

IBM MQ Review

This unit reviews IBM MQ basic concepts and components. It also reviews the runtime options for IBM MQ on-premises and in the cloud.

Unit objectives

- Simplify connections between applications with enterprise messaging
- Understand the deployment options for IBM MQ
- Describe key concepts and terminology related to enterprise messaging and IBM MQ

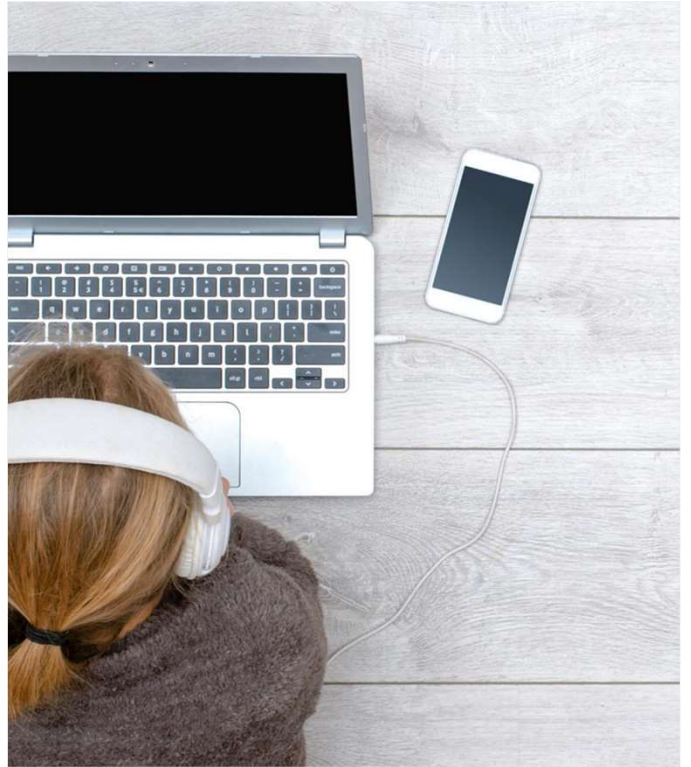
This slide lists the learning objectives for this unit. After you finish this unit, you should be comfortable with these concepts.

Topics

- Why IBM MQ
- IBM MQ Deployment options
- IBM MQ Terminology

This unit contains three main topics.

Topic: Why IBM MQ



This topic explains what IBM MQ is, and why IBM MQ is trusted.

How can independent applications communicate?

Applications can use different methods to exchange data, such as HTTP or REST/HTTP APIs



But what if?

What if B fails while A is waiting for a response?

What if B needs to handle a high workload of different priority requests?

What if the application data is lost?

What if I need to send the application data securely?

What if B is not mine, but instead belongs to a business partner or supplier?

What if applications use different processors? operating systems? communication protocols?

Applications can exchange data using several different methods, but direct communication between two independent applications has some drawbacks. Many of the disadvantages of direct, synchronous, communication are felt by application developers, who must create and adapt their code for flexibility and scalability.

As an example, one way to directly connect applications is with HTTP. It is relatively easy to connect two applications together with HTTP, but HTTP is characterized as synchronous in nature. In this context, synchronous means that to exchange data, **both** the sending and receiving applications must be available and ready to execute. Each HTTP request must receive a corresponding response. Direct, synchronous communication imposes limitations on the sending and receiving applications, for example:

- Both applications A and B must always be available for A to continue.
- A cannot do anything while B is processing A's request.
- What if B fails while A is waiting for it to complete? Is the entire transaction lost?
- What if B needs to handle a high workload of different priority requests?

HTTP APIs, such as REST-like interfaces are also an effective way to expose and consume data. HTTP APIs can be secured easily, and they can be called by internal and external developers. However, REST APIs are also synchronous.

To achieve stable communication using REST and HTTP APIs, your application developers would need to add retry or failover logic to the applications; otherwise the requests could be lost and information might be missed. This can become overly complex and lead to other bottlenecks, for example, if an application or microservice is constantly retrying, so it cannot move on to process anything else. It **is** possible to implement asynchronous communication over HTTP, but then the simplistic and the ubiquitous nature of HTTP is lost.

Another challenge with direct communication is that your applications and systems are not all identical. Applications written last week need to communicate with applications that were written 20 years ago.

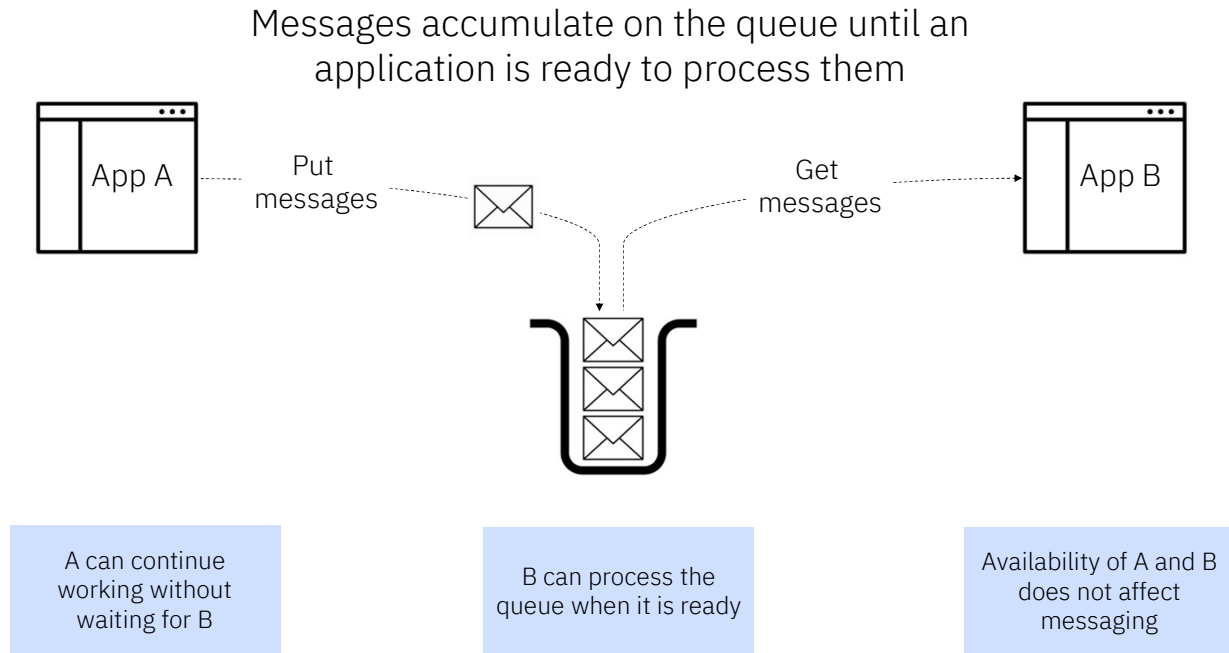
Another risk in this example is that the App A developers are writing code that is specific to the receiving application. This

might not be a problem if App A and App B both belong to the same development team, or if App B never changes, but what if the sender and receiver belong to separate teams--or separate enterprises? Direct communication tightly couples these applications together. As systems become more tightly coupled, their reliance on each other increases. The cost of a failure increases. Scaling systems independently to respond to changing requirements becomes unmanageable. Adding new connections can be labor-intensive and expensive.

Terminology:

- **HTTP:** Hypertext Transfer Protocol
- **REST:** Representational State Transfer
- **API:** Application Programming Interface

What is message queuing?



Message queuing is a communication style in which data is packaged in messages, which are stored on queues. Message queuing allows applications to communicate indirectly through queues, rather than calling each other directly.

Queues allow applications to put and get messages as required so that applications can run independently of each other, at different speeds, at different times, and in different locations. Another aspect of messaging-style communication is that no logical connection between the two applications required: applications only need to connect to a queue.

With message-based application connectivity, all the work of connecting, polling, handling failure, and implementing change is removed from the application. Instead, message-oriented connectivity provides a dedicated layer for handling communication. It is decoupled from the sending and receiving application, which provides the flexibility to react to business conditions.

Messaging uses an asynchronous model, which means that an application that generates messages does not have to execute at the same time as an application that consumes those messages. Reducing the requirements for simultaneous availability reduces complexity and can improve overall availability. Messages can be sent to specific applications or distributed to many different applications at the same time.

Using message-based connectivity allows your developers to concentrate on the business problem instead of worrying about matters such as recovery, reliability, and operating system differences.

Enterprise messaging challenges

Getting data where it needs to be in your business: reliably, securely, and quickly

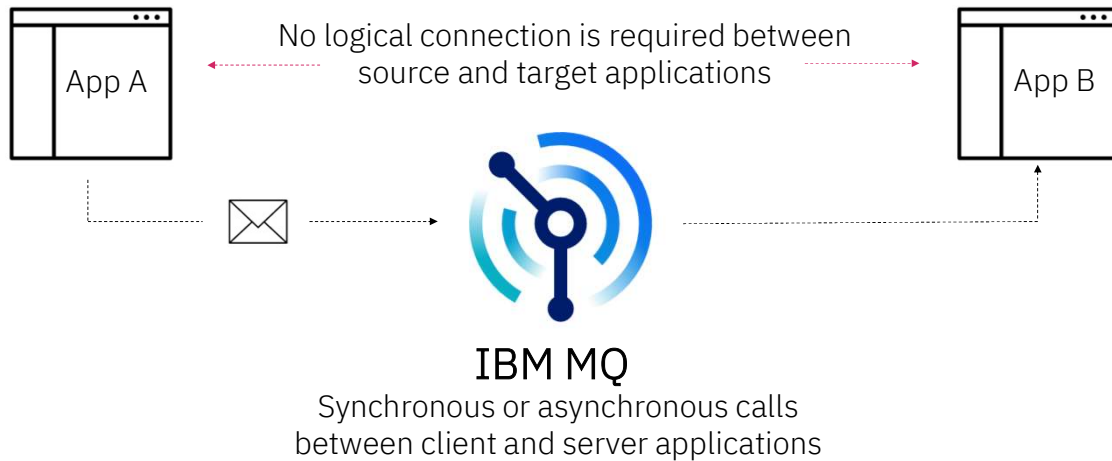
- I need everything to communicate, regardless of age or language
- I need to know what happened with my data
- Parts of my business are in the Cloud
- My data is sensitive, it must be secure
- I need to integrate with partners and external organizations
- I cannot afford ANY data loss
- I cannot afford downtime
- Business imperatives and environments change fast

Enterprises in a connected economy face many technical and business challenges. Enterprises need a flexible infrastructure to remain competitive. Some of the challenges that face a connected business include:

- The need to connect and integrate older systems and software with new technologies.
- The ability to track data and ensure that data is secure.
- The flexibility to support parts of the business in the Cloud.
- Data sources in the enterprise can include value chain extranets, databases, the Internet, customers, transactions, and existing systems and applications.
- The amount of data that is exchanged between these entities is also increased, and organizations must be ready to accept high amounts of information.
- Old “batch” processes, if not disappearing, are diminishing as the requirements for real-time and on-demand information increases.
- Applications must be resilient and capable of handling failure, yet users keep expecting the same if not a better and quicker response than when they used to have a direct connection to the data.
- Customers and business partners have higher expectations, and the inability to meet the expectation results in a loss of business.
- Extending existing processes would get prohibitively expensive; applications and infrastructures need to be flexible so that adapt to changing requirements.

What is IBM MQ?

IBM MQ is enterprise-grade messaging software that enables independent applications to send messages to each other



IBM MQ is the market-leader in messaging integration software. Originally introduced in 1993, it has always provided an available, reliable, scalable, secure, and high-performance transport mechanism. IBM MQ is powerful enterprise messaging software that underpins a range of industry-types worldwide.

IBM MQ's effective messaging solves the following common business problems, among others:

- If one application fails, many other applications fail. It costs the business a lot in downtime and recovery.
- An organization's applications run on different hardware and operating systems, and are written in different programming languages.
- A process was originally designed for one purpose, but now it must change to meet new requirements without breaking other applications.
- The business expanded and the capacity of the system can no longer cope with the workload demand.

IBM MQ communication is fast, and because the messaging is asynchronous, the business has the assurance that data will not be lost. With MQ, data arrives on time, in order, and once only. IBM MQ offers a range of Qualities of Service.

IBM MQ includes administrative tools that simplify the management of the MQ components and the messages that it processes.

IBM MQ provides asynchronous messaging for a wide range of systems. Applications that were written in an obsolete language can communicate with those applications that are written in the current popular languages. That value of universality is core to the product, and this value has not changed in all the time that IBM MQ has been available.

An application developer has a choice of programming interfaces and programming languages to connect to IBM MQ. The applications connecting to IBM MQ can be written in any one of many different programming languages and to many different APIs: from C and COBOL, to Java and .Net to NodeJS and Ruby.

IBM MQ connects virtually any commercial IT system, enabling communication across a broad range of computing

platforms, application environments and APIs, and communications protocols for security-rich message delivery.

IBM MQ enterprise messaging features (1 of 2)



Insulate your business from risks

Unlike its competitors, IBM MQ will never lose a message or deliver a message more than once.



Simple multi-style messaging

Connect diverse systems with support for message queueing, events or PubSub, and making transactions.



Market-leading messaging, everywhere

Get the same proven, security-rich and high-performance messaging solution, however and wherever you deploy.



“Always-on” cloud native messaging

Minimal infrastructure and a light footprint make MQ perfect for cloud-native, highly available deployments.



24x7x365 technical support

A dedicated IBM MQ team is available to help you, along with a large user community.



Secure, enterprise-wide connectivity

Keep data safe with TLS secured communications, identity access management, message-level security and more.

This slide lists some advantages of IBM MQ. The list is continued on the next slide.

Terminology:

- **PubSub:** Publish-subscribe
- **TLS:** Transport Layer Security

IBM MQ enterprise messaging features (2 of 2)



Flexible deployment options

Deploy to public or private cloud in virtual machines or containers on Docker, Kubernetes/Cri-O or Red Hat® OpenShift®.



Uniform clusters

Automated and intelligent workload balancing lets you design applications for scale.



High availability, designed to be easy

Keep your business going with high availability that's simple to set up and offers rapid recovery times.



A way to unlock the value of existing data

Harness mission-critical data as it flows around the enterprise to power analytics and AI.



Security by design

Protect data from threats with user access management, TLS-secured communications, and more.



Freedom to develop

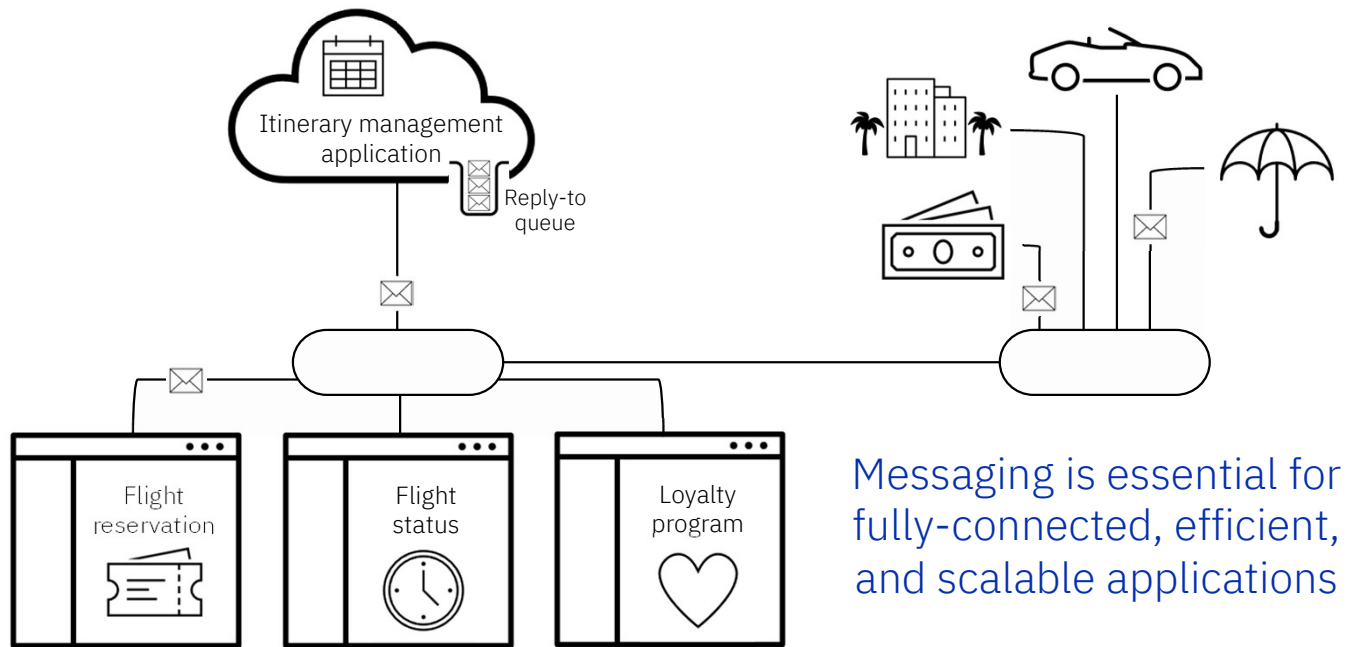
Broad language, API and messaging protocol support (including AMQP) plus simple management tools.

This slide lists more advantages of IBM MQ. The list is a continuation of the preceding slide.

Terminology:

- **CRI-O:** Container Runtime Interface using OCI (Open Container Initiative)
- **TLS:** Transport Layer Security
- **AI:** Artificial Intelligence
- **API:** Application Programming Interface
- **AMQP:** Advanced Message Queuing Protocol

IBM MQ messaging in action



This figure shows an example of IBM MQ messaging at work in the travel industry. Imagine all of the data that must pass between applications when a customer makes a travel reservation using an airline's website.

To build an itinerary, the itinerary management application must communicate with internal programs, such as the flight reservation application and the customer loyalty application. The management application also needs to exchange data with external systems such as credit card payment systems, auto rental providers, and travel insurance providers. The data flowing between the applications in this example is time sensitive and it contains financial data. This use case is an excellent fit for IBM MQ.

IBM MQ's parallel processing ability allows an application to send several requests without waiting for a reply to one request before sending the next. All the requests can then be processed in parallel. In the example in the figure, the itinerary management application must make a number of requests, such as making a rental car or hotel reservation. By using queues, these steps do not have to be completed serially.

The application can process the replies when they are all received, and produce a consolidated answer. The program logic might also specify what to do when only a partial set of replies is received within a specified period.

Who uses IBM MQ?

90%

90% of the top 100 in the 2022 Forbes Global 2000.

80%

80% of the top 10 oil and gas companies in the 2022 Forbes Global 2000.

70%

70% of the top 10 healthcare companies in the 2022 Forbes Global 2000.

60%

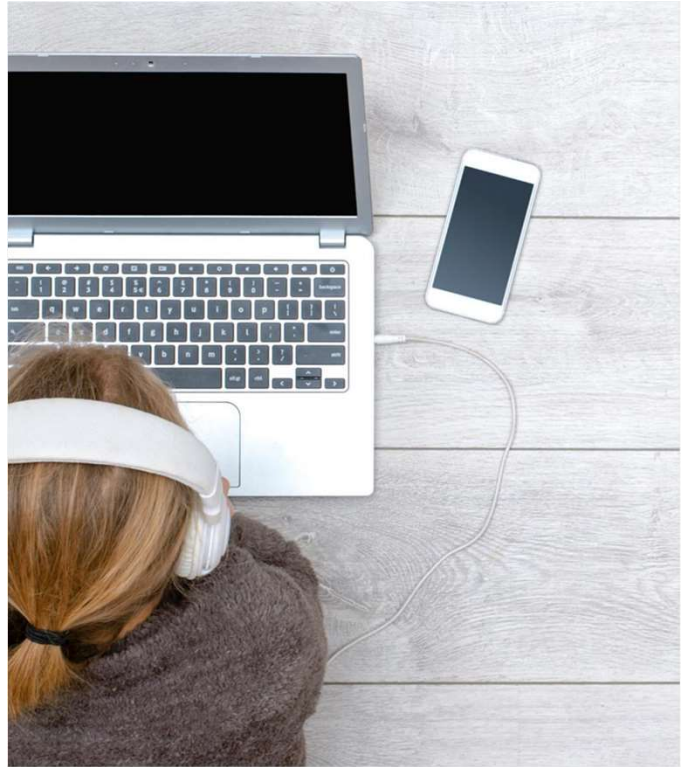
60% of the top 10 media companies in the 2022 Forbes Global 2000.

Source:
<https://www.ibm.com/products/mq>

Thousands of businesses worldwide depend on IBM MQ. Customers in almost every industry value and trust IBM MQ, because MQ:

- Provides a reliable solution for bridging the components in a service-oriented architecture
- Connects various IT systems across a wide range of hardware and operating systems
- Builds on 20+ years of messaging expertise and continued IBM investment
- Helps you share and exchange critical business information with ease, confidence, and security

Topic: IBM MQ Deployment options



This topic lists and describes the different ways you can deploy IBM MQ, such as in VMs, containers, or other platforms.

Where does IBM MQ run?



Software: Traditional software deployment on bare metal or virtualized operating systems. Provide **automated deployments, simplified HA/DR/Scalability, integration into self service portals and DevOps.**



Containers: Embrace container technology to accelerate the **developer and operational agility** of the organization. Deploy across on-premise and public cloud providers, utilizing the platform for **self-service, DevOps, Availability and Scalability.**



MQ Appliance: Consolidate the IBM MQ estate onto an **optimized hardware appliance.** Simplify operational activities by benefiting from **predictable performance** and out of the box **high availability and disaster recovery** topologies.



Software as a Service: Allow **IBM to manage** the deployment, availability and system monitoring of the IBM MQ estate with **IBM and AWS Clouds.** Empower developer teams to provision MQ within minutes, reducing dependencies across teams.

IBM MQ is available in several form factors, so you can run IBM MQ in a way that makes sense to your business.

Software

IBM MQ software is the most common option because it is the most versatile. Software can be deployed to bare metal or virtual machines running on premises in a private data center or in public clouds including AWS, Azure, Google Cloud, and so on. Software is a great option for adoption of hybrid-multi cloud strategies. IBM MQ runs where applications are running, so MQ software is supported on a vast range of platforms including Windows, Linux on Power, Linux x86, Linux on z/OS, z/OS, AIX, HPE Non-Stop, and more.

Containers

MQ software can also be deployed in containers. Containers allow you to package an IBM MQ queue manager or IBM MQ client application, with all its dependencies, into a standardized unit for software development. You can run IBM MQ in a pre-packaged container provided in IBM MQ Advanced and IBM MQ Advanced for Developers. This IBM MQ Advanced certified container offers a supported image and Helm chart that can be used to deploy a production-ready IBM MQ image into Red Hat® OpenShift, or IBM Cloud Kubernetes Service. IBM MQ can also be run in an IBM Cloud Pak for Integration container, or in a container that you build.

MQ Appliance

You can also run IBM MQ in an appliance. A powerful and efficient hardware appliance, IBM MQ Appliance eases IBM MQ setup and maintenance. All the hardware and software is installed on a single box, reducing Total Cost of Ownership (TCO) of a messaging estate. High availability and disaster recovery are out of the box. The hardware and software of the IBM MQ Appliance has been tuned by IBM MQ experts for enhanced performance.

Software as a service (SaaS)

IBM MQ is also available as a managed service, called IBM MQ on Cloud. With this managed service, IBM automatically updates and patches the service, runs the operations, and provisions onto either IBM Cloud or AWS. This solution is a good fit for enterprises who have embraced cloud computing and want to focus on the logic of connecting applications and

other endpoints.

Terminology:

- **HA/DR:** High Availability (HA) means that minor failures or individual server failures won't take the monitoring system offline. Disaster Recovery (DR) involves a set of policies, tools and procedures to enable the recovery or continuation of vital technology infrastructure and systems following a natural or human-induced disaster.
- **DevOps:** Development Operations is a methodology in the software development and IT industry. Used as a set of practices and tools, DevOps integrates and automates the work of software development and IT operations as a means for improving and shortening the systems development life cycle.
- **IT:** Information technology is the use of computers to create, process, store, retrieve and exchange all kinds of data and information. IT forms part of information and communications technology.
- **AWS:** Amazon Web Services
- **AIX:** Advanced Interactive eXecutive
- **HPE:** Hewlett Packard Enterprise

IBM MQ connects data within and across **hybrid multi-cloud**, ensuring mission-critical applications, wherever they are, get the data they need.

Private datacenters

Software for VMs or containers and a dedicated **MQ Appliance**

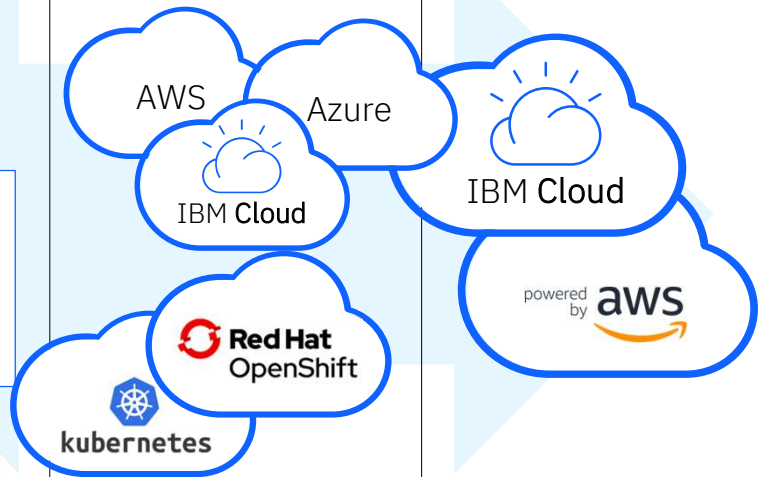


IBM Z	Linux
AIX	
Windows	IBMi
HPE NonStop	zLinux
Appliance	



Public clouds

Client-managed software deployed to **VMs or in containers**



Software as a Service (SaaS)

IBM-managed SaaS, hosted in **IBM Cloud and AWS public cloud**

This figure shows where IBM MQ can be deployed. IBM strives for consistency across all these form factors so you can use the same skills and have the same experience wherever and however you deploy IBM MQ.

Terminology:

- **VM:** Virtual machine
- **AWS:** Amazon Web Services
- **SaaS:** Software as a service

IBM MQ on z/OS

Market-leading messaging combined with the market-leading IBM Z.

Available as IBM MQ Advanced for z/OS and IBM MQ Advanced for z/OS VUE.

Key takeaways:

- A focused set of Messaging Offerings available for the z/OS platform
- IBM MQ for z/OS provides strong integration with all other major software offerings running on z/OS
- IBM MQ for z/OS has unique qualities, a product that has been built to exploit the strength of the underlying platform



- Low latency - The problem of latency tends to be additive in high volume, repetitive transactions
- Very secure - Data can not be sniffed, intercepted or modified with built-in AMS
- Avoid encryption / decryption overhead - Since exchange is so secure, the need to encrypt maybe eliminated
- Security identity assertion across interface - Avoid coding identity "aliases" in different locations across enterprise
- Avoid TCP/IP stack processing overhead - Reduces overall system CPU usage
- Single thread of execution across interface - Avoid task switching overhead
- Natural fit within each execution environment - Same MQ API regardless of environment
- Reduced complexity for debug and troubleshooting - When sender and receiver are in same OS environment, one set of tools may be used

IBM MQ for z/OS brings the strength and dependency of IBM MQ software to IBM Z (the mainframe). This enables you move data anywhere in that environment, and take advantage of several benefits, including:

- Native connectivity to z/OS applications.
- You can use shared queues and queue sharing groups, to implement high availability of IBM MQ resources. Shared queues and queue sharing groups are functions unique to IBM MQ for z/OS on the z/OS platform.
- You can control IBM MQ and enter control commands on z/OS by using ISPF panels. IBM ISPF for z/OS is a development tool set for IBM System z that provides host-based software development, including software configuration management.
- On z/OS, MQ uses its System Authorization Facility (SAF) to route requests for authority checks to an external security manager such as the z/OS Resource Access Control Facility (RACF). RACF provides the tools to manage user access to critical resources.

Terminology:

- **VUE:** Value Unit Edition
- **AMS:** Advanced Message Security
- **TCP/IP:** Transmission Control Protocol/Internet Protocol
- **CPU:** Central processing unit
- **API:** Application Programming Interface
- **OS:** Operating system

IBM MQ software in containers

MQ Operator and MQ Advanced Certified Container

Certified container

Production ready container image with built-in options for HA.

Automatically configure MQSC, certificates and QM.INI at container start-up.



Operators

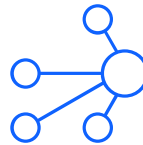
IBM MQ Operator simplifies the administration, allowing MQ to be deployed, updated and managed as a K8s native resource.



Loosely coupled

Building large networks of messaging servers is fundamental to MQ.

MQ Clustering enables building of dynamic networks with automatic routing from source to target.



Light footprint

MQ is designed to be lightweight and to scale to run at any size. IBM itself maintains ½ core queue managers in production.



IBM MQ is available in containers. Generally speaking, there are two ways you can choose to deploy MQ software in containers:

1. Use IBM MQ Advanced certified container operator: These IBM MQ Advanced certified container images are designed to run in Red Hat OpenShift, and are deployed by an OpenShift operator. These containers can also run in OpenShift as part of the IBM Cloud Pak® for Integration.
2. Build your own container: You can create your own container images. This is the most flexible container solution, but it requires you to have strong skills in configuring containers, and to "own" the resultant container. Visit the following websites for more information about building your own containers.
 - <https://github.com/ibm-messaging/mq-container/blob/master/docs/building.md>
 - <https://github.com/ibm-messaging/mq-helm>

Terminology:

- **HA** High availability
- **MQSC**: MQ Script Command
- **K8s**: Kubernetes
- **QM.INI**: A queue manager configuration file, **qm.ini**, contains information relevant to a specific queue manager

Cloud native messaging with IBM Cloud Pak for Integration

Builds on the capabilities within the MQ Advanced container

Cloud native deployments

Deploy IBM MQ using the MQ Operator, simplifying the administration, deployment, updating and management.

Cloud native high availability

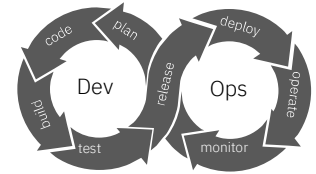
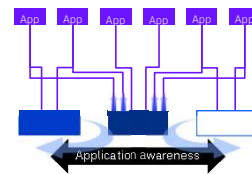
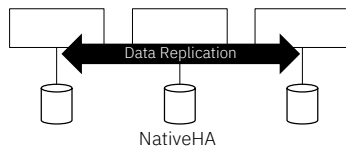
Exclusive access to MQ NativeHA, providing cloud native high availability only available to Cloud Pak for Integration clients.

Cloud native scalability

Active / Active topologies, transparently scaling MQ with zero down time to handle the most demanding workloads in the world.

Cloud native DevOps

Proven DevOps patterns to assure a consistent, agile, and repeatable experience, built on the latest Cloud Native DevOps tooling.



IBM Cloud Pak for Integration (CP4I) brings IBM's leading integration solutions including IBM MQ, IBM App Connect, IBM API Connect, IBM Data Power Gateway Virtual, IBM Event Automation, and IBM Aspera software into a single, unified solution where you can start small and grow.

IBM MQ is a core part of the IBM Cloud Pak for Integration. It provides the assured connectivity that mission critical applications require. Enterprises who are running IBM MQ on other form factors can purchase the IBM Cloud Pak for Integration and continue to deploy where they are today, while having the flexibility to utilize other parts of the Cloud Pak and plan for their journey to containers on Red Hat OpenShift.

Terminology:

- **HA:** High availability
- **DevOps:** Development Operations is a methodology in the software development and IT industry. Used as a set of practices and tools, DevOps integrates and automates the work of software development and IT operations as a means for improving and shortening the systems development life cycle.

IBM MQ appliance

MQ Advanced in a box!

A preoptimized messaging appliance with end-to-end encryption; configured to get the most out of IBM MQ



Reduces TCO – consolidate existing MQ estate, plus reduced spend on configuration and maintenance.

Simple maintenance – firmware updates with fixes and features, plus simple licensing!

Business continuity – built-in support for HA/DR, **simple to set up and scale** ... just add another appliance!

Specific for IBM MQ – hardware built specifically for MQ. No need to configure servers or OS in order to get going.

Preoptimized – tuned and tested by the experts.

Enhanced security – no files, nor user exits and Advanced Message Security as standard.

The IBM MQ Appliance provides the application connectivity performance of MQ software in a physical messaging appliance. It offers rapid deployment of enterprise messaging with easier administration. Performance and message throughput are optimized for the appliance's capability and configuration.

The MQ Appliance is a tamper proof appliance. It includes MQ Advanced Message Security, which provides built-in end-to-end data encryption of messages in motion and at rest.

The MQ Appliance is pre-tuned and optimized to provide the best performance. The appliance works seamlessly alongside other MQ deployments.

The IBM MQ Appliance is often used in these cases:

- **Consolidation and repeatable deployment:** The MQ Appliance enables you to consolidate MQ projects onto a single unit comprised of both hardware and firmware. It is pre-configured specifically for MQ use, and installation and maintenance are simple. This reduces time and expertise needed to use it. With fewer appliances, maintenance is simplified and standardized, as there are less components to update.
- **Out of the box HA/DR:** The MQ Appliance provides a High Availability and Disaster Recovery (HA/DR) solution out-of-the-box, by failing over between paired appliances. This is far simpler to set up than HA/DR on MQ software plus supporting hardware/cloud. If you require additional failover appliances, it is as simple as connecting the appliance and mirroring settings from an existing appliance.

Terminology:

- **TCO:** Total cost of ownership
- **HA/DR:** High Availability (HA) means that minor failures or individual server failures won't take the monitoring system offline. Disaster Recovery (DR) involves a set of policies, tools and procedures to enable the recovery or continuation of vital technology infrastructure and systems following a natural or human-induced disaster.
- **OS:** Operating System

Software as a Service: IBM MQ on Cloud

The IBM MQ on Cloud (MQoC) service enables clients to use IBM MQ as a managed offering.

IBM will handle upgrades, patches and many of the operational management tasks, allowing clients to focus on the integration of MQ with their applications.

- Managed for clients
- Security-rich message delivery
- Pay per workload
- Enabled for multi-cloud

Client configured
and monitored

Queues, topics, channels,
clustering, applications

IBM-managed
and operated

MQ installation, basic configuration,
security, maintenance

Hardware, virtualization,
servers, network, storage



Get up and running with MQ in minutes

- Administrators can provision additional MQ capacity in under 10 minutes, instead of months
- Relieves the MQ admins of their dependency on other teams



Admins can stay MQ focused

- Clients no longer have to maintain, monitor, upgrade or patch their systems running MQ
- Reduces the total cost of ownership of MQ by ~40%



Seamlessly scale MQ

- Clients can scale the size of their MQ estate hourly along with their application needs

IBM MQ on Cloud service enables you to use IBM MQ as a managed offering. IBM will handle upgrades, patches and many of the operational management tasks, allowing your team to focus on the integration of MQ with their applications.

- **Managed for you:** Enables administrators to adapt infrastructure to business needs and gives developers the freedom to build applications without worry.
- **Security-rich message delivery:** Helps protect data on the move and at rest, without altering applications, to reduce risk to your business and your customers.
- **Pay per workload:** Ensures business continuity for the ultimate quality of service with once-and-only-once delivery that's equipped for failover.
- **Enabled for multicloud:** Uses only a single standard API for multicloud connectivity. Deploy to IBM Cloud or AWS and use the same administration experience across all MQ deployments for simple, rapid multi-cloud messaging management.

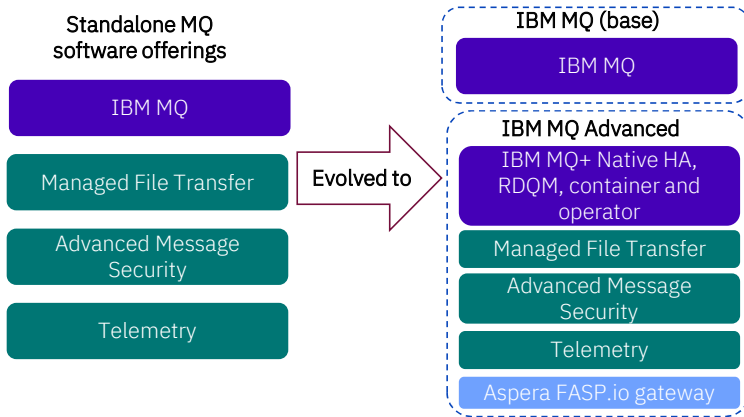
Terminology:

- **SaaS:** Software as a service
- **AWS:** Amazon Web Services

IBM MQ Advanced

IBM MQ Advanced was created in 2015 to include Managed File Transfer, Advanced Message Security and Telemetry features in a single package.

IBM MQ is still a stand-alone offering if additional features are not required.



Advanced Reliability

- Native HA for cloud native 3-node quorum HA data replication, supported in multiple Kubernetes environments (RHOS OCP, Tanzu, etc) and available through MQ Advanced and CP4I
- Replicated Data Queue Managers (RDQM) failover on Linux x86-64 with reduced network storage costs
- Pre-packaged MQ container and Operator to effortlessly deploy a production-ready IBM MQ image.



Advanced Connectivity

- IBM-supported Sink and Source connectors for Apache Kafka
- Send files over the MQ network with Managed File Transfer
- IBM Aspera fasp.io Gateway for rapid message transfer - increasing existing network capacity
- MQ Telemetry for connectivity via MQTT to remote sensors, actuators, and telemetry devices.



Advanced Security

- Full end-to-end-encryption for unrivaled security of company and client data
- Compliance with regulations and confidence for audits

IBM MQ Advanced is IBM MQ software with additional features including end-to-end message encryption and data integrity, and options for improved availability and disaster recovery.

IBM MQ Advanced is a single license entitlement that, in addition to IBM MQ itself, provides entitlement to:

- Managed File Transfer
- Advanced Message Security
- IBM MQ Telemetry Transport
- Replicated Data Queue Manager
- Bridge to blockchain
- IBM MQ Advanced certified container
- Kafka connectors

Terminology:

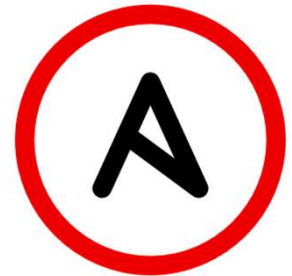
- **HA:** High availability
- **RDQM:** Replicated data queue managers
- **FASP:** Fast Adaptive and Secure Protocol
- **AWS:** Amazon Web Services
- **RHOS OCP:** Red Hat OpenShift OpenShift Container Platform
- **CP4I:** Cloud Pak for Integration
- **MQTT:** Message Queuing Telemetry Transport

MQ-Ansible

Use Red Hat Ansible to automate the installation and configuration of IBM MQ on Ubuntu, Red Hat, Windows and IBM AIX machines



<https://github.com/ibm-messaging/mq-ansible>



MQ-Ansible is an automation tool designed to streamline the process when you need to deploy IBM MQ on distributed operating systems.

MQ-Ansible is a collection of Ansible playbooks, modules, and other objects. The playbooks and roles in this collection deliver an installation of IBM MQ Advanced on a target machine. The roles have been implemented to set up the required users on the machine, download the software, install and configure IBM MQ, copy over a configuration command file ready to be run on the target machine, and setup and start the web console. Developers can change this file to customize the configuration of their queue managers.

Terminology:

- **AIX:** Advanced Interactive eXecutive

Topic: IBM MQ Terminology



This topic defines some IBM MQ terms. You learn more terms later in this course; this topic describes MQ basic terminology only.

Messages

- A message is a collection of data sent by one application and intended for another application
- Messages consist of three parts:
 - **MQ message descriptor (MQMD):** Identifies the message and contains additional control information; added by your applications or IBM MQ
 - **Additional properties:** An optional set of user-definable elements that describe the message; your applications can view and modify these properties
 - **Payload:** The actual data sent from an application, such as a bank transaction or healthcare record



Messages can be persistent or non-persistent

The application data that MQ exchanges is sent in the form of messages.

A message is a chunk of data sent by one application and intended for another application. A message is any information that one application uses to communicate to another. A message can convey a request for a service, or it can be a reply to a request. A message might also report on the progress of another message: to confirm its arrival or report on an error, for example. A message can also carry information for which no reply is expected.

Messages consist of the MQ message descriptor (MQMD) and the payload. Optionally the application can also provide message properties:

- The MQ message descriptor (MQMD) contains information about the message. For example, the MQMD contains information about the sending application. Other examples of information in the MQMD are the type of message and the priority that is assigned to the message by the sending application.
- Message properties are an optional set of user-definable elements that describe the message without being part of the payload. Applications that receive messages can choose whether to inspect these properties.
- The message data portion of an MQ message contains the actual payload, such as a bank transaction or healthcare record. The sending application determines the contents of the message data portion of the message. The structure of the data is defined by the application programs that use it, and MQ is mostly unconcerned with its format or content.

The sending application supplies both the message descriptor and the application data when it puts a message on a queue. The application that puts the messages on the queue sets some of the fields in the message descriptor; the queue manager sets others on behalf of the application. Both the message descriptor and the application data are returned to the application that gets the message from the queue.

Message persistence

Persistent messages

- Stored to disk
- Queued messages are recovered following a system failure
- Recovered from logged data after a queue manager restart if disks are intact

Non-persistent messages

- Stored in system memory as much as possible (better performance)
- Queued messages are lost when a server fails or restarts
- Discarded when a queue manager stops for any reason

Granular by design

- Persistence is a field in the MQ message descriptor (MQMD)
- This means application developers and IBM MQ administrators can control persistence on a granular, message-by-message basis

The MQMD (MQ message descriptor) contains a persistence attribute that controls whether a message survives restarts of the queue manager.

If the application identifies the message as persistent, IBM MQ keeps a failure-tolerant recovery log of all actions on that message. If a queue manager restarts before the message is delivered to its target, the message is recovered from the logged data. If an application contains critical business data that must be reliably maintained, it should identify a message as persistent. Examples of persistent messages are stock trades or banking transactions.

If the application identifies the message as non-persistent, the message is stored in system memory only. If a queue manager restarts before the message is delivered to its target, the message is discarded. An application can identify a message as non-persistent when the loss of data is not crucial.





When planning your MQ deployment, consider that persistent messaging does not fit every use case:

- Messages that contain critical business data, such as the receipt of payment for an order, must be reliably maintained and must not be lost in a failure. These messages are good candidates for persistence.
- Some messages might contain query data only, where the loss of the data is not crucial because the query can be repeated. In this case, performance is considered more important than data integrity. Making these types of messages persistent might not benefit the business, and might not be a good use case for persistence.

MQ assures once-only delivery of persistent messages and it assures at-most-once delivery of non-persistent messages. Both persistence options allow developers to write applications with the knowledge that there are no duplicated messages.

Queues

Types of queues

Local	QLOCAL		Owned by the queue manager to which the application is connected. You can get messages from and put messages to local queues.
Remote	QREMOTE		Defined as a local queue on one queue manager, but redirects messages to a local queue on a different queue manager. An application does not need to know whether a queue is local or remote.
Alias	QALIAS		A pointer to a local queue or a locally owned remote queue.
Model	QMODEL		A template to create a dynamic local queue.

A queue is a named object on which applications can put messages, and from which applications can get messages.

Queues are used either to hold messages that are sent by applications or are pointers to other queues or topics.



Only queues that are defined as local queues (QLOCAL) hold messages

An MQ queue is a named object on which applications can put messages, and from which applications can get messages.

IBM MQ supports different types of queues for storing, identifying, and moving messages. This figure summarizes some of the types of queues that MQ uses.

Messages are stored on *local queues*, which are owned by the queue manager to which the application connects.

Remote queues and *alias queues* are pointers to other queues but do not hold messages. Messages can be directed to remote or alias queues, but the final target is a local queue. A remote queue is also known as a local definition of a remote queue. It is not possible to retrieve messages from a remote queue or an alias queue; messages are always in the target local queue that is identified in the definition of the remote or alias queue.

An alias queue is an MQ object that refers indirectly to another queue. The target queue can be a local queue or a local definition of a remote queue.

A local definition of a remote queue, or a remote queue definition, is an MQ object owned by one queue manager that refers to a queue owned by another queue manager.

A model queue is an MQ object whose attributes are used as a template for creating a dynamic local queue. When an application opens a model queue, the queue manager creates a dynamic queue. A dynamic queue that is created from a model queue is one of the following types:

- A temporary dynamic queue, which is deleted when it is closed and does not survive a queue manager restart
- A permanent dynamic queue, whose deletion on the MQCLOSE call is optional and which does survive a queue manager restart

Of the two types of dynamic queues, only a permanent dynamic queue can store persistent messages. A typical use of a dynamic queue is as a reply-to queue for a client program that is sending requests to a server.

Every queue is owned by a queue manager, and that queue manager can own many queues. However, each queue must have a name that is unique within that queue manager.

Authorized applications can retrieve messages from a queue according to the following retrieval algorithms:

- First-in-first-out (FIFO).
- Message priority, as defined in the message descriptor. Messages that have the same priority are retrieved on a FIFO basis.
- A program request for a specific message.

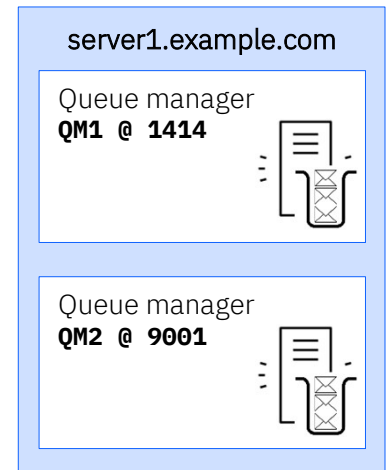
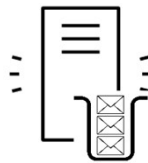
If needed, applications can also select messages from a queue for retrieval that match a selection string.

The graphics on this figure are used throughout this course to identify the different types of queues.

Of these four types of queues, only a local queue can store messages.

Queue managers

- A queue manager is a system program which owns and controls IBM MQ objects and processes
- Queue managers provide queuing services to applications and manages the queues that belong to them
- Multiple queue managers can be hosted on the same machine if they have different names and listener ports
- You need a name and a listener port to create a queue manager
- Administrators interact with queue managers to configure IBM MQ objects



Queue managers are the main components in an MQ messaging network. Queue managers host the other objects in the network, such as the queues and the channels that connect the queue managers together.

Queue managers define the attributes of MQ objects. The values of these attributes affect how MQ processes these objects. You create and manage objects by using MQ commands and interfaces. Each queue manager provides facilities for reliable messaging.

A server can contain more than one queue manager but each queue manager on the server must have a unique name. Visit the following URL for more information about rules for naming IBM MQ objects:

<https://www.ibm.com/docs/en/ibm-mq/9.4?topic=objects-rules-naming-mq>

The queue manager communicates by using a TCP connection. In a distributed environment, each queue manager is assigned a unique TCP/IP port. The default TCP port for a queue manager is 1414. The port number 1414 is assigned to MQ by the Internet Assigned Numbers Authority. When a server contains more than one queue manager, each queue manager must have a different port number.

In this example, the server that is named server1.example.com runs two queue managers:

1. A queue manager named QM1, which is listening on TCP port 1414
2. A queue manager named QM2, which is listening on TCP port 9001

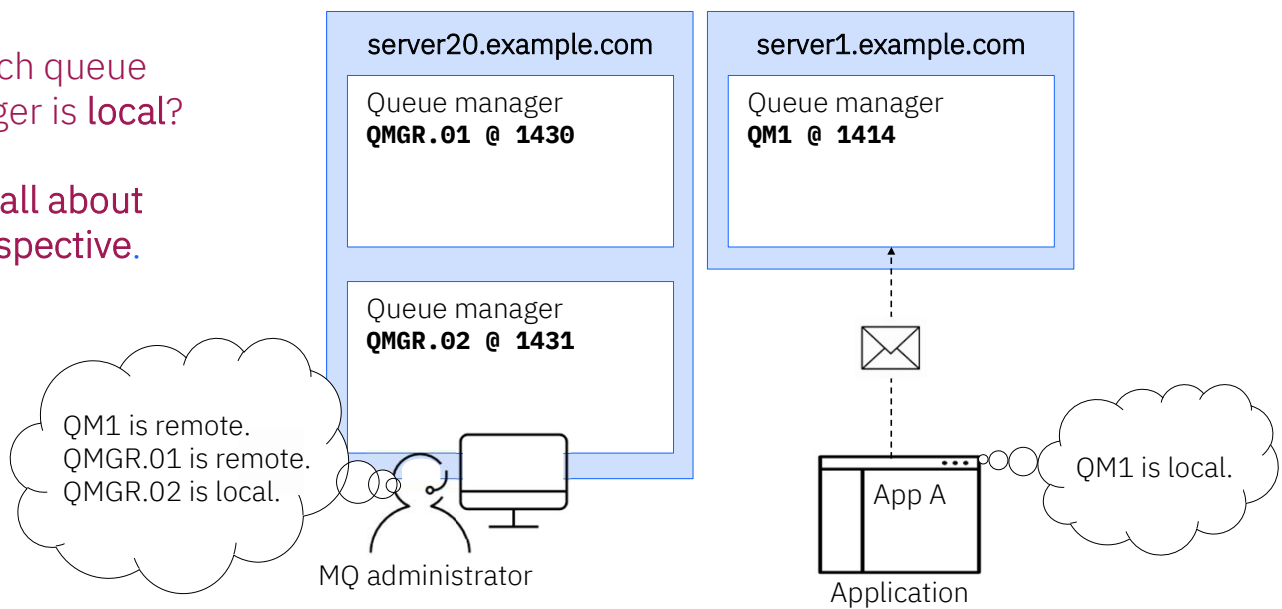
Each queue is owned by a queue manager. The queue manager is responsible for maintaining the queues it owns, and for storing all the messages it receives onto the appropriate queues. The messages might be put on the queue by application programs, or by a queue manager as part of its normal operation.

The graphics on this figure are used throughout this course to identify queue managers.

Local and remote queue managers

Which queue manager is **local**?

It's all about perspective.



To avoid confusion, you must understand what is meant by local and remote when you reference a queue manager in an MQ network.

A local queue manager is the currently referenced queue manager. All other queue managers are remote queue managers.

Whether a queue manager is local or remote depends on whether the administrator or application is directly connected to a queue manager. A local queue manager is the currently referenced queue manager; a remote queue manager is any queue manager that is not currently referenced.

The local queue manager designation changes when work moves to another queue manager. For example, if an administrator is connected to queue manager QMGR.02, then that queue manager is the local queue manager and the other queue managers (QMGR.01 and QM1) are remote queue managers. If an application is connected to the queue manager that is named QM1, the other queue managers (QMGR.01 and QMGR.02) are the remote queue managers.

Special-purpose local queues

Examples of special-purpose queues

Transmission queue (XMITQ) If messages are destined for a remote queue, the local queue manager holds them on a transmission queue until they can be forwarded to the remote queue manager.	Dead-letter queue A designated queue where a queue manager puts messages that cannot be delivered.
Initiation queues IBM MQ can trigger an application to start automatically when messages are available to retrieve. If triggering is enabled for a queue and a trigger event occurs, the queue manager sends a trigger message to an initiation queue.	Reply-to queues A receiving application can send a reply back to the sender. These replies are messages, which are put on a reply-to queue.

Some local queues have special purposes in MQ. You learn more about these special-purpose queues throughout this course.

Transmission queue:

Transmission queues (XMITQ) are local queues that temporarily store messages that are destined for remote queue managers. The name of a transmission queue is part of a remote queue definition. You must define at least one transmission queue for each remote queue manager to which the local queue manager is to send messages directly. Each queue manager can have a default transmission queue.

Think of a transmission queue as a staging point for messages that are destined for a remote queue manager. If the destination is unavailable, messages build up on the transmission queue until the connection can be successfully completed. Transmission queues are transparent to the application. When an application opens a remote queue, the queue manager internally creates a reference to the relevant transmission queue and messages are put there.

Dead-letter queue:

A *dead-letter queue* is a designated queue where a queue manager puts messages that cannot otherwise be delivered. Sometimes messages cannot be delivered, for reasons including:

- The destination queue is full.
- The destination queue does not exist.
- Message puts have been inhibited on the destination queue.
- The sender is not authorized to use the destination queue.
- The message is too large.

It is not mandatory for a queue manager to have a dead-letter queue, but it is a good practice and critical to the health of the queue manager. A queue manager has at most one DLQ.

The SYSTEM queue SYSTEM.DEAD.LETTER.QUEUE is not automatically assigned as the queue manager's dead-

letter queue but can be assigned as the dead-letter when the queue manager is created. The SYSTEM queue SYSTEM.DEAD.LETTER.QUEUE queue is often designated to be the dead-letter queue; however, any other local queue can serve this purpose if it is identified as the dead-letter queue to the queue manager.

Initiation queue:

An *initiation queue* is a local queue that is used to implement triggering. A queue manager puts a trigger message on an initiation queue when a trigger event occurs. A trigger event is a logical combination of conditions that is detected by a queue manager. For example, a trigger event might be generated when the number of messages on a queue reaches a predefined depth. This event causes the queue manager to put a trigger message on a specified initiation queue. Such a trigger message is transparent to the programmer, but it is a powerful mechanism to cause applications or channels to start executing only when there is work to be processed. If a queue manager is to use triggering, at least one initiation queue must be defined for that queue manager.

Reply-to queue:

When an application sends a request message, the application that receives the message can send back a reply message to the sending application. This message is put on a queue, called a *reply-to queue*, which is normally a local queue to the sending application. The name of the reply-to queue is specified by the sending application as part of the message descriptor.

Other special-purpose queues:

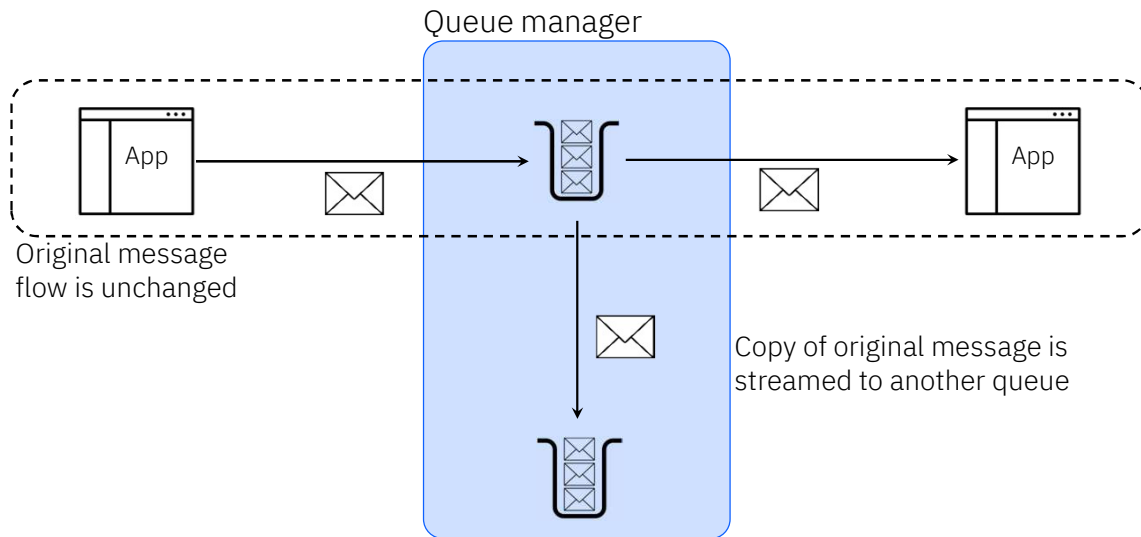
Queue managers also use several type of special-purpose queues, for example:

- The *command queue*, SYSTEM.ADMIN.COMMAND.QUEUE, is a local queue to which suitably authorized applications can send MQSC commands for processing. The MQ command server retrieves these commands. The command server validates the commands, passes the valid ones on for processing by the queue manager, and returns any responses to the appropriate reply-to queue.
- When a queue manager detects an instrumentation event, which can be either a channel, queue manager, performance, or logger event, the queue manager puts a message that describes that event on an *event queue*. A system management application can monitor an event queue, get the messages put on these queues, and then take appropriate action. A queue manager generates event messages when certain things happen. For example, events can occur when a queue is nearly full or when an application is not authorized to open a queue. These event messages are written to one of the predefined event queues and can then be processed by management tools. All event queues have similar names that indicate the type of event that is held there. SYSTEM.ADMIN.QMGR.EVENT and SYSTEM.ADMIN.CHANNEL.EVENT are examples of queue manager event queues.
- The purpose of the *default queues* is to identify the default values of the attributes of any new queue that you create. One default queue exists for each of the four types of queues: local, alias, remote, and model. You need to include in the definition of a queue only those attributes whose values are different from the default values when you create a queue. You can change the default value of an attribute by redefining the appropriate default queue.
- *System queues*: When a queue manager is created, several queues are created for the use of the queue manager. These queues start with the SYSTEM prefix and should not be changed, deleted, or

used to send messages. For example, the SYSTEM.ADMIN.STATISTICS.QUEUE holds queue manager statistics data.

Streaming queues

Put a near-identical copy of every message to a second queue



The streaming queues feature of IBM MQ allows you to configure a queue to put a near-identical copy of every message to a second queue. You configure streaming on individual queues, and the messages are streamed by the queue manager, not by the application itself.

This means that in almost all cases the application putting messages to the original queue is completely unaware that streaming is taking place. Similarly, the application consuming messages from the original queue is unaware that message streaming has taken place.

Streaming queues can be useful in cases where you need to create a copy of your messages. For example:

- Streaming messages to Apache Kafka using the Kafka Connect source connector for IBM MQ.
- Performing analysis on the data going through the system.
- Storing messages for recovery at a later time.
- Capturing a set of messages to use in development and test systems.
- Consuming IBM MQ event messages from the system event queues, and sending additional copies to other queues or topics.

You can configure streaming queues to ensure that the original messages remain unaffected by the streaming process. This ensures that core business applications do not observe any impact from the streaming.

You can configure streaming queues in one of two modes:

- **Best effort**
 - In this mode, the queue manager considers it more important that delivery of the original message is not affected by delivery of the streamed message.
 - If the original message can be delivered, but the streamed message cannot, the original message is still delivered to its queue. This mode is best suited to those applications, where it is important for the original business application to remain unaffected by the streaming process.
- **Must duplicate**

- In this mode, the queue manager ensures that both the original message and the streamed message are successfully delivered to their queues.
- If, for some reason, the streamed message cannot be delivered to its queue, for example, because the second queue is full, then the original message is not delivered to its queue either. The putting application receives an error reason code and must try to put the message again.

In most cases, the copy of the message delivered to the second queue is a duplicate of the original message. This includes all of the message descriptor fields, including the message ID and correlation ID. The streamed messages are intended to be very close copies of the original messages, so that they are easier to find and, if necessary, replay them back into another IBM MQ system.

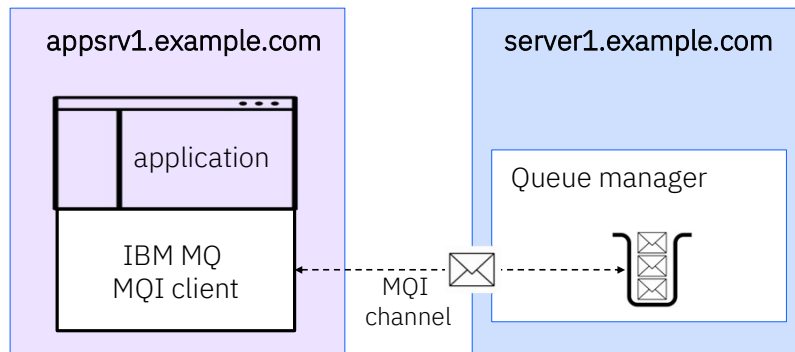
There are some message descriptor fields that are not retained on the streamed message. The following changes are made to the streamed message before it is placed on the second queue:

- The expiry of the streamed message is set to MQEI_UNLIMITED, regardless of the expiry of the original message. If CAPEXPY has been set on the secondary queue its value is used to set the expiry time of the streamed message.
- If any of the following report options are set on the original message, they are not enabled on the streamed message. This is to ensure that no unexpected report messages are delivered to applications that are not designed to receive them:
 - Activity reports
 - Expiration reports
 - Exception reports
 - Confirmation on arrival (COA)
 - Confirmation on delivery (COD)

It is possible to stream messages from a local queue to a cluster queue and to stream messages from cluster queue instances to a local queue.

IBM MQ MQI Clients

An IBM MQ MQI (Message Queue Interface) client allows an application that runs on one system to connect to a queue manager that is running on another system



An IBM MQ MQI client is a separate component of IBM MQ that can be installed on a system on which no queue manager runs. IBM MQ MQI clients are an efficient way of implementing IBM MQ messaging and queuing without the need for the queue manager runtime code to reside on the client system. IBM MQ MQI clients are sometimes referred to as MQ clients, or simply clients.

Using an IBM MQ client, an application running on the same system as the client can connect to a queue manager that is running on another system. The application can issue MQI calls to that queue manager. Such an application is called an IBM MQ client application and the queue manager is called a server queue manager.

An application that you want to run in the IBM MQ MQI client environment must first be linked with the relevant client library. When the application issues an MQI call, the IBM MQ MQI client directs the request to a queue manager, where it is processed and from where a reply is sent back to the IBM MQ MQI client.

An IBM MQ client application and a server queue manager communicate with each other by using a bidirectional MQI channel. The input parameters of an MQI call flow in one direction on an MQI channel, and the output parameters flow in the opposite direction.

The following benefits are available by using an MQ client:

- There is no need for a licensed MQ server installation on the client machine.
- Hardware requirements on the client system are reduced.
- System administration requirements on the client system are reduced.
- An application that uses an MQ client can connect to multiple queue managers on different machines.
- A client application can be connected to more than one queue manager server simultaneously.

Applications can connect to queue managers either on the same system (BINDINGS mode) or remotely over a network (CLIENT mode).

- In BINDINGS mode, clients use shared memory, semaphores, and other local facilities to connect to a queue manager

installed on the same system. BINDINGS mode provides better performance for IBM MQ applications than using network connections.

- In CLIENT mode, clients use a TCP/IP network connection to communicate with a queue manager on a different system. You use the CLIENT transport mode when the application and queue manager are located on different servers. Optionally, CLIENT mode can also be used when the client and the queue manager are installed on the same machine with a loopback connection.

Channels

Messages move through three types of channels:

- **Message channels** move messages between queue managers
- **Message Queue Interface (MQI) channels** move messages from applications to a queue manager
- **AMQP (Advanced Message Queuing Protocol) channels** connect an AMQP messaging application with a queue manager

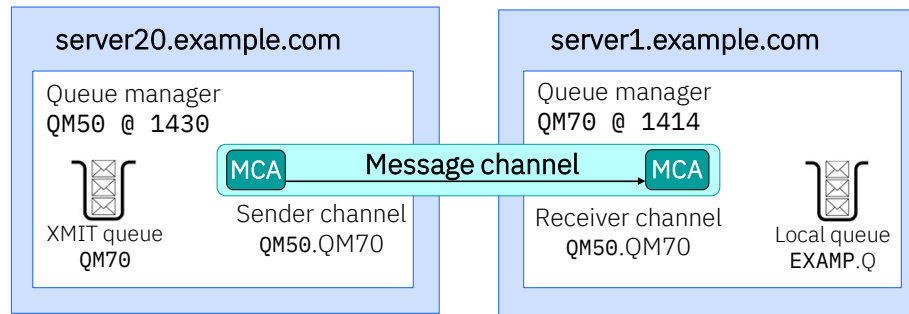
MQ uses channels to move messages. Two commonly used types of channels are: *message channels* and *Message Queue Interface (MQI) channels*. These types of channels are described on the next slides.

You can also use AMQP (Advanced Message Queuing Protocol) channels. AMQP channels connect an AMQP messaging application with a queue manager, enabling the application to exchange messages with IBM MQ applications. An AMQP channel allows you to develop an application using MQ Light, and then deploy it as an enterprise application, taking advantage of the enterprise-level facilities provided by IBM MQ.

Message channels

Message channels move messages between queue managers

- Unidirectional
- Subtypes include: sender, receiver, server, requester, cluster-sender, cluster-receiver



A message channel is a connection between two queue managers that is used to transport messages. The queue managers might exist on the same system, or a different systems on the same platform, or on different platforms.

A message channel is a one-way link. Messages are only sent in one direction along the channel. When you define message channels, you must define one channel object at the queue manager that is to send messages. You must also define another, complementary channel, at the queue manager that is to receive the messages. Sender-receiver channels are the most commonly used message channels.

A message channel connects two queue managers by using two message channel agents (MCAs). An MCA is a program that transfers messages from a transmission queue to a communication link, and from a communication link into the target queue. The MCA on one end takes messages from local transmission queue and puts them on a communication link. The MCA on other end receives the messages and puts them on target queue. This pair of MCAs, one sending and one receiving, make up a channel and move messages from one queue manager to another. MQ was designed to support several network protocols for the MCAs, and these other protocols can still be used, but it is now rare to find anything other than TCP/IP.

Each end of a message channel has a separate definition. Both definitions contain the name of the message channel. Among other things, the definition at each end of a message channel also indicates:

- Whether it is the sending end or the receiving end of the channel
- The communications protocol to use

A transmission queue is required at the sending end of a message channel. So, only the definition of the message channel at the sending end contains the name of the transmission queue.

This figure shows an example of how the message channels are paired to provide a communications link between two queue managers.

In the example, the queue manager that is named QM50 is configured with a sender channel that is named QM50.QM70 that can send messages to the queue manager that is named QM70. The queue manager that is named QM70 is configured with a receiver channel with the same name that can receive messages from the queue manager that is named QM50.

During processing, the sender channel takes messages from transmission queue named QM70 and sends them to the remote queue manager of the same name. It is a good practice to name the channels to be the same as the 'from' and 'to' queue managers, in the correct from-to order. If another channel pair had to be defined having the sender in QM70 and receiver in QM50, then each sender and receiver would be named QM70.QM50.

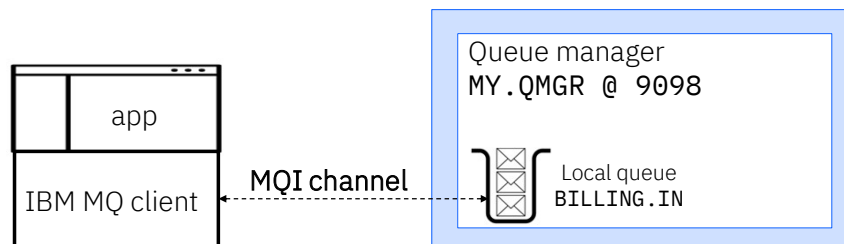
In this example, the name of the transmission queue (XMIT) on the sender is QM70. Another good practice is to create a transmission queue with the same name as the name of the destination queue manager. In the case a remote queue object on the sending queue manager is created with no transmission queue specified, a queue with the same name as the remote queue manager is used.

This simple example shows only a single message channel between two queue managers. In a real-world MQ environment, many queue managers within the infrastructure are connected by many logical channels over a communications network. Messages automatically flow across these channels from the initial producer of a message to the eventual consumer of that message based on the configuration of the queue managers in the infrastructure. Changes can be made to the configuration of queues and channels transparently for the applications. For example, a receiving application can be moved to a new machine, and a route to that machine can be defined without needing any changes to sending applications.

Message Queue Interface (MQI) channels

Message Queue Interface (MQI) channels move messages from applications to a queue manager

- Bidirectional
- Subtypes include: SVRCONN and CLNTCONN



An MQI channel provides a communications link between a client application and a queue manager. It is a two-way link and is used for the transfer of MQI calls and responses only, including MQPUT calls that contain message data and MQGET calls that result in the return of message data. MQI channels are sometimes also referred to as client-connection channels.

Applications issue MQI calls using the MQ client. An MQI channel starts when the client application sends an MQCONN or MQCONNX call to connect to the server queue manager. It ends when the client application sends an MQDISC call to disconnect from the server queue manager.

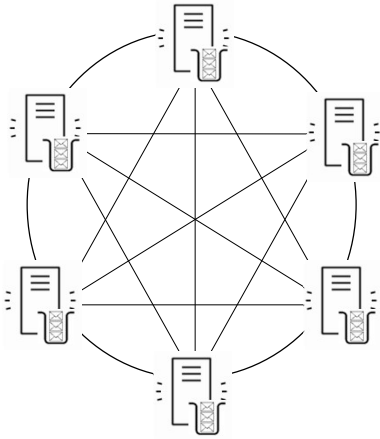
There are two types of MQI channel definitions:

1. Client-connection channel (CLNTCONN): This type is for the IBM MQ MQI client.
2. Server-connection channel (SVRCONN): This type is for the server running the queue manager, with which the MQ client is to communicate.

Point-to-point and publish/subscribe

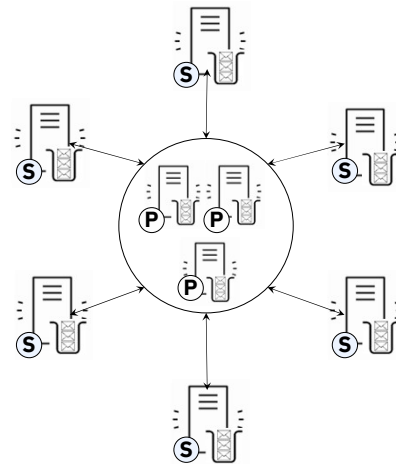
Point-to-point

One message put on the queue, one message received



Publish/subscribe

One message published, multiple messages received



Ⓢ = Subscribers Ⓟ = Publishers

MQ supports two message-queuing styles: point-to-point and publish/subscribe.

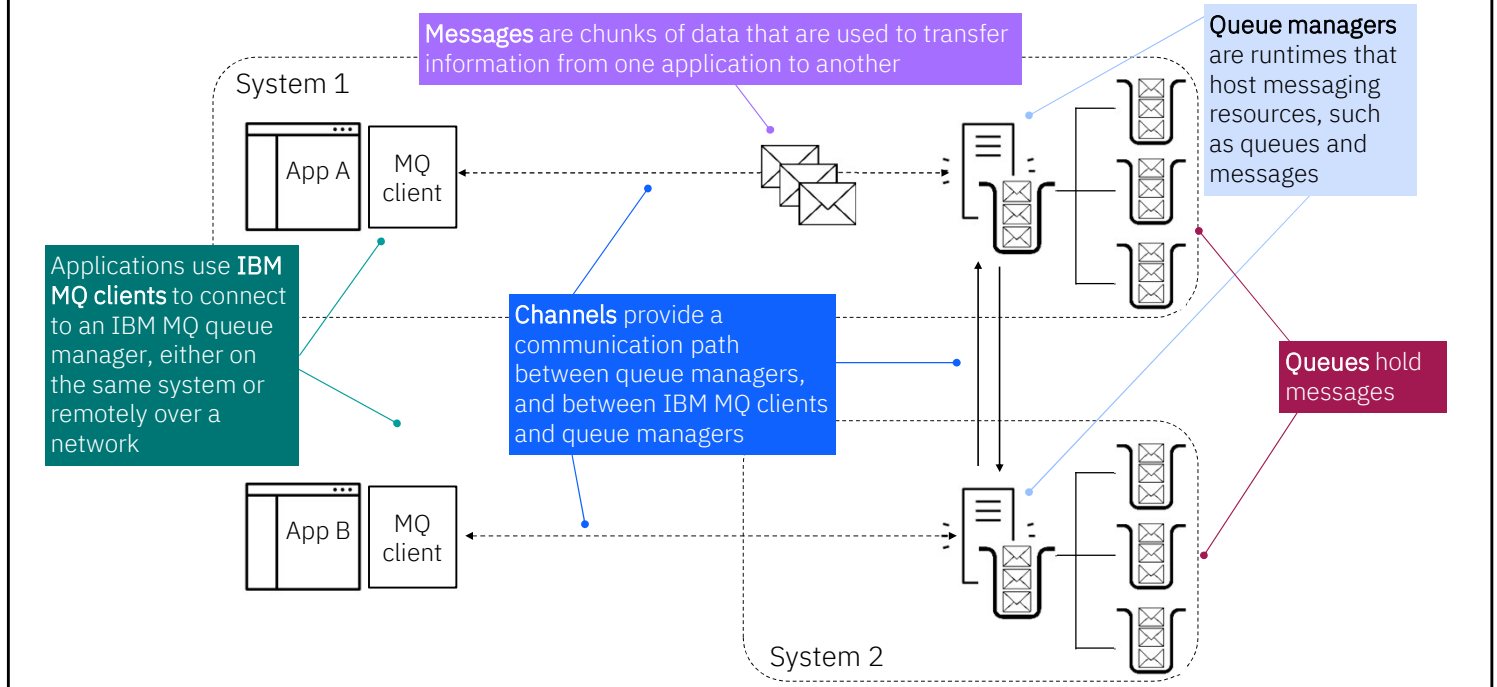
In point-to-point messaging, a sending application must know information about the receiving application before it can send a message to that application. For example, the sending application might need to know the name of the queue to which to send the information, and might also specify a queue manager name.

The point-to-point messaging style requires connections across all queue managers that need to get or put the message. You can manually define all the connections in a distributed queuing environment, or you can create a cluster and let IBM MQ define much of the connection detail for you.

In publish/subscribe messaging, a copy of each message that is published by an application is delivered to every interested application. There might be many, one, or no interested applications. In publish/subscribe, an interested application is known as a subscriber and the messages are queued on a queue that is identified by a subscription. A subscription is a way of matching messages to interested consumers. Instead of working directly with a queue, a message producer publishes messages to a topic. IBM MQ works out which subscriptions are interested in that topic, and delivers copies of the message to each subscription queue. Publish/subscribe needs a connection between the publisher and subscriber. After the publish/subscribe infrastructure is defined, no additional connectivity needs to be defined unless new servers or queue managers are added to the infrastructure.

With publish/subscribe messaging, you can decouple the provider of information from the consumers of that information. The receiver does not need to know where messages originated, only that they subscribed to the correct topic. The sender of a publication does not need to know where the messages are going. In this course, you learn about the point-to-point messaging style.

Example IBM MQ system



The preceding slides introduced you to some key terms that are commonly used to describe IBM MQ objects. The purpose of this figure is to reinforce the key terms and concepts you have learned in this unit.

Messages contain application data.

Queues hold messages.

Channels connect queue managers and client applications. Queue manager manages messages, queues, channels, and other IBM MQ services and resources

What is a message queue?

A message queue is a named destination to which messages can be sent. Messages accumulate on queues until they are retrieved by applications that service those queues.

What is a queue?

A queue is a data structure that is used to store messages.

Each queue is owned by a queue manager. The queue manager is responsible for maintaining the queues that it owns, and for storing all the messages it receives onto the appropriate queues. The messages might be put on the queue by application programs, or by a queue manager as part of its normal operation.

What is a message?

A message is a string of bytes that is meaningful to the applications that use it. Messages are used to transfer information from one application program to another (or between different parts of the same application). The applications can be running on the same platform, or on different platforms.

What is a channel?

Channels are logical communication links that provide a paths throughout an MQ network. Different types of channels are

available for different types of connections, including:

- Between distributed queue managers (message channels)
- Between an IBM MQ MQI client and an IBM MQ server (MQI channels)
- Between two IBM MQ servers

What does an IBM MQ administrator do?

- Install, apply maintenance, and configure IBM MQ
- Create queue managers
- Create queues, clusters, channels, or any objects that users request
- Administer publish/subscribe topologies
- Administer security and TLS keystores and truststores
- Back up file systems and object definitions
- Monitor disk space
- Identify problems
- Participate in infrastructure and capacity planning sessions
- Establish naming standards
- Analyze performance
- Respond to requests from application developers

As an administrator, you are responsible for the installation, configuration, and maintenance of IBM MQ. You create and monitor the welfare of the queue managers, and other IBM MQ objects. Some of your work will be addressing requests from developers.

Some of the main tasks and job duties of an MQ administrator are listed on this slide.

Unit summary

- Simplify connections between applications with enterprise messaging
- Understand the deployment options for IBM MQ
- Describe key concepts and terminology related to enterprise messaging and IBM MQ

This slide lists the learning objectives for this unit. Now that you have finished this unit, you should be comfortable with these concepts.

Review questions



1. True or False: You can use IBM MQ is on IBM Cloud as a managed service.
2. What is a message?
 - a) A chunk of data sent by one application and intended for another application
 - b) An operating system setting that enables or disables IP communication
 - c) An operating system setting that enables or disables virtualization
 - d) A communication path between queue managers
 - e) A communication path between a queue manager and a client
3. What are two types of IBM MQ channels?
 - a) Message channels and MQI channels
 - b) Block storage channels and object storage channels
 - c) Printer queues and backlog channels
 - d) Internal channels and external channels
 - e) UHF channels and MQI channels

Review answers



1. [True](#) or False: You can use IBM MQ is on IBM Cloud as a managed service.

The answer is [True](#).

2. What is a message?

- a) [A chunk of data sent by one application and intended for another application](#)
- b) An operating system setting that enables or disables IP communication
- c) An operating system setting that enables or disables virtualization
- d) A communication path between queue managers
- e) A communication path between a queue manager and a client

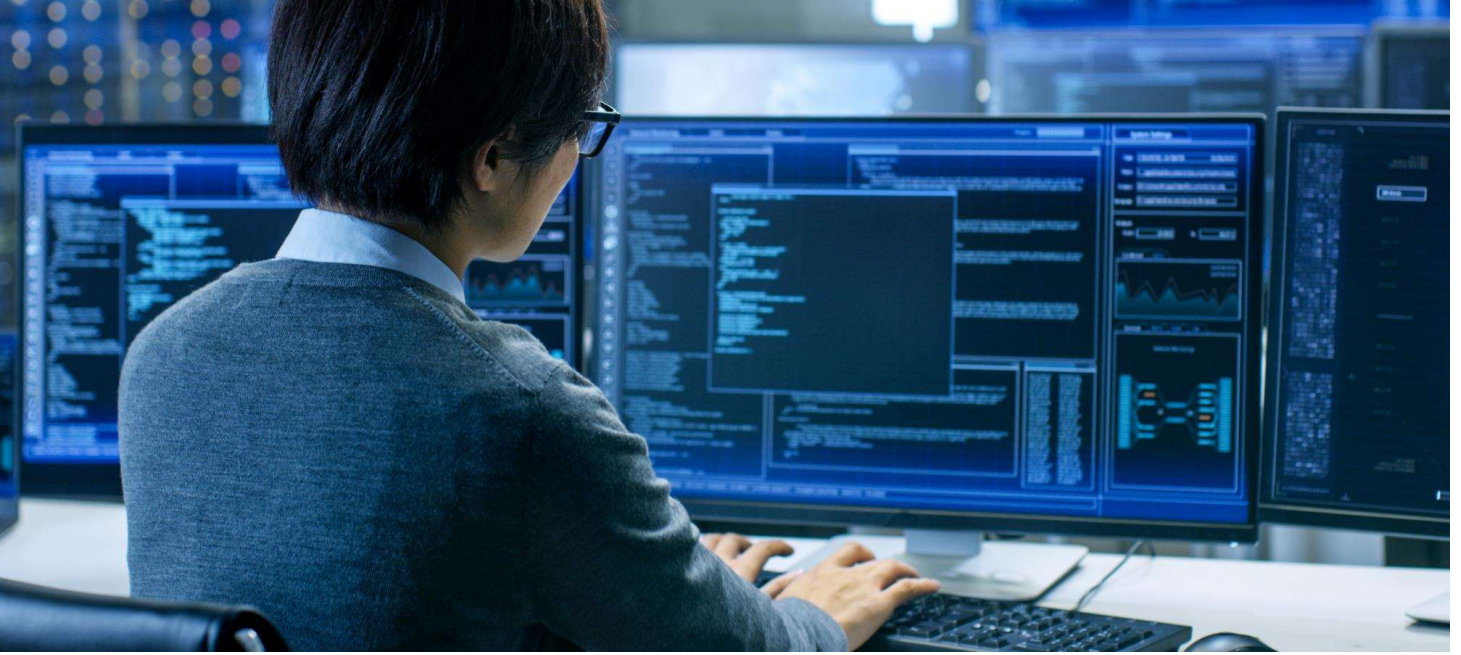
The answer is [A](#).

3. What are two types of IBM MQ channels?

- a) [Message channels and MQI channels](#)
- b) Block storage channels and object storage channels
- c) Printer queues and backlog channels
- d) Internal channels and external channels
- e) UHF channels and MQI channels

The answer is [A](#).

Exercise: Verifying your lab environment



This unit includes a lab exercise: Verifying your lab environment.

Exercise objectives

- Access the desktop of your lab server
- Confirm that IBM MQ is installed

To complete the lab exercises in this course, you use a remote virtual machine (VM). IBM MQ is already installed on your VM. In this initial exercise, you learn how to access your lab VM. You then verify that IBM MQ is ready to use.

About the IBM MQ sample programs	amqsput Read text from standard input and put messages on a queue	amqsget Get messages from a queue and write to standard output	amqsbcg Browse messages on a queue and show both application data and message descriptor
	amqsgbr Browse messages on a queue and show application data only	amqsreq Read text from standard input, put messages on a queue, and write messages from the reply-to queues to standard output	Plus, other sample programs to test other use cases

During the exercises for this unit, you test your IBM MQ lab environment. A useful set of test tools are the IBM MQ sample programs. In most deployments, the MQ sample programs are installed when MQ is installed. You can use this set of programs to simulate applications that send and receive data with IBM MQ. Each program performs a specific task, to test different scenarios. You will use these sample programs throughout this course. This slide is a partial list of the sample programs.

The names of the samples start with the prefix **amq**. The fourth character indicates the programming language, and the compiler where necessary:

- s**: C language
- 0**: COBOL language on both IBM and Micro Focus compilers
- i**: COBOL language on IBM compilers only
- m**: COBOL language on Micro Focus compilers only

The eighth character of the executable indicates whether the sample runs in local binding mode or client mode. If there is no eighth character, then the sample runs in local bindings mode. If the eighth character is 'c' then the sample runs in client mode.

In the real world, these sample programs are often used for tasks like the following examples:

- Test new MQ deployments
- Troubleshoot existing MQ environments
- Test MQ-connected applications
- Learn how to use MQI calls in your own applications by reading the source code

