Amazon MQ Migration Guide

Amazon MQ

Amazon MQ: Amazon MQ Migration Guide

Table of Contents

What is the Amazon MQ migration guide?

Amazon MQ is a managed message broker service that makes it easy to migrate to a message broker in the cloud. Amazon MQ currently supports Apache ActiveMQ and RabbitMQ engines. Amazon MQ for ActiveMQ simplifies the migration of commercial brokers, such as IBM MQ and TIBCO Enterprise Management Service (EMS), to the cloud. Amazon MQ for ActiveMQ brokers are compatible with popular APIs and protocols, such as Java Message Service (JMS), allowing you to migrate applications with minimal code changes. Amazon MQ for RabbitMQ offers cross-region data replication capabilities.

Concepts for migrating to Amazon MQ

Before migrating, review the following key concepts to consider when migrating a commercial message broker to Amazon MQ.

Messaging protocols

You can connect your broker to Amazon MQ without any code changes if you currently use one of the following industry-standard protocols:

- AMQP
- MQTT
- MQTT over WebSocket
- OpenWire
- STOMP
- STOMP over WebSocket

For more information about connecting to an Amazon MQ managed broker, see Working Examples of Using Java Message Service (JMS) with ActiveMQ in the *Amazon MQ Developer Guide*.

Message persistence

To replicate *persistence mode* or *sync point control* options with Amazon MQ, you can deploy your brokers as active/standby brokers. In the active/standby deployment, brokers use shared storage across multiple Availability Zones, with an optional time to live (TTL).

For more information about how Amazon MQ ensures message durability, see the section called "Availability options".

Network options

Depending on the interlopability of your applications and the type of access that they need, you can permit public access, VPN access, or VPC access using Amazon Virtual Private Cloud (Amazon VPC).

In a *hybrid architecture* where on-premises systems need access to resources in the cloud, we recommend setting up your Amazon MQ managed brokers with *public network access*. You can also achieve a hybrid solution by using AWS VPN or AWS Direct Connect.

> **Tip**
>
> If your resources are primarily deployed within the AWS Cloud, we recommend configuring your Amazon MQ brokers with Amazon VPC. For network access across multiple VPCs, you can use VPC peering.

Availability options

Amazon MQ supports durability-optimized brokers backed by Amazon Elastic File System (Amazon EFS). You can configure single-instance brokers (one broker in one Availability Zone) or active/standby brokers (two brokers in two different Availability Zones). In either configuration, Amazon MQ can automatically provision infrastructure for high message durability by storing messages redundantly across multiple Availability Zones.

> **Note**
>
> In the event of a broker or Availability Zone failure, active/standby brokers automatically fail over to a standby instance in another Availability Zone.

To achieve high availability and message durability, you can use a network of brokers. A network of brokers is a series of simultaneously active single-instance or active/standby brokers that allows you to rapidly scale your throughput and connection count. You can configure a network of brokers in a variety of topologies depending on your application's needs.

Messaging patterns

Amazon MQ offers the following topology options to support a variety of messaging patterns:

- Point-to-point
- Request-response
- Hub and spoke
- Mesh
- Enterprise service bus

For more information about using Amazon MQ to set up the right broker topology for your cloud architecture, see Amazon MQ Broker Architecture in the *Amazon MQ Developer Guide*.

> **Important**
>
> Revising a broker configuration or an ActiveMQ user does not immediately apply those changes. For your changes to take effect, you must wait for the next maintenance window or reboot the broker. For more information, see Amazon MQ Broker Configuration Lifecycle in the *Amazon MQ Developer Guide*.

Performance and scalability

With Amazon MQ you can scale your messaging middleware horizontally, vertically, or in a hybrid model.

Horizontal scaling enables you to increase your throughput and connection count without interruptions, because your resources remain active and online. To scale horizontally, you can deploy a network of brokers in an active/standby configuration across multiple Availability Zones.

To scale your resources *vertically*, you can increase the compute capacity of your broker instances from `mq.t2.micro` (1 vCPU and 1 GiB) up to `mq.m5.4xlarge` (16 vCPU and 64 GiB). For more information about Amazon MQ instance types, see Instance Types in the *Amazon MQ Developer Guide*.

> **Note**
>
> Choosing larger broker instance types might not improve overall system throughput. Overall latency is due to many factors, such as message size, the type of protocol, the number of active producers and consumers, consumption speed, and message persistence.

Amazon MQ also supports creating throughput-optimized message brokers backed by Amazon Elastic Block Store (Amazon EBS). These brokers are ideal for applications such as high-volume order processing, stock trading, and text processing.

To instruct Amazon MQ to optimize for queues with slow consumers, set the `concurrentStorageAndDispatchQueues` attribute to `false`.

The following table shows the throughput of an `mq.m5.2xlarge` broker configured with these options:

- **Broker instance** – `mq.m5.2xlarge`
- **Persistent** – TRUE
- **Client** – `m5.xlarge`
- **CSAD** – TRUE
- **Protocol** – OpenWire

		Producers/Consumers			
Message size	**Metrics**	**25**	**50**	**100**	**200**
1 KB	TPS	2,250	4,300	8,467	16,334
	CPU%	8%	15%	27%	58%
5 KB	TPS	2,067	3,834	7,150	14,516
	CPU%	10%	17%	30%	63%
10 KB	TPS	1,900	3,467	7,083	11,334
	CPU%	15%	24%	48%	80%

		Producers/Consumers			
Message size	**Metrics**	**25**	**50**	**100**	**200**
50 KB	TPS	1,592	2,917	4,500	4,917
	CPU%	30%	52%	83%	92%
100 KB	TPS	1,250	2,184	2,513	2,770
	CPU%	42%	72%	85%	92%

> **Note**
>
> Performance numbers can vary depending on multiple configuration parameters. For more information on Amazon MQ throughput measurements, see Throughput Benchmarks.

You can measure the throughput of your Amazon MQ brokers using JMS Benchmark.

Latency

You can set up your Amazon MQ brokers for low-latency messaging, with latency often as low as single-digit milliseconds. Use an *always-on* connection to help reduce the amount of time that it takes to deliver messages to a consumer.

Using *in-memory* storage can further reduce overall latency across your messaging architecture. For more information on how different storage types can affect latency, see Differences Between Storage Types in the *Amazon MQ Developer Guide*.

Destination options

You can set up Amazon MQ managed brokers as *queues* or *topics*. Amazon MQ queues are, by default, *first in, first out (FIFO)* queues, also known as ordered queues. You can scale FIFO queues using Message Groups. You can configure your broker destinations with *at-least-once delivery*, *at-most-once delivery*, or *exactly-once delivery* options.

Topics in Amazon MQ use the *publisher/subscriber* pattern and can be durable or non-durable. Amazon MQ topics also support Virtual Destinations, where publishers broadcast messages to a pool of subscribers through queues. We recommend using this method instead of durable topics.

> **Tip**
>
> You can optimize and fine-tune the performance of your topics. For more information, see Performance Tuning on the Apache ActiveMQ website.

Security and authentication

With Amazon MQ you control who is allowed to create or modify brokers, and which applications are allowed to send and receive messages. For more information about authentication options, and how to integrate the Lightweight Directory Access Protocol (LDAP) with your Amazon MQ brokers, see Messaging Authentication and Authorization for ActiveMQ in the *Amazon MQ Developer Guide*.

Connections to Amazon MQ brokers use Transport Layer Security (TLS). To isolate your brokers in a private virtual network, you can restrict access to a private endpoint within a VPC. To control network access to your brokers, you can configure security groups in the VPC. For more information, see Security Best Practices for Amazon MQ in the *Amazon MQ Developer Guide*.

Amazon MQ encrypts messages at rest and in transit using encryption keys that it manages and stores securely in AWS Key Management Service (AWS KMS). AWS KMS helps reduce the operational burden and complexity involved in protecting sensitive data. With encryption at rest, you can build security-sensitive applications that meet encryption compliance and regulatory requirements.

> **Tip**
>
> For additional security, we highly recommend designing your application to use client-side encryption.

For more information about Amazon MQ security and how messaging data is encrypted, see Data Protection in Amazon MQ in the *Amazon MQ Developer Guide*.

Broker quotas

By default, each Amazon MQ broker can support 1,000 connections (or 100 connections for `mq.t2.micro` brokers). To allow multiple consumers to share connections to your Amazon MQ brokers and to improve overall performance, we recommend using *pooled connections*. For more information, see Always Use Connection Pooling in the *Amazon MQ Developer Guide*.

You can request an increase for many broker usage quotas for your AWS account. For more information, see AWS service quotas in the *AWS General Reference*.

Configuration options

Amazon MQ supports standard JMS features including point-to-point (message queues), publish-subscribe (topics), request/reply, persistent and non-persistent modes, JMS transactions, and distributed (XA) transactions.

Amazon MQ brokers can also support more complex messaging patterns such as composite destinations, which enable producers to send the same message to multiple destinations, and virtual destinations, which enable publishers to broadcast messages via a topic to a pool of receivers subscribing through queues.

For more information, see Amazon MQ Broker Configuration Parameters in the *Amazon MQ Developer Guide*.

Cost estimation

With Amazon MQ, you pay only for the provisioned capacity that you use. Factors such as broker instance type, the amount of data stored on each broker instance, and the AWS Region in which you deploy your brokers can affect your total cost of ownership. To estimate your broker costs for Amazon MQ, you can us the AWS Pricing Calculator.

> **Note**
>
> For data transferred in and out of Amazon MQ, you pay standard AWS data transfer charges.

To get started, Amazon MQ offers a Free Tier, which includes up to 750 hours of a single-instance `mq.t2.micro` or `mq.t3.micro` broker per month, and up to 5 GB of durability-optimized storage

per month for one year. For more information on the Free Tier, pricing, and associated costs, see Amazon MQ Pricing.

Options for migrating to Amazon MQ

The first step in migrating your commercial message broker to Amazon MQ is determining the right migration approach for your existing application architecture. The following are the most common approaches to migration:

Topics

Rehost

The rehost ("lift and shift") approach is ideal for moving on-premises workloads to the cloud when time is critical and ambiguity is high. Rehosting is the process of moving your existing applications from one infrastructure to another. The most significant benefit of this approach is that it enables you to migrate without changing large portions of application code. Fewer changes can also reduce the amount of re-training needed for engineers.

Replatform

The replatform strategy involves moving applications almost as-is, but replacing some components to take advantage of a cloud architecture. Here you might make a number of cloud-based optimizations to achieve some tangible benefit, but you aren't changing the core architecture of your application. You may be looking only to reduce the amount of time that you spend managing your broker.

Refactor (re-architect)

Refactoring (also known as "re-architecting") maximizes the benefits of moving to the cloud. Refactoring requires research, planning, experimentation, implementation, and deployment. These efforts generally provide the greatest rate of return in the form of reduced hardware and storage costs, less operational maintenance, and the most flexibility to meet future business needs. In

many cases, it involves breaking up the application into independent services and transitioning to a microservices architecture.

Phased migration

If you are interested in a phased, incremental migration approach, we recommend using a JMS proxy implementation such as Camel. For example, you can use the JMS Bridge Sample project, which allows you to bridge from your existing on-premises messaging broker to Amazon MQ.

You can also use enterprise integration patternssample to learn how to use Apache Camel and Amazon MQ to implement common patterns for routing, message transformation, and integration with other AWS services. In the sample, you will use Amazon EKS to scale Apache Camel. You can apply the same approach to migrate from IBM MQ or TIBCO EMS to Amazon MQ.

Migrating to Amazon MQ without service interruption

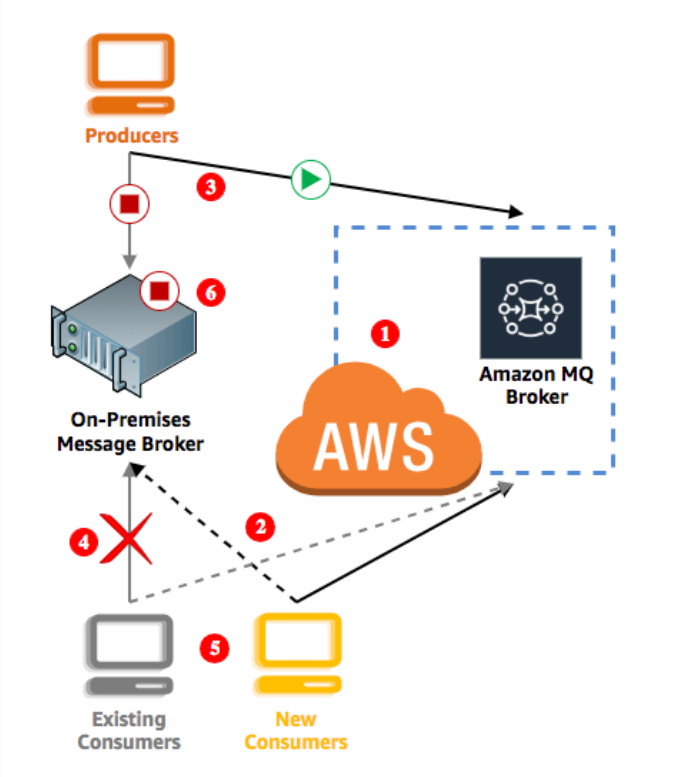
Use the following topics to learn more about potential service interruptions before migrating your on-premises message broker to Amazon MQ.

Migrating to Amazon MQ without service interruption

You can migrate from an on-premises message broker to an Amazon MQ broker in the AWS Cloud without service interruption.

> **Important**
>
> This scenario might cause messages to be delivered out of order.

The following diagrams illustrate the scenario of migrating from an on-premises message broker to an Amazon MQ broker in the AWS Cloud without service interruption.

On-Premises Message Broker	Migration to Amazon MQ with Standard (Unordered) Queues
	

To migrate to Amazon MQ without service interruption

1. Create and configure an Amazon MQ broker and note your broker's endpoint, for example:

```
ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617
```

2. For either of the following cases, use the Failover Transport to allow your consumers to randomly connect to your on-premises broker's endpoint or your Amazon MQ broker's endpoint. For example:

```
failover:(ssl://on-premises-broker.example.com:61617,ssl://
b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617)?
randomize=true
```

Do one of the following:

- One by one, point each existing consumer to your Amazon MQ broker's endpoint.
- Create new consumers and point them to your Amazon MQ broker's endpoint.

> **Note**
>
> If you scale up your consumer fleet during the migration process, it is a best practice to scale it down afterward.

3. One by one, stop each existing producer, point the producer to your Amazon MQ broker's endpoint, and then restart the producer.

4. Wait for your consumers to drain the destinations on your on-premises broker.

5. Change your consumers' Failover transport to include only your Amazon MQ broker's endpoint. For example:

```
failover:(ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-
east-2.amazonaws.com:61617)
```

6. Stop your on-premises broker.

Migrating from IBM MQ queue manager to Amazon MQ

You can migrate you IBM MQ system to Amazon MQ while keeping the same configuration.

Terminologies

The following table is a list of common IBM MQ concepts, and how they relate to Amazon MQ.

IBM MQ	**Amazon MQ**		
Component	**Description**	**Component**	**Description**
Standard Queue Manager	Standard Queue Manager is similar to a broker but one server can have multiple IBM MQ Queue Managers	Broker	Broker in Amazon MQ is equivalent to IBM MQ Queue Manager.
Gateway Queue Manager	A gateway Queue Manager is used to balance the workload. by routing traffic across multiple IBM MQ Queue Managers.	Network of Brokers	Amazon MQ provides Network of Brokers to route traffic to multiple brokers.
Local Queues	A local queue is a definition of both a queue and the set of messages that are associated with a queue. The hosting IBM MQ Queue Manager receives messages in its local queues. Local queues	Queues	This is equivalent to queues in Amazon MQ.

IBM MQ	Amazon MQ		
	support several IBM MQ specific properties like `put`, `get`, `inhibits`, etc.		
Remote Queues	Remote queues are used to refer to a queue which physically exists on a different IBM MQ Queue Manager, but connectivity is established using channels (Sender Channel and Receiver Channel)	Network of Brokers	This is achieved using a Network of Brokers. Networks of Brokers connect to one another on a Network Connector. With a Netowork of Brokers, you can create connections across different Regions and Availability Zones.
Alias Queues	Alias queues are used to give a different name to a different physical queue. It allows setting different security settings to the same physical queue.	Composite and Virtual Destinations	Composite Destinations and Virtual destinations provide the same functionality in the Amazon MQ world. You can achieve this by making a minor change in a broker configuration file.
Model Queues	Acts as a template for a queue definition.	Destination Policy	This can be achieved using Destination Policies in the broker configuration file

IBM MQ	Amazon MQ		
Transmission Queues	Used for remote queue connectivity	Network of Brokers	A network of Brokers provides equivalent configuration using Network Connector
Topics	Pub-Sub destination	Topics	Topics in Amazon MQ
Server connection channel	Basic construct through which direct connectivity from a remote applicati on client to an MQ server is established. Admins can create multiple connection channels to enforce different policies. A default server connection channel is created for each IBM MQ Queue Manager. Allows for monitoring of connections from remote clients.	Transport Connector	Provides the details for the connection point for producers and consumers to connect to a broker.

IBM MQ	Amazon MQ		
Client Connection Channel	Client connection channel in IBM MQ allows a connection mode where a config file (client connection definition table, or CCDT) is kept on the client application side which specifies options around connecting to an IBM MQ Queue Manager. Usually, legacy applications prior to JMS were written using this mechanism.	Transport Connector	Transport Connector provides the details for the connection point for producers and consumers to connect to a broker.
Channel level Firewall	Channel level Firewall is used to blacklist/ whitelist IP address, protocols	Security Group	With Amazon MQ, use the Security Group at the broker level and use NACL at the subnet level to create a firewall.

IBM MQ	Amazon MQ		
Error Logs	Each IBM MQ Queue Manager has a configuration around transaction logs. It's similar to database logs. It has a size and policy (circular /linear). Any time there is a shared JMS transaction, the state is written in a log file for rollback or commit.	Amazon CloudWatch	CloudWatch logs are fully integrated with Amazon MQ. For more information, see Monitoring Amazon MQ Brokers Using Amazon CloudWatch.
Listener	A listener is an IBM MQ process that listens for connections to the IBM MQ Queue Manager.	Broker Instance	Amazon MQ is a managed broker and acts as a listener by itself.
LDAP Authentication	IBM MQ provides out of box connectivity to LDAP and group-based access to QM, queues, topics, etc. It also supports fine-grained access control at queue and topic levels.	IAM and LDAP Authentication	Simple authentication is available with Amazon MQ. Fine-grained access is only available at the broker user level. Fine-Grained Access Control at queue and topic level with access options to `Get`, `Put`. To learn more, see Messaging Authentication and Authorization.

IBM MQ	Amazon MQ		
Message channel agent (MCA)	MCA is used to enforce an alternate level of security. The default configura tion is at the server connection channel level.	Not Applicable	Not Applicable
Channel exit programs	Channel exit programs are programs that are called at defined places in the processing sequence of a Message channel agent (MCA). Users and vendors can write their own channel exit programs. Some are supplied by IBM.	Plugins	Limited plugins supported. To learn more, see *Supported plugins*
Queue Manager Group	Queue Manager Group is used to support a cluster of IBM MQ Queue Managers. It is based on providing a high availability but providing a set of IBM MQ Queue Manager for connectivity.	Network of Brokers	Using Network of Brokers and each broker with Active-St andby configuration

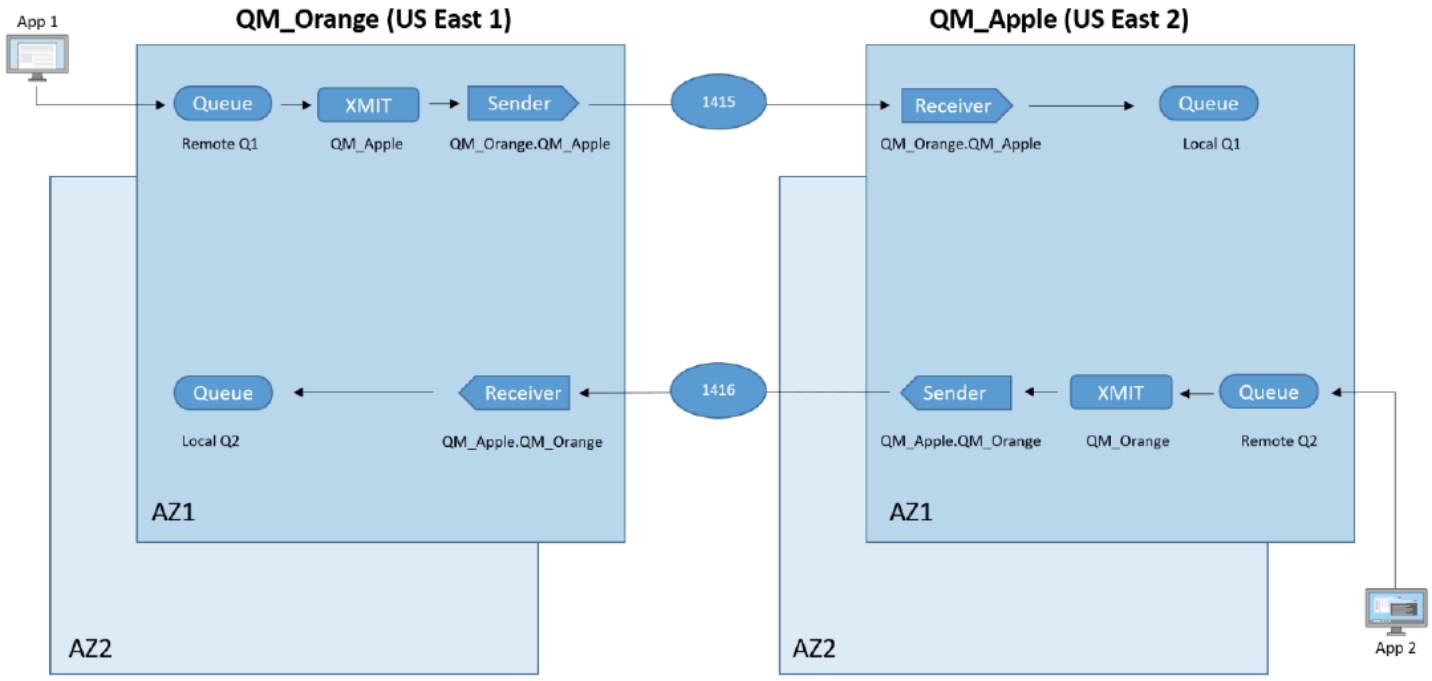
IBM MQ	Amazon MQ		
Sender / Receiver Channels	A message channel, a unidirectional communications link between two IBM MQ Queue Managers. WebSphere MQ uses message channels to transfer messages between the IBM MQ Queue Managers. To send messages in both direction s, you must define a channel for each direction.	Network Connector	A duplex communica tion channel between 2 brokers.

Which IBM MQ architectures are used for migrating to Amazon MQ?

You can migrate from IBM MQ to Amazon MQ using IBM MQ high availability topology running on AWS or IBM MQ HA/DR topology running on-premises.

Option one: IBM MQ high availability topology running on AWS

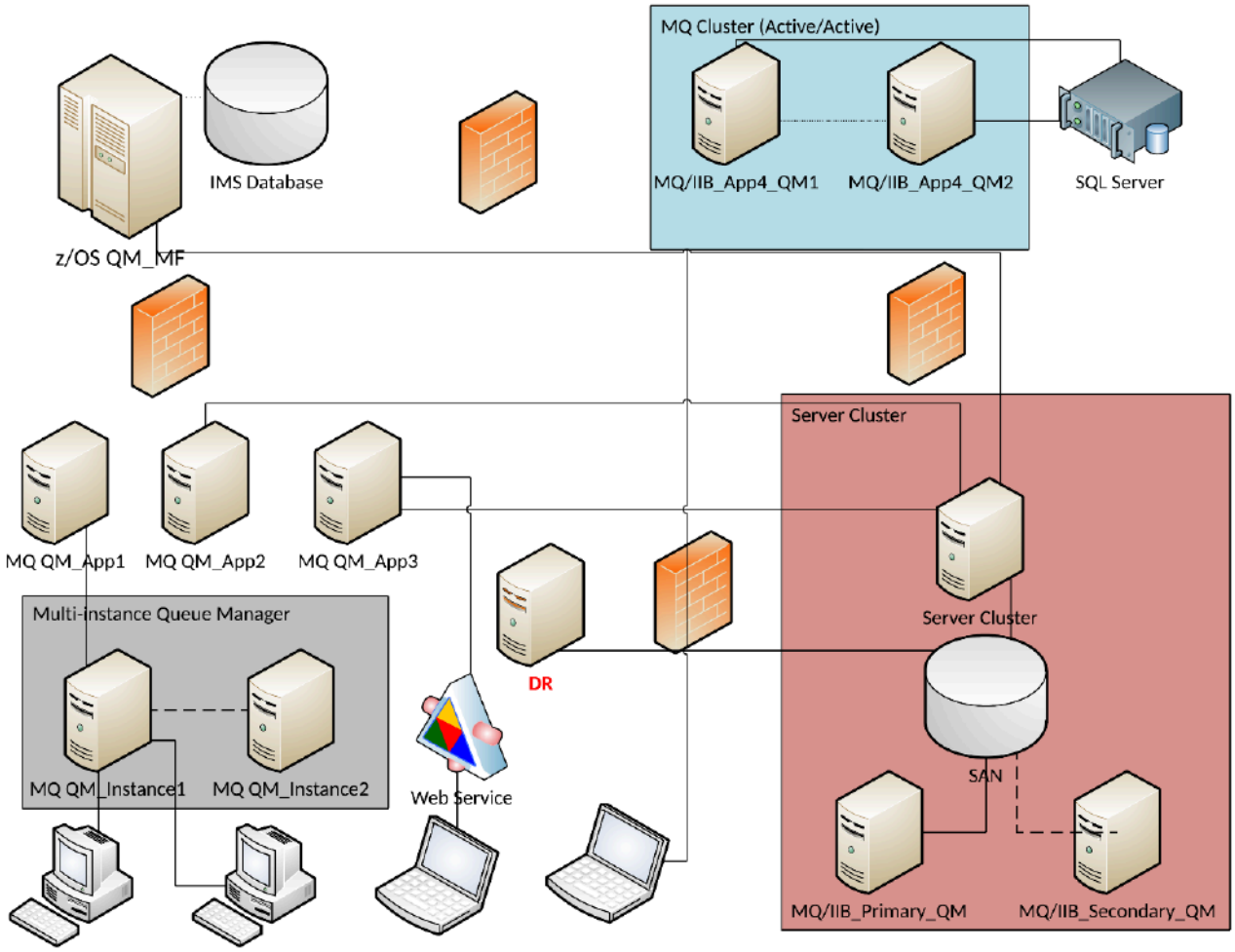
The below diagram shows a typical architecture of IBM MQ connections between two IBM MQ queue managers in a High Availability cluster as seen in many enterprise applications. IBM MQ queue manager **QM_ORANGE** is deployed in the *us-east-1* region and **QM_APPLE** is deployed in the *us-east-2* region.



For application *App 1* to communicate with *App 2*:

1. *App 1* uses a communications channel to send messages to **QM_ORANGE** on `Remote Q1`.
2. Messages from several such queues, though not shown in the diagram, are pooled into transmission `Queue Q` **QM_APPLE**.
3. Sender channels read messages from the transmission queue, and communicate with a receiver channel to place messages on `Local Q1` on queue manager **QM_APPLE** ,which are then consumed by *App 2*.

Option Two: IBM MQ HA/DR topology running on-premises



In the above diagram, the **MQ Cluster** is comprised of two separate queue managers and all messages are routed via cluster channels and queueing. If one queue manager fails, messages are then re-routed to another queue manager.

The **Server Cluster** is made up of just a single queue manager with three distinct IP addresses. In this configuration, all applications are connected to the Server Cluster IP addresses. If failure is detected, the *SAN* begins pointing to the secondary server. In case, of a failure event, channel connections do not have to be changed, and connectivity remains uninterrupted for users.

The **Multi-Instance queue manager** is comprised of a single queue manager with identical queues on both servers, and two distinct IP addresses. If failure is detected, a queue manager must be manually activated on the seconrd server and channel connections must be changed, resulting in possible service interruptions.

To ensure disaster recovery, data is replicated in real-time to a separate server in a different location. In case of a disaster, manual effort is required to process the data stored on the Disaster Recovery (DR) server, and change channel connections, resulting in possible service interruptions.

Replicating IBM MQ architecture with Amazon MQ

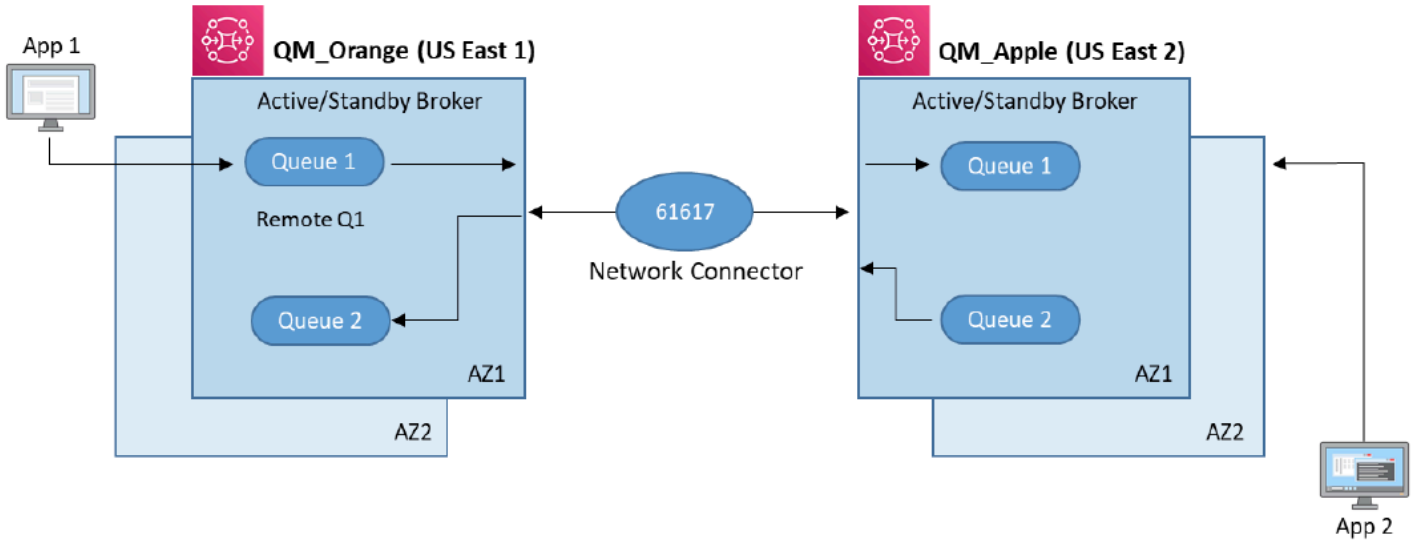
Amazon MQ provides a variety of broker configurations, various instance sizes for different workloads, and broker options such as single instance, single instance mesh, active/standby instance or active/standby mesh for high availabilty and message durability. To learn more about supported broker options, see Amazon MQ Broker Architecture.

In this section, we replicate the IBM MQ system shown in the previous section with Amazon MQ, while keeping the same configuration.

Amazon MQ manages the work involved in setting up an ActiveMQ message broker, from provisioning the infrastructure capacity (server instances and storage) to installing the broker software. Once your broker is up and running, Amazon MQ automates common administrative tasks such as patching the ActiveMQ software that power your brokers.

> **Note**
>
> If you wish to use a single region, you can simply deploy your Amazon MQ brokers in one region with the active/standby configuration. You can also optimize the performance of your Amazon MQ brokers by taking advantage of the Apache ActiveMQ optimization settings.

The following diagram illustrates Amazon MQ configured across two regions with a linear connection between two active/standby brokers:



For *App 1* to communicate with *App 2*:

1. Client applications can use a *transport* connector and put messages onto a Queue or publish to a Topic.
2. Brokers connect to each other over a *network* connector either in one direction or both directions in cases where request-reply messaging is required.
3. Queues and users can be created and managed in the AWS Console. To learn more, see Amazon MQ Basic Elements.

Note

- A *transmit queue* in IBM MQ is a local queue, except it is used to forward messages to a remote IBM MQ queue manager through a sender-receiver channel pair. In Amazon MQ, a transmit queue is not required. Once 2 brokers are connected using a network connector, they begin to share all queues/topics, and their data.
- A *remote queue* in IBM MQ is a local impression of a remote queue available at a remote IBM MQ queue manager. For external applications, there is no difference between local or remote queues. In Amazon MQ, there is no remote queue mechanism and it is not required.

- Sender or receiver channels in IBM MQ are used as network paths to connect 2 IBM MQ queue managers. In Amazon MQ, this functionality is implemented using network connectors.
- Currently, Amazon MQ only supports JMS 1.1. Applications written for JMS 2.0 can be migrated to Amazon MQ using the Qpid JMS library, which uses *AMQP* instead of the default, higher-performing *Openwire* protocol. For more details, refer to the Amazon MQ workshop.

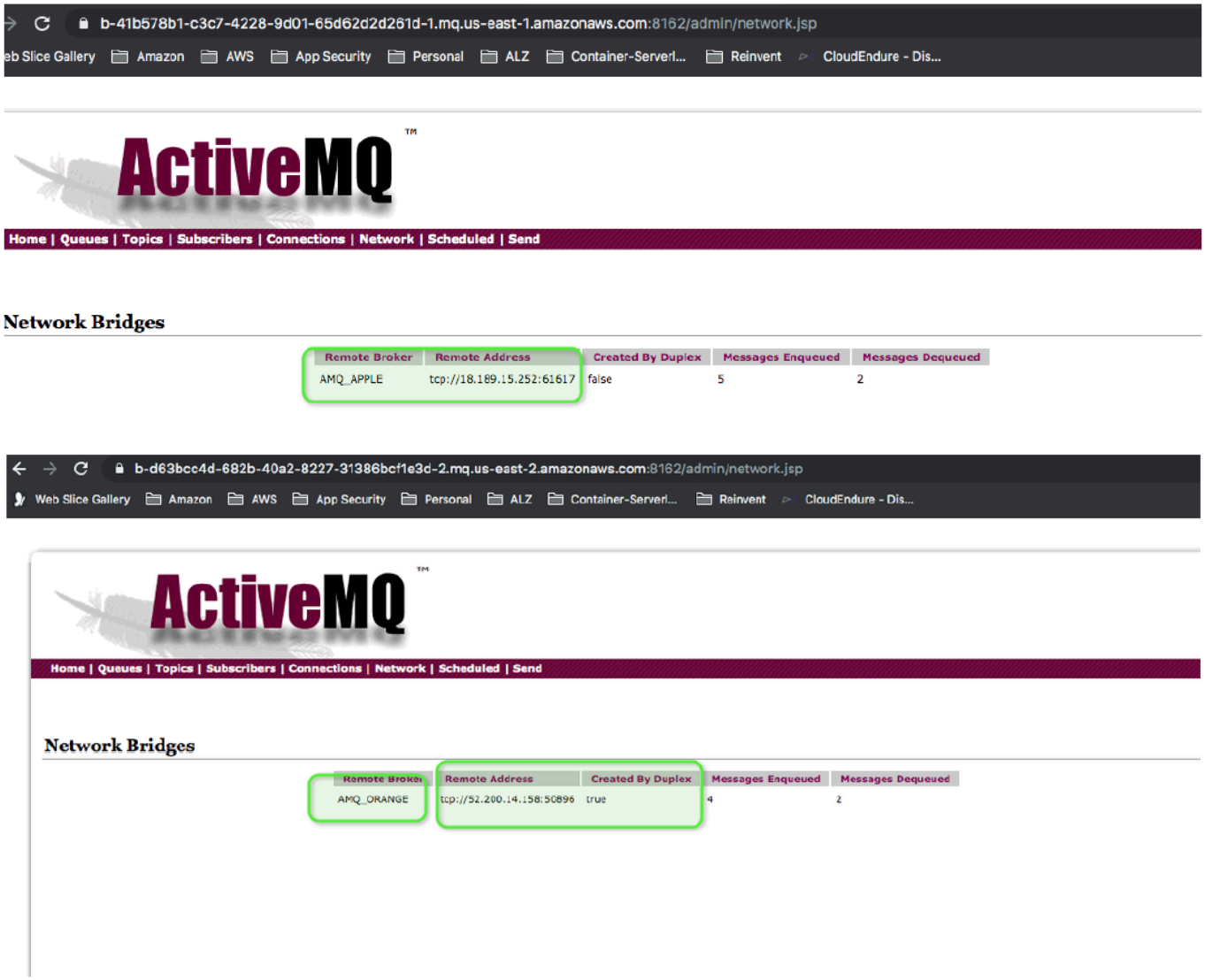
Re-platforming IBM MQ to Amazon MQ

The following procedure shows how you can migrate an IBM MQ to an equivalent Amazon MQ without impacting *App 1* or *App 2*:

1. Create an active/standby broker in *us-east-1* and another in *us-east-2* named as **AMQ_ORANGE** and **AMQ_APPLE**.
2. Create a *Network Bridge* between 2 brokers by adding a duplex network connector definition to one of the queues:

```
<networkConnectors>
    <networkConnector duplex="true" name="connector_AMQ_ORANGE_to_AMQ_APPLE"
 uri="masterslave:(ssl://b-d63bcc4d-682b-40a2-8227-31386bcf1e3d-1.mq.us-
east-2.amazonaws.com:61617,ssl://b-d63bcc4d-682b-40a2-8227-31386bcf1e3d-2.mq.us-
east-2.amazonaws.com:61617)" userName="amqadmin"/>
</networkConnectors>
```

   After the reboot of **AMQ_ORANGE**, there should be a Network Bridge created between both brokers as illustrated below:



> **Note**
>
> Steps 1 and 2 can be replicated using a AWS CloudFormation template. For more information about using AWS CloudFormation to set up Amazon MQ brokers, see the Amazon MQ AWS CloudFormation Template Reference.

3. Log in to IBM MQ Queue Manager Host and list the queues/topics definitions. In **QM_ORANGE**, you can list the queues and topics from IBM MQ using the following command:

```
$ sudo dmpmqcfg -m QM_ORANGE -t queue -o 1line |
    grep -v "SYSTEM" |
    grep -v "AUTHREC" |
    grep -v "*" |
```

```
    gawk -F: '{ print $1 }‘
```

The output:

```
DEFINE QREMOTE('Q1') RQMNAME('QM_APPLE') RNAME('Q1') XMITQ('QM_APPLE') REPLACE
DEFINE QLOCAL('Q2') DISTL(NO) MAXDEPTH(5000) REPLACE
DEFINE QLOCAL('QM_APPLE') GET(DISABLED) MAXDEPTH(5000) USAGE(XMITQ) REPLACE
```

In the example above, Q1 is the link to the remote queue, QM_APPLE is the transit queue, and Q2 is the local queue. We only need local queue Q2 for the Amazon MQ setup, which can be defined in the broker configuration as <queue physicalName="Q2"/>

Q1 is a local queue on QM_APPLE and Q2 is a local queue in QM_ORANGE. You can configure these resources accordingly in **AMQ_APPLE** and **AMQ_ORANGE** by using the following configuration

```
<destinations>
    <queue physicalName="localQ1"/>
</destinations>
```

Similairly, get the list of queues and topics from QM_APPLE.

4. Manually create a dead letter queue strategy in the AMQ configuration file.

   The defaultdead letter queue in ActiveMQ is called ActiveMQ.DLQ. All un-deliverable messages will get routed to this queue. To streamline this process, you can set up an individualDeadLetterStrategy in the destination policy map of the activemq.xml configuration file, allowing you to specify a specific dead letter queue prefix for a given queue or topic. You can apply this strategy using a wild-card so that all queues can be set up with their own dead-letter queues, as is shown in the example below:

```
<broker>
    <destinationPolicy>
        <policyMap>
            <policyEntries>
                <!-- Set the following policy on all queues using the '>' wildcard
 -->
                <policyEntry queue=">">
                    <deadLetterStrategy>
```

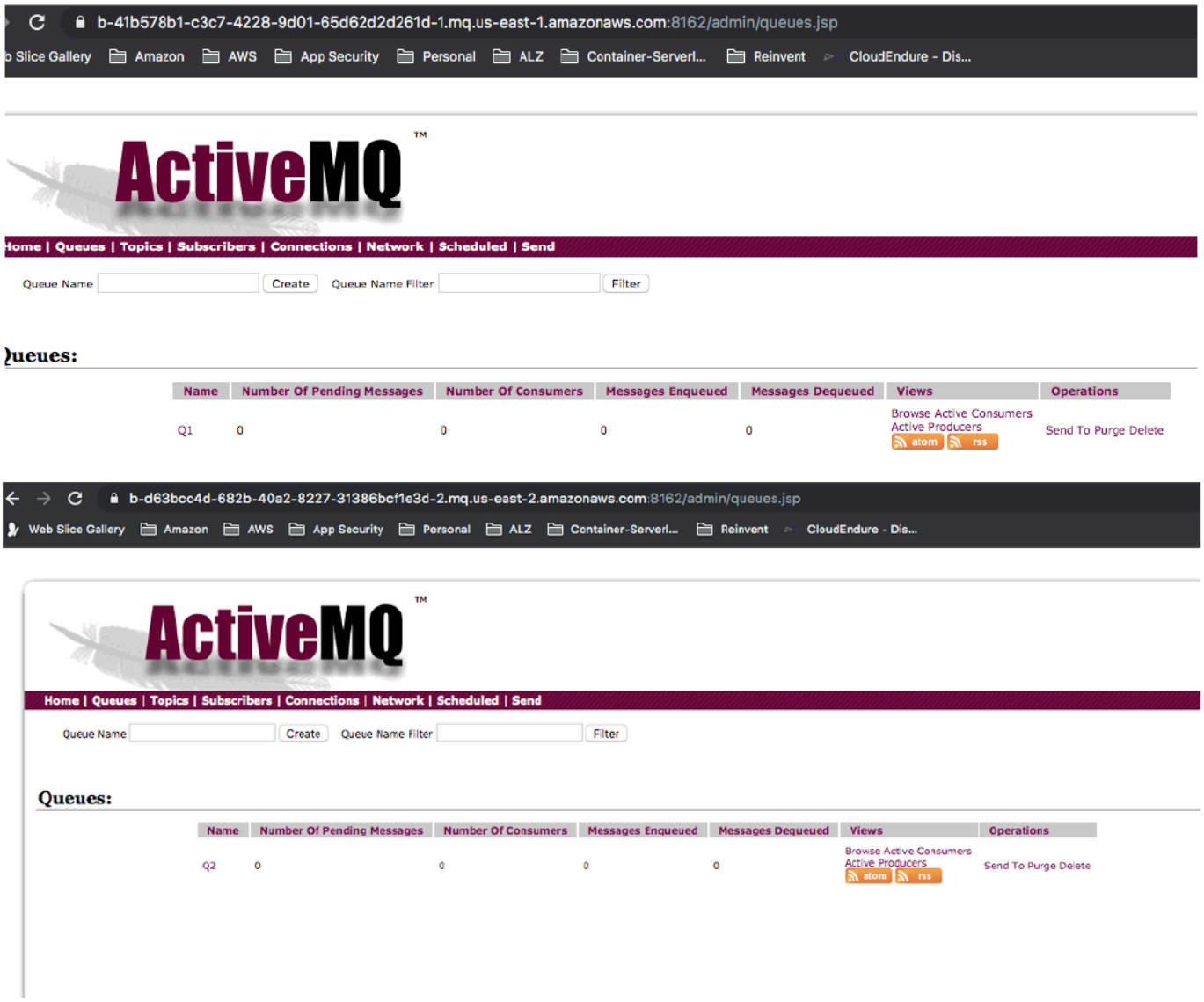
```
                        <!-- Use the prefix 'DLQ.' for the destination name, and
 make the DLQ a queue rather than a topic -->
                        <individualDeadLetterStrategy queuePrefix="DLQ."
 useQueueForQueueMessages="true"/>
                    </deadLetterStrategy>
                </policyEntry>
            </policyEntries>
        </policyMap>
    </destinationPolicy>
</broker>
```

Note

Dead-Letter queue expiration - By default, ActiveMQ will **never** expire messages sent to a Dead-Letter Queue (DLQ). However, beginning with ActiveMQ 5.12, the `deadLetterStrategy` supports an expiration attribute whose value is given in milliseconds as shown below:

```
<broker>
    <destinationPolicy>
        <policyMap>
            <policyEntries>
                <policyEntry queue="QueueWhereItIsOkToExpireDLQEntries">
                    <deadLetterStrategy>
                        <.... expiration="300000"/>
                    </deadLetterStrategy>
                </policyEntry>
            </policyEntries>
        </policyMap>
    </destinationPolicy>
</broker>
```

5. Create local queue Q1 on **AMQ_ORANGE** and Q2 on **AMQ_APPLE** as shown in the following:



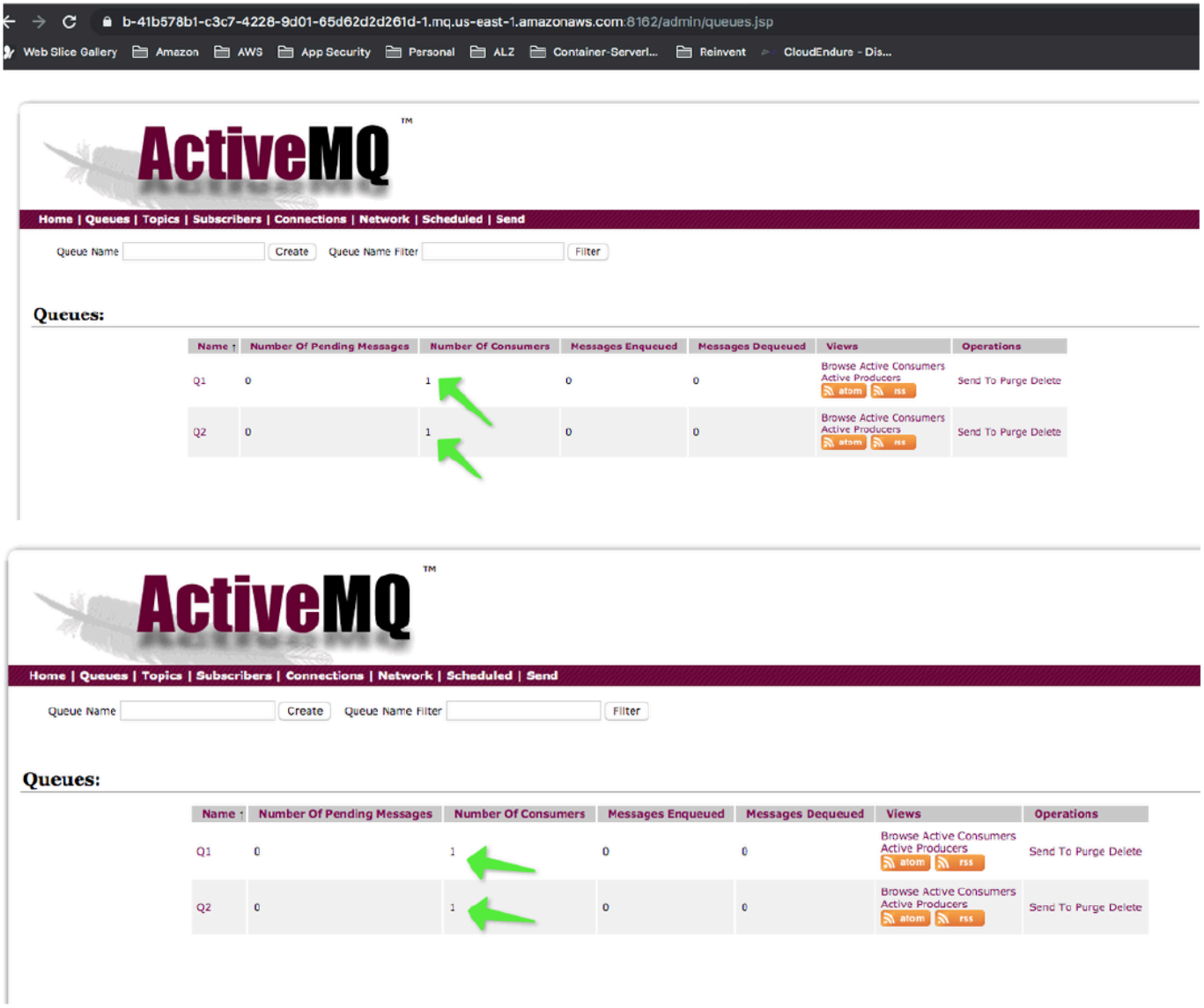
Validating your migration to Amazon MQ

Learn how to test and validate the availbability of your brokers using the following procedure.

1. Subscribe to Q1 on **AMQ_APPLE** and Q2 on **AMQ_ORANGE**. Using a Network Bridge, create a queue replica on both sides.

> **Note**
>
> The process for external subscribers is the same as subscribing to local queues.

The following example shows the **AMQ_ORANGE** broker with consumers in *us-east-1* and **AMQ_APPLE** with consumers in *us-east-2* :



2. Both queues are now available to both brokers, producers can send messages to any broker, and subscribers can receive messages from any broker. For *JMS 1.1* compliant applications, change the endpoint URL to an ActiveMQ failover URL.

> **Note**
>
> To learn more about a phased migration approach from IBM MQ to Amazon MQ, refer to this post.

Migrating from TIBCO EMS server to Amazon MQ

You can migrate from TIBCO EMS to Amazon MQ.

Terminologies

The following is a list of common TIBCO EMS concepts and how they relate to Amazon MQ.

TIBCO EMS	**Amazon MQ**		
Component	**Description**	**Component**	**Description**
EMS Server	TIBCO EMS Server is a message broker that supports standards-based Java Message Service (JMS) 1.1 and 2.0. It also supports TIBCO proprietary messaging formats, **FTL**, **Rendezvous**, and **SmartSockets**.	Broker	Broker in Amazon MQ is equivalent to TIBCO EMS Server. It provides support for industry-standard APIs such as JMS and NMS, and protocols such as **AMQP**, **STOMP**, **MQTT**, and **WebSocket**.
Static Destination	Configuration information for a static destination is stored in configura tion files for the EMS server.	Startup Destination	Amazon MQ allows you to create destinations when the broker is started by configuring Startup Destinations.
Dynamic Destination	Dynamic Destinati on is created as required by the client application and exists as long as there are messages or	Destination	In Amazon MQ, a destination is, by default, created automatically when it is used. You can use the *Delete Inactive Destinations* feature

TIBCO EMS	Amazon MQ		
	consumers associated with a destination.		in order to replicate the behavior of Dynamic Destinations in TIBCO EMS.
Temporary Destination	A Temporary Destination is used for reply messages in request/reply interactions.	Temporary Destination	A Temporary Destination is used for reply messages in request/reply interactions.
Queue	A queue is a mode to provide point to point messaging channel from producers to consumers	Queue	Similar to that of TIBCO EMS, Amazon MQ's Queue is a mode to provide point-to-point messaging channels from producers to consumers
Route	TIBCO EMS servers have to enable and configure routing to route messages to one or more servers. A route forwards messages between corresponding global destinations.	Network Connector	Networks of Brokers can connectto each other on a Network Connector and allows connections across Availability Zones and region.

TIBCO EMS	Amazon MQ		
Routed Queue	A Routed Queue has to be configure d on another EMS server for messages to be forwarded to or from a Queue. Queue messages can travel only one hop to the home queue, and one hop from the home queue.	Network of Brokers	This is achieved using a network of brokers. Amazon MQ provides a richer feature-set to work with destination behavior in Networks of Brokers.
Topic	Topics implement *publish and subscribe* messaging, and are equivalent to topics in Amazon MQ.	Topics	Topics in Amazon MQ are equivalent to topics in TIBCO EMS.
Global Topic	A Global Topic has to be configured on another EMS server for messages to be forwarded to or from a Topic. In a multi-hop zone, Topic messages are forwarded to all servers connected by routers within the zone. In a one-hop zone, topic messages travel only one hop.	Network of Brokers	This is achieved using a network of brokers. Amazon MQ provides a richer feature-set to work with destination behavior in Networks of Brokers.

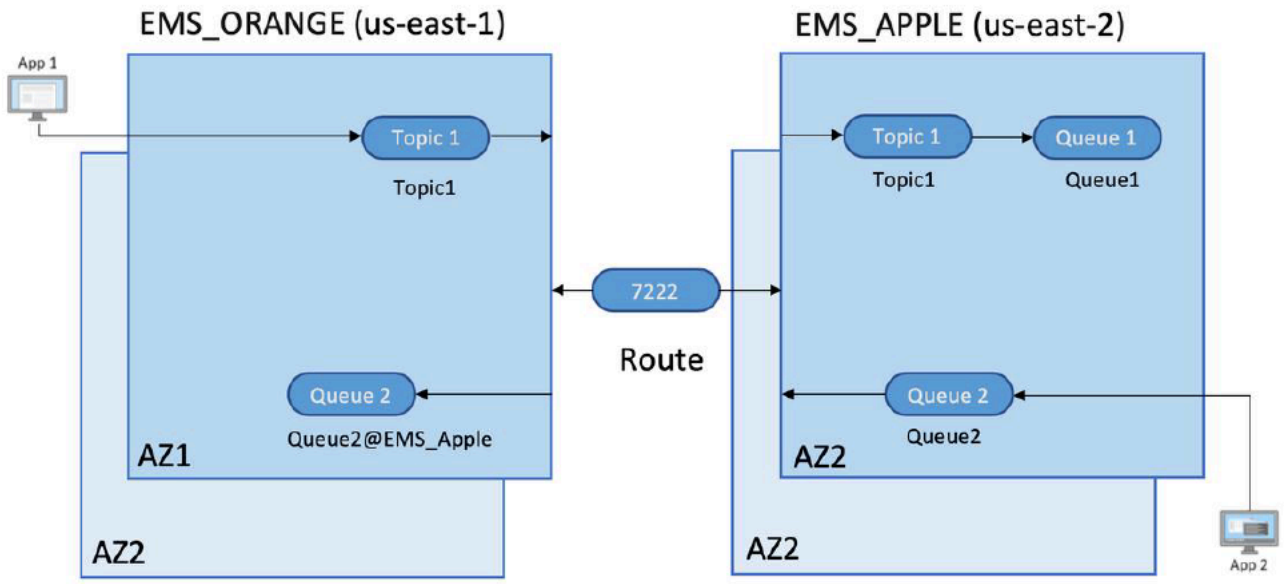
TIBCO EMS	Amazon MQ		
Destination Bridges	This allows all messages delivered to one destination to also be delivered to the bridged destination. It is most commonly used to create durability of messages published on a topic using the topic to queue bridge.	Virtual Topics and Composite Destinations	Topic to queue bridge-like functionality can be achieved using Virtual Topics in Amazon MQ. Other bridges can be migrated using Composite Destinations in Amazon MQ.
Message Store	EMS server writes persistent messages to disk and provides file-based and database stores. For file-based stores, you have to truncate the files periodically to relinquish disk space.	N/A	Amazon MQ also supports message persistence. Amazon MQ is a managed service and the overall storage is fully managed by AWS.

Which TIBCO EMS architectures are used for migrating to Amazon MQ?

You can migrate from TIBCO EMS to Amazon MQ using cross-regional architecture in AWS or TIBCO EMS high availability architecture.

Option 1: TIBCO EMS cross-regional architecture in AWS

The below diagram shows the typical architecture of TIBCO EMS routing between two TIBCO EMS Servers in different regions, common in many enterprise systems. TIBCO EMS Server **EMS_ORANGE** is deployed in the *us-east-1* region and **EMS_APPLE** is deployed in the *us-east-2* region:



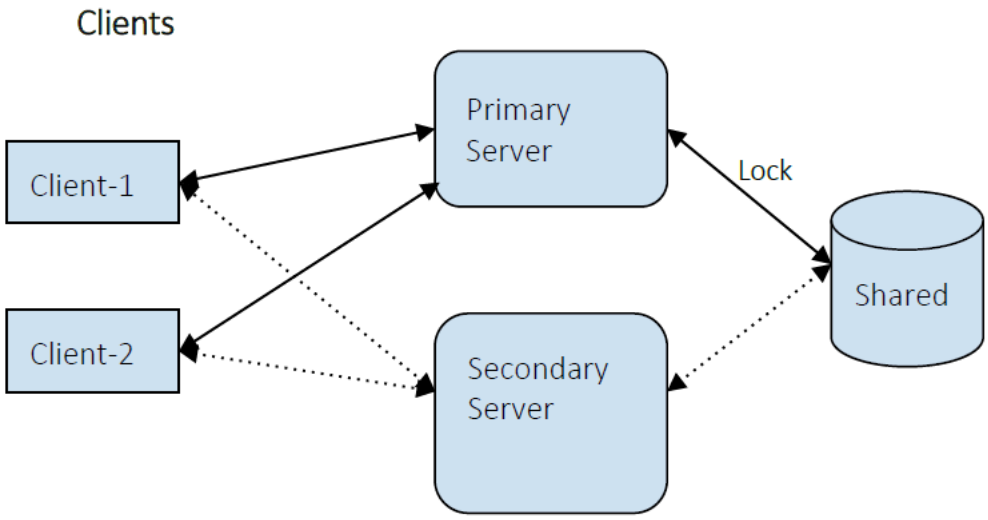
For application *App 1* to communicate with *App 2*:

1. *App 1* uses a topic destination, *Topic1* on server **EMS_ORANGE** to publish messages.
2. Published messages are transmitted to topic *Topic1* on server **EMS_APPLE** using the configured route.
3. On **EMS_APPLE**, a bridge is configured to move messages from topic, *Topic1* to queue, *Queue1*. Messages are then consumed by *App 2*.

Option 2: TIBCO EMS high availability architecture

In this configuration, High availability is provided by configuring a pair of servers, *Primary* and *Secondary*. In a typical enterprise architecture, two high availability configurations, *shared* and *unshared*. The shared state setup is the most widely used setup in enterprise settings. The

following diagram demonstrates the Shared State configuration for a pair of messaging servers:



In the above diagram, a pair of messaging servers share a state by sharing file-based storage. The primary server attains the lock on the shared storage capacity, becomes active, and accepts client connections, while the secondary server remains in passive mode. Meanwhile, the primary and secondary servers will be made aware of one another's status via periodic, heartbeat pings.

In te case of a failover, the secondary server will assume the state of the primary server, and acquire the lock on the shared state.

> **Note**
>
> The above configuration is unable to support more than two servers, and data replication across the servers for durability.

Replicating TIBCO EMS architecture with Amazon MQ

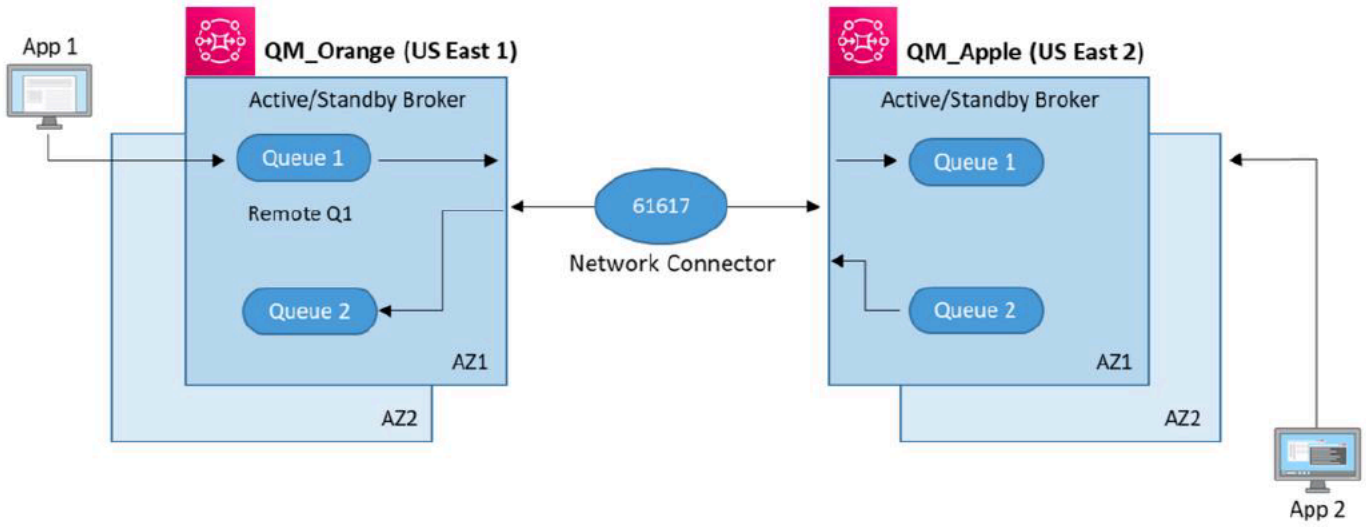
Amazon MQ provides a variety of broker configurations, various instance sizes for different workloads, and broker options such as single instance, single instance mesh, active/standby instance or active/standby mesh for high availabilty and message durability. To learn more about supported broker options, see Amazon MQ Broker Architecture.

In this section, we replicate the architecture of the TIBCO EMS system shown in the previous section with Amazon MQ while keeping the same configuration.

Note

If you wish to use a single region, you can simply deploy your Amazon MQ brokers in one region with the active/standby configuration. You can also optimize the performance of your Amazon MQ brokers by taking advantage of the Apache ActiveMQ optimization settings.

The following diagram illustrates Amazon MQ configured across two regions with a linear connection between two active/standby brokers:



For *App 1* to communicate with *App 2*:

1. Client applications can use a *transport* connector and put messages onto a Queue or publish to a Topic.
2. Brokers connect to each other over a *network* connector either in one direction or both directions in cases where request-reply messaging is required.
3. Queues and users can be created and managed in the AWS Console. To learn more, see Amazon MQ Basic Elements.

Note

- A *Global Topic* with the same name has to be created on other EMS Servers for forwarding messages to the Topic on those EMS Servers. In Amazon MQ, a *global topic*

is not required. Once 2 brokers are connected using a network connector, they begin to share all queues/topics, and their data.

- In Amazon MQ, a *routed queue* as implemented by a TIBCO EMS server is not required.

- A network bridge from a topic to a queue can be used in TIBCO EMS architecture to avoid the naming issue with routed queues and to provide multi-hop capability between EMS servers using a Topic. In Amazon MQ, queue names are consistent and all topic/queue messages are shared among a Networks of Brokers.

- Currently, Amazon MQ only supports JMS 1.1. Applications written for JMS 2.0 can be migrated to Amazon MQ using the Qpid JMS library, which uses *AMQP* instead of the default, higher-performing *Openwire* protocol. For more details, refer to the Amazon MQ workshop.

Re-platforming TIBCO EMS to Amazon MQ

You can use the following procedure to migrate the TIBCO EMS architecture shown here to an equivalent Amazon MQ architecture without impacting *App 1* or *App 2*:

1. Create an active/standby broker in *us-east-1* and another in *us-east-2* named as **AMQ_ORANGE** and **AMQ_APPLE**.
2. Create a *Network Bridge* between 2 brokers by adding a duplex network connector definition to one of the queues:

```
<networkConnectors>
    <networkConnector duplex="true" name="connector_AMQ_ORANGE_to_AMQ_APPLE"
 uri="masterslave:(ssl://b-d63bcc4d-682b-40a2-8227-31386bcf1e3d-1.mq.us-
east-2.amazonaws.com:61617,ssl://b-d63bcc4d-682b-40a2-8227-31386bcf1e3d-2.mq.us-
east-2.amazonaws.com:61617)" userName="amqadmin"/>
</networkConnectors>
```

After the reboot of **AMQ_ORANGE**, there should be a Network Bridge created between both brokers as illustrated below:

Network Bridges

Remote Broker	Remote Address	Created By Duplex	Messages Enqueued	Messages Dequeued
AMQ_APPLE	tcp://3.134.122.213:61617	false	0	0

> **Note**
>
> Steps 1 and 2 can be replicated using a AWS CloudFormation template. For more information about using AWS CloudFormation to set up Amazon MQ brokers, see the Amazon MQ AWS CloudFormation Template Reference.

3. Retrieve the list of static TIBCO EMS server destinations from the config files, `queues.conf` and `topics.conf` or by using the following `tibemsadmin` commands:

```
show queues * static
show topics * static
```

When finished, update the Amazon MQ broker **AMQ_ORANGE** configuration file to add startup destinations as shown here:

```
<destinations>
    <queue physicalName="FOO.BAR"/>
    <topic physicalName="SOME.TOPIC"/>
</destinations>
```

4. Destination properties for TIBCO EMS can be found in queues.conf and topics.conf files. Per Destination level Policy can be set in Amazon MQ using the `destinationPolicy` section in the configuration file.

5. Retrieve the list of TIBCO EMS Bridges from `bridges.conf`. For example, the Bridge from source topic `NOTIFY.FOOBAR` to target queues `FOO` and `BAR` is shown as:

```
[topic:NOTIFY.FOOBAR]
```

```
queue=F00
queue=BAR
```

When finished, up the Amazon MQ broker **AMQ_ORANGE** configuration file to add Composite Destinations that match TIBCO EMS bridges.

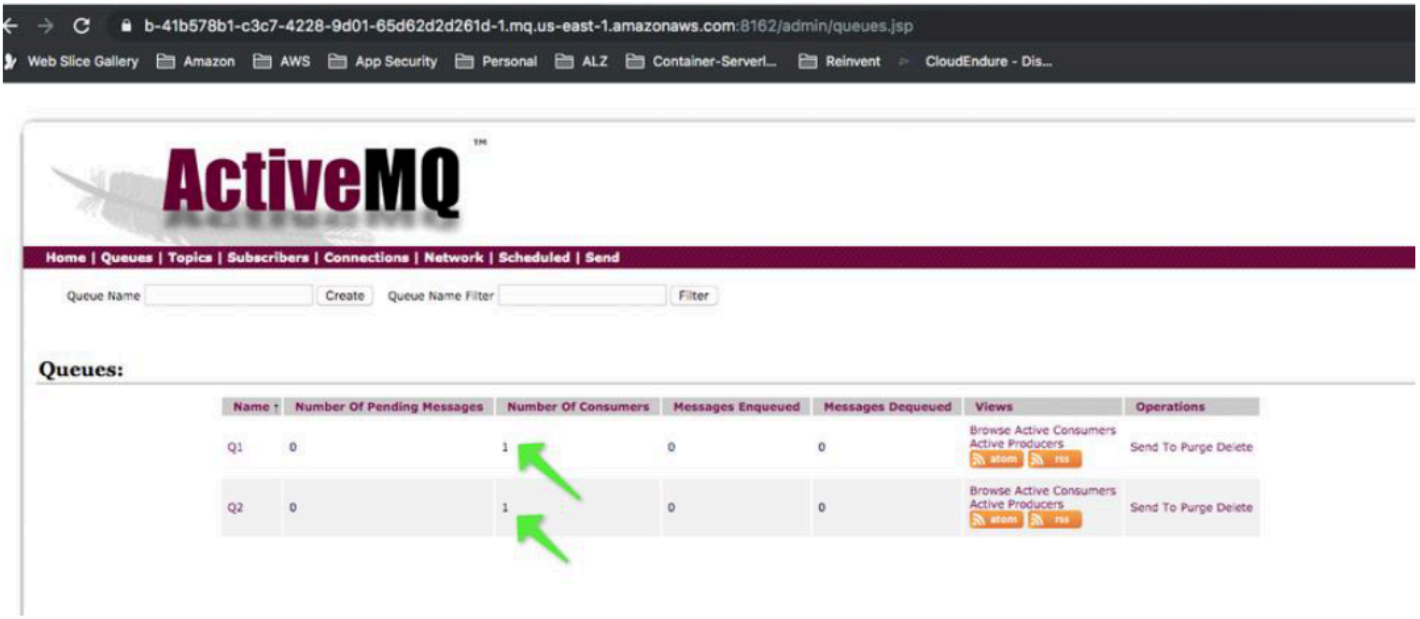
> **Note**
>
> *Simple Topic to Queue* bridges are needed in TIBCO EMS to support *m-hop* routing. In Amazon MQ this is not needed and queues can be used directly with a Network of Brokers.

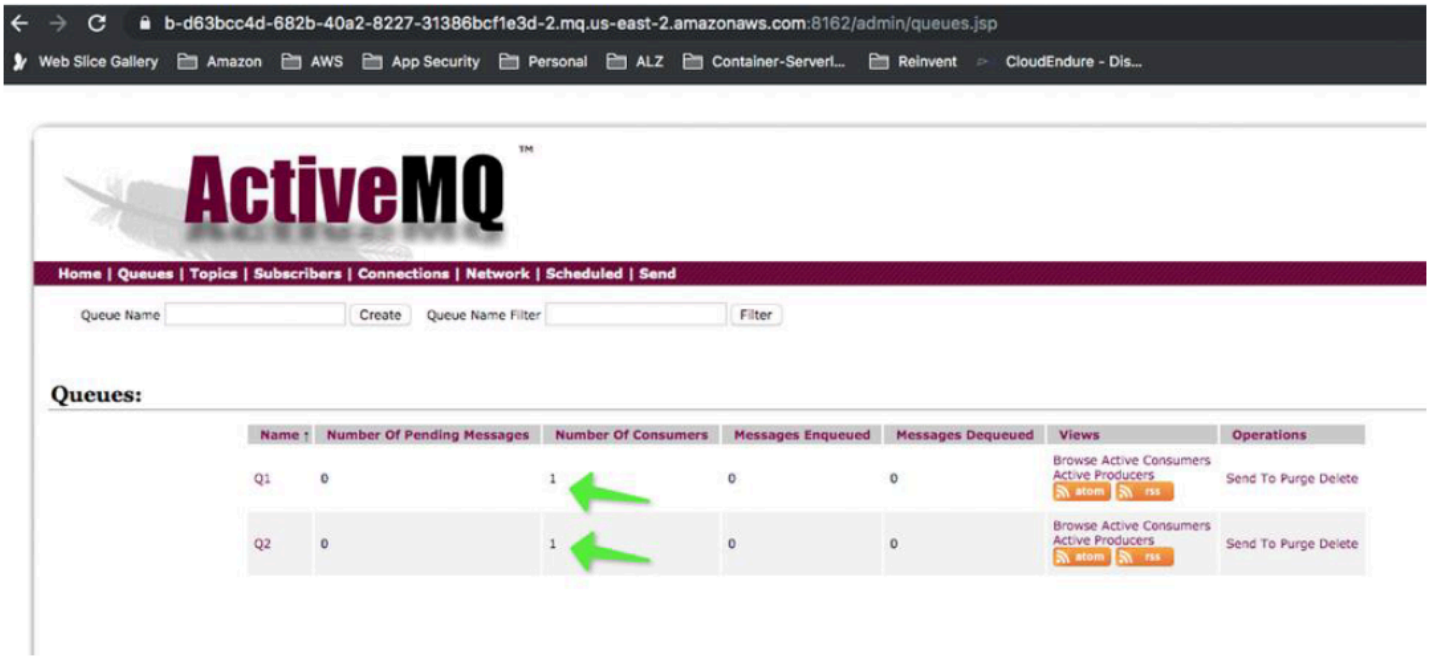
Validating your migration to Amazon MQ

In the the section called "TIBCO EMS architecture for migration" section, a *Topic to Queue* bridge was used to forward messages to other EMS servers. In Amazon MQ, *App 1* would send messages directly to Q1 because messages on a queue are forwarded in a Network of Brokers.

In the TIBCO EMS example, messages from *App 2* are sent to Q2 and then forwarded to Q2@EMS_APPLE. In Amazon MQ, the queue name, Q2, would be the same on both message brokers, simplifying the configuration of *App 1*.

The following example shows the **AMQ_ORANGE** broker with consumers in *us-east-1* and **AMQ_APPLE** with consumers in *us-east-2*


b-41b578b1-c3c7-4228-9d01-65d62d2d261d-1.mq.us-east-1.amazonaws.com:8162/admin/queues.jsp
Web Slice Gallery
Amazon
AWS
App Security
Personal
ALZ
Container-Serverl...
Reinvent
CloudEndure - Dis...
ActiveMQ
Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send
Queue Name
Create
Queue Name Filter
Filter
Queues:
Name
Number Of Pending Messages
Number Of Consumers
Messages Enqueued
Messages Dequeued
Views
Operations
Q1
0
1
0
0
Browse Active Consumers
Active Producers
atom
rss
Send To Purge Delete
Q2
0
1
0
0
Browse Active Consumers
Active Producers
atom
rss
Send To Purge Delete


b-d63bcc4d-682b-40a2-8227-31386bcf1e3d-2.mq.us-east-2.amazonaws.com:8162/admin/queues.jsp
Web Slice Gallery
Amazon
AWS
App Security
Personal
ALZ
Container-Serverl...
Reinvent
CloudEndure - Dis...
ActiveMQ
Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send
Queue Name
Create
Queue Name Filter
Filter
Queues:
Name
Number Of Pending Messages
Number Of Consumers
Messages Enqueued
Messages Dequeued
Views
Operations
Q1
0
1
0
0
Browse Active Consumers
Active Producers
atom
rss
Send To Purge Delete
Q2
0
1
0
0
Browse Active Consumers
Active Producers
atom
rss
Send To Purge Delete

Migrating to Amazon MQ for RabbitMQ

Amazon MQ supports migration from self managed Classic Mirrored Queues to Amazon MQ managed Classic Mirrored Queues on all supported cluster instances running RabbitMQ version `3.10.x` or higher.

You can export the configuration from your self-managed RabbitMQ cluster and import it into Amazon MQ for RabbitMQ. Queue, exchange, user, and policy definitions are imported. You can edit the exported JSON from the existing RabbitMQ cluster to remove the definitions that are not supported. The current default queue type is Classic Mirrored Queues. We recommend Customers use the latest Amazon MQ for RabbitMQ supported version when using Classic Mirrored Queues.

> **Important**
>
> Amazon MQ for RabbitMQ has an enforced policy of `ha-mode=all` and `ha-sync-mode=automatic` which will override any custom policy.

Prerequisites

Complete the following prerequisites before migrating to Amazon MQ for RabbitMQ:

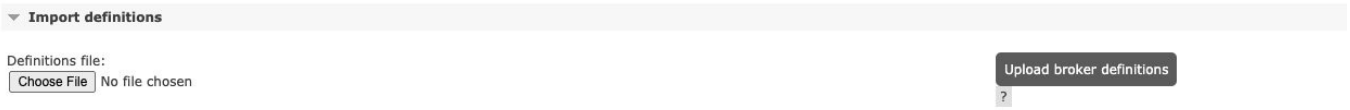
- Review the Concepts for migrating to Amazon MQ and Options for migrating to Amazon MQ for migration to AWS managed Amazon MQ
- Create a RabbitMQ broker: You will import the configuration from your self-managed RabbitMQ cluster into an existing Amazon MQ for RabbitMQ broker. For instructions on how to create a Amazon MQ for RabbitMQ broker, see Creating and connecting to a RabbitMQ broker.

Step 1: Exporting definitions

To export the definitions from a self-managed RabbitMQ cluster and import them into Amazon MQ for RabbitMQ, do the following:

1. Go to the RabbitMQ console of your existing self-managed cluster by signing on to any of the brokers. Choose the overview tab, then select `Export Definitions` to produce a link to export the definition.

2. Next, login to the Amazon MQ RabbitMQ console. Navigate to the existing broker you would like to apply the configurations to. Click on the overview tab, then click import definitions to upload the configuration file that you exported in the previous step.



3. Once the configuration file is imported, you can view all the queues and exchange definitions that were defined in the self-managed broker.

Step 2: Moving existing messages to your new Amazon MQ managed broker

Amazon MQ for RabbitMQ currently supports the Federation and Shovel plugins for moving messages from a self-managed RabbitMQ broker to an Amazon MQ for RabbitMQ broker.

Shovel

The Shovel plugin is used to move messages from an on-premises RabbitMQ broker without internet access to a private Amazon MQ managed broker. The Shovel plug in is configured on the on-premises broker to push messages to the Amazon MQ managed broker. Using the Shovel plug in requires a VPN connection between the Amazon Managed VPC and the on-premises network. For more information on using the Shovel plug in, see How do I set up the RabbitMQ Shovel plugin on my Amazon MQ broker?

Federation

The Federation plugin facilitates moving messages from a public upstream broker to a downstream broker. The plugin is configured on the downstream broker, which in this case is

the new Amazon MQ for RabbitMQ broker created in the previous section. For more information on using the RabbitMQ federation plug in, see the RabbitMQ Federation Plugin documentation.

Additional resources

Versions

When you create a new Amazon MQ for RabbitMQ broker, you can specify any supported RabbitMQ engine version. The different engine versions support certain features. For optimal performance, we suggest using the latest engine version.

Sizing

The broker instance type determines system throughput. Before migrating, review the sizing documentation to determine the best broker instance type for your application.

Limits

Amazon MQ for RabbitMQ brokers, configurations, users, data storage, and API throttling have default limits. To request an increase for a limit, see AWS Service Quotas in the Amazon Web Services General Reference.

Best practices

Learn more about ensuring effective performance in all areas of your RabbitMQ application by reviewing the RabbitMQ best practices documentation.

Amazon MQ for ActiveMQ Throughput benchmarks

Benchmarking can help you choose the correct instance type and size for your workload messaging requirements. Scenarios for benchmarking include:

- **Cluster Stability**: understanding how stable your cluster is during increasing, fluctuating, and stable load types.
- **Defining performance limits**: approximating the maximum performance and throughput capabilities (i.e. cluster limits) of your cluster to help better scale your broker nodes when the number of messages published to your broker increases.
- **Optimal architecture and parameters**: determining the most suitable architecture/parameters for your clusters, such as the number of destinations, persistent mode, message size, etc.

Amazon MQ provides benchmarking figures for the different instance types and sizes available for Amazon MQ for ActiveMQ.

Amazon MQ uses the ActiveMQ Maven 2 Performance Test to calculate benchmarks. Amazon MQ tests with an active/stanby deployment broker with an EFS volume as a storage type. The results are from a performance test conducted on a ECS cluster with a Fargate deployment which consists of a configuration of 4vCPUs and 16 GiBs of memory (equivalent to an EC2 m5.xlarge instance). Concurrent Store And Dispatch Queues (CSAD) are set to true for all tests performed. The duration for each test conducted is 5 minutes.

> **Note**
>
> When run in your own environment, results may differ by 3-6%.

The following tables provide performance and throughput benchmarks for Amazon MQ supported instance types to help you choose the correct instance sizes for your messaging workload.

Topics

- mq.m5.2xlarge
- mq.m5.4xlarge

mq.m4.large

Configuration options:

- **Broker Instance -** `mq.m4.large`
- **Persistent -** TRUE
- **Client -** `m5.xlarge`
- **CSAD -** TRUE
- **Protocol -** Openwire

		Producers/Consumers		
Message size	**Metrics**	**25**	**50**	**100**
1KB	TPS	1849	3335	4665
	CPU%	29%	37%	47%
5KB	TPS	1672	2561	2970
	CPU%	33%	47%	76%
10KB	TPS	1586	1670	2268
	CPU%	44%	87%	89%

mq.m5.large

Configuration options:

- **Broker Instance -** `mq.m5.large`
- **Persistent -** TRUE
- **Client -** `m5.xlarge`

- **CSAD -** TRUE
- **Protocol -** Openwire

		Producers/Consumers		
Message size	**Metrics**	**25**	**50**	**100**
1KB	TPS	2247	4041	7566
	CPU%	26%	32%	48%
5KB	TPS	1636	3205	4443
	CPU%	37%	63%	58%
10KB	TPS	1668	3104	3227
	CPU%	40%	53%	86%

`mq.m5.xlarge`

Configuration options:

- **Broker Instance -** `mq.m5.xlarge`
- **Persistent -** TRUE
- **Client -** `m5.xlarge`
- **CSAD -** TRUE
- **Protocol -** Openwire

		Producers/Consumers		
Message size	**Metrics**	**25**	**50**	**100**
1KB	TPS	2255	3932	7453
	CPU%	28%	32%	54%

		Producers/Consumers		
Message size	**Metrics**	**25**	**50**	**100**
5KB	TPS	1766	3495	6215
	CPU%	29%	51%	82%
10KB	TPS	1641	3240	5613
	CPU%	36%	61%	89%

mq.m5.2xlarge

Configuration options:

- **Broker Instance -** `mq.m5.2xlarge`
- **Persistent -** TRUE
- **Client -** `m5.xlarge`
- **CSAD -** TRUE
- **Protocol -** Openwire

		Producers/Consumers		
Message size	**Metrics**	**25**	**50**	**100**
1KB	TPS	2025	4089	8093
	CPU%	12%	18%	35%
5KB	TPS	1865	3736	6845
	CPU%	15%	27%	54%
10KB	TPS	1747	3511	7057
	CPU%	18%	36%	67%

mq.m5.4xlarge

Configuration options:

- **Broker Instance -** `mq.m5.4xlarge`
- **Persistent -** TRUE
- **Client -** `m5.xlarge`
- **CSAD -** TRUE
- **Protocol -** Openwire

		Producers/Consumers		
Message size	**Metrics**	**25**	**50**	**100**
1KB	TPS	2094	4055	8153
	CPU%	6%	9%	17%
5KB	TPS	1742	3586	7158
	CPU%	7%	13%	25%
10KB	TPS	1733	3288	6671
	CPU%	9%	16%	31%

Supported plugins for Amazon MQ

A plugin in Amazon MQ is a software module that adds a specific feature to a broker. Amazon MQ managed brokers support the following plugins:

- authorizationPlugin: Allows you to control access at the granularity level of destinations or of individual messages.
- discardingDLQBrokerPlugin: Provides fine-grained options to discard your dead-letter queue.
- redeliveryPlugin: Enables you to replace the regular DLQ handling with re-delivery to the original destination following a delay period.
- forcePersistencyModeBrokerPlugin: Allows you to force every incoming message to be *persistent* or *non-persistent*. This is useful if you've set up a broker usage policy to process only persistent or non-persistent messages.
- statisticsBrokerPlugin: Enables you to retrieve statistics from the broker or its destinations.
- timeStampingBrokerPlugin: Allows you to update a JMS Client's timestamp on a message with a broker timestamp. You can trust the timestamp set on your Amazon MQ brokers when client-side machine clocks are known to be incorrect.

Amazon MQ Migration Guide document history

The following table lists changes to the *Amazon MQ Amazon MQ Migration Guide*. For Amazon MQ feature releases and improvements, see Amazon MQ Release Notes.

Date	Documentation Update
May 14, 2023	Amazon MQ updated the Amazon MQ for ActiveMQ throughput benchmarks. Throughput benchmarks have been updated for messages sizes 1KB, 5KB, and 10KB with 25, 50, and 100 producers/consumers. Untested values were removed from the documentation. For more information on the testing procedure and the updated benchmarking figures, see Amazon MQ for ActiveMQ Throughput benchmarks.
May 14, 2023	Amazon MQ is now offering the Amazon MQ for RabbitMQ Migration Guide for migrating from self-managed RabbitMQ to Amazon MQ for RabbitMQ Classic Mirrored Queues.
October 5, 2020	Amazon MQ is now offering a comprehensive guide for migrating on-premises commercial message-brokers, such as IBM MQ and TIBCO EMS, to the cloud.