )

    print(f"[*] Attempting to connect to RabbitMQ...")

    try:
        connection = pika.BlockingConnection(parameters)
        print(f"[*] Connected")

        channel = connection.channel()
        channel.queue_declare(queue='test_queue', durable=True)
        print(f"[*] Queue declared")

        def callback(ch, method, properties, body):
            print(f"[*] Received {body.decode()}")
            ch.basic_ack(delivery_tag=method.delivery_tag)

        channel.basic_consume(
            queue='test_queue',
            on_message_callback=callback,
            auto_ack=False
        )

        print(f"[*] Waiting for messages. Press CTRL+C to exit")
        channel.start_consuming()

    except pika.exceptions.AMQConnectionError as e:
        print(f"Connection failed: {str(e)}")
        time.sleep(1)
    except KeyboardInterrupt as e:
        print(f"[*] Stopping consumer...")
        connection.close()
    except Exception as e:
        print(f"Unexpected error: {str(e)}")
        sys.exit(1)

    sys.exit(0)

if __name__ == '__main__':
    start_consumer()
```

```
#!/usr/bin/env python3
import pika
import sys

def start_consumer():
    credentials = pika.PlainCredentials('user1', 'nodepass')
    parameters = pika.ConnectionParameters(
        host='192.168.0.114',
        port=5672,
        credentials=credentials,
        heartbeat=60,
        connection_attempts=5,
        retry_delay=5
    )

    print(f"[*] Attempting to connect to RabbitMQ...")

    try:
        connection = pika.BlockingConnection(parameters)
        print(f"[*] Connected")

        channel = connection.channel()
        channel.queue_declare(queue='test_queue', durable=True)
        print(f"[*] Queue declared")

        def callback(ch, method, properties, body):
            print(f"[*] Received {body.decode()}")
            ch.basic_ack(delivery_tag=method.delivery_tag)

        channel.basic_consume(
            queue='test_queue',
            on_message_callback=callback,
            auto_ack=False
        )

        print(f"[*] Waiting for messages. Press CTRL+C to exit")
        channel.start_consuming()

    except pika.exceptions.AMQConnectionError as e:
        print(f"Connection failed: {str(e)}")
        time.sleep(1)
    except KeyboardInterrupt as e:
        print(f"[*] Stopping consumer...")
        connection.close()
    except Exception as e:
        print(f"Unexpected error: {str(e)}")
        sys.exit(1)

    sys.exit(0)

if __name__ == '__main__':
    start_consumer()
```

- 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

```
#!/usr/bin/env python3
import pika
import sys

def start_consumer():
    credentials = pika.PlainCredentials('user1', 'nodepass')
    parameters = pika.ConnectionParameters(
        host='192.168.0.111',
        port=5672,
        credentials=credentials,
        heartbeat=60,
        connection_attempts=5,
        retry_delay=5
    )

    print(f"[*] Attempting to connect to RabbitMQ...")

    try:
        connection = pika.BlockingConnection(parameters)
        print(f"[*] Connected")

        channel = connection.channel()
        channel.queue_declare(queue='test_queue', durable=True)
        print(f"[*] Queue declared")

        def callback(ch, method, properties, body):
            print(f"[*] Received {body.decode()}")
            ch.basic_ack(delivery_tag=method.delivery_tag)

        channel.basic_consume(
            queue='test_queue',
            on_message_callback=callback,
            auto_ack=False
        )

        print(f"[*] Waiting for messages. Press CTRL+C to exit")
        channel.start_consuming()

    except pika.exceptions.AMQConnectionError as e:
        print(f"Connection failed: {str(e)}")
        time.sleep(1)
    except KeyboardInterrupt as e:
        print(f"[*] Stopping consumer...")
        connection.close()
    except Exception as e:
        print(f"Unexpected error: {str(e)}")
        sys.exit(1)

    sys.exit(0)

if __name__ == '__main__':
    start_consumer()
```

```
#!/usr/bin/env python3
import pika
import sys

def start_consumer():
    credentials = pika.PlainCredentials('user1', 'nodepass')
    parameters = pika.ConnectionParameters(
        host='192.168.0.114',
        port=5672,
        credentials=credentials,
        heartbeat=60,
        connection_attempts=5,
        retry_delay=5
    )

    print(f"[*] Attempting to connect to RabbitMQ...")

    try:
        connection = pika.BlockingConnection(parameters)
        print(f"[*] Connected")

        channel = connection.channel()
        channel.queue_declare(queue='test_queue', durable=True)
        print(f"[*] Queue declared")

        def callback(ch, method, properties, body):
            print(f"[*] Received {body.decode()}")
            ch.basic_ack(delivery_tag=method.delivery_tag)

        channel.basic_consume(
            queue='test_queue',
            on_message_callback=callback,
            auto_ack=False
        )

        print(f"[*] Waiting for messages. Press CTRL+C to exit")
        channel.start_consuming()

    except pika.exceptions.AMQConnectionError as e:
        print(f"Connection failed: {str(e)}")
        time.sleep(1)
    except KeyboardInterrupt as e:
        print(f"[*] Stopping consumer...")
        connection.close()
    except Exception as e:
        print(f"Unexpected error: {str(e)}")
        sys.exit(1)

    sys.exit(0)

if __name__ == '__main__':
    start_consumer()
```

```
#!/usr/bin/env python3
import pika
import sys

def start_consumer():
    credentials = pika.PlainCredentials('user1', 'nodepass')
    parameters = pika.ConnectionParameters(
        host='192.168.0.114',
        port=5672,
        credentials=credentials,
        heartbeat=60,
        connection_attempts=5,
        retry_delay=5
    )

    print(f"[*] Attempting to connect to RabbitMQ...")

    try:
        connection = pika.BlockingConnection(parameters)
        print(f"[*] Connected")

        channel = connection.channel()
        channel.queue_declare(queue='test_queue', durable=True)
        print(f"[*] Queue declared")

        def callback(ch, method, properties, body):
            print(f"[*] Received {body.decode()}")
            ch.basic_ack(delivery_tag=method.delivery_tag)

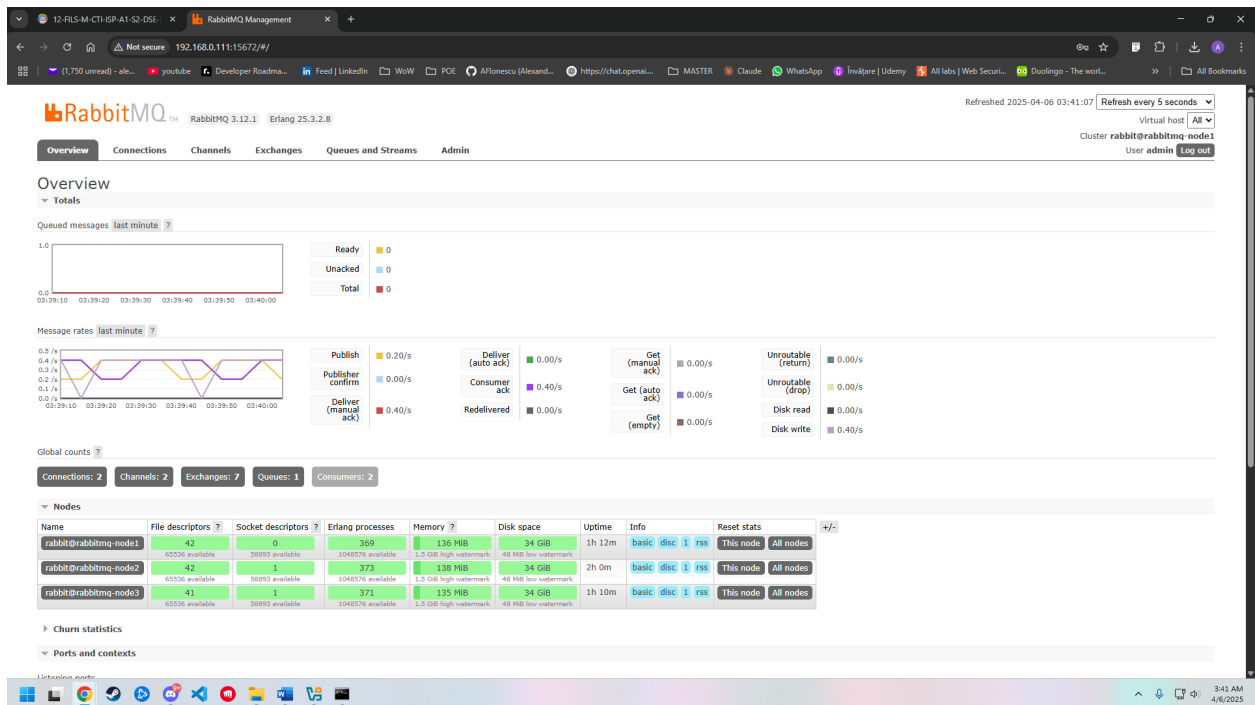
        channel.basic_consume(
            queue='test_queue',
            on_message_callback=callback,
            auto_ack=False
        )

        print(f"[*] Waiting for messages. Press CTRL+C to exit")
        channel.start_consuming()

    except pika.exceptions.AMQConnectionError as e:
        print(f"Connection failed: {str(e)}")
        time.sleep(1)
    except KeyboardInterrupt as e:
        print(f"[*] Stopping consumer...")
        connection.close()
    except Exception as e:
        print(f"Unexpected error: {str(e)}")
        sys.exit(1)

    sys.exit(0)

if __name__ == '__main__':
    start_consumer()
```



RabbitMQ Management Connections

Refreshed 2025-04-06 03:39:11 | Refresh every 5 seconds

Virtual host: All | Cluster: rabbit@rabbitmq-node1 | User: admin | Log out

### Connections

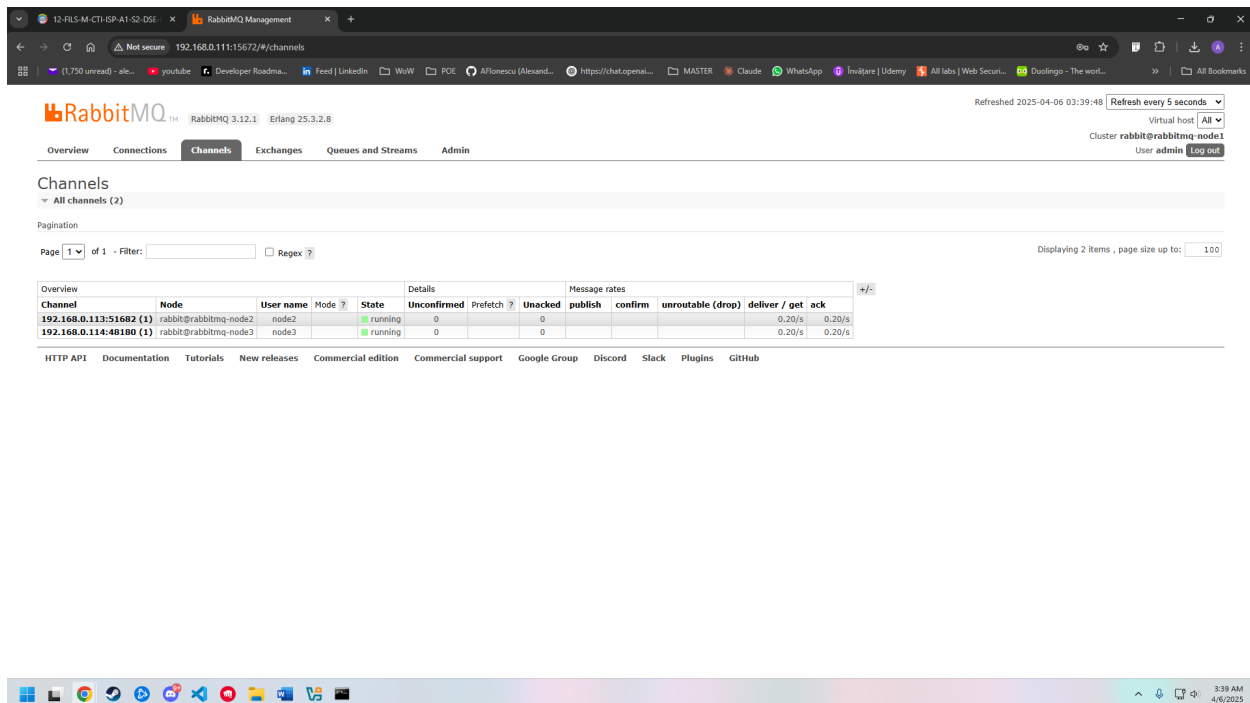
All connections (2)

Pagination: Page 1 of 1 - Filter: | Regexp

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	o	AMQP 0-9-1	1	4 B/s	25 B/s
192.168.0.114:48180	rabbit@rabbitmq-node3	node3	running	o	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 <i>rabbitmq.gpg</i> and verified with <code>sudo apt update</code>
Cluster Formation Failures	Verified <i>identical .erlang.cookie</i> 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 <code>sudo rabbitmqctl set_permissions</code>
High Availability Setup Issues	Configured HA policy using <code>sudo rabbitmqctl set_policy ha-all "" '{"ha-mode":"all", "ha-sync-mode":"automatic"}'</code>

## 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