



AWS  
re:Invent

**API 313**

# Nondisruptive strategies for application migration

**Trevor Dyck**

Senior Manager, Product Management  
Amazon Web Services

# Agenda

Migration considerations and challenges

Migrating existing message-oriented middleware

Step 1: Can you migrate?

Step 2: Proof of concept

Step 3: Nondisruptive migration (and demo)

# Related breakouts

**API202-R** Building a bridge solution from IBM MQ to Amazon MQ

**API307** Build efficient and scalable distributed applications using Amazon MQ

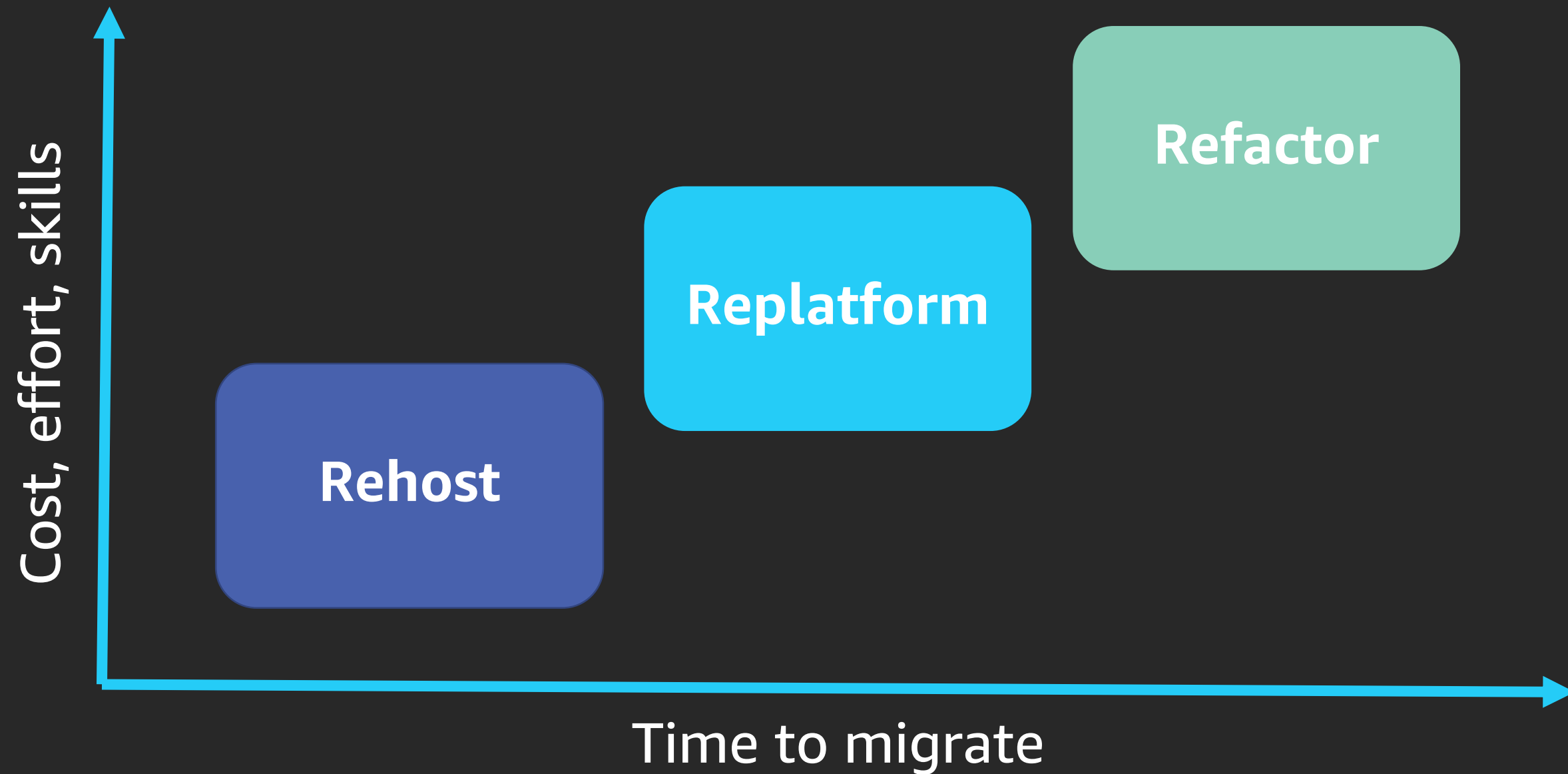
**API312** How to select the right application-integration service

# Migration considerations and challenges

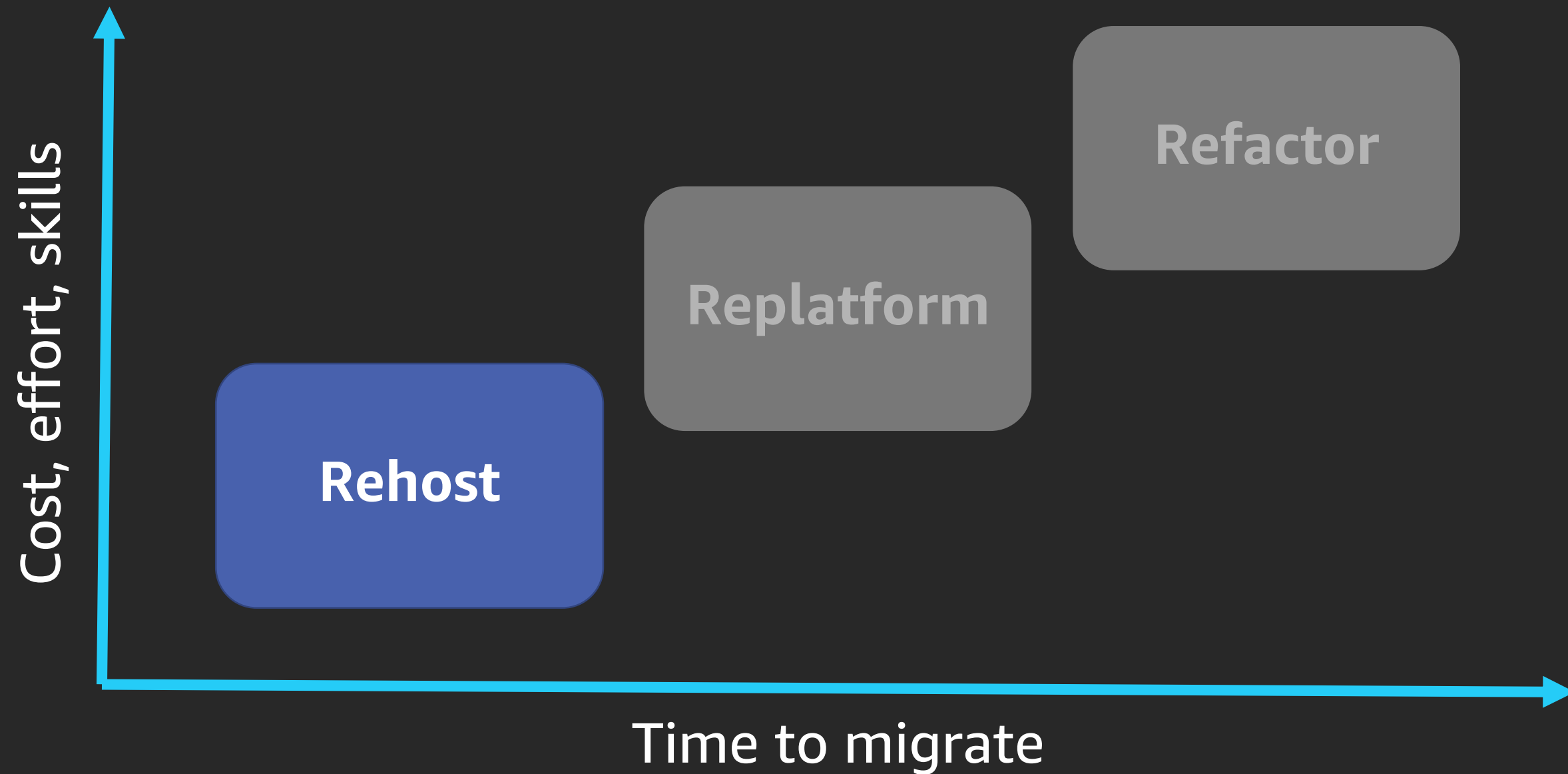
# Key considerations when migrating applications to the cloud

- Business goals
  - e.g., reduce cost, increase productivity and agility, increase scale, reduce operational overhead
- Application capacity
  - Number of users, amount of traffic
- Costs of migration

# Migration strategies: Rehost, replatform, or refactor

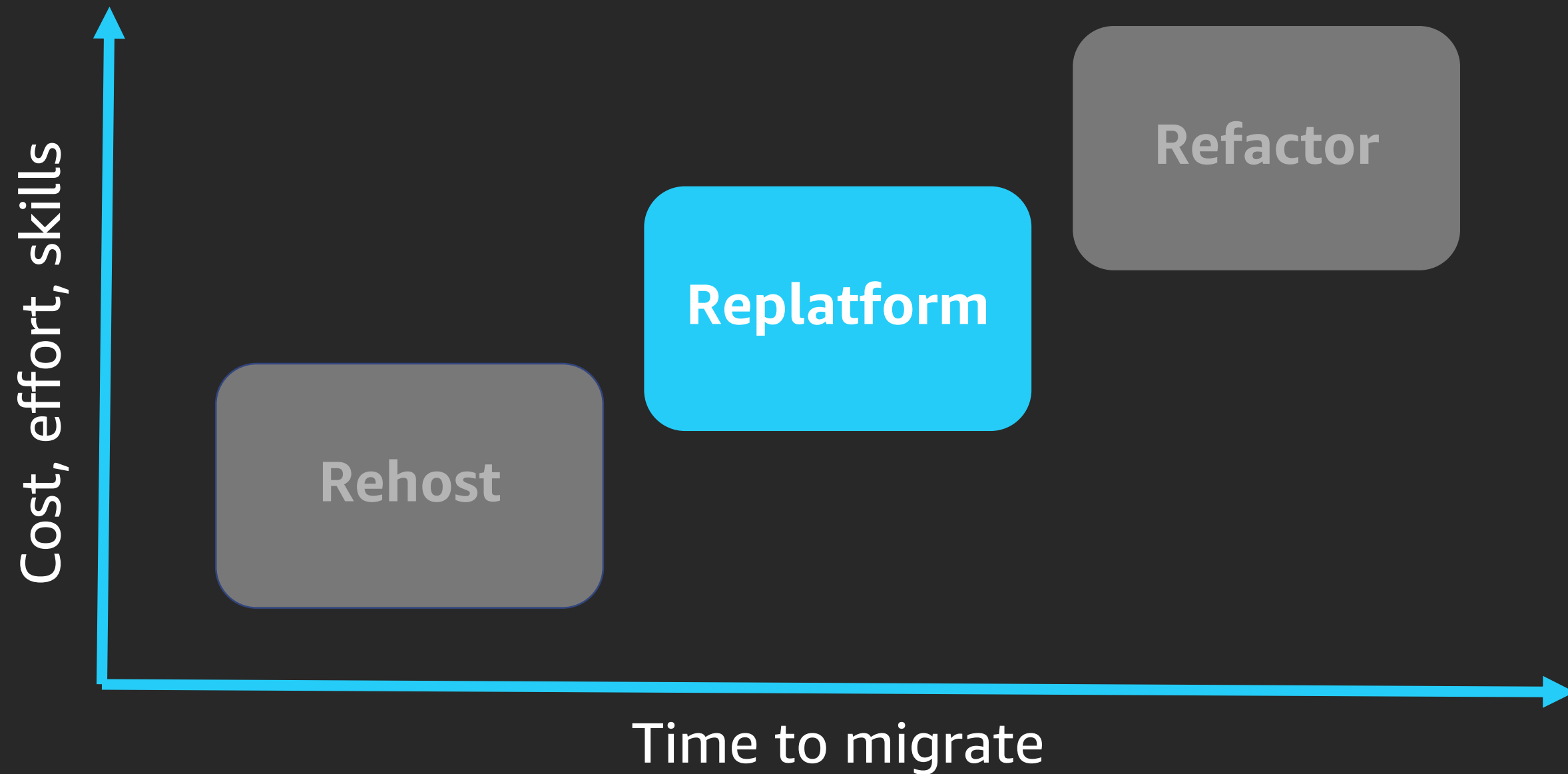


# Migration strategies: Rehost

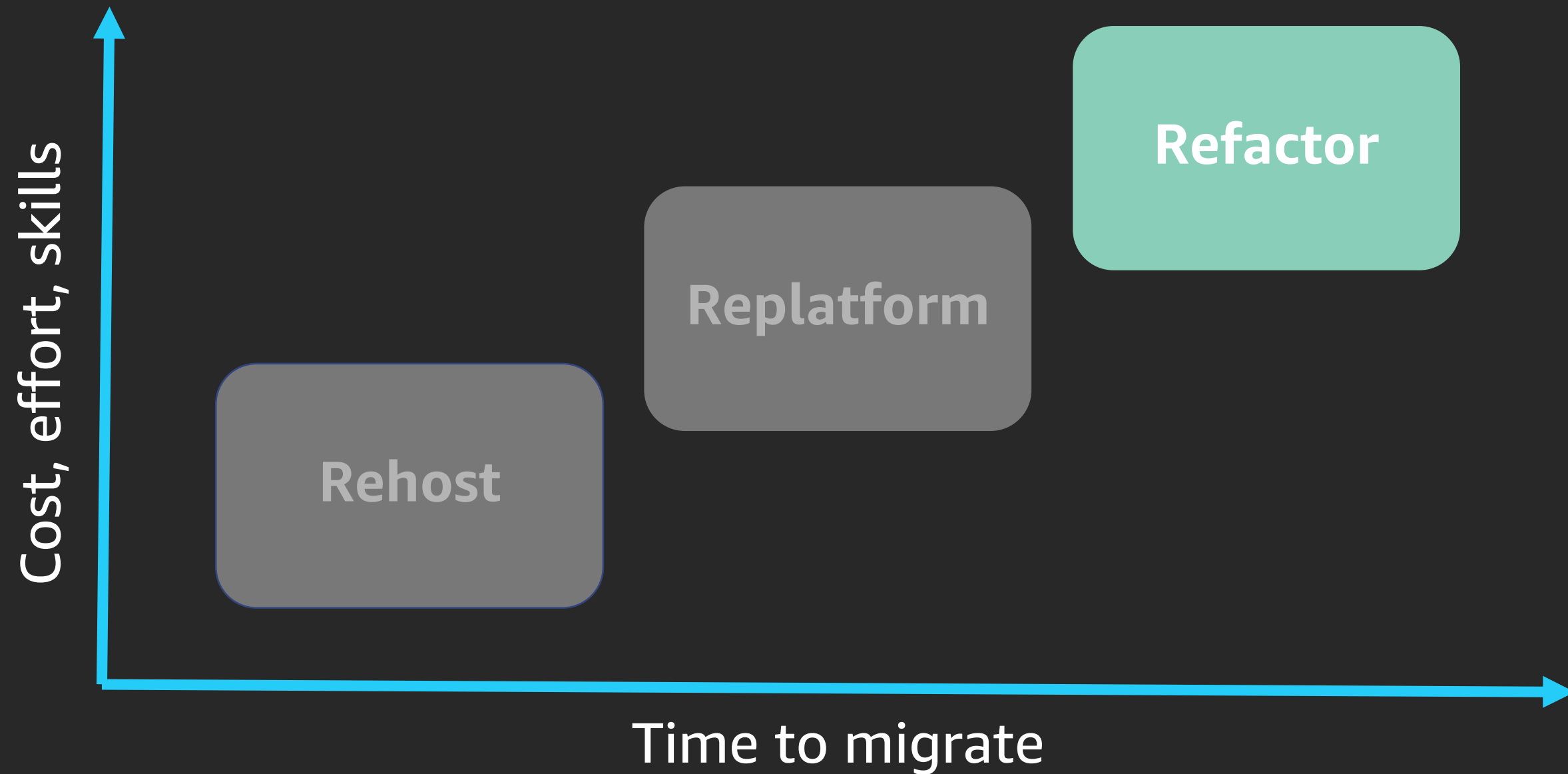




# Migration strategies: Replatform



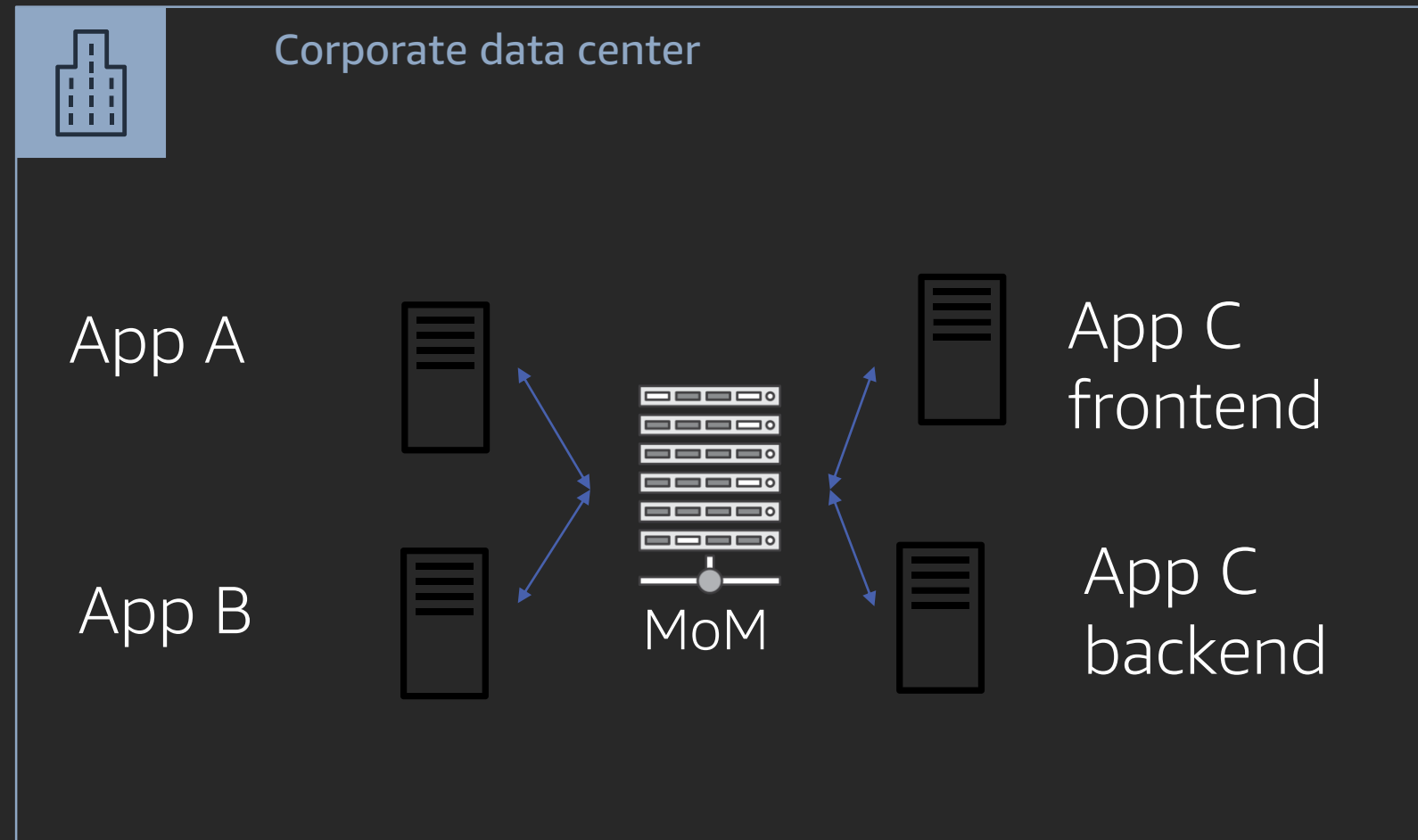
# Migration strategies: Refactor



# What about your application integration layer?

- Migrating apps also entails migrating the middleware they are using to integrate:
  - APIs
  - Orchestration/workflow
  - Messaging

# Enterprise messaging



Message-oriented middleware (MoM) or message broker

# Migrating existing message-oriented middleware

# Traditional message-oriented middleware (MoM)

- Many applications running on-premises today use:
  - IBM WebSphere MQ
  - TIBCO EMS
  - RedHat JBoss A-MQ
  - RabbitMQ
- Many others exist: Oracle AQ, MSMQ, SonicMQ, etc.

# Migration strategies for message brokers:

## Rehost

- Run your existing message broker (for example IBM MQ or RabbitMQ) in AWS
- Run on Amazon Elastic Compute Cloud (Amazon EC2) or Amazon Elastic Container Service (Amazon ECS)
- Options include **AWS Quick Starts** for deployment, **AWS Marketplace** offerings
- Infrastructure is now managed, but you still need to manage your broker

# Migration strategies for message brokers:

## Replatform

- **Amazon MQ** provides an API-compatible, managed message broker service
- Offload broker management to Amazon, reduces your operational overhead and cost
- Less disruptive than a refactor, minimal (or zero) code changes

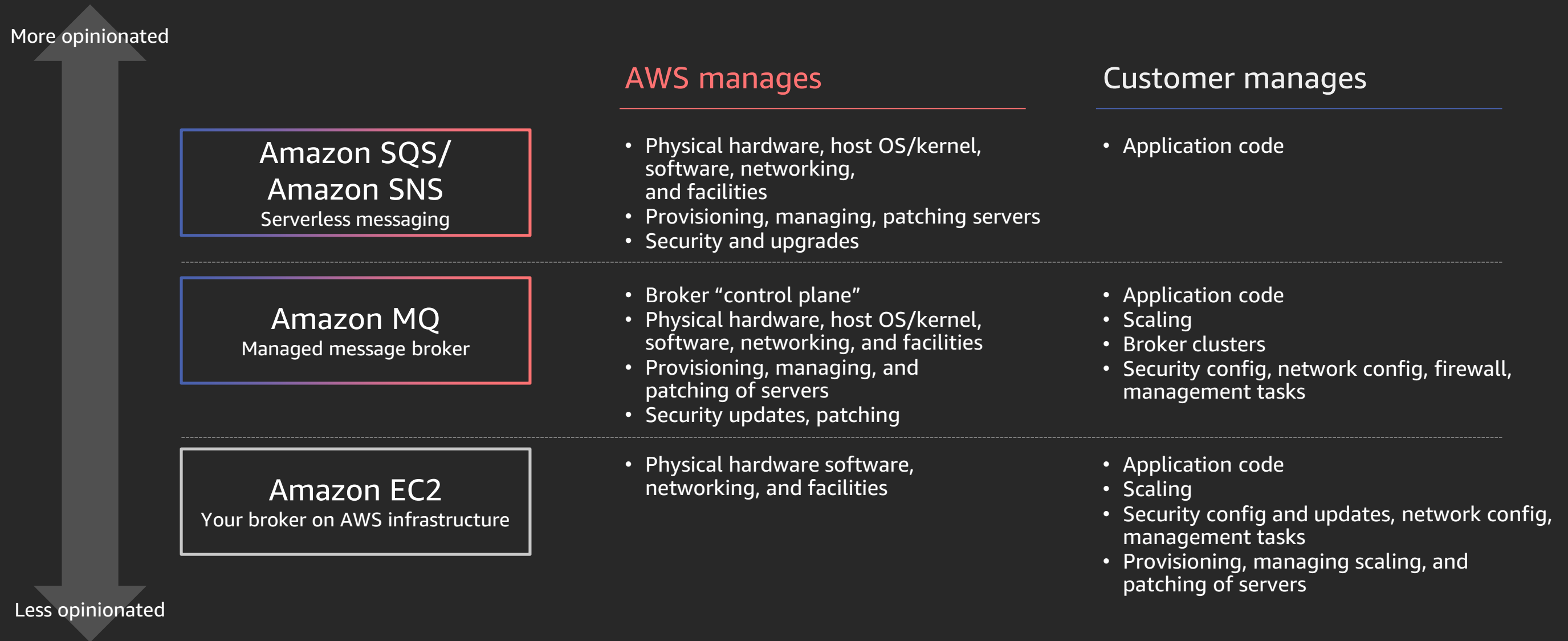


# Migration strategies for message brokers:

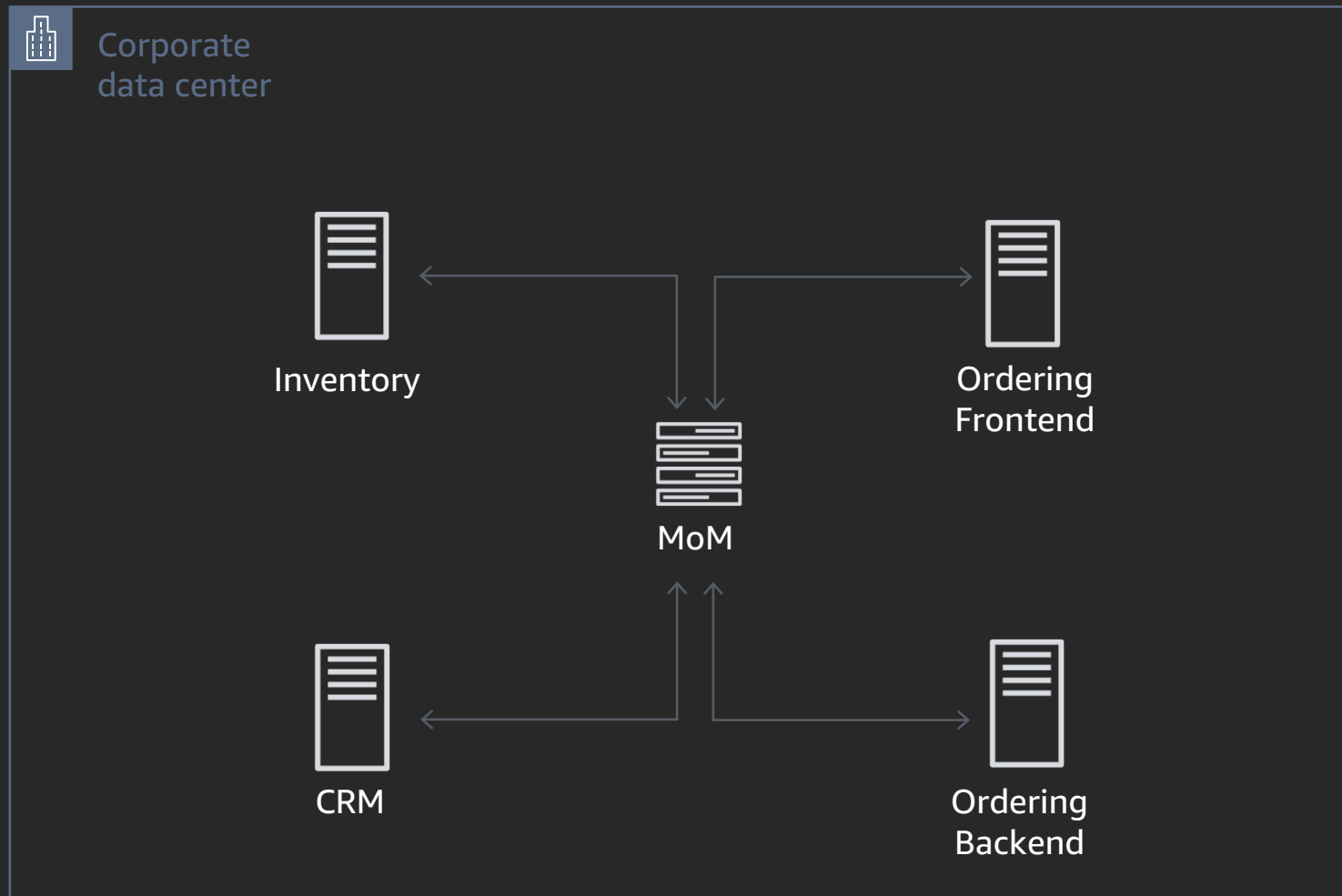
## Refactor

- Rewrite portions of your applications to use serverless messaging: **Amazon Simple Queue Service (Amazon SQS)** and **Amazon Simple Notification Service (Amazon SNS)**
- Consider this if you are refactoring your applications for serverless (e.g., using **AWS Lambda**)
- Even less operational overhead (e.g., scales nearly infinitely), but up-front investment to refactor code

# Comparison of operational responsibility



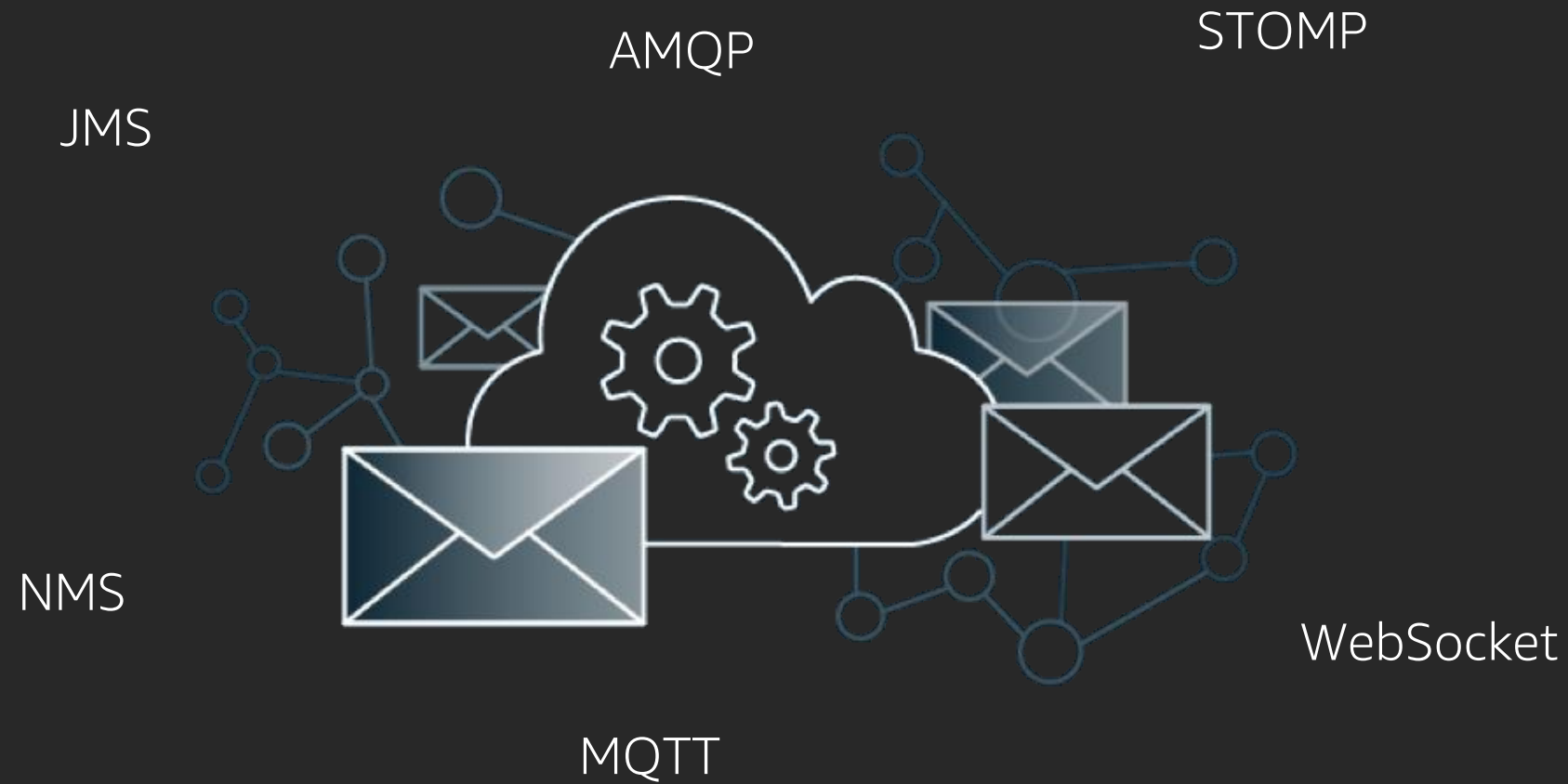
# Mission-critical enterprise app considerations



- Been around for years
- Critical to business
- High risk to modify
- Heavy investments in people, licensing, support
- Difficult to manage
- Commercial and/or open source

# **Nondisruptive strategy: Replatform with Amazon MQ**

# Amazon MQ: Migrate without modifying code




Fully managed, **open-source** Apache ActiveMQ  
Compatible with **industry-standard** APIs and protocols

# Amazon MQ: Compatible with key MoM features



Queues and topics  
(with FIFO)



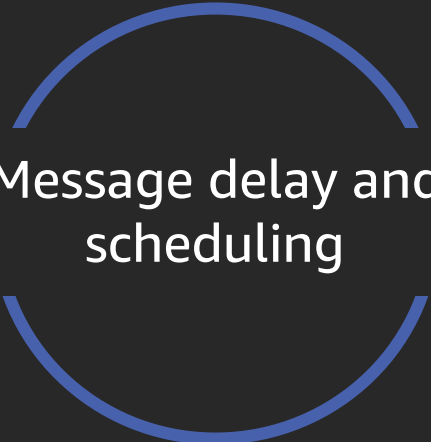
Transient and  
persistent messages




Large message sizes




Message filtering



Message delay and  
scheduling



Local and distributed  
transactions (XA)

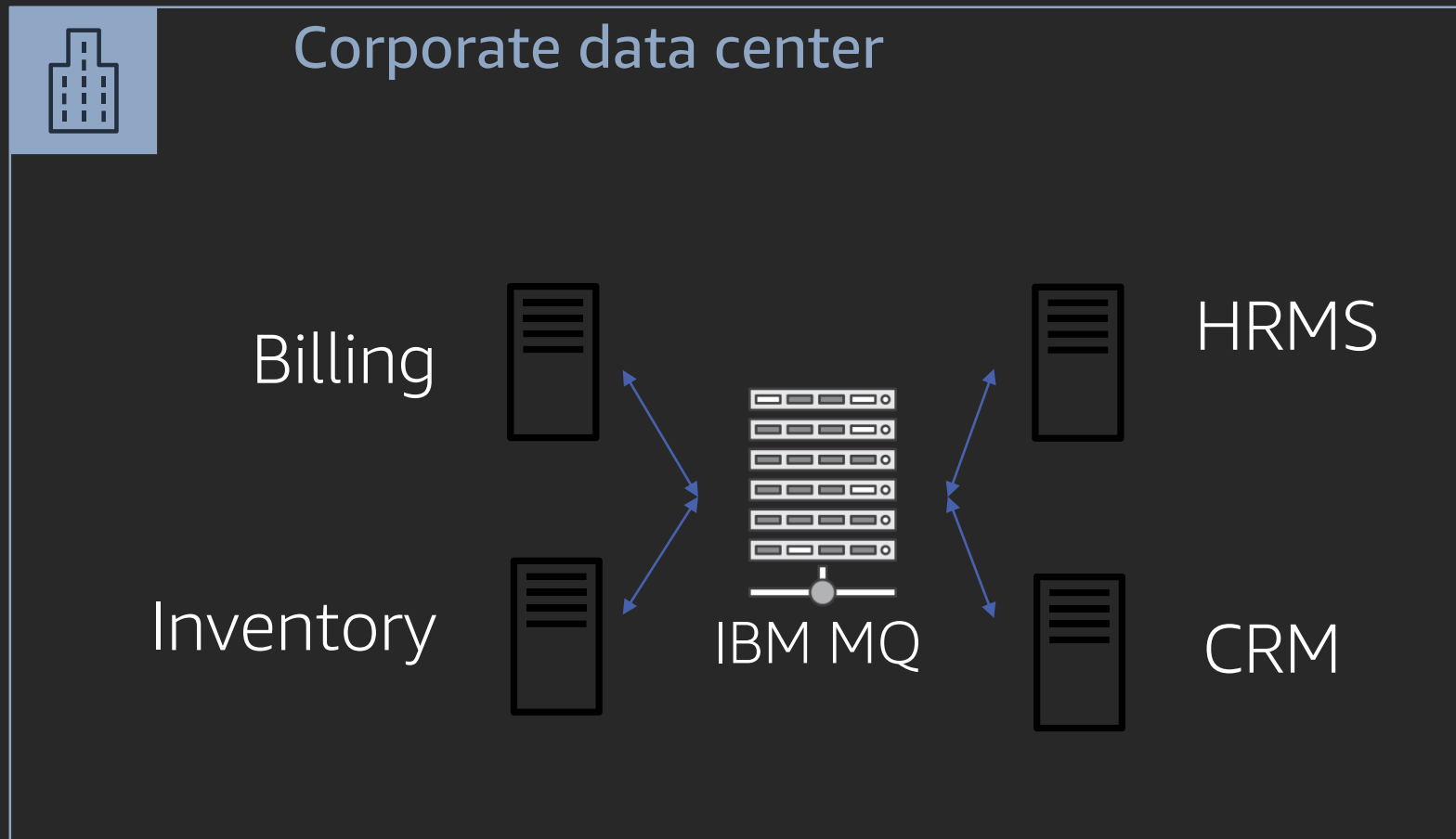


Virtual and composite  
destinations



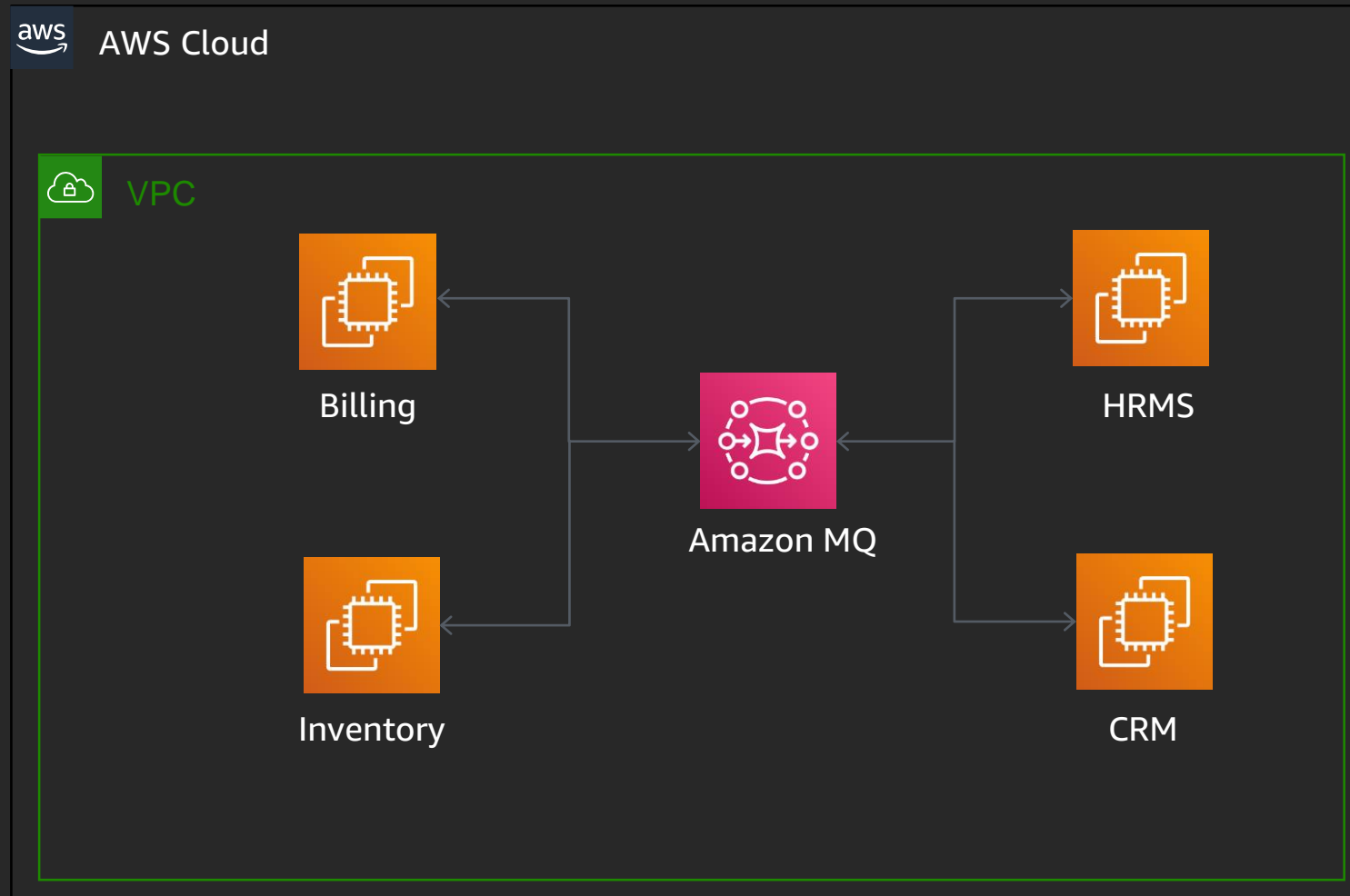
Redelivery policy

# Case study: Example Corp.



- Migrating from data center to AWS
- Legacy apps (billing, inventory, etc.)
- Building new apps in cloud (e.g., analytics)
- Migrate to managed services where possible

# Replatform: Replace with managed broker



- + Minimal code change
- + AWS manages messaging system
- + Highly available broker
- + Reliable message storage
- + Better overall reliability
- + No expensive licenses



# Step 1: Can you migrate?

# Assess your existing messaging needs

- APIs/protocols (JMS, AMQP, WebSocket, STOMP, MQTT)
- Use cases (e.g., basic queuing, pub-sub, transactions, request-reply)
- Message ordering
- Availability and durability (how tolerant is your system to message loss)
- Performance (connections, throughput, latency)
- Security and compliance
- Monitoring and logging
- New requirements that you didn't need on-prem (e.g., security, compliance)?

# What if all requirements are not met?

- Concepts in existing system may have different names (e.g., RabbitMQ: “exchanges” vs. ActiveMQ “virtual topic”)
  - Make sure to map current concepts to new terminology
- Are the features still needed? Sometimes there are legacy features that are not needed anymore
- May dictate need to rehost; lift-and-shift your existing broker

# Case study: Example Corp. – Current on-prem situation

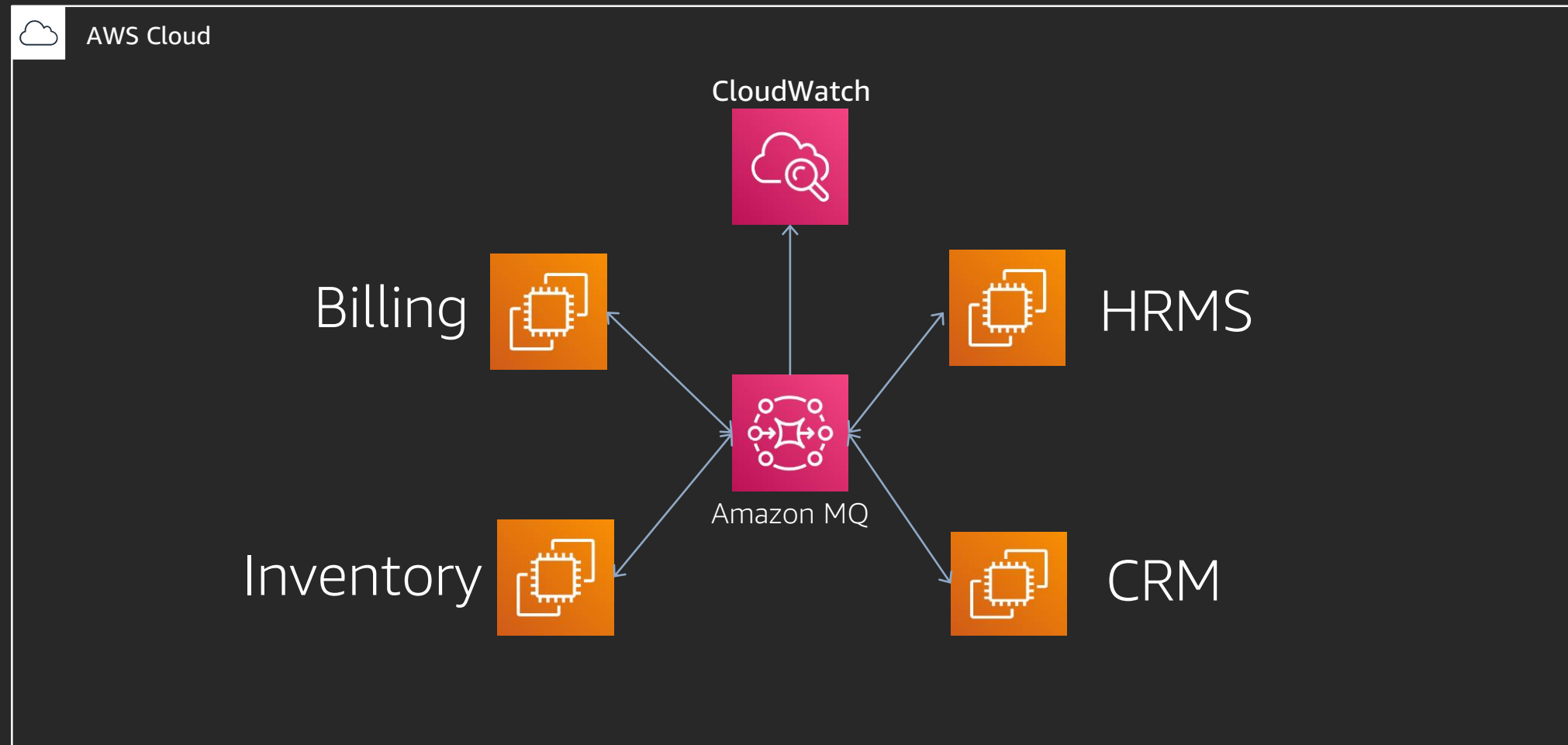
- Four applications using IBM MQ.
- JMS 1.1 APIs used to send/receive messages.
- 20 queues (two require FIFO ordering).
- Messages can't be lost.
- Can't have any downtime during migration. No SLA internally.
- Total throughput 1,200 messages per second.
- No encryption on-prem (*new* mandate in cloud).
- Healthcare company (HIPAA compliance).
- Use JMX-based tools for monitoring (Nagios).

# Mapping requirements to Amazon MQ

- ✓ JMS 1.1 compatible
- ✓ FIFO (ordered) queues
- ✓ Persistent multi-AZ storage, 99.9% durability
- ✓ Active/standby with 99.9% uptime SLA
- Performance?
- ✓ Mandatory TLS on all connections
- ✓ Encrypted storage with CMK
- ✓ HIPAA
- ✓ Amazon CloudWatch for monitoring

# Step 2: Proof of concept

# Proof of concept



# Proof of concept

Goal – Ensure applications work with new message broker (Amazon MQ)

1. Install on-prem applications first in AWS
2. Create an Amazon MQ broker
  - a. mq.m5.2xlarge for 1,200 messages per second
  - b. Active/Standby mode for high availability
3. Modify app configurations to use JMS endpoint in Amazon MQ instead of IBM MQ (one at a time)
4. Run applications and ensure messages are flowing
5. Use CloudWatch to monitor message flow through queues
6. Test performance



# Performance considerations when migrating

- Need to consider both throughput (mgs/sec), latency (msec) , and number of connections
- Common issue: “I got X mps on-prem and I get Y mps with Amazon MQ” ( $Y < X$ )
- Why?
  - On-prem may have simple storage solution (e.g., RAID array in a single rack)
  - Customers running on EC2 are often using Amazon Elastic Block Store (Amazon EBS) storage (single Availability Zone)
  - Amazon MQ uses Amazon Elastic File System (Amazon EFS), which replicates across multiple AZs (physical facilities) and is designed for eleven nines durability. This comes at some performance trade-off.
- How fast is an Amazon MQ message broker?

It depends...



# Example Amazon MQ performance (guideline only)

Producers/consumers	Throughput (msgs/sec.)	Latency (msec)
25	1,000–2,000	5–10
50	3,000–4,000	5–10
100	7,000–8,000	5–10
200	15,000–16,000	5–10

Note: using mq.m5.2xlarge, 1 KB messages, persistent mode, OpenWire protocol, `concurrentStoreAndDispatchQueues` = TRUE.

You must test with your specific scenario.

<https://aws.amazon.com/blogs/compute/measuring-the-throughput-for-amazon-mq-using-the-jms-benchmark>

# Tuning performance

- Nonpersistent mode (memory only) much faster than persistent mode, if your application can tolerate potential message loss
- Binary protocols (OpenWire) faster than text-based (STOMP, MQTT)
- `concurrentStoreAndDispatchQueues` flag
- Connections: Default limit 1,000 per broker, but you can request a service limit increase (larger instances can handle more)
- Network of brokers can be used to horizontally scale, distributing the load across more nodes (if message ordering is not required)

# Slow consumers

- ActiveMQ (and Amazon MQ by extension) performs best with fast consumers

*fast* = able to keep up with the rate of messages generated by producers

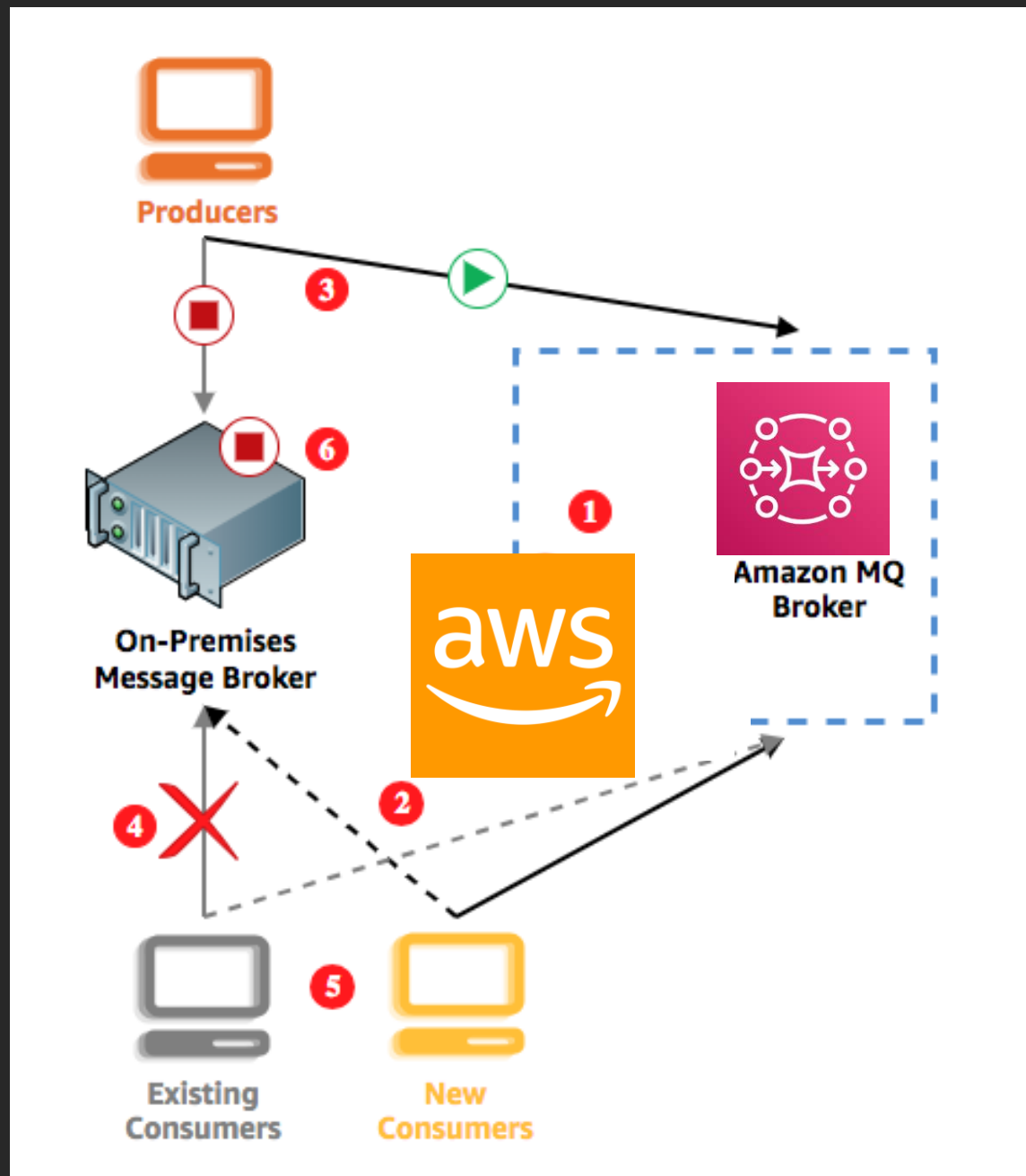
*slow* = queue builds up a backlog of unacknowledged messages, potentially causing a decrease in producer throughput.

- To get the best performance with *slow* consumers set the `concurrentStoreAndDispatchQueues` parameter to false

```
<persistenceAdapter>  
<kahaDB concurrentStoreAndDispatchQueues="false"/>  
</persistenceAdapter>
```

# Step 3: Nondisruptive migration

# Migrating without service interruption



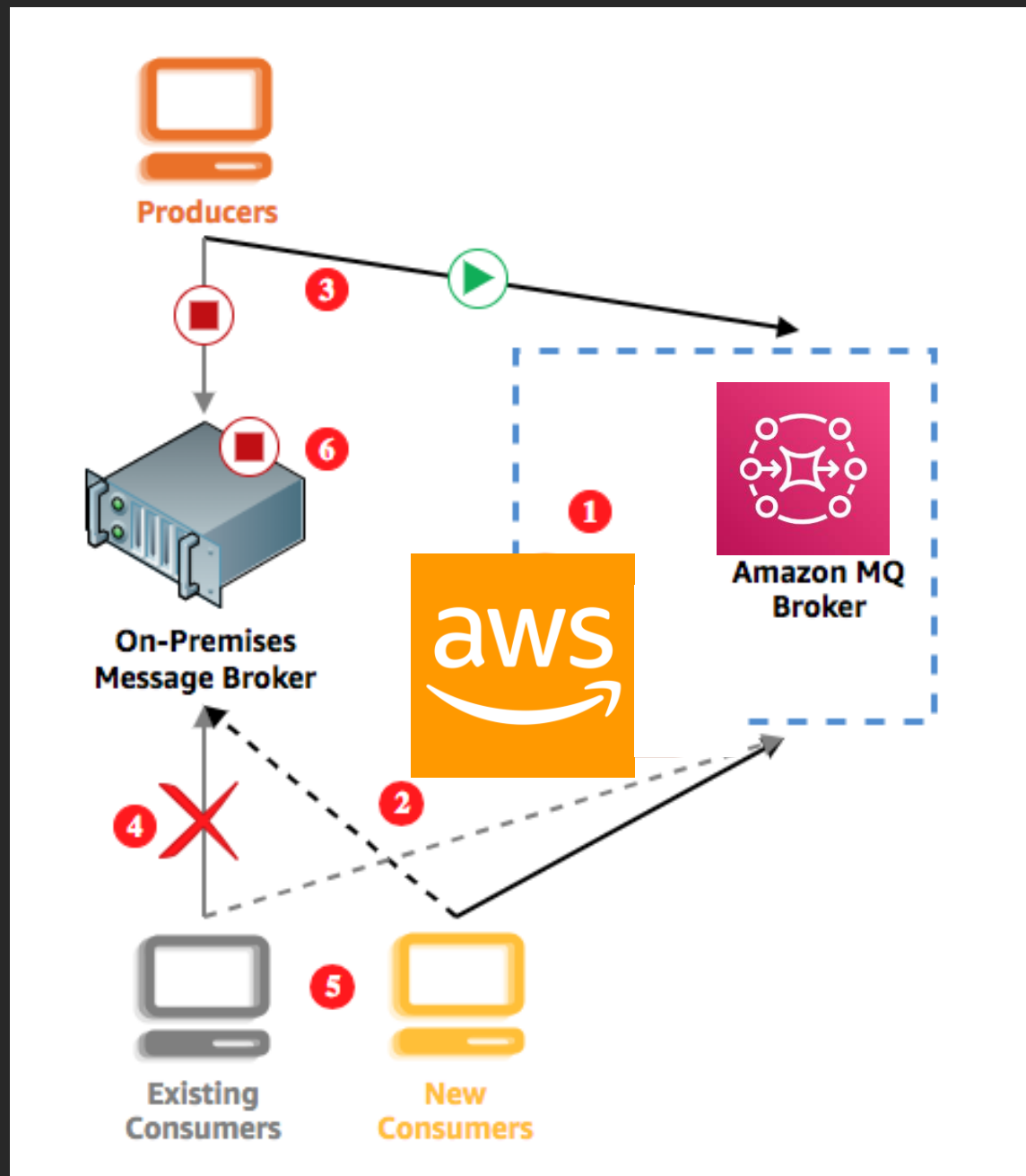
1. Create Amazon MQ broker

2. Modify consumers (or create new consumers) to consume from both on-prem and Amazon MQ endpoints

Example with ActiveMQ:

```
failover:(ssl://old:61617,  
ssl://new:61617)
```

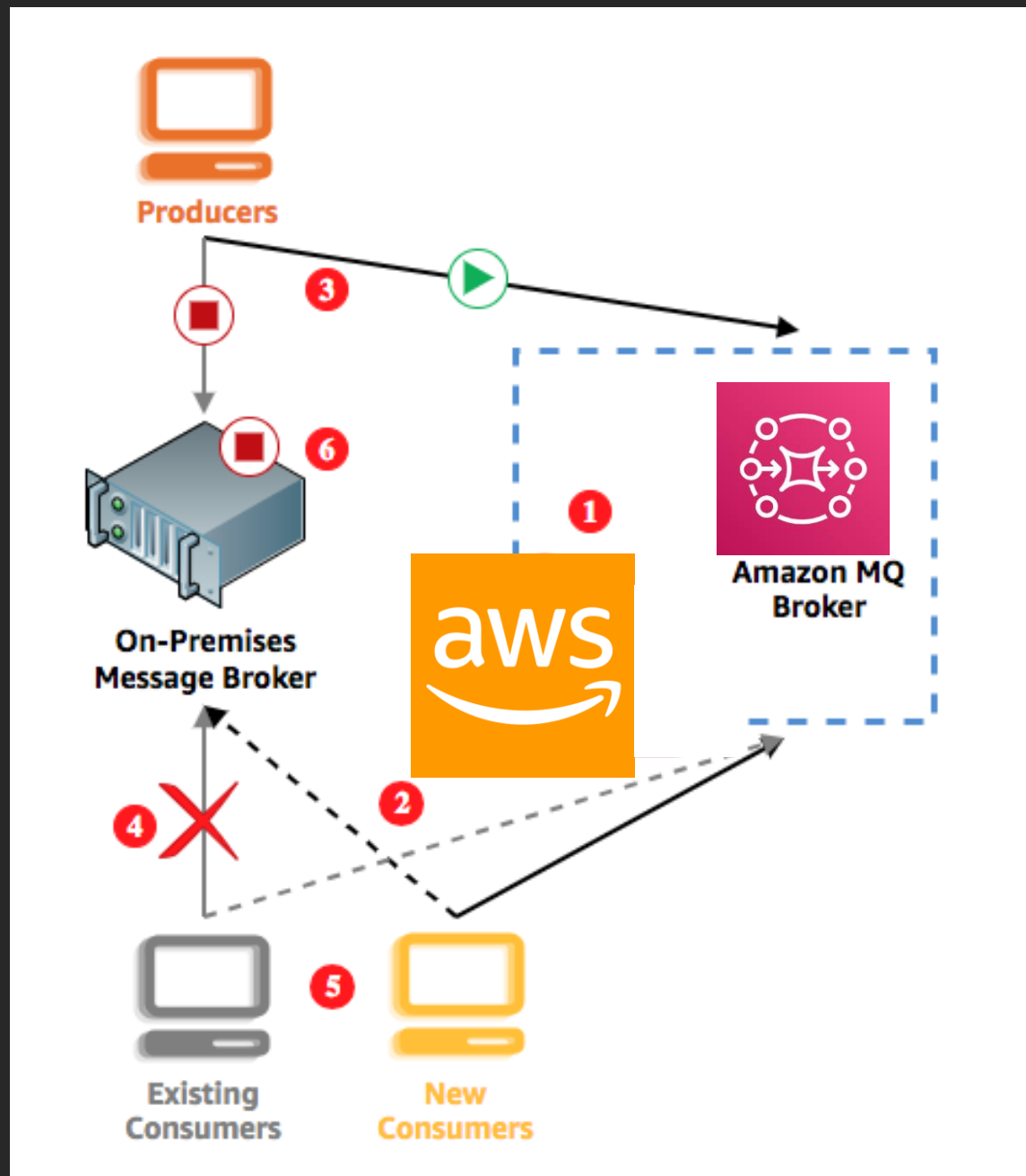
# Migrating without service interruption



3. Stop each existing producer, point the producer to the new broker's endpoint, then restart the producer
4. Wait for your consumers to drain the destinations on your on-premises broker



# Migrating without service interruption



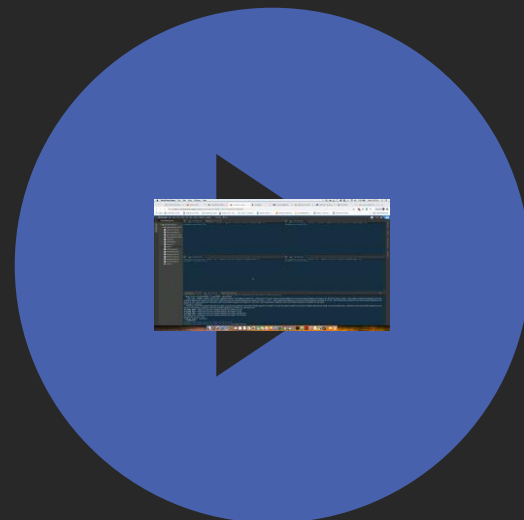
5. Stop your on-premises broker

6. Change your consumers' failover transport to include only your Amazon MQ broker's endpoint

Example with ActiveMQ:

`failover:(ssl://new:61617)`

# Demo

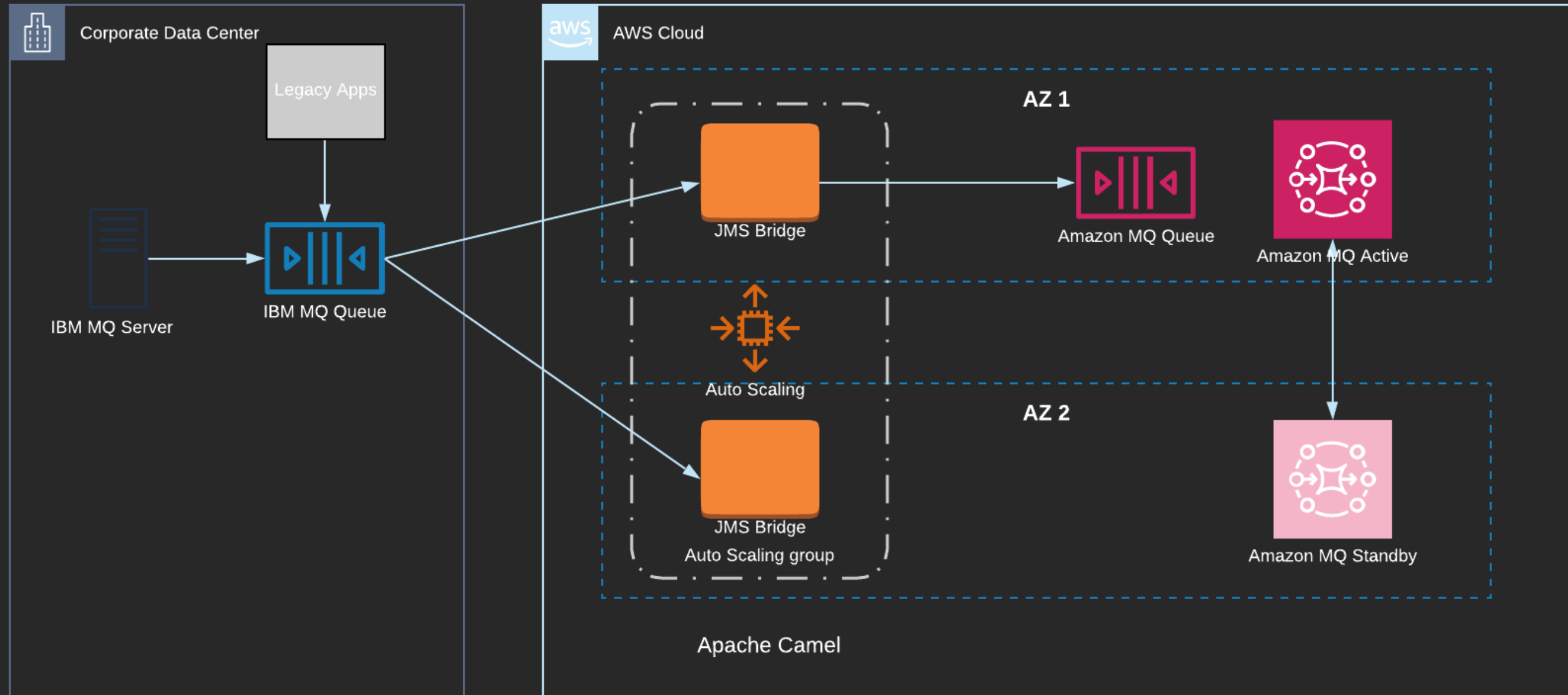


# Hybrid architectures

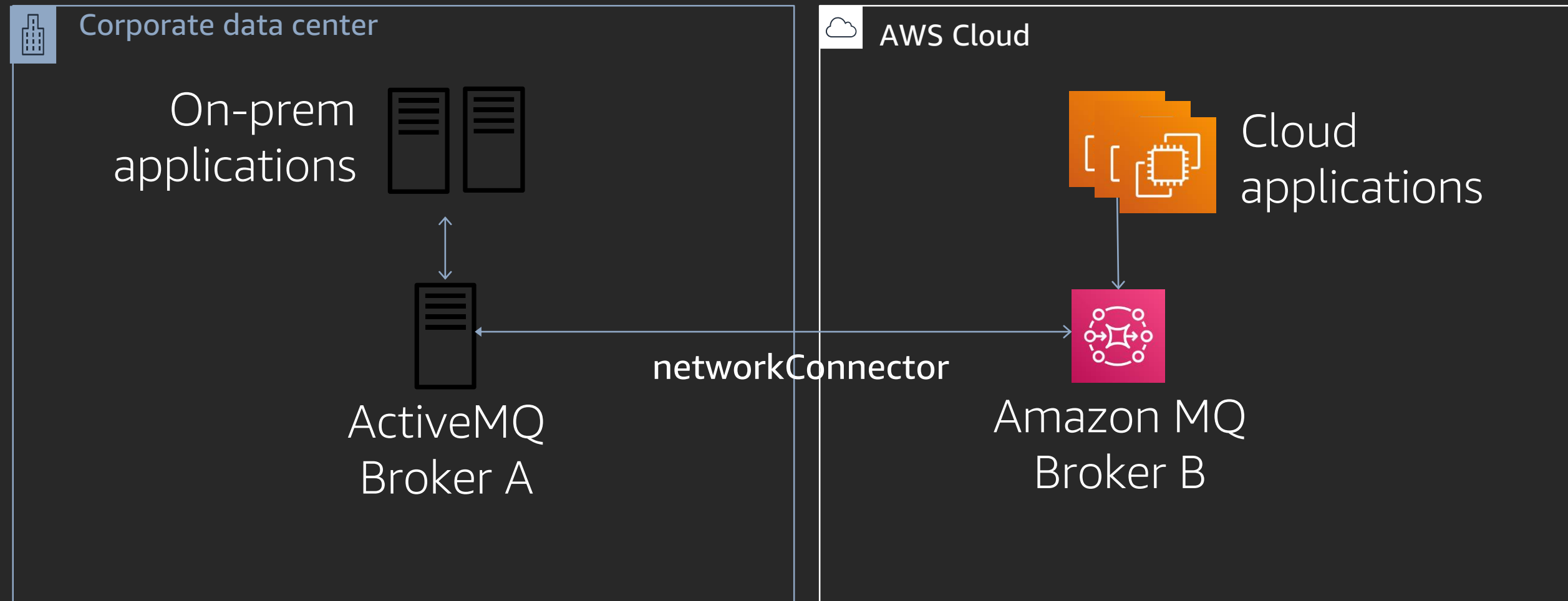
# Hybrid solution to connect existing with modern

- Incrementally migrate applications
- AWS manages messaging system
- Highly available broker
- Durable message store
- Better overall reliability
- Create new cloud apps
- Integrate with on-premises systems

# AnyCompany use case: Hybrid solution to connect on-prem to cloud



# Hybrid model: Network of brokers



<https://aws.amazon.com/blogs/compute/running-activemq-in-a-hybrid-cloud-environment-with-amazon-mq/>

# Hybrid model: Network of brokers

- Network of brokers supports distributed queues/topics
- *networkConnectors* are configured between brokers, can be unidirectional or bidirectional
- Client can connect to any broker in the network, brokers intelligently forward to wherever consumers are present
- Other brokers act like consumers when bridged via networkConnector
- Useful for hybrid: forward messages from on-prem to cloud (or vice versa, if needed)
- Can migrate gradually by turning up cloud apps then turning down on-prem apps



# Sample broker configuration: Network

```
<networkConnector
    name="Q:hybridconnector"
    duplex="true"
    uri="static:(ssl://b-foo.mq.us-east-1.amazonaws.com:61617)"
    userName="username"
    password="password"
    networkTTL="2"
    dynamicOnly="false">
    <staticallyIncludedDestinations>
        <queue physicalName="queuenam" />
    </staticallyIncludedDestinations>
</networkConnector>
```

<https://aws.amazon.com/blogs/compute/running-activemq-in-a-hybrid-cloud-environment-with-amazon-mq/>

“With Amazon MQ we now have a clear path toward an iterative cloud migration that would have been a challenge with an on-prem solution. Migrating consumers and producers individually will be easier given that the queues are already in AWS.”

**Kevin Thorley**

Enterprise Architect, Dealer.com

# More information

- Blog: “Migrating from RabbitMQ to Amazon MQ”
- <https://aws.amazon.com/blogs/compute/migrating-from-rabbitmq-to-amazon-mq/>
- Bench Accounting: “From ActiveMQ To Amazon MQ: Why And How We Moved To AWS’s Managed Solution”
- <https://medium.com/bench-engineering/from-activemq-to-amazon-mq-why-and-how-we-moved-to-awss-managed-solution-afeba3ea7e23>

# Thank you!



Please complete the session  
survey in the mobile app.