IBM MQ: Security, HA, Kafka

April 2026

Chris Leonard
STSM, IBM MQ
ChrisL@uk.ibm.com

AskMessaging@uk.ibm.com

Security

highlights

- Token-based authentication with JWT
- Initial support for ‘long’ user IDs
- TLS Certificate Management
- TLS Support for CCDT Retrieval
- Enhanced Security with Global Security Kit (GSKit 9)
- Positive authentication events - audit logs for both successful and unsuccessful events
- RDQM HA and DR data encrypted in transit
- FIPS mode configuration for AMS
- FIPS mode in MQ Explorer
- Server-side encryption support for AMS in containers
- OpenID Connect (OIDC) for the MQ Appliance, Multi-Factor Authentication support
- Customizable SSL socket factories for JWT/CCDT with MQ classes for JMS

- And many more...

Token-based authentication

What is a JSON Web Token?

A token is a simple structure that contains information about a user and can easily be transferred between parties over the internet.

A JWT can be cryptographically signed to form a JWS. This allows for the cryptographic verification of the information inside the JWT.

Token-based authentication

IBM MQ enables password-less authentication with JSON Web Tokens (JWT) and JSON Web Key Sets (JWKS) enabling a more secure, performant, scalable and portable approach, aligned with cloud architectures and multi factor authentication (MFA)

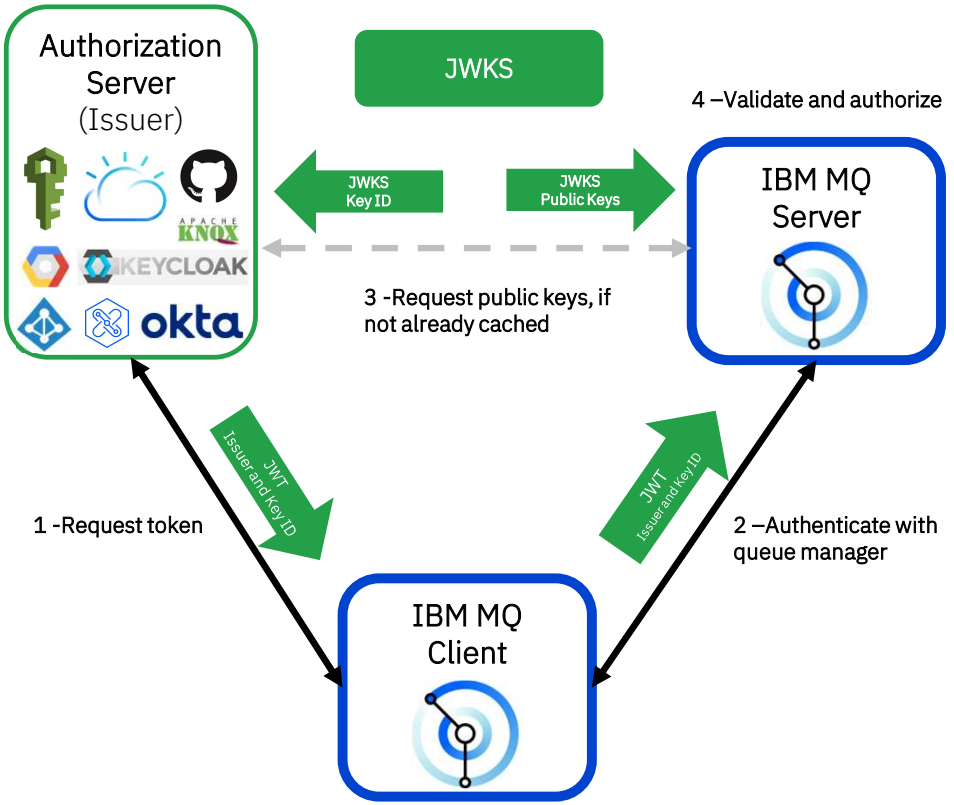
Secure — Eliminates security vulnerability of password transmission/storage. Digitally signed avoiding tampering and each queue manager performs its own validation

Scalable — One issuer for many applications authenticating with multiple queue managers and other services. No server-side session, simplifying scaling and load balancing.

Performant — Token validated by receiver with no call to issuer, and user information already within the token, improving performance and reducing dependencies

Portable — Tokens can be used across multiple, diverse applications or other endpoints, enabling easier secure collaboration between enterprises and platforms

Easier — Using a JWKS issuer makes it easier to manage certificate rotation, expiry, and revocation. Keys are cached to tolerate outages and improve performance



Long User ID support

Traditionally, MQs concept of a ‘UserID’ has always been 12 characters

- Some exceptions (LDAP, Windows, but always mapping down to a 12 char ID in some form)

MQ 9.4.3 partially lifts this restriction – applications can connect, authenticate, and be granted authorities using a ‘long’ (1024 char) identity.

- Opt in at queue manager wide level

This does NOT change the MQMD ‘UserID’ which will be truncated, so this identity cannot be used for ‘downstream’ checks at present, e.g.

- Reply authority
- Command server
- PUTAUT
- SUBUSER

Setting authorities:

```
setmqaut -m demo -t qmgr -p abeards@uk.ibm.com +all
The setmqaut command completed successfully.

setmqaut -m demo -t queue -n DEV.QUEUE.1 -p abeards@uk.ibm.com +all
The setmqaut command completed successfully.
```

Identity in sent message:

```
GET of message number 1, CompCode: 0 Reason:0
*Message descriptor****
StrucId MD : 1 Version: 2
[...]
Format: 'MQSTR
* Identity Context
UserIdentifier: ‘abeards@uk+’
[...]
```

TLS certificate management

New command “dspmqcert” available

Quickly see information about certificates within IBM MQ keystores

Filter on certificates expiring within a certain timeframe

Output in different machine-readable formats

When run as an automated process, enables you to build alerts when certificates are near expiry

Use **-e** to specify error exit code (127), use in conjunction with days to expiry **-d**

example: **dspmqcert -m QM1 -e -d 30**

```
Usage: dspmqcert [-h] [-o <output>] [-d <days>] [-m <pattern>] [-e] [-f <path>]
Options:
    -h this help
    -o output: <json|csv|text> default text
    -d days: print certs due to expire within these days
    -m pattern: match queue manager names, ie. QM1, ERR*, default *
    -a include CA intermediate certificates
    -e exit error (127) if -d specified and there are certs expiring
    -f path: location of INI file mapping QM names and keystore location

This command displays certificate information for Queue Managers

If not specified by -f, the default path for the keystore is:

    <QM_DATA_DIRECTORY>/<QUEUE_MANAGER_NAME>/ssl/key.kdb

Passwords are mandatory and are read from environment variables. Example
invocation:

    KBPASS_QM1=changeit dspmqcert -m QM1
```

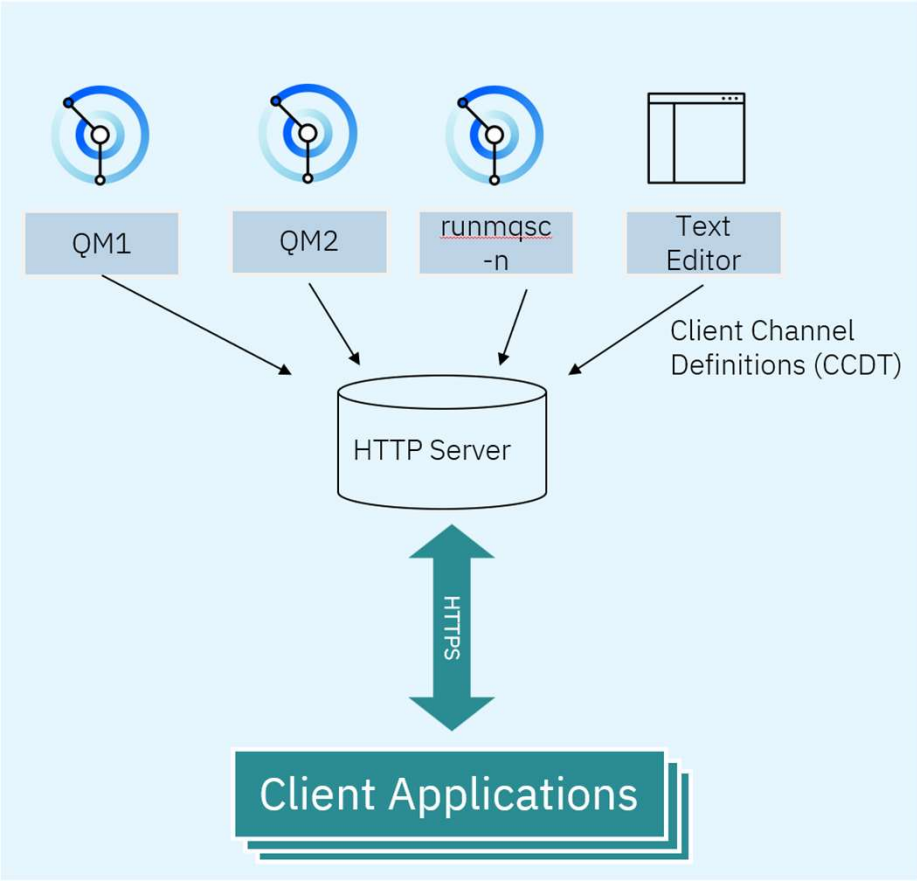
TLS support for CCDT retrieval

Centralised client configuration has become ever more important:

- Uniform clusters
- JSON CCDTs

IBM MQ 9.4.1 added configuration of key store allowing MQ client libraries to access CCDTs from web server

Provides another option for secure, central management of connectivity across multiple client endpoints

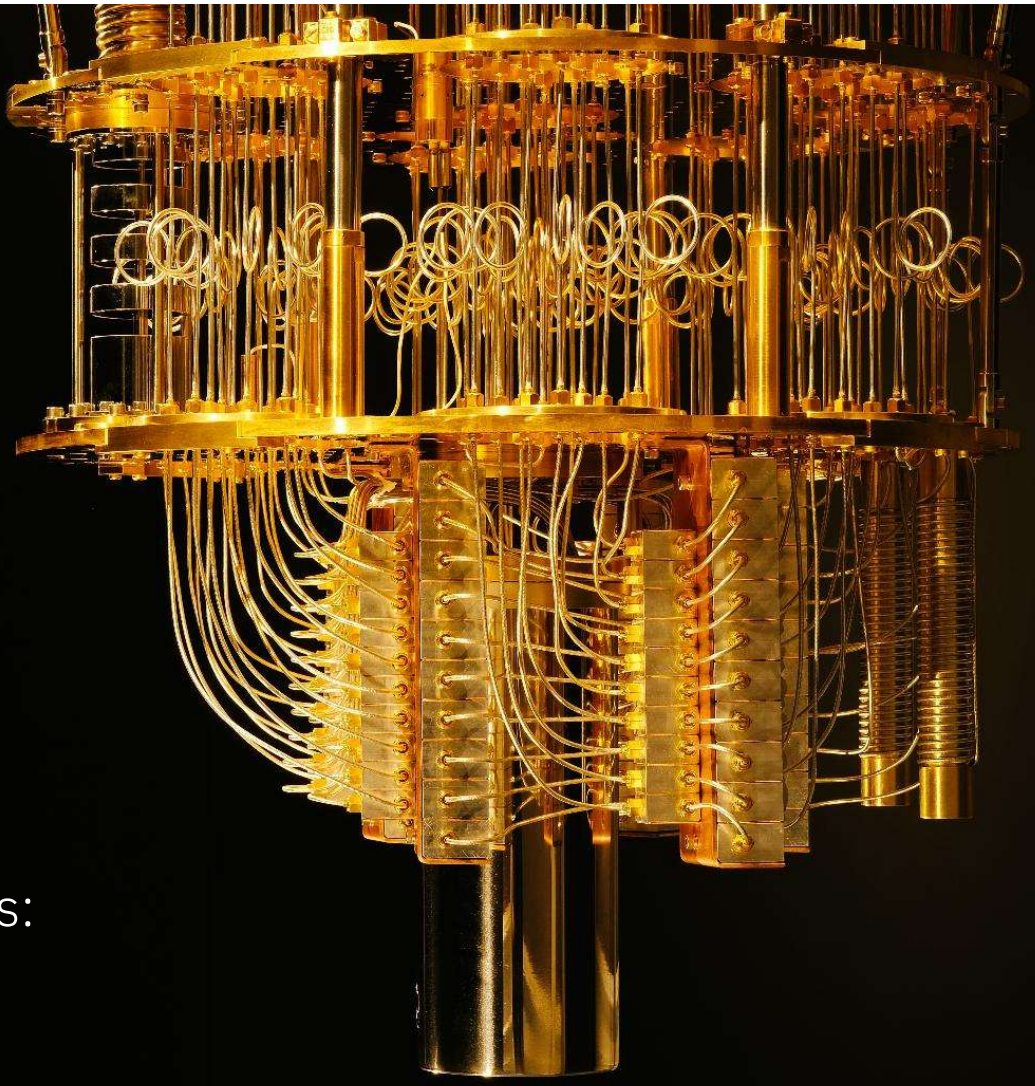


The Quantum Threat

Quantum Computers will be able to break TLS encryption in the future

Harvest Now -> Store -> Decrypt Later

The Quantum Threat



Actions now

Catalog, Prepare

Use TLS 1.3

Use AES-128+ SHA-256+

Actions later

New algorithms protect against this:

- FIPS 203 ML-KEM
- FIPS 204 ML-DSA
- FIPS 205 SLH-DSA

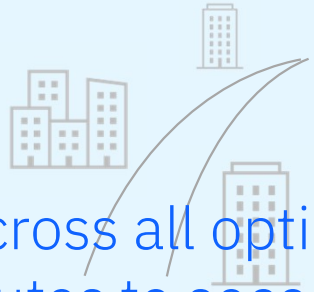
Resilience

highlights

- Native HA in OCP/Kubernetes
- Cross Region Replication (CRR) in OCP/Kubernetes
- Native HA and CRR on Linux (x86-64, IBM Z, POWER System)

IBM MQ High Availability

Queue manager HA options



RPO zero across all options
RTO from minutes to seconds

External solution

System managed HA

Multi-instance queue managers

User builds system from their own HA components

Any distributed platform

Integrated solution

11

z/OS system

Replicated data queue managers RHEL x86

MQ Appliance

Full solution provided as a platform from IBM

Platform specific options

Native solution

Native HA

HA replication and recovery built into the running queue manager

Kubernetes and Linux only

Native HA queue manager for high availability

The primary HA option for IBM MQ, built on cloud native principles

Native replication to simple block storage, performed by the queue manager itself

Multi platform deployment, available both in containers and for Linux on VMs and bare metal

Dependency free “shared-nothing” architecture – no shared storage, no operating system or external HA requirements

Automatic failover between instances via “raft” algorithm for quorum across 3 instances (RTO = seconds)

Optimised synchronous replication to 2/3 of the instances to minimise latency (RPO = 0)

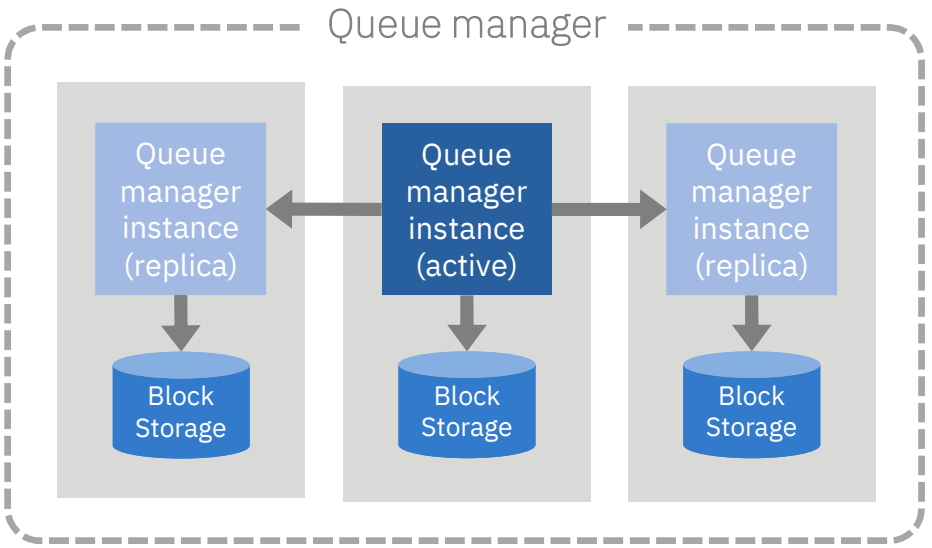
Data movement optimised by only replicating the recovery log

Secure transmission using TLS to align with zero trust security

Mature technology since Native HA is build on IBM MQ’s founding concepts: write ahead logging, channels, and media images

Considerations:

- Linux only
- Requires IBM MQ Advanced



Best for:

- Linux or container deployment
- Low seconds failover
- Minimized dependencies

Supported platforms: RED HAT OPENSHIFT kubernetes Linux VM/bare metal

Native HA – Cross Region Replication

Optimized replication over distances for disaster recovery and maintenance

Optimized asynchronous replication, handles high latency networks between regions

Safe manual failover, ensures partition safety and avoid split brain scenario

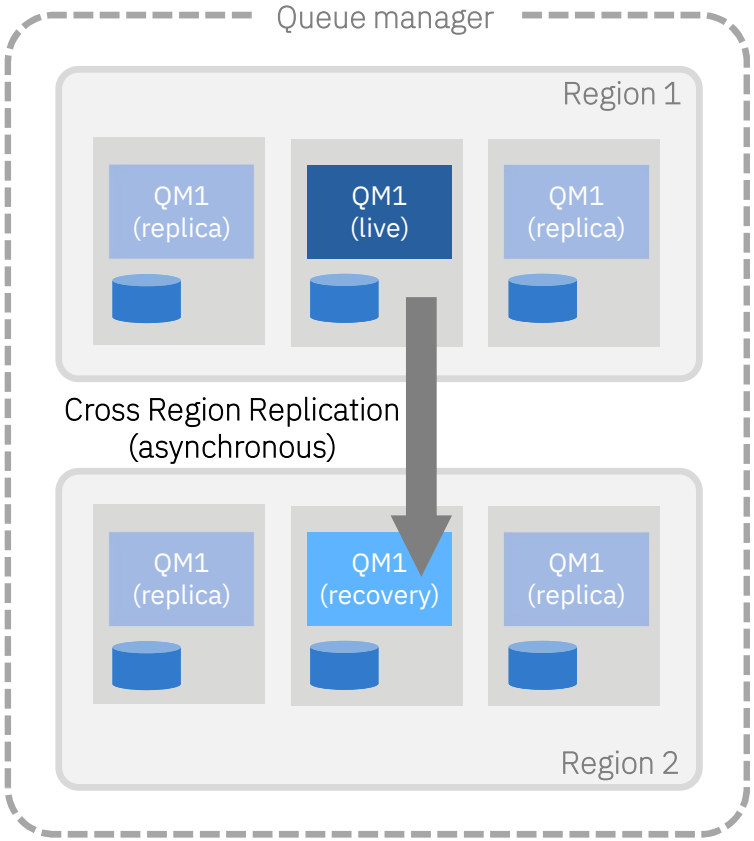
Secure by default, enforced encryption between for cross region communication

RPO=zero for planned switchover, enables seamless maintenance and migration

RPO close to zero for unplanned failover, minimises data loss in disaster recovery scenarios

Consistent approach using the same underlying technology as Native HA

- Native implementation with replication performed by the queue manager itself
- Multi platform deployment, available in containers and for Linux on VMs and bare metal
- Dependency free with no storage or operating system or external HA requirements
- Optimised transmission as only replicates append logs and utilizes compression

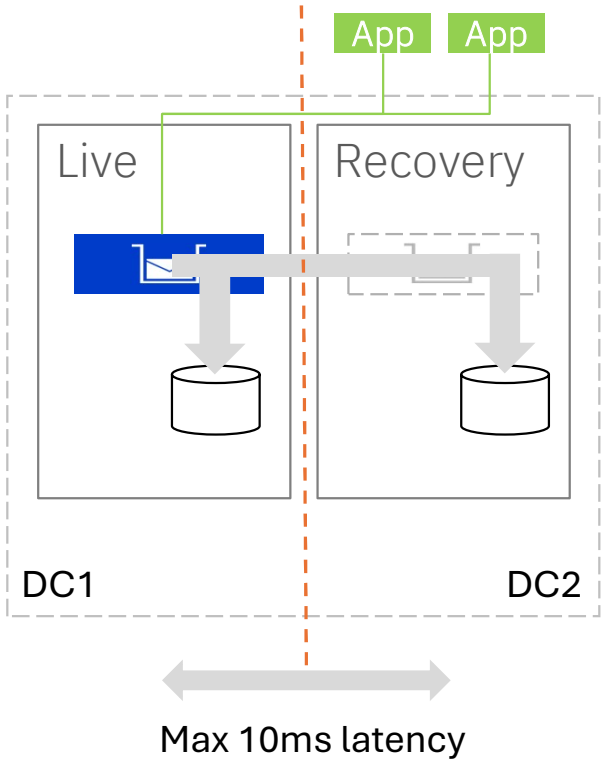


Supported platforms: RED HAT OPENSHIFT kubernetes Linux VM/bare metal

Future plans: Native HA – In-Region Replication

Optimized replication over distances for disaster recovery and maintenance

- Native HA In-region replication to provide a similar topology to an RDQM pair configured for synchronous replication (disaster recovery)
- Live and Recovery roles manually designated through configuration
  - Manual failover (DR)
  - Support a maximum network latency of 10ms (5ms or less ideally)
  - Choice of ‘strict’ or ‘eventual’ data consistency
  - Initially - either Cross-region or In-region, not both



Functional Completeness

core highlights

- Streaming queues
- Apache Kafka connectors

But first – event streaming or messaging?

Using the right tool for the job

In the grey area, it *might* look like **either technology could do the job** — especially for simple patterns. But each technology is optimized for different types of outcomes, and customers get the best results when they align their choice to the design center of the product.

Leading organizations often use both technologies, each where strengths accelerate delivery and reduce operational complexity.

Common deployment scenarios for IBM MQ

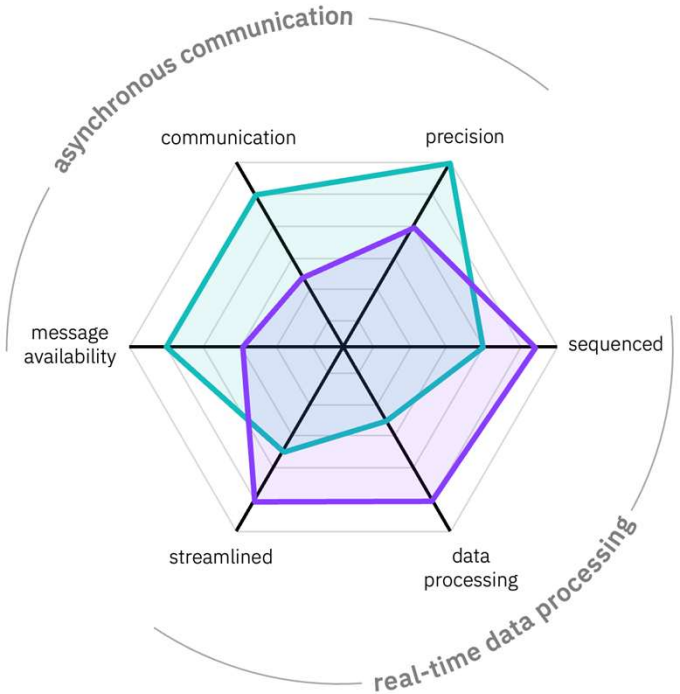
- Coordinating **business-critical application interactions**
- Ensuring **ordered, once-only delivery** for operations and transactions
- Supporting **asynchronous communication** across diverse or distributed systems
- Integrating core systems that need **predictable behavior** and **strong guarantees**

Common deployment scenarios for Kafka

- Streaming **continuous data flows** to many consumers simultaneously
- Enabling **analytics, enrichment, and transformation** in motion
- Building **real-time data pipelines** into data platforms and cloud services
- Supporting event-driven architectures that need **high-throughput event propagation**

Messaging (IBM MQ) excels where **communication** is key

"Make things happen"



Event streaming (Kafka) excels where **data processing** is key

"React to what's happening"

But first – event streaming or messaging?

Using the right tool for the job

What does AI say?

Quick Decision Guide:

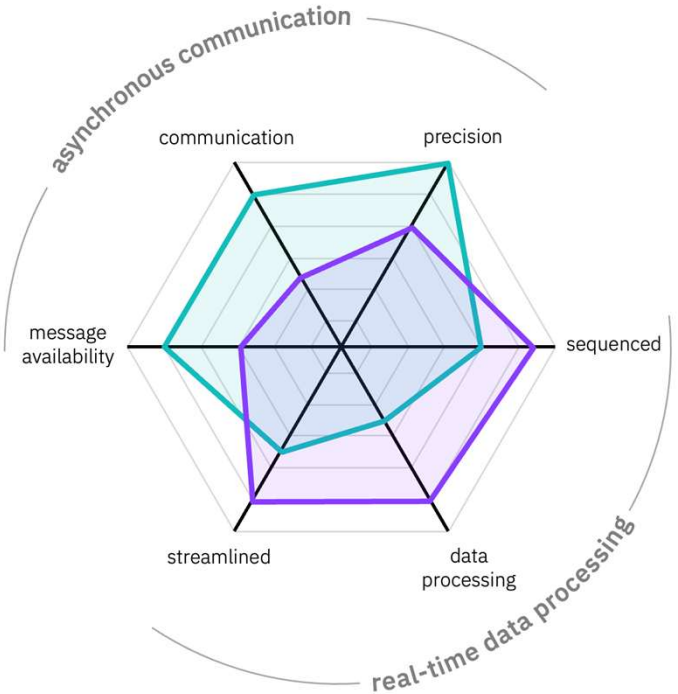
Choose Kafka when you need: streaming analytics, event sourcing, high throughput, data distribution to multiple systems, or building event-driven microservices.

Choose IBM MQ when you need: guaranteed delivery, transactional messaging, request-reply patterns, enterprise integration, or connecting to legacy systems.

Many organizations use both technologies together, with IBM MQ handling critical transactional workflows and Kafka managing high-volume event streams and analytics pipelines.

Messaging (IBM MQ) excels where **communication** is key

"Make things happen"



Event streaming (Kafka) excels where **data processing** is key

"React to what's happening"

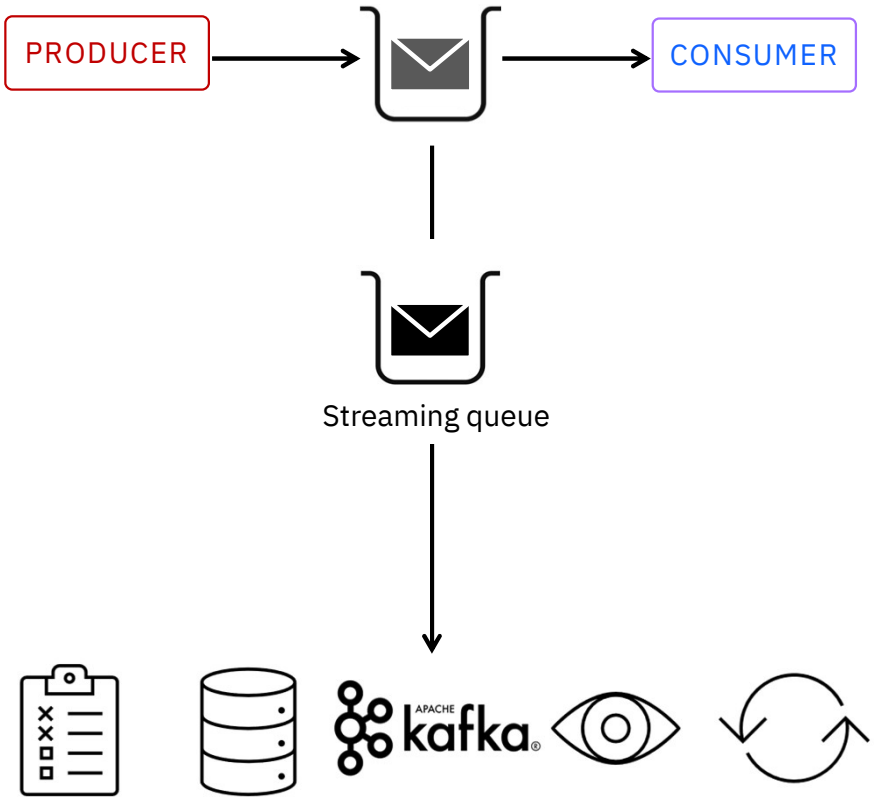
Streaming queues

Tap into the value of existing data flowing through MQ

Zero change to existing applications

Creates a copy of every message sent to the application queue and use them for:

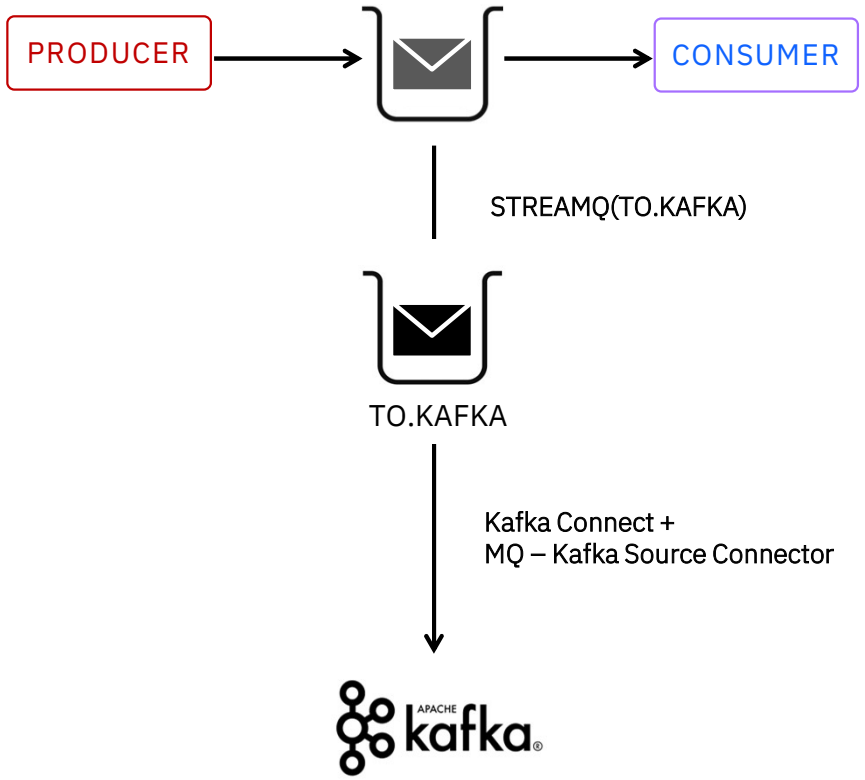
- Stream processing
- Auditing
- Replay
- Investigation
- Test data



MQ - Kafka Connectors

We see many customers wanting to use MQ and Kafka together

Either because they want to stream a copy of existing data moving through MQ into Kafka

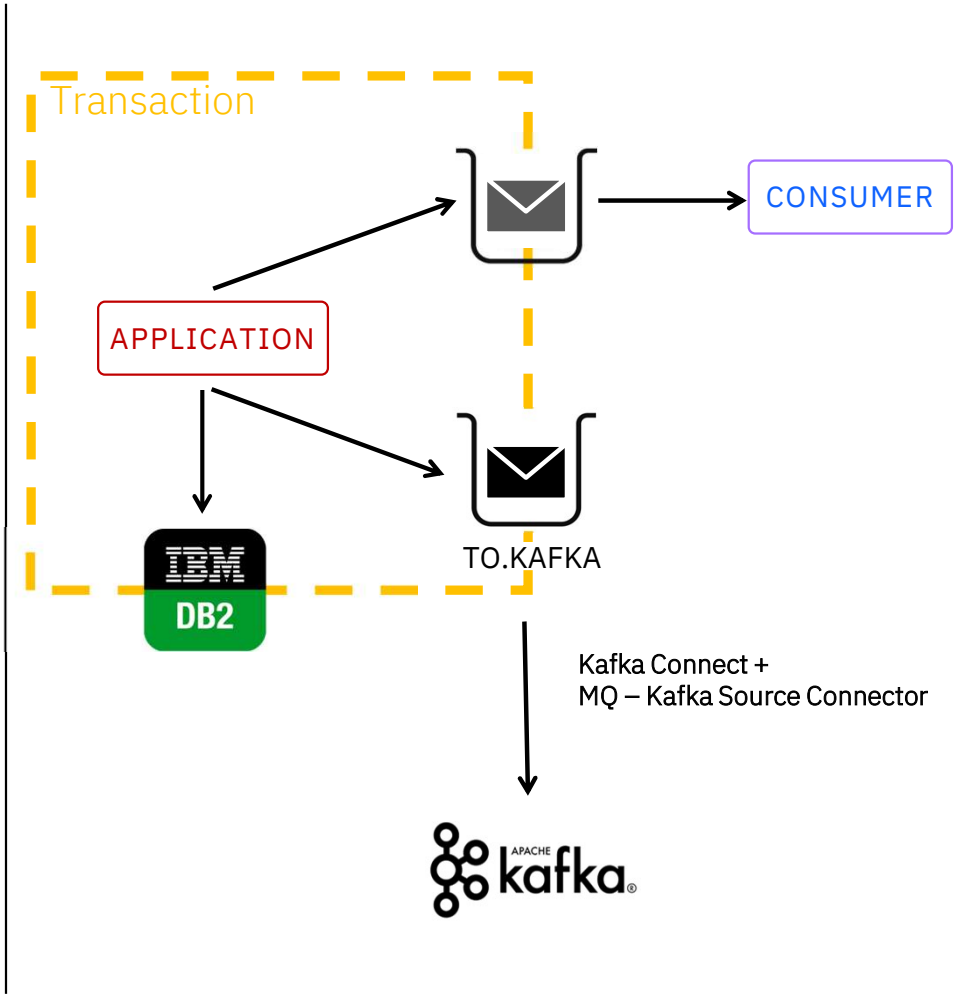


MQ - Kafka Connectors

We see many customers wanting to use MQ and Kafka together

Either because they want to stream a copy of existing data moving through MQ into Kafka

Or because they want a way to get data into Kafka that ensures data is only sent if their transaction commits



MQ - Kafka Connectors

Sink and source connectors exist for connecting MQ queues to Kafka topics

The source connector is most commonly used, enabling MQ to be used as a source for events into Kafka

Connectors are fully supported with one of:

- IBM MQ Advanced / IBM MQ Advanced VUE
- MQ Appliance
- Cloud Pak for Integration
- IBM Event Streams

From 9.4.3 IBM provides a fully supported Kafka Connect runtime in addition to the connectors