Connecting Two EC2 Instances Using AWS MQ (RabbitMQ)

Goal

- **EC2–1 → Producer** (sends messages)
- **EC2–2 → Consumer** (receives messages)
- Both connect to **Amazon MQ**
- Messaging happens through a **RabbitMQ broker**

1️⃣ Big Difference vs SQS (Context)

Amazon SQS	Amazon MQ
Serverless	Broker-based
AWS APIs	Industry protocols
Queue URL	Broker endpoint
IAM auth	Username + password
Polling	Persistent connection

📌 Amazon MQ behaves like a **traditional RabbitMQ server**, but AWS manages it.

2️⃣ Architecture (Simple View)

```
EC2 (Producer)
     |
     |  AMQP over TLS (5671)
     v
 Amazon MQ (RabbitMQ Broker)
     ^
     |
EC2 (Consumer)
```

3️⃣ STEP 1: Create an Amazon MQ Broker

1. Go to **Amazon MQ**
2. Click **Create broker**
3. Choose engine:
    a. **RabbitMQ**
4. Deployment mode:
    a. Single-instance (lab)
    b. Active/Standby (prod)
5. Give broker a name
6. Create broker

4️⃣ STEP 2: Configure Broker Authentication

1. Create a **broker user**
2. Set **username & password**
3. Save credentials (used by apps)

📌 Amazon MQ uses **broker credentials**, not IAM.

5️⃣ STEP 3: Network Configuration (CRITICAL)

VPC

- Broker and both EC2s must be in the **same VPC**

Security Group (Broker)

Allow inbound:

- **TCP 5671** (AMQPS / TLS)
- Source:
    - EC2 security group **or**
    - VPC CIDR (example: `10.0.0.0/16`)

EC2 outbound:

- Allow all (default)

6️⃣ STEP 4: Get Broker Endpoint

Go to:

Amazon MQ → Broker → Connections

You will see:

```
amqps://b-xxxx.mq.us-east-1.on.aws:5671
```

Use **ONLY the hostname**:

```
b-xxxx.mq.us-east-1.on.aws
```

❌ Do NOT use `https://`

❌ Do NOT include port in host

7️⃣ STEP 5: Prepare Both EC2 Instances

On **both EC2s**:

```
sudo yum update -y
sudo yum install python3 python3-pip -y
```

Verify:

```
pip3 --version
```

Install RabbitMQ client:

```
pip3 install pika
```

8️⃣ STEP 6: Create Queue (One-Time)

Using **RabbitMQ Web UI (port 15672)**:

1. Login with broker credentials
2. Create queue:
    a. Name: `Testing`
    b. Durable: YES

📌 This is important and must match application code.

9️⃣ STEP 7: Producer EC2 – Send Message (TLS + Durable)

Create `producer.py`:

```
import pika
import ssl

credentials = pika.PlainCredentials('Azure_Wolf', 'Deadzone@786924')

ssl_context = ssl.create_default_context()

connection = pika.BlockingConnection(
    pika.ConnectionParameters(
        host='b-31cee6a9-4a12-4714-938b-a08cc255df88.mq.us-east-1.on.aws',
        port=5671,
        credentials=credentials,
        ssl_options=pika.SSLOptions(ssl_context)
    )
)

channel = connection.channel()

# IMPORTANT: durable=True must match existing queue
channel.queue_declare(queue='Testing', durable=True)

channel.basic_publish(
    exchange='',
    routing_key='Testing',
    body='Hello from Producer EC2'
)
```

```
print("Message sent")
connection.close()
```

Run:

```
python3 producer.py
```

Expected output:

```
Message sent
```

🔟 STEP 8: Consumer EC2 – Receive Message (TLS + Durable)

Create `consumer.py`:

```
import pika
import ssl

credentials = pika.PlainCredentials('Azure_Wolf', 'Deadzone@786924')

ssl_context = ssl.create_default_context()

connection = pika.BlockingConnection(
    pika.ConnectionParameters(
        host='b-31cee6a9-4a12-4714-938b-a08cc255df88.mq.us-east-1.on.aws',
        port=5671,
        credentials=credentials,
        ssl_options=pika.SSLOptions(ssl_context)
    )
)

channel = connection.channel()
channel.queue_declare(queue='Testing', durable=True)

def callback(ch, method, properties, body):
    print("Received:", body.decode())
    ch.basic_ack(delivery_tag=method.delivery_tag)

channel.basic_consume(
```

```
    queue='Testing',
    on_message_callback=callback
)

print("Waiting for messages...")
channel.start_consuming()
```

Run:

```
python3 consumer.py
```

Expected output:

```
Received: Hello from Producer EC2
```

1️⃣1️⃣ STEP 9: Verify in RabbitMQ UI

- `Ready` → goes from `1` to `0`
- No `Unacked` messages
- Queue state: **running**

This confirms end-to-end success.

1️⃣2️⃣ What You Achieved (Plain English)

- Producer EC2 sent a message
- Amazon MQ securely accepted it (TLS)
- Queue persisted the message (durable)
- Consumer EC2 received and acknowledged it
- No EC2s talked directly to each other

1️⃣3️⃣ Common Errors You Avoided

❌ Using non-TLS port

❌ Including protocol in hostname

❌ Redeclaring queue with wrong durability

❌ VPC mismatch

1️⃣4️⃣ One-Line Interview / Submission Answer

Two EC2 instances communicate through Amazon MQ by securely connecting to a RabbitMQ broker using TLS, durable queues, and broker-level authentication.