Week 2 Tasks:

Project Overview

Objective

Build a highly available, secure RabbitMQ messaging cluster with 3 nodes, implement message routing with dead-letter queues, and monitoring and maintenance procedures.

Step-by-Step Implementation

Phase 1: Environment Setup

1.1 Virtual Machine Configuration

# Each VM has two network adapters:

- Adapter 1: Bridged Adapter

- Adapter 2: Host-Only (for VM communication and static IP)

# Configure static IPs on Host-Only network

sudo nano /etc/netplan/.yaml

#Network Configuration:

network:

ethernets:

enp0s8:

addresses: [192.168.56.3/24] #host only IP

dhcp4: false

enp0s9:

dhcp4: true

# Configure /etc/hosts on all nodes

sudo nano /etc/hosts

/etc/hosts content

127.0.0.1 localhost

192.168.56.101 vm4

192.168.56.102 vm5

192.168.56.103 vm6

Phase 2: RabbitMQ Installation and TLS Configuration

Install Essential Dependencies

sudo apt-get update -y

sudo apt-get install curl gnupg -y

Enable RabbitMQ application repository:

echo "deb http://www.rabbitmq.com/debian/ testing main" >> /etc/apt/sources.list



Add the verification key for the package:

curl http://www.rabbitmq.com/rabbitmq-signing-key-public.asc | sudo apt-key add -

Update the sources with our new addition from above:

apt-get update



Enable apt HTTPS Transport

In order for apt to be able to download RabbitMQ and Erlang packages from the Team RabbitMQ apt repositories or Launchpad, the apt-transport-https package must be installed:

**sudo apt-get install apt-transport-https**

Install Erlang Packages:

sudo apt-get install -y erlang-base \

erlang-asn1 erlang-crypto erlang-eldap erlang-ftp erlang-inets \

erlang-mnesia erlang-os-mon erlang-parsetools erlang-public-key \

erlang-runtime-tools erlang-snmp erlang-ssl \

erlang-syntax-tools erlang-tftp erlang-tools erlang-xmerl

And finally, download and install RabbitMQ:

sudo apt-get install rabbitmq-server



Generate TLS Certificates

# Create Certificate Authority

openssl genrsa -out ca.key 2048

openssl req -new -x509 -days 3650 -key ca.key -out ca.crt -subj "/CN=RabbitMQ-CA"

# Create server certificate

openssl genrsa -out server.key 2048

openssl req -new -key server.key -out server.csr -subj "/CN=rabbitmq"

openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 3650

# Create directory and copy certificates

sudo mkdir -p /etc/rabbitmq/ssl

sudo cp ca.crt server.crt server.key /etc/rabbitmq/ssl/

sudo chown rabbitmq:rabbitmq /etc/rabbitmq/ssl/\*

sudo chmod 644 /etc/rabbitmq/ssl/\*.crt

sudo chmod 600 /etc/rabbitmq/ssl/\*.key

Configure RabbitMQ with TLS

sudo nano /etc/rabbitmq/rabbitmq.conf

RabbitMQ Configuration:

# TLS Configuration

listeners.ssl.default = 5671

ssl\_options.cacertfile = /etc/rabbitmq/ssl/ca.crt

ssl\_options.certfile = /etc/rabbitmq/ssl/server.crt

ssl\_options.keyfile = /etc/rabbitmq/ssl/server.key

ssl\_options.verify = verify\_peer

ssl\_options.fail\_if\_no\_peer\_cert = false

# TLS Security - Only TLS 1.2+ with strong ciphers

ssl\_options.versions.1 = tlsv1.2

ssl\_options.ciphers.1 = ECDHE-ECDSA-AES256-GCM-SHA384

ssl\_options.ciphers.2 = ECDHE-RSA-AES256-GCM-SHA384

# Persistence

disk\_free\_limit.absolute = 1GB

default\_user = admin

default\_pass = securepassword

# Enable Management Plugin

management.ssl.port = 15671

management.ssl.cacertfile = /etc/rabbitmq/ssl/ca.crt

management.ssl.certfile = /etc/rabbitmq/ssl/server.crt

management.ssl.keyfile = /etc/rabbitmq/ssl/server.key

Add nodes   
  
#Configure Erlang Cookie (Cluster Security Key)

Copying the erlang cookie to the other nodes

sudo scp /var/lib/rabbitmq/.erlang.cookie user@vm5:/var/lib/rabbitmq/.erlang.cookie

sudo scp /var/lib/rabbitmq/.erlang.cookie user@vm6:/var/lib/rabbitmq/.erlang.cookie

Setting proper permissions for the file

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

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

#Form the Cluster

# Join nodes to cluster (from vm5 and vm6)

sudo rabbitmqctl stop\_app

sudo rabbitmqctl join\_cluster rabbit@vm4

sudo rabbitmqctl start\_app

#Verify Cluster Status

sudo rabbitmqctl cluster\_status

Create Virtual Host and User

# Create vhost and user

sudo rabbitmqctl add\_vhost /myapp

sudo rabbitmqctl add\_user myuser mypassword

sudo rabbitmqctl set\_permissions -p /myapp myuser ".\*" ".\*" ".\*"

Configure Queues with Dead Letter Exchange

# Create Dead Letter Exchange

sudo rabbitmqadmin declare exchange --vhost=/myapp name=dlx type=direct

# Create DATA queue with TTL and DLX

sudo rabbitmqadmin declare queue --vhost=/myapp name=DATA \

arguments='{"x-dead-letter-exchange":"dlx","x-message-ttl":60000}'

# Create DATA\_SIDELINE queue

sudo rabbitmqadmin declare queue --vhost=/myapp name=DATA\_SIDELINE

# Bind DLX to DATA\_SIDELINE

sudo rabbitmqadmin declare binding --vhost=/myapp source=dlx destination=DATA\_SIDELINE routing\_key=DATA

Python Publisher script:

import pika

import ssl

# Group all configuration settings in one place

CONFIG = {

'host': 'rabbitmq',

'vhost': 'myvhost',

'user': 'myuser',

'pass': 'mypassword',

'port': 5671,

'ca\_file': 'ca.crt',

'queue': 'DATA'

}

def main():

"""Connects to RabbitMQ, sends messages, and closes the connection."""

try:

# 1. Use an AMQPS URL to simplify connection parameters

amqp\_url = (f"amqps://{CONFIG['user']}:{CONFIG['pass']}@"

f"{CONFIG['host']}:{CONFIG['port']}/{CONFIG['vhost']}")

# 2. SSL options are still needed for the custom CA file

context = ssl.create\_default\_context(ssl.Purpose.SERVER\_AUTH, cafile=CONFIG['ca\_file'])

context.minimum\_version = ssl.TLSVersion.TLSv1\_2

ssl\_options = pika.SSLOptions(context)

parameters = pika.URLParameters(amqp\_url)

parameters.ssl\_options = ssl\_options

# 3. Use a 'with' statement for automatic connection closing

with pika.BlockingConnection(parameters) as connection:

channel = connection.channel()

queue\_args = {

"x-dead-letter-exchange": "dlx",

"x-message-ttl": 60000

}

channel.queue\_declare(queue=CONFIG['queue'], durable=True, arguments=queue\_args)

print("Publishing 5 messages...")

for i in range(5):

message\_body = f'Secure Message #{i+1}'

channel.basic\_publish(

exchange='',

routing\_key=CONFIG['queue'],

body=message\_body,

properties=pika.BasicProperties(delivery\_mode=2) # persistent message

)

print(f" [x] Sent '{message\_body}'")

print("Messages sent.")

except Exception as e:

print(f"An error occurred: {e}")

if \_\_name\_\_ == '\_\_main\_\_':

main()

Creating a Consumer:

import pika

import ssl

import sys

import time

# 1. Group all configuration settings in one place

CONFIG = {

'host': 'rabbitmq',

'vhost': 'myvhost',

'user': 'myuser',

'pass': 'myuser',

'port': 5671,

'ca\_file': 'ca.crt',

'queue': 'DATA'

}

def main():

"""Establishes a connection and starts consuming messages from RabbitMQ."""

connection = None

try:

# 2. SSL/TLS setup

credentials = pika.PlainCredentials(CONFIG['user'], CONFIG['pass'])

context = ssl.create\_default\_context(ssl.Purpose.SERVER\_AUTH, cafile=CONFIG['ca\_file']) # <-- FIXED: Used CONFIG dictionary

context.minimum\_version = ssl.TLSVersion.TLSv1\_2

ssl\_options = pika.SSLOptions(context, CONFIG['host'])

parameters = pika.ConnectionParameters(

host=CONFIG['host'],

port=CONFIG['port'],

virtual\_host=CONFIG['vhost'],

credentials=credentials,

ssl\_options=ssl\_options

)

connection = pika.BlockingConnection(parameters)

channel = connection.channel()

# 3. Define and declare the queue to match the server's state <-- FIXED: Added this block

queue\_args = {

"x-dead-letter-exchange": "dlx",

"x-message-ttl": 60000

}

channel.queue\_declare(queue=CONFIG['queue'], durable=True, arguments=queue\_args)

def callback(ch, method, properties, body):

"""Processes a received message."""

print(f" [x] Received '{body.decode()}'")

time.sleep(1) # Simulate work

print(" [x] Done processing.")

ch.basic\_ack(delivery\_tag=method.delivery\_tag)

# 4. Start consuming

channel.basic\_qos(prefetch\_count=1)

channel.basic\_consume(queue=CONFIG['queue'], on\_message\_callback=callback)

print(f" [\*] Waiting for messages in queue '{CONFIG['queue']}'. To exit press CTRL+C")

channel.start\_consuming()

except KeyboardInterrupt:

print('Interrupted by user. Closing connection.')

except Exception as e:

print(f"An error occurred: {e}")

finally:

if connection and connection.is\_open:

connection.close()

if \_\_name\_\_ == '\_\_main\_\_':

main()

Move messages from data\_sideline to data and delete from data\_sideline

import pika

import ssl

import sys

# 1. Group all configuration settings in one place

CONFIG = {

'host': 'rabbitmq',

'vhost': 'myvhost',

'user': 'myuser',

'pass': 'myuser',

'port': 5671,

'ca\_file': 'ca.crt',

'source\_queue': 'DATA\_SIDELINE',

'destination\_queue': 'DATA'

}

def main():

"""Connects to RabbitMQ and moves all messages from one queue to another."""

connection = None

messages\_moved = 0

try:

# 2. Setup a secure TLS connection

credentials = pika.PlainCredentials(CONFIG['user'], CONFIG['pass'])

context = ssl.create\_default\_context(ssl.Purpose.SERVER\_AUTH, cafile=CONFIG['ca\_file'])

context.minimum\_version = ssl.TLSVersion.TLSv1\_2

ssl\_options = pika.SSLOptions(context, CONFIG['host'])

parameters = pika.ConnectionParameters(

host=CONFIG['host'],

port=CONFIG['port'],

virtual\_host=CONFIG['vhost'],

credentials=credentials,

ssl\_options=ssl\_options

)

connection = pika.BlockingConnection(parameters)

channel = connection.channel()

print(f"[\*] Connected to vhost '{CONFIG['vhost']}'. Checking for messages in '{CONFIG['source\_queue']}'...")

# 3. Loop through messages in the source queue one by one

while True:

# Get a single message from the source queue

method\_frame, header\_frame, body = channel.basic\_get(queue=CONFIG['source\_queue'])

# If the queue is empty, method\_frame will be None

if method\_frame is None:

print("[\*] Queue is empty. No more messages to move.")

break

# 4. Republish the message to the destination queue

# We use the original properties (headers, etc.) from the message

channel.basic\_publish(

exchange='', # The default exchange

routing\_key=CONFIG['destination\_queue'],

properties=header\_frame,

body=body

)

# 5. Acknowledge the original message to remove it from the source queue

channel.basic\_ack(delivery\_tag=method\_frame.delivery\_tag)

messages\_moved += 1

print(f" [✔] Moved message: {body.decode()}")

except Exception as e:

print(f"An error occurred: {e}")

finally:

print(f"\n[\*] Complete. Total messages moved: {messages\_moved}")

if connection and connection.is\_open:

connection.close()

if \_\_name\_\_ == '\_\_main\_\_':

main()

Network Partition Simulation and Recovery

# Simulate network partition (block port 25672)

echo "Healing the network partition..." sudo iptables -D INPUT -p tcp --dport 25672 -s vm5 -j DROP

# Check cluster status

sudo rabbitmqctl cluster\_status

# Manual recovery

sudo rabbitmqctl stop\_app

sudo rabbitmqctl force\_boot

sudo rabbitmqctl start\_app

# Auto-recovery configuration

echo "cluster\_partition\_handling = autoheal" | sudo tee -a /etc/rabbitmq/rabbitmq.conf