9.4

IBM MQ

Nota

Antes de utilizar esta información y el producto al que da soporte, lea la información en "Avisos" en la página 307.

Esta edición se aplica a la versión 9 release 4 de IBM® MQ y a todos los releases y modificaciones posteriores hasta que se indique lo contrario en nuevas ediciones.

Contenido

Visión general técnica de IBM MQ

Utilice IBM MQ para conectar las aplicaciones y gestionar la distribución de información en toda la organización.

IBM MQ permite que los programas se comuniquen entre sí a través de una red de componentes diferentes (procesadores, sistemas operativos, subsistemas y protocolos de comunicación) utilizando una interfaz de programación de aplicaciones coherente. Las aplicaciones diseñadas y escritas mediante esta interfaz se conocen como aplicaciones de colocación de mensajes en colas.

Utilice los subtemas siguientes para obtener información sobre la gestión de colas de mensajes y otras características suministradas por IBM MQ.

Conceptos relacionados

Introducción a IBM MQ

Dónde encontrar información acerca de los requisitos del producto y el soporte

Tareas relacionadas

Planificación de una arquitectura de IBM MQ

Referencia relacionada

“Características principales y ventajas de la colocación de mensajes en colas” en la página 7
Esta información destaca algunas de las características y las ventajas de la colocación de mensajes en colas. Describe características como la seguridad y la integridad de los datos en la colocación de mensajes en colas.

Introducción a la colocación de mensajes en colas

Los productos de IBM MQ permiten que los programas se comuniquen entre sí a través de una red de componentes diferentes (procesadores, sistemas operativos, subsistemas y protocolos de comunicación) utilizando una interfaz de programación de aplicaciones coherente.

Las aplicaciones diseñadas y escritas con esta interfaz se conocen como aplicaciones de *colocación de mensajes en colas* porque utilizan el tipo de *mensajería* y el tipo *colocación en colas*:

- Mensajería significa que los programas se comunican entre sí enviando datos en mensajes, en lugar de invocándose entre sí.
- Encolamiento significa poner mensajes en colas a modo de almacenamiento, lo que permite a los programas ejecutar independientemente unos de otros,a velocidades y tiempos diferentes, en ubicaciones distintas y sin existir una conexión lógica entre ellos.

Durante muchos años se ha utilizado la colocación de mensajes en colas en el proceso de datos. Actualmente, es lo que se utiliza con mayor frecuencia en el correo electrónico. Sin la colocación en colas, enviar un mensaje electrónico a través de largas distancias requiere que cada nodo en la ruta esté disponible para reenviar mensajes, y que los destinatarios estén conectados y sean conscientes del hecho de que se les está intentando enviar un mensaje. En un sistema de colocación en colas, los mensajes se almacenan en nodos intermedios hasta que el sistema esté listo para reenviarlos. En el destino final se almacenan en un buzón hasta que el destinatario esté listo para leerlos.

Aun así, muchas transacciones empresariales complejas se procesan hoy en día sin la colocación en colas. En una red grande, el sistema podría mantener miles de conexiones en un estado preparado para utilizarse. Si una parte del sistema sufre un problema, muchas partes del sistema podrían quedar inservibles.

La colocación de mensajes en colas se puede considerar como un correo electrónico para los programas. En un entorno de colas de mensaje, cada programa que conforma una parte de una serie de aplicaciones realiza una función bien definida e independiente en respuesta a una solicitud específica. Para comunicarse con otro programa, un programa debe transferir un mensaje a una cola predefinida. El otro programa recupera el mensaje de la cola y procesa las solicitudes así como la información contenida

en el mensaje. Por lo tanto, la colocación de mensajes en colas es un tipo de comunicación programa a programa.

La colocación en colas es un mecanismo mediante el cual los mensajes se guardan hasta que la aplicación esté preparada para utilizarlos. La colocación en colas permite realizar lo siguiente:

- Comunicarse entre programas (que pueden estar ejecutándose en entornos diferentes) sin tener que escribir el código de comunicación.
- Seleccionar el orden en el que un programa procesa los mensajes.
- Equilibrar las cargas en un sistema organizando que más de un programa atienda una cola cuando el número de mensajes excede un umbral.
- Aumentar la disponibilidad de las aplicaciones organizando que un sistema alternativo atienda las colas si el sistema primario no está disponible.

¿Qué es una cola de mensajes?

Una cola de mensajes, conocida simplemente como cola es un destino específico al que se pueden enviar mensajes. Los mensajes se acumulan en las colas hasta que los recuperan programas que atienden esas colas.

Las colas residen en un gestor de colas y son gestionada por él; (consulte "Terminología sobre la colocación de mensajes en colas" en la página 9). La naturaleza física de una cola depende del sistema operativo en el que se ejecuta el gestor de colas. Una cola puede ser un área de almacenamiento intermedio volátil en la memoria de un sistema o un conjunto de datos en un dispositivo de almacenamiento permanente (por ejemplo un disco). La gestión física de las colas es responsabilidad del gestor de colas y no es visible para los programas de aplicación que participan.

Los programas sólo acceden a las colas a través de los servicios externos del gestor de colas. Pueden abrir una cola, transferirle mensajes, extraer mensajes de ella y cerrarla. También pueden establecer y realizar consultas sobre los atributos de las colas.

Diferentes estilos de colas de mensaje

Punto a punto

Un mensaje se coloca en la cola y una aplicación recibe ese mensaje.

En la mensajería punto a punto, una aplicación emisora debe tener información sobre la aplicación receptora antes de que pueda enviar un mensaje a esa aplicación. Por ejemplo, la aplicación emisora puede necesitar saber el nombre de la cola a la que enviar la información, y también puede especificar un nombre de gestor de colas.

Publicación/suscripción

Una copia de cada mensaje publicado por una aplicación de publicación se entrega a cada aplicación interesada. Puede que haya muchas, una o ninguna aplicación interesada. En la publicación/suscripción, una aplicación interesada se conoce como un suscriptor y los mensajes se ponen en cola en una cola identificada mediante una suscripción.

La mensajería de publicación/suscripción permite separar el proveedor de información de los consumidores de dicha información. Para poder enviar y recibir información, no es necesario que la aplicación de envío y la de recepción sepan mucho la una de la otra. Para obtener más información, consulte "Mensajería de publicación/suscripción" en la página 63.

Ventajas de la colocación de mensajes en colas para el diseñador y el desarrollador de aplicaciones

IBM MQ permite que los programas de aplicación utilicen la *colocación de mensajes en colas* para participar en el proceso controlado por mensajes. Los programas de aplicación pueden comunicarse entre distintas plataformas utilizando los productos de software colocación de mensajes en colas adecuados. Por ejemplo, las aplicaciones de z/OS pueden comunicarse a través de IBM MQ for z/OS.

Las aplicaciones son independientes del funcionamiento de las comunicaciones subyacentes. Algunas de la ventajas de la colocación de mensajes en colas son las siguientes:

- Puede diseñar aplicaciones utilizando pequeños programas que se pueden compartir entre muchas aplicaciones.
- Puede crear rápidamente nuevas aplicaciones reutilizando estos bloques de creación.
- Las aplicaciones escritas para utilizar técnicas de colocación de mensajes en colas no se ven afectadas por los cambios en el modo en que los gestores de colas funcionan.
- No es necesario utilizar ningún protocolo de comunicación. El gestor de colas se encarga automáticamente de todos los aspectos de la comunicación.
- Los programas que reciben mensajes no tienen que estar en ejecución en el momento en que se les envían mensajes. Los mensajes se retienen en las colas.

Los diseñadores pueden reducir el coste de sus aplicaciones porque el desarrollo es más rápido, se necesitan menos desarrolladores y los requisitos de aptitudes de programación son menores que los de las aplicaciones que no utilizan colas de mensajes.

IBM MQ implementa una interfaz de programación de aplicaciones común conocida como *interfaz de cola de mensajes* (o MQI) dondequiera que se ejecuten las aplicaciones. Esto facilita el traslado de programas de aplicación de una plataforma a otra.

Para obtener detalles sobre MQI, consulte Visión general de la interfaz de cola de mensajes.

Características principales y ventajas de la colocación de mensajes en colas

Esta información destaca algunas de las características y las ventajas de la colocación de mensajes en colas. Describe características como la seguridad y la integridad de los datos en la colocación de mensajes en colas.

Las características principales de las aplicaciones que utilizan las técnicas de colocación de mensajes en colas son:

- “No hay conexiones directas entre programas” en la página 7
- “Comunicación independiente del tiempo” en la página 8
- “Programas pequeños” en la página 8
- “Procesamiento dirigido por mensajes” en la página 8
- “Procesamiento controlado por sucesos” en la página 9
- “Prioridad de mensaje” en la página 9
- “Seguridad” en la página 9
- “Integridad de datos” en la página 9
- “Soporte de recuperación” en la página 9

Nota: A la hora de considerar clientes y servidores de IBM MQ, no tiene que cambiar una aplicación de servidor para dar soporte a IBM MQ MQI clients adicionales en plataformas nuevas. Asimismo, IBM MQ MQI client puede, sin cambios, funcionar con tipos adicionales de servidores.

No hay conexiones directas entre programas

La colocación de mensajes en colas es una técnica para la comunicación indirecta de programa a programa. Se puede utilizar dentro de cualquier aplicación en la que los programas se comuniquen entre sí. La comunicación se produce cuando un programa transfiere mensajes a una cola (propiedad de un gestor de colas) y otro programa obtiene los mensajes de la cola.

Los programas pueden obtener mensajes que fueron colocados en una cola por otros programas. Los otros programas pueden estar conectados al mismo gestor de colas como programa receptor, o bien a otro gestor de colas. Este otro gestor de colas puede estar en otro sistema, en un sistema informático diferente, o incluso en una empresa distinta.

No hay conexiones físicas entre los programas que se comunican utilizando colas de mensajes. Un programa envía mensajes a una cola propiedad de un gestor de colas y otro programa recupera mensajes de la cola (consulte Figura 1 en la página 8).

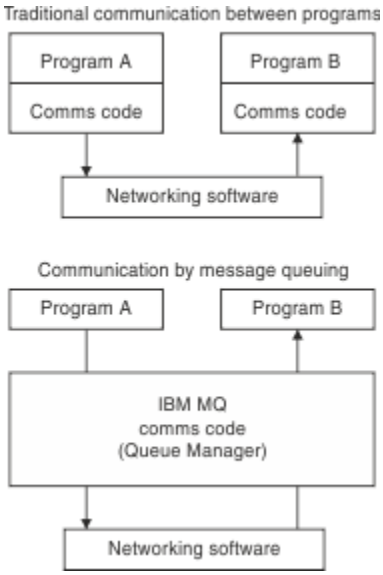


Figura 1. Colocación de mensajes en colas en comparación con la comunicación tradicional

Al igual que ocurre con el correo electrónico, los mensajes individuales que forman parte de una transacción viajan a través de una red de acuerdo con un sistema de almacén y reenvío. Si un enlace entre los nodos falla, el mensaje se conserva hasta que el enlace se ha restaurado o el operador o el programa redirige el mensaje.

El mecanismo por el que un mensaje se mueve de cola en cola queda oculto en los programas. Por lo tanto, los programas son más simples.

Comunicación independiente del tiempo

Los programas que solicitan que otros realicen trabajos no tienen que esperar la respuesta a su solicitud. Pueden realizar otros trabajos y procesar la respuesta ya sea cuando llegue o en un momento posterior. Cuando se escribe una aplicación de mensajería, no es necesario saber (ni preguntarse) cuándo un programa envía un mensaje o cuando el destino puede recibir el mensaje. El mensaje no se pierde; el gestor de colas lo conserva hasta que el destinatario esté listo para procesarlo. El mensaje permanece en la cola hasta que un programa lo elimina. Esto significa que los programas de aplicación emisor y receptor están separados: el emisor pueda seguir procesando sin tener que esperar a que el receptor acuse recibo del mensaje. De hecho, no es necesario que la aplicación de destino esté ejecutándose cuando se envía el mensaje. Puede recuperar el mensaje una vez que se ha iniciado.

Programas pequeños

La colocación de mensajes en colas permite utilizar la ventajas de los programas pequeños e independientes. En lugar de un único gran programa que realiza secuencialmente todas las partes de un trabajo, puede distribuir el trabajo en varios programas más pequeños e independientes. El programa solicitante envía mensajes a cada uno de los programas separados, solicitándoles que realicen su función; cuando cada programa está completo, los resultados se envían de nuevo en forma de uno o más mensajes.

Procesamiento dirigido por mensajes

Al llegar a una cola, los mensajes pueden iniciar automáticamente una aplicación utilizando un mecanismo llamado *desencadenamiento*. Si es necesario, las aplicaciones se pueden detener después de haber procesado el mensaje o mensajes.

Procesamiento controlado por sucesos

Los programas se pueden controlar de acuerdo con el estado de las colas. Por ejemplo, puede disponer que un programa se inicie tan pronto como un mensaje llegue a una cola, o puede especificar que el programa no se inicie hasta que haya, por ejemplo, 10 los mensajes por encima de una determinada prioridad en la cola, o 10 mensajes de cualquier prioridad en la cola.

Prioridad de mensaje

Un programa puede asignar una prioridad a un mensaje cuando pone el mensaje en una cola. Esto determina la posición en la cola a la que se añade el nuevo mensaje.

Los programas pueden obtener mensajes de una cola en el orden en el que los mensajes están en la cola, o bien obteniendo un mensaje específico. (Un programa puede desear obtener un mensaje específico si está buscando la respuesta a una solicitud que envió anteriormente.)

Seguridad

Se proporcionan funcionalidades de seguridad, incluyendo la autenticación de aplicaciones cuando utilizan un gestor de colas, comprobaciones de autorización cuando utilizan recursos como, por ejemplo, una cola en el gestor de colas y cifrado de datos de mensajes en la red y dentro de las colas. Para obtener más información sobre seguridad, consulte Seguridad de canal.

Integridad de datos

La integridad de los datos se proporciona mediante unidades de trabajo. La sincronización del inicio y el final de las unidades de trabajo está totalmente soportada como una opción en cada MQGET o MQPUT, permitiendo que el resultado de la unidad de trabajo se confirma o se restituya. El soporte de punto de sincronismo funciona de forma interna o externa a IBM MQ en función de la forma de coordinación de puntos de sincronismo seleccionada para la aplicación.

Soporte de recuperación

Para que sea posible la recuperación, se registran todas las actualizaciones de IBM MQ persistentes. Si es necesaria la recuperación, todos los mensajes permanentes se restauran, todas las transacciones en curso se retrotraen y todas las confirmaciones y restituciones del punto de sincronismo se manejan de la manera habitual empleada por el gestor de puntos de sincronismo que tiene el control. Para obtener más información sobre mensajes permanentes, consulte Persistencia de mensajes.

Terminología sobre la colocación de mensajes en colas

Esta información ofrece una perspectiva de algunos términos que se emplean con relación a la colocación de mensajes en colas.

Estos incluyen:

- Canales
- Clúster
- IBM MQ MQI client
- z/OS Transferencia a colas dentro del grupo
- Mensaje
- Agente de canal de mensajes
- Descriptor de mensaje
- Punto a punto
- Publicar/Suscribir
- Cola

- Gestor de colas
- z/OS Grupo de compartición de colas
- z/OS Cola compartida
- Suscripción
- Tema

Canales

Los canales se utilizan para mover los mensajes desde un gestor de colas a otro y protegen las aplicaciones de los protocolos de comunicaciones subyacentes. Los gestores de colas pueden estar en sistemas diferentes de la misma plataforma o de plataformas distintas. Los mensajes que se envían pueden tener su origen en muchas ubicaciones:

- Programas de aplicación escritos por el usuario que transfieren datos de un nodo a otro.
- Aplicaciones de administración escritas por usuario que utilizan mandatos PCF o MQAI.
- IBM MQ Explorer.
- Gestores de colas que envían mensajes de sucesos de instrumentación a otro gestor de colas.
- Gestores de colas que envían mandatos de administración remotos a otro gestor de colas. Por ejemplo, utilizando los mandatos MQSC o la administrative REST API.

Para obtener más información sobre los canales, consulte “Definiciones de canal” en la página 32.

Clúster

Un *clúster* es una red de gestores de colas que están asociados lógicamente de alguna forma.

En una red IBM MQ que utiliza colas distribuidas sin agrupación en clúster, cada gestor de colas es independiente. Si un gestor de colas necesita enviar mensajes a otro, debe haber definido una cola de transmisión y un canal en el gestor de colas remoto.

Hay dos razones distintas para utilizar clústeres: reducir la administración del sistema y mejorar la disponibilidad y el equilibrio de la carga de trabajo.

En cuanto establezca incluso el clúster más pequeño, se beneficiará de una administración simplificada del sistema. Los gestores de colas que forman parte de un clúster necesitan menos definiciones y, por lo tanto, el riesgo de cometer un error en las definiciones es menor.

Puede obtener información adicional sobre agrupaciones en clúster consultando Clústeres.

IBM MQ MQI client

IBM MQ MQI *clientes* son componentes instalables de forma independiente de IBM MQ. Un cliente MQI permite ejecutar aplicaciones IBM MQ con un protocolo de comunicaciones, interactuar con uno o varios servidores de interfaz de cola de mensajes (MQI) en otras plataformas y conectarse a sus gestores de colas.

Para obtener detalles completos sobre cómo instalar y utilizar los componentes de IBM MQ MQI client, consulte los temas siguientes:

- AIX Instalación de un cliente IBM MQ en AIX
- Linux Instalación de un cliente IBM MQ en Linux®
- Windows Instalación de un cliente IBM MQ en Windows
- IBM i Instalación de un cliente IBM MQ en IBM i

y Configuración de las conexiones entre el servidor y el cliente.

Transferencia a colas entre grupos

z/OS

Los gestores de colas de un grupo de compartición de colas se pueden comunicar utilizando canales normales o bien pueden utilizar una técnica que se denomina *transferencia a colas dentro del grupo* (IGQ), que cual permite realizar una transferencia rápida de mensajes sin definir canales. Esto solo se aplica a IBM MQ for z/OS.

Para obtener más información sobre la transferencia a colas dentro del grupo, consulte "Intra-group queuing" en la página 215.

Mensaje

En la colocación de mensajes en colas, un mensaje es una colección de datos enviados por un mensaje y dirigidos a otro programa. Consulte Mensajes de IBM MQ.

Para obtener información sobre los tipos de mensajes, consulte Tipos de mensajes.

Agente de canal de mensajes

Un agente de canal de mensajes es un extremo de un canal. Un par de agentes de canal de mensajes, uno emisor y otro receptor, configuran un canal y mueven mensajes de un gestor de colas a otro.

Para obtener información sobre cómo se utilizan los agentes de canal de mensajes, consulte Introducción a la gestión de colas distribuidas.

Descriptor de mensaje

Un mensaje de IBM MQ consta de información de control y datos de aplicación.

La información de control está definida en una estructura de descriptor de mensaje (MQMD) y contiene elementos como:

- El tipo de mensaje
- Un identificador del mensaje
- La prioridad de entrega del mensaje

La estructura y el contenido de los datos de la aplicación están determinados por los programas participantes, no por IBM MQ.

Para obtener más información, consulte MQMD.

Mensajería punto a punto

En la mensajería punto a punto, cada mensaje se traslada de una aplicación de producción a una aplicación de consumo. Los mensajes se transfieren mediante la aplicación de producción que coloca los mensajes en una cola, y la aplicación de consumo los obtiene de dicha cola.

Mensajería de publicación/suscripción

En la mensajería de publicación/suscripción, una copia de cada mensaje publicado por una aplicación de publicación se entrega a todas las aplicaciones interesadas. Puede haber muchas, una o ninguna aplicación interesada. En la publicación/suscripción, una aplicación interesada se conoce como un suscriptor y los mensajes se ponen en cola en una cola identificada mediante una suscripción.

Para obtener más información, consulte "Mensajería de publicación/suscripción" en la página 63.

Cola

Un destino designado al que se pueden enviar mensajes. Los mensajes se acumulan en las colas hasta que los recuperan programas que atienden esas colas.

Para obtener más información, consulte “Colas” en la página 20.

Gestor de colas

Un *gestor de colas* es un programa del sistema que proporciona servicios de colocación de mensajes en colas a las aplicaciones.

Proporciona una interfaz de programación de aplicaciones para que los programas puedan transferir y obtener mensajes de las colas. Un gestor de colas proporciona funciones adicionales para que los administradores puedan crear colas nuevas, alterar las propiedades de las colas existentes y controlar el funcionamiento del gestor de colas.

Para que los servicios de colas de mensajes de IBM MQ estén disponibles en un sistema, debe haber un gestor de colas en ejecución. Puede haber más de un gestor de colas en ejecución en un único sistema (por ejemplo, para separar un sistema de prueba de un sistema *activo*). En una aplicación, cada gestor de colas está identificado con un *manejador de conexión* (*Hconn*).

Muchas aplicaciones diferentes pueden utilizar los servicios del gestor de colas al mismo tiempo y estas aplicaciones pueden no estar relacionadas en absoluto. Para que un programa utilice los servicios de un gestor de colas, debe establecer una conexión con ese gestor de colas.

Para que las aplicaciones envíen mensajes a las aplicaciones que están conectadas a otros gestores de colas, los gestores de colas deben ser capaces de comunicarse entre sí. IBM MQ implementa un protocolo de *almacén y reenvío* para garantizar los mensajes se entregan de un modo seguro entre dichas aplicaciones.

Para obtener más información, consulte “Gestores de colas” en la página 29.

Grupo de compartición de colas

Los gestores de colas que pueden acceder al mismo conjunto de colas compartidas forman un grupo denominado *grupo de compartición de colas* (QSG). Se comunican entre sí con un recurso de acoplamiento (CF) que almacena las colas compartidas. Esto solo se aplica a IBM MQ for z/OS.

Para obtener más información, consulte “Shared queues and queue sharing groups” en la página 171.

Cola compartida

Una *cola compartida* es un tipo de cola local con mensajes a los que pueden acceder uno o varios gestores de colas que están en un sysplex. No es lo mismo que una cola compartida por más de una aplicación, que utiliza el mismo gestor de colas. Esto solo se aplica a IBM MQ for z/OS.

Suscripción

Una aplicación de publicación/suscripción puede registrar un interés en mensajes sobre temas específicos. Cuando una aplicación hace esto, se la denomina suscriptor y el término suscripción define cómo se ponen en cola los mensajes coincidentes para su proceso.

Una suscripción contiene información sobre la identidad del suscriptor y la identidad de la cola de destino en la que se van a colocar publicaciones. También contiene información sobre cómo se colocará una publicación en la cola de destino.

Para obtener más información, consulte “Suscriptores y suscripciones” en la página 66.

Tema

Un tema es una serie de caracteres que describe el asunto de la información que se publica en un mensaje de publicación/suscripción.

Los temas son fundamentales para la entrega satisfactoria de mensajes en un sistema de publicación/suscripción. En lugar de incluir una dirección de destino específica en cada mensaje, un publicador asigna un tema a cada mensaje. El gestor de colas correlaciona el tema con una lista de suscriptores que se han suscrito a ese tema, y entrega el mensaje a cada uno de esos suscriptores.

Para obtener más información, consulte "Temas" en la página 69.

Mensajes y colas

Los mensajes y las colas son los componentes básicos de un sistema de colocación de mensajes en colas.

¿Qué es un mensaje?

Un *mensaje* es una serie de bytes que tiene un significado para las aplicaciones que lo utilizan. Los mensajes se utilizan para transferir información de un programa de aplicación a otro (o a distintas partes de la misma aplicación). Las aplicaciones pueden estar ejecutándose en la misma o en distintas plataformas.

Un mensaje IBM MQ se compone de:

- *Los datos de la aplicación.*El contenido y la estructura de los datos de la aplicación los definen los programas de aplicación que los utilizan.
- *Un descriptor de mensaje.*El descriptor del mensaje identifica el mensaje y contiene información de control adicional, como el tipo de mensaje y la prioridad que ha asignado al mensaje la aplicación emisora.

  El formato del descriptor de mensaje se define en IBM MQ. Si desea una descripción completa del descriptor de mensajes, consulte MQMD - descriptor de mensajes.
- *Propiedades de mensaje.* Metadatos del mensaje. El contenido de las propiedades del mensaje lo definen los programas de aplicación de las utilizan. Puede obtener información adicional consultando Propiedades de un mensaje.

Longitudes de los mensajes

La longitud máxima predeterminada de un mensaje es de 4 MB, aunque puede aumentarla a una longitud máxima de 100 MB (donde 1 MB equivale a 1.048.576 bytes). En la práctica, la longitud del mensaje puede estar limitada por:

- La longitud máxima de mensajes definida para la cola receptora
- La longitud máxima de mensajes definida para el gestor de colas
- La longitud máxima de mensajes definida por la cola
- La longitud máxima de mensajes definida por la aplicación emisora o receptora
- La cantidad de almacenamiento disponible para el mensaje.

Pueden ser necesarios varios mensajes para enviar toda la información que una aplicación necesita.

¿Cómo envían y reciben mensajes las aplicaciones?

Los programas de aplicación envían y reciben mensajes utilizando las **llamadas MQI**.

Por ejemplo, para transferir un mensaje a una cola, una aplicación:

1. Abre la cola necesaria emitiendo una llamada `MQOPEN` de MQI
2. Emite una llamada `MQPUT` de MQI para transferir el mensaje a la cola

Otra aplicación puede recuperar el mensaje de la misma cola emitiendo una llamada `MQGET` de MQI.

Para obtener más información sobre las llamadas MQI, consulte Llamadas MQI.

¿Qué es una cola?

Una *cola* es una estructura de datos que se utiliza para almacenar mensajes.

Cada cola es propiedad de un *gestor de colas*. El gestor de colas es responsable del mantenimiento de las colas de su propiedad y del almacenamiento de todos los mensajes que recibe en las colas adecuadas. Los mensajes se pueden transferir a la cola mediante programas de aplicación o mediante un gestor de colas como parte de sus operaciones normales.

Colas predefinidas y colas dinámicas

Las colas pueden distinguirse por la forma en que se han creado:

- Las **colas predefinidas** las crea un administrador mediante los mandatos MQSC o PCF adecuados.Las colas predefinidas son permanentes; existen independientemente de las aplicaciones que las utilizan y sobreviven a los reinicios de IBM MQ.
- Las **colas dinámicas** se crean cuando una aplicación emite una petición MQOPEN especificando el nombre de una *cola modelo*. La cola creada se basa en una *plantilla de definición de cola* que es la cola modelo.Puede crear una cola modelo utilizando el mandato MQSC DEFINE QMODEL. Los atributos de una cola modelo, por ejemplo, el número máximo de mensajes que se pueden almacenar en la cola, son heredados por cualquier cola dinámica que se cree a partir de la cola modelo.

  Las colas modelo tienen un atributo que especifica si la cola dinámica va a ser persistente o temporal. Las colas persistentes perduran después de los reinicios del gestor de colas y de la aplicación; las colas temporales se pierden al reiniciar.

Recuperación de mensajes de colas

Las aplicaciones debidamente autorizadas pueden recuperar mensajes de una cola en función de los siguientes algoritmos de recuperación:

- Primero en entrar, primero en salir (FIFO).
- Prioridad del mensaje, según lo definido en el descriptor del mensaje. Los mensajes que tienen la misma prioridad se recuperan según el método FIFO.
- Una petición de programa para un mensaje específico.

La petición MQGET de la aplicación determina el método utilizado.

Objetos de IBM MQ

Los gestores de colas definen las propiedades de los objetos de IBM MQ. Los valores de estas propiedades afectan al modo en que IBM MQ procesa estos objetos. Puede crear y gestionar objetos utilizando mandatos e interfaces de IBM MQ. Desde sus aplicaciones, utilice la interfaz de cola de mensajes (MQI) para controlar los objetos. Los objetos se identifican mediante un *descriptor de objeto* (MQOD) de IBM MQcuando se direccionan desde un programa.

Administración de objetos

La administración de objetos incluye las tareas siguientes:

- Iniciar y detener gestores de colas.
- Crear objetos, especialmente colas, para aplicaciones.
- Mostrar o alterar los atributos de objetos.
- Suprimiendo objetos.
- Trabajar con canales para crear vías de comunicación con gestores de colas existentes en otros sistemas (remotos).
- Crear *clústeres* de gestores de colas para simplificar el proceso global de administración o para equilibrar la carga de trabajo.

Con la excepción de las colas dinámicas, los objetos deben definirse ante el gestor de colas para poder trabajar con ellos.

Cuando utiliza un mandato de IBM MQ para llevar a cabo una operación de administración de objetos, el gestor de colas comprueba si dispone del nivel de autorización necesario para realizar la operación. Del mismo modo, cuando una aplicación utiliza la llamada MQOPEN para abrir un objeto, el gestor de colas comprueba si la aplicación dispone del nivel de autorización necesario antes de permitir el acceso a dicho objeto. Las comprobaciones se realizan en el nombre del objeto que se abre.

Puede definir y gestionar objetos mediante los métodos siguientes:

- Los mandatos PCF que se describen en Guía de consulta sobre formatos de mandatos programables y Automatización de tareas de administración
- Los mandatos MQSC que se describen en Mandatos MQSC
- z/OS Los paneles de operaciones y de control de IBM MQ for z/OS , descritos en Administración IBM MQ for z/OS
- Windows Linux IBM MQ Explorer (solamente para sistemas Windows y Linux for Intel). Para obtener más información, consulte Introducción a MQ Explorer.

También puede gestionar objetos mediante los métodos siguientes:

- Con los mandatos de control, que se escriben desde un teclado. Consulte Administración de IBM MQ for Multiplatforms utilizando mandatos de control.
- Las llamadas de la interfaz de administración de IBM MQ (MQAI) en un programa. Consulte Interfaz de administración de IBM MQ (MQAI).

ALW En el caso de las secuencias de mandatos de IBM MQ en AIX, Linux, and Windows, puede utilizar el recurso MQSC para ejecutar una serie de mandatos que se guardan en un archivo. Para obtener más información, consulte Administración de IBM MQ utilizando mandatos MQSC.

IBM i En el caso de las secuencias de mandatos de IBM MQ para IBM i que utiliza regularmente, puede escribir programas CL. Para obtener más información, consulte Gestión de IBM MQ for IBM i utilizando mandatos CL.

z/OS En el caso de las secuencias de mandatos de IBM MQ for z/OS que utiliza regularmente, puede escribir programas de administración que creen mensajes que contienen mandatos y que colocan estos mensajes en la cola de entrada del mandato del sistema. El gestor de colas procesa los mensajes en esta cola del mismo modo que procesa los mandatos entrados desde la línea de mandatos o bien desde las operaciones y los paneles de control. Esta técnica se describe en Escribir programas para administrar IBM MQ y se muestra en la aplicación de ejemplo Mail Manager que se suministra con IBM MQ for z/OS. Para obtener una descripción de este ejemplo, consulte Programas de ejemplo para IBM MQ for z/OS.

Atributos de objetos

Las propiedades de un objeto se definen mediante sus atributos. Algunas pueden especificarse, mientras que otras sólo pueden verse.

Por ejemplo, la longitud máxima de mensaje que una cola puede admitir está definida por su atributo **MaxMsgLength**; este atributo se puede especificar cuando se crea una cola. El atributo **DefinitionType** especifica cómo se ha creado la cola; este atributo sólo se puede visualizar.

En IBM MQ, hay dos formas de referirse a un atributo:

- Utilizando su nombre PCF, por ejemplo, **MaxMsgLength**.
- Utilizando su nombre de mandato MQSC, por ejemplo, MAXMSGL.

Grupos de compartición de colas

z/OS

Los gestores de colas que pueden acceder al mismo conjunto de colas compartidas forman un grupo llamado *grupo de compartición de colas* (QSG) y se comunican entre sí utilizando un recurso de acoplamiento (CF) que almacena las colas compartidas. Tenga en cuenta que un QSG no es estrictamente un objeto.

Una cola compartida es un tipo de cola local con mensajes a los que pueden acceder uno o varios gestores de colas que están en un grupo de compartición de colas. No es lo mismo que una cola compartida por más de una aplicación, que utiliza el mismo gestor de colas.

Los grupos de compartición de colas tienen un nombre de hasta cuatro caracteres. El nombre debe ser exclusivo en la red y debe ser diferente de los nombres de gestor de colas.

Importante: Las colas compartidas y los grupos de compartición de colas solo están soportados en IBM MQ for z/OS.

Consulte “Shared queues and queue sharing groups” en la página 171 para obtener más información.

Objetos predeterminados del sistema

Los *objetos predeterminados del sistema* son un conjunto de definiciones de objetos que se crean automáticamente siempre que se crea un gestor de colas. Puede copiar y modificar cualquiera de estas definiciones de objetos para utilizarlas en aplicaciones de su instalación.

Los nombres de los objetos predeterminados tienen la raíz SYSTEM; por ejemplo, la cola local predeterminada es SYSTEM.DEFAULT.LOCAL.QUEUE y el canal receptor predeterminado es SYSTEM.DEF.RECEIVER. No se puede cambiar el nombre de estos objetos; es obligatorio que los objetos predeterminados tengan estos nombres.

Cuando se define un objeto, todos los atributos que no se especifican explícitamente se copian del objeto predeterminado adecuado. Por ejemplo, si define una cola local, los atributos que no especifique se tomarán de la cola predeterminada SYSTEM.DEFAULT.LOCAL.QUEUE.

Consulte Objetos del sistema y predeterminados para obtener más información.

Tipos de objetos

Muchas de las tareas de administración consisten en manipular varios tipos de IBM MQ *objetos*.

Para obtener información sobre la denominación de objetos IBM MQ, consulte “Denominación de objetos de IBM MQ” en la página 37.

Para obtener información acerca de los objetos predeterminados creados en un gestor de colas, consulte “Objetos predeterminados del sistema” en la página 16.

Para obtener información sobre los distintos tipos de objetos de IBM MQ , consulte lo siguiente:

Objetos de información de autenticación

Un objeto de información de autenticación proporciona las definiciones necesarias para realizar la comprobación de revocación de certificados.

El objeto de información de autenticación del gestor de colas forma parte del soporte de IBM MQ para la seguridad de la capa de transporte (TLS). Proporciona las definiciones necesarias para comprobar los certificados revocados. Las autoridades de certificación revocan los certificados que ya no son fiables.

Puede utilizar el mandato MQSC **DEFINE AUTHINFO** para definir un objeto de información de autenticación. Para obtener más información sobre los atributos de los objetos de información de autenticación, consulte **DEFINE AUTHINFO**.

Puede utilizar los siguientes mandatos de control de IBM MQ con un objeto de información de autenticación:

- **setmqaut** (otorgar o revocar autorización)
- **dspmqaut** (visualizar autorización de objeto)
- **dmpmqaut** (volcar autorizaciones)
- **rcrmqobj** (volver a crear objeto)
- **rcdmqimg** (registrar imagen de soporte)
- **dspmqfls** (visualizar nombres de archivos)

Para obtener una visión general de TLS y el uso de los objetos de información de autenticación, consulte Conceptos de seguridad de la capa de transporte (TLS) y Protocolos de seguridad TLS en IBM MQ.

Canales

Los canales son objetos que proporcionan una vía de comunicación de un gestor de colas a otro.

Consulte “Canales” en la página 30 para obtener más información.

Objetos de información de comunicación

IBM MQ Multicast ofrece baja latencia, alta diseminación y mensajería de multidifusión fiable. Se precisa un objeto de información de comunicación (COMMINFO) para utilizar la transmisión Multidifusión.

Consulte “IBM MQ Multicast” en la página 108 para obtener más información.

Un objeto COMMINFO es un objeto de IBM MQ que contiene los atributos asociados a la transmisión de multidifusión. Para obtener más información sobre estos atributos, consulte DEFINE COMMINFO. Para obtener más información sobre cómo crear un objeto COMMINFO, consulte Iniciación con la multidifusión.

Escuchas

Los *escuchas* son procesos que aceptan peticiones de red de otros gestores de colas o aplicaciones cliente e inician los canales asociados.

Los *procesos de escucha* se pueden iniciar utilizando el mandato de control **runmqlsr** .

Los *objetos de escucha* son objetos de IBM MQ que le permiten gestionar el inicio y la detención de procesos de escucha desde el ámbito de un gestor de colas. Al definir los atributos de un objeto de escucha, hace lo siguiente:

- Configurar el proceso de escucha.
- Especificar si el proceso de escucha se inicia y se detiene automáticamente cuando se inicia y se detiene el gestor de colas.

Importante: z/OS Los objetos de escucha no están soportados en IBM MQ for z/OS. Para obtener más información sobre cómo IBM MQ for z/OS implementa la escucha, utilizando el iniciador de canal, consulte “The channel initiator on z/OS” en la página 167.

Listas de nombres

Una *lista de nombres* es un objeto de IBM MQ que contiene una lista de nombres de clúster, nombres de colas o nombres de objetos de información de autenticación. En un clúster, se puede utilizar para identificar una lista de clústeres para los que el gestor de colas contiene los repositorios.

Una lista de nombres es un objeto de IBM MQ que contiene una lista de otros objetos de IBM MQ. Normalmente, las listas de nombres las utilizan aplicaciones como los supervisores desencadenantes y se utilizan para identificar un grupo de colas. La ventaja de utilizar una lista de nombres es que se mantiene independientemente de las aplicaciones, es decir, se puede actualizar sin detener ninguna de las aplicaciones que la utilizan. Además, si una aplicación no se ejecuta correctamente, ello no afecta a la lista de nombres y otras aplicaciones podrán seguir utilizándola.

Las listas de nombres también se utilizan con los clústeres de gestores de colas para mantener una lista de clústeres a los que hace referencia más de un objeto IBM MQ .

Puede definir y modificar listas de nombres utilizando los mandatos MQSC DEFINE NAMELIST y ALTER NAMELIST .

Nota: z/OS De forma alternativa, en z/OS, puede utilizar los paneles de operaciones y control de IBM MQ for z/OS

Los programas pueden utilizar MQI para averiguar qué colas están incluidas en estas listas de nombres. La organización de las listas de nombres es responsabilidad del diseñador de aplicaciones y administrador del sistema.

Para obtener una lista de los atributos de lista de nombres disponibles para utilizar, consulte Atributos para listas de nombres,

Definiciones de procesos

Los objetos de definición de proceso permiten que las aplicaciones se inicien sin necesidad de que intervenga el operador definiendo los atributos de la aplicación para que los utilice el gestor de colas.

El objeto de definición de proceso define una aplicación que se inicia como respuesta a un suceso desencadenante en un gestor de colas de IBM MQ. Los atributos de la definición de proceso incluyen el ID de aplicación, el tipo de aplicación y los datos específicos de la aplicación. Para obtener más información, consulte *Colas de inicio* en “Colas utilizadas para fines específicos por IBM MQ” en la página 27.

Para que una aplicación se pueda iniciar sin la necesidad de la intervención del operador, tal como se describe en Inicio de aplicaciones de IBM MQ utilizando desencadenantes, los atributos de la aplicación deben ser conocidos para el gestor de colas. Estos atributos se definen en un *objeto de definición de proceso*.

El atributo **ProcessName** es fijo cuando se crea el objeto. Sin embargo, puede cambiar otros atributos utilizando los mandatos de IBM MQ.

Nota: z/OS De forma alternativa, en z/OS, puede utilizar los paneles de operaciones y de control de IBM MQ for z/OS .

Puede averiguar los valores de todos los atributos utilizando MQINQ - Consultar atributos de objetos.

Para obtener una lista de los atributos de definición de proceso disponibles para utilizar, consulte Atributos para definiciones de proceso.

Colas

Un IBM MQ *cola* es un objeto con nombre en el que las aplicaciones pueden transferir mensajes y del que las aplicaciones pueden obtener mensajes.

Consulte “Colas” en la página 20 para obtener más información.

Gestores de colas

IBM MQ Los gestores de colas proporcionan servicios de colocación en colas a las aplicaciones y gestionan las colas que les pertenecen.

Consulte “Gestores de colas” en la página 29 para obtener más información.

Servicios

Los objetos de *servicio* son una forma de definir programas para que se ejecuten cuando se inicie o se detenga un gestor de colas.

Los programas pueden ser de uno de los tipos siguientes:

Servidores

Un *servidor* es un objeto de servicio que tiene el parámetro SERVTYPE establecido en SERVER. Un objeto de servicio de servidor es la definición de un programa que se ejecutará cuando se inicie un gestor de colas especificado. Sólo se puede ejecutar una instancia de un proceso de servidor al mismo tiempo. Mientras se está ejecutando, el estado de un proceso de servidor puede supervisarse con el mandato MQSC, DISPLAY SVSTATUS. Normalmente, los objetos de servicio de servidor son definiciones de programas como, por ejemplo, manejadores de mensajes no entregados o supervisores de desencadenantes, sin embargo, los programas que se pueden ejecutar no se limitan a los que se suministran con IBM MQ. Además, un objeto de servicio de servidor se puede definir para que incluya un mandato que se ejecutará cuando el gestor de colas especificado se apague para finalizar el programa.

Mandatos

Un *mandato* es un objeto de servicio que tiene el parámetro SERVTYPE establecido en COMMAND. Un objeto de servicio de mandato es la definición de un programa que se ejecutará cuando se inicie o se detenga un gestor de colas especificado. Se pueden ejecutar varias instancias de un proceso de mandato al mismo tiempo. Los objetos de servicio de mandato se diferencian de los objetos de servicio de servidor en que, una vez que se haya ejecutado el programa, el gestor de colas no lo supervisará. Normalmente, los objetos de servicio de mandato son definiciones de programas de corta duración que realizan una tarea específica, como por ejemplo iniciar una o más tareas diferentes.

Importante: z/OS Los objetos de servicio no están soportados en IBM MQ for z/OS.

Para obtener más información, consulte Trabajar con servicios.

Clases de almacenamiento

z/OS

Una clase de almacenamiento correlaciona una o más colas con un conjunto de páginas.

Esto significa que los mensajes de dicha cola se almacenan (están sujetos al almacenamiento intermedio) en ese conjunto de páginas.

Las clases de almacenamiento solamente están soportadas en IBM MQ for z/OS.

Para obtener más información sobre las clases de almacenamiento, consulte Planificación del entorno de IBM MQ en z/OS.

Objetos de tema

Un *objeto de tema* es un objeto de IBM MQ que permite asignar atributos específicos, no predeterminados a temas.

Un *tema* se define por una aplicación que se publica en, o se suscribe a, una *serie de temas* determinada. Una serie de tema puede especificar una jerarquía de temas separándolos con un carácter de barra inclinada (/). Esto se puede visualizar con un *árbol de temas*. Por ejemplo, si una aplicación se publica en las series de temas `/Sport/American Football` y `/Sport/Soccer`, se crea un árbol de temas que tiene un nodo padre `Sport` y dos hijos, `American Football` y `Soccer`.

Los temas heredan sus atributos del primer nodo administrativo padre encontrado en su árbol de temas. Si no hay nodos de tema administrativo en un árbol de temas específico, todos los temas heredan los atributos del objeto de tema base, SYSTEM.BASE.TOPIC.

Puede crear un objeto de tema en cualquier nodo de un árbol de temas especificando la serie de temas de dicho nodo en el atributo TOPICSTR del objeto del tema. También puede definir otros atributos para el nodo de tema administrativo. Para obtener más información sobre estos atributos, consulte Los mandatos MQSC o Automatización de la administración utilizando mandatos PCF. Cada objeto de tema hereda, de forma predeterminada, sus atributos del nodo de tema administrativo padre más cercano.

Los objetos del tema también se pueden utilizar para ocultar todo el árbol de temas de los desarrolladores de aplicación. Si se crea un objeto de tema denominado `FOOTBALL.US` para el tema `/`

`Sport/American Football`, una aplicación puede publicar o suscribirse al objeto denominado `FOOTBALL.US` en lugar de a la serie `/Sport/American Football` con el mismo resultado.

Si indica un carácter #, +, / ó * en una serie de tema de un objeto de tema, el carácter se tratará como un carácter normal en la serie y se considerará que forma parte de la serie de temas asociada a un objeto de tema.

Para obtener más información sobre los objetos de temas, consulte "Mensajería de publicación/ suscripción" en la página 63.

Conceptos relacionados

"Introducción a la colocación de mensajes en colas" en la página 5
Los productos de IBM MQ permiten que los programas se comuniquen entre sí a través de una red de componentes diferentes (procesadores, sistemas operativos, subsistemas y protocolos de comunicación) utilizando una interfaz de programación de aplicaciones coherente.

Referencia relacionada

Mandatos MQSC

Colas

Introducción a las colas y los atributos de cola de IBM MQ.

Los mensajes se almacenan en una cola, de modo que si la aplicación que realiza la transferencia está esperando una respuesta a su mensaje, es libre de realizar otros trabajos mientras espera esa respuesta. Las aplicaciones acceden a una cola utilizando la Interfaz de colas de mensajes (MQI), que se describe en Visión general de la interfaz de colas de mensajes.

Para que un mensaje pueda transferirse a una cola, la cola ya se tiene que haber creado. Una cola es propiedad de un gestor de colas, y ese gestor de colas puede poseer muchas colas. Sin embargo, cada cola debe tener un nombre que sea exclusivo dentro de ese gestor de colas.

Una cola se mantiene a través de un gestor de colas. En la mayoría de los casos, cada cola la gestiona físicamente el gestor de colas pero esto no es evidente para un programa de aplicación. Las colas compartidas de IBM MQ for z/OS las puede gestionar cualquier gestor de colas del grupo de compartición de colas.

Para crear una cola, puede utilizar mandatos de IBM MQ (MQSC), mandatos PCF o interfaces específicas de la plataforma. Por ejemplo, las operaciones y los paneles de control de IBM MQ for z/OS son específicos de la plataforma.

Puede crear colas locales para trabajos temporales *dinámicamente* desde la aplicación. Por ejemplo, puede crear colas de *respuestas* (que no son necesarias después de que finalice una aplicación). Para obtener más información, consulte "Colas dinámicas y de modelo" en la página 25.

Antes de utilizar una cola, debe abrirla, especificando qué desea hacer con ella. Por ejemplo, puede abrir una cola para:

- Examinar mensajes únicamente (no recuperarlos)
- Recuperar mensajes (y compartir el acceso con otros programas o con acceso exclusivo)
- Transferir mensajes a la cola
- Realizar consultas sobre los atributos de la cola
- Establecer los atributos de la cola

Para obtener una lista completa de las opciones que puede especificar cuando se abre una cola, consulte MQOPEN – Abrir objeto.

Atributos de colas

Algunos de los atributos de una cola se especifican cuando la cola está definida, y no se puede cambiar después (por ejemplo, el tipo de la cola). Otros atributos de colas se pueden agrupar en aquellos que se pueden cambiar:

- Mediante el gestor de colas durante el proceso de la cola (por ejemplo, la profundidad actual de una cola)
- Con mandatos únicamente (por ejemplo, el texto descriptivo de la cola)
- Mediante aplicaciones, utilizando la llamada MQSET (por ejemplo, si se permiten operaciones de transferencia en la cola)

Puede averiguar los valores de todos los atributos utilizando la llamada MQINQ.

Los atributos que son comunes a más de un tipo de cola son los siguientes:

QName
Nombre de la cola.

QType
Tipo de la cola.

QDesc
Texto descriptivo de la cola.

InhibitGet
Si se permite que los programas obtengan mensajes de la cola. Sin embargo, nunca puede obtener mensajes de colas remotas.

InhibitPut
Si se permite que los programas transfieran mensajes a la cola.

DefPriority
Prioridad predeterminada de los mensajes transferidos a la cola.

DefPersistence
Persistencia predeterminada de los mensajes transferidos a la cola

Ámbito
Controla si una entrada para esta cola también existe en un servicio de nombres.

z/OS El atributo **Scope** no está soportado en z/OS.

Para obtener una descripción completa de estos atributos, consulte Atributos para colas.

Formas de definir las colas

Puede definir colas en IBM MQ utilizando el mandato MQSC DEFINE o el mandato PCF Crear cola. Los mandatos especifican el tipo de cola y sus atributos. Por ejemplo, un objeto de cola local tiene atributos que especifican lo que sucede cuando las aplicaciones hacen referencia a dicha cola en llamadas MQI. Son ejemplos de atributos:

- Si las aplicaciones pueden recuperar mensajes de la cola (GET habilitado).
- Si las aplicaciones pueden transferir mensajes a la cola (PUT habilitado).
- Si el acceso a la cola es exclusivo de una aplicación o lo comparten varias aplicaciones.
- El número máximo de mensajes que pueden almacenarse en la cola al mismo tiempo (profundidad máxima de cola).
- La longitud máxima de los mensajes que se pueden transferir a la cola.

También hay varias interfaces específicas de la plataforma que puede utilizar para definir colas.

Conceptos relacionados

“Colas de clúster” en la página 59
Una cola de clúster es una cola que se aloja en un gestor de colas de clúster y que está disponible para otros gestores de colas del clúster.

“Colas de mensajes no entregados” en la página 50
La cola de mensajes no entregados es la cola a la que se envían los mensajes que no pueden enviarse a su destino correcto. Cada gestor de colas suele tener una cola de mensajes no entregados.

Automatización de administración utilizando mandatos PCF

IBM MQ Console: Trabajar con colas

Tareas relacionadas

Administración de IBM MQ utilizando mandatos MQSC

Creación y configuración de gestores de colas y objetos con MQ Explorer

IBM i Gestión de IBM MQ for IBM i utilizando mandatos CL

z/OS Orígenes desde los que puede emitir mandatos MQSC y PCF en IBM MQ for z/OS

Referencia relacionada

“Comparison between shared queues and cluster queues” en la página 59

This information is designed to help you compare shared queues and cluster queues, and decide which might be more suitable for your system.

Información relacionada

“What is a shared queue?” en la página 171

Colas locales

Las colas de transmisión, de iniciación, de mensajes no entregados, predeterminada, de canal y de sucesos son tipos de cola local.

Una cola se conoce en un programa como *local* si es propiedad del gestor de colas al que está conectado el programa. Puede obtener mensajes y transferirlos en las colas locales.

El objeto de definición de cola contiene la información de definición de la cola, así como los mensajes físicos colocados en cola.

Cada gestor de colas puede tener algunas colas locales que utiliza para fines especiales:

Colas de transmisión

Cuando una aplicación envía un mensaje a una cola remota, el gestor de colas local almacena el mensaje en una cola local especial, llamada *cola de transmisión*. Las aplicaciones pueden transferir mensajes directamente a una cola de transmisión aunque también pueden transferirse indirectamente a través de una definición de cola remota.

Cuando un gestor de colas envía mensajes a un gestor de colas remoto, identifica la cola de transmisión mediante la secuencia siguiente:

1. La cola de transmisión nombrada en el atributo XMITQ de la definición local de una cola remota.
2. Una cola de transmisión con el mismo nombre que el gestor de colas remoto. Este es el valor predeterminado en XMITQ de la definición local de una cola remota.
3. La cola de transmisión nombrada en el atributo DEFXMITQ del gestor de colas local.

Un *agente de canal de mensajes* es un programa de canal asociado a la cola de transmisión, y que entrega el mensaje a su siguiente destino. El siguiente destino es el gestor de colas al que está conectado el canal de mensajes. No es, necesariamente, el mismo gestor de colas que el destino final del mensaje. Cuando el mensaje se entregue en su siguiente destino, se suprimirá de la cola de transmisión. Es posible que el mensaje tenga que pasar por muchos gestores de colas en su ruta hacia el destino final. Debe definir una cola de transmisión en cada gestor de colas, junto con la ruta, que cada una contiene mensajes que están esperando a transmitirse en el siguiente destino. Una cola de transmisión normal contiene mensajes para el destino siguiente, aunque los mensajes podrían tener destinos eventuales distintos. Una cola de transmisión de clúster contiene mensajes para varios destinos. El valor de `correlID` de cada mensaje identifica el canal en el que se coloca el mensaje, para transferirlo a su siguiente destino.

En un gestor de colas puede definir varias colas de transmisión. Puede definir varias colas de transmisión para el mismo destino; cada uno se utiliza para una clase diferente de servicio. Por ejemplo, es posible que desee crear colas de transmisión diferentes para los mensajes pequeños y grandes que se dirijan al mismo destino. De este modo, puede transferir los mensajes utilizando canales de mensajes diferentes, de forma que los mensajes grandes no retrasarán a los más pequeños. Todos los mensajes dirigidos a colas de clúster o a temas de clúster, se colocan en una

sola cola de transmisión de clúster SYSTEM.CLUSTER.TRANSMIT.QUEUE, de forma predeterminada. Como opción, puede cambiar el valor predeterminado, y separar el tráfico de los mensajes que van a gestores de colas de clúster diferentes en colas de transmisión de clúster diferentes. Si establece el atributo de gestor de colas DEFCLXQ en CHANNEL, cada canal de clúster emisor crea una cola de transmisión de clúster distinta. Como alternativa, puede definir manualmente las colas de transmisión de clúster que los canales de clúster emisor deben utilizar.

Las colas de transmisión pueden desencadenar un agente de canal de mensajes para enviar mensajes hacia adelante; consulte Inicio de aplicaciones IBM MQ utilizando desencadenantes.

z/OS En IBM MQ for z/OS, si se utiliza la transferencia a colas entre grupos, la cola de transmisión recibe servicio de un *agente de transferencia a colas entre grupos*. Se utiliza una cola de transmisión compartida cuando se utiliza la transferencia a colas entre grupos IBM MQ for z/OS.

Colas de inicio

Una *cola de inicio* es una cola local en la que el gestor de colas coloca un mensaje desencadenante cuando se produce un suceso desencadenante en una cola de aplicación.

Un suceso desencadenante es un suceso diseñado para provocar que un programa empiece a procesar una cola. Por ejemplo, un suceso puede ser que lleguen más de 10 mensajes. Para obtener más información sobre cómo funciona el desencadenante, consulte Inicio de aplicaciones IBM MQ utilizando desencadenantes.

Cola de mensajes no entregados (mensaje no entregado)

Una *cola de mensajes no entregados (mensaje no entregado)* es una cola local en la que el gestor de colas coloca mensajes que no puede entregar.

Cuando el gestor de colas coloca un mensaje en la cola de mensajes no entregados, añade una cabecera al mensaje. La información de cabecera incluye el motivo por el cual el gestor de colas ha colocado el mensaje en la cola de mensajes no entregados. También contiene el destino del mensaje original, la fecha y la hora en que el gestor de colas haya colocado el mensaje en la cola de mensajes no entregados.

Las aplicaciones también pueden utilizar la cola para mensajes que no pueden entregar. Para obtener más información, consulte Utilización de la cola de mensajes no entregados.

Cola de mandatos del sistema

La *cola de mandatos del sistema* es una cola a la que aplicaciones debidamente autorizadas pueden enviar mandatos de IBM MQ. Estas colas reciben los mandatos PCF, MQSC y CL, como soporte en su plataforma, en preparación para que el gestor de colas los ejecute.

z/OS En IBM MQ for z/OS , la cola se denomina SYSTEM.COMMAND.INPUT ; en otras plataformas se denomina SYSTEM.ADMIN.COMMAND.QUEUE. Los mandatos aceptados varían según la plataforma. Para obtener más detalles, consulte Referencia de formatos de mandato programable.

Colas predeterminadas del sistema

Las *colas predeterminadas del sistema* contienen las definiciones iniciales de las colas para el sistema. Cuando se crea una definición de cola, el gestor de colas copia la definición de la cola predeterminada del sistema apropiado. El proceso de crear una definición de cola es diferente del de crear una cola dinámica. La definición de la cola dinámica se basa en la cola de modelo que elija como plantilla para la cola dinámica.

Colas de sucesos

Colas de sucesos contienen mensajes de sucesos. Estos mensajes los notifica el gestor de colas o un canal.

Colas remotas

Para un programa, una cola es *remota* si pertenece a un gestor de colas diferentes de aquel al que está conectado el programa.

Cuando se ha establecido un enlace de comunicaciones, un programa puede enviar un mensaje a una cola remota. Un programa nunca podrá obtener un mensaje de una cola remota.

El objeto de definición de cola, creado al definir una cola remota sólo guarda la información necesaria para que el gestor de colas localice la cola al que desea que vaya el mensaje. Este objeto se conoce como la *definición local de una cola remota*. Todos los atributos de la cola remota se guardan en el gestor de colas que lo posee, porque es una cola local para ese gestor de colas.

Cuando se abre una cola remota, para identificar la cola debe especificar uno de los siguientes:

- El nombre de la definición local que define la cola remota. Desde el punto de vista de una aplicación, es lo mismo que abrir una cola local. Una aplicación no necesita saber si una cola es local o remota.

  Para crear una definición local de una cola remota en todas las plataformas excepto IBM i, utilice el mandato DEFINE QREMOTE .

  IBM i En IBM i, utilice el mandato CRTMQMQ.
- El nombre del gestor de colas remoto y el nombre de la cola tal como se le conoce para dicho gestor de colas remoto.

Las definiciones locales de colas remotas tienen tres atributos además de los atributos comunes que se describen en “Atributos de colas” en la página 20. Estos tres atributos son:

RemoteQName
El nombre por el que se conoce en el gestor de colas propietario de la cola.

RemoteQMgrName
El nombre del gestor de colas propietario.

XmitQName
El nombre de la cola de transmisión local que se utiliza cuando se reenvían mensajes a otros gestores de colas.

Para obtener más información sobre estos atributos, consulte Atributos para colas.

Si utiliza la llamada MQINQ para la definición local de una cola remota, el gestor de colas devolverá los atributos de la definición local únicamente, que es el nombre de cola remota, el nombre del gestor de colas remoto y el nombre de la cola de transmisión, no los atributos de la cola local que coinciden en el sistema remoto.

Véase también Colas de transmisión.

Colas alias

Un *alias de cola* es un objeto de IBM MQ que se puede utilizar para acceder a otra cola o a un tema. Esto significa que puede haber más de un programa que trabaje con la misma cola, accediendo a ella con nombres diferentes.

La cola resultante de la resolución de un nombre de alias, que se conoce como la cola base, puede ser cualquiera de los tipos de colas siguientes, soportados por la plataforma:

- Una cola local
- La definición local de una cola remota.
- z/OS Una cola compartida, que es un tipo de cola local solo disponible en IBM MQ for z/OS.
- Una cola predefinida
- Una cola dinámica

Un nombre de alias también se puede resolver en un tema. Si actualmente una aplicación transfiere mensajes a una cola, se puede conseguir que se publique en un tema convirtiendo el nombre de la cola en un alias para el tema. No es necesario ningún cambio en el código de aplicación.

Nota: Un alias no se puede resolver directamente en otro alias en el mismo gestor de colas.

Un ejemplo del uso de las colas alias es que un administrador del sistema otorgue diferentes autorizaciones de acceso al nombre de cola base (es decir, la cola en la que se resuelve el alias) y al nombre de cola alias. Esto significa que un programa o usuario pueden estar autorizados a utilizar la cola alias, pero no la cola base.

Como alternativa, se puede establecer una autorización para inhibir las operaciones de transferir (put) para el nombre de alias, pero permitirlas para la cola base.

En algunas aplicaciones, el uso de las colas alias significa que los administradores del sistema pueden cambiar fácilmente la definición de un objeto de cola alias sin tener que modificar la aplicación.

IBM MQ realiza comprobaciones de autorización con el nombre de alias cuando los programas intentan utilizar ese nombre. No comprueba si el programa está autorizado a acceder al nombre en el que se resuelve el alias. Por consiguiente, un programa puede estar autorizado a acceder a un nombre de cola alias, pero no el nombre de cola resuelto.

Además de los atributos de cola generales que se describen en "Colas" en la página 20, las colas de alias tienen un atributo **BaseQName**. Es el nombre de la cola base en el que se resuelve el nombre de alias. Para obtener una descripción más completa de este atributo, consulte BaseQName (MQCHAR48).

Los atributos *InhibitGet* y **InhibitPut** (consulte "Colas" en la página 20) de las colas alias pertenecen al nombre de alias. Por ejemplo, si el nombre de colas alias ALIAS1 se resuelve en el nombre de la cola base BASE, las inhibiciones ALIAS1 sólo afectan a ALIAS1 y no se inhibe BASE. Sin embargo, las inhibiciones en BASE también afectan a ALIAS1.

Los atributos *DefPriority* y **DefPersistence** también pertenecen al nombre de alias. Por lo tanto, puede por ejemplo asignar diferentes propiedades a diferentes alias de la misma cola base. Además, puede cambiar estas propiedades sin tener que cambiar las aplicaciones que utilizan los alias.

Colas dinámicas y de modelo

Esta información ofrece una perspectiva de las colas dinámicas, las propiedades de las colas dinámicas temporales y permanentes, los usos de las colas dinámicas y algunas consideraciones que hay que tomar en cuenta cuando se utilizan colas dinámicas, y colas modelo.

Cuando un programa de aplicación emite una llamada MQOPEN para abrir una cola modelo, el gestor de colas crea dinámicamente una instancia de una cola local con los mismos atributos que la cola modelo. Según el valor del campo *DefinitionType* de la cola modelo, el gestor de colas crea un cola dinámica temporal o permanente (consulte Creación de colas dinámicas).

Propiedades de las colas dinámicas temporales

Las *colas dinámicas temporales* tienen las propiedades siguientes:

- z/OS No pueden ser colas compartidas, accesibles desde los gestores de colas de un grupo de compartición de colas.

  Tenga en cuenta que los grupos de compartición de colas solo están disponibles en IBM MQ for z/OS.
- Sólo contienen mensajes persistentes
- Son irrecuperables.
- Se suprimen cuando se inicia el gestor de colas.
- Se suprimen cuando la aplicación que ha emitido la llamada MQOPEN que creó la cola cierra la cola o termina.
  - Si hay mensajes confirmados en la cola, se suprimen.
  - Si hay alguna llamada no confirmada MQGET, MQPUT o MQPUT1 para la cola en este momento, la cola se marca como suprimida lógicamente, y sólo se suprimirá físicamente (después de que estas llamadas se hayan confirmado) como parte del proceso de cierre, o cuando la aplicación finalice.

- Si la cola está en uso en este momento (por la aplicación que la crea o cualquier otra), la cola se marca como suprimida lógicamente, y sólo se suprimirá físicamente cuando se cierre la última aplicación que utiliza la cola.
- Los intentos de acceder a una cola suprimida lógicamente (que no se para cerrarla) fallarán con el código de razón MQRC_Q_DELETED.
- MQCO_NONE, MQCO_DELETE y MQCO_DELETE_PURGE se tratan todos como MQCO_NONE cuando se han especificado en una llamada QCLOSE para la llamada MQOPEN correspondiente que ha creado la cola.

Propiedades de las colas dinámicas permanentes

Las *colas dinámicas permanentes* tienen las propiedades siguientes:

- Contienen mensajes persistentes o no persistentes.
- Son recuperables en caso de anomalías del sistema.
- Se suprimen cuando una aplicación (no necesariamente la que ha emitido la llamada MQOPEN que ha creado la cola) cierre satisfactoriamente la cola utilizando la opción MQCO_DELETE o MQCO_DELETE_PURGE.
  - Una petición de cierre con la opción MQCO_DELETE falla si todavía hay algún mensaje (confirmado o sin confirmar) en la cola. Una petición de cierre con la opción MQCO_DELETE_PURGE será satisfactoria aunque haya mensajes confirmados en la cola (los mensajes que están suprimiéndose como parte del cierre), pero no será satisfactoria si hay alguna llamada no confirmada MQGET, MQPUT o MQPUT1 pendiente para la cola.
  - Si la petición de supresión es satisfactoria, pero la cola está en uso (por la aplicación que la crea o cualquier otra), la cola se marca como suprimida lógicamente y sólo se suprimirá físicamente cuando se cierre la última aplicación que utiliza la cola.
- No se suprimen si las cierra una aplicación que no está autorizada a suprimir la cola, a menos que la aplicación de cierre emitiera la llamada MQOPEN que creó la cola. Se llevan a cabo comprobaciones de autorización con respecto al identificador de usuario (o un identificador de usuario alternativo si se especificó MQOO_ALTERNATE_USER_AUTHORITY) que se ha utilizado para validar la llamada MQOPEN correspondiente.
- Se pueden suprimir de la misma manera que una cola normal.

Usos de las colas dinámicas

Puede utilizar colas dinámicas para:

- Aplicaciones que no requieren que se guarden colas después de que la aplicación haya finalizado.
- Aplicaciones que requieren que las respuestas a los mensajes para ser procesados por otra aplicación. Estas aplicaciones pueden crear dinámicamente una cola de respuestas abriendo una cola modelo. Por ejemplo, una aplicación de cliente puede:
  1. Crear una cola dinámica.
  2. Suministrar el nombre en el campo **ReplyToQ** de la estructura de descriptor de mensaje del mensaje de solicitud.
  3. Colocar la petición en una cola que un servidor está procesando.

A continuación, el servidor puede colocar el mensaje de respuesta en la cola de respuestas. Finalmente, el cliente puede procesar la respuesta, y cerrar la cola de respuestas con la opción de supresión.

Consideraciones al utilizar colas dinámicas

Tenga en cuenta las cuestiones siguientes al utilizar colas dinámicas:

- En un modelo cliente-servidor, cada cliente debe crear y utilizar su propia dinámica cola de respuestas. Si una cola dinámica de respuestas se comparte entre más de un cliente, la supresión de la cola de respuestas podría demorarse debido a que hay actividad no confirmada en la cola

o porque otro cliente está utilizando la cola. Además, la cola puede marcarse como suprimida lógicamente e inaccesible para las peticiones posteriores de la API (distintas de MQCLOSE).

- Si el entorno de aplicación requiere que se compartan colas dinámicas entre aplicaciones, asegúrese de que la cola sólo se cierra (con la opción de supresión) cuando toda la actividad que hay en ella se haya confirmado. Esta acción la debe realizar el último usuario. De este modo se asegura que la supresión de la cola no se retrasa, y se minimiza el período en el que la cola es inaccesible porque se ha marcado como suprimida lógicamente.

Colas modelo

Una *cola modelo* es una plantilla de definición de cola que se utiliza al crear una cola dinámica.

Puede crear una cola local dinámicamente desde un programa IBM MQ, nombrando la cola modelo que desea utilizar como plantilla para los atributos de cola. En ese momento puede cambiar algunos atributos de la nueva cola. Pero no puede cambiar **DefinitionType**. Si, por ejemplo, necesita una cola persistente, seleccione una cola de modelo con el tipo de definición establecido en permanente. Algunas aplicaciones conversacionales pueden utilizar colas dinámicas para albergar las respuestas a sus consultas porque probablemente no necesitan mantener estas colas después de que hayan procesado las respuestas.

Debe especificar el nombre de una cola modelo en el *descriptor de objeto* (MQOD) de la llamada MQOPEN. Utilizando los atributos de la cola modelo, el gestor de colas crea de forma automática y dinámica una cola local.

El usuario puede especificar un nombre (completo) para la cola dinámica o la raíz de un nombre (por ejemplo, ABC) y dejar que el gestor de colas le añada una parte exclusiva o bien dejar que el gestor de colas asigne automáticamente un nombre exclusivo completo. Si el gestor de colas asigna el nombre, lo coloca en la estructura MQOD.

No puede emitir una llamada MQPUT1 directamente a una cola modelo, pero puede emitir una llamada MQPUT1 a la cola dinámica que se ha creado abriendo una cola modelo.

MQSET y MQINQ no se pueden emitir contra una cola de modelo. La apertura de una cola de modelo con MQOO_INQUIRE o MQOO_SET, resulta en posteriores llamadas de MQINQ y MQSET realizadas contra la cola creada dinámicamente.

Los atributos de una cola modelo son un subconjunto de los de una cola local. Para obtener una descripción más completa, consulte Atributos para colas.

Colas utilizadas para fines específicos por IBM MQ

IBM MQ utiliza algunas colas locales para fines específicos relacionados con su funcionamiento.

Debe definir estas colas para que IBM MQ las pueda utilizar.

Colas de inicio

Las colas de inicio son colas que se utilizan para la activación. Un gestor de colas coloca un mensaje desencadenante en una cola de inicio cuando se produce un suceso desencadenante. Un suceso desencadenante es una combinación lógica de condiciones detectada por un gestor de colas. Por ejemplo, puede generarse un suceso desencadenante cuando el número de mensajes de una cola alcanza una profundidad predefinida. Este suceso hace que el gestor de colas transfiera un mensaje desencadenante a una cola de inicio especificada. Este mensaje desencadenante lo recupera un *supervisor desencadenante*, una aplicación especial que supervisa una cola de inicio. El supervisor desencadenante inicia entonces el programa de aplicación que se haya especificado en el mensaje desencadenante.

Si un gestor de colas ha de utilizar el mecanismo de activación, debe definirse al menos una cola de inicio para ese gestor de colas. Consulte Gestión de objetos para el mecanismo de activación, runmqtrm e Inicio de aplicaciones IBM MQ utilizando desencadenantes

Colas de transmisión

Las colas de transmisión son colas que almacenan temporalmente mensajes destinados a un gestor de colas remoto. Debe definir al menos una cola de transmisión para cada gestor de colas remoto al

que el gestor de colas local va a enviar mensajes directamente. Estas colas también se utilizan en la administración remota; consulte Administración remota desde un gestor de colas local. Para obtener información sobre el uso de colas de transmisión en colas distribuidas, consulte Técnicas de gestión de colas distribuidas de IBM MQ.

Cada gestor de colas puede tener una cola de transmisión predeterminada. Si un gestor de colas que no forme parte de un clúster transfiere un mensaje a una cola remota, la acción predeterminada es utilizar la cola de transmisión predeterminada. Si existe una cola de transmisión que tenga el mismo nombre que el gestor de colas de destino, el mensaje se coloca en dicha cola de transmisión . Si existe una definición de alias de gestor de colas, en la que el parámetro **RQMNAME** coincida con el gestor de colas de destino, y se especifica el parámetro **XMITQ**, el mensaje se coloca en la cola de transmisión a la que **XMITQ** ponga nombre. Si no existe el parámetro **XMITQ**, el mensaje se coloca en la cola local que se indique en el mensaje.

Colas de transmisión de clúster

Cada gestor de colas de un clúster tiene una cola de transmisión de clúster denominada SYSTEM.CLUSTER.TRANSMIT.QUEUE, y una cola de transmisión de clúster modelo, SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. Las definiciones de estas colas se crean, de forma predeterminada, cuando se define un gestor de colas. Si el atributo **DEFCLXQ** del gestor de colas se establece en CHANNEL, se crea, de forma automática, una cola de transmisión de clúster dinámico permanente para cada canal de clúster emisor que se cree. Las colas se denominan SYSTEM.CLUSTER.TRANSMIT. *ChannelName*. También puede definir las colas de transmisión de clúster de clúster manualmente.

Un gestor de colas que forme parte del clúster envía los mensajes de una de estas colas a otros gestores de colas que pertenezcan al mismo clúster.

Durante la resolución de nombres, una cola de transmisión de clúster tiene prioridad sobre la cola de transmisión predeterminada, y una cola de transmisión de clúster específica tiene prioridad sobre SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Colas de mensajes no entregados

Una cola de mensajes no entregados es una cola que almacena mensajes que no pueden direccionarse a sus destinos correctos. Un mensaje no se puede direccionar cuando, por ejemplo, la cola de destino está llena. La cola de mensajes no entregados que se suministra con MQSeries se llama SYSTEM.DEAD.LETTER.QUEUE.

Para las colas distribuidas, debe definir una cola de mensajes no entregados en cada gestor de colas implicado.

Colas de mandatos

La cola de mandatos, SYSTEM.ADMIN.COMMAND.QUEUE, es una cola local a la que las aplicaciones debidamente autorizadas pueden enviar mandatos MQSC para procesarlos. A continuación, estos mandatos los recupera un componente IBM MQ denominado servidor de mandatos. El servidor de mandatos valida los mandatos, pasa los que son válidos al gestor de colas para que los procese y devuelve las respuestas a la cola de respuestas apropiada.

Al crear un gestor de colas, se crea automáticamente una cola de mandatos para el mismo.

Colas de respuestas

Cuando una aplicación envía un mensaje de solicitud, la aplicación que lo recibe puede devolver un mensaje de respuesta a la aplicación emisora. Este mensaje se transfiere a una cola, llamada cola de respuestas, que suele ser una cola local de la aplicación emisora. El nombre de la cola de respuestas lo especifica la aplicación emisora como parte del descriptor del mensaje.

Colas de sucesos

Los sucesos de instrumentación, que pueden utilizarse para supervisar gestores de colas, independientemente de las aplicaciones MQI.

Cuando se produce un suceso de instrumentación, el gestor de colas coloca un mensaje de suceso en una cola de sucesos. Este mensaje puede leerlo entonces una aplicación de supervisión, que puede informar a un administrador o iniciar alguna acción correctora si el suceso indica un problema.

Nota: Los sucesos desencadenantes son diferentes de los sucesos de instrumentación. Los sucesos desencadenantes no los causan las mismas condiciones, y no generan mensajes de sucesos.

Para obtener más información sobre los sucesos de instrumentación, consulte Sucesos de instrumentación.

Gestores de colas

Una introducción a los *gestores de colas* y a los servicios en colas que proporcionan a las aplicaciones.

Un programa debe tener una conexión con un gestor de colas para poder utilizar los servicios de ese gestor de colas. Un programa puede realizar esta conexión explícitamente (utilizando la llamada MQCONN o MQCONNX), o la conexión se puede realizar implícitamente (este depende de la plataforma y el entorno en el que se ejecuta el programa).

Un gestor de colas de IBM MQ garantiza las acciones siguientes:

- Los atributos de objeto se modifican conforme a los mandatos recibidos.
- Los sucesos especiales, tales como sucesos desencadenantes o sucesos de instrumentación, se generan cuando se cumplen las condiciones adecuadas.
- Los mensajes se colocan en la cola correcta, según lo solicitado por la aplicación que ha efectuado la llamada MQPUT. En caso de que esto no sea posible, se notifica a la aplicación y se emite un código de razón apropiado.

Cada cola pertenece a un solo gestor de colas y se dice que es una *cola local* respecto a ese gestor de colas. El gestor de colas al que está conectada una aplicación es el *gestor de colas local* respecto a esa aplicación. Para la aplicación, las colas que pertenecen a su gestor de colas local son colas locales.

Una *cola remota* es una cola que pertenece a otro gestor de colas. Un *gestor de colas remoto* es cualquier gestor de colas que no sea el gestor de colas local. Un gestor de colas remoto puede estar en una máquina remota de la red o puede estar en la misma máquina que el gestor de colas local. IBM MQ ofrece soporte a varios gestores de colas en la misma máquina.

En algunas llamadas MQI, se puede utilizar un objeto gestor de colas. Por ejemplo, puede efectuar consultas sobre los atributos del objeto gestor de colas utilizando la llamada MQI MQINQ.

Atributos de los gestores de colas

Asociados a cada gestor de colas hay un conjunto de atributos (o propiedades) que definen sus características.Algunos de los atributos de un gestor de colas se arreglan cuando éste se crea; puede cambiar otros utilizando los mandatos de IBM MQ. Puede realizar consultas sobre los valores de todos los atributos, excepto los que se utiliza para el cifrado de TLS (Transport Layer Security), mediante la llamada MQINQ.

Los atributos fijos incluyen los siguientes:

- El nombre del gestor de colas
- La plataforma en la que se ejecuta el gestor de colas (por ejemplo, Windows)
- El nivel de los mandatos de control del sistema al que el gestor de colas da soporte
- La prioridad máxima que se puede asignar a los mensajes procesados por el gestor de colas
- El nombre de la cola a la que los programas pueden enviar mandatos de IBM MQ
- La longitud máxima de mensajes que el gestor de colas puede procesar z/OS (solo arreglado en IBM MQ for z/OS)
- Si el gestor de colas ofrece soporte al recurso de puntos de sincronismo cuando los programas transfieren y obtienen mensajes

Los atributos *modificables* incluyen los siguientes:

- Una descripción de texto del gestor de colas

- El identificador del juego de caracteres que el gestor de colas utiliza para las series de caracteres cuando procesa llamadas MQI
- El intervalo de tiempo que el gestor de colas utiliza para restringir el número de mensajes desencadenantes
- z/OS El intervalo de tiempo que el gestor de colas utiliza para determinar la frecuencia con la que se deben explorar en las colas los mensajes que han caducado (solo IBM MQ for z/OS)
- El nombre de la cola de mensajes no entregados del gestor de colas (mensaje no entregado)
- El nombre de la cola de transmisión predeterminada del gestor de colas
- El número máximo de manejadores abiertos para cualquier conexión individual
- La habilitación e inhabilitación de diversas categorías de informes de sucesos
- El número máximo de mensajes sin confirmar dentro de una unidad de trabajo

Gestores de colas y gestión de carga de trabajo

Puede configurar un clúster de gestores de colas que tenga más de una definición para la misma cola (por ejemplo, los gestores de colas en el clúster podrían ser clones de entre sí). Los mensajes de una cola determinada los puede manejar cualquier gestor de colas que aloja una instancia de la cola. Un algoritmo de gestión de carga de trabajo decide qué gestor de colas maneja el mensaje y reparte la carga de trabajo entre los gestores de colas; consulte El algoritmo de gestión de carga de trabajo de clúster para obtener más información.

Canales

Un *canal* es un enlace de comunicación lógico, utilizado por gestores de colas distribuidos, entre un servidor de IBM MQ MQI client y un servidor de IBM MQ o entre dos servidores de IBM MQ.

Los canales se utilizan para mover los mensajes desde un gestor de colas a otro y protegen las aplicaciones de los protocolos de comunicaciones subyacentes. Los gestores de colas pueden estar en sistemas diferentes de la misma plataforma o de plataformas distintas. Los mensajes que se envían pueden tener su origen en muchas ubicaciones:

- Programas de aplicación escritos por el usuario que transfieren datos de un nodo a otro.
- Aplicaciones de administración escritas por usuario que utilizan mandatos PCF o MQAI.
- IBM MQ Explorer.
- Gestores de colas que envían mensajes de sucesos de instrumentación a otro gestor de colas.
- Gestores de colas que envían mandatos de administración remotos a otro gestor de colas. Por ejemplo, utilizando los mandatos MQSC o la administrative REST API.

Un canal tiene dos definiciones: una en cada extremo de la conexión. Para que los gestores de colas se comuniquen entre sí, debe definir un objeto canal en el gestor de colas que va a enviar los mensajes y otro complementario en el gestor de colas que va a recibirlos. Debe utilizarse el mismo *nombre de canal* en cada extremo de la conexión y el *tipo de canal* utilizado debe ser compatible.

Hay tres categorías de canal en IBM MQ, con diferentes tipos de canal dentro de estas categorías:

- Canales de mensaje, que son unidireccionales y transfieren mensajes de un gestor de colas a otro.
- Canales MQI, que son bidireccionales y transfieren llamadas MQI de un IBM MQ MQI client a un gestor de colas y respuestas de un gestor de colas a un cliente de IBM MQ.
- Canales AMQP, que son bidireccionales y conectan un cliente AMQP a un gestor de colas en una máquina servidor. IBM MQ utiliza canales AMQP para transferir llamadas y respuestas AMQP entre aplicaciones AMQP y gestores de colas

Canales de mensajes

El objetivo de un canal de mensajes es transferir mensajes de un gestor de colas a otro. Los canales de mensajes no son necesarios para el entorno de cliente-servidor.

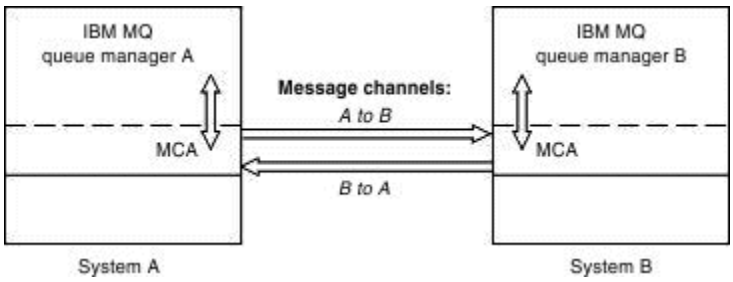


Figura 2. Canales de mensajes entre dos gestores de colas

Un canal de mensajes es un enlace unidireccional. Si desea que un gestor de colas remoto responda a los mensajes enviados por un gestor local, debe configurar un segundo canal para que devuelva las respuestas al gestor de colas local.

Un canal de mensajes conecta dos gestores de colas utilizando los *agentes de canal de mensajes* (MCA). Hay un agente de canal de mensajes en cada extremo de un canal. Puede permitir que MCA transfiera los mensajes utilizando varias hebras. Este proceso se conoce como *canalización*. El proceso de canalización permite que MCA transfiera los mensajes de forma más eficaz y mejora el rendimiento de los canales. Para obtener más información sobre la interconexión, consulte Atributos de los canales.

Para obtener más información sobre los canales, consulte la sección Llamadas de salida de canal y estructuras de datos y “Componentes de la gestión de colas distribuidas” en la página 47.

Canales MQI

Un canal de interfaz de cola de mensajes (MQI) conecta un IBM MQ MQI client a un gestor de colas en una máquina servidor y se establece cuando se emite una llamada `MQCONN` o `MQCONNX` desde una aplicación de IBM MQ MQI client.

Es un enlace bidireccional y se utiliza para la transferencia de llamadas MQI y respuestas únicamente, incluyendo llamadas `MQPUT` que contienen datos de mensajes y llamadas `MQGET` que hacen que se devuelvan los datos de mensajes. Hay diferentes formas de crear y utilizar definiciones de canal (consulte Definición de canales MQI).

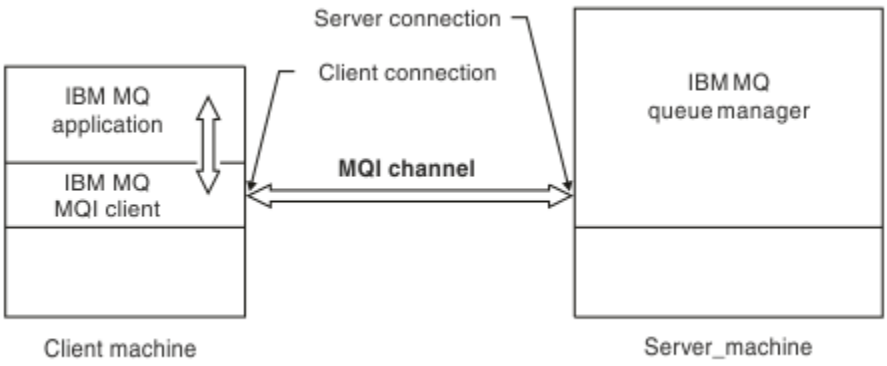


Figura 3. Conexión con el cliente y con el servidor en un canal MQI

z/OS Un canal MQI se puede utilizar para conectar un cliente con un único gestor de colas o con un gestor de colas que forma parte de un grupo de compartición de colas (consulte Conexión de un cliente a un grupo de compartición de colas).

Hay dos tipos de canal para definiciones de canal MQI. Definen el canal MQI bidireccional.

Canal de conexión con el cliente
Este tipo es para IBM MQ MQI client.

Canal de conexión con el servidor
Este tipo es para el servidor que ejecuta el gestor de colas, con el que la aplicación de IBM MQ, que se ejecuta en un entorno de IBM MQ MQI client, se va a comunicar.

Canales AMQP

Sólo hay un tipo de canal AMQP.

Utilice el canal para conectar una aplicación de mensajería AMQP con un gestor de colas, lo cual permite que la aplicación intercambie mensajes con aplicaciones IBM MQ. Un canal AMQP le permite desarrollar una aplicación utilizando MQ Light y desplegarlo después como una aplicación de empresa, beneficiándose de los recursos a nivel de empresa que proporciona IBM MQ.

Canales de conexión de cliente

Los *canales de conexión de cliente* son objetos que proporcionan una vía de comunicación de un IBM MQ MQI client a un gestor de colas.

Los canales de conexión de cliente se utilizan en la gestión de colas distribuidas para trasladar mensajes entre un gestor de colas y un cliente. Permiten que las aplicaciones no tengan que preocuparse por los protocolos de comunicaciones subyacentes. El cliente puede existir en la misma plataforma que el gestor de colas o en una diferente.

Definiciones de canal

Consulte “Definiciones de canal” en la página 32 si desea obtener descripciones de cada tipo de canal.

Conceptos relacionados

“Gestión de colas distribuidas y clústeres” en la página 43
La gestión de colas distribuidas hace referencia al envío de mensajes desde un gestor de colas a otro. El gestor de colas receptor puede estar en la misma máquina o en otra; puede estar próximo o en el otro lado del mundo. Puede ejecutarse en la misma plataforma que el gestor de colas local o puede estar en cualquiera de las plataformas soportadas por IBM MQ. Puede definir manualmente todas las conexiones en un entorno de colas distribuidas o puede crear un clúster y dejar que IBM MQ defina automáticamente gran parte del detalle de conexión.

Visión general de la Interfaz de cola de mensajes

Tareas relacionadas

Administración de objetos de IBM MQ remotos

Detención de canales MQI

Configurar conexiones entre el servidor y el cliente

Referencia relacionada

Llamadas de salida de canal y estructuras de datos

“Comunicaciones” en la página 36
IBM MQ MQI clients utiliza canales MQI para comunicarse con el servidor.

Definiciones de canal

Tablas que describen los distintos tipos de canales de mensajes y canales MQI que IBM MQ utiliza.

Cuando se hace referencia a canales de mensajes, la palabra canal se utiliza a menudo como sinónimo de definición de canal. Normalmente, el contexto permite discernir si se habla de un canal completo, con dos extremos, o de una definición de canal, con un único extremo.

Canales de mensajes

El canal de mensajes puede ser de los siguientes tipos:

Tipo de definición de canal de mensajes	Descripción
Emisor	Un canal emisor es un canal de mensajes que utiliza el gestor de colas para enviar mensajes a otros gestores. Para enviar mensajes utilizando un canal emisor, debe crear también un canal receptor en el otro gestor de colas con el mismo nombre que el canal emisor. También puede utilizar canales emisores con canales peticionarios si implementa un mecanismo de "devolución de llamada".
Servidor	Un canal servidor es un canal de mensajes que el gestor de colas utiliza para enviar mensajes a los otros gestores. Para enviar mensajes utilizando un canal emisor, debe crear también un canal receptor en el otro gestor de colas con el mismo nombre que el canal servidor. También puede utilizar los canales servidores con canales peticionarios. En ese caso, la definición de canal peticionario en el otro extremo del canal solicita que se inicie la definición de canal servidor. El servidor envía mensajes al peticionario. El servidor también puede iniciar la comunicación, siempre y cuando sepa el nombre de conexión del canal asociado.
Receptor	Un canal receptor es un canal de mensajes que utiliza el gestor de colas para recibir mensajes de otros gestores de colas. Para recibir mensajes utilizando un canal receptor, debe crear también un canal emisor o un canal servidor en el otro gestor de colas con el mismo nombre que el canal receptor.
Peticionario	Un canal peticionario es un canal de mensajes que utiliza el gestor de colas para recibir mensajes de otros gestores de colas. Un canal peticionario puede solicitar que se inicie el canal de socio definido en el extremo remoto. Si el canal de socio es un canal de servidor, el canal de servidor acepta la solicitud de inicio y empieza a enviar mensajes, desde la cola de transmisión identificada en la definición del canal de servidor, al canal de solicitante. Si el canal de socio es un canal emisor, el canal emisor acepta la solicitud de inicio pero, a continuación, cierra la conexión con el solicitante. A continuación, el canal emisor se inicia, negocia una sesión con el canal solicitante de socio y comienza a enviar mensajes desde la cola de transmisión identificada en la definición del canal emisor. Este último caso proporciona esencialmente un mecanismo de devolución de llamada en el que el canal solicitante solicita al canal emisor que devuelva la llamada.

Tipo de definición de canal de mensajes	Descripción
Clúster emisor	La definición de canal de clúster emisor (CLUSSDR) define el extremo emisor de un canal en el que un gestor de colas de clúster puede enviar información de clúster a uno de los depósitos completos. El canal de clúster emisor se utiliza para notificar el depósito de cualquier cambio que se realice en el estado del gestor de colas, por ejemplo la adición o supresión de una cola. Se utiliza también para transmitir mensajes. Los gestores de colas de depósito completo propiamente tienen canales de clúster emisor que se apuntan uno al otro. Se utilizan para comunicarse entre sí los cambios de estado del clúster. No tiene mucha importancia saber qué depósito completo apunta la definición de canal CLUSSDR del gestor de colas. Tras el contacto inicial, se definirán automáticamente más objetos de gestor de colas de clúster, según sea necesario, para que el gestor de colas pueda enviar la información de clúster a cada depósito completo, y los mensajes a cada gestor de colas.
Clúster receptor	La definición de canal de clúster receptor (CLUSRCVR) define el extremo receptor de un canal en el que un gestor de colas de clúster puede recibir mensajes de otros gestores de colas del clúster. Un canal de clúster receptor también puede transportar información sobre el clúster destinada para el repositorio. Al definir el canal de clúster receptor, el gestor de colas indica a los otros gestores de colas de clúster que se encuentra disponible para recibir mensajes. Necesita como mínimo un canal de clúster receptor para gestor de colas de clúster.

Debe definir los dos extremos de todos los canales para tener una definición de canal para cada extremo del canal. Los dos extremos del canal deben ser compatibles.

Puede tener las siguientes combinaciones de definiciones de canal:

- Emisor-Receptor
- Servidor-Receptor
- Peticionario-Servidor
- Peticionario-Servidor (devolución de llamada)
- Emisor del clúster-Receptor del clúster

Agentes de canal de mensajes

Todas las definiciones de canal que cree pertenecen a un gestor de colas determinado. Un gestor de colas puede tener varios canales del mismo o de diferentes tipos. En cada extremo del canal hay un programa, el agente de canal de mensajes (MCA). En cada extremo del canal, el MCA de llamada coge los mensajes de la cola de transmisión y los envía a través del canal. En el otro extremo del canal, el MCA de respuesta recibe los mensajes y los entrega al gestor de colas remoto.

Un MCA de llamada puede estar asociado con un canal emisor, servidor o peticionario. Un MCA de respuesta puede estar asociado con cualquier tipo de canal de mensajes.

IBM MQ admite las siguientes combinaciones de tipos de canal en los dos extremos de una conexión:

De llamada		Dirección del flujo de mensajes	De respuesta	
Tipo de canal	**¿Es necesario un escucha?**		**¿Es necesario un escucha?**	**Tipo de canal**
Emisor	No	De canal de llamada a canal de respuesta	Sí	Receptor
Servidor	No	De canal de llamada a canal de respuesta	Sí	Receptor
Servidor	No	De canal de llamada a canal de respuesta	Sí	Peticionario
Peticionario	No	De canal de respuesta a canal de llamada	Sí	Servidor
Peticionario	Sí	De canal de respuesta a canal de llamada	Sí	Emisor

Canales MQI

Los canales MQI pueden ser de los siguientes tipos:

Tipo de canal MQI	Descripción
Conexión de servidor	Un canal de conexión con el servidor es un canal MQI bidireccional que se utiliza para conectar un cliente de IBM MQ con un servidor de IBM MQ. El canal de conexión con el servidor es el extremo del servidor del canal.
Conexión de cliente	Un canal de conexión con el cliente es un canal MQI bidireccional que se utiliza para conectar un cliente de IBM MQ con un servidor de IBM MQ. IBM MQ Explorertambién utiliza conexiones de cliente para conectarse a gestores de colas remotos. El canal de conexión de cliente es el extremo del cliente del canal. Cuando cree el canal de conexión con el cliente, se creará un archivo en el sistema que alojará el gestor de colas. A continuación, debe copiar el archivo de conexión de cliente en el sistema cliente de IBM MQ.

Multi ***Soporte de varias hebras-interconexiones***

Opcionalmente, puede permitir que un agente de canal de mensajes (MCA) transfiera mensajes utilizando varias hebras. Este proceso, denominado *canalización*, permite al MCA transferir mensajes de forma más eficiente, con menos estados de espera, lo que mejora el rendimiento del canal. Cada MCA está limitado a un máximo de dos hebras.

Puede controlar la interconexión con el parámetro *PipeLineLength* en el archivo qm.ini . Este parámetro se añade a la stanza Channels.

Nota: El canalizado solo es efectivo para los canales TCP/IP.

Cuando se utiliza la interconexión, los gestores de colas en ambos extremos del canal deben estar configurados para tener una longitud de *PipeLine* mayor que 1.

Consideraciones sobre la salida de canal

La interconexión puede hacer que algunos programas de salida fallen, porque:

- Es posible que las salidas no se llamen en serie.
- Las salidas se pueden llamar alternativamente desde diferentes hebras.

Compruebe el diseño de los programas de salida antes de utilizar la interconexión:

- Las salidas deben ser reentrantes en todas las etapas de su ejecución.
- Cuando utilice llamadas MQI, recuerde que no puede utilizar el mismo descriptor de contexto MQI cuando se invoca la salida desde hebras diferentes.

Considere una salida de mensajes que abra una cola y utilice su manejador para llamadas MQPUT en todas las invocaciones posteriores de la salida. Esto falla en modalidad de interconexión porque se llama a la salida desde distintas hebras. Para evitar esta anomalía, mantenga un descriptor de contexto de cola para cada hebra y compruebe el identificador de hebra cada vez que se invoque la salida.

Comunicaciones

IBM MQ MQI clients utiliza canales MQI para comunicarse con el servidor.

Se debe crear una definición de canal en los extremos de IBM MQ MQI client y del servidor de la conexión. Cómo crear definiciones de canal se explica en Definición de canales MQI.

Los protocolos de transmisión posibles se muestran en la tabla siguiente:

Tabla 1. Protocolos de transmisión para canales MQI

Plataforma de cliente	LU6.2	TCP/IP	NetBIOS	SPX
IBM i		Sí		
Linux AIX Sistemas AIX and Linux	Sí [1]	Sí		
Windows	Sí	Sí	Sí	Sí

Nota:

1. Linux LU6.2 no está soportado en las plataformas siguientes:
   - Linux (plataforma POWER)
   - Linux (plataforma x86-64)
   - Linux (plataforma zSeries s390x)

En Protocolos de transmisión - combinación de plataformas IBM MQ MQI client y de servidor se muestran las posibles combinaciones de plataformas IBM MQ MQI client y de servidor, utilizando estos protocolos de transmisión.

Una aplicación de IBM MQ en un IBM MQ MQI client puede utilizar todas las llamadas MQI del mismo modo que cuando el gestor de colas es local. **MQCONN** o **MQCONNX** asocia la aplicación de IBM MQ con el gestor de colas seleccionado, creando un *descriptor de conexión*. A continuación, el gestor de colas conectado procesa otras llamadas que utilizan ese manejador de conexión. La comunicación de IBM MQ MQI client requiere una conexión activa entre el cliente y el servidor, a diferencia de la comunicación entre gestores de colas, que es independiente de la conexión e independiente del tiempo.

El protocolo de transmisión se especifica utilizando la definición de canal y no afecta a la aplicación. Por ejemplo, una aplicación de Windows se puede conectar a un gestor de colas a través de TCP/IP y a otro gestor de colas a través de NetBIOS.

Consideraciones sobre el rendimiento

El protocolo de transmisión que utilice puede afectar al rendimiento del sistema cliente y servidor de IBM MQ. En determinadas situaciones en las que la transmisión es lenta, puede utilizar la compresión de canal de IBM MQ .

Denominación de objetos de IBM MQ

El convenio de denominación adoptado para los objetos de IBM MQ depende del objeto. El nombre de las máquinas y los ID de usuario que utiliza con IBM MQ también están sujetos a algunas restricciones de denominación.

Cada instancia de un gestor de colas se reconoce por su nombre. Este nombre debe ser exclusivo dentro de la red de gestores de colas interconectados, de modo que un gestor de colas pueda identificar inequívocamente el gestor de colas de destino al que se envía un determinado mensaje.

En los demás tipos de objetos, cada objeto tiene un nombre asociado con el que puede hacerse referencia al mismo. Estos nombres deben ser exclusivos dentro de un gestor de colas y de un tipo de objeto. Por ejemplo, puede haber una cola y un proceso con el mismo nombre, pero no puede haber dos colas con el mismo nombre.

En IBM MQ, los nombres pueden tener un máximo de 48 caracteres, con la excepción de los *canales* que tienen un máximo de 20 caracteres. Para obtener más información sobre la denominación de objetos IBM MQ, consulte “Reglas de denominación de objetos de IBM MQ” en la página 37.

El nombre de las máquinas y los ID de usuario que utiliza con IBM MQ también están sujetos a algunas restricciones de denominación:

- Asegúrese de que el nombre de la máquina no contiene espacios. IBM MQ no da soporte a nombres de máquina que incluyan espacios. Si instala IBM MQ en una máquina de este tipo, no puede crear ningún gestor de colas.
- En el caso de las autorizaciones de IBM MQ, los nombres de los ID de usuario y los grupos no deben tener más de 20 caracteres (no se permiten espacios).
- Windows Un servidor de IBM MQ for Windows no da soporte a la conexión de un IBM MQ MQI client si el cliente se ejecuta con un ID de usuario que contiene el carácter @, por ejemplo, abc@d.

Conceptos relacionados

“Nombres de archivo de IBM MQ” en la página 40
Cada gestor de colas de IBM MQ, cola, definición de proceso, lista de nombres, canal, canal de conexión cliente, escucha, servicio y objeto de información de autenticación se representa mediante un archivo. Puesto que los nombres de objetos no son necesariamente nombres de archivo válidos, el gestor de colas convierte el nombre de objeto en un nombre de archivo válido cuando es necesario.

Referencia relacionada

“Reglas de denominación de objetos de IBM MQ” en la página 37
Los nombres de objeto de IBM MQ tienen longitudes máximas y distinguen entre mayúsculas y minúsculas. No todos los caracteres están soportados para cada tipo de objeto y muchos objetos tienen reglas relativas a la exclusividad de nombres.

Reglas de denominación de objetos de IBM MQ

Los nombres de objeto de IBM MQ tienen longitudes máximas y distinguen entre mayúsculas y minúsculas. No todos los caracteres están soportados para cada tipo de objeto y muchos objetos tienen reglas relativas a la exclusividad de nombres.

Hay muchos tipos diferentes de objeto IBM MQ y los objetos de cada tipo pueden tener todos el mismo nombre porque existen en espacios de nombres de objetos separados: por ejemplo, una cola local y un canal emisor pueden tener ambos el mismo nombre. Sin embargo, un objeto no puede tener el mismo nombre que otro objeto en el mismo espacio de nombres: por ejemplo, una cola local no puede tener el mismo nombre que una cola de modelo y un canal emisor no puede tener el mismo nombre que un canal receptor.

Los siguientes objetos de IBM MQ existen en espacios de nombres de objetos separados:

- Información de autenticación
- Canal
- Canal de cliente
- Escucha
- Lista de nombres
- Proceso
- Cola
- Servicio
- Clase de almacenamiento
- Suscripción
- Tema

Longitud de caracteres de nombres de objeto

En general, los nombres de objeto de IBM MQ pueden tener hasta 48 caracteres de longitud. Esta regla se aplica a los siguientes objetos:

- Información de autenticación
- Clúster
- Escucha
- Lista de nombres
- Definición de proceso
- Cola
- Gestor de colas
- Servicio
- Suscripción
- Tema

Existen restricciones:

1. z/OS En los sistemas z/OS, los gestores de colas deben tener un máximo de 4 caracteres y deben estar en mayúsculas y sólo caracteres numéricos.
2. La longitud máximo de los nombres de objeto de canal y los nombres de canal de conexión con el cliente es 20 caracteres. Consulte Definición de los canales para obtener más información sobre los canales.
3. Las series de tema pueden tener un máximo de 10240 bytes. Todos los nombres de objeto de IBM MQ distinguen entre mayúsculas y minúsculas.
4. Los nombres de suscripción pueden tener un máximo de 10240 bytes y pueden contener espacios.
5. La longitud máxima de los nombres de clase de almacenamiento es 8 caracteres.
6. La longitud máxima de los nombres de estructura de CF es 12 caracteres .

Caracteres en nombres de objeto

Los caracteres válidos para los nombres de objeto de IBM MQ son:

Caracteres	Restricciones
De la A a la Z en mayúsculas	• Ninguna

Caracteres	Restricciones
De la a a la z en minúsculas	• En los scripts MQSC, los nombres con caracteres en minúsculas deben ir entre comillas simples. Esto impide que los caracteres en minúsculas se convertirán en mayúsculas. • Los sistemas que utilizan EBCDIC Katakana no pueden utilizar los caracteres de a- z en minúsculas en los nombres de objeto. • z/OS Podrían haber restricciones al utilizar caracteres en minúsculas en los sistemas z/OS; por ejemplo, los nombres de gestor de colas no pueden contener caracteres en minúsculas. • IBM i En los sistemas IBM i, cuando se utilizan mandatos CL, los nombres con caracteres en minúsculas deben ir entre comillas simples. Esto impide que los caracteres en minúsculas se convertirán en mayúsculas.
Numéricos de 0 a 9	• Ninguna
Punto (.)	• Ninguna
Subrayado (_)	• Multi Ninguna • z/OS Evite utilizar nombres con caracteres de subrayado iniciales o finales porque los paneles de operaciones y de control de IBM MQ for z/OS no pueden manejarlos.
Barra inclinada (/)	• Windows En los sistemas Windows, el primer carácter de un nombre de gestor de colas no puede ser una barra inclinada. • IBM i En los sistemas IBM i, cuando se utilizan mandatos CL, los nombres que contienen una barra inclinada deben ir entre comillas simples. • z/OS Ninguna
Signo de porcentaje (%)	• ALW Ninguna • z/OS Si utiliza RACF como gestor de seguridad externa para IBM MQ for z/OS, no utilice % en nombres de objeto porque los nombres no están incluidos en las comprobaciones de seguridad cuando se utilizan perfiles genéricos de RACF. • IBM i En los sistemas IBM i, cuando se utilizan mandatos CL, los nombre que contienen un símbolo de porcentaje deben ir entre comillas simples.

Existen algunas reglas generales en relación con los caracteres en los nombres de objeto:

1. No se permiten los espacios en blanco iniciales o intercalados.
2. No se permiten caracteres de idiomas nacionales.
3. Cualquier nombre que sea menor que la longitud total del campo puede rellenarse por la derecha con espacios en blanco. Todos los nombres abreviados devueltos por el gestor de colas se rellenan siempre con espacios en blanco a la derecha.

Nombres de cola

El nombre de una cola tiene dos partes:

- El nombre de un gestor de colas
- El nombre local de la cola como es conocido para dicho gestor de colas

Cada parte del nombre de cola tiene 48 caracteres de longitud.

Para hacer referencia a una cola local, puede omitir el nombre del gestor de colas (sustituyéndolo por caracteres en blanco o utilizando un carácter nulo inicial). Sin embargo, todos los nombres de cola devueltos a un programa por IBM MQ contienen el nombre del gestor de colas.

z/OS Una cola compartida, accesible a cualquier gestor de colas en su grupo de compartición de colas, no puede tener el mismo nombre que cualquier cola local no compartida en el mismo grupo de compartición de colas. Esta restricción evita la posibilidad de que una aplicación abra por error una cola compartida cuando se pretendía abrir una cola local, o viceversa. Las colas compartidas y los grupos de compartición de colas solo están disponibles en IBM MQ for z/OS.

Para hacer referencia a una cola remota, un programa debe incluir el nombre del gestor de colas en el nombre de cola completo, o debe haber una definición local de la cola remota.

Cuando una aplicación utiliza un nombre de cola, dicho nombre puede ser el nombre de una cola local (o un alias de una) o el nombre de una definición local de una cola remota, pero la aplicación no necesita saber cual, a menos que deba obtener un mensaje de la cola (cuando la cola debe ser local). Cuando la aplicación abre el objeto de cola, la llamada de MQOPEN realiza una función de resolución de nombre para determinar en qué cola realizar operaciones posteriores. La importancia de esto es que la aplicación no tiene ninguna dependencia incorporada en las colas particulares que se están definiendo en ubicaciones determinadas en una red de gestores de colas. Por lo tanto, si un administrador del sistema reubica las colas en la red y cambia sus definiciones, no es necesario cambiar las aplicaciones que utilizan estas colas.

Nombres de objeto reservados

Los nombres de objeto que empiezan por SYSTEM. están reservados para los objetos definidos por el gestor de colas. Puede utilizar los mandatos **Alter**, **Define** y **Replace** para cambiar estas definiciones de objeto para adaptarse a la instalación. Los nombres definidos para IBM MQ se listan de forma completa en Nombres de cola.

z/OS En IBM MQ for z/OS, el nombre de estructura de aplicación de recurso de acoplamiento CSQSYSAPPL está reservado.

Conceptos relacionados

Nombre de instalación en AIX, Linux, and Windows

Nombres de archivo de IBM MQ

Cada gestor de colas de IBM MQ, cola, definición de proceso, lista de nombres, canal, canal de conexión cliente, escucha, servicio y objeto de información de autenticación se representa mediante un archivo. Puesto que los nombres de objetos no son necesariamente nombres de archivo válidos, el gestor de colas convierte el nombre de objeto en un nombre de archivo válido cuando es necesario.

La vía de acceso predeterminada a un directorio de gestor de colas es la siguiente:

- Un prefijo, que se define en la información de configuración de IBM MQ:
  - Linux AIX En AIX and Linux, el prefijo predeterminado es `/var/mqm`. El prefijo se configura en la stanza `DefaultPrefix` del archivo de configuración mqs.ini.
  - Windows En los sistemas Windows de 32 bits, el prefijo predeterminado es `C:\Archivos de programa (x86)\IBM\WebSphere MQ`. En los sistemas Windows de 64 bits, el prefijo predeterminado es `C:\Archivos de programa\IBM\MQ`. Tanto en las instalaciones de 32 bits como de 64 bits, los directorios de datos se instalan en `C:\ProgramData \IBM \MQ`. El prefijo se configura en la stanza `DefaultPrefix` del archivo de configuración mqs.ini.

  Cuando está disponible, el prefijo se puede cambiar utilizando la página de propiedades de IBM MQ en IBM MQ Explorer; de lo contrario, edite el archivo de configuración de `mqs.ini` manualmente.
- El nombre del gestor de colas se transforma en un nombre de directorio válido. Por ejemplo, el gestor de colas:

  ```
  queue.manager
  ```

  se representaría como:

  ```
  queue!manager
  ```

  Este proceso se denomina *transformación de nombres*.

En IBM MQ, puede asignar a un gestor de colas un nombre que contenga hasta 48 caracteres.

Por ejemplo, podría asignar el siguiente nombre a un gestor de colas:

```
QUEUE.MANAGER.ACCOUNTING.SERVICES
```

Sin embargo, cada gestor de colas se representa mediante un archivo y existen limitaciones con respecto a la longitud máxima que pueden tener los nombres de archivo y a los caracteres que se pueden utilizar en los nombres. Como resultado, los nombres de archivo que representan objetos se transforman automáticamente para cumplir los requisitos del sistema de archivos.

Las reglas que controlan la transformación de un nombre de gestor de colas son las siguientes:

1. Transformar caracteres individuales:
   - De . a!
   - De / a &
2. Si el nombre sigue sin ser válido:
   a. Se trunca a ocho caracteres
   b. Se añade un sufijo numérico de tres caracteres

Por ejemplo, suponiendo el prefijo predeterminado y un gestor de colas con el nombre `queue.manager`:

- Windows En Windows con NTFS o FAT32, el nombre del gestor de colas se convierte en:

  ```
  C:\Archivos de programa\IBM\MQ\qmgrs\queue!manager
  ```

- Windows En Windows con FAT, el nombre de gestor de colas se convierte en:

  ```
  C:\Archivos de programa\IBM\MQ\qmgrs\queue!ma
  ```

- Linux AIX En AIX and Linux, el nombre del gestor de colas pasa a ser:

```
/var/mqm/qmgrs/queue!manager
```

El algoritmo de transformación también permite distinguir entre nombres que sólo difieren en el tipo de letra (mayúsculas/minúsculas) en sistemas de archivos que no son sensibles a mayúsculas y minúsculas.

Transformación de nombres de objetos

Los nombres de objetos no son necesariamente nombres válidos para los sistemas de archivos. Es posible que tenga que transformar los nombres de objetos. El método utilizado es distinto del empleado para los nombres de gestores de colas porque, aunque sólo hay unos pocos nombres de gestores de colas por cada máquina, puede haber un gran número de otros objetos para cada gestor de colas. En el sistema de archivos se representan colas, definiciones de proceso, listas de nombres, canales, canales de conexión con el cliente, escuchas, servicios y objetos de información de autenticación.

Cuando el proceso de transformación genera un nuevo nombre, no hay ninguna relación simple con el nombre de objeto original. Puede utilizar el mandato **dspmqfls** para efectuar conversiones entre nombres de objetos reales y transformados.

Referencia relacionada

dspmqfls (visualizar nombres de archivos)

Información relacionada

Stanza AllQueueManagers del archivo mqs.ini

IBM i Nombres de objetos en IBM i

Un gestor de colas tiene una biblioteca de gestor de colas asociada que tiene un nombre exclusivo. Es posible que los nombres del gestor de colas y los nombres de objetos deban transformarse para cumplir los requisitos del sistema de archivos integrado (IFS) de IBM i.

Cuando se crea un gestor de colas, IBM MQ asocia una biblioteca del gestor de colas con él. A esta biblioteca del gestor de colas se le da un nombre exclusivo, de no más de 10 caracteres, que se basa principalmente en el nombre del gestor de colas definido por el usuario. Tanto el gestor de colas como la biblioteca del gestor de colas se colocan en un directorio que también se basa en el nombre del gestor de colas con el prefijo `/QIBM/UserData/mqm`. A continuación, se ofrece un ejemplo de un gestor de colas, una biblioteca del gestor de colas y un directorio:

Nombre del gestor de colas	ORANGE
Nombre de la biblioteca del gestor de colas	QMORANGE
Directorio	`/QIBM/UserData/mqm/ORANGE`

Todos los nombres de gestor de colas y nombres de biblioteca del gestor de colas se graban en stanzas en el archivo `/QIBM/UserData/mqm/mqs.ini`.

Directorios y archivos IFS de IBM MQ

Integrated File System (IFS) de IBM ilo utiliza de forma extensiva IBM MQ para almacenar datos. Para obtener más información acerca del IFS consulte la publicación *Integrated File System Introduction*.

Cada objeto de IBM MQ , por ejemplo, un canal o un gestor de colas, se representa mediante un archivo. Puesto que los nombres de objetos no son necesariamente nombres de archivo válidos, el gestor de colas convierte el nombre de objeto en un nombre de archivo válido cuando es necesario.

La vía de acceso a un directorio de gestor de colas se forma a partir de los siguientes elementos:

- Un prefijo, que se define en el archivo de configuración del gestor de colas, `qm.ini`. El prefijo predeterminado es `/QIBM/UserData/mqm`.
- Un literal, `qmgrs`.
- Un nombre de gestor de colas codificado, que es el nombre del gestor de colas transformado en un nombre de directorio válido. Por ejemplo, el gestor de colas `queue/manager` está representado por `queue&manager`.

Este proceso se denomina transformación de nombres.

Transformación de nombres de gestores de colas del IFS

En IBM MQ, puede asignar a un gestor de colas un nombre que contenga hasta 48 caracteres.

Por ejemplo, puede denominar a un gestor de colas `QUEUE/MANAGER/ACCOUNTING/SERVICES`. Del mismo modo que cada gestor de colas crea una biblioteca, cada gestor de colas se representa mediante un archivo. Debido a puntos de código variantes en EBCDIC, existen limitaciones a los caracteres que se pueden utilizar en el nombre. Como resultado, los nombres de los archivos del IFS que representan objetos se transforman automáticamente para cumplir los requisitos del sistema de archivos.

Utilizando el ejemplo de un gestor de colas con el nombre `queue/manager`, transformando el carácter `/` en &y asumiendo el prefijo predeterminado, el nombre del gestor de colas en IBM MQ for IBM i se convierte en `/QIBM/UserData/mqm/qmgrs/queue&manager`.

Transformación de nombres de objetos

Los nombres de objetos no son necesariamente nombres válidos del sistema de archivos, por lo que es posible que los nombres de objetos deban transformarse. El método utilizado es distinto del empleado para los nombres de los gestores de colas porque, aunque sólo hay unos pocos nombres de gestores de colas para cada máquina, puede haber un gran número de otros objetos para cada gestor de colas. Sólo las definiciones de procesos, colas y listas de nombres están representadas en el sistema de archivos; los canales no se ven afectados por estas consideraciones.

Cuando el proceso de transformación genera un nuevo nombre, no hay ninguna relación simple con el nombre de objeto original. Puede utilizar el mandato DSPMQMOBJN para ver los nombres transformados para objetos IBM MQ.

Gestión de colas distribuidas y clústeres

La gestión de colas distribuidas hace referencia al envío de mensajes desde un gestor de colas a otro. El gestor de colas receptor puede estar en la misma máquina o en otra; puede estar próximo o en el otro lado del mundo. Puede ejecutarse en la misma plataforma que el gestor de colas local o puede estar en cualquiera de las plataformas soportadas por IBM MQ. Puede definir manualmente todas las conexiones en un entorno de colas distribuidas o puede crear un clúster y dejar que IBM MQ defina automáticamente gran parte del detalle de conexión.

Gestión de colas distribuidas

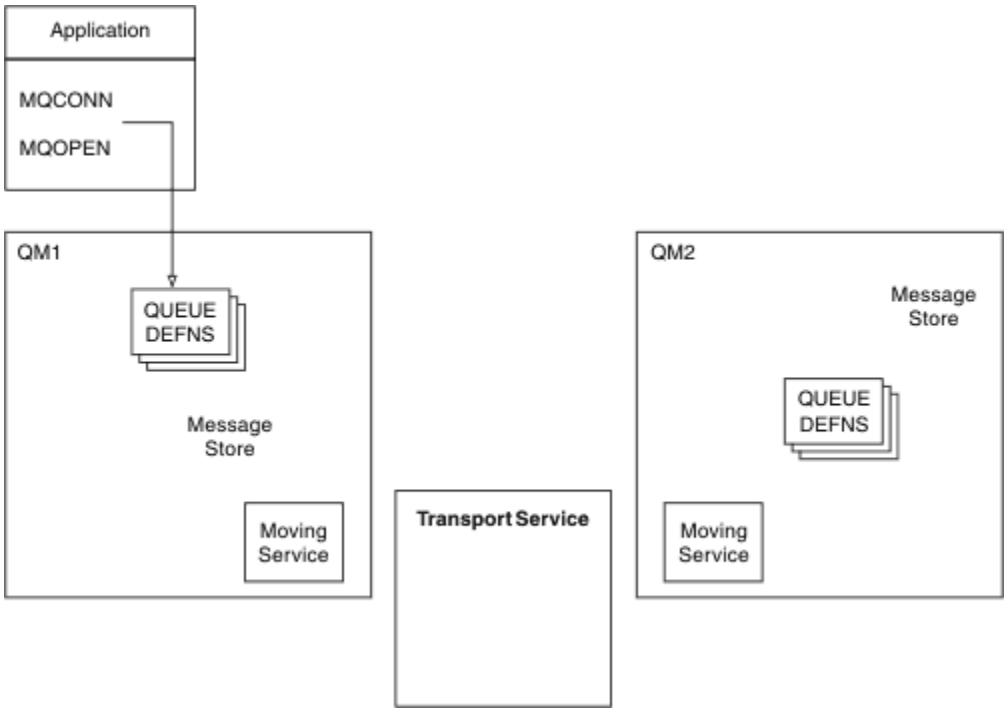


Figura 4. Visión general de los componentes de la gestión de colas distribuidas

En la figura anterior:

- Una aplicación utiliza la llamada MQCONN para conectarse a un gestor de colas. A continuación, la aplicación utiliza la llamada MQOPEN para abrir una cola de modo que pueda poner mensajes en dicha cola.
- Cada gestor de colas tiene una definición para cada una de sus colas. Puede tener definiciones de *colas locales* (es decir, alojadas en este gestor de colas) y definiciones de *colas remotas* (es decir, alojadas en otros gestores de colas).
- Si los mensajes van destinados a una cola remota, el gestor de colas local los retiene en una *cola de transmisión*, que los persiste en un almacén de mensajes hasta que puedan reenviarse al gestor de colas remoto.
- Cada gestor de colas contiene un software de comunicaciones, conocido como *servicio de mudanzas*, que utiliza para comunicarse con otros gestores de colas.
- El *servicio de transporte* es independiente del gestor de colas y puede ser uno de los siguientes (dependiendo de la plataforma):
  - Systems Network Architecture Advanced Program-to Program Communication (SNA APPC)
  - Protocolo de control de la transmisión/Protocolo de Internet (TCP/IP)
  - Network Basic Input/Output System (NetBIOS)
  - Sequenced Packet Exchange (SPX)

Componentes necesarios para enviar un mensaje

Si se va a enviar un mensaje a un gestor de colas remoto, el gestor de colas local necesita las definiciones de una *cola de transmisión* y de un *canal*. Un canal es un enlace de comunicaciones unidireccional entre dos gestores de colas. Puede transportar mensajes destinados a cualquier número de colas d el gestor de colas remoto.

Cada extremo de un canal tiene una definición independiente, definiéndose, por ejemplo, como el extremo emisor y el extremo receptor. Un canal sencillo consta de una definición de canal *emisor* en el gestor de colas local y una definición de canal *receptor* en el gestor de colas remoto. Estas dos definiciones deben tener el mismo nombre y juntas conforman un canal.

El software que maneja el envío y recepción de mensajes es el *Agente de canal de mensajes* (MCA). Hay un *agente de canal de mensajes* (MCA) en cada extremo de un canal.

Cada gestor de colas debe tener un *cola de mensajes no entregados* (también conocido como *cola de mensajes no entregados*). Los mensajes se colocan en esta cola si no se pueden entregar en su destino.

La figura siguiente muestra la relación entre gestores de colas, colas de transmisión, canales y los MCA:

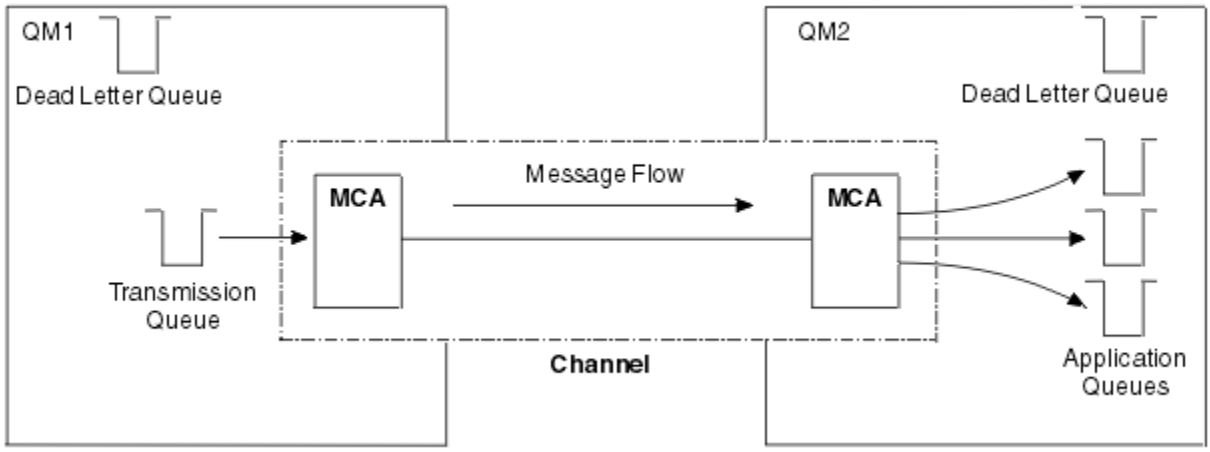


Figura 5. Envío de mensajes

Componentes necesarios para devolver un mensaje

Si la aplicación necesita que el gestor de colas remoto devuelva mensajes, debe definir otro canal, que se ejecutará en el sentido contrario entre los gestores de colas, tal como se muestra en la figura siguiente:

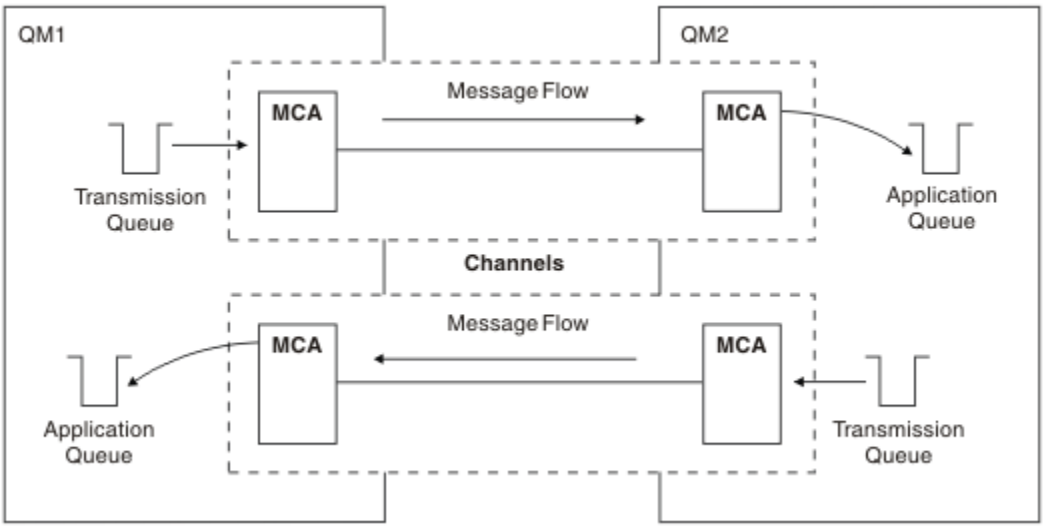


Figura 6. Envío de mensajes en ambos sentidos

Clústeres

En lugar de definir manualmente todas las conexiones en un entorno de gestión de colas distribuidas, puede agrupar un conjunto de gestores de colas en un clúster. Cuando se hace esto, los gestores de colas pueden poner sus colas a disposición de otros gestores de colas en el clúster sin necesidad de crear explícitamente definiciones de canal, definiciones de cola remota ni colas de transmisión por cada destino. Cada gestor de colas de un clúster tiene una cola de transmisión única que transmite mensajes a cualquier otro gestor de colas del clúster. Por cada gestor de colas sólo debe definir un canal de clúster receptor y un canal de clúster emisor, los canales adicionales se gestionan automáticamente por el clúster.

Un cliente IBM MQ se puede conectar a un gestor de colas que forma parte de un clúster, del mismo modo que se puede conectar a cualquier otro gestor de colas. Al igual que ocurre con una gestión de colas distribuidas configurada manualmente, la llamada MQPUT se utiliza para poner un mensaje en una cola de cualquier gestor de colas. Puede utilizar la llamada MQGET para recuperar los mensajes de una cola local.

Los gestores de colas en las plataformas que soportan los clústeres no deben formar parte de un clúster. Puede seguir utilizando la configuración manual de gestión de colas distribuidas, así como (o en lugar de) utilizar clústeres.

Ventajas de utilizar clústeres

La agrupación en clúster proporciona dos beneficios clave:

- Los clústeres simplifican la administración de las redes IBM MQ, que normalmente requieren la configuración de muchas definiciones de objeto para canales, colas de transmisión y colas remotas. Esta situación es especialmente útil en las redes de gran tamaño con posibilidad de cambios, donde deben interconectarse muchos gestores de colas. Se trata de una arquitectura particularmente difícil de configurar y mantener activa.
- Los clústeres se pueden utilizar para distribuir la carga de trabajo del tráfico de mensajes entre colas y gestores de colas en el clúster. Este tipo de distribución permite distribuir la carga de trabajo de mensajes de una sola cola entre varias instancias equivalentes de esa cola ubicadas en varios gestores de colas. Esta distribución de la carga de trabajo se puede utilizar para lograr una mayor resistencia a las anomalías del sistema y mejorar el rendimiento de escalado de flujos de mensajes especialmente activos en un sistema. En este tipo de entorno, cada una de las instancias de las colas distribuidas tienen aplicaciones consumidoras que procesan los mensajes. Para obtener más información, consulte Utilización de clústeres para la gestión de carga de trabajo.

Cómo se direccionan los mensajes en un clúster

Puede pensarse en un clúster como una red de gestores de colas mantenida por un administrador de sistemas meticuloso. Siempre que defina una cola de clúster, el administrador del sistema crea automáticamente las definiciones de cola remota correspondientes, según sea necesario en los otros gestores de colas.

No es necesario crear definiciones de cola de transmisión porque IBM MQ proporciona una cola de transmisión en cada gestor de colas del clúster. Esta única cola de transmisión se puede utilizar para llevar mensajes a cualquier otro gestor de colas del clúster. No está limitado a utilizar una cola de transmisión único. Un gestor de colas puede utilizar varias colas de transmisión para separar los mensajes que van a cada gestor de colas en un clúster. Por lo general, un gestor de colas utiliza una sola cola de transmisión de clúster. Puede cambiar el atributo de gestor de colas DEFCLXQ, para que el gestor de colas utilice una cola de transmisión de clúster diferente para cada gestor de colas en un clúster. También puede definir las colas de transmisión de clúster de clúster manualmente.

Todos los gestores de colas que se unen a un clúster acuerdan trabajar de esta forma. Envían información sobre ellos mismos y sobre las colas que contienen, y reciben información sobre los otros miembros del clúster.

Para asegurar que no se pierda información cuando un gestor de colas deja de estar disponible, se especifican dos gestores de colas en el clúster para que actúen como *repositorios completos*. Dichos gestores de colas almacenan una información completa sobre todos los gestores de colas y colas en el clúster. Todos los demás gestores de colas del clúster solo almacenan información sobre los gestores de colas y colas con los que intercambian mensajes. Estos gestores de colas se conocen como *repositorios parciales*. Para obtener más información, consulte “Repositorio de clúster” en la página 57.

Para poder formar parte de un clúster, un gestor de colas debe tener dos canales; un canal de clúster emisor y un canal de clúster receptor:

- Un canal de clúster emisor es un canal de comunicación como un canal emisor. Debe crear manualmente un canal de clúster emisor en un gestor de colas para conectarse a un repositorio completo que ya es miembro del clúster.

- Un canal de clúster receptor es un canal de comunicación como un canal receptor. Debe crear manualmente un canal de clúster receptor. El canal actúa como el mecanismo para que el gestor de colas reciba las comunicaciones de clúster.

Todos los demás canales necesarios para la comunicación entre este gestor de colas y otros miembros del clúster se crean automáticamente.

La figura siguiente muestra los componentes de un clúster llamado CLUSTER:

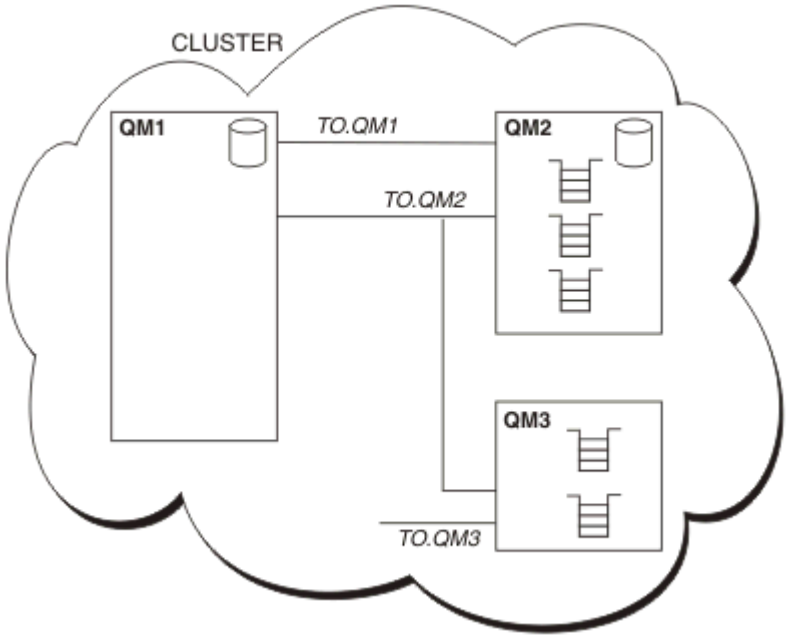


Figura 7. Un clúster de gestores de colas

- CLUSTER contiene tres gestores de colas, QM1, QM2 y QM3.
- QM1 y QM2 alojan repositorios completos de información sobre los gestores de colas y las colas del clúster.
- QM2 y QM3 alojan algunas colas de clúster, es decir, colas a las que puede acceder cualquier otro gestor de colas del clúster.
- Cada gestor de colas tiene un canal de clúster receptor denominado TO.qmgr en el que puede recibir mensajes.
- Cada gestor de colas también tiene un canal de clúster emisor mediante el que puede enviar información a uno de los gestores de colas de repositorio.
- QM1 y QM3 envían al depósito de QM2 y QM2 envía al depósito de QM1.

Componentes de la gestión de colas distribuidas

Los componentes de la gestión de colas distribuidas son canales de mensaje, agentes de canal de mensaje, colas de transmisión, escuchas e iniciadores de canal y programas de salida de canal. La definición de cada extremo de un canal de mensajes puede ser de varios tipos.

Los canales de mensajes son los canales que transportan mensajes de un gestor de colas a otro. No hay que confundir los canales de mensajes con los canales MQI. Hay dos tipos de canales MQI, de conexión con el servidor (SVRCONN) y de conexión con el cliente (CLNTCONN). Para más información, consulte Canales.

La definición de cada extremo de un canal de mensajes puede ser de uno de los tipos siguientes:

- Emisor (SDR)

- Receptor (RCVR)
- Servidor (SVR)
- Peticionario (RQSTR)
- Emisor de clúster (CLUSSDR)
- Receptor de clúster (CLUSRCVR)

Un canal de mensajes se define utilizando uno de estos tipos definidos en un extremo y un tipo compatible en el otro extremo. Las posibles combinaciones son:

- Emisor-receptor
- Peticionario-servidor
- Peticionario-emisor (devolución de llamada)
- Servidor-receptor
- Emisor de clúster-receptor de clúster

Las instrucciones detalladas para crear un canal emisor-receptor se incluyen en Definición de los canales. Para obtener ejemplos de los parámetros necesarios para configurar los canales emisor-receptor, consulte la Información de configuración de ejemplo aplicable a su plataforma. Para conocer los parámetros necesarios para definir un canal de cualquier tipo, consulte DEFINE CHANNEL.

Canales emisor-receptor

Un emisor en un sistema inicia el canal para que pueda enviar mensajes al otro sistema. El emisor solicita al receptor en el otro extremo del canal que se inicie. El emisor envía mensajes desde su cola de transmisión al receptor. El receptor pone los mensajes en la cola de destino. La Figura 8 en la página 48 ilustra este proceso.

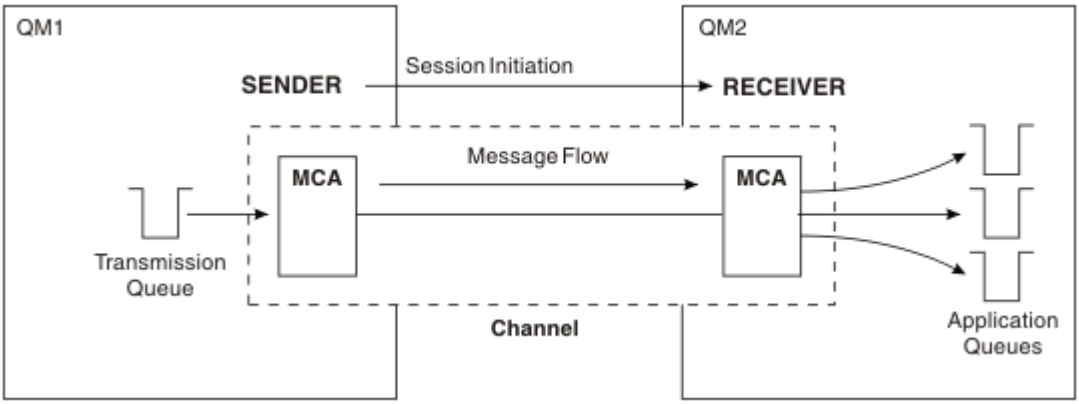


Figura 8. Un canal emisor-receptor

Canales peticionario-servidor

Un peticionario en un sistema inicia el canal para que pueda recibir mensajes del otro sistema. El peticionario solicita al servidor en el otro extremo del canal que se inicie. El servidor envía mensajes al peticionario desde la cola de transmisión definida en su definición de canal.

Un canal servidor también puede iniciar la comunicación y enviar mensajes a un peticionario. Esto sólo se aplica a servidores *totalmente calificados*, es decir, canales servidor que tienen el nombre de conexión del socio especificado en la definición de canal. Un peticionario puede iniciar un servidor totalmente calificado o puede ser éste el que inicie la comunicación con un peticionario.

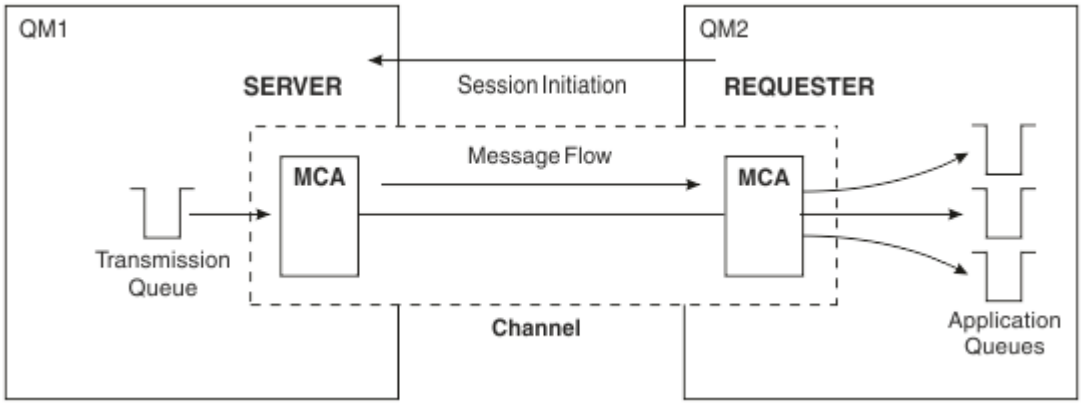


Figura 9. Un canal peticionario-servidor

Canales peticionario-emisor

El peticionario inicia el canal y el emisor finaliza la llamada. Seguidamente, el emisor reinicia la comunicación en función de la información de su definición de canal (conocida como *devolución de llamada*). Envía mensajes desde la cola de transmisión al peticionario.

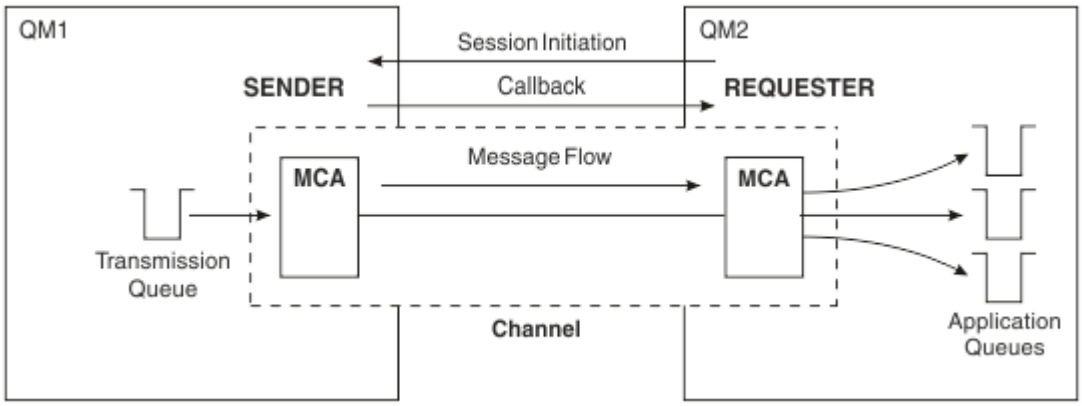


Figura 10. Un canal peticionario-emisor

Canales servidor-receptor

Se trata del mismo caso que el de los canales emisor-receptor, pero sólo se aplica a servidores *totalmente calificados*, es decir, canales servidor que tienen el nombre de conexión del socio especificado en la definición de canal. El inicio del canal debe llevarse a cabo en el extremo servidor del enlace. La ilustración de este proceso es idéntica a la ilustración de la Figura 8 en la página 48.

Canales de clúster emisor

En un clúster, cada gestor de colas tiene un canal de clúster emisor mediante el que puede enviar información del clúster a uno de los gestores de cola de repositorio completo. Los gestores de colas también pueden enviar mensajes a otros gestores de colas en los canales de clúster emisor.

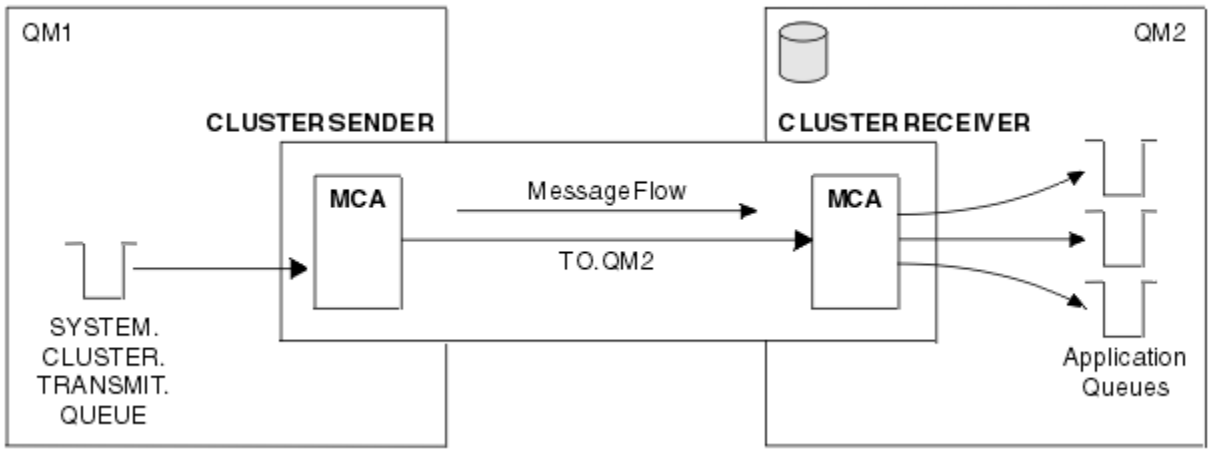


Figura 11. Un canal de clúster emisor

Canales de clúster receptor

En un clúster, cada gestor de colas tiene un canal de clúster receptor en el que puede recibir mensajes e información sobre el clúster. La ilustración de este proceso es idéntica a la ilustración de la Figura 11 en la página 50.

Colas de mensajes no entregados

La cola de mensajes no entregados es la cola a la que se envían los mensajes que no pueden enviarse a su destino correcto. Cada gestor de colas suele tener una cola de mensajes no entregados.

Una *cola de mensajes no entregados* (DLQ), denominada a veces *cola de mensajes sin entregar* es una cola que retiene los mensajes que no se pueden entregar a sus colas de destino, por ejemplo, porque la cola no exista o porque esté llena. Las colas de mensajes no entregados también se utilizan en el extremo emisor de un canal para los errores de conversión de datos. Cada gestor de colas en una red suele tener una cola local que se utiliza como cola de mensajes no entregados para que los mensajes que no pueden entregarse en su destino puedan almacenarse para recuperarlos posteriormente.

Los gestores de colas, los agentes de canal de mensajes (MCA) y las aplicaciones pueden transferir mensajes a las DLQ. Todos los mensajes de la DLQ deben tener como prefijo una estructura de *cabecera de mensaje no entregado*, MQDLH. El campo *Reason* de la estructura MQDLH contiene un código de razón que indica el motivo por el cual el mensaje está en la DLQ.

Normalmente debe definirse una cola de mensajes no entregados por cada gestor de colas. Si no lo hace y el MCA no puede poner un mensaje, se deja en la cola de transmisión y el canal se detiene. Además, si no se pueden entregar mensajes no permanentes de forma rápida (consulte Mensajes rápidos y no permanentes) y no existe ninguna cola de mensajes no entregados en el sistema de destino, estos mensajes se descartan.

No obstante, la utilización de colas de mensajes no entregados puede afectar al orden en que se entregan los mensajes, por lo que puede optar por no utilizarlas.

Tareas relacionadas

Trabajar con colas de mensajes no entregados

Resolución de problemas de mensajes no entregados

Referencia relacionada

runmqdlq (ejecutar manejador de cola de mensajes no entregados)

Definiciones de colas remotas

Las definiciones de colas remotas son definiciones de las colas que pertenecen a otro gestor de colas.

Mientras que las aplicaciones pueden recuperar mensajes sólo de colas locales, pueden colocar mensajes en colas locales o colas remotas. Por lo tanto, además de una definición para cada una de sus colas locales, un gestor de colas puede tener *definiciones de colas remotas*. La ventaja de las definiciones de colas remotas es que permiten que una aplicación transfiera un mensaje a una cola remota sin tener que especificar el nombre de la cola remota o el gestor de colas remoto o el nombre de la cola de transmisión. Las definiciones de colas remotas le ofrecen independencia de la ubicación.

Existen otros usos para las definiciones de colas remotas, que se describen más adelante.

Cómo acceder al gestor de colas remoto

Puede que no siempre tenga un canal entre cada uno de los gestores de colas de origen y de destino. Hay otras formas de conectarlos, incluyendo los saltos múltiples y el compartición de canales, utilizando canales diferentes y agrupaciones.

Saltos múltiples

Si no hay ningún enlace de comunicación directo entre el gestor de colas de origen y el gestor de colas de destino, es posible pasar por un *gestor de colas intermedio* (o más de uno) de camino al gestor de colas de destino. Esto se conoce como *salto múltiple*.

Debe definir canales entre todos los gestores de colas y colas de transmisión en los gestores de colas intermedios. Esto se muestra en la Figura 12 en la página 51.

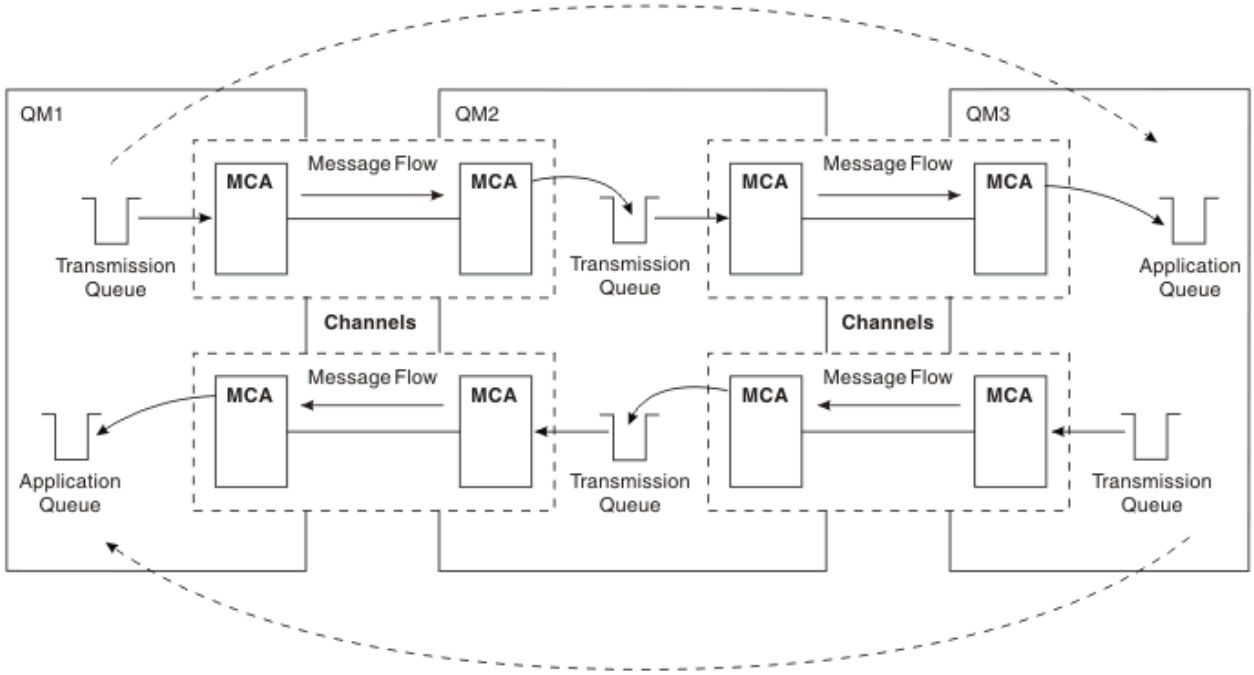


Figura 12. Paso a través de gestores de colas intermedios

Compartición de canales

Como diseñador de aplicaciones, tiene la opción de obligar a las aplicaciones a especificar el nombre del gestor de colas remoto junto con el nombre de la cola, o de crear una *definición de cola remota* para cada cola remota. Esta definición contiene el nombre del gestor de colas remoto, el nombre de la cola y el nombre de la cola de transmisión. De cualquier modo, los mensajes de todas las aplicaciones que utilizan colas en la misma ubicación remota se envían a través de la misma cola de transmisión. Esto se muestra en la Figura 13 en la página 52.

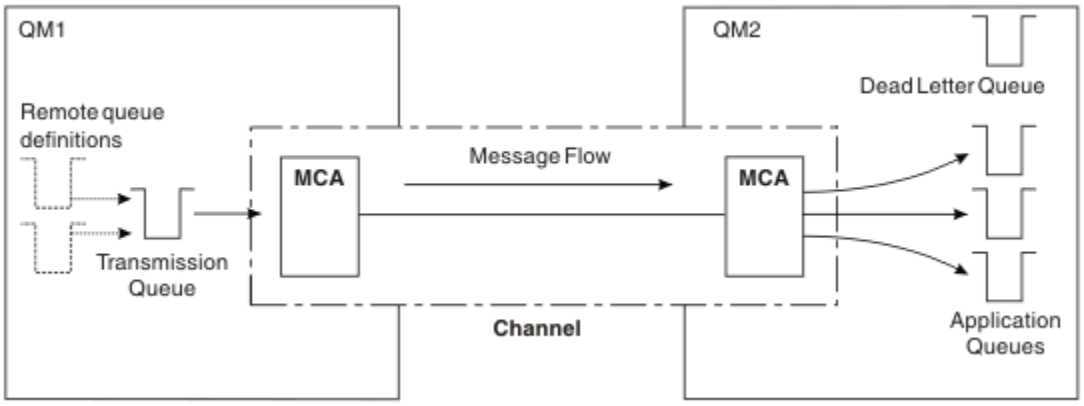


Figura 13. Compartición de una cola de transmisión

En la Figura 13 en la página 52 se muestra que los mensajes de varias aplicaciones a varias colas remotas pueden utilizar el mismo canal.

Utilización de canales diferentes

Si tiene que enviar mensajes de tipos distintos entre dos gestores de colas, puede definir más de un canal entre los dos. Hay ocasiones en que se necesitan canales alternativos, puede que por razones de seguridad o tal vez porque a veces interese elegir una velocidad de entrega mayor comparada con la del tráfico de mensajes habitual.

Para configurar un segundo canal deberá definir otro canal y otra cola de transmisión y crear una definición de cola remota especificando la ubicación y el nombre de la cola de transmisión. A continuación, las aplicaciones pueden utilizar cualquier canal, pero los mensajes siguen entregándose a las mismas colas de destino. Esto se muestra en la Figura 14 en la página 52.

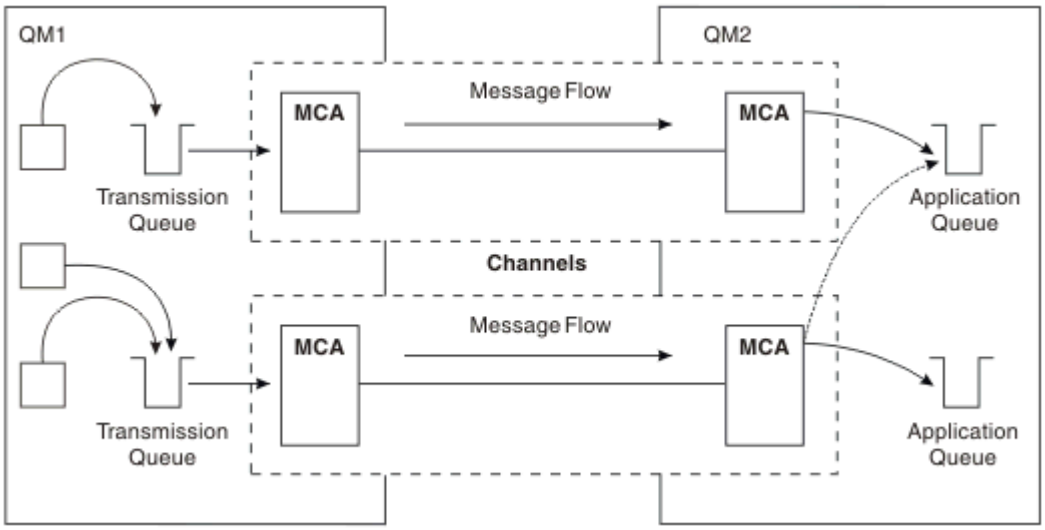


Figura 14. Utilización de varios canales

Cuando se utilizan definiciones de colas remotas para especificar una cola de transmisión, las aplicaciones **no** deben especificar la ubicación (es decir, el gestor de colas de destino) por sí mismas. Si lo hacen, el gestor de colas no utiliza las definiciones de cola remota. Las definiciones de colas remotas le ofrecen independencia de la ubicación. Las aplicaciones pueden poner mensajes en una cola *lógica* sin saber dónde se encuentra y se puede modificar la cola *física* sin tener que cambiar las aplicaciones.

Utilización de clústeres

Cada gestor de colas de un clúster define un canal de clúster receptor. Cuando otro gestor de colas desea enviar un mensaje a ese gestor de colas, define el correspondiente canal de clúster emisor automáticamente. Por ejemplo, si hay más de una instancia de una cola en un clúster, el canal de clúster emisor se puede definir para cualquiera de los gestores de colas que aloja la cola. IBM MQ utiliza un algoritmo de gestión de carga de trabajo que utiliza una rutina en rueda para seleccionar un gestor de colas disponible adonde se dirige un mensaje. Puede obtener información adicional consultando Clústeres.

Información de direccionamiento

Cuando una aplicación transfiere mensajes que están destinados a un gestor de colas remoto, el gestor de colas local añade una cabecera de transmisión antes de colocarlos en la cola de transmisión. Esta cabecera contiene el nombre de la cola y el gestor de colas de destino, es decir, la *información de direccionamiento*.

En un entorno de gestor de colas único, la dirección de una cola de destino se establece cuando una aplicación abre una cola para colocar mensajes en ella. Como la cola de destino está en el mismo gestor de colas, no se necesita información de direccionamiento.

En un entorno de gestión de colas distribuidas, el gestor de colas necesita saber no sólo el nombre de la cola de destino, sino también la ubicación de esa cola (es decir, el nombre del gestor de colas), y la ruta a la ubicación remota (es decir, la cola de transmisión). Esta información de direccionamiento se encuentra en la cabecera de transmisión. El canal receptor elimina la cabecera de transmisión y utiliza la información que hay en ella para localizar la cola de destino.

Puede evitar la necesidad de que las aplicaciones especifiquen el nombre del gestor de colas de destino si utiliza una definición de cola remota. Esta definición especifica el nombre de la cola remota, el nombre del gestor de colas remoto al que van destinados los mensajes y el nombre de la cola de transmisión utilizado para transportar los mensajes.

¿Qué son los alias?

Los alias se utilizan para ofrecer una calidad de servicio a los mensajes. El alias del gestor de colas permite al administrador del sistema modificar el nombre de un gestor de colas de destino sin tener que cambiar las aplicaciones. También permite que el administrador del sistema modifique la ruta a un gestor de colas de destino o establecer una ruta que implica pasar por otros gestores de colas (saltos múltiples). El alias de la cola de respuestas proporciona una calidad de servicio a las respuestas.

Los alias de gestor de colas y los alias de cola de respuestas se crean utilizando una definición de cola remota que tiene un RNAME vacío. Estas definiciones no definen colas reales; las utiliza el gestor de colas para determinar nombres de colas físicas, nombres de gestores de colas y colas de transmisión.

Las definiciones de alias se caracterizan por tener un RNAME vacío.

Resolución de nombres de colas

La resolución de nombres de colas se lleva a cabo en cada gestor de colas cada vez que se abre una cola. Su finalidad es identificar la cola de destino, el gestor de colas de destino (que puede ser local) y la ruta a ese gestor de colas (que puede ser un valor nulo). El nombre resuelto tiene tres partes: el nombre del gestor de colas, el nombre de la cola y, si el gestor de colas es remoto, la cola de transmisión.

Si existe una definición de cola remota, no se hace referencia a las definiciones de alias. El nombre de cola proporcionado por la aplicación se resolverá en el nombre de la cola de destino, el gestor de colas remoto y la cola de transmisión especificada en la definición de cola remota. Para obtener información más detallada sobre la resolución de nombres de colas, consulte Resolución de nombres de colas.

Si no hay ninguna definición de cola remota y se especifica un nombre de gestor de colas o se ha resuelto mediante el servicio de nombres, el gestor de colas comprueba si hay una definición de alias de gestor de colas que coincida con el nombre del gestor de colas suministrado. Si existe, se utiliza la información de la definición para resolver el nombre del gestor de colas en el nombre del gestor de colas de destino. La

definición de alias de gestor de colas también se puede utilizar para determinar la cola de transmisión del gestor de colas de destino.

Si el nombre de cola resuelto no es una cola local, tanto el nombre del gestor de colas como el nombre de la cola se incluyen en la cabecera de transmisión de todos los mensajes transferidos por la aplicación a la cola de transmisión.

La cola de transmisión utilizada normalmente tiene el mismo nombre que el gestor de colas resuelto, a menos que lo modifique una definición de cola remota o una definición de alias de gestor de colas. Si no ha definido una cola de transmisión de este tipo pero sí ha definido una cola de transmisión predeterminada, se utilizará éste.

z/OS Los nombres de gestores de colas que se ejecutan en z/OS tienen un límite de cuatro caracteres.

Definiciones de alias de gestor de colas

Las definiciones de alias de gestor de colas se aplican cuando una aplicación que abre una cola para transferir un mensaje, especifica el nombre de la cola **y** el nombre del gestor de colas.

Las definiciones de alias de gestor de colas tienen tres usos:

- Al enviar mensajes, volver a correlacionar el nombre del gestor de colas
- Al enviar mensajes, modificar o especificar la cola de transmisión
- Al recibir mensajes, determinar si el gestor de colas local es el destino previsto para los mensajes

Mensajes de salida: volver a correlacionar el nombre del gestor de colas

Las definiciones de alias de gestor de colas pueden utilizarse para correlacionar el nombre del gestor de colas especificado en una llamada MQOPEN. Por ejemplo, una llamada MQOPEN especifica el nombre de cola THISQ y el nombre de gestor de colas YOURQM. En el gestor de colas local, hay la siguiente definición de alias de gestor de colas:

```
DEFINE QREMOTE (YOURQM) RQMNAME(REALQM)
```

Esto muestra que el gestor de colas real que se va a utilizar, cuando una aplicación transfiere mensajes al gestor de colas YOURQM, es REALQM. Si el gestor de colas local es REALQM, pone los mensajes en la cola THISQ, que es una cola local. Si el gestor de colas local no se llama REALQM, direcciona el mensaje a una cola de transmisión de clúster denominada REALQM. El gestor de colas cambia la cabecera de transmisión para que diga REALQM en vez de YOURQM.

Mensajes de salida: modificar o especificar la cola de transmisión

Figura 15 en la página 55 un ejemplo en el que los mensajes llegan al gestor de colas QM1 con cabeceras de transmisión que muestran nombres de cola del gestor de colas QM3. En este ejemplo, se puede acceder a QM3 mediante saltos múltiples a través de QM2.

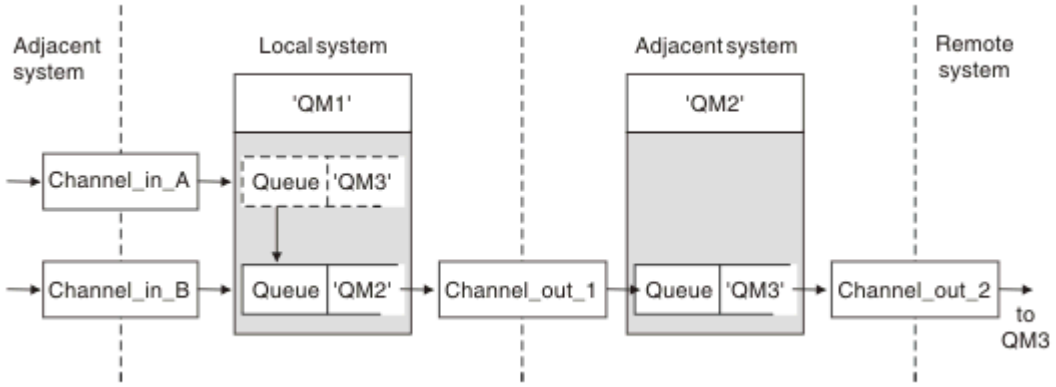


Figura 15. Alias del gestor de colas

Todos los mensajes para QM3 se capturan en QM1 con un alias de gestor de colas. El alias de gestor de colas se denomina QM3 y contiene la definición QM3 a través de la cola de transmisión QM2. La definición se parece a la del ejemplo siguiente:

```
DEFINE QREMOTE (QM3) RNAME(' ') RQMNAME(QM3) XMITQ(QM2)
```

El gestor de colas coloca los mensajes en la cola de transmisión QM2 pero no modifica la cabecera de la cola de transmisión debido a que el nombre del gestor de colas de destino, QM3, no se modifica.

Todos los mensajes que llegan a QM1 y muestran una cabecera de transmisión que contiene un nombre de cola en QM2 también se colocan en la cola de transmisión QM2. De este modo, los mensajes con destinos diferentes se reúnen en una cola de transmisión común de un sistema adyacente adecuado para transmitirlos a sus destinos.

Mensajes de entrada: determinar el destino

Un MCA receptor abre la cola a la que se hace referencia en la cabecera de transmisión. Si existe una definición de alias de gestor de colas con el mismo nombre que el gestor de colas al que se hace referencia, entonces el nombre del gestor de colas recibido en la cabecera de transmisión es sustituido por el RQMNAME de esa definición.

Este proceso tiene dos usos:

- Dirigir mensajes a otro gestor de colas
- Modificar el nombre del gestor de colas para que sea el mismo que el del gestor de colas local

Definiciones de alias de colas de respuesta

Una definición de alias de cola de respuestas especifica nombres alternativos para la información de respuesta en el descriptor de mensaje. La ventaja que tiene es que se puede cambiar el nombre de una cola o de un gestor de colas sin tener que modificar las aplicaciones.

Resolución de nombres de colas

Cuando una aplicación responde a un mensaje, utiliza los datos del *descriptor de mensaje* del mensaje que ha recibido para averiguar el nombre de la cola a la que tiene que responder. La aplicación emisora indica donde hay que enviar las respuestas y asocia esta información a sus mensajes. Este concepto debe coordinarse como parte del diseño de la aplicación.

La resolución de nombres de colas tiene lugar en el extremo emisor de la aplicación, antes de que el mensaje se transfiera a una cola. Por lo tanto, la resolución de nombres de colas se produce antes de la interacción con la aplicación remota a la que el mensaje se está enviando. Esta es la única situación en la que la resolución de nombres tiene lugar en un momento en el que no se abre la cola.

Resolución de nombres de cola utilizando un alias de gestor de colas

Normalmente, una aplicación especifica una cola de respuestas y deja el nombre del gestor de colas de respuestas en blanco. El gestor de colas completa su nombre en el momento de la transferencia. Este método funciona bien excepto cuando desea que se utilice un canal alternativo para las respuestas, por ejemplo, un canal que utiliza la cola de transmisión `QM1_relief` en lugar del canal de retorno predeterminado que utiliza la cola de transmisión `QM1`. En esta situación, los nombres de los gestores de colas especificados en las cabeceras de la cola de transmisión no coinciden con los nombres “reales” de los gestores de colas, pero se vuelven a especificar utilizando las definiciones de alias del gestor de colas. Para devolver las respuestas siguiendo rutas alternativas, también es necesario correlacionar los datos de la cola de respuestas, utilizando las definiciones de alias de cola de respuestas.

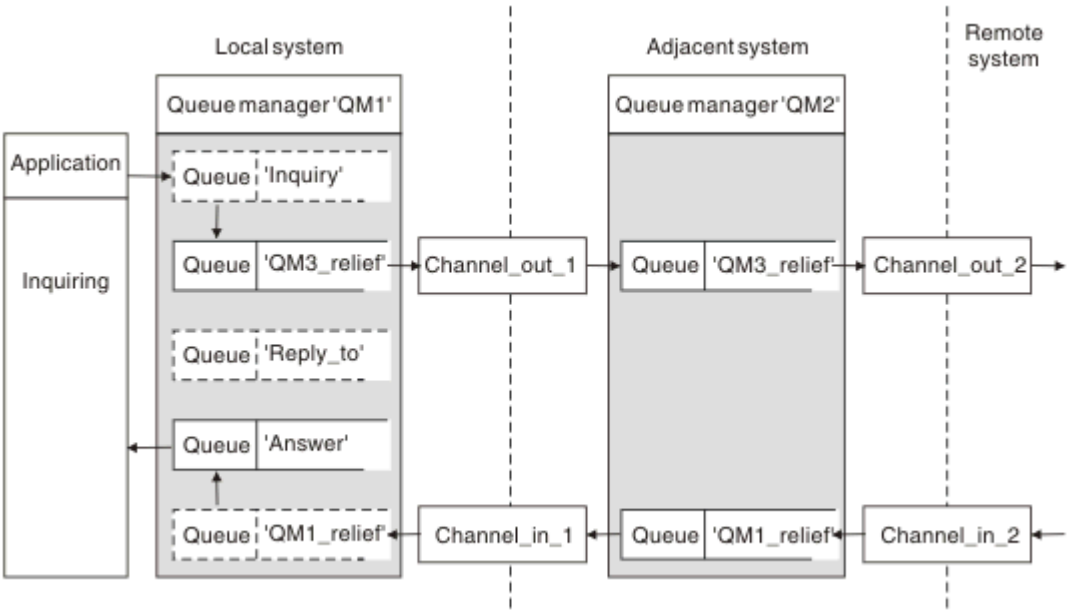


Figura 16. Alias de cola de respuestas utilizados para cambiar la ubicación de las respuestas

En el ejemplo de Figura 16 en la página 56:

1. La aplicación transfiere un mensaje utilizando la llamada MQPUT y especificando la información siguiente en el descriptor de mensaje:

   ```
   ReplyToQ='Reply_to'
   ReplyToQMgr=' '
   ```

   ReplyToQMgr debe estar en blanco para poder utilizar el alias de cola de respuestas.
2. Crea una definición de alias de cola de respuesta denominada `Reply_to`, que contiene el nombre `Answery` el nombre de gestor de colas `QM1_relief`.

   ```
   DEFINE QREMOTE ('Reply_to') RNAME ('Answer')
   RQMNAME ('QM1_relief')
   ```
3. Los mensajes se envían con un descriptor de mensaje que muestra `ReplyToQ='Answer'` y `ReplyToQMgr='QM1_relief'`.
4. La especificación de la aplicación debe incluir la información de que las respuestas se encuentran en la cola `Answer` en lugar de `Reply_to`.

Para prepararse para las respuestas debe crear el canal de retorno paralelo, definiendo:

- En QM2, la cola de transmisión denominada QM1_relief

```
DEFINE QLOCAL ('QM1_relief') USAGE(XMITQ)
```

- En QM1, el alias de gestor de colas QM1_relief

```
DEFINE QREMOTE ('QM1_relief') RNAME() RQMNAME(QM1)
```

  Este alias de gestor de colas termina la cadena de canales de retorno paralelos y captura los mensajes destinados a QM1.

Si cree que podría necesitar hacerlo en el futuro, asegúrese de que las aplicaciones utilizan el nombre de alias desde el principio. Por ahora, es un alias de cola normal para la cola de respuestas, pero más adelante puede cambiarse por un alias de gestor de colas.

Nombre de cola de respuestas

Hay que prestar atención a la hora de denominar las colas de respuestas. El motivo por el que una aplicación transfiere un nombre de cola de respuestas al mensaje es que puede especificar la cola a la que se envían sus respuestas. Cuando se crea una definición de alias de cola de respuestas con este nombre, la cola de respuestas real (es decir, una definición de cola local) no puede tener el mismo nombre. Por lo tanto, la definición de alias de cola de respuestas debe contener un nombre de cola nuevo además del nombre del gestor de colas y la especificación de la aplicación debe incluir la información de que sus respuestas se encuentra en esta otra cola.

Las aplicaciones ahora tienen que recuperar los mensajes de una cola distinta de la que habían denominado como cola de respuestas cuando transmitieron el mensaje original.

Componentes de clúster

Los clústeres están formados por gestores de colas, repositorios de clúster, canales de clúster y colas de clúster.

Consulte los subtemas siguientes para obtener información sobre cada uno de los componentes del clúster:

Conceptos relacionados

Comparación de agrupación en clúster y gestión de colas distribuidas

Tareas relacionadas

Configuración de un clúster de gestores de colas

Configurar un nuevo clúster

Repositorio de clúster

Un repositorio es una recopilación de información sobre los gestores de colas que son miembros de un clúster.

La información del repositorio incluye los nombres de los gestores de colas, sus ubicaciones, sus canales, las colas que alojan y otra información. La información se almacena en forma de mensajes en una cola llamada SYSTEM.CLUSTER.REPOSITORY.QUEUE. Esta cola es uno de los objetos predeterminados. Multi En Multiplatforms, se define al crear un gestor de colas de IBM MQ. z/OS En IBM MQ for z/OS, se define como parte de la personalización del gestor de colas.

Depósito completo y depósito parcial

Normalmente, dos gestores de colas en un clúster contienen un repositorio completo. Los gestores de colas restantes contienen todos un depósito parcial.

Un gestor de colas que aloja un conjunto completo de información sobre todos los gestores de colas del clúster tiene un repositorio completo. Los otros gestores de colas del clúster tienen repositorios parciales que contienen un subconjunto de la información de los repositorios completos.

Un repositorio parcial contiene información sólo sobre los gestores de colas con los que el gestor de colas tiene que intercambiar mensajes. Los gestores de colas solicitan actualizaciones de la información que necesitan, de modo que si ésta cambia, el gestor de colas de repositorio completo les envía la nueva información. Durante la mayor parte del tiempo, un repositorio parcial contiene toda la información que un gestor de colas necesita para funcionar en el clúster. Cuando un gestor de colas necesita información adicional, realiza consultas al depósito completo y actualiza su depósito parcial. Los gestores de colas utilizan la cola `SYSTEM.CLUSTER.COMMAND.QUEUE` para solicitar y recibir actualizaciones de los repositorios.

Cuando migre los gestores de colas que son miembros de un clúster, migre los repositorios completos antes que los repositorios parciales. Esto se debe a que un repositorio antiguo no puede almacenar los atributos nuevos introducidos en un release más reciente. Los tolera, pero no los almacena.

Gestor de colas de clúster

Un gestor de colas de clúster es un gestor de colas que es miembro de un clúster.

Un gestor de colas puede ser miembro de más de un clúster. Cada gestor de colas de clúster debe tener un nombre que sea exclusivo en todos los clústeres de los que es miembro.

Un gestor de colas de clúster puede alojar colas, lo cual anuncia a los otros gestores de colas del clúster. Sin embargo, no tiene por qué hacerlo. En vez de ello, puede suministrar mensajes a las colas alojadas en otro lugar del clúster y recibir únicamente las respuestas que vayan dirigidas explícitamente a él.

z/OS En IBM MQ for z/OS, un gestor de colas de clúster puede ser miembro de un grupo de compartición de colas. En este caso, comparte sus definiciones de cola con otros gestores de colas del mismo grupo de compartición de colas.

Los gestores de colas de clúster son autónomos. Tienen control total sobre las colas y los canales que definen. Sus definiciones no sea pueden modificar con otros gestores de colas excepto los gestores de colas del mismo grupo de compartición de colas. Los gestores de colas de repositorio no controlan las definiciones de otros gestores de colas del clúster. Contienen un conjunto completo de todas las definiciones, para utilizarlas cuando sea necesario. Un clúster es una federación de gestores de colas.

Después de crear o modificar una definición en un gestor de colas de clúster, la información se envía al gestor de colas de repositorio completo. Los otros repositorios del clúster se actualizan posteriormente.

Gestor de colas de repositorio completo

Un gestor de colas de repositorio completo es un gestor de colas de clúster que contiene una representación completa de los recursos del clúster. Para asegurar la disponibilidad, configure dos o más gestores de colas de repositorio completo en cada clúster. Los gestores de colas de repositorio completo reciben información enviada por los otros gestores de colas del clúster y actualizan sus repositorios. Se envían mensajes entre sí para asegurarse de que ambos se mantienen actualizados con información nueva sobre el clúster.

Gestores de colas y repositorios

Cada clúster tiene al menos un gestor de colas, preferiblemente dos, que contienen repositorios completos de información sobre los gestores de colas, las colas y los canales de un clúster. Estos repositorios también contienen solicitudes de los otros gestores de colas del clúster de actualizaciones de la información.

Los otros gestores de colas contienen cada uno un repositorio parcial, que contiene información sobre el subconjunto de colas y gestores de colas con los tienen que comunicarse. Los gestores de colas crean sus repositorio parciales realizando consultas cuando tienen que acceder por primera vez a otra cola o gestor de colas. Solicitan que se les notifique de cualquier nueva información sobre esa cola o gestor de colas.

Cada gestor de colas almacena su información de repositorio en mensajes en una cola llamada SYSTEM.CLUSTER.REPOSITORY.QUEUE. Los gestores de colas intercambien información de repositorio en mensajes en una cola llamada SYSTEM.CLUSTER.COMMAND.QUEUE.

Cada gestor de colas que se une a un clúster define un canal de clúster emisor, CLUSSDR, a uno de los repositorios. Se entera inmediatamente de qué otros gestores de colas del clúster contienen repositorios completos. A partir de entonces, el gestor de colas puede solicitar información de cualquiera de los repositorios. Cuando el gestor de colas envía información al repositorio elegido, también envía información al otro repositorio (si hay uno).

Un repositorio completo se actualiza cuando el gestor de colas que lo aloja recibe nueva información de uno de los gestores de colas al que está enlazado. La nueva información también se envía al otro repositorio, para reducir el riesgo de que se retrase si un gestor de colas de repositorio está fuera de servicio. Debido a que toda la información se envía dos veces, los repositorios tienen que descartar duplicados. Cada elemento de información contiene un número de secuencia, que los repositorios utilizan para identificar los duplicados. Todos los repositorios se mantienen sincronizados entre sí mediante el intercambio de mensajes.

Colas de clúster

Una cola de clúster es una cola que se aloja en un gestor de colas de clúster y que está disponible para otros gestores de colas del clúster.

Una definición de cola de clúster se anuncia en otros gestores de colas del clúster. Los otros gestores de colas del clúster pueden transferir mensajes a una cola de clúster sin necesidad de que haya una definición de cola remota correspondiente. Una cola de clúster se puede anunciar en más de un clúster utilizando una lista de nombres de clúster.

Cuando se anuncia una cola, cualquier gestor de colas del clúster puede poner mensajes en ella. Para transferir un mensaje, el gestor de colas debe averiguar, en los repositorios completos, donde está alojada la cola. A continuación, añade información de direccionamiento al mensaje y pone el mensaje a una cola de transmisión de clúster.

Una cola de clúster puede ser una cola que se comparte entre miembros de un grupo de compartición de colas en IBM MQ for z/OS.

Tareas relacionadas

Definición de cola de clúster

Comparison between shared queues and cluster queues

This information is designed to help you compare shared queues and cluster queues, and decide which might be more suitable for your system.

Channel Initiator costs

In cluster queues, messages are sent by channels, so allow for channel initiator costs in addition to application costs. There are costs in the network because channels get and put messages. These costs are not present with shared queues, which therefore use less processing power than cluster queues when moving messages between queue managers in a queue sharing group.

Availability of messages

When putting to a queue, cluster queues send the message to one of the queue managers with active channels connected to your queue manager. On the remote queue manager, if applications used to process the messages are not working, the messages are not processed and wait until the applications start. Similarly, if a queue manager is shut down, any messages on the queue manager are not made available until the queue manager restarts. These instances show lower message availability than when using shared queues.

When using shared queues, any application in the queue sharing group can get messages that are sent. If you shut down one queue manager in the queue sharing group, messages are available to the other queue managers, providing higher message availability than when using cluster queues.

Capacity

A coupling facility is more expensive than a disk; therefore the cost of storing 1,000,000 messages in a local queue is lower than having a coupling facility with enough capacity to store the same number of messages.

Sending to other queue managers

Shared-queue messages are only available within a queue sharing group. If you want to use a queue manager outside of the queue sharing group, you must use channels. You can use clustering to workload balance between multiple remote distributed queue managers.

Workload balancing

You can use clustering to give weight to which channels and queue managers get a proportion of the messages sent. For example, you can send 60% of messages to one queue manager, and 40% of messages to another queue manager. This instance does not depend on the ability of the remote queue manager to process work. The system with the first queue manager might be overloaded, and the system with the second queue manager might be idle, but most of the messages still go to the first queue manager.

With shared queues, two CICS® systems can get messages. If one system is overloaded, the other system takes over most the workload.

Canales de clúster

En cada repositorio completo, puede definir manualmente un canal de clúster receptor y un conjunto de canales de clúster emisor para conectarse a sendos repositorios completos en el clúster. Cuando añada un repositorio parcial, debe definir manualmente un canal de clúster receptor y un canal de clúster emisor individual que se conectarán a uno de los repositorios completos. El clúster definirá automáticamente los canales de clúster emisor adicionales cuando sea necesario. Los canales de clúster emisor definidos automáticamente toman sus atributos de la definición de canal de clúster receptor correspondiente en el gestor de colas receptor.

Canal de clúster receptor: CLUSRCVR

Una definición de canal CLUSRCVR define el extremo de un canal en el que un gestor de colas de clúster puede recibir mensajes de otros gestores de colas del clúster.

Debe definir al menos un canal CLUSRCVR para cada gestor de colas de clúster. Al definir el canal CLUSRCVR, el gestor de colas indica a los otros gestores de colas de clúster que está disponible para recibir mensajes.

Una definición de canal CLUSRCVR permite a otros gestores de colas definir automáticamente las definiciones de canal de clúster emisor correspondientes. Consulte la sección “Canales de clúster emisor definidos automáticamente” en la página 61 de este artículo.

Canal de clúster emisor: CLUSSDR

El administrador debe definir un canal CLUSSDR desde cada gestor de colas de repositorio completo a cualquier otro gestor de colas de repositorio completo del clúster. Todas las actualizaciones intercambiadas por los repositorios completos fluyen exclusivamente en estos canales. Al definir manualmente estos canales, puede controlar la red de repositorios completos de forma explícita.

Cuando añada un gestor de colas de repositorio parcial a un clúster, puede definir manualmente un solo canal CLUSSDR para conectarse a uno de los repositorios completos. Existen pocas diferencias

en el repositorio completo que elija, porque, después de realizar el contacto inicial, los objetos adicionales del gestor de colas del clúster del gestor de colas, incluidos los canales CLUSSDR, se definen automáticamente según sea necesario. Esto permite que el gestor de colas envíe información de clúster a cualquier repositorio completo y enviar mensajes a cualquier gestor de colas del clúster.

Tal como se explica en la sección de este artículo, los canales de emisor definidos automáticamente se basan en la configuración del canal de clúster receptor. Por lo tanto, las propiedades de canal que establezca en los canales de clúster deben definirse de forma idéntica en CLUSSDR y los canales del clúster receptor coincidentes, o sólo en los canales del clúster receptor.

Sólo debe definir manualmente canales CLUSSDR por los motivos descritos anteriormente. Es decir, para conectar inicialmente un repositorio parcial a un repositorio completo, o para conectar dos repositorios completos. La configuración manual de un canal CLUSSDR que se conecta a un repositorio parcial o a un gestor de colas que no está en el clúster, hace que se emitan mensajes de error como AMQ9427 y AMQ9428. Aunque es posible que algunas veces esto sea inevitable como situación temporal, por ejemplo, al modificar la ubicación de un repositorio completo, la definición manual se deberá suprimir lo antes posible.

Canales de clúster emisor definidos automáticamente

Normalmente, cuando añade un gestor de colas de repositorio parcial a un clúster, sólo puede definir manualmente dos canales de clúster en el gestor de colas:

- Un canal de clúster emisor (CLUSSDR) a un gestor de colas de repositorio completo para el clúster.
- Un canal de clúster receptor (CLUSRCVR).

El canal CLUSSDR que defina permitirá que el gestor de colas establezca un contacto inicial con el clúster. Tras realizar el contacto inicial, el clúster definirá automáticamente canales CLUSSDR adicionales cuando sea necesario.

Un canal CLUSSDR definido automáticamente toma sus atributos de la definición de canal CLUSRCVR correspondiente en el gestor de colas receptor. Incluso si hay un canal CLUSSDR definido manualmente, se utilizan los atributos del canal CLUSSDR definido automáticamente. Supongamos, por ejemplo, que define un canal CLUSRCVR sin especificar un número de puerto en el parámetro **CONNAME**, y define manualmente un canal CLUSSDR que especifica un número de puerto. Cuando el canal CLUSSDR definido automáticamente sustituye el que se ha definido manualmente, el número de puerto (tomado del canal CLUSRCVR) queda en blanco. Se utiliza el número de puerto predeterminado y el canal falla.

Cuando existan diferencias de configuración entre un canal CLUSSDR definido manualmente y la definición de canal CLUSRCVR correspondiente, algunas diferencias entrarán en vigor inmediatamente (por ejemplo, los parámetros de equilibrio de carga de trabajo) y otras entrarán en vigor sólo tras reiniciar el canal (por ejemplo, la configuración de TLS).

Para evitar confusiones, en la medida de lo posible tenga en cuenta las directrices siguientes:

- Sólo defina manualmente canales CLUSSDR para que apunten a repositorios completos.
- Cuando haya definido manualmente canales CLUSSDR, configúrelos para que coincidan exactamente con la definición de canal CLUSRCVR correspondiente en el gestor de colas receptor.

Consulte también Trabajar con canales definidos automáticamente.

Conceptos relacionados

Trabajar con canales definidos automáticamente

Cómo trabajar con colas de transmisión de clúster y canales de clúster emisor

Tareas relacionadas

Configurar un nuevo clúster

Añadir un gestor de colas a un clúster

Temas de clúster

Los temas de clúster son temas administrativos con el atributo **cluster** definido. La información sobre temas de clúster se envía a todos los miembros de un clúster y se combina con temas locales para crear partes de un espacio de tema que abarque varios gestores de colas. Esto permite que los mensajes publicados sobre un tema en un gestor de colas se entreguen a las suscripciones de otros gestores de colas del clúster.

Cuando se define un tema de clúster en un gestor de colas, la definición de tema de clúster se envía a los gestores de colas de depósito completo. Los depósitos completos propagan entonces la definición de tema de clúster a todos los gestores de colas del clúster, dejando el mismo tema de clúster disponible para publicadores y suscriptores en cualquier gestor de colas del clúster. El gestor de colas en los que se crea un tema de clúster se conoce como host de tema de clúster. El tema de clúster puede utilizarlo cualquier gestor de colas del clúster, pero las modificaciones de un tema de clúster deben realizarse en el gestor de colas donde se ha definido dicho tema (el host), momento en el cual la modificación se propaga a todos los miembros del clúster a través de los depósitos completos.

Para obtener información sobre cómo configurar temas de clúster para utilizar el *direccionamiento directo* o el *direccionamiento de host de tema* y sobre la herencia de temas de clúster y suscripciones de comodín, consulte Definición de temas de clúster.

Para obtener información sobre los mandatos a utilizar para visualizar temas de clúster, consulte la información relacionada.

Conceptos relacionados

Trabajar con temas administrativos

Trabajar con suscripciones

Referencia relacionada

DISPLAYTOPIC

DISPLAYTPSTATUS

DISPLAYSUB

Objetos de clúster predeterminado

Multi En Multiplatforms, los objetos de clúster predeterminados se incluyen en el conjunto de objetos predeterminados creados automáticamente cuando se define un gestor de colas. z/OS En z/OS, las definiciones de objetos de clúster predeterminadas se pueden encontrar en los ejemplos de personalización.

Nota: Puede modificar las definiciones de canal predeterminadas igual que cualquier otra definición de canal, ejecutando mandatos MQSC o PCF. No altere las definiciones de cola predeterminadas, excepto SYSTEM.CLUSTER.HISTORY.QUEUE.

SYSTEM.CLUSTER.COMMAND.QUEUE
: Cada gestor de colas de un clúster tiene una cola local llamada SYSTEM.CLUSTER.COMMAND.QUEUE que se usa para transferir mensajes al repositorio completo. El mensaje contiene cualquier información nueva o modificada sobre el gestor de colas, o cualquier solicitud de información sobre otros gestores de colas. SYSTEM.CLUSTER.COMMAND.QUEUE normalmente está vacía.

SYSTEM.CLUSTER.HISTORY.QUEUE
: Cada gestor de colas de un clúster tiene una cola local denominada SYSTEM.CLUSTER.HISTORY.QUEUE. SYSTEM.CLUSTER.HISTORY.QUEUE se utiliza para almacenar el historial de información de estado del clúster con fines de servicio.

  En los valores de objeto predeterminados, SYSTEM.CLUSTER.HISTORY.QUEUE se establece en PUT (ENABLED). Para suprimir la recopilación de historial, cambie el valor a PUT (DISABLED).

SYSTEM.CLUSTER.REPOSITORY.QUEUE
: Cada gestor de colas de un clúster tiene una cola local denominada SYSTEM.CLUSTER.REPOSITORY.QUEUE. Esta cola se utiliza para almacenar toda la información del depósito completo. Esta cola normalmente no está vacía.

SYSTEM.CLUSTER.TRANSMIT.QUEUE
Cada gestor de colas tiene una definición para una cola local denominada SYSTEM.CLUSTER.TRANSMIT.QUEUE. SYSTEM.CLUSTER.TRANSMIT.QUEUE es la cola de transmisión predeterminada para todos los mensajes a todas las colas y gestores de colas que están dentro de clústeres. Puede cambiar la cola de transmisión predeterminada para cada canal de clúster emisor a SYSTEM.CLUSTER.TRANSMIT. *ChannelName*, cambiando el atributo de gestor de colas DEFCLXQ . No puede suprimir SYSTEM.CLUSTER.TRANSMIT.QUEUE. También se utiliza para definir comprobaciones de autorización si la cola de transmisión predeterminada que se utiliza es SYSTEM.CLUSTER.TRANSMIT.QUEUE o SYSTEM.CLUSTER.TRANSMIT. *ChannelName*.

SYSTEM.DEF.CLUSRCVR
Cada clúster tiene una definición de canal CLUSRCVR predeterminada llamada SYSTEM.DEF.CLUSRCVR. SYSTEM.DEF.CLUSRCVR se utiliza para proporcionar valores predeterminados para todos los atributos que no especifique al crear un canal de clúster receptor en un gestor de colas del clúster.

SYSTEM.DEF.CLUSSDR
Cada clúster tiene una definición de canal CLUSSDR predeterminada llamada SYSTEM.DEF.CLUSSDR. SYSTEM.DEF.CLUSSDRse utiliza para proporcionar valores predeterminados para todos los atributos que no especifique al crear un canal de clúster emisor en un gestor de colas del clúster.

Conceptos relacionados
Cómo trabajar con objetos de clúster predeterminado

Mensajería de publicación/suscripción

La mensajería de publicación/suscripción le permite separar el proveedor de información de los consumidores de esa información. La aplicación emisora y la aplicación receptor no necesitan saber nada una de la otra para la información que va a enviarse y recibirse.

Antes de que una aplicación IBM MQ de punto a punto pueda enviar un mensaje a otra aplicación, necesita saber algo sobre ella. Por ejemplo, necesita conocer el nombre de la cola a la que se va a enviar la información, y puede ser que también especifique un nombre de gestor de colas.

La publicación/suscripción de IBM MQ elimina la necesidad de que la aplicación sepa algo sobre la aplicación de destino. Todas las aplicaciones emisoras tienen que hacer esto:

- *Colocar* un mensaje IBM MQ que contenga la información que la aplicación desea.
- Asignar el mensaje a un tema que indique el asunto de la información.
- Deje que IBM MQ maneje la distribución de esa información.

Del mismo modo, no es necesario que la aplicación de destino sepa nada sobre el origen de la información que recibe.

La figura siguiente muestra el sistema de publicación/suscripción más simple. Hay un publicador, un gestor de colas, y un suscriptor. El suscriptor realiza una suscripción en un gestor de colas, se envía una publicación desde el publicador al gestor de colas y luego el gestor de colas reenvía la publicación al suscriptor.

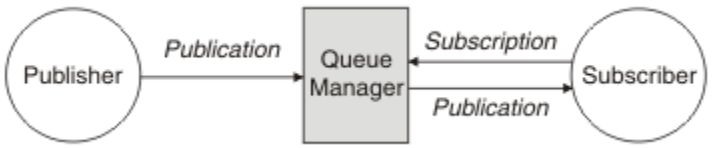


Figura 17. Configuración de publicación/suscripción simple

Un sistema de publicación/suscripción típico tiene más de un publicador y más de un suscriptor en muchos temas diferentes y con frecuencia tiene más de un gestor de colas. Una aplicación puede ser tanto un publicador como un suscriptor.

Otra diferencia significativa entre la mensajería de publicación/suscripción y la de punto a punto es que un mensaje enviado a una cola punto a punto solo es procesado por una única aplicación consumidora. Un mensaje publicado en un tema de publicación/suscripción, donde más de un suscriptor ha registrado un interés, es procesado por cualquier suscriptor interesado.

Componentes de publicación/suscripción

La publicación/suscripción es el mecanismo por el que los suscriptores pueden recibir información, en forma de mensajes, de los publicadores. Los gestores de colas controlan las interacciones entre publicadores y suscriptores utilizando los recursos estándar de IBM MQ.

Un sistema de publicación/suscripción típico tiene más de un publicador y más de un suscriptor en muchos temas diferentes y con frecuencia tiene más de un gestor de colas. Una aplicación puede ser tanto un publicador como un suscriptor.

El proveedor de la información recibe el nombre de *publicador*. Los publicadores suministran información sobre un asunto sin necesidad de saber nada acerca de las aplicaciones que están interesadas en la información. Los publicadores generan esta información en forma de mensajes, denominados *publicaciones*, que quieren publicar y definir el tema de estos mensajes.

El consumidor de la información recibe el nombre de *suscriptor*. Los suscriptores crean *suscripciones* que describen el tema en el que está interesado el suscriptor. Por lo tanto, la suscripción determina qué publicaciones se reenvían al suscriptor. Los suscriptores pueden hacer varias suscripciones y pueden recibir información de muchos publicadores diferentes.

La información publicada se envía en un mensaje de IBM MQ y el asunto de la información se identifica por su *tema*. El publicador especifica el tema cuando publica la información, y el suscriptor especifica los temas sobre los que desea recibir publicaciones. Al suscriptor se le envía información sólo de los temas a los que se ha suscrito.

Es la existencia de temas lo que permite que los proveedores y consumidores de la información que se debe desacoplarse de la mensajería de publicación/suscripción eliminando la necesidad de incluir un destino específico en cada mensaje como se requiere en la mensajería punto a punto.

Todas las interacciones entre publicadores y suscriptores están controladas por un gestor de colas. El gestor de colas recibe los mensajes de los publicadores, y las suscripciones de los suscriptores (a un rango de temas). El trabajo del gestor de colas es direccionar los mensajes publicados para los suscriptores que hayan registrado su interés en el tema de los mensajes.

Los recursos estándar de IBM MQ se utilizan para distribuir mensajes, de modo que las aplicaciones pueden utilizar todas las características que están disponibles para las aplicaciones de IBM MQ existentes. Esto significa que puede utilizar mensajes permanentes para obtener la entrega asegurada una única vez, y que los mensajes pueden ser parte de una unidad transaccional de trabajo para asegurarse de que los mensajes se entregan al suscriptor sólo si son confirmados por el publicador.

Publicadores y publicaciones

En la publicación/suscripción de IBM MQ, un publicador es una aplicación que hace que la información sobre un tema especificado esté disponible para un gestor de colas en forma de un mensaje IBM MQ estándar llamado publicación. Un publicador puede publicar información sobre más de un tema.

Los publicadores utilizan el verbo MQPUT para poner un mensaje en un tema abierto anteriormente, este mensaje es una publicación. A continuación, el gestor de colas local direcciona la publicación a todos los suscriptores que tengan suscripciones al tema de la publicación. Un mensaje publicado puede ser consumido por más de un suscriptor.

Además de distribuir aplicaciones a todos los suscriptores locales que tengan las suscripciones adecuadas, un gestor de colas también puede distribuir la publicación a otros gestores de colas conectados al mismo, directamente o a través de una red de gestores de colas que tengan suscriptores al tema.

En una red de publicación/suscripción de IBM MQ, una aplicación de publicación también puede ser un suscriptor.

Publicaciones bajo punto de sincronismo

Los editores pueden emitir llamadas MQPUT o MQPUT1 en el punto de sincronismo para incluir todos los mensajes entregados a los suscriptores en una unidad de trabajo. Si se ha especificado la opción MQPMO_RETAIN, o las opciones de entrega de temas NPMSGDLV o PMSGDLV con los valores ALL o ALLDUR, el gestor de colas utiliza llamadas MQPUT o MQPUT1 internas en el punto de sincronismo, dentro del ámbito de la llamada MQPUT o MQPUT1 de el publicador.

Información de estado y de sucesos

Las publicaciones se pueden clasificar como publicaciones de estado tales como, por ejemplo, el precio actual de una acción o como publicaciones de sucesos tales como, por ejemplo, el intercambio de dicha acción.

Publicaciones de estado

Las *publicaciones de estado* contienen información sobre el estado actual de algo como, por ejemplo, el precio de una acción o el resultado actual de un partido de fútbol. Cuando sucede algo (por ejemplo, el precio de la acción cambia o el resultado del partido de fútbol cambia), la información de estado anterior ya no es necesaria porque se reemplaza por la nueva información.

Un suscriptor desea recibir la versión actual de la información de estado al iniciarse, y que se le envíe nueva información siempre que el estado cambie.

Si una publicación contiene información de estado, se publica frecuentemente como una publicación retenida. Normalmente un nuevo suscriptor desea la información de estado actual inmediatamente; el suscriptor no desea esperar a que el suceso causante de la información vuelva a publicarse. Los suscriptores recibirán automáticamente la publicación retenida de un tema cuando se suscriban, a menos que el suscriptor utilice las opciones MQSO_PUBLICATIONS_ON_REQUEST o MQSO_NEW_PUBLICATIONS_ONLY.

Publicaciones de sucesos

Las *publicaciones de sucesos* contienen información sobre sucesos específicos que se producen, como intercambiar acciones o marcar un gol. Cada suceso es independiente de los demás.

Un suscriptor desea recibir información sobre sucesos a medida que éstos se producen.

Publicaciones retenidas

De forma predeterminada, después de enviar una publicación a todos los suscriptores interesados, se descarta. Sin embargo, un publicador puede especificar que una copia de una publicación está retenida para que se pueda enviar a suscriptores futuros que estén interesados en el tema.

La supresión de publicaciones después de haberlas enviado a todos los suscriptores interesados es adecuada para la información de sucesos, pero no siempre es adecuada para la información de estado. Al retener un mensaje, los nuevos suscriptores no tienen que esperar a que se publique la información de nuevo antes de recibir información de estado inicial. Por ejemplo, un suscriptor con una suscripción al precio de una acción recibirá el precio actual inmediatamente, sin tener que esperar a que el precio de la acción cambie (y, por consiguiente, vuelva a publicarse).

El gestor de colas puede retener sólo una publicación para cada tema, por lo que la publicación retenida existente de un tema se suprime cuando una nueva publicación retenida llega al gestor de colas. Sin embargo, la supresión de la publicación existente no puede producirse de forma síncrona con la llegada de la nueva publicación retenida. Por lo tanto, siempre que sea posible, no debe tener más de un publicador enviando publicaciones retenidas sobre cualquier tema.

Los suscriptores pueden especificar que no desean recibir publicaciones retenidas mediante la opción de suscripción MQSO_NEW_PUBLICATIONS_ONLY. Los suscriptores existentes pueden solicitar que les sean enviadas copias duplicadas de publicaciones retenidas.

Hay ocasiones en que tal vez no desee retener publicaciones, ni siquiera para la información de estado:

- Si todas las suscripciones a un tema se realizan antes de haber efectuado ninguna publicación sobre ese tema, y no espera o no permite nuevas suscripciones, no es necesario retener publicaciones porque se entregan al conjunto completo de los suscriptores la primera vez que se publiquen.
- Si las publicaciones se producen con frecuencia como, por ejemplo, cada segundo, un nuevo suscriptor (o un suscriptor se recupere de una anomalía) recibe el estado actual casi inmediatamente después de su suscripción inicial, por lo que no es necesario retener estas publicaciones.
- Si las publicaciones son grandes, puede terminar necesitando una cantidad considerable de espacio de almacenamiento para almacenar la publicación retenida para cada tema. En un entorno de varios gestores de colas, las publicaciones retenidas las almacenan todos los gestores de colas de la red que tienen una suscripción coincidente.

A la hora de decidir si se utilizan publicaciones retenidas, considere cómo las aplicaciones suscriptoras pueden recuperarse de una anomalía. Si el publicador no utiliza publicaciones retenidas, la aplicación suscriptora podría tener que almacenar localmente su estado actual.

Para asegurarse de que una publicación está retenida, utilice la opción de colocar mensaje MQPMO_RETAIN. Si se utiliza esta opción y la publicación no se puede retener, el mensaje no se publica y la llamada falla con MQRC_PUT_NOT_RETAINED.

Si un mensaje es una publicación retenida, esto se indica mediante la propiedad de mensaje MQIsRetained. La persistencia de un mensaje es la que era cuando fue publicado originalmente.

Conceptos relacionados

Consideraciones de diseño acerca de las publicaciones retenidas en los clústeres de publicación/suscripción

Publicaciones bajo punto de sincronismo

En la publicación/suscripción de IBM MQ, el punto de sincronización lo pueden utilizar los publicadores o internamente el gestor de colas.

Los publicadores de sincronización al utilizar MQPUT/MQPUT1 pueden emitir llamadas con la opción MQPMO_SYNCPOINT. Todos los mensajes entregados a los suscriptores se cuentan hasta el número máximo de mensajes no confirmados en una unidad de trabajo. El atributo de gestor de colas MAXUMSGS especifica este límite. Si se alcanza el límite, el publicador recibe el código de razón 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_LLEGED.

Cuando un publicador emite llamadas MQPUT/MQPUT1 mediante MQPMO_NO_SYNCPOINT con la opción MQPMO_RETAIN, o las opciones de entrega NPMSGDLV/PMSGDLV con los valores ALL o ALLDUR, el gestor de colas utiliza puntos de sincronismo internos para garantizar que los mensajes se entreguen según se ha solicitado. El publicador puede recibir el código de razón 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_LLEGED si se alcanza el límite dentro del ámbito de la llamada del publicador MQPUT/MQPUT1.

Suscriptores y suscripciones

En la publicación/suscripción de IBM MQ, un suscriptor es una aplicación que solicita información sobre un tema específico a un gestor de colas en una red de publicación/suscripción. Un suscriptor puede recibir mensajes, sobre los mismos temas o temas distintos, de más de un publicador.

Las suscripciones pueden crearse manualmente utilizando un mandato MQSC o por las aplicaciones. Estas suscripciones se emiten para el gestor de colas local y contienen información acerca de las publicaciones que el suscriptor desea recibir:

- El tema en el que está interesado el suscriptor; esto puede resolverse en varios temas si se utilizan comodines.
- Se debe aplicar una serie de selección opcional para los mensajes publicados.
- Un manejador en una cola (conocida como *cola de suscriptores*), en la que deben colocarse las publicaciones seleccionadas y el CorrelId opcional.

El gestor de colas local almacena información de suscripción y cuando recibe una publicación, explora la información para determinar si existe una suscripción que coincida con el tema de la publicación y la serie de selección. Para cada suscripción coincidente, el gestor de colas dirige la publicación a la cola de suscriptores del suscriptor. La información que un gestor de colas almacena sobre las suscripciones se puede visualizar utilizando los mandatos DIS SUB y DIS SBSTATUS.

Una suscripción se suprime sólo cuando se produce uno de los siguientes sucesos:

- El suscriptor cancela la suscripción mediante la llamada MQCLOSE (si la suscripción se realizó como no duradera).
- La suscripción caduca.
- El administrador del sistema suprime la suscripción mediante el mandato DELETE SUB.
- La aplicación suscriptora finaliza (si la suscripción se realizó como no duradera).
- El gestor de colas se detiene o se reinicia (si la suscripción se realizó como no duradera).

Cuando obtenga mensajes, utilice las opciones adecuadas en la llamada MQGET. Si su aplicación solo procesa mensajes para una suscripción, como mínimo, debe utilizar `get-by-correlid`, como se muestra en el programa C de ejemplo `amqssbxa.c` y en el suscriptor MQ no gestionado. El **CorrelId** que se va a utilizar se devuelve de MQSUB en el MQSD.Campo **SubCorrelId** .

Conceptos relacionados

Suscripciones clonadas y compartidas

Referencia relacionada

Ejemplos de cómo definir la propiedad sharedSubscription

Colas gestionadas y publicación/suscripción

Al crear una suscripción, puede elegir utilizar colas gestionadas. Si utiliza colas gestionadas, una cola de suscripciones se crea automáticamente al crear una suscripción. Las colas gestionadas se ordenan automáticamente según la durabilidad de la suscripción. La utilización de colas gestionadas significa que no tiene que preocuparse de la creación de las colas para recibir publicaciones y que las publicaciones no consumidas se eliminan de las colas de suscriptores automáticamente si se cierra una conexión de suscripción no duradera.

Si una aplicación no necesita utilizar una cola en concreto como su cola suscriptor, el destino de las publicaciones que recibe, puede hacer uso de las *suscripciones gestionadas* mediante la opción de suscripción MQSO_MANAGED. Si crea una suscripción gestionada, el gestor de colas devuelve un manejador de objetos al suscriptor para una cola de suscriptores que el gestor de colas crea donde se recibirán publicaciones. Esto se debe a que una *suscripción gestionada* es aquella en la que IBM MQ maneja la suscripción. Se devolverá el manejador de objetos de la cola permitiéndole examinar, obtener o consultar en la cola (no es posible transferir o establecer atributos de una cola gestionada a menos que se dé explícitamente acceso a las colas dinámicas temporales).

La durabilidad de la suscripción determina si la cola gestionada permanece cuando se interrumpe la conexión de la aplicación suscriptora con el gestor de colas.

Las suscripciones gestionadas son especialmente útiles cuando se utilizan con las suscripciones no duraderas porque, cuando la conexión de la aplicación finaliza, los mensajes no consumidos permanecerán en la cola de suscriptores ocupando espacio en el gestor de colas indefinidamente. Si está utilizando una suscripción gestionada, la cola gestionada será una cola dinámica temporal y como tal se suprimirá junto con los mensajes no consumidos cuando se interrumpa la conexión por alguna de las razones siguientes:

- Se utiliza MQCLOSE con MQCO_REMOVE_SUB y se cierra el Hobj gestionado.
- Se pierde una conexión con una aplicación utilizando una suscripción no duradera (MQSO_NON_DURABLE).
- Se elimina una suscripción debido a que ha caducado y se ha cerrado el Hobj gestionado.

También se pueden utilizar suscripciones gestionadas con suscripciones duraderas pero es posible que desee dejar mensajes no consumidos en la cola de suscriptores para que se puedan recuperar cuando se

reabre la conexión. Por este motivo, las colas gestionadas para suscripciones duraderas toman la forma de una cola dinámica permanente y permanecerá cuando se interrumpa la conexión de la aplicación suscriptora con el gestor de colas.

Puede establecer una caducidad en la suscripción si desea utilizar una cola gestionada dinámica permanente, de forma que aunque la cola permanezca existiendo tras rota la conexión, no seguirá existiendo de forma indefinida.

Si se suprime la cola gestionada recibirá un mensaje de error.

Las colas gestionadas que se crean se denominan con números al final (indicaciones horarias) para que cada una sea exclusiva.

Durabilidad de suscripción

Las suscripciones pueden configurarse para ser duraderas o no duraderas. La perdurabilidad de la suscripción determina qué sucede con las suscripciones cuando las aplicaciones suscriptoras se desconectan de un gestor de colas.

Suscripciones duraderas

Las suscripciones duraderas siguen existiendo cuando se cierra la conexión con el gestor de colas de la aplicación de suscripción. Si una suscripción es duradera, cuando la aplicación de suscripción se desconecta, la suscripción permanece en su lugar y la puede utilizar la aplicación de suscripción cuando se vuelva a conectar solicitando la suscripción de nuevo utilizando el **SubName** que se devolvió cuando se creó la suscripción.

Cuando se suscribe de forma duradera, un nombre de suscripción (**SubName**) es obligatorio. Los nombres de suscripción deben ser exclusivos en un gestor de colas de modo que pueda utilizarse para identificar una suscripción. Este medio de identificación es necesario al especificar una suscripción que se desea reanudar, si ha cerrado deliberadamente la conexión con la suscripción (utilizando la opción MQCO_KEEP_SUB) o si se ha desconectado del gestor de colas. Puede reanudar una suscripción existente utilizando la llamada MQSUB con la opción MQSO_RESUME. Los nombres de suscripción también se visualizan si utiliza el mandato DISPLAY SBSTATUS con SUBTYPE ALL o ADMIN.

Cuando una aplicación ya no requiere una suscripción duradera, puede eliminarse utilizando la llamada de función MQCLOSE con la opción MQCO_REMOVE_SUB o se puede suprimir manualmente mediante el mandato MQSC DELETE SUB.

Puede utilizar el atributo de tema **DURSUB** para especificar si las suscripciones duraderas pueden realizarse en un tema o no.

En la devolución de una llamada MQSUB mediante la opción MQSO_RESUME, la caducidad de la suscripción se ha establecido en la caducidad original de la suscripción y no en la hora de caducidad restante.

Un gestor de colas continúa enviando publicaciones para satisfacer una suscripción duradera aunque la aplicación de suscriptor no esté conectada. Esto conduce a una acumulación de mensajes en la cola de suscriptores. La forma más sencilla de evitar este problema es utilizar una suscripción no duradera cuando sea adecuado. No obstante, donde sea necesario utilizar suscripciones duraderas, puede evitarse una acumulación de mensajes si el suscriptor se suscribe utilizando la opción Publicaciones retenidas. Un suscriptor puede controlar cuándo recibe publicaciones utilizando la llamada MQSUBRQ.

Suscripciones no duraderas

Las suscripciones no duraderas sólo existen mientras permanece abierta la conexión de la aplicación suscriptora al gestor de colas. La suscripción se elimina cuando la aplicación suscriptora se desconecta del gestor de colas deliberadamente o por pérdida de la conexión. Cuando se cierra la conexión, la información sobre la suscripción se elimina del gestor de colas y ya no se muestra si se muestran suscripciones mediante el mandato DISPLAY SBSTATUS. No se envían más mensajes a la cola de suscriptores.

Lo que le sucede con cualquier publicación no consumida en la cola de suscriptores para las suscripciones no duraderas se determina de la manera siguiente.

- Si una aplicación de suscripción utiliza un destino gestionado, las publicaciones que no se hayan consumido se eliminan automáticamente.
- Si la aplicación suscriptora proporciona un manejador para su propia cola de suscriptores cuando se suscribe, los mensajes no consumidos no se eliminan automáticamente. Es responsabilidad de la aplicación borrar la cola si es lo pertinente. Si la cola es compartida por más de un suscriptor u otras aplicaciones punto a punto, puede que no sea adecuado borrar la cola por completo.

Aunque no es necesario para las suscripciones no duraderas, el gestor de colas utiliza un nombre de suscripción si se ha proporcionado uno. Los nombres de suscripción deben ser exclusivos en el gestor de colas de modo que pueda utilizarse para identificar una suscripción.

Conceptos relacionados

Suscripciones clonadas y compartidas

Tareas relacionadas

Utilización de suscripciones compartidas de JMS 2.0

Referencia relacionada

Ejemplos de cómo definir la propiedad sharedSubscription

Series de selección

Una *serie de selección* es una expresión que se aplica a una publicación para determinar si coincide con una suscripción. Las series de selección pueden incluir caracteres comodín.

Al suscribirse, además de especificar un tema, puede especificar una serie de selección para seleccionar publicaciones en función de sus propiedades de mensaje.

La serie de selección se evalúa contra el mensaje como transferidos por el publicador antes de que se modifique para entregar a cada suscriptor. Tenga en cuenta que al usar campos en la serie de selección, es posible que se modifiquen como parte de la operación de publicación. Por ejemplo, los campos MQMD `UserIdentifier`, `MsgId`y `CorrelId`.

Las series de selección no deben hacer referencia a ninguno de los campos de propiedad del mensaje añadidos por el gestor de colas como parte de la operación de publicación (consulte Propiedades del mensaje de publicación/suscripción), excepto para la propiedad del mensaje `MQTopicString`, que contiene la serie del tema para la publicación.

Conceptos relacionados

Restricciones y reglas de series de selección

Temas

Un tema es el asunto de la información que se publica en un mensaje de publicación/suscripción.

Los mensajes en los sistemas de punto a punto se envían a una dirección de destino específica. Los mensajes de sistemas de publicación/suscripción basados en el asunto se envían a los suscriptores basándose en el asunto que describe el contenido del mensaje. En los sistemas basados en contenido, los mensajes se envían a los suscriptores basándose en el contenido del mensaje propiamente dicho.

El sistema de publicación/suscripción de IBM MQ es un sistema de publicación/suscripción basado en temas. Un publicador crea un mensaje y lo publica con una serie de tema que se ajusta mejor al asunto de la publicación. Para recibir publicaciones, un suscriptor crea una suscripción con una serie de tema que coincide con un patrón para seleccionar temas de publicación. El gestor de colas entrega publicaciones a los suscriptores que tienen suscripciones que coinciden con el tema de publicación y tienen autorización para recibir las publicaciones. El artículo “Series de tema” en la página 70 describe la sintaxis de series de temas que identifican el asunto de una publicación. Los suscriptores también crean series de temas para seleccionar los temas que se van a recibir. Las series de temas que los suscriptores crean pueden contener uno de los dos esquemas comodín alternativos que coincida con el patrón con las series de temas en las publicaciones. La coincidencia de patrón se describe en “Esquemas de comodín” en la página 71.

En la publicación/suscripción basada en el asunto, los publicadores o administradores son responsables de clasificar los asuntos en temas. Los asuntos se organizan normalmente de forma jerárquica, en árboles de temas, utilizando el carácter ' / ' para crear subtemas en la serie de tema. Consulte “Árboles de temas” en la página 77 para ver ejemplos de árboles de temas. Los temas son nodos del árbol de temas. Los temas pueden ser nodos hoja sin subtemas, o nodos intermedios con subtemas.

En paralelo con la organización de asuntos en un árbol de temas jerárquico, puede asociar los temas con objetos de temas administrativos. Asigne atributos a un tema, como si el tema es distribuido en un clúster, asociándolo con un objeto de tema administrativo. La asociación se realiza dando nombre al tema utilizando el atributo TOPICSTR del objeto de tema administrativo. Si no asocia explícitamente un objeto de tema administrativo a un tema, el tema hereda los atributos de su antecesor más cercano en el árbol de temas que *tenga* asociado a un objeto de tema administrativo. Si no ha definido ningún padre, lo hereda de SYSTEM.BASE.TOPIC. Los objetos de tema administrativo se describen en “Objetos de tema administrativo” en la página 78.

Nota: Aunque herede todos los atributos de un tema de SYSTEM.BASE.TOPIC, defina un tema raíz para los temas que se hereda directamente de SYSTEM.BASE.TOPIC. Por ejemplo, en el espacio de temas de los estados de EE.UU., USA/Alabama USA/Alaska, etc., USA es el tema raíz. La finalidad principal del tema raíz es crear espacios de tema discretos y no solapados para evitar que las publicaciones coincidan con las suscripciones erróneas. También significa que puede cambiar los atributos del tema raíz para afectar a todo el espacio de temas. Por ejemplo, puede establecer el nombre del atributo **CLUSTER**.

Cuando hace referencia a un tema como un publicador o un suscriptor, tiene la opción de suministrar una serie de tema o hacer referencia a un objeto de tema. O bien puede hacer ambas cosas, en cuyo caso la serie de tema que suministre define un subtema del objeto de tema. El gestor de colas identifica el tema añadiendo la serie de tema al prefijo de serie de tema especificado en el objeto de tema, insertando un ' / ' adicional entre las dos series de tema, por ejemplo, *serie de tema/serie de objeto*. “Combinación de series de tema” en la página 75 lo describe de forma más detallada. La serie de tema resultante se utiliza para identificar el tema y asociarlo con un objeto de tema administrativo. El objeto de tema administrativo no es necesariamente el mismo objeto de tema que el objeto de tema correspondiente al tema maestro.

En la publicación/suscripción basada en contenido, defina qué mensajes desea recibir proporcionando series de selección que busquen en el contenido de todos los mensajes. IBM MQ ofrece una forma intermedia de publicación/suscripción basada en contenidos utilizando selectores de mensajes que exploran propiedades de mensajes en vez del contenido completo del mensaje; consulte Selectores. El uso arquetípico de selectores de mensajes es suscribirse a un tema y, a continuación, calificar la selección con una propiedad numérica. El selector le permite especificar que está interesado en los valores sólo en un rango determinado; algo que no se puede hacer con comodines basados en caracteres o en temas. Si tiene que filtrar según el contenido completo del mensaje, debe utilizar IBM Integration Bus.

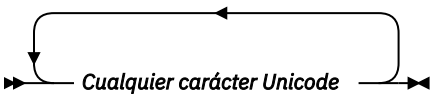
Series de tema

Etiquetar información que se publica como tema utilizando una serie de tema. Suscríbase a grupos de temas utilizando series de temas comodín basados en caracteres o temas.

Temas

Una *serie de tema* es una serie de caracteres que identifica el tema de un mensaje de publicación/suscripción. Puede utilizar cualquier carácter que desee al construir una serie de tema.

Serie de tema



Tres caracteres tienen un significado especial en la publicación/suscripción de IBM WebSphere MQ 7. Están permitidos en cualquier lugar de una serie de tema, pero debe utilizarlos con precaución. El uso de los caracteres especiales se explica en “Esquema de comodín basado en temas” en la página 72.

Una barra inclinada (/)

El separador de nivel de tema. Utilice el carácter '/' para estructurar el tema en un árbol de temas.

Evite niveles de temas vacíos, '//', si puede. Corresponden a nodos en la jerarquía de temas sin series de temas. Un '/' inicial o final en una serie de tema corresponde a un nodo vacío inicial o final y también debe evitarse.

El signo de hash (#)

Se utiliza en combinación con '/' para construir un carácter comodín multinivel en las suscripciones. Tenga cuidado al utilizar '#' junto a '/' en series de temas utilizadas para referirse a temas publicados. "Ejemplos de series de temas" en la página 71 muestra un uso racional de '#'.

Las series '.../#/...', '#/...' y '.../#' tienen un significado especial en las series de temas de suscripción. Las series coinciden con todos los temas en uno o varios niveles de la jerarquía de temas. Por lo tanto, si ha creado un tema con una de esas secuencias, no podría suscribirse a él sin tener que suscribirse también a todos los temas en varios niveles de la jerarquía de temas.

El signo más (+)

Se utiliza en combinación con '/' para construir un carácter comodín de un solo nivel en las suscripciones. Tenga cuidado al utilizar '+' junto a '/' en series de temas utilizadas para referirse a temas publicados.

Las series '.../+/...', '+/...' y '.../+' tienen un significado especial en las series de temas de suscripción. Las cadenas coinciden con todos los temas en un nivel de la jerarquía de temas. Por lo tanto, si ha creado un tema con una de esas secuencias, no podría suscribirse a él sin tener que suscribirse también a todos los temas en un nivel de la jerarquía de temas.

Ejemplos de series de temas

```
IBM/Business Area#/Results
IBM/Diversity/%African American
```

Referencia relacionada

TOPIC

Esquemas de comodín

Existen dos esquemas comodín que se utilizan para suscribirse a varios temas. La elección del esquema es una opción de suscripción.

MQSO_WILDCARD_TOPIC

Seleccione los temas a los que se va a suscribir mediante el esquema comodín basado en temas.

Este es el valor predeterminado si no se selecciona explícitamente ningún esquema comodín.

MQSO_WILDCARD_CHAR

Seleccione los temas a los que se va a suscribir mediante el esquema comodín basado en caracteres.

Establezca el esquema especificando el parámetro **wschema** en el mandato DEFINE SUB. Para obtener más información, consulte DEFINE SUB.

Nota: Las suscripciones que se han creado antes de IBM WebSphere MQ 7.0 siempre utilizan el esquema comodín basado en caracteres.

Ejemplos

```
IBM/+/Results
#/Results
IBM/Software/Results
IBM/*ware/Results
```

Esquema de comodín basado en temas
Los comodines basados en temas permiten a los suscriptores suscribirse a más de un tema a la vez.

Los comodines basados en temas son una característica potente del sistema de temas en la publicación/suscripción de IBM MQ. El comodín multinivel y el comodín de un solo nivel pueden utilizarse para suscripciones, pero el publicador de un mensaje no puede utilizarlos dentro de un tema.

El esquema de comodín basado en temas le permite seleccionar publicaciones agrupadas por nivel de tema. En cada nivel de la jerarquía de temas, se puede elegir si la cadena de la suscripción del nivel de tema tiene que coincidir exactamente con la cadena de la publicación o no. Por ejemplo, la suscripción `IBM/+/Results` selecciona todos los temas,

```
IBM/Software/Results
IBM/Services/Results
IBM/Hardware/Results
```

Hay dos tipos de comodines.

Comodín multinivel

- El comodín multinivel se utiliza en suscripciones. Cuando se utiliza en una publicación se trata como un literal.
- El carácter comodín multinivel `'#'` se utiliza para buscar coincidencias con cualquier número de niveles dentro de un tema. Por ejemplo, al utilizar el árbol de temas de ejemplo, si se suscribe a `'USA/Alaska/#'`, recibirá mensajes sobre los temas `'USA/Alaska'` y `'USA/Alaska/Juneau'`.
- El comodín multinivel puede representar cero o más niveles. Por consiguiente, `'USA/#'` también puede coincidir con el singular `'USA'`, donde `'#'` representa cero niveles. El separador de nivel de tema no tiene sentido en este contexto ya que no hay ningún nivel para separar.
- El comodín multinivel sólo es eficaz cuando se especifica solo o junto al carácter separador de nivel de tema. Por lo tanto, `'#'` y `'USA/#'` son temas válidos cuando el carácter `'#'` se trata como un comodín. Sin embargo, aunque `'USA#'` también es una serie de tema válida, el carácter `'#'` no se considera un comodín y no tiene ningún significado especial. Para obtener más información, consulte “Cuando los comodines basados en temas no son tales” en la página 74.

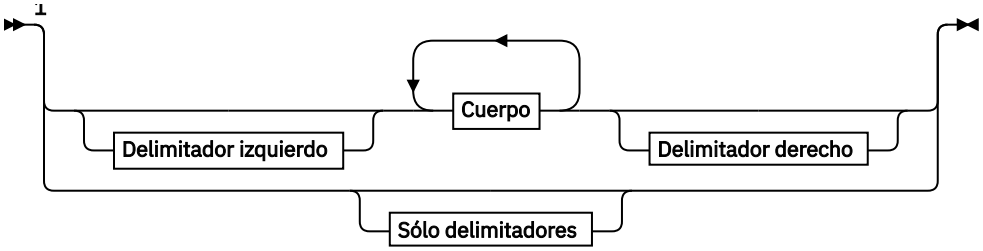
Comodín de un solo nivel

- El comodín de un solo nivel se utiliza en suscripciones. Cuando se utiliza en una publicación se trata como un literal.
- El carácter comodín de un solo nivel `'+'` coincide con un nivel de tema, y sólo uno. Por ejemplo, `'USA/+'` coincide con `'USA/Alabama'`, pero no con `'USA/Alabama/Auburn'`. Dado que el comodín de un solo nivel coincide con un solo nivel, `'USA/+'` no coincide con `'USA'`.
- El comodín de un solo nivel puede utilizarse en cualquier nivel del árbol de temas, y junto con el comodín multinivel. El comodín de un solo nivel debe especificarse junto al separador de nivel de tema, excepto en el caso de que se especifique solo. Por lo tanto, `'+'` y `'USA/+'` son temas válidos cuando el carácter `'+'` se trata como un comodín. Sin embargo, aunque `'USA+'` también es una serie de tema válida, el carácter `'+'` no se considera un comodín y no tiene ningún significado especial. Para obtener más información, consulte “Cuando los comodines basados en temas no son tales” en la página 74.

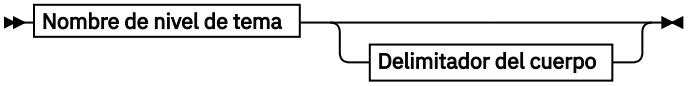
La sintaxis del esquema de comodín basado en temas no tiene caracteres de escape. Si `'#'` y `'+'` se tratan como comodines o no, depende de su contexto. Consulte “Cuando los comodines basados en temas no son tales” en la página 74 para obtener más información.

Nota: El inicio y el fin de una serie de tema se tratan de una manera especial. Utilizando `'$'` para indicar el final de la serie, `'$#/...'` es un comodín multinivel y `'$/#/..'`. es un nodo vacío en la raíz, seguido de un comodín multinivel.

Serie de comodín basado en temas

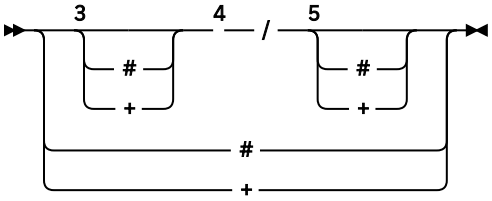


Cuerpo



Nombre de nivel de tema

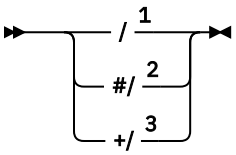
Sólo delimitadores



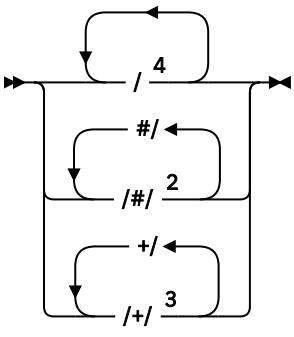
Notas:

[1] Una serie de tema de longitud cero o nula no es válida
[2] Se recomienda no utilizar ninguno de los `*, ?, %` en las series de nombre de nivel para la compatibilidad entre esquemas de comodín basados en caracteres y basados en temas.
[3] Estos casos son equivalentes al patrón de *delimitador izquierdo*.
[4] / sin comodines coincide con un solo tema vacío.
[5] Estos casos son equivalentes al patrón de *delimitador derecho*.
[6] Coincide con todos los temas.
[7] Coincide con todos los temas en los que sólo hay un nivel.

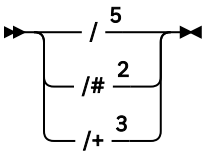
Delimitador izquierdo



Delimitador de cuerpo



Delimitador derecho



Notas:

[1] La serie de tema empieza con un tema vacío
[2] Coincide con cero o más niveles. Varias series de coincidencias multinivel tienen el mismo efecto que una serie de coincidencias multinivel.
[3] Coincide exactamente con un nivel.
[4] // es un tema vacío: un objeto de tema sin serie de tema.
[5] La serie de tema finaliza con un tema vacío

Cuando los comodines basados en temas no son tales

Los caracteres comodín '+' y '#' no tienen ningún significado especial cuando están mezclados con otros caracteres (incluidos ellos mismos) en un nivel de tema.

Esto significa que se pueden publicar los temas que contienen '+' o '#' junto con otros caracteres en un nivel de tema.

Por ejemplo, considere los dos temas siguientes:

1. level0/level1/+/level4/#
2. level0/level1/#+/level4/level#

En el primer ejemplo, los caracteres '+' y '#' se tratan como comodines y, por tanto, no son válidos en una serie de tema en el que debe publicarse, pero son válidos en una suscripción.

En el segundo ejemplo, los caracteres '+' y '#' no se tratan como comodines y, por lo tanto, la serie de tema puede publicarse y es posible suscribirse a ella.

Ejemplos

```
IBM/+/Results
#/Results
IBM/Software/Results
```

Esquema de comodín basado en caracteres
El esquema de comodín basado en caracteres permite seleccionar temas en base a la coincidencia de caracteres tradicional.

Puede seleccionar todos los temas en varios niveles de una jerarquía de temas utilizando la serie '*'. Utilizar '*' en el esquema de comodín basado en temas es equivalente a utilizar la serie de comodín basada en temas '#'

' *x*/*/*y* ' es equivalente a ' *x*/#/*y* ' en el esquema basado en temas y selecciona todos los temas de la jerarquía de temas entre los niveles ' *x* y *y* ', donde ' *x* ' y ' *y* ' son nombres de tema que no están en el conjunto de niveles devueltos por el comodín.

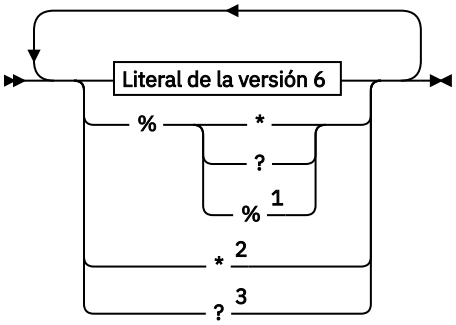
'/+/' en el esquema basado en temas no tiene un equivalente exacto en el esquema basado en caracteres. 'IBM/*/Results' también seleccionaría 'IBM/Patents/Software/Results'. Puede construir siempre consultas con los dos esquemas que dan lugar a coincidencias exactas sólo si el conjunto de nombres de tema en cada nivel de la jerarquía es exclusivo.

Utilizados de forma general, '*' y '?' en el esquema basado en caracteres no tienen equivalente en el esquema basado en temas. El esquema basado en temas no realiza la coincidencia parcial

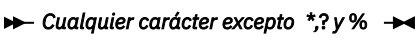
mediante comodines. La suscripción de comodines basad en caracteres `'IBM/*ware/Results'` no tiene equivalente basado en temas.

Nota: Las coincidencias que utilizan suscripciones de comodines basadas en caracteres son más lentas que las coincidencias que utilizan suscripciones basadas en temas.

Serie de comodines basada en caracteres



Literal de la versión 6



Notas:

1. Significa escapar el siguiente carácter, para tratarlo como un literal. `'%'` debe ir seguido de `'*'`, `'?'` o `'%'`. Consulte "Ejemplos de series de temas" en la página 71.
2. Significa la coincidencia con ninguno o más caracteres en una suscripción.
3. Significa la coincidencia exacta de un carácter en una suscripción.

Ejemplos

```
IBM/*/Results
IBM/*ware/Results
```

Combinación de series de tema

Cuando se crean suscripciones o se abren temas para que pueda publicar mensajes en ellos, la serie de tema se puede formar combinando dos series de subtema o "subtemas" separados. La aplicación o el mandato administrativo proporcionan un subtema como serie de tema y el otro es la serie de tema asociada a un objeto de tema. Puede utilizar cualquiera de los dos subtemas como tema de serie único o bien combinarlos para formar un nombre de tema nuevo.

Por ejemplo, cuando se define una suscripción utilizando el tema MQSC **DEFINE SUB**, el mandato puede adoptar como atributo **TOPICSTR** (serie de tema) o **TOPICOBJ** (objeto de tema) o bien ambos. Si sólo se proporciona **TOPICOBJ**, la serie de tema asociada a dicho objeto de tema se utiliza como serie de tema. Si sólo se proporciona **TOPICSTR**, se utiliza como serie de tema. Si se proporcionan ambos, se concatenan para formar una única serie de tema con el formato **TOPICOBJ** / **TOPICSTR**, donde la serie de tema **TOPICOBJ** configurada siempre es la primera y las dos partes de la serie siempre van separadas por un carácter "/".

Asimismo, en un programa MQI, el nombre de tema completo se crea mediante MQOPEN. Está compuesto por los campos utilizados en las llamadas de MQI de publicación/suscripción en el orden listado:

1. El atributo **TOPICSTR** del objeto de tema, nombrado en el campo **ObjectName**.
2. El parámetro **ObjectString** que define el subtema que la aplicación suministra.

La serie de tema resultante se devuelve en el parámetro **ResObjectString**.

Se considera que estos campos están presentes si el primer carácter de cada campo no es un valor en blanco o un carácter nulo y la longitud de campo es mayor que cero. Si sólo uno de los campos está

presente, se utiliza sin modificaciones como nombre del tema. Si ninguno de los campos tiene un valor, la llamada no se ejecuta correctamente y genera el código de razón `MQRC_UNKNOWN_OBJECT_NAME` o `MQRC_TOPIC_STRING_ERROR` si el nombre de tema completo no es válido.

Si ambos campos están presentes, se inserta un carácter "/" entre los dos elementos del nombre de tema combinado resultante.

La tabla siguiente muestra ejemplos de concatenación de series de tema:

Tabla 2. Ejemplo de concatenación de series de tema

TOPICSTR del objeto de tema	**Serie de tema proporcionada por la aplicación o el mandato DEFINE SUB**	**Nombre de tema completo**	**Comentario**
Fútbol/Resultados	`' '`	Fútbol/Resultados	El TOPICSTR del objeto de tema se utiliza sin nada más.
`' '`	Fútbol/Resultados	Fútbol/Resultados	El ObjectString/ TOPICSTR del objeto de tema se utiliza sin nada más.
Fútbol	Resultados	Fútbol/Resultados	Se añade un carácter "/" en el punto de concatenación.
Fútbol	/Resultados	Fútbol//Resultados	Se produce un 'nodo vacío' entre las dos series. Esto es diferente de "Fútbol/Resultados".
/Fútbol	Resultados	/Fútbol/Resultados	El tema empieza por un 'nodo vacío'. Esto es diferente de "Fútbol/ Resultados".

Se considera que el carácter "/" es un carácter especial que proporciona la estructura para el nombre de tema completo en “Árboles de temas” en la página 77. El carácter "/" no se debe utilizar con ninguna otra finalidad, sino la estructura del árbol de tema se verá afectada. El tema "`/Football`" no es el mismo que el tema "`Football`".

Nota: Si utiliza un objeto de tema al crear una suscripción, el valor de la serie de tema del objeto de tema se fija en la suscripción durante la definición. Los cambios posteriores en el objeto de tema no afectan a la serie de tema en la que se define la suscripción.

Caracteres comodín en las series de tema

Los siguientes caracteres comodín son caracteres especiales:

- signo más (+)
- signo de número (#)
- asterisco (*)
- signo de interrogación (?)

Los caracteres comodín sólo tienen un significado especial cuando se utilizan en una suscripción. Estos caracteres no se consideran como no válidos cuando se utilizan en otros lugares, sin embargo, debe asegurarse de que comprende cómo se utilizan y tal vez prefiera no utilizarlos en series de tema al publicar o definir objetos de tema.

Si publica en una serie de tema con # o + junto con otros caracteres (incluidos ellos mismos) dentro de un nivel de tema, la serie de tema se puede suscribir a cualquiera de los dos esquemas comodín.

Si publica en una serie de tema con # o + como únicos caracteres entre dos caracteres /, una aplicación no se podrá suscribir de forma explícita a una serie de tema utilizando el esquema de comodín MQSO_WILDCARD_TOPIC. El resultado de esta situación es que la aplicación obtiene más publicaciones de lo previsto.

No debe utilizar un carácter comodín en la serie de tema de un objeto de tema definido. Si lo hace, el carácter se tratará como un carácter literal cuando el objeto lo utilice un publicador y como carácter comodín cuando lo utilice una suscripción. Esto puede llevar a la confusión.

Fragmento de código de ejemplo

Este fragmento de código, extraído del programa de ejemplo, Ejemplo 2: Publicador para una serie de tema combina un objeto de tema con una serie de tema variable:

```
MQOD   td = {MQOD_DEFAULT}; /* Object Descriptor      */
td.ObjectType = MQOT_TOPIC;  /* Object is a topic      */
td.Version = MQOD_VERSION_4;  /* Descriptor needs to be V4  */
strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
td.ObjectString.VSPtr = topicString;
td.ObjectString.VSLength = (MQLONG)strlen(topicString);
td.ResObjectString.VSPtr = resTopicStr;
td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
```

Árboles de temas

Cada tema que se define es un elemento, o nodo, del árbol de temas. El árbol de temas puede estar vacío para empezar o puede contener temas que se hayan definido anteriormente utilizando mandatos MQSC o PCF. Puede definir un nuevo tema utilizando los mandatos crear tema o especificando el tema por primera vez en una publicación o suscripción.

Aunque puede utilizar cualquier serie de caracteres para definir la serie de tema de un tema, es aconsejable elegir una serie de tema que se ajuste a una estructura de árbol jerárquica. Un diseño cuidadoso de series de temas y árboles de temas puede ser de ayuda en las siguientes operaciones:

- Suscribirse a varios temas.
- Establecer políticas de seguridad.

Aunque puede construir un árbol de temas como una estructura plana lineal, es mejor crear un árbol de temas en una estructura jerárquica con uno o más temas raíz. Para obtener más información sobre la planificación de seguridad y los temas, consulte Seguridad de publicación/suscripción.

La Figura 18 en la página 77 muestra un ejemplo de un árbol de temas con un tema raíz.

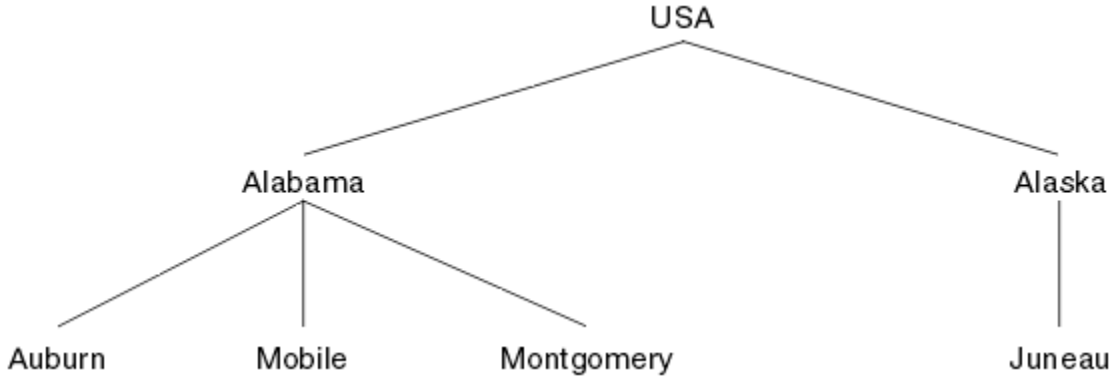


Figura 18. Ejemplo de un árbol de temas

Cada serie de caracteres de la figura representa un nodo en el árbol de temas. Una serie de tema completa se crea agregando nodos de uno o más niveles del árbol de temas. Los niveles se separan

mediante el carácter "/". El formato de una serie de tema especificada al completo es: "raíz/nivel2/nivel3".

Los temas válidos del árbol de temas mostrado en la Figura 18 en la página 77 son:

"USA"
"USA/Alabama"
"USA/Alaska"
"USA/Alabama/Auburn"
"USA/Alabama/Mobile"
"USA/Alabama/Montgomery"
"USA/Alaska/Juneau"

Cuando diseñe series de temas y árboles de temas, recuerde que el gestor de colas no interpreta la serie de tema propiamente dicha ni intenta deducir un significado de la misma. Simplemente utiliza la serie de tema para enviar mensajes seleccionados a los suscriptores de ese tema.

Los siguientes principios se aplican a la creación y el contenido de un árbol de temas:

- No hay ningún límite en cuanto al número de niveles de un árbol de temas.
- No hay ningún límite en cuanto a la longitud del nombre de un nivel de un árbol de temas.
- Puede haber un número ilimitado de nodos "raíz"; es decir, puede haber un número ilimitado de árboles de temas.

Tareas relacionadas

Reducción del número de temas no deseados en el árbol de temas

Objetos de tema administrativo

Mediante un objeto de tema administrativo puede asignar a los temas atributos específicos que no son predeterminados.

En la Figura 19 en la página 78 se muestra cómo un tema de alto nivel de `Sport` dividido en distintos temas que cubren distintos deportes puede visualizarse como árbol de temas:

Figura 19. Visualización de un árbol de temas

En la Figura 20 en la página 78 se muestra cómo el árbol de temas puede dividirse todavía más para separar distintos tipos de información sobre cada deporte:

Figura 20. Árbol de temas ampliado

Para crear el árbol de temas ilustrado, no es necesario definir ningún objeto de tema administrativo. Cada uno de los nodos de este árbol se define mediante una serie de tema creada en una operación de publicación o suscripción. Cada tema del árbol hereda los atributos de su padre. Los atributos se heredan del objeto de tema padre, porque, de forma predeterminada, todos los atributos están establecidos en

ASPARENT. En este ejemplo, cada tema tiene los mismos atributos que el tema `Sport`. El tema `Sport` no tiene ningún objeto de tema administrativo y hereda sus atributos de SYSTEM.BASE.TOPIC.

Tenga en cuenta que no se recomienda otorgar autorizaciones a los usuarios no mqm en el nodo raíz del árbol de temas, que es SYSTEM.BASE.TOPIC, ya que las autorizaciones se heredan, pero no pueden restringirse. Por lo tanto, si se otorgan autorizaciones en este nivel, se otorgan autorizaciones a todo el árbol. Debe otorgar la autorización en un nivel de tema más abajo en la jerarquía.

Los objetos de temas administrativos pueden utilizarse para definir atributos específicos para nodos concretos en el árbol de temas. En el ejemplo siguiente, el objeto de tema administrativo se define para establecer la propiedad de suscripciones duraderas DURSUB del tema de fútbol en el valor NO:

```
DEFINE TOPIC(FOOTBALL.EUROPEAN)
TOPICSTR('Sport/Soccer')
DURSUB(NO)
DESCR('Administrative topic object to disallow durable subscriptions')
```

El árbol de temas ahora se puede visualizar como:

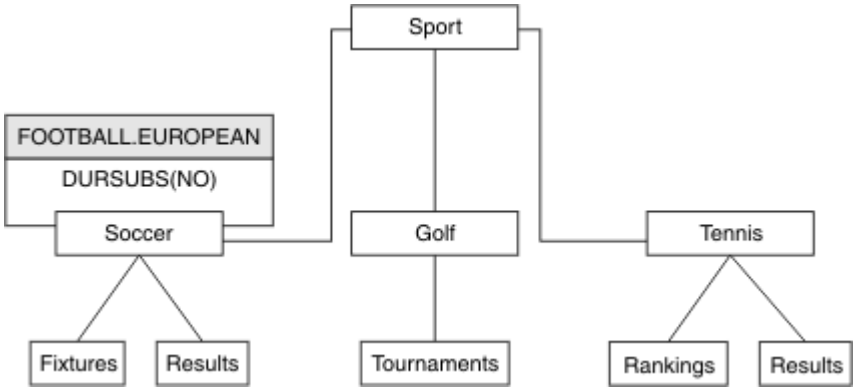


Figura 21. Visualización de un objeto de tema administrativo asociado con el tema Sport/Soccer

Las aplicaciones que se suscriban a temas por debajo de Soccer en el árbol pueden seguir utilizando las series de tema que utilizaban antes de añadir el objeto de tema administrativo. Sin embargo, ahora se puede escribir una aplicación para suscripción utilizando el nombre de objeto `FOOTBALL.EUROPEAN` en lugar de la serie `/Sport/Soccer`. Por ejemplo, para suscribir en `/Sport/Soccer/Results`, una aplicación puede especificar `MQSD.ObjectName` como `FOOTBALL.EUROPEAN` y `MQSD.ObjectString` como `Results`.

Con esta característica, puede ocultar una parte del árbol de temas ante los desarrolladores de aplicaciones. Defina un objeto de tema administrativo en un nodo determinado del árbol de temas, de modo que los desarrolladores de aplicaciones puedan definir sus propios temas como hijos del nodo. Los desarrolladores deben tener conocimiento sobre el tema padre, pero no sobre los demás nodos del árbol padre.

Herencia de atributos

Si un árbol de temas tiene muchos objetos de temas administrativos, de forma predeterminada, cada objeto de tema administrativo hereda sus atributos de su tema administrativo padre más cercano. El ejemplo anterior se ha ampliado en Figura 22 en la página 80:

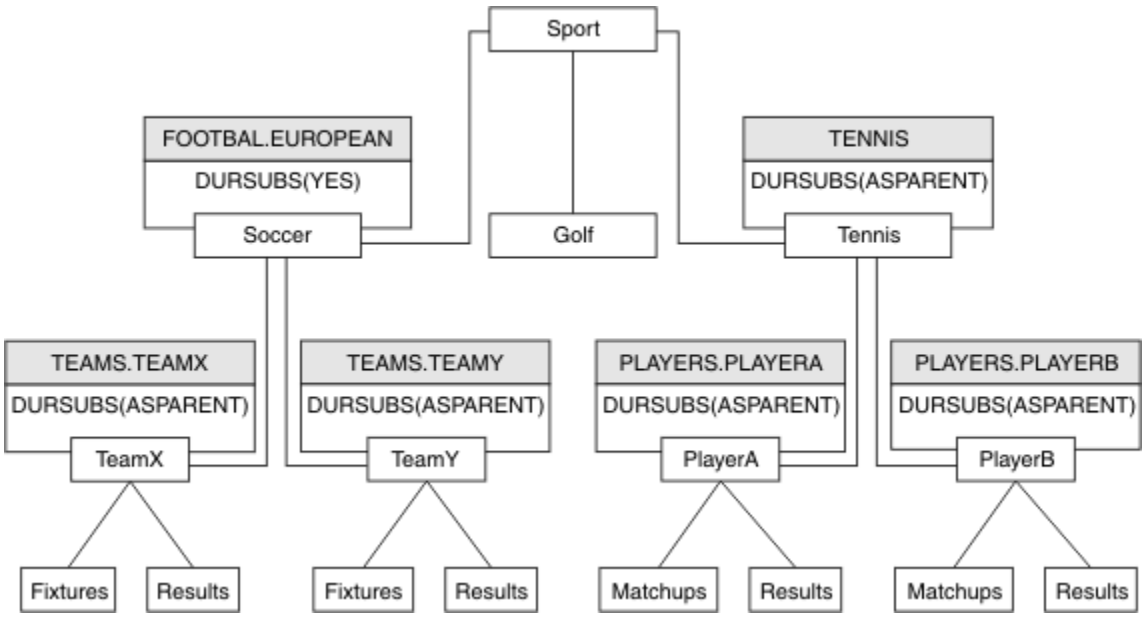


Figura 22. Árbol de temas con varios objetos de temas administrativos

Por ejemplo, utilice la herencia para asignar a todos los temas hijo de `/Sport/Soccer` la propiedad que establece que las suscripciones no son duraderas. Cambie el atributo DURSUB de `FOOTBALL.EUROPEAN` a `NO`.

Este atributo puede establecerse utilizando el siguiente mandato:

```
ALTER TOPIC(FOOTBALL.EUROPEAN) DURSUB(NO)
```

Todos los objetos de tema administrativo de temas hijo de `Sport/Soccer` tienen la propiedad DURSUB establecida en el valor predeterminado ASPARENT. Después de cambiar el valor de la propiedad DURSUB de `FOOTBALL.EUROPEAN` a `NO`, los temas hijo de `Sport/Soccer` heredan el valor de la propiedad DURSUB `NO`. Todos los temas hijo de `Sport/Tennis` heredan el valor de DURSUB del objeto `SYSTEM.BASE.TOPIC`. `SYSTEM.BASE.TOPIC` tiene el valor YES.

Intentar crear una suscripción duradera para el tema `Sport/Soccer/TeamX/Results` ahora daría un error; no obstante, intentar realizar una suscripción duradera a `Sport/Tennis/PlayerB/Results` daría buen resultado.

Control del uso de comodines con la propiedad WILDCARD

Utilice la propiedad WILDCARD de **`Topic`** de MQSC o la propiedad `WildcardOperation` de Topic de PFC equivalente para controlar la entrega de publicaciones a las aplicaciones de suscriptor que utilizan nombres de serie de tema comodín. La propiedad WILDCARD puede tener uno de los dos valores siguientes:

WILDCARD
El comportamiento de las suscripciones comodín con respecto a este tema.

PASSTHRU
Las suscripciones realizadas en un tema con comodines menos específico que la serie de tema en este objeto de tema reciben publicaciones creadas para este tema y para series de tema más específicas que este tema.

BLOCK
Las suscripciones realizadas en un tema con comodín menos específico que la serie de tema en este objeto de tema no reciben publicaciones realizadas para este tema o para series de tema más específicas que este tema.

El valor de este atributo se utiliza cuando se definen las suscripciones. Si modifica este atributo, el conjunto de temas que abarcan las suscripciones existentes no se ve afectado por la modificación. Este escenario también se aplica si se cambia la topología cuando se crean o suprimen objetos de tema; el conjunto de temas que coinciden con las suscripciones creadas después de la modificación del atributo WILDCARD se crea utilizando la topología modificada. Si desea forzar que el conjunto de temas coincidentes se vuelva a evaluar para las suscripciones existentes, debe reiniciar el gestor de colas.

En el ejemplo, “Ejemplo: crear el clúster de publicación/suscripción Sport” en la página 84, puede seguir los pasos para crear la estructura de árbol de temas mostrada en Figura 23 en la página 81.

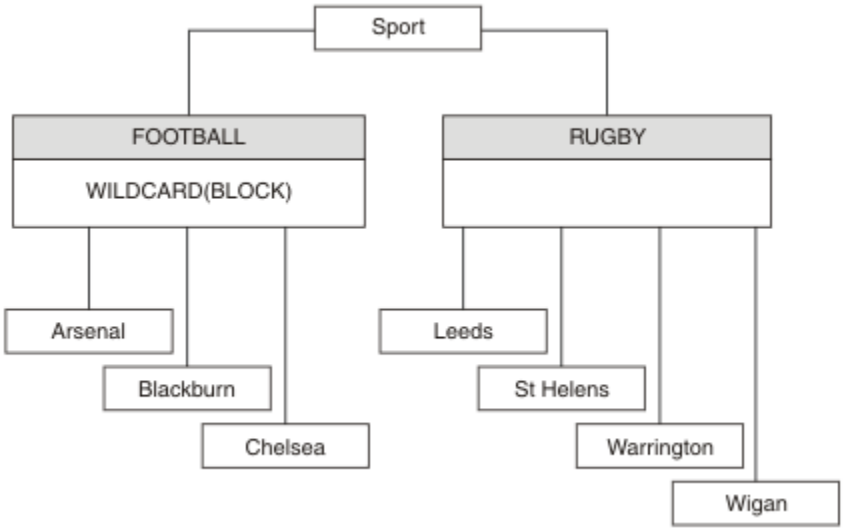


Figura 23. Árbol de temas que utiliza la propiedad WILDCARD BLOCK

Un suscriptor que utiliza la serie de tema comodín # recibe todas las publicaciones para el tema `Sport` y el subárbol `Sport/Rugby`. El suscriptor no recibe publicaciones en el subárbol `Sport/Football`, debido a que el valor de la propiedad `WILDCARD` del tema `Sport/Football` es `BLOCK`.

PASSTHRU es el valor predeterminado. Puede establecer la propiedad `WILDCARD` en el valor PASSTHRU en los nodos del árbol `Sport`. Si los nodos no tienen la propiedad `WILDCARD` con el valor `BLOCK`, definir PASSTHRU no altera el comportamiento observado por los suscriptores en los nodos del árbol `Sports`.

En el ejemplo, cree suscripciones para ver cómo el valor de comodín afecta a las publicaciones que se entregan; consulte Figura 27 en la página 86. Ejecute el mandato de publicación de Figura 30 en la página 87 para crear algunas publicaciones.

```
pub QMA
```

Figura 24. Publicación en QMA

Los resultados se muestran en Tabla 3 en la página 82. Observe cómo el establecimiento de la propiedad `WILDCARD` en `BLOCK` impide que las suscripciones con comodines puedan recibir publicaciones para temas del ámbito del comodín.

Tabla 3. Publicaciones recibidas en QMA			
Suscripción	**Serie de tema**	**Publicaciones recibidas**	**Notas**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Todas las publicaciones del subárbol Football bloqueadas por WILDCARD(BLOCK) en Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) en Sports/Football impide la suscripción de comodín en Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	El valor predeterminado WILDCARD en Sports/Rugby no impide la suscripción comodín en Leeds.

Nota:

Supongamos que una suscripción tiene un comodín que coincide con un objeto de tema con la propiedad WILDCARD con el valor BLOCK. Si la suscripción también tiene una serie de tema a la derecha del comodín coincidente, la suscripción nunca recibe una publicación. El conjunto de publicaciones que no están bloqueadas son publicaciones para temas que son padres del comodín bloqueado. Las publicaciones para temas que son hijos del tema que tiene el valor BLOCK para la propiedad están bloqueadas por el comodín. Por consiguiente, las series de publicación que incluyen un tema a la derecha del comodín nunca reciben publicaciones coincidentes.

Definir el valor de la propiedad WILDCARD en BLOCK no significa que no pueda suscribirse utilizando una serie de tema que incluya comodines. Una suscripción de este tipo es normal. La suscripción tiene un tema explícito que coincide con el tema que tiene un objeto de tema con la propiedad WILDCARD establecida en el valor BLOCK. Utiliza comodines para temas que son padres o hijos del tema con la propiedad WILDCARD establecida en el valor BLOCK. En el ejemplo de Figura 23 en la página 81, una suscripción como Sports/Football/# puede recibir publicaciones.

Comodines y temas de clúster

Las definiciones de tema de clúster se propagan a todos los gestores de colas de un clúster. Una suscripción a un tema de clúster en un gestor de colas de un clúster da lugar a que el gestor de colas cree suscripciones de proxy. Se crea una suscripción de proxy cada dos gestores de colas del clúster. Las suscripciones que utilizan series de temas que contienen caracteres, combinados con temas de clúster, pueden dificultar la capacidad de predecir el comportamiento. El comportamiento se describe en el siguiente ejemplo.

En el clúster configurado para el ejemplo, “Ejemplo: crear el clúster de publicación/suscripción Sport” en la página 84, QMB tiene el mismo conjunto de suscripciones que QMA, pero QMB no ha recibido ninguna publicación después de que el editor haya publicado en QMA, consulte Figura 24 en la página 81. Aunque los temas Sports/Football y Sports/Rugby son temas de clúster, las suscripciones definidas en fullsubs.tst no hacen referencia al tema de clúster. No se propaga ninguna suscripción de proxy de QMB a QMA. Sin suscripciones de proxy, ninguna publicación para QMA se reenvía a QMB.

Podría parecer que algunas de las suscripciones, como Sports/#/Leeds, hacen referencia a un tema de clúster, en este caso Sports/Rugby. En realidad, la suscripción Sports/#/Leeds se resuelve en el objeto de tema SYSTEM.BASE.TOPIC.

A continuación se indica la regla para resolver el objeto de tema al que hace referencia una suscripción como Sports/#/Leeds. Trunque la serie de tema en el primer comodín. Explore a la izquierda en la serie del tema para buscar el primer tema que tiene un objeto de tema administrativo asociado. El objeto de tema puede especificar un nombre de clúster, o definir un objeto de tema local. En el ejemplo,

Sports/#/Leeds, la serie de tema después del truncamiento es Sports, que no tiene objeto de tema, y, por tanto, Sports/#/Leeds hereda de SYSTEM.BASE.TOPIC, que es un objeto de tema local.

Para ver cómo la suscripción a temas agrupados en clúster puede cambiar el modo en que funciona la propagación de comodín, ejecute el script por lotes upsubs.bat. El script borra las colas de suscripción y agrega las suscripciones de tema de clúster de fullsubs.tst. Ejecute puba.bat de nuevo para crear un lote de publicaciones; consulte Figura 24 en la página 81.

En la Tabla 4 en la página 83 se muestra el resultado de añadir dos nuevas suscripciones al mismo gestor de colas en el que se han publicado las publicaciones. El resultado es como se esperaba, las suscripciones nuevas reciben una publicación cada una, y el número de publicaciones recibidas por las demás suscripciones no cambia. Los resultados inesperados se producen en el otro gestor de colas de clúster; consulte Tabla 5 en la página 83.

Tabla 4. Publicaciones recibidas en QMA

Suscripción	**Serie de tema**	**Publicaciones recibidas**	**Notas**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Todas las publicaciones del subárbol Football bloqueadas por WILDCARD(BLOCK) en Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) en Sports/Football impide la suscripción de comodín en Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	El valor predeterminado WILDCARD en Sports/Rugby no impide la suscripción comodín en Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal recibe una publicación porque la suscripción no tiene ningún comodín.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds recibirá una publicación en cualquier caso.

En la Tabla 5 en la página 83 se muestra el resultado de añadir las dos nuevas suscripciones en QMB y publicar en QMA. Recuerde que QMB no recibe publicaciones sin estas dos nuevas suscripciones. Tal como se esperaba, las dos nuevas suscripciones reciben publicaciones, ya que Sports/FootBall y Sports/Rugby son ambos temas de clúster. QMB ha reenviado suscripciones de proxy para Sports/Football/Arsenal y Sports/Rugby/Leeds a QMA, que luego ha reenviado las publicaciones a QMB.

El resultado inesperado es que las dos suscripciones Sports/# y Sports/#/Leeds que antes no recibían publicaciones, ahora reciben publicaciones. El motivo es que las publicaciones Sports/Football/Arsenal y Sports/Rugby/Leeds reenviadas a QMB para las demás suscripciones están ahora disponibles para cualquier suscriptor conectado a QMB. En consecuencia las suscripciones a los temas locales Sports/# y Sports/#/Leeds reciben la publicación Sports/Rugby/Leeds. Sports/#/Arsenal continúa sin recibir una publicación, porque Sports/Football tiene su propiedad WILDCARD con el valor establecido en BLOCK.

Tabla 5. Publicaciones recibidas en QMB

Suscripción	**Serie de tema**	**Publicaciones recibidas**	**Notas**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*Todas las publicaciones al subárbol Football bloqueadas por WILDCARD(BLOCK) en Sports/Football*

Tabla 5. Publicaciones recibidas en QMB (continuación)			
Suscripción	**Serie de tema**	**Publicaciones recibidas**	**Notas**
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) en Sports/Football impide la suscripción de comodín en Arsenal
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*El valor predeterminado de WILDCARD en Sports/Rugby no impide la suscripción comodín en Leeds.*
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal recibe una publicación porque la suscripción no tiene ningún comodín.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds recibirá una publicación en cualquier caso.

En muchas aplicaciones, no es aconsejable que una suscripción influya en el comportamiento de otra suscripción. Un uso importante de la propiedad WILDCARD con el valor BLOCK es hacer que las suscripciones a la misma serie de tema que contengan comodines se comporten de manera uniforme. Si la suscripción se encuentra en el mismo gestor de colas que el publicador o uno diferente, los resultados de la suscripción son los mismos.

Comodines y corrientes

Para una nueva aplicación escriba en la API de publicación/suscripción, el efecto es que una suscripción a * no recibe publicaciones. Para recibir todas las publicaciones de Sports, debe suscribirse a Sports/*, o Sports/#, y lo mismo para las publicaciones de Business.

El comportamiento de una aplicación de publicación/suscripción en cola existente no cambia cuando el intermediario de publicación/suscripción se migra a una versión posterior de IBM MQ. La propiedad **StreamName** de los mandatos **Publish**, **Register Publisher** o **Subscriber** se correlaciona con el nombre del tema al que se ha migrado la corriente.

Comodines y puntos de suscripción

Para una nueva aplicación escriba en la API de publicación/suscripción, el efecto de la migración es que una suscripción a * no recibe publicaciones. Para recibir todas las publicaciones de Sports, debe suscribirse a Sports/*, o Sports/#, y lo mismo para las publicaciones de Business.

El comportamiento de una aplicación de publicación/suscripción en cola existente no cambia cuando el intermediario de publicación/suscripción se migra a una versión posterior de IBM MQ. La propiedad **SubPoint** de los mandatos **Publish**, **Register Publisher** o **Subscriber** se correlaciona con el nombre del tema al que se ha migrado la suscripción.

Ejemplo: crear el clúster de publicación/suscripción Sport

Los pasos que se indican a continuación crean un clúster, CL1, con cuatro gestores de colas: dos repositorios completos, CL1A y CL1B, y dos repositorios parciales, QMA y QMB. Los repositorios completos se utilizan sólo para las definiciones de clúster. QMA se designa como host de temas de clúster. Las suscripciones duraderas se definen en QMA y QMB.

Nota: El ejemplo está codificado para Windows. Debe volver a codificar Create qmgrs.bat y Create pub.bat para configurar y probar el ejemplo en otras plataformas.

1. Cree los archivos de script.
   a. Create topics.tst

b. Crear wildsubs.tst
c. Crear fullsubs.tst
d. Crear qmgrs.bat
e. crear pub.bat

2. Ejecute Create qmgrs.bat para crear la configuración.

```
qmgrs
```

Cree los temas de Figura 23 en la página 81. El script de la figura 5 crea los temas de clúster `Sports/Football` y `Sports/Rugby`.

Nota: La opción REPLACE no sustituye las propiedades TOPICSTR de un tema. TOPICSTR es una propiedad que normalmente cambia en el ejemplo para probar distintos árboles de temas. Para cambiar temas, suprima primero el tema.

```
DELETE TOPIC ('Sports')
DELETE TOPIC ('Football')
DELETE TOPIC ('Arsenal')
DELETE TOPIC ('Blackburn')
DELETE TOPIC ('Chelsea')
DELETE TOPIC ('Rugby')
DELETE TOPIC ('Leeds')
DELETE TOPIC ('Wigan')
DELETE TOPIC ('Warrington')
DELETE TOPIC ('St. Helens')

DEFINE TOPIC ('Sports')   TOPICSTR('Sports')
DEFINE TOPIC ('Football')  TOPICSTR('Sports/Football') CLUSTER(CL1) WILDCARD(BLOCK)
DEFINE TOPIC ('Arsenal')  TOPICSTR('Sports/Football/Arsenal')
DEFINE TOPIC ('Blackburn') TOPICSTR('Sports/Football/Blackburn')
DEFINE TOPIC ('Chelsea')  TOPICSTR('Sports/Football/Chelsea')
DEFINE TOPIC ('Rugby')   TOPICSTR('Sports/Rugby')  CLUSTER(CL1)
DEFINE TOPIC ('Leeds')   TOPICSTR('Sports/Rugby/Leeds')
DEFINE TOPIC ('Wigan')   TOPICSTR('Sports/Rugby/Wigan')
DEFINE TOPIC ('Warrington') TOPICSTR('Sports/Rugby/Warrington')
DEFINE TOPIC ('St. Helens') TOPICSTR('Sports/Rugby/St. Helens')
```

Figura 25. Suprimir y crear temas: topics.tst

Nota: Suprima los temas, ya que REPLACE no sustituye series de tema.

Cree suscripciones con comodines. Los comodines corresponden a los temas con objetos de temas en Figura 23 en la página 81. Cree una cola para cada suscripción. Las colas se borran y las suscripciones se suprimen cuando se ejecuta o vuelve a ejecutar el script.

Nota: La opción REPLACE no sustituye las propiedades TOPICOBJ o TOPICSTR de una suscripción. TOPICOBJ o TOPICSTR son las propiedades que cambian normalmente en el ejemplo para probar distintas suscripciones. Para cambiarlas, suprima primero la suscripción.

```
DEFINE QLOCAL(QSPORTS)  REPLACE
DEFINE QLOCAL(QSARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS)  REPLACE
CLEAR QLOCAL(QSPORTS)
CLEAR QLOCAL(QSARSENAL)
CLEAR QLOCAL(QSLEEDS)

DELETE SUB  (SPORTS)
DELETE SUB  (SARSENAL)
DELETE SUB  (SLEEDS)
DEFINE SUB  (SPORTS)  TOPICSTR('Sports/#')     DEST(QSPORTS)
DEFINE SUB  (SARSENAL) TOPICSTR('Sports/+/Arsenal') DEST(QSARSENAL)
DEFINE SUB  (SLEEDS)  TOPICSTR('Sports/+/Leeds')  DEST(QSLEEDS)
```

Figura 26. Cree suscripciones comodín: wildsubs.tst

Cree suscripciones que hagan referencia a lo objetos de tema de clúster.

Nota:

El delimitador, /, se inserta automáticamente entre la serie de tema a la que hace referencia TOPICOBJ y la serie de tema definida por TOPICSTR.

La definición, DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) crea la misma suscripción. TOPICOBJ se utiliza como método rápido para hacer referencia a la serie de tema que ya se ha definido. La suscripción, cuando se crea, ya no hace referencia al objeto de tema.

```
DEFINE QLOCAL(QFARSENAL) REPLACE
DEFINE QLOCAL(QRLEEDS)  REPLACE
CLEAR QLOCAL(QFARSENAL)
CLEAR QLOCAL(QRLEEDS)

DELETE SUB  (FARSENAL)
DELETE SUB  (RLEEDS)
DEFINE SUB  (FARSENAL) TOPICOBJ('Football') TOPICSTR('Arsenal') DEST(QFARSENAL)
DEFINE SUB  (RLEEDS)  TOPICOBJ('Rugby')  TOPICSTR('Leeds')  DEST(QRLEEDS)
```

Figura 27. Suprima y cree suscripciones: fullsubs.tst

Cree un clúster con dos depósitos. Cree dos repositorios parciales para publicación y suscripción. Vuelva a ejecutar el script para suprimirlo todo y empezar de nuevo. El script también crea la jerarquía de temas y las suscripciones comodín iniciales.

Nota:

En otras plataformas, escriba un script parecido o escriba todos los mandatos. El uso de un script permite suprimirlo todo con mayor rapidez y empezar de nuevo con una configuración idéntica.

```
@echo off
set port.CL1B=1421
set port.CLIA=1420
for %%A in (CL1A CL1B QMA QMB) do call :createQM %%A
call :configureQM CL1A CL1B %port.CL1B% full
call :configureQM CL1B CL1A %port.CL1A% full
for %%A in (QMA QMB) do call :configureQM %%A CL1A %port.CL1A% partial
for %%A in (topics.tst wildsubs.tst) do runmqsc QMA < %%A
for %%A in (wildsubs.tst) do runmqsc QMB < %%A
goto:eof

:createQM
echo Configure Queue manager %1
endmqm -p %1
for %%B in (dlt crt str) do %%Bmqm %1
goto:eof

:configureQM
if %1==CL1A set p=1420
if %1==CL1B set p=1421
if %1==QMA set p=1422
if %1==QMB set p=1423
echo configure %1 on port %p% connected to repository %2 on port %3 as %4 repository
echo DEFINE LISTENER(LST%1) TRPTYPE(TCP) PORT(%p%) CONTROL(QMGR) REPLACE | runmqsc %1
echo START LISTENER(LST%1) | runmqsc %1
if full==%4 echo ALTER QMGR REPOS(CL1) DEADQ(SYSTEM.DEAD.LETTER.QUEUE) | runmqsc %1
echo DEFINE CHANNEL(TO.%2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('LOCALHOST(%3)') CLUSTER(CL1)
REPLACE | runmqsc %1
echo DEFINE CHANNEL(TO.%1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('LOCALHOST(%p%)')
CLUSTER(CL1) REPLACE | runmqsc %1
goto:eof
```

Figura 28. Cree gestores de colas: qmgrs.bat

Actualice la configuración añadiendo las suscripciones a los temas de clúster.

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

Figura 29. Actualice las suscripciones: upsubs.bat

Ejecute pub.bat, con un gestor de colas como parámetro, para publicar mensajes que contengan la serie de tema de publicación. Pub.bat utiliza el programa de ejemplo **amqspub**.

```
@echo off
@rem Provide queue manager name as a parameter
set S=Sports
set S=6 Sports/Football Sports/Football/Arsenal
set S=6 Sports/Rugby Sports/Rugby/Leeds
for %%B in (6) do echo %%B | amqspub %%B %1
```

Figura 30. Publique: pub.bat

Corrientes y temas

La publicación/suscripción en cola tiene el concepto de una corriente de publicación que no existe en el modelo de publicación/suscripción integrada. En la publicación/suscripción en cola, las corrientes ofrecen un modo de separar el flujo de información para distintos temas. Una corriente se implementa como un tema de nivel superior que se puede correlacionar con un identificador de tema diferente administrativamente.

La corriente predeterminada SYSTEM.BROKER.DEFAULT.STREAM se configura automáticamente para todos los intermediarios y gestores de colas de una red y no es necesaria ninguna configuración adicional para utilizar la corriente predeterminada. Piense en la corriente por omisión como en un espacio de temas por omisión no indicado. Los temas publicados en la corriente predeterminada están disponibles inmediatamente para todos los gestores de colas conectados, con la publicación/suscripción en cola habilitada. Las corrientes indicadas son como espacios de temas independientes indicados. La corriente indicada debe definirse en todos los intermediarios donde se utilice.

Si los publicadores y suscriptores están en distintos gestores de colas, después de los intermediarios se conecten en la misma jerarquía de intermediarios, no será necesaria ninguna configuración adicional para que las publicaciones y las suscripciones fluyan entre ellos. La misma interoperabilidad funciona a la inversa.

Corrientes indicadas

Un diseñador de soluciones que trabaja con el modelo de programación de publicación/suscripción en cola, puede haber decidido colocar todas las publicaciones de deporte en una corriente con nombre denominada Sport. Para que la corriente esté disponible para un gestor de colas que se ejecuta en IBM MQ con la publicación/suscripción en cola habilitada, la corriente debe añadirse manualmente.

Las aplicaciones de publicación/suscripción en cola que se suscriben a Soccer/Results en la corriente Sport funcionan sin cambios. Las aplicaciones de publicación/suscripción integradas que se suscriben al tema Sport utilizando MQSUB y proporcionando la serie de tema Soccer/Results, reciben también las mismas publicaciones.

La tarea de añadir una corriente se describe en el tema Adición de una corriente. Puede que tenga que añadir corrientes manualmente por dos razones.

1. Sigue desarrollando sus aplicaciones de publicación/suscripción en cola que se ejecutan en gestores de colas de versiones posteriores, en lugar de migrar las aplicaciones a la interfaz MQI de publicación/suscripción integrada.
2. La correlación por omisión de corrientes a temas llevas a una "colisión" en el espacio de temas, y las publicaciones de una corriente tienen la misma serie de temas que las publicaciones de otros sitios.

Autorizaciones

De forma predeterminada, en la raíz del árbol de temas hay varios objetos de tema: SYSTEM.BASE.TOPIC, SYSTEM.BROKER.DEFAULT.STREAM y SYSTEM.BROKER.DEFAULT.SUBPOINT. Las autoridades (por ejemplo, para la publicación o suscripción) las determinan las autoridades en el SYSTEM.BASE.TOPIC ; las autorizaciones sobre SYSTEM.BROKER.DEFAULT.STREAM o SYSTEM.BROKER.DEFAULT.SUBPOINT se ignoran. Si SYSTEM.BROKER.DEFAULT.STREAM o SYSTEM.BROKER.DEFAULT.SUBPOINT se suprimen y se vuelven a crear con una serie de tema no vacía, las autorizaciones definidas en dichos objetos se utilizan del mismo modo que un objeto de tema normal.

Correlación entre corrientes y temas

Una corriente de publicación/suscripción en cola se imita en IBM MQ creando una cola y dándole el mismo nombre que la corriente. En algunas ocasiones la cola se llama cola de corriente, porque así es como aparece en las aplicaciones de publicación/suscripción en cola. La cola es identificada al motor de publicación/suscripción añadiéndola a la lista de nombres especial llamada SYSTEM.QPUBSUB.QUEUE.NAMELIST. Puede añadir tantas corrientes como necesite, añadiendo colas especiales adicionales a la lista de nombres. Para finalizar debe añadir temas, con los mismos nombres que las corrientes, y las mismas series de temas que el nombre de tema, para poder publicar y suscribirse a los temas.

Sin embargo, en circunstancias excepcionales, puede dar a los temas correspondientes a las corrientes cualquier serie de tema que decida cuando defina los temas. La finalidad de la serie de tema es otorgar al tema un nombre exclusivo en el espacio del tema. Generalmente, el nombre de corriente cumple ese propósito a la perfección. A veces, un nombre de corriente y un nombre de tema existente entran en conflicto. Para resolver el problema, elija otra serie de tema para el tema asociado a la corriente. Elija cualquier serie de tema asegurándose de que sea exclusivo.

La serie de tema definida en las definición del tema recibe prefijo de modo normal a la serie de tema ofrecida por los publicadores y suscriptores utilizando las llamadas de MQI MQOPEN o MQSUB. Las aplicaciones que hacen referencia a temas utilizando objetos de tema no se ven afectados por la elección de serie de tema con prefijo, por eso puede elegir cualquier serie de tema que mantenga las publicaciones como únicas en el espacio de tema.

Para volver a correlacionar corrientes diferentes en temas diferentes se necesita que los prefijos utilizados para las series de temas sean únicos para separar totalmente un conjunto de temas de otro. Debe definir una convención de nombres de tema universal a la que debe adherirse estrictamente para que la correlación funcione.

En IBM MQ, utilice el mecanismo de prefijo para volver a correlacionar una serie de tema con otro lugar del espacio de temas.

Nota: Cuando elimine una corriente, elimine primero todas las suscripciones en la corriente. Esta acción es esencial si alguna de las suscripciones es origina desde otros intermediarios en la jerarquía de intermediarios.

Puntos de suscripción y temas

Los puntos de suscripción con nombre se emulan mediante temas y objetos de tema.

Para añadir puntos de suscripción manualmente, consulte Adición de un punto de suscripción.

Puntos de suscripción de IBM MQ

IBM MQ correlaciona puntos de suscripción con distintos espacios de temas dentro del árbol de temas de IBM MQ. Los temas de los mensajes de mandato sin un punto de suscripción se correlacionan sin ningún cambio en la raíz del árbol de temas de IBM MQ y heredan las propiedades de SYSTEM.BASE.TOPIC.

Los mensajes de mandato con un punto de suscripción se procesan utilizando la lista de objetos de tema de SYSTEM.QPUBSUB.SUBPOINT.NAMELIST. El nombre del punto de suscripción en el mensaje de mandato se compara con la serie de tema para cada uno de los objetos de tema en la lista. Si se

encuentra una coincidencia, el nombre de punto de suscripción se antepone, como un nodo de tema, a la serie de tema. El tema hereda sus propiedades del objeto de tema asociado que se encuentra en SYSTEM.QPUBSUB.SUBPOINT.NAMELIST.

El efecto de utilizar puntos de suscripción es crear un espacio de temas distinto para cada punto de suscripción. El espacio de temas se basa en un tema que tiene el mismo nombre que el punto de suscripción. Los temas de cada espacio de temas heredan sus propiedades del objeto de tema con el mismo nombre que el punto de suscripción.

Las propiedades no establecidas en el objeto de tema coincidente se heredan, de forma normal, de SYSTEM.BASE.TOPIC.

Las aplicaciones de publicación/suscripción en cola existentes, utilizando cabeceras de mensajes MQRFH2 , siguen funcionando estableciendo la propiedad **SubPoint** en los mensajes de mandato Publish o Register subscriber . El punto de suscripción se combina con la serie del tema en el mensaje de mandato y el tema lo procesa como cualquier otro.

Las aplicaciones de IBM MQ no se ven afectadas por los puntos de suscripción. Si una aplicación utiliza un tema que hereda información de uno de los objetos de tema coincidentes, dicha aplicación interoperará con una aplicación encolada utilizando el punto de suscripción coincidente.

Ejemplo

Una aplicación existente de publicación/suscripción de WebSphere Message Broker (ahora llamado IBM Integration Bus) migrada a IBM MQ ha creado dos objetos de tema, GBP y USD, con las correspondientes cadenas de tema 'GBP' y 'USD'.

Los publicadores existentes en el tema NYSE/IBM/SPOT, que han migrado para ejecutarse en IBM MQ y que utilizan el punto de suscripción USD crean publicaciones sobre el tema USD/NYSE/IBM/SPOT. Asimismo, los suscriptores existentes en NYSE/IBM/SPOT, utilizando el punto de suscripción USD crean suscripciones a USD/NYSE/IBM/SPOT.

Suscríbase al precio al contado en dólares en un programa de publicación/suscripción de IBM MQ llamando a MQSUB. Cree una suscripción utilizando el objeto de tema USD y la serie de tema 'NYSE/IBM/SPOT', tal como se muestra en el fragmento de código 'C'.

```
strncpy(sd.ObjectName, "USD", MQ_TOPIC_NAME_LENGTH);
sd.ObjectString.VSPtr = "NYSE/IBM/SPOT";
sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
```

1. Establezca el atributo CLUSTER de los objetos de tema USD y GBP en el host de tema de clúster.
2. Suprima todas las copias de los objetos de tema USD y GBP en otros gestores de colas del clúster.
3. Asegúrese de que USD y GBP estén definidos en SYSTEM.QPUBSUB.SUBPOINT.NAMELIST en cada gestor de colas del clúster.

Ejemplo de publicación/suscripción de gestor de colas individual

Figura 31 en la página 90 ilustra una única configuración básica de publicación/suscripción de gestor de colas. El ejemplo muestra la configuración de un servicio de noticias, donde hay información disponible de los publicadores sobre varios temas:

- Publicador 1 es la información de publicación acerca de los resultados deportivos utilizando el tema Deporte
- Publicador 2 es la información de publicación sobre los precios de las acciones utilizando el tema Acciones
- Publicador 3 es la información de publicación sobre las reseñas de películas utilizando el tema Películas, y sobre los programas de televisión utilizando el tema TV

Tres suscriptores han registrado su interés en distintos temas, de forma que el gestor de colas les envía información en la que están interesados:

- El suscriptor 1 recibe los resultados deportivos y los precios de las acciones
- El suscriptor 2 recibe las reseñas de películas
- El suscriptor 3 recibe los resultados deportivos

Ninguno de los suscriptores han registrado estar interesado en la programación de televisión, por lo que esta no se distribuye.

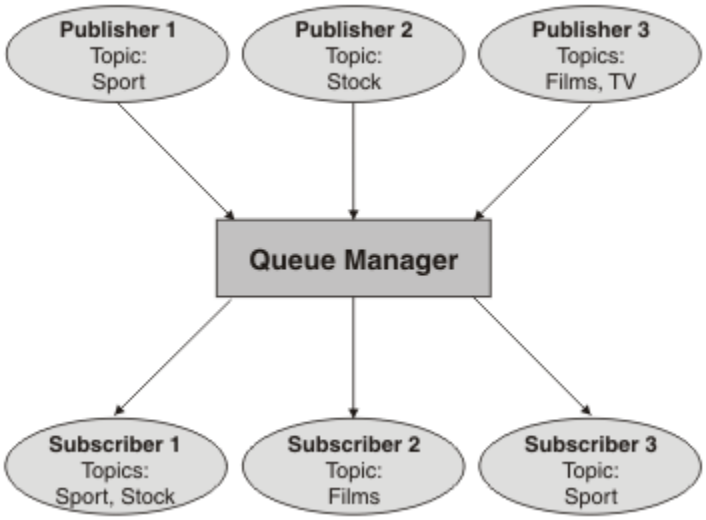


Figura 31. Ejemplo de publicación/suscripción de gestor de colas único

Redes de publicación/suscripción distribuidas

Cada gestor de colas compara los mensajes publicados en un tema con las suscripciones creadas localmente que se han suscrito a ese tema. Puede configurarse una red de gestores de colas de forma que los mensajes publicados por una aplicación conectada a un gestor de colas se entreguen a las suscripciones coincidentes creadas en otros gestores de colas de la red. Esto requiere una configuración adicional a simples canales entre gestores de colas.

Una configuración de publicación/suscripción distribuida es un conjunto de gestores de colas conectados entre sí. Los gestores de colas pueden estar todos en el mismo sistema físico o pueden estar distribuidos entre varios sistemas físicos. Cuando se conectan gestores de colas entre sí, los suscriptores pueden suscribirse a un gestor de colas y recibir mensajes publicados inicialmente en otro gestor de colas. Para ilustrar esto, la siguiente figura añade un segundo gestor de colas a la configuración descrita en "Ejemplo de publicación/suscripción de gestor de colas individual" en la página 89.

- El gestor de colas 2 es utilizado por el editor 4 para publicar información de previsión meteorológica, utilizando un tema Tiempo e información sobre las condiciones de tráfico en las rutas principales utilizando un tema Tráfico.
- El editor 4 también utiliza el gestor de colas, y se suscribe a la información sobre las condiciones de tráfico utilizando el tema Tráfico.
- El editor 3 también se suscribe a información sobre las condiciones meteorológicas, aunque utiliza un gestor de colas diferente del editor. Esto es posible debido a que los gestores de colas se enlazan entre sí.

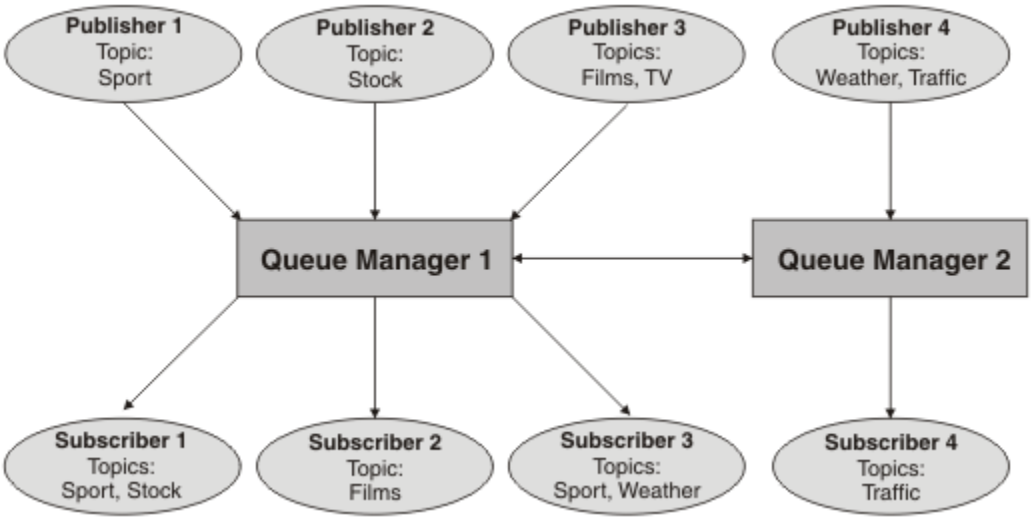


Figura 32. Ejemplo de publicación/suscripción con dos gestores de colas

Puede conectar manualmente los gestores de colas en una jerarquía padre e hijo o puede crear un clúster de publicación/suscripción y dejar que IBM MQ defina automáticamente gran parte del detalle de conexión. También pueden utilizarse ambas topologías conjuntamente, por ejemplo, uniendo varios clústeres en una jerarquía.

Visión general de un clúster de publicación/suscripción

Un clúster de publicación/suscripción es un clúster estándar al que se le han añadido uno o más objetos de tema. Cuando se define un objeto de tema administrativo en cualquier gestor de colas de un clúster, y se pone dicho objeto de tema en clúster especificando un nombre de clúster, los publicadores y suscriptores del tema podrán conectar con cualquiera de los gestores de colas del clúster y los mensajes publicados se direccionarán a los suscriptores a través de canales de clúster entre gestores de colas.

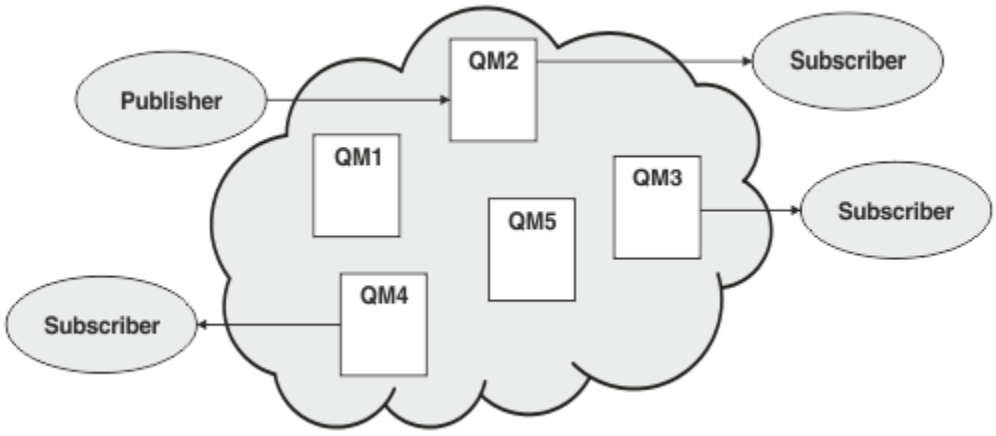


Figura 33. Clúster de publicación/suscripción

Hay dos maneras de configurar el modo en que los mensajes de publicación/suscripción se direccionan en un clúster:

- direccionamiento directo
- direccionamiento de host de tema

Cuando se configura un tema en clúster por direccionamiento directo, los mensajes publicados en un gestor de colas se envían directamente desde ese gestor de colas a todas las suscripciones en cualquier otro gestor de colas del clúster. Esto puede proporcionar la ruta más directa para las publicaciones, pero

da lugar a que todos los gestores de colas del clúster tengan conocimiento de todos los demás gestores de colas, pudiendo en potencia haber canales de clúster establecidos entre todos ellos.

Cuando se utiliza el direccionamiento de host de tema, los mensajes publicados en un gestor de colas se envían desde ahí a un gestor de colas que aloja una definición del objeto de tema administrado. Ese *gestor de colas de host de temas* direcciona el mensaje a cada suscripción en cualquier otro gestor de colas del clúster. Si los publicadores o los suscriptores no están ubicados en los gestores de colas de host de tema, las publicaciones seguirán una ruta más larga. Sin embargo, la ventaja es que solo los gestores de cola de host de tema tienen conocimiento de todos los demás gestores de colas del clúster y (posiblemente) tienen canales de clúster establecidos con ellos.

Para obtener más información, consulte “Clústeres de publicación/suscripción” en la página 93.

Visión general de las jerarquías de publicación/suscripción

Una jerarquía de publicación/suscripción es un conjunto de gestores de colas conectados mediante canales en una estructura jerárquica. Cada gestor de colas identifica su gestor de colas *padre*, tal y como se describe en Conexión de un gestor de colas a una jerarquía de publicación/suscripción.

Los publicadores y suscriptores de un tema pueden conectarse con cualquier gestor de colas de la jerarquía y los mensajes fluyen entre ellos valiéndose de la conectividad jerárquica de los gestores de colas.

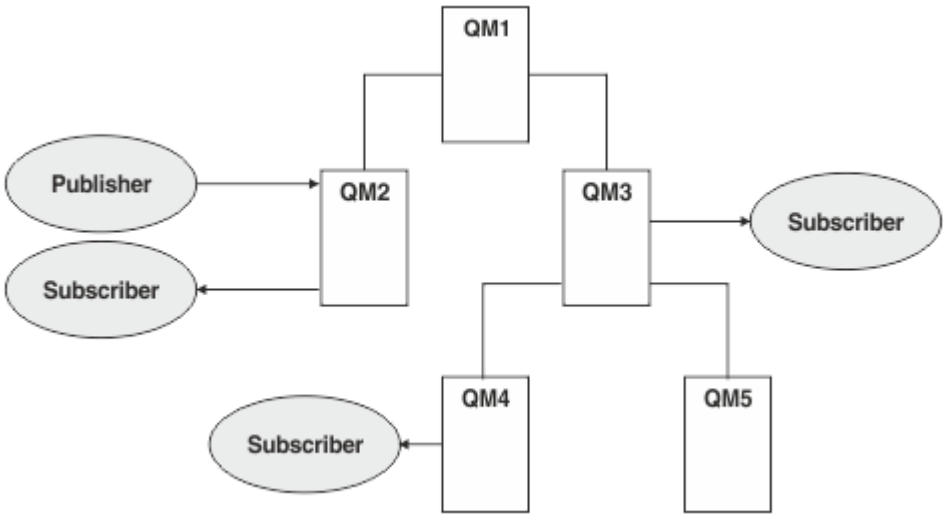


Figura 34. Jerarquía de publicación/suscripción

En la figura anterior, las publicaciones entregadas a los suscriptores en QM3 y QM4 se han direccionado de QM2 a QM1 y luego a QM3 y finalmente a QM4.

Las jerarquías dan un control directo sobre las relaciones existentes entre cada gestor de colas en la jerarquía. Esto permite un control preciso sobre el direccionamiento de los mensajes desde los publicadores a los suscriptores y resulta de especial utilidad cuando se direccionan entre redes de gestores de colas con conectividad limitada. Debe prestarse especial atención a la disponibilidad y capacidad de cada uno de los gestores de colas a través de los cuales se direcciona un mensaje en su camino desde el publicador al suscriptor.

Para obtener más información, consulte “Jerarquías de publicación/suscripción” en la página 96.

Distribución de una publicación entre gestores de colas

Además de las opciones de direccionamiento, existen dos enfoques en la distribución de las publicaciones en una red de gestores de colas:

- Solo enviar las publicaciones desde un gestor de colas a los gestores de colas que en ese momento alojen una suscripción a esa publicación.
- Enviar todas las publicaciones a todos los gestores de colas, y dejar que ellos las comparen con sus suscripciones.

El primer enfoque da lugar a que solo se envíen mensajes de publicación cuando es necesario, pero requiere que los gestores de colas compartan un cierto nivel de conocimiento de las suscripciones. El segundo enfoque no requiere compartición de conocimiento de suscripciones, pero puede dar lugar al envío innecesario de mensajes de publicación entre los gestores de colas.

De forma predeterminada, IBM MQ utiliza el método anterior, en el que las publicaciones sólo se envían a los gestores de colas que tienen suscripciones para ellos. El conocimiento de las suscripciones se propaga entre los gestores de colas en forma de *suscripciones de proxy*. Cuál de los dos será el más eficiente en el uso de una topología distribuida de publicación/suscripción dependerá de la distribución y del tiempo de vida de las suscripciones, y de la frecuencia de las publicaciones. Consulte Rendimiento de suscripción en redes de publicación/suscripción.

Conceptos relacionados

"Árboles de temas" en la página 77
Cada tema que se define es un elemento, o nodo, del árbol de temas. El árbol de temas puede estar vacío para empezar o puede contener temas que se hayan definido anteriormente utilizando mandatos MQSC o PCF. Puede definir un nuevo tema utilizando los mandatos crear tema o especificando el tema por primera vez en una publicación o suscripción.

Escenarios de jerarquías de publicación/suscripción

Tareas relacionadas

Diseño de clústeres de publicación/suscripción

Clústeres de publicación/suscripción

Un clúster de publicación es un clúster estándar de gestores de colas interconectados en el que las publicaciones se mueven automáticamente desde las aplicaciones de publicación hasta las suscripciones que existen en cualquiera de los gestores de colas en el clúster. Hay dos opciones para direccionar publicaciones a través de un clúster de publicación/suscripción: *direccionamiento directo* y *direccionamiento de host de temas*. El direccionamiento elegido dependerá del tamaño y de los patrones de actividad esperados del clúster.

Un clúster que se utiliza para la mensajería de publicación/suscripción no es diferente de un clúster IBM MQ estándar. Como tal, los gestores de colas del clúster de publicación/suscripción pueden existir en sistemas separados físicamente y cada par de gestores de colas se conectan automáticamente entre ellos con canales de clúster cuando sea necesario. Para obtener más información, consulte Clústeres.

Para configurar un clúster estándar de gestores de colas para la mensajería de publicación/suscripción, defina uno o más objetos de tema administrados en un gestor de colas del clúster. Para hacer que el tema sea un tema de clúster, configure la propiedad **CLUSTER** con el nombre del clúster. Al hacerlo, cualquier tema que utilice un publicador o suscriptor en ese punto o por debajo en el árbol de temas será compartido entre todos los gestores de colas del clúster y los mensajes publicados en una rama en clúster del árbol de temas se direccionarán de forma automática a suscripciones en otros gestores de colas del clúster.

Solo se envía una copia de cada mensaje entre el gestor de colas publicador y cada uno de los demás gestores de colas, independientemente del número de suscriptores al mensaje en el gestor de colas de destino. Al llegar a un gestor de colas con una o más suscripciones, el mensaje se duplica en todas las suscripciones.

Cualquier gestor de colas que se una al clúster tendrá conocimiento automático de todos los temas en clúster y los publicadores y suscriptores en ese gestor de colas participarán de forma automática en el clúster.

Una actividad de publicación/suscripción no agrupada en clúster también puede tener lugar en un clúster de publicación/suscripción, trabajando con series de tema que no entran en un objeto de tema agrupado en clúster.

Hay dos opciones para direccionar publicaciones a través de un clúster de publicación/suscripción: *direccionamiento directo* y *direccionamiento de host de temas*. Para elegir el direccionamiento de mensajes a utilizar en el clúster, establezca la propiedad **CLROUTE** en el objeto de tema administrado en uno de los valores siguientes:

- **DIRECT**
- **TOPICHOST**

De forma predeterminada, el direccionamiento de temas es **DIRECT**. Cuando se configura un tema de clúster de direccionamiento directo en un gestor de colas, todos los gestores de colas del clúster reconocen los otros gestores de colas del clúster. Al realizar operaciones de publicación y suscripción, cada gestor de colas puede conectarse directamente a todos los otros gestores de colas del clúster.

A partir de IBM MQ 8.0, en su lugar, puede configurar el direccionamiento de temas como **TOPICHOST**. Cuando se utiliza el direccionamiento de host de temas, todos los gestores de colas del clúster pasan a reconocer los gestores de colas del clúster que alojan la definición del tema direccionado (es decir, los gestores de colas en los que se ha definido el objeto de tema). Cuando se realizan operaciones de publicación y suscripción, los gestores de colas del clúster sólo se conectan a estos gestores de colas de host de temas, no directamente entre sí. Los gestores de colas de host de temas son responsables del direccionamiento de publicaciones desde los gestores de colas en los que se publican publicaciones y los gestores de colas con suscripciones coincidentes.

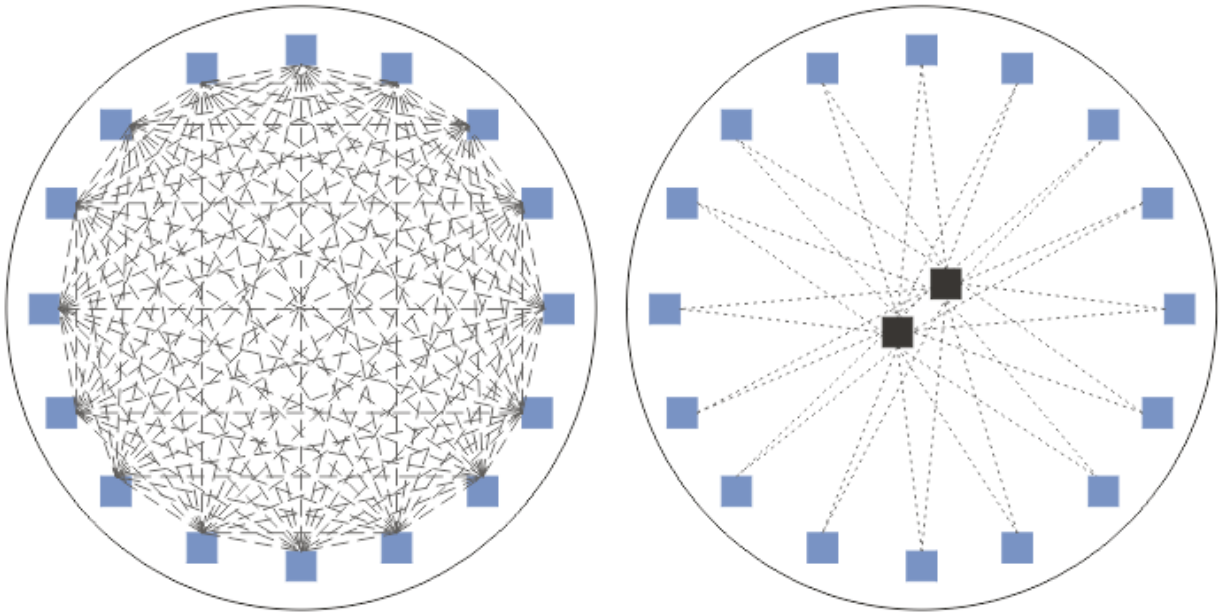


Figura 35. Direccionamiento directo y direccionamiento de host de temas

Descripción general del direccionamiento directo

Cuando se configura un objeto de tema administrado para direccionamiento directo, solo hay que definir dicho objeto de tema en uno de los gestores de colas del clúster para que todos los gestores de colas tengan conocimiento de él. La elección del gestor de colas en el que se define el tema no afecta al comportamiento de la mensajería de publicación/suscripción del tema.

Cada mensaje fluye directamente del gestor de colas publicador a cada suscripción en los demás gestores de colas del clúster sin pasar por gestores de colas intermedios.

De forma predeterminada, los mensajes solo se envían a otros gestores de colas en el clúster que alojan una o más suscripciones.

- Esto depende de que cada gestor de colas informe directamente a todos los demás gestores de colas del clúster de todos los temas que en ese momento tengan una o más suscripciones a él. Esto da como resultado que todos los gestores de colas del clúster tengan conocimiento de todos los temas

suscritos y que cualquier gestor de colas que aloje una suscripción establezca un canal a todos los demás gestores de colas. Esto es independiente de que cada gestor de colas tenga o no un publicador.

- El conocimiento de cada tema suscrito individual en todos los gestores de colas puede eliminarse cambiando a un modelo de envío de todas las publicaciones a todos los gestores de colas del clúster independientemente de que tengan o no suscripciones. Esto disminuye el tráfico de conocimiento de las suscripciones pero es probable que aumente el tráfico de publicaciones y el número de canales que establece cada gestor de colas. Consulte Rendimiento de suscripción en redes de publicación/suscripción.

Los flujos de mensajes de publicación/suscripción que utilizan temas en clúster de direccionamiento directo pueden abarcar múltiples clústeres de publicación/suscripción añadiendo un gestor de colas de cada clúster a una jerarquía de publicación/suscripción. Consulte Combinación de espacios de temas de varios clústeres.

Para obtener una exposición más detallada del direccionamiento directo, consulte Direccionamiento directo en clústeres de publicación/suscripción.

Descripción general del direccionamiento de host de tema

Cuando un objeto de tema administrado está configurado para el direccionamiento de host de tema, las publicaciones procedentes de un gestor de colas en el clúster se direccionan a través de un gestor de colas en el que está configurado el objeto de tema (un "host de tema"), y desde ahí van a los gestores de colas donde están las suscripciones.

- Esto depende de que cada gestor de colas informe a todos los hosts de tema de cada tema que en ese momento tenga una o más suscripciones. Todo gestor de colas que aloje una suscripción establece un canal con cada host de tema para el tema al que está asociada la suscripción.
- A los gestores de colas que no alojan temas no se les informa de otros gestores de colas que no alojan temas en el clúster a efectos de publicación/suscripción, y no se establecen canales entre ellos a tal fin.
- Si la aplicación de publicación está conectada con un gestor de colas que aloja el tema, los mensajes publicados se direccionan directamente a los gestores de colas en los que se han creado suscripciones coincidentes sin necesidad de un 'salto' adicional. De forma similar, si las suscripciones coincidentes se crean en el único gestor de colas que aloja el tema, los mensajes publicados en ese tema se direccionarán directamente a ese gestor de colas sin necesidad de un salto adicional.
- Las suscripciones en el mismo gestor de colas que el publicador se satisfacen sin direccionar antes las publicaciones a los hosts del objeto de tema.

En cuanto a las colas en clúster, múltiples gestores de colas pueden configurar el mismo objeto de tema administrativo. Esto proporciona una mayor disponibilidad del direccionamiento de mensajes y un escalado horizontal mediante un equilibrado de cargas. En el caso de los objetos de tema direccionados por host, cuando múltiples gestores de colas configuran el mismo tema nombrado para la misma rama del árbol de temas, se hace que cada host de tema tenga conocimiento de los temas suscritos por cada gestor de colas que aloja una suscripción.

- Cuando se publica un mensaje, se envía a uno de los gestores de colas de host de tema para que lo reenvíe a los gestores de colas que alojan la suscripción. La elección del gestor de colas de host de tema se ajusta a las mismas reglas de equilibrado de carga predeterminadas que las colas punto a punto en clúster.
- Si un gestor de colas publicador no puede contactar con uno o más gestores de colas de host de tema, los mensajes se direccionan a los restantes gestores de colas que alojan tema disponibles.

Cada publicación en un tema de una rama direccionada del árbol de temas se reenvía a uno de los hosts de tema incluso si no hay suscripciones a este tema en ninguna parte del clúster. De forma predeterminada, los mensajes solo se envían desde aquí a otros gestores de colas en el clúster que alojan una o más suscripciones.

- Esto se apoya en que se informe a cada gestor de colas de host de tema de todas las cadenas de tema suscritas en cada gestor de colas del clúster.

- El conocimiento de cada tema suscrito individual puede eliminarse cambiando a un modelo de envío de todas las publicaciones direccionadas a un host de tema en todos los gestores de colas del clúster independientemente de que tengan o no suscripciones. Esto reduce el tráfico de conocimiento de suscripciones, pero es probable que incremente el tráfico de publicaciones y potencialmente el número de canales establecidos con cada gestor de colas que aloja temas. Consulte Rendimiento de suscripción en redes de publicación/suscripción.

Los flujos de mensajes de publicación/suscripción que utilizan temas en clúster direccionados por host de tema **no pueden** abarcar múltiples clústeres de publicación/suscripción mediante el uso de una jerarquía de publicación/suscripción.

Para obtener una exposición más detallada del direccionamiento de host de tema, consulte Direccionamiento de host de tema en clústeres de publicación/suscripción.

Jerarquías de publicación/suscripción

Una jerarquía de publicación/suscripción se crea enlazando los gestores de colas entre sí mediante canales y luego definiendo una relación padre-hijo entre pares de gestores de colas. Un mensaje fluye desde un publicador a las suscripciones a través de relaciones directas en una jerarquía. Observe que esto podría suponer múltiples "saltos" para llegar ahí.

Solo se envía una copa de un mensaje entre cualquier par de gestores de colas, independientemente del número de suscriptores al mensaje en el gestor de colas de destino. Al llegar a un gestor de colas con una o más suscripciones, el mensaje se duplica en todas las suscripciones.

De forma predeterminada, los mensajes solo se envían a otros gestores de colas en la jerarquía que están en la ruta a una suscripción en otro gestor de colas:

- Esto requiere que cada gestor de colas informe a cada relación directa de todos los temas que en ese momento tienen una o más suscripciones a él, ya sea en este gestor de colas o en uno de sus otras relaciones. Esto da lugar a que todos los gestores de colas en la jerarquía tengan conocimiento de todos los temas a los que se está suscrito.
- Este comportamiento puede modificarse para que siempre se envíen publicaciones a todos los gestores de colas de la jerarquía independientemente de las suscripciones que pueda haber. Esto elimina la necesidad de propagar la información de suscripciones por la jerarquía, pero puede incrementar el tráfico de publicaciones.

Cuando se crea un clúster, hay que procurar no crear un bucle que provoque que los mensajes estén dando vueltas por la red sin parar. Tales bucles no pueden crearse en una jerarquía.

Todo gestor de colas debe tener un nombre de gestor de colas exclusivo.

Los flujos de mensajes de publicación/suscripción pueden abarcar múltiples clústeres de publicación/suscripción. Para hacerlo, añada un gestor de colas de cada clúster a una jerarquía de publicación/suscripción.

Para obtener una exposición más detallada, consulte Direccionamiento en jerarquías de publicación/suscripción.

Suscripciones de proxy en una red de publicación/suscripción

Una suscripción de proxy es una suscripción realizada por un gestor de colas para temas publicados en otro gestor de colas. Una suscripción de proxy fluye entre los gestores de colas para cada serie de tema individual suscrita mediante una suscripción. Las suscripciones de proxy no se crean explícitamente, el gestor de colas lo hace por usted.

Puede conectar gestores de colas entre sí en un clúster de publicación/suscripción, o en una jerarquía de publicación/suscripción. Las suscripciones de proxy fluyen entre los gestores de colas conectados. Las suscripciones de proxy provocan que las publicaciones en un tema creado por un publicador conectado a un gestor de colas sean recibidas por los suscriptores a dicho tema que están conectados a otros gestores de colas. Consulte “Redes de publicación/suscripción distribuidas” en la página 90.

En las topologías de publicación/suscripción con muchos miles de suscripciones a cadenas de tema individuales, o donde la existencia de estas suscripciones puede cambiar rápidamente, se debe tener en cuenta la sobrecarga de la propagación de suscripciones de proxy. Además de la agregación automática descrita en el resto de este tema, pueden efectuarse cambios manuales de configuración que limiten aún más el flujo de publicaciones y suscripciones de proxy entre gestores de colas conectados, y eso reduce la latencia de espera en la propagación de una suscripción de proxy a todos los gestores de colas conectados. Consulte Rendimiento de suscripción en redes de publicación/suscripción.

Las suscripciones de proxy no contienen ningún selector utilizado por suscripciones locales, y las cadenas de tema de suscripción que contengan comodines podrían simplificarse. Esto puede resultar en que haya publicaciones que coincidan con suscripciones de proxy donde las suscripciones reales no lo hacen, lo que da lugar a un flujo de publicaciones adicional entre gestores de colas. El gestor de colas que aloja las suscripciones filtra tales discrepancias de forma que no se devuelvan publicaciones adicionales a la suscripción.

Agregación de suscripciones de proxy

Las suscripciones de proxy se agregan utilizando un sistema de eliminación de duplicados. Para una determinada serie de tema resuelta, se envía una suscripción de proxy en la primera suscripción local o suscripción de proxy recibida. Las posteriores suscripciones a la misma serie de tema utiliza esta suscripción de proxy existente.

La suscripción de proxy se cancela después de cancelarse la última suscripción local o suscripción de proxy recibida.

Agregación de publicaciones

Cuando hay más de una suscripción a la misma serie de tema en un gestor de colas, sólo se envía una sola copia de cada publicación que coincida con dicha serie de tema de otros gestores de colas en la topología de publicación/suscripción. Al llegar el mensaje, el gestor de colas local entrega una copia del mensaje a cada suscripción coincidente.

Es posible que más de una suscripción de proxy coincida con la serie de tema de una sola publicación cuando las suscripciones de proxy contienen comodines. Si se publica un mensaje en un gestor de colas que coincide con dos o más suscripciones de proxy creadas por un único gestor de colas conectado, sólo una copia de la publicación se reenvía al gestor de colas remoto para satisfacer las diversas suscripciones de proxy.

Conceptos relacionados

Detección de bucles en una red distribuida de publicación/suscripción

Comodines en suscripciones de proxy

Las suscripciones pueden utilizar comodines en cadenas de tema para coincidir con múltiples cadenas de tema en las publicaciones.

Hay dos esquemas de comodín que puede utilizar una suscripción: *basado en tema* y *basado en carácter*. Consulte “Esquemas de comodín” en la página 71.

En IBM MQ , todas las suscripciones de proxy para suscripciones comodín se convierten para utilizar comodines basados en temas. Si se encuentra un comodín basado en carácter, se sustituirá por el carácter `#` hasta la `/` más cercana. Por ejemplo, `/aaa/bbb/c*d` se convierte a `/aaa/bbb/#`. La conversión produce que los gestores de colas remotos envíen algunas publicaciones más de aquellas a las que se suscribieron de forma explícita. El gestor de colas local filtra las publicaciones adicionales cuando entrega las publicaciones a sus suscriptores locales.

Control del uso de comodines con la propiedad WILDCARD

Utilice la propiedad WILDCARD de **`Topic`** de MQSC o la propiedad `WildcardOperation` de Topic de PFC equivalente para controlar la entrega de publicaciones a las aplicaciones de suscriptor que

utilizan nombres de serie de tema comodín. La propiedad WILDCARD puede tener uno de los dos valores siguientes:

WILDCARD
El comportamiento de las suscripciones comodín con respecto a este tema.

PASSTHRU
Las suscripciones realizadas en un tema con comodines menos específico que la serie de tema en este objeto de tema reciben publicaciones creadas para este tema y para series de tema más específicas que este tema.

BLOCK
Las suscripciones realizadas en un tema con comodín menos específico que la serie de tema en este objeto de tema no reciben publicaciones realizadas para este tema o para series de tema más específicas que este tema.

El valor de este atributo se utiliza cuando se definen las suscripciones. Si modifica este atributo, el conjunto de temas que abarcan las suscripciones existentes no se ve afectado por la modificación. Este escenario también se aplica si se cambia la topología cuando se crean o suprimen objetos de tema; el conjunto de temas que coinciden con las suscripciones creadas después de la modificación del atributo WILDCARD se crea utilizando la topología modificada. Si desea forzar que el conjunto de temas coincidentes se vuelva a evaluar para las suscripciones existentes, debe reiniciar el gestor de colas.

En el ejemplo, "Ejemplo: crear el clúster de publicación/suscripción Sport" en la página 84, puede seguir los pasos para crear la estructura de árbol de temas mostrada en Figura 23 en la página 81.

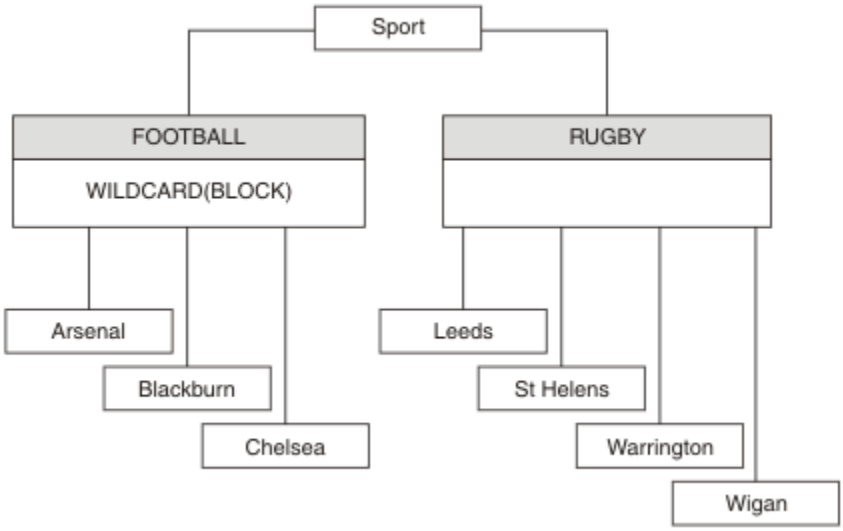


Figura 36. Árbol de temas que utiliza la propiedad WILDCARD BLOCK

Un suscriptor que utiliza la serie de tema comodín # recibe todas las publicaciones para el tema `Sport` y el subárbol `Sport/Rugby`. El suscriptor no recibe publicaciones en el subárbol `Sport/Football`, debido a que el valor de la propiedad `WILDCARD` del tema `Sport/Football` es BLOCK.

PASSTHRU es el valor predeterminado. Puede establecer la propiedad `WILDCARD` en el valor PASSTHRU en los nodos del árbol `Sport`. Si los nodos no tienen la propiedad `WILDCARD` con el valor BLOCK, definir PASSTHRU no altera el comportamiento observado por los suscriptores en los nodos del árbol `Sports`.

En el ejemplo, cree suscripciones para ver cómo el valor de comodín afecta a las publicaciones que se entregan; consulte Figura 27 en la página 86. Ejecute el mandato de publicación de Figura 30 en la página 87 para crear algunas publicaciones.

```
pub QMA
```

Figura 37. Publicación en QMA

Los resultados se muestran en Tabla 3 en la página 82. Observe cómo el establecimiento de la propiedad WILDCARD en BLOCK impide que las suscripciones con comodines puedan recibir publicaciones para temas del ámbito del comodín.

Tabla 6. Publicaciones recibidas en QMA

Suscripción	Serie de tema	Publicaciones recibidas	Notas
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Todas las publicaciones del subárbol Football bloqueadas por WILDCARD(BLOCK) en Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) en Sports/Football impide la suscripción de comodín en Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	El valor predeterminado WILDCARD en Sports/Rugby no impide la suscripción comodín en Leeds.

Nota:

Supongamos que una suscripción tiene un comodín que coincide con un objeto de tema con la propiedad WILDCARD con el valor BLOCK. Si la suscripción también tiene una serie de tema a la derecha del comodín coincidente, la suscripción nunca recibe una publicación. El conjunto de publicaciones que no están bloqueadas son publicaciones para temas que son padres del comodín bloqueado. Las publicaciones para temas que son hijos del tema que tiene el valor BLOCK para la propiedad están bloqueadas por el comodín. Por consiguiente, las series de publicación que incluyen un tema a la derecha del comodín nunca reciben publicaciones coincidentes.

Definir el valor de la propiedad WILDCARD en BLOCK no significa que no pueda suscribirse utilizando una serie de tema que incluya comodines. Una suscripción de este tipo es normal. La suscripción tiene un tema explícito que coincide con el tema que tiene un objeto de tema con la propiedad WILDCARD establecida en el valor BLOCK. Utiliza comodines para temas que son padres o hijos del tema con la propiedad WILDCARD establecida en el valor BLOCK. En el ejemplo de Figura 23 en la página 81, una suscripción como Sports/Football/# puede recibir publicaciones.

Comodines y temas de clúster

Las definiciones de tema de clúster se propagan a todos los gestores de colas de un clúster. Una suscripción a un tema de clúster en un gestor de colas de un clúster da lugar a que el gestor de colas cree suscripciones de proxy. Se crea una suscripción de proxy cada dos gestores de colas del clúster. Las suscripciones que utilizan series de temas que contienen caracteres, combinados con temas de clúster, pueden dificultar la capacidad de predecir el comportamiento. El comportamiento se describe en el siguiente ejemplo.

En el clúster configurado para el ejemplo, “Ejemplo: crear el clúster de publicación/suscripción Sport” en la página 84, QMB tiene el mismo conjunto de suscripciones que QMA, pero QMB no ha recibido ninguna publicación después de que el editor haya publicado en QMA, consulte Figura 24 en la página 81. Aunque los temas Sports/Football y Sports/Rugby son temas de clúster, las suscripciones definidas en fullsubs.tst no hacen referencia al tema de clúster. No se propaga ninguna suscripción de proxy de QMB a QMA. Sin suscripciones de proxy, ninguna publicación para QMA se reenvía a QMB.

Podría parecer que algunas de las suscripciones, como Sports/#/Leeds, hacen referencia a un tema de clúster, en este caso Sports/Rugby. En realidad, la suscripción Sports/#/Leeds se resuelve en el objeto de tema SYSTEM.BASE.TOPIC.

A continuación se indica la regla para resolver el objeto de tema al que hace referencia una suscripción como Sports/#/Leeds. Trunque la serie de tema en el primer comodín. Explore a la izquierda en la serie del tema para buscar el primer tema que tiene un objeto de tema administrativo asociado. El objeto de tema puede especificar un nombre de clúster, o definir un objeto de tema local. En el ejemplo, Sports/#/Leeds, la serie de tema después del truncamiento es Sports, que no tiene objeto de tema, y, por tanto, Sports/#/Leeds hereda de SYSTEM.BASE.TOPIC, que es un objeto de tema local.

Para ver cómo la suscripción a temas agrupados en clúster puede cambiar el modo en que funciona la propagación de comodín, ejecute el script por lotes upsubs.bat. El script borra las colas de suscripción y agrega las suscripciones de tema de clúster de fullsubs.tst. Ejecute puba.bat de nuevo para crear un lote de publicaciones; consulte Figura 24 en la página 81.

En la Tabla 4 en la página 83 se muestra el resultado de añadir dos nuevas suscripciones al mismo gestor de colas en el que se han publicado las publicaciones. El resultado es como se esperaba, las suscripciones nuevas reciben una publicación cada una, y el número de publicaciones recibidas por las demás suscripciones no cambia. Los resultados inesperados se producen en el otro gestor de colas de clúster; consulte Tabla 5 en la página 83.

Tabla 7. Publicaciones recibidas en QMA

Suscripción	**Serie de tema**	**Publicaciones recibidas**	**Notas**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Todas las publicaciones del subárbol Football bloqueadas por WILDCARD(BLOCK) en Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) en Sports/Football impide la suscripción de comodín en Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	El valor predeterminado WILDCARD en Sports/Rugby no impide la suscripción comodín en Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal recibe una publicación porque la suscripción no tiene ningún comodín.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds recibirá una publicación en cualquier caso.

En la Tabla 5 en la página 83 se muestra el resultado de añadir las dos nuevas suscripciones en QMB y publicar en QMA. Recuerde que QMB no recibe publicaciones sin estas dos nuevas suscripciones. Tal como se esperaba, las dos nuevas suscripciones reciben publicaciones, ya que Sports/FootBall y Sports/Rugby son ambos temas de clúster. QMB ha reenviado suscripciones de proxy para Sports/Football/Arsenal y Sports/Rugby/Leeds a QMA, que luego ha reenviado las publicaciones a QMB.

El resultado inesperado es que las dos suscripciones Sports/# y Sports/#/Leeds que antes no recibían publicaciones, ahora reciben publicaciones. El motivo es que las publicaciones Sports/Football/Arsenal y Sports/Rugby/Leeds reenviadas a QMB para las demás suscripciones están ahora disponibles para cualquier suscriptor conectado a QMB. En consecuencia las suscripciones a los temas locales Sports/# y Sports/#/Leeds reciben la publicación Sports/Rugby/Leeds. Sports/#/Arsenal continúa sin recibir una publicación, porque Sports/Football tiene su propiedad WILDCARD con el valor establecido en BLOCK.

Tabla 8. Publicaciones recibidas en QMB			
Suscripción	**Serie de tema**	**Publicaciones recibidas**	**Notas**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*Todas las publicaciones al subárbol Football bloqueadas por WILDCARD(BLOCK) en Sports/Football*
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) en Sports/Football impide la suscripción de comodín en Arsenal
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*El valor predeterminado de WILDCARD en Sports/Rugby no impide la suscripción comodín en Leeds.*
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal recibe una publicación porque la suscripción no tiene ningún comodín.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds recibirá una publicación en cualquier caso.

En muchas aplicaciones, no es aconsejable que una suscripción influya en el comportamiento de otra suscripción. Un uso importante de la propiedad WILDCARD con el valor BLOCK es hacer que las suscripciones a la misma serie de tema que contengan comodines se comporten de manera uniforme. Si la suscripción se encuentra en el mismo gestor de colas que el publicador o uno diferente, los resultados de la suscripción son los mismos.

Comodines y corrientes

Para una nueva aplicación escriba en la API de publicación/suscripción, el efecto es que una suscripción a * no recibe publicaciones. Para recibir todas las publicaciones de Sports, debe suscribirse a Sports/*, o Sports/#, y lo mismo para las publicaciones de Business.

El comportamiento de una aplicación de publicación/suscripción en cola existente no cambia cuando el intermediario de publicación/suscripción se migra a una versión posterior de IBM MQ. La propiedad **StreamName** de los mandatos **Publish**, **Register Publisher** o **Subscriber** se correlaciona con el nombre del tema al que se ha migrado la corriente.

Comodines y puntos de suscripción

Para una nueva aplicación escriba en la API de publicación/suscripción, el efecto de la migración es que una suscripción a * no recibe publicaciones. Para recibir todas las publicaciones de Sports, debe suscribirse a Sports/*, o Sports/#, y lo mismo para las publicaciones de Business.

El comportamiento de una aplicación de publicación/suscripción en cola existente no cambia cuando el intermediario de publicación/suscripción se migra a una versión posterior de IBM MQ. La propiedad **SubPoint** de los mandatos **Publish**, **Register Publisher** o **Subscriber** se correlaciona con el nombre del tema al que se ha migrado la suscripción.

Ejemplo: crear el clúster de publicación/suscripción Sport

Los pasos que se indican a continuación crean un clúster, CL1, con cuatro gestores de colas: dos repositorios completos, CL1A y CL1B, y dos repositorios parciales, QMA y QMB. Los repositorios completos se utilizan sólo para las definiciones de clúster. QMA se designa como host de temas de clúster. Las suscripciones duraderas se definen en QMA y QMB.

Nota: El ejemplo está codificado para Windows. Debe volver a codificar Create qmgrs.bat y Create pub.bat para configurar y probar el ejemplo en otras plataformas.

1. Cree los archivos de script.
   a. Create topics.tst
   b. Crear wildsubs.tst
   c. Crear fullsubs.tst
   d. Crear qmgrs.bat
   e. crear pub.bat
2. Ejecute Create qmgrs.bat para crear la configuración.

```
qmgrs
```

Cree los temas de Figura 23 en la página 81. El script de la figura 5 crea los temas de clúster `Sports/Football` y `Sports/Rugby`.

Nota: La opción REPLACE no sustituye las propiedades TOPICSTR de un tema. TOPICSTR es una propiedad que normalmente cambia en el ejemplo para probar distintos árboles de temas. Para cambiar temas, suprima primero el tema.

```
DELETE TOPIC ('Sports')
DELETE TOPIC ('Football')
DELETE TOPIC ('Arsenal')
DELETE TOPIC ('Blackburn')
DELETE TOPIC ('Chelsea')
DELETE TOPIC ('Rugby')
DELETE TOPIC ('Leeds')
DELETE TOPIC ('Wigan')
DELETE TOPIC ('Warrington')
DELETE TOPIC ('St. Helens')

DEFINE TOPIC ('Sports')   TOPICSTR('Sports')
DEFINE TOPIC ('Football')  TOPICSTR('Sports/Football') CLUSTER(CL1) WILDCARD(BLOCK)
DEFINE TOPIC ('Arsenal')  TOPICSTR('Sports/Football/Arsenal')
DEFINE TOPIC ('Blackburn') TOPICSTR('Sports/Football/Blackburn')
DEFINE TOPIC ('Chelsea')  TOPICSTR('Sports/Football/Chelsea')
DEFINE TOPIC ('Rugby')   TOPICSTR('Sports/Rugby')  CLUSTER(CL1)
DEFINE TOPIC ('Leeds')   TOPICSTR('Sports/Rugby/Leeds')
DEFINE TOPIC ('Wigan')   TOPICSTR('Sports/Rugby/Wigan')
DEFINE TOPIC ('Warrington') TOPICSTR('Sports/Rugby/Warrington')
DEFINE TOPIC ('St. Helens') TOPICSTR('Sports/Rugby/St. Helens')
```

Figura 38. Suprimir y crear temas: topics.tst

Nota: Suprima los temas, ya que REPLACE no sustituye series de tema.

Cree suscripciones con comodines. Los comodines corresponden a los temas con objetos de temas en Figura 23 en la página 81. Cree una cola para cada suscripción. Las colas se borran y las suscripciones se suprimen cuando se ejecuta o vuelve a ejecutar el script.

Nota: La opción REPLACE no sustituye las propiedades TOPICOBJ o TOPICSTR de una suscripción. TOPICOBJ o TOPICSTR son las propiedades que cambian normalmente en el ejemplo para probar distintas suscripciones. Para cambiarlas, suprima primero la suscripción.

```
DEFINE QLOCAL(QSPORTS)  REPLACE
DEFINE QLOCAL(QSARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS)  REPLACE
CLEAR QLOCAL(QSPORTS)
CLEAR QLOCAL(QSARSENAL)
CLEAR QLOCAL(QSLEEDS)

DELETE SUB  (SPORTS)
DELETE SUB  (SARSENAL)
DELETE SUB  (SLEEDS)
DEFINE SUB  (SPORTS)  TOPICSTR('Sports/#')     DEST(QSPORTS)
DEFINE SUB  (SARSENAL) TOPICSTR('Sports/+/Arsenal') DEST(QSARSENAL)
DEFINE SUB  (SLEEDS)  TOPICSTR('Sports/+/Leeds')  DEST(QSLEEDS)
```

Figura 39. Cree suscripciones comodín: wildsubs.tst

Cree suscripciones que hagan referencia a lo objetos de tema de clúster.

Nota:

El delimitador, `/`, se inserta automáticamente entre la serie de tema a la que hace referencia `TOPICOBJ` y la serie de tema definida por `TOPICSTR`.

La definición, `DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL)` crea la misma suscripción. `TOPICOBJ` se utiliza como método rápido para hacer referencia a la serie de tema que ya se ha definido. La suscripción, cuando se crea, ya no hace referencia al objeto de tema.

```
DEFINE QLOCAL(QFARSENAL) REPLACE
DEFINE QLOCAL(QRLEEDS)  REPLACE
CLEAR QLOCAL(QFARSENAL)
CLEAR QLOCAL(QRLEEDS)

DELETE SUB  (FARSENAL)
DELETE SUB  (RLEEDS)
DEFINE SUB  (FARSENAL) TOPICOBJ('Football') TOPICSTR('Arsenal') DEST(QFARSENAL)
DEFINE SUB  (RLEEDS)  TOPICOBJ('Rugby')  TOPICSTR('Leeds')  DEST(QRLEEDS)
```

Figura 40. Suprima y cree suscripciones: fullsubs.tst

Cree un clúster con dos depósitos. Cree dos repositorios parciales para publicación y suscripción. Vuelva a ejecutar el script para suprimirlo todo y empezar de nuevo. El script también crea la jerarquía de temas y las suscripciones comodín iniciales.

Nota:

En otras plataformas, escriba un script parecido o escriba todos los mandatos. El uso de un script permite suprimirlo todo con mayor rapidez y empezar de nuevo con una configuración idéntica.

```
@echo off
set port.CL1B=1421
set port.CLIA=1420
for %%A in (CL1A CL1B QMA QMB) do call :createQM %%A
call :configureQM CL1A CL1B %port.CL1B% full
call :configureQM CL1B CL1A %port.CL1A% full
for %%A in (QMA QMB) do call :configureQM %%A CL1A %port.CL1A% partial
for %%A in (topics.tst wildsubs.tst) do runmqsc QMA < %%A
for %%A in (wildsubs.tst) do runmqsc QMB < %%A
goto:eof

:createQM
echo Configure Queue manager %1
endmqm -p %1
for %%B in (dlt crt str) do %%Bmqm %1
goto:eof

:configureQM
if %1==CL1A set p=1420
if %1==CL1B set p=1421
if %1==QMA set p=1422
if %1==QMB set p=1423
echo configure %1 on port %p% connected to repository %2 on port %3 as %4 repository
echo DEFINE LISTENER(LST%1) TRPTYPE(TCP) PORT(%p%) CONTROL(QMGR) REPLACE | runmqsc %1
echo START LISTENER(LST%1) | runmqsc %1
if full==%4 echo ALTER QMGR REPOS(CL1) DEADQ(SYSTEM.DEAD.LETTER.QUEUE) | runmqsc %1
echo DEFINE CHANNEL(TO.%2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('LOCALHOST(%3)') CLUSTER(CL1)
REPLACE | runmqsc %1
echo DEFINE CHANNEL(TO.%1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('LOCALHOST(%p%)')
CLUSTER(CL1) REPLACE | runmqsc %1
goto:eof
```

Figura 41. Cree gestores de colas: qmgrs.bat

Actualice la configuración añadiendo las suscripciones a los temas de clúster.

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

Figura 42. Actualice las suscripciones: upsubs.bat

Ejecute pub.bat, con un gestor de colas como parámetro, para publicar mensajes que contengan la serie de tema de publicación. Pub.bat utiliza el programa de ejemplo **amqspub**.

```
@echo off
@rem Provide queue manager name as a parameter
set S=Sports
set S=6 Sports/Football Sports/Football/Arsenal
set S=6 Sports/Rugby Sports/Rugby/Leeds
for %%B in (6) do echo %%B | amqspub %%B %1
```

Figura 43. Publique: pub.bat

Conceptos relacionados

Suscripción de comodín y publicaciones retenidas

Ámbito de la publicación

Cuando se configura un clúster o jerarquía de publicación/suscripción, el ámbito de una publicación controla si los gestores de colas reenvían una publicación a gestores de colas remotos. Utilice el atributo de tema **PUBSCOPE** para administrar el ámbito de las publicaciones.

Si una publicación no se reenvía a los gestores de colas remotos, sólo los suscriptores locales recibirán la publicación.

Cuando se utiliza un clúster de publicación/suscripción, el ámbito de las publicaciones está controlado principalmente por la definición de objetos de tema en clúster en determinados puntos del árbol de

temas. Debe configurarse el ámbito de publicación para que permita el flujo de publicaciones a otros gestores de colas del clúster. Solo debería restringirse el ámbito de publicación de un tema en clúster cuando se necesita un control de grano fin de determinados temas en gestores de colas concretos.

Cuando se utiliza una jerarquía de publicación/suscripción, el ámbito de las publicaciones está controlado principalmente por este atributo junto con el atributo de ámbito de suscripción.

El atributo **PUBSCOPE** se utiliza para determinar el ámbito de publicaciones realizadas en un tema específico. Puede establecer el atributo en uno de los valores siguientes:

QMGR
La publicación se entrega únicamente a los suscriptores locales. Estas publicaciones se denominan *publicaciones locales*. Las publicaciones locales no se reenvían a gestores de colas remotos y, por lo tanto, los suscriptores conectados a gestores de colas remotos no las reciben.

TODOS
La publicación se entrega a suscriptores locales y a suscriptores conectados a gestores de colas remotos en un clúster o jerarquía de publicación/suscripción. Estas publicaciones se denominan *publicaciones globales*.

ASPARENT
Utilice el valor **PUBSCOPE** del tema padre en el árbol de temas.

Los publicadores también pueden especificar si una publicación es local o global utilizando la opción de transmitir mensajes `MQPMO_SCOPE_QMGR`. Si se utiliza esta opción, altera temporalmente cualquier comportamiento que se ha establecido utilizando el atributo de tema **PUBSCOPE**.

Conceptos relacionados

"Objetos de tema administrativo" en la página 78
Mediante un objeto de tema administrativo puede asignar a los temas atributos específicos que no son predeterminados.

Tareas relacionadas

Configuración de redes de publicación/suscripción distribuidas

Ámbito de la suscripción

El ámbito de una suscripción controla si una suscripción en un gestor de cola, recibe publicaciones publicadas en otro gestor de colas en una jerarquía o clúster de publicación/suscripción o sólo publicaciones desde publicadores locales.

Limitar el ámbito de suscripción a un gestor de colas detiene las suscripciones de proxy para que se reenvíen a otros gestores de cola en la topología de publicación/suscripción. Esto reduce el tráfico de mensajería de publicación/suscripción entre gestores de cola.

Cuando se utiliza un clúster de publicación/suscripción, el ámbito de las suscripciones está controlado principalmente por la definición de objetos de tema en clúster en determinados puntos del árbol de temas. Debe configurarse el ámbito de suscripción para que permita el flujo de suscripciones de proxy a otros gestores de colas del clúster. Solo debería restringirse el ámbito de suscripción de un tema en clúster cuando se necesita un control de grano fin de determinados temas en gestores de colas concretos.

Cuando se utiliza una jerarquía de publicación/suscripción, el ámbito de las suscripciones está controlado principalmente por este atributo junto con el atributo de ámbito de publicación.

El atributo de tema **SUBSCOPE** se utiliza para determinar el ámbito de las suscripciones realizadas a un tema específico. Puede establecer el atributo en uno de los valores siguientes:

QMGR
Una suscripción sólo recibe publicaciones locales y las suscripciones de proxy no se propagan a gestores de cola remotos.

TODOS
Una suscripción proxy se propaga a gestores de colas remotos en una jerarquía o clúster de publicación/suscripción y el suscriptor recibe publicaciones locales y remotas.

ASPARENT
Utilice el valor **SUBSCOPE** del tema padre en el árbol de temas.

Cuando el ámbito de suscripción para un tema se establece en ALL, ya sea directamente o resuelto a través de ASPARENT, las suscripciones individuales a ese tema pueden restringir su ámbito a QMGR especificando MQSO_SCOPE_QMGR al crear la suscripción. Una suscripción a un tema que tenga un ámbito de QMGR no puede ampliar el ámbito a ALL.

Conceptos relacionados

"Objetos de tema administrativo" en la página 78
Mediante un objeto de tema administrativo puede asignar a los temas atributos específicos que no son predeterminados.

Tareas relacionadas

Configuración de redes de publicación/suscripción distribuidas

Espacios de temas

Un espacio de temas es un conjunto de temas al que uno puede suscribirse y en el que se puede publicar. Un gestor de colas en una topología de publicación/suscripción distribuida tiene un espacio de temas que en potencia incluye temas que se han suscrito y en los que se ha publicado en los gestores de colas conectados de dicha topología.

Nota: Para obtener una descripción general de los temas en un gestor de colas como, por ejemplo, objetos de tema administrativos, cadenas de tema y árboles de temas, consulte "Temas" en la página 69. En adelante, cuando se mencione *temas* en este artículo se hará referencia a *cadenas de tema* a menos que se indique otra cosa.

Los temas se crean inicialmente de alguna de las maneras siguientes:

- administrativamente, cuando se define un objeto de tema o una suscripción duradera
- dinámicamente, cuando una aplicación crea dinámicamente una publicación o una suscripción en un tema nuevo.

Los temas se propagan a otros gestores de colas tanto a través de suscripciones proxy como mediante la creación de objetos de tema de clúster administrativo. Las suscripciones proxy dan como resultado que se reenvíen las publicaciones desde el gestor de colas al que un publicador está conectado, a los gestores de colas de suscriptores.

Las suscripciones proxy se propagan entre todos los gestores de colas que están conectados entre sí mediante relaciones padre-hijo en una jerarquía de gestor de colas. El resultado es que puede suscribirse en un gestor de colas a un tema definido en cualquier otro gestor de colas de la jerarquía. Siempre que haya una vía de acceso conectada entre los gestores de colas, no importa cómo están conectados los gestores de colas.

Las suscripciones de proxy también se propagan en las suscripciones a temas de clúster en un clúster de publicación/suscripción. Un tema de clúster es un tema que está conectado a un objeto de tema que tiene el atributo **CLUSTER**, o hereda el atributo de su padre. Los temas que no son temas de clúster son conocidos como temas locales y no se replican en el clúster. No se propaga ninguna suscripción proxy al clúster desde suscripciones a temas locales.

En resumen, las suscripciones proxy se crean para los suscriptores en dos circunstancias.

1. Un gestor de colas es un miembro de una jerarquía, y una suscripción proxy se reenvía al padre e hijos del gestor de colas.
2. Un gestor de colas es un miembro de un clúster y la serie de tema de suscripción se resuelve en un tema que está asociado a un objeto de tema de clúster. Cuando el tema es un tema de clúster de *direccionamiento directo*, las suscripciones de proxy se reenvían a todos los miembros del clúster. Cuando el tema es un tema de clúster de *direccionamiento de host de temas*, las suscripciones de proxy solo se reenvían a los gestores de colas del clúster que hayan definido el objeto de tema en clúster. Para obtener más información, consulte "Clústeres de publicación/suscripción" en la página 93.

Si un gestor de colas es un miembro de un clúster y una jerarquía, las suscripciones proxy son propagadas por ambos mecanismos sin entregar publicaciones duplicadas al suscriptor.

Los espacios de temas de tres topologías de publicación/suscripción se describen en la lista siguiente:

- "Caso 1. Clústeres de publicación/suscripción" en la página 107.
- "Caso 2. Jerarquías de publicación/suscripción" en la página 107.

En temas independientes, las tareas de configuración siguientes describen cómo combinar espacios de temas.

- Creación de un único espacio de temas en un clúster de publicación/suscripción.
- Combinación de espacios de temas de múltiples clústeres.
- Combinación y aislamiento de espacios de temas en múltiples clústeres.
- Publicación y suscripción en espacios de temas en múltiples clústeres.

Caso 1. Clústeres de publicación/suscripción

En el ejemplo, presuponga que el gestor de colas no está conectado a una jerarquía de publicación/suscripción.

Si un gestor de colas es miembro de un clúster de publicación/suscripción, su espacio de temas se compone de temas locales y temas de clúster. Los temas locales se asocian con objetos de tema sin el atributo **CLUSTER**. Si un gestor de colas tiene definiciones de objetos de tema local, su espacio de temas es diferente de otro gestor de colas del clúster que también tenga sus propios objetos de tema definidos localmente.

En un clúster de publicación/suscripción, no puede suscribirse a un tema definido en otro gestor de colas, a menos que el tema al que se suscribe se resuelve en un objeto de tema de clúster.

Cuando se necesitan las mismas definiciones nombradas de un objeto de tema de clúster en múltiples gestores de colas (por ejemplo, cuando se utiliza un *direccionamiento de host de temas*) es importante que todas las definiciones coincidan donde sea necesario. Puede obtener información adicional consultando Creación de un único espacio de temas en un clúster de publicación/suscripción.

Una definición local de un objeto de tema, si la definición es para un tema de clúster o un tema local, tiene prioridad sobre el mismo objeto de tema definido en otro lugar del clúster. Se utiliza el tema definido localmente, incluso si el objeto definido en otro lugar es más reciente.

Es importante que un objeto de tema de clúster esté asociado con la misma serie de tema en todas partes del clúster. No puede modificar la serie de tema con la que está asociado un objeto de tema. Para asociar el mismo objeto de tema con una serie de tema diferente, debe suprimir el objeto de tema y volver a crearlo con la nueva serie de tema. Si el tema está en clúster, tiene el efecto de suprimir las copias del objeto de tema almacenado en los demás miembros del clúster y, a continuación, crear copias del nuevo objeto de tema en todas partes del clúster. Todas las copias del objeto de tema hacen referencia a la misma serie de tema.

Es posible crear accidentalmente dos definiciones del mismo objeto de tema designado en diferentes gestores de colas en el clúster, con diferentes series de tema. Esto puede dar como resultado un comportamiento confuso, porque varias definiciones del mismo objeto de tema con distintas series de tema pueden producir resultados diferentes en función de cómo y dónde se hace referencia al tema. Consulte Múltiples definiciones de temas de clúster con el mismo nombre para obtener información sobre este aspecto importante.

Caso 2. Jerarquías de publicación/suscripción

En el ejemplo, presuponga que el gestor de colas no es miembro de un clúster de publicación/suscripción.

En IBM MQ, si un gestor de colas es miembro de una jerarquía de publicación/suscripción, su espacio de temas consta de todos los temas definidos localmente y en gestores de colas conectados. El espacio de temas de todos los gestores de colas de una jerarquía es el mismo. No hay ninguna división de los temas en temas locales y temas de globales.

Establezca una de las opciones **PUBSCOPE** y **SUBSCOPE** en QMGR, para impedir que una publicación en un tema fluya de un publicador a un suscriptor conectado a gestores de colas distintos en la jerarquía.

Supongamos que define un objeto de tema `Alabama` con la serie de tema `USA/Alabama` en el gestor de colas QMA. El resultado es el siguiente:

1. Ahora, el espacio de temas en QMA incluye el objeto de tema `Alabama` y la serie de tema `USA/Alabama`.
2. Una aplicación o un administrador puede crear una suscripción en QMA utilizando el nombre de objeto de tema `Alabama`.
3. Una aplicación puede crear una suscripción a cualquier tema, incluido `USA/Alabama`, en cualquier gestor de colas de la jerarquía. Si QMA no se ha definido localmente, el tema `USA/Alabama` se resuelve en el objeto de tema `SYSTEM.BASE.TOPIC`.

Conceptos relacionados

"Ámbito de la publicación" en la página 104
Cuando se configura un clúster o jerarquía de publicación/suscripción, el ámbito de una publicación controla si los gestores de colas reenvían una publicación a gestores de colas remotos. Utilice el atributo de tema **PUBSCOPE** para administrar el ámbito de las publicaciones.

"Ámbito de la suscripción" en la página 105
El ámbito de una suscripción controla si una suscripción en un gestor de cola, recibe publicaciones publicadas en otro gestor de colas en una jerarquía o clúster de publicación/suscripción o sólo publicaciones desde publicadores locales.

Tareas relacionadas

Configuración de redes de publicación/suscripción distribuidas

IBM MQ Multicast

IBM MQ Multicast ofrece baja latencia, alta diseminación y mensajería de multidifusión fiable.

La multidifusión es una forma eficaz de mensajería de publicación/suscripción porque se puede escalar a un elevado número de suscriptores sin efectos negativos en el rendimiento. IBM MQ permite una mensajería multidifusión fiable mediante el uso de acuses de recibo, acuses de recibo negativos y números de secuencia para lograr una mensajería de baja latencia y alta diseminación.

La entrega equilibrada de IBM MQ Multicast permite una entrega casi simultánea y garantiza que ningún destinatario tenga ventaja. Dado que IBM MQ Multicast utiliza la red para suministrar mensajes, no es necesario utilizar un motor de publicación/suscripción para diseminar los datos. Después de correlacionar un tema con una dirección de grupo, no es necesario utilizar un gestor de colas, porque los publicadores y los suscriptores pueden operar en una modalidad de igual a igual. Esto permite que la carga se reduzca en los servidores de gestores de colas y el servidor de gestores de colas deja de ser un punto de posibles anomalías.

Conceptos iniciales sobre la multidifusión

IBM MQ Multicast se puede integrar fácilmente en los sistemas y aplicaciones existentes utilizando el objeto Communication Information (COMMINFO). Dos campos de objeto TOPIC permiten la configuración rápida de objetos TOPIC existentes para dar soporte u omitir el tráfico de multidifusión.

Objetos necesarios para la multidifusión

La siguiente información ofrece una breve descripción general de los dos objetos necesarios para IBM MQ Multicast:

Objeto COMMINFO
El objeto COMMINFO contiene los atributos asociados con la transmisión multidifusión. Para obtener más información sobre los parámetros de objeto COMMINFO, consulte DEFINE COMMINFO.

El único campo de COMMINFO que se DEBE establecer es el nombre del objeto COMMINFO. Este nombre se utiliza para identificar el objeto COMMINFO en un tema. El campo **GRPADDR** del objeto COMMINFO debe comprobarse para asegurarse de que el valor es una dirección de grupo de multidifusión válida.

Objeto TOPIC
Un tema es el asunto de la información que se publica en un mensaje de publicación/suscripción y un tema se define creando un objeto TOPIC. Para obtener más información sobre los parámetros de objeto TOPIC, consulte DEFINE TOPIC.

Se pueden utilizar temas existentes con la multidifusión cambiando los valores de los parámetros siguientes del objeto TOPIC: **COMMINFO** y **MCAST**.

- **COMMINFO** Este parámetro especifica el nombre del objeto de información de comunicación de multidifusión.
- **MCAST** Este parámetro especifica si la multidifusión está permitida en esta posición del árbol de temas. De forma predeterminada, **MCAST** se establece en ASPARENT, lo que significa que el atributo de multidifusión del tema se hereda del padre. Establecer **MCAST** en HABILITADO permite el tráfico de multidifusión en este nodo.

Redes y temas de multidifusión

La siguiente información es una visión general de lo que sucede en las suscripciones con diferentes tipos de suscripción y definición de tema. En todos estos ejemplos se presupone que el parámetro **COMMINFO** del objeto TOPIC se ha establecido en el nombre de un objeto COMMINFO válido:

Tema establecido en multidifusión habilitado
Si el parámetro **MCAST** de la serie de tema se establece en ENABLED, las suscripciones de los clientes con capacidad de multidifusión están permitidas y se realiza una suscripción de multidifusión a menos que:

- Sea una suscripción duradera desde un cliente con capacidad de multidifusión.
- Sea una suscripción no gestionada desde un cliente con capacidad de multidifusión.
- Sea una suscripción desde un cliente sin capacidad de multidifusión.

En estos casos, se realiza una suscripción sin capacidad de multidifusión y suscripciones se degradan a una publicación/suscripción normal.

Tema establecido en multidifusión inhabilitado
Si el parámetro de la serie de tema **MCAST** se establece en DISABLED, siempre se realiza una suscripción sin multidifusión y suscripciones se degradan a una publicación/suscripción normal.

Tema establecido sólo en multidifusión
Si el parámetro **MCAST** de la serie de tema se establece en ONLY, las suscripciones de los clientes con capacidad de multidifusión están permitidas y se realiza una suscripción de multidifusión a menos que:

- Es una suscripción duradera: las suscripciones duraderas se rechazan con el código de razón 2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
- Es una suscripción no gestionada: las suscripciones no gestionadas se rechazan con el código de razón 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
- Es una suscripción de un cliente sin capacidad multidifusión: estas suscripciones se rechazan con el código de razón 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
- Es una suscripción de una aplicación enlazada localmente: estas suscripciones se rechazan con el código de razón 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY

Windows Linux AIX

MQ TelemetryVisión general de

MQ Telemetry incluye un servicio de telemetría (MQXR) que forma parte de un gestor de colas, los clientes de telemetría que puede escribir usted mismo o descargar de forma gratuita y las interfaces

administrativas de línea de mandatos y explorador. La telemetría se refiere a la recopilación de datos de una amplia gama de dispositivos remotos y a la administración de los mismos. Con MQ Telemetry puede integrar la recopilación de datos y el control de dispositivos con aplicaciones web.

El MQ Telemetry es un componente de IBM MQ. La actualización de estas versiones es básicamente la instalación de una versión posterior de IBM MQ.

Las aplicaciones de ejemplo siguen estando disponibles de forma gratuita en Eclipse Paho y MQTT.org. Consulte Programas de ejemplo deIBM MQ Telemetry Transport.

Puesto que MQ Telemetry es un componente de IBM MQ, MQ Telemetry se puede instalar con el producto principal o después de que se haya instalado el producto principal. Para obtener información sobre la migración, consulte Migración de MQ Telemetry en Windows y Migración de MQ Telemetry en Linux.

Los componentes incluidos en MQ Telemetry son los siguientes:

Canales de telemetría
Utilice los canales de telemetría para gestionar la conexión de clientes MQTT a IBM MQ. Los canales de telemetría utilizan nuevos objetos de IBM MQ , como `SYSTEM.MQTT.TRANSMIT.QUEUE`, para interactuar con IBM MQ.

Servicio de telemetría (MQXR)
Los clientes de MQTT utilizan el servicio de telemetría de `SYSTEM.MQXR.SERVICE` para conectarse a canales de telemetría.

Soporte de IBM MQ Explorer para MQ Telemetry
MQ Telemetry se puede administrar utilizando IBM MQ Explorer.

Documentación
La documentación de MQ Telemetry se incluye en la documentación estándar del producto IBM MQ . La documentación de SDK para Java y clientes C se suministra en la documentación del producto y como Javadoc y HTML.

Conceptos de telemetría

Puede recopilar información de todo el entorno que le rodea para tomar decisiones sobre qué se debe hacer. Como consumidor puede comprobar qué tiene en el almacén antes de decidir qué comida comprar. Puede conocer cuánto tiempo dura un viaje si se marcha en ese momento, antes de reservar una conexión. Puede comprobar qué síntomas tiene antes de decidir si ir al médico. Puede comprobar cuándo va a llegar el autobús y decidir así si va esperarlo. La información con la que tomará estas decisiones se le presentará directamente en los medidores o dispositivos, de forma impresa o en pantalla. No importa dónde esté ni cuándo necesite esta información, puede recopilar estos datos, reunirlos, analizarlos y actuar según ellos.

Si las fuentes de información están muy dispersas o inaccesibles, resulta difícil y costoso reunir información precisa. Si desea hacer muchos cambios o le resulta difícil hacerlos, éstos no se realizarán o lo harán cuando ya no sean tan efectivos.

¿Qué ocurre si se reduce en gran parte los costes de la recopilación de la información y del control de dispositivos que estén muy dispersos, conectándolos con tecnología digital a Internet? Podrá analizar la información mediante recursos de Internet y de la empresa. Tendrá más oportunidades de tomar decisiones fundamentadas en la información que ha obtenido, y actuar según convenga.

Las tendencias tecnológicas y las presiones económicas y del entorno provocan que se den estos cambios:

1. Se reduce el coste de conexión y control de sensores y mecanismos de acceso gracias a la estandarización y conexión a procesadores digitales de bajo coste.
2. Internet y las tecnologías relacionadas se utilizan cada vez más para conectar dispositivos. En algunos países, los teléfonos móviles han superado a los ordenadores personales en el número de conexiones a aplicaciones de Internet. Otros dispositivos, seguramente, seguirán el mismo camino.

3. Internet y las tecnologías relacionadas facilitan mucho que una aplicación pueda obtener datos. El fácil acceso a los datos está haciendo que se utilicen los análisis de datos para transformar los datos de los sensores en información útil para muchas soluciones.
4. El uso inteligente de los recursos es a menudo una forma más rápida y barata de reducir las emisiones de dióxido de carbono y los costes. Las alternativas: encontrar nuevos recursos o desarrollar nuevas tecnologías para utilizar los recursos existentes pueden ser la solución a largo plazo. A corto plazo, el desarrollo de nuevas tecnologías o la búsqueda de nuevos recursos resulta a menudo más arriesgado, lento y costoso que mejorar las soluciones con las que ya se cuenta.

Ejemplo

A continuación se muestra un ejemplo de cómo estas tendencias crean nuevas oportunidades para interactuar de forma inteligente con el entorno.

El Convenio Internacional para la Seguridad de la Vida Humana en el Mar (SOLAS) necesita el Sistema de Identificación Automática (AIS) se despliegue en muchos buques. Es necesario en buques mercantes de más de 300 toneladas y buques de pasajeros. El sistema AIS es ante todo un sistema para evitar colisiones en el tráfico costero. Lo utilizan las autoridades marítimas para supervisar y controlar las aguas costeras.

Entusiastas de todo el mundo están desplegando estaciones de seguimiento AIS de bajo coste y subiendo información sobre el tráfico costero a Internet. Otros están creando aplicaciones que combinan la información de AIS con otra información de Internet. Los resultados se ponen en los sitios web, y se publican utilizando Twitter y SMS.

En una aplicación puede combinarse información procedente de estaciones AIS cerca de Southampton con información de propietarios de buques y geográfica. La aplicación suministra información en directo sobre llegadas y salidas de transbordadores a Twitter. Los usuarios habituales que utilizan los transbordadores entre Southampton y la Isla de Wight se suscriben al canal de noticias utilizando Twitter o SMS. Si el canal de información muestra que el transbordador llega tarde, los viajeros pueden retrasar su salida y tomar el transbordador cuando atraque más tarde que la hora de llegada programada.

Para obtener más ejemplos, consulte “Casos de uso de telemetría” en la página 113.

Tareas relacionadas

Instalación del MQ Telemetry
Administración de MQ Telemetry
Migración de MQ Telemetry en Windows
Migración de MQ Telemetry en Linux
Desarrollo de aplicaciones para MQ Telemetry
Resolución de problemas de MQ Telemetry

Referencia relacionada

Referencia de MQ Telemetry

Windows Linux AIX

Introducción a MQ Telemetry

Cada vez es más común que las personas, las empresas y los gobiernos quieran utilizar MQ Telemetry para interactuar de forma más inteligente con el entorno en el que viven y trabajan. MQ Telemetry conecta todo tipo de dispositivos a Internet y a la empresa, y reduce los costes de la creación de aplicaciones para dispositivos inteligentes.

¿Qué es MQ Telemetry?

- Es una característica de IBM MQ que amplía la estructura de mensajería universal que proporciona IBM MQ a un amplio rango de sensores remotos, mecanismos de acceso y dispositivos de telemetría. MQ Telemetry amplía IBM MQ para que pueda interconectar aplicaciones empresariales inteligentes, servicios y responsables de decisiones con redes de dispositivos instrumentados.
- Las partes principales de MQ Telemetry son:

El servicio de MQ Telemetry (MQXR).
Este servicio se ejecuta dentro del servidor IBM MQ y utiliza el protocolo IBM MQ Telemetry Transport (MQTT) para comunicarse con dispositivos de telemetría.

Las aplicaciones MQTT que se desarrollan.
Estas aplicaciones controlan la información que se transmite entre los dispositivos de telemetría y el gestor de colas de IBM MQ y las acciones que se toman en respuesta a dicha información. Como ayuda para la creación de estas aplicaciones, se utilizan las bibliotecas cliente de MQTT.

¿Qué puede hacer por mí?

- MQTT es un transporte de mensajería abierto que permite crear implementaciones de MQTT para una amplia gama de dispositivos.
- Los clientes MQTT se pueden ejecutar en pequeños dispositivos ligeros que tienen recursos limitados.
- MQTT funciona de forma eficaz en redes en las que el ancho de banda es bajo, donde el coste de enviar datos resulta caro o que pueden ser frágiles.
- La entrega de mensajes está asegurada y es independiente de la aplicación.
- Los programadores de aplicaciones no necesitan tener conocimientos de programación de comunicaciones.
- Los mensajes se pueden intercambiar con otras aplicaciones de mensajería. Estas pueden ser otra aplicación de telemetría, una MQI, JMS o una aplicación de mensajería empresarial.

¿Cómo lo utilizo?

- Se proporcionan scripts de ejemplo que funcionan con una aplicación cliente IBM MQ Telemetry Transport v3 de ejemplo (`mqttv3app.jar`). Consulte Programas de ejemplo de IBM MQ Telemetry Transport.
- Utilice IBM MQ Explorer y sus herramientas asociadas para administrar la característica de telemetría de IBM MQ.
- Utilice las bibliotecas como ayuda para la creación de aplicaciones MQTT que se conectan a un gestor de colas y que utilizan la mensajería de publicación/suscripción.
- Distribuya la aplicación y la biblioteca cliente al dispositivo en el que deba ejecutarse la aplicación.

¿Cómo funciona?

- MQTT es un protocolo de publicación y suscripción. Una aplicación de cliente MQTT puede publicar mensajes en un servidor MQTT o bien suscribirse a mensajes enviados por aplicaciones que se conectan a un servidor MQTT.
- Las aplicaciones cliente MQTT utilizan bibliotecas cliente que implementan el transporte de mensajes MQTT.
- Una aplicación de cliente MQTT básica funciona como un cliente MQ estándar pero se puede ejecutar en una variedad mucho más amplia de plataformas y redes.
- El servicio MQ Telemetry (MQXR) convierte un gestor de colas IBM MQ en un servidor MQTT.
- Cuando un gestor de colas de IBM MQ actúa como servidor MQTT, otras aplicaciones que se conectan al gestor de colas pueden suscribirse y recibir los mensajes del cliente de MQTT .
- El gestor de colas actúa como direccionador, distribuyendo los mensajes procedentes de las aplicaciones de publicación a las aplicaciones suscriptoras.
- Los mensajes puede distribuirse entre los diferentes tipos de aplicaciones cliente. Por ejemplo, entre los clientes de telemetría y los clientes JMS.

Nota: MQ Telemetry sustituye los nodos SCADA que se eliminaron de la versión 7 de WebSphere Message Broker (conocida ahora omo IBM Integration Bus) y se ejecuta en Windows, Linux y AIX.

Windows Linux AIX

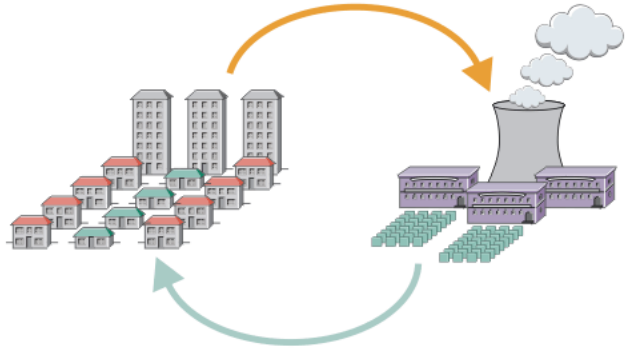
Casos de uso de telemetría

La telemetría es la detección automatizada, la medición de datos y el control de dispositivos remotos. Su principal ventaja es la transmisión de datos desde dispositivos a un punto de control central La telemetría también incluye el envío de información de configuración y control a los dispositivos.

MQ Telemetry conecta pequeños dispositivos mediante el MQTT protocol y conecta los dispositivos a otras aplicaciones utilizando IBM MQ. MQ Telemetry llena el hueco existente entre los dispositivos e internet facilitando la creación de "soluciones inteligentes". Las soluciones inteligentes ponen a disposición de las aplicaciones que supervisan y controlan dispositivos la abundante información que puede encontrarse en Internet y en las aplicaciones de empresas.

En los diagramas siguientes muestran algunos usos típicos de MQ Telemetry:

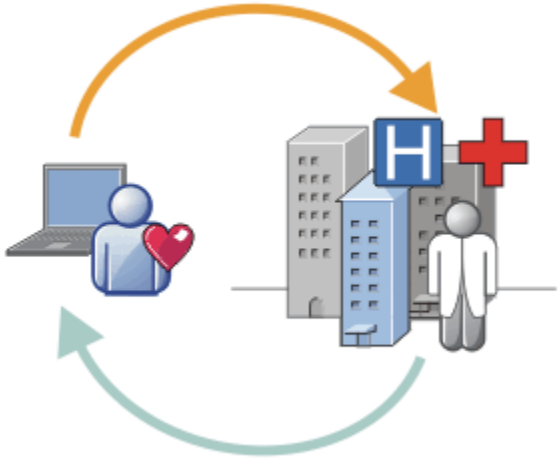
Telemetría: Electricidad inteligente



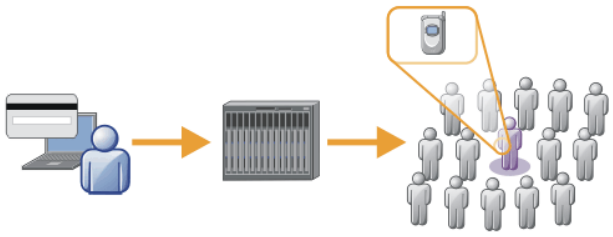
- Mensaje de MQTT que contiene los datos de uso energético enviados al proveedor de servicios.
- MQ Telemetry envía MANDATOS DE CONTROL basados en el análisis de datos de uso energético.
- Para obtener más información, consulte el caso siguiente: “Caso de uso de telemetría: supervisión y control de la electricidad del hogar” en la página 115

Telemetría: Servicios de salud inteligente

- MQ Telemetry envía datos del estado de salud a su Hospital y Doctor.
- Se pueden enviar alertas o comentarios de mensajes MQTT basados en el análisis de datos sanitarios.
- Para obtener más información, consulte el caso siguiente: “Caso de uso de telemetría: supervisión del paciente en el hogar” en la página 114



Telemetría: Uno entre la multitud



- Una simple tarjeta de transacciones se envía al servidor del banco.
- MQ Telemetry identifica a una persona entre miles, alertando al cliente de que se ha utilizado su tarjeta.
- MQ Telemetry puede utilizar la entrada más sencilla de información y localizar a esa persona.

Los casos de uso descritos en los subtemas son ejemplos reales. Ilustran algunas formas de utilizar la telemetría y algunos de los problemas comunes que esta tecnología debe resolver.

Windows Linux AIX Caso de uso de telemetría: supervisión del paciente en el hogar

En la colaboración entre IBM y un proveedor de asistencia médica en un sistema de cuidados cardíacos para el paciente, un desfibrilador cardioversor se comunica con un hospital. Los datos sobre el paciente y el dispositivo implantado se transfieren utilizando la telemetría RF en el dispositivo MQTT situado en casa del paciente.

La transferencia suele realizarse durante la noche, a un transmisor situado junto a la cama. El transmisor transfiere los datos de forma segura a través del teléfono del sistema al hospital, donde se analizan los datos.

El sistema reduce el número de visitas que un paciente debe realizar al médico. Detecta cuándo el paciente o el dispositivo necesitan algún tipo de atención y, en caso de emergencia, avisa al médico de guardia.

La colaboración entre IBM y el proveedor de asistencia médica presenta características que son comunes a un número de casos de uso de telemetría:

Imperceptibilidad

El dispositivo no necesita ninguna intervención del usuario aparte de proporcionarle electricidad, una línea de teléfono y pasar una parte del día cerca del mismo. Su funcionamiento es fiable y es fácil de utilizar.

Para eliminar la necesidad de que el paciente tenga que configurar el dispositivo, el proveedor del dispositivo preconfigura el mismo. El paciente sólo tiene que enchufarlo. El hecho de que el paciente no tenga que configurarlo, simplifica el funcionamiento del dispositivo y reduce la posibilidad de que se configure de forma incorrecta.

El cliente MQTT está integrado como parte del dispositivo. El desarrollador del dispositivo incorpora la implementación del cliente MQTT en el dispositivo y el desarrollador o proveedor configura el cliente MQTT como parte de la preconfiguración.

El cliente MQTT se proporciona como un archivo JAR de Java SE, que el desarrollador incluye en su aplicación Java. En entornos que no son Java, como por ejemplo, éste, el desarrollador del dispositivo puede implementar un cliente con un lenguaje diferente utilizando los formatos y el protocolo MQTT publicados. Como alternativa, el desarrollador puede utilizar uno de los clientes C suministrados como bibliotecas compartidas para plataformas Windows, Linux y ARM.

Conexión asimétrica

La comunicación entre el desfibrilador y el hospital tiene características de red asimétrica. Se utilizan dos redes diferentes para solucionar los diferentes problemas de la recopilación de datos del paciente y su envío al hospital. Entre el paciente y el dispositivo MQTT, se utiliza una red RF de baja potencia y corto alcance. El transmisor se conecta con el hospital utilizando una conexión VPN con TCP/IP a través de una línea telefónica de ancho de banda bajo.

Es a menudo poco viable encontrar una forma de conectar cada dispositivo directamente a una red de protocolo de Internet. Una solución común es utilizar dos redes, conectadas mediante un concentrador. El dispositivo MQTT es un concentrador sencillo, que almacena información del paciente y la reenvía al hospital.

Seguridad

El médico debe poder confiar en la autenticidad de los datos del paciente, y el paciente desea que se respete la privacidad de sus datos.

En algunos casos, basta con cifrar la conexión utilizando VPN o TLS. En otras situaciones, es recomendable mantener la seguridad de los datos incluso después de haberse almacenado.

En ocasiones, el dispositivo de telemetría no es seguro. Por ejemplo, si se comparte una casa. El usuario del dispositivo debe identificarse para garantizar así que los datos provengan del paciente correcto. El propio dispositivo debe autenticarse en el servidor utilizando TLS y el servidor debe hacer lo mismo con el dispositivo.

El canal de telemetría entre el dispositivo y el gestor de colas da soporte a JAAS para la autenticación de usuarios y a TLS para el cifrado de la comunicación y la autenticación del dispositivo. El acceso a una publicación está controlado por el gestor de autorizaciones sobre objetos en IBM MQ.

El identificador que se utiliza para autenticar al usuario puede correlacionarse con un identificador diferente como, por ejemplo, una identidad de paciente común. Un identificador común simplifica la configuración de la autorización en los temas de publicación de IBM MQ.

Conectividad

La conexión entre el dispositivo MQTT y el hospital utiliza la línea de acceso telefónico y funciona con un ancho de banda bajo de 300 baudios.

Para funcionar correctamente a 300 baudios, el MQTT protocol solamente añade unos cuantos bytes adicionales a un mensaje además de las cabeceras TCP/IP.

El MQTT protocol proporciona una única mensajería de tipo *enviar y olvidar*, que mantiene las latencias bajas. También puede utilizar varias transmisiones para garantizar la entrega *al menos una vez* y *exactamente una vez* si la entrega garantizada es más importante que el tiempo de respuesta. Para garantizar la entrega, los mensajes se almacenan en el dispositivo hasta que se han entregado de forma satisfactoria. Si el dispositivo está conectado mediante una conexión inalámbrica, resulta especialmente útil la entrega garantizada.

Escalabilidad

Los dispositivos de telemetría se despliegan normalmente en grandes cantidades, de decenas de miles a millones.

La conexión de tantos dispositivos a un sistema requiere grandes exigencias de una solución. Existen exigencias empresariales como el coste de los dispositivos y del software y exigencias administrativas para la gestión de licencias, dispositivos y usuarios. Entre las exigencias técnicas se incluyen la carga en la red y en los servidores.

La apertura de conexiones consume más recursos de red que el mantenimiento de conexiones abiertas. Sin embargo, en un caso como éste que utiliza líneas de teléfono, el gasto de conexiones reside en no dejar las conexiones abiertas más de lo necesario. Los datos de las transferencias son, principalmente, de tipo de proceso por lotes. Se puede planificar que las conexiones se realicen durante la noche para evitar una franja de hora punta repentina de conexiones a la hora de acostarse.

En el lado del cliente, la mínima configuración necesaria de éste resulta útil para la escalabilidad de los clientes. El cliente MQTT está incorporado en el dispositivo. No existen requisitos de configuración ni debe integrarse ningún paso de aceptación de la licencia cliente MQTT en el despliegue de los dispositivos a los pacientes.

En el servidor, MQ Telemetry tiene un objetivo inicial de 50.000 conexiones abiertas por cada gestor de colas.

Las conexiones se gestionan utilizando IBM MQ Explorer. IBM MQ Explorer filtra las conexiones para que se visualicen en un número gestionable. Si se elige correctamente una planificación de asignaciones de identificadores a clientes, es posible filtrar las conexiones según la ubicación geográfica o alfabéticamente por el nombre del paciente.

Windows Linux AIX Caso de uso de telemetría: supervisión y control de la electricidad del hogar

Los medidores inteligentes recopilan más detalles sobre el consumo de electricidad que los tradicionales.

Los medidores inteligentes a menudo van unidos a una red local de telemetría para poder supervisar y controlar los diferentes aparatos que hay en la casa. Algunos están conectados de forma remota para que puedan supervisarse y controlarse a distancia.

La conexión remota puede establecerse de forma individual, la puede establecer una compañía de suministro eléctrico, o se puede establecer a través de un punto de control central. El punto de control central puede leer la utilización que se ha hecho de la electricidad y proporcionar datos del uso. Puede proporcionar datos que afectan a la utilización como, por ejemplo, información continuada sobre los costes y las condiciones meteorológicas. Puede limitar la carga para mejorar la eficacia global del suministro de electricidad.

Los medidores inteligentes están empezando a estar muy extendidos. Por ejemplo, el gobierno del Reino Unido está considerando el despliegue de medidores inteligentes en cada hogar del país para 2020.

Los casos de uso sobre mediciones en los hogares tienen una serie de características comunes:

Imperceptibilidad

A menos que el usuario desee expresamente intervenir en el ahorro de electricidad utilizando el medidor, éste no necesita que el usuario intervenga. No debe reducir la fiabilidad del suministro de electricidad a aparatos individuales.

Un cliente MQTT puede estar incorporado en el software desplegado con el medidor y no requiere una instalación o configuración aparte.

Conexión asimétrica

La comunicación entre los aparatos y el medidor inteligente exige diferentes estándares de conectividad entre el medidor y el punto de conexión remoto.

La conexión del medidor inteligente al aparato debe ser de alta disponibilidad y cumplir con estándares de red de área doméstica.

Es probable que la red remota utilice varias conexiones físicas. Algunas de ellas, como un teléfono móvil, tienen un alto coste de transmisión y pueden ser intermitentes. La especificación MQTT v3 está destinada a las conexiones remotas y las conexiones entre adaptadores locales y el medidor inteligente.

La conexión entre los enchufes y los aparatos y el medidor utiliza una red de área doméstica, por ejemplo Zigbee. MQTT para redes de sensores (MQTT-S) se ha diseñado para funcionar con Zigbee y otros protocolos de red con un ancho de banda reducido. MQ Telemetry no ofrece soporte directamente a MQTT-S. Requiere una pasarela para conectar MQTT-S a MQTT v3.

Al igual que ocurre con la supervisión del paciente en el hogar, las soluciones para supervisar y controlar la electricidad del hogar necesitan múltiples redes conectadas que utilizan el medidor inteligente como un concentrador.

Seguridad

Existe una serie de cuestiones sobre seguridad relacionadas con los medidores inteligentes. Entre estas cuestiones se incluyen la no repudiación de la transmisión, la autorización de cualquier acción de control que se inicie y la privacidad de los datos del consumo de electricidad.

Para garantizar la privacidad, los datos transferidos entre el medidor y el punto de control remoto mediante MQTT se pueden cifrar mediante TLS. Para garantizar la autorización de las acciones de control, la conexión de MQTT entre el medidor y el punto de control remoto se puede autenticar mediante TLS.

Conectividad

La naturaleza física de la red remota puede variar considerablemente. Puede utilizar una conexión con ancho de banda existente o una red de móviles con alto coste de llamada y disponibilidad intermitente. Para un coste elevado, intermitente, las conexiones MQTT son un protocolo eficaz y fiable; consulte “Caso de uso de telemetría: supervisión del paciente en el hogar” en la página 114.

Escalabilidad

Las compañías eléctricas y puntos de control central planean, con el tiempo, desplegar decenas de millones de medidores inteligentes. En un principio, la cantidad de medidores por despliegue se cifra en decenas de cientos de miles. Este número es comparable con el destino inicial de MQTT de 50.000 conexiones cliente abiertas por cada gestor de colas.

Un aspecto importante de la arquitectura de la supervisión y control de la electricidad en el hogar es utilizar el medidor inteligente como un concentrador de redes. Cada adaptador de cada aparato es un sensor diferente. Al conectarlos a un concentrador local utilizando MQTT, el concentrador puede concentrar los flujos de datos a una única sesión TCP/IP con el punto de control central y almacenar también mensajes durante un corto periodo para superar las interrupciones de la sesión.

Las conexiones remotas deben dejarse abiertas en los casos de uso de electricidad en el hogar por dos razones. En primer lugar, porque se tarda mucho tiempo en abrir las conexiones, en relación al tiempo que se tarda en enviar las solicitudes. El tiempo que se tarda en abrir muchas conexiones para enviar solicitudes de "limitación de carga" en un intervalo corto es muy largo. En segundo lugar, porque para recibir solicitudes de limitación de carga de una compañía de suministro de electricidad, el cliente debe haber abierto la conexión antes. Con MQTT, las conexiones siempre las inicia el cliente y para recibir solicitudes de limitación de carga de la compañía de suministro de electricidad, la conexión debe dejarse abierta.

Si el índice de conexiones abiertas es crítico, o el servidor inicia solicitudes de tiempo crítico, la solución suele ser mantener muchas conexiones abiertas.

Windows Linux AIX Casos de uso de telemetría: identificación por radiofrecuencia (RFID)

RFID es la utilización de un código RFID incorporado para identificar y realizar seguimiento de un objeto de forma inalámbrica. Los códigos RFID pueden leerse hasta en un rango de varios metros y fuera de la línea de visión del lector RFID. Los códigos pasivo se activan mediante un lector RFID. Los códigos activos transmiten sin activación externa. Los códigos activos deben tener una fuente de alimentación eléctrica. Los códigos pasivos pueden utilizar una fuente de alimentación eléctrica para aumentar su rango de alcance.

RFID se utiliza en muchas aplicaciones, y los tipos de casos de uso varían enormemente. Los casos de uso de RFID y los de supervisión del paciente en el hogar y el de supervisión y control de electricidad en el hogar tienen algunas similitudes y diferencias.

Imperceptibilidad

En muchos casos de uso, se despliega el lector RFID en grandes cantidades y debe funcionar sin intervención del usuario. El lector incluye un cliente MQTT incorporado para comunicarse con un punto de control central.

Por ejemplo, en un almacén de distribución, un lector utiliza un sensor de movimiento para detectar los palés. Activa los códigos RFID de los elementos que hay en el palé y envía datos y solicitudes a las aplicaciones centrales. Los datos se utilizan para actualizar la ubicación de la mercancía. Las solicitudes controlan lo que ocurre con un palé cercano, por ejemplo, si se está moviendo hacia una bahía determinada. Los sistemas de equipajes de aerolíneas y aeropuertos utilizan de está forma el RFID.

En algunos casos de uso de RFID, el lector tiene un entorno informático estándar, como Java Platform, Micro Edition (Java ME). En estos casos, el cliente MQTT se puede desplegar en un paso de configuración distinto, después de la fabricación.

Conexión asimétrica

Los lectores RFID pueden estar separados del dispositivo de control local que contiene un cliente MQTT o bien cada lector puede incorporar un cliente MQTT. La elección de cada tipología depende de factores generales, geográficos y relativos a las comunicaciones.

Seguridad

La privacidad y autenticidad son asuntos que van unidos a los códigos RFID. Los códigos RFID son discretos y pueden controlarse, falsificarse o ser forzadas de forma secreta.

La solución a los temas de seguridad de RFID aumenta la posibilidad de despliegue de nuevas soluciones RFID. Aunque la seguridad está expuesta en el código RFID y el lector local, la utilización del proceso de información central puede contrarrestar diferentes amenazas. Por ejemplo, puede detectarse un intento de forzar el código relacionando directamente los niveles de mercancía con las entregas y los envíos.

Conectividad

Las aplicaciones RFID utilizan normalmente el almacenamiento y envío por lotes de información recopilada a partir de lectores RFID y las consultas inmediatas. En el caso de uso del almacén de distribución, el lector RFID está conectado todo el tiempo. Cuando se lee un código, éste queda publicado junto con la información sobre el lector. La aplicación del almacén publica una respuesta y la devuelve al lector.

En la aplicación de almacén, la red suele ser fiable y las solicitudes inmediatas pueden utilizar mensajes de tipo *enviar y olvidar* para conseguir un rendimiento de latencia baja. Los datos que se almacenan y envían por lotes pueden utilizar la mensajería de tipo *exactamente una vez* para minimizar los costes de administración asociados a la pérdida de datos.

Escalabilidad

Si la aplicación de RFID necesita respuestas inmediatas, en uno o dos segundos, los lectores RFID deben permanecer conectados.

Windows Linux AIX Casos de uso de telemetría: detección del entorno

La detección del entorno utiliza la telemetría para recoger información sobre los niveles y la calidad de las aguas de los ríos, la contaminación atmosférica y otros datos ambientales.

Los sensores se ubican normalmente en lugares remotos sin acceso a comunicación por cable. Los anchos de banda inalámbricos son caros y poco seguros. Normalmente, se conectan una serie de sensores en una zona geográfica pequeña a un dispositivo de supervisión local que se encuentra en una ubicación segura. Las conexiones locales pueden ser por cable o inalámbricas.

Imperceptibilidad

Lo normal es que los dispositivos de sensores estén en lugares poco accesibles, tengan poca potencia y estén repartidos en mayores cantidades que los dispositivos de supervisión central. Los sensores son a veces dispositivos "no inteligentes" y el dispositivo de supervisión local incluye adaptadores para transformar y almacenar los datos del sensor. Es probable que el dispositivo de supervisión incorpore un sistema de ámbito general que dé soporte a Java Platform, Standard Edition (Java SE) o Java Platform, Micro Edition (Java ME). Es poco probable que la invisibilidad sea un requisito importante a la hora de configurar el cliente de MQTT.

Conexión asimétrica

Con las posibilidades de los sensores y el coste y ancho de banda de la conexión remota, se obtiene normalmente un concentrador de supervisión local conectado a un servidor central.

Seguridad

A menos que la solución sea para uso militar o defensivo, la seguridad no es un requisito muy importante.

Conectividad

En muchos de los usos no es necesario una supervisión continua o una disponibilidad inmediata de los datos. Los datos de excepción como, por ejemplo, una alerta en el nivel de crecida de un río, sí necesitan enviarse de forma inmediata. Los datos de los sensores se agregan al supervisor local para

reducir los costes de conexión y comunicación, y se transfieren utilizando conexiones planificadas. Los datos de excepción se reenvían al supervisor tan pronto como se detecten.

Escalabilidad

Los sensores se concentran en torno a concentradores locales y los datos de los sensores se agregan en paquetes que se transmiten según se haya planificado. Ambos factores reducen la carga en el servidor central que se impondría si se utilizaran los sensores conectados directamente.

Windows Linux AIX Casos de uso de telemetría: aplicaciones móviles

Las aplicaciones móviles son aplicaciones que se ejecutan en dispositivos inalámbricos. Los dispositivos pueden ser plataformas de aplicaciones genéricas o dispositivos personalizados.

Las plataformas genéricas incluyen dispositivos de mano, como teléfonos y PDA, y dispositivos portátiles como, por ejemplo, los ordenadores portátiles. Los dispositivos personalizados utilizan hardware especial adaptado a aplicaciones específicas. Un ejemplo de dispositivo móvil personalizado es aquél que se utiliza para entregas de paquetes "firmados". Las aplicaciones para dispositivos móviles personalizados a menudo se crean para una plataforma de software genérica.

Imperceptibilidad

El despliegue de aplicaciones móviles personalizadas se gestiona y puede incluir la configuración de la aplicación de cliente MQTT. La imperceptibilidad no suele ser un requisito importante al configurar el cliente MQTT.

Conexión asimétrica

A diferencia de la topología de concentrador local de los casos de uso anteriores, los clientes móviles se conectan de forma remota. La capa de aplicación de cliente se conecta directamente a una aplicación en el concentrador central.

Seguridad

Cuando existe seguridad física reducida, el dispositivo móvil y su usuario deben autenticarse. Se utiliza TLS para confirmar la identidad del dispositivo, y JAAS para el usuario.

Conectividad

Si la aplicación móvil depende de la cobertura inalámbrica, debe poder funcionar sin conexión y debe poder hacer frente de forma eficaz a una conexión que se interrumpa. En este entorno, el objetivo es permanecer conectado, pero la aplicación debe poder almacenar y enviar mensajes. A menudo, los mensajes son pedidos o confirmaciones de entregas, por lo que son importantes desde el punto de vista empresarial. Tienen que almacenarse y reenviarse de forma fiable.

Escalabilidad

La escalabilidad no es un punto importante. La cantidad de clientes de aplicaciones no suelen ser más de varios miles o decenas de miles en los casos de uso de aplicaciones móviles personalizadas.

Windows Linux AIX Conexión de dispositivos de telemetría a un gestor de colas

Los dispositivos de telemetría se conectan a un gestor de colas utilizando un cliente MQTT v3. El cliente MQTT v3 utiliza TCP/IP para conectarse a un escucha TCP/IP denominado servicio de telemetría (MQXR).

Cuando se conecta un dispositivo de telemetría con un gestor de colas, el cliente MMQTT inicia una conexión TCP/IP utilizando el método `MqttClient.connect`. Al igual que los clientes IBM MQ un cliente MQTT debe estar conectado al gestor de colas para enviar y recibir mensajes. La conexión se realiza en el servidor utilizando un escucha TCP/IP, instalado con MQ Telemetry, denominado servicio de telemetría (MQXR). Cada gestor de colas ejecuta, como máximo, un servicio de telemetría (MQXR).

El servicio de telemetría (MQXR) utiliza la dirección de socket remoto que establece cada cliente en el método `MqttClient.connect` para asignar la conexión a un canal de telemetría. Una dirección de socket es la combinación de nombre de host TCP/IP y número de puerto. Si varios clientes utilizan la misma dirección de socket remoto, todos estarán conectados al mismo canal de telemetría, a través del servicio de telemetría (MQXR).

Si hay varios gestores de colas en un servidor, divida los canales de telemetría entre los gestores de colas. Asigne las direcciones de socket remoto entre los gestores de colas. Defina cada canal de telemetría con una dirección de socket remoto exclusiva. Dos canales de telemetría no deben utilizar la misma dirección de socket.

Si se configura la misma dirección de socket remoto para canales de telemetría que estén en varios gestores de colas, el primer canal de telemetría en conectarse, gana. Los canales subsiguientes que se conectan en la misma dirección fallan.

Si hay varios adaptadores de red en el servidor, divida las direcciones de socket remoto entre los canales de telemetría. La asignación de direcciones de socket es totalmente arbitraria, siempre que cualquier dirección de socket específica se haya configurado en sólo un canal de telemetría.

Configure IBM MQ para conectar clientes MQTT utilizando los asistentes proporcionados en el suplemento de MQ Telemetry para IBM MQ Explorer. De forma alternativa, siga las instrucciones de Configuración de un gestor de colas para telemetría en Linux y AIX y Configuración de un gestor de colas para telemetría en Windows para configurar la telemetría manualmente.

Referencia relacionada

Propiedades de MQXR

Windows Linux AIX Protocolos de conexión de telemetría

MQ Telemetry da soporte a TCP/IP IPv4 y IPv6 y TLS.

Windows Linux AIX Servicio de telemetría (MQXR)

El servicio de telemetría (MQXR) es un escucha TCP/IP, que se gestiona como un servicio de IBM MQ.. Cree el servicio utilizando un asistente de IBM MQ Explorer o con un mandato **runmqsc**.

El servicio de MQ Telemetry (MQXR) se denomina SYSTEM.MQXR.SERVICE.

El asistente de configuración de ejemplo de telemetría, que se proporciona en la función de MQ Telemetry para IBM MQ Explorer, crea el servicio de telemetría y un canal de telemetría de ejemplo; consulte Verificación de la instalación de MQ Telemetry utilizando IBM MQ Explorer.

Cree la configuración de ejemplo desde la línea de mandatos; consulte Verificación de la instalación de MQ Telemetryutilizando la línea de mandatos.

El servicio de telemetría (MQXR) se inicia y se detiene automáticamente con el gestor de colas. Controle el servicio utilizando la carpeta de servicios en IBM MQ Explorer. Para ver el servicio, debe pulsar el icono para impedir que IBM MQ Explorer filtre objetos SYSTEM en la pantalla.

Para ver un ejemplo de cómo crear el servicio manualmente, consulte

- Linux AIX Creación de `SYSTEM.MQXR.SERVICE` en Linux.
- Windows Creación de `SYSTEM.MQXR.SERVICE` en Windows.

Estos temas también especifican la clave predeterminada para requerir que se cifren las frases de contraseña para los canales TLS MQTT. Para obtener más información, consulte Cifrado de las frases de contraseña TLS de MQTT.

Windows Linux AIX Canales de telemetría

Cree canales de telemetría para crear conexiones con diferentes propiedades, por ejemplo, JAAS (Java Authentication and Authorization Service) o la autenticación TLS, o para gestionar grupos de clientes.

Cree canales de telemetría utilizando el asistente de **New Telemetry Channel** , proporcionado en la función MQ Telemetry para IBM MQ Explorer. Configure un canal, utilizando el asistente, para aceptar conexiones de clientes MQTT o un puerto TCP/IP específico. Desde IBM WebSphere MQ 7.1, puede configurar MQ Telemetry utilizando el programa de línea de mandatos, **runmqsc**.

Cree varios canales de telemetría en diferentes puertos para gestionar más fácilmente grandes cantidades de conexiones de cliente, dividiendo los clientes en grupos. Cada canal de telemetría tiene un nombre diferente.

Puede configurar los canales de telemetría con atributos de seguridad diferentes para crear tipos de conexiones diferentes. Cree varios canales para aceptar conexiones de cliente en diferentes direcciones TCP/IP. Utilice TLS para cifrar mensajes y autenticar el canal de telemetría y el cliente; consulte Configuración de TLS de clientes y canales de telemetría de MQTT. Especifique el ID de usuario para simplificar el acceso a los objetos de IBM MQ. Especifique una configuración JAAS para autenticar el usuario de MQTT con JAAS; consulte Identificación, autorización y autenticación de clientes MQTT.

Windows Linux AIX Protocolo IBM MQ Telemetry Transport

El protocolo IBM MQ Telemetry Transport (MQTT) v3 se ha diseñado para intercambiar mensajes entre pequeños dispositivos con un ancho de banda reducido o conexiones caras y para enviar mensajes de forma segura. Utiliza TCP/IP.

El MQTT protocol está publicado; consulte Formato y protocolo de IBM MQ Telemetry Transport. La versión 3 del protocolo utiliza la opción de publicar/suscribir y soporta tres calidades de servicio: *enviar y olvidar*, *al menos una vez* y *exactamente una vez*.

Los mensajes son reducidos gracias al pequeño tamaño de las cabeceras del protocolo y a la carga del mensaje en matriz de bytes. Las cabeceras constan de una cabecera fija de 2 bytes y de hasta 12 bytes de otras cabeceras variables adicionales. El protocolo utiliza 12 bytes de cabeceras variables para suscribirse y conectarse y sólo 2 bytes para cabeceras variables para la mayoría de las publicaciones.

Con tres calidades de servicio, se puede compensar la baja latencia con la fiabilidad; consulte Calidades de servicio proporcionadas por un cliente del MQTT. *Enviar y olvidar* no utiliza ningún almacenamiento de dispositivo permanente y únicamente una transmisión para enviar o recibir una publicación. *Al menos una vez* y *exactamente una vez* requieren el almacenamiento permanente en el dispositivo para mantener el estado del protocolo y guardar un mensaje hasta que se acuse recibo de él.

Windows Linux AIX Clientes MQTT

Una aplicación de cliente MQTT se encarga de recopilar información del dispositivo de telemetría, conectar con el servidor y publicar la información en el servidor. También puede suscribirse a temas, recibir publicaciones y controlar el dispositivo de telemetría.

A diferencia de las aplicaciones cliente de IBM MQ, las aplicaciones cliente MQTT no son aplicaciones de IBM MQ. No especifican ningún gestor de colas al que conectarse. No se limitan a utilizar interfaces de programación de IBM MQ específicas. En su lugar, los clientes MQTT implementan el protocolo MQTT 3. Puede escribir su propia biblioteca de clientes para intercambiar información con el MQTT protocol en el lenguaje de programación y en la plataforma que desee. Consulte Formato y protocolo de IBM MQ Telemetry Transport.

Para simplificar la escritura de aplicaciones cliente de MQTT, utilice las bibliotecas cliente C, Java y JavaScript que integran el MQTT protocol para diversas plataformas. Si incorpora estas bibliotecas en las apps de MQTT , un cliente MQTT totalmente funcional puede ser tan corto como 15 líneas de código. MQTTLas bibliotecas cliente de están disponibles gratuitamente en Eclipse Paho y MQTT.org. Consulte Programas de ejemplo de IBM MQ Telemetry Transport.

La aplicación de cliente MQTT se encarga siempre de iniciar una conexión con un canal de telemetría. Una vez conectado, la aplicación cliente de MQTT o una aplicación de IBM MQ puede iniciar un intercambio de mensajes.

Las aplicaciones cliente de MQTT y las aplicaciones de IBM MQ se publican y se suscriben al mismo conjunto de temas. Una aplicación de IBM MQ también puede enviar un mensaje directamente a una

aplicación cliente de MQTT sin que la aplicación cliente cree primero una suscripción. Consulte Configurar gestión de colas distribuidas para enviar mensajes a clientes MQTT.

Las aplicaciones cliente de MQTT están conectadas a IBM MQ utilizando un canal de telemetría. El canal de telemetría actúa como un puente entre los distintos tipos de mensajes utilizados por MQTT y IBM MQ. Crea publicaciones y suscripciones en el gestor de colas, en nombre de la aplicación de cliente MQTT. El canal de telemetría envía publicaciones que coinciden con las suscripciones de una aplicación de cliente MQTT del gestor de colas a la aplicación de cliente MQTT.

Windows Linux AIX

Envío de un mensaje a un cliente MQTT

Las aplicaciones de IBM MQ pueden enviar mensajes de clientes MQTT v3 publicando en las suscripciones creadas por los clientes o enviando mensajes directamente. Los clientes MQTT pueden enviarse mensajes entre sí publicando en los temas suscritos por otros clientes.

Un cliente MQTT se suscribe a una publicación, que recibe de IBM MQ

Realice la tarea, “Publicación de un mensaje en el programa de utilidad cliente MQTT de IBM MQ Explorer” en la página 124 para enviar una publicación de IBM MQ a un cliente MQTT.

El modo estándar para que un cliente MQTT v3 reciba mensajes es que cree una suscripción a un tema o conjunto de temas. En el fragmento de código de ejemplo, Figura 44 en la página 123, el cliente de MQTT se suscribe utilizando la serie de tema `"MQTT Examples"`. Una aplicación IBM MQ C, Figura 45 en la página 123, publica en el tema utilizando la serie de tema `"MQTT Examples"`. En el fragmento de código de la Figura 46 en la página 123, el cliente MQTT recibe la publicación en el método de devolución de llamada, `messageArrived`.

Para obtener más información sobre cómo configurar IBM MQ para enviar publicaciones en respuesta a suscripciones de clientes MQTT, consulte Publicación de un mensaje en respuesta a una suscripción de cliente MQTT.

Una aplicación IBM MQ envía un mensaje directamente a un cliente MQTT

Realice la tarea, “Envío de un mensaje a un cliente MQTT utilizando IBM MQ Explorer” en la página 128 para enviar un mensaje directamente desde IBM MQ a un cliente MQTT.

Un mensaje enviado de esta manera a un cliente MQTT se denomina un mensaje no solicitado. Los clientes MQTT v3 reciben mensajes no solicitados como publicaciones, con un conjunto de nombres de temas. El servicio de telemetría (MQXR) establece el nombre del tema en el nombre de la cola remota.

Para obtener más información sobre cómo configurar IBM MQ para enviar mensajes directamente a clientes MQTT, consulte Envío de un mensaje a un cliente directamente.

Un cliente MQTT publica un mensaje

Un cliente MQTT v3 puede publicar un mensaje que recibe otro cliente MQTT v3, pero no puede enviar un mensaje no solicitado. El fragmento de código de la Figura 47 en la página 124 muestra cómo un cliente MQTT v3, escrito en Java, publica un mensaje.

El patrón típico para enviar un mensaje a un cliente MQTT v3 específico, es que cada cliente cree una suscripción a su propio `ClientIdentifier`. Efectúe la tarea “Publicación de un mensaje en un cliente MQTT v3 específico” en la página 130 para publicar un mensaje de un cliente MQTT a otro cliente MQTT utilizando `ClientIdentifier` como serie de tema.

Fragmentos de código de ejemplo

En el fragmento de código de la Figura 44 en la página 123 se muestra cómo un cliente MQTT escrito en Java crea una suscripción. También necesita un método de devolución de llamada, `messageArrived` para poder recibir publicaciones para la suscripción.

```
String    clientId = String.format("%-23.23s",
                     System.getProperty("user.name") + "_" +
                     (UUID.randomUUID().toString())).trim()).replace('-', '_');
MqttClient  client = new MqttClient("localhost", clientId);
String topicString = "MQTT Examples";
int            QoS = 1;
client.subscribe(topicString, QoS);
```

Figura 44. Suscriptor de cliente MQTT v3

El fragmento de código de Figura 45 en la página 123 muestra cómo una aplicación IBM MQ escrita en C envía una publicación. El fragmento de código se extrae de la tarea, Crear un publicador en un tema variable.

```
/* Define and set variables to.defaults */
/* Omitted lines declaring variables    */
char * topicName   = ""
char * topicString = "MQTT Examples"
char * publication = "Hello world!";
do {
  MQCONN(qMgrName, &Hconn, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  td.ObjectType = MQOT_TOPIC;    /* Object is a topic             */
  td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4     */
  strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
  td.ObjectString.VSPtr = topicString;
  td.ObjectString.VSLength = (MQLONG)strlen(topicString);
  MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
  MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
```

Figura 45. Publicador de IBM MQ

Cuando llega la publicación, el cliente MQTT llama al método `messageArrived` de la clase `MqttCallback` del cliente de aplicaciones MQTT .

```
public class CallBack implements MqttCallback {
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
// ... Other callback methods
}
```

Figura 46. Método de `messageArrived`

En la Figura 47 en la página 124 se muestra MQTT v3 publicando un mensaje en la suscripción creada en la Figura 44 en la página 123.

```
String          address = "localhost";
String         clientId = String.format("%-23.23s",
                          System.getProperty("user.name") + "_" +
                          (UUID.randomUUID().toString())).trim()).replace('-', '_');
MqttClient       client = new MqttClient(address, clientId);
String      topicString = "MQTT Examples";
MqttTopic         topic = client.getTopic(Example.topicString);
String     publication  = "Hello world";
MqttMessage     message = new MqttMessage(publication.getBytes());
MqttDeliveryToken token = topic.publish(message);
```

Figura 47. Publicador de cliente MQTT v3

Windows Linux AIX Publicación de un mensaje en el programa de utilidad cliente MQTT de IBM MQ Explorer

Siga los pasos de esta tarea para publicar un mensaje utilizando IBM MQ Explorer y suscríbase a él con el programa de utilidad de cliente de MQTT. En una tarea adicional se le muestra cómo configurar un alias de gestor de colas, en lugar de establecer la cola de transmisión predeterminada en `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Antes de empezar

La tarea presupone que está familiarizado con IBM MQ y IBM MQ Explorer y que se han instalado IBM MQ y la característica de MQ Telemetry.

El usuario que cree los recursos del gestor de colas para esta tarea deberá tener permisos suficientes para poder hacerlo. A efectos de demostración, se supone que el ID de usuario de IBM MQ Explorer es miembro del grupo `mqm`.

Acerca de esta tarea

En la tarea, cree un tema en IBM MQ y suscríbase a él utilizando el programa de utilidad de cliente de MQTT. Cuando se publica en el tema utilizando IBM MQ Explorer, el cliente MQTT recibe la publicación.

Procedimiento

Efectúe una de las tareas siguientes:

- Ha instalado MQ Telemetry, pero todavía no lo ha iniciado. Efectúe la tarea: "Inicio de la tarea sin haber definido todavía el servicio de telemetría (MQXR)" en la página 125.
- Ha ejecutado la telemetríca de IBM MQ telemetría antes, pero desea utilizar un nuevo gestor de colas para realizar la demostración. Efectúe la tarea: "Inicio de la tarea sin haber definido todavía el servicio de telemetría (MQXR)" en la página 125.
- Desea efectuar la tarea utilizando un gestor de colas existente que no tiene definidos los recursos de telemetría. No desea ejecutar el asistente **Definir configuración de ejemplo**.
  a. Efectúe una de las tareas siguientes para configurar la telemetría:
     - Configuración de un gestor de colas para la telemetría en Linux y AIX
     - Configuración de un gestor de colas para la telemetría en Windows
  b. Efectúe la tarea: "Inicio de la tarea después ejecutar el servicio de telemetría (MQXR)" en la página 126
- Si desea efectuar la tarea utilizando un gestor de colas existente que ya tenga definidos los recursos, realice la tarea: "Inicio de la tarea después ejecutar el servicio de telemetría (MQXR)" en la página 126.

Qué hacer a continuación

Efectúe los pasos indicados en "Envío de un mensaje a un cliente MQTT utilizando IBM MQ Explorer" en la página 128 para enviar un mensaje directamente al programa de utilidad cliente.

Inicio de la tarea sin haber definido todavía el servicio de telemetría (MQXR)

Cree un gestor de colas y ejecute el asistente **Definir configuración de ejemplo** para definir recursos de telemetría de ejemplo para el gestor de colas. Publique un mensaje utilizando IBM MQ Explorer y suscríbase a él con el programa de utilidad cliente de MQTT.

Acerca de esta tarea

Al configurar los recursos de ejemplo mediante el asistente **Definir configuración de ejemplo**, se establecen los permisos del ID de usuario invitado. Considere detenidamente si desea autorizar al ID de usuario invitado de esta forma. `guest` en Windows, y `nobody` en Linux, tienen permiso para publicar y suscribirse a la raíz del árbol de temas, y para colocar mensajes en `SYSTEM.MQTT.TRANSMIT.QUEUE`.

El asistente también establece la cola de transmisión predeterminada en `SYSTEM.MQTT.TRANSMIT.QUEUE`, que puede interferir con las aplicaciones que se ejecuten en un gestor de colas existente. Es posible, pero laborioso, configurar la telemetría y no utilizar la cola de transmisión predeterminada; efectúe lo siguiente en la tarea: "Utilización de un alias de gestor de colas" en la página 127. En esta tarea, creará un gestor de colas para evitar la posibilidad de interferir en las colas de transmisión predeterminadas existentes.

Procedimiento

1. Utilizando IBM MQ Explorer, cree e inicie un nuevo gestor de colas.
   a) Pulse el botón derecho del ratón en la carpeta `Queue Managers` > **Nuevo** > **Gestor de colas...**. Escriba un nombre de gestor de colas > **Finalizar**.

      Piense en un nombre de gestor de colas; por ejemplo, `MQTTQMGR`.
2. Cree e inicie el servicio de telemetría (MQXR), y cree un canal de telemetría de ejemplo.
   a) Abra la carpeta `Queue Managers\`*`QmgrName`*`\Telemetry`.
   b) Pulse **Definir configuración de ejemplo...** > **Finalizar**

      Marque el recuadro de selección **Iniciar programa de utilidad cliente MQTT**.
3. Cree una suscripción para `MQTT Example` utilizando el programa de utilidad de cliente de MQTT.
   a) Pulse **Conectar**.

      El **Historial del cliente** registra un suceso de `Connected` .
   b) Escriba `MQTT Example` en el campo **Suscripción \ Tema** > **Suscribir**.

   El **Historial del cliente** registra un suceso de `Subscribed` .
4. Cree `MQTTExampleTopic` en IBM MQ.
   a) Pulse con el botón derecho del ratón en la carpeta `Queue Managers\`*`QmgrName`*`\Topics` de **MQ Explorer**> **Nuevo** > **Tema**.
   b) Escriba `MQTTExampleTopic` como **Nombre** > **Siguiente**.
   c) Escriba `MQTT Example` como **Serie de tema** > **Finalizar**.
   d) Pulse **Aceptar** para cerrar la ventana de acuse de recibo.
5. Publique `Hello World!` en el tema `MQTT Example` utilizando IBM MQ Explorer.
   a) Pulse la carpeta `Queue Managers\`*`QmgrName`*`\Topics` en IBM MQ Explorer.
   b) Pulse el botón derecho del ratón `MQTTExampleTopic` > **Probar publicación...**
   c) Escriba `Hello World!` en el campo **Datos de mensaje** > **Publicar mensaje** > Vaya la ventana de programa de utilidad cliente de MQTT.

   El **Historial del cliente** registra un suceso de `Received` .

Inicio de la tarea después ejecutar el servicio de telemetría (MQXR)

Cree un canal de telemetría y un tema. Autorice al usuario a utilizar el tema y la cola de transmisión de telemetría. Publique un mensaje utilizando IBM MQ Explorer y suscríbase a él con el programa de utilidad cliente de MQTT.

Antes de empezar

En esta versión de la tarea, un gestor de colas, *QmgrName*, está definido y en ejecución. Hay un servicio de telemetría (MQXR) definido y se está ejecutando. Es posible que el servicio de telemetría (MQXR) se haya creado manualmente, o al ejecutar el asistente **Definir configuración de ejemplo**.

Acerca de esta tarea

En esta tarea debe configurar un gestor de colas existente para enviar una publicación al programa de utilidad cliente MQTT.

En el paso "1" en la página 126 de la tarea se establece la cola de transmisión predeterminada en SYSTEM.MQTT.TRANSMIT.QUEUE, que puede interferir con las aplicaciones que se ejecuten en un gestor de colas existente. Es posible, pero laborioso, configurar la telemetría y no utilizar la cola de transmisión predeterminada; efectúe lo siguiente en la tarea: "Utilización de un alias de gestor de colas" en la página 127.

Procedimiento

1. Establezca SYSTEM.MQTT.TRANSMIT.QUEUE como la cola de transmisión predeterminada.
   a) Pulse con el botón derecho del ratón en Queue Managers*QmgrName* folder > **Propiedades...**
   b) Pulse **Comunicación** en el navegador.
   c) Pulse **Seleccionar ...** > Seleccionar SYSTEM.MQTT.TRANSMIT.QUEUE > **Aceptar** > **Aceptar**.
2. Cree un canal de telemetría MQTTExampleChannel para conectar el programa de utilidad cliente de MQTT a IBM MQ e inicie el programa de utilidad cliente de MQTT.
   a) Pulse con el botón derecho del ratón en la carpeta Queue Managers*QmgrName* \Telemetry\Channels de **MQ Explorer**> **Nuevo** > **Canal de telemetría...**.
   b) Escriba MQTTExampleChannel en el campo **Nombre de canal** > **Siguiente** > **Siguiente**.
   c) Cambie el **ID de usuario fijo** en el panel de autorización del cliente al ID de usuario que va a publicar y suscribirse a MQTTExample > **Siguiente**.
   d) Deje seleccionado **Iniciar programa de utilidad cliente** > **Finalizar**.
3. Cree una suscripción para MQTT Example utilizando el programa de utilidad de cliente de MQTT.
   a) Pulse **Conectar**.

      El **Historial del cliente** registra un suceso de Connected .
   b) Escriba MQTT Example en el campo **Suscripción \ Tema** > **Suscribir**.

   El **Historial del cliente** registra un suceso de Subscribed .
4. Cree MQTTExampleTopic en IBM MQ.
   a) Pulse con el botón derecho del ratón en la carpeta Queue Managers*QmgrName*\Topics de **MQ Explorer**> **Nuevo** > **Tema**.
   b) Escriba MQTTExampleTopic como **Nombre** > **Siguiente**.
   c) Escriba MQTT Example como **Serie de tema** > **Finalizar**.
   d) Pulse **Aceptar** para cerrar la ventana de acuse de recibo.
5. Si desea que un usuario, que no está en el grupo mqm, publique y se suscriba al tema MQTTExample, haga lo siguiente:
   a) Autorice al usuario a publicar y suscribirse al tema MQTTExampleTopic:

```
setmqaut -m qMgrName -t topic -n MQTTExampleTopic -p User ID -all +pub +sub
```

b) Autorice al usuario a transferir un mensaje a la cola SYSTEM.MQTT.TRANSMIT.QUEUE:

```
setmqaut -m qMgrName -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p User ID -all +put
```

6. Publique Hello World! en el tema MQTT Example utilizando IBM MQ Explorer.
   a) Pulse la carpeta Queue Managers*QmgrName*\Topics en IBM MQ Explorer.
   b) Pulse el botón derecho del ratón MQTTExampleTopic > **Probar publicación...**
   c) Escriba Hello World! en el campo **Datos de mensaje** > **Publicar mensaje** > Vaya la ventana de programa de utilidad cliente de MQTT.

El **Historial del cliente** registra un suceso de Received .

Utilización de un alias de gestor de colas

Publique un mensaje en el programa de utilidad cliente MQTT utilizando IBM MQ Explorer sin establecer la cola de transmisión predeterminada en SYSTEM.MQTT.TRANSMIT.QUEUE.

La tarea es una continuación de la tarea anterior, y en ella se utiliza un alias de gestor de colas para evitar establecer la cola de transmisión predeterminada en SYSTEM.MQTT.TRANSMIT.QUEUE.

Antes de empezar

Complete la tarea "Inicio de la tarea sin haber definido todavía el servicio de telemetría (MQXR)" en la página 125 o la tarea "Inicio de la tarea después ejecutar el servicio de telemetría (MQXR)" en la página 126.

Acerca de esta tarea

Cuando un cliente de MQTT crea una suscripción, IBM MQ envía su respuesta utilizando ClientIdentifier, como nombre del gestor de colas remoto. En esta tarea, utiliza ClientIdentifier, MyClient.

Si no hay ninguna cola de transmisión o alias de gestor de colas llamado MyClient, la respuesta se coloca en la cola de transmisión predeterminada. Al establecer la cola de transmisión predeterminada en SYSTEM.MQTT.TRANSMIT.QUEUE, el cliente MQTT obtiene la respuesta.

Puede evitar establecer la cola de transmisión predeterminada en SYSTEM.MQTT.TRANSMIT.QUEUE utilizando los alias de gestor de colas. Debe configurar un alias de gestor de colas para cada ClientIdentifier. Normalmente, hay demasiados clientes para que resulte práctico utilizar los alias de gestor de colas. A menudo ClientIdentifier es imprevisible, haciendo que sea imposible configurar la telemetría esta forma.

Sin embargo, en algunas circunstancias puede que tenga que configurar la cola de transmisión predeterminada en un valor distinto de SYSTEM.MQTT.TRANSMIT.QUEUE. Los pasos de Procedimiento configuran un alias de gestor de colas en lugar de establecer la cola de transmisión predeterminada en SYSTEM.MQTT.TRANSMIT.QUEUE.

Procedimiento

1. Elimine SYSTEM.MQTT.TRANSMIT.QUEUE como la cola de transmisión predeterminada.
   a) Pulse con el botón derecho del ratón en Queue Managers*QmgrName* folder > **Propiedades...**
   b) Pulse **Comunicación** en el navegador.
   c) Quite SYSTEM.MQTT.TRANSMIT.QUEUE del campo **Cola de transmisión predeterminado** > **Aceptar**.
2. Compruebe que ya no pueda crear una suscripción con el programa de utilidad cliente MQTT.

a) Pulse **Conectar**.

El **Historial del cliente** registra un suceso de `Connected` .

b) Escriba `MQTT Example` en el campo **Suscripción \ Tema** > **Suscribir**.

El **Historial de cliente** registra un suceso `Subscribe failed` y un suceso `Connection lost` .

3. Cree un alias de gestor de colas para el `ClientIdentifier`, `MyClient`.
   a) Pulse con el botón derecho del ratón en la carpeta `Queue Managers\`*`QmgrName`*`\Queues` > **Nuevo** > **Definición de cola remota**.
   b) Llame a la definición, `MyClient` > **Siguiente**.
   c) Escriba `MyClient` en el campo **Gestor de colas remoto**.
   d) Escriba `SYSTEM.MQTT.TRANSMIT.QUEUE` en el campo **Cola de transmisión** > **Finalizar**.
4. Vuelva a conectar el programa de utilidad cliente MQTT.
   a) Compruebe que el **ClientIdentifier** se haya establecido en `MyClient`.
   b) **Conectar**

El **Historial del cliente** registra un suceso de `Connected` .

5. Cree una suscripción para `MQTT Example` utilizando el programa de utilidad de cliente de MQTT.
   a) Pulse **Conectar**.

   El **Historial del cliente** registra un suceso de `Connected` .

   b) Escriba `MQTT Example` en el campo **Suscripción \ Tema** > **Suscribir**.

El **Historial del cliente** registra un suceso de `Subscribed` .

6. Publique `Hello World!` en el tema `MQTT Example` utilizando IBM MQ Explorer.
   a) Pulse la carpeta `Queue Managers\`*`QmgrName`*`\Topics` en IBM MQ Explorer.
   b) Pulse el botón derecho del ratón `MQTTExampleTopic` > **Probar publicación...**
   c) Escriba `Hello World!` en el campo **Datos de mensaje** > **Publicar mensaje** > Vaya la ventana de programa de utilidad cliente de MQTT.

El **Historial del cliente** registra un suceso de `Received` .

Windows Linux AIX Envío de un mensaje a un cliente MQTT utilizando IBM MQ Explorer

Envíe un mensaje al programa de utilidad cliente de MQTT transfiriendo un mensaje a una cola de IBM MQ utilizando IBM MQ Explorer. En la tarea se muestra cómo configurar una definición de cola remota para enviar un mensaje directamente a un cliente MQTT.

Antes de empezar

Lleve a cabo la tarea “Publicación de un mensaje en el programa de utilidad cliente MQTT de IBM MQ Explorer” en la página 124. Deje el programa de utilidad cliente MQTT conectado.

Acerca de esta tarea

En la tarea se muestra cómo enviar un mensaje a un cliente MQTT utilizando la cola en lugar de publicar en un tema. No se crea ninguna suscripción en el cliente. En el paso “2” en la página 129 de la tarea se muestra que se ha suprimido la suscripción anterior.

Procedimiento

1. Elimine las suscripciones existentes desconectando y volviendo a conectar el programa de utilidad cliente MQTT.

La suscripción se elimina porque, a menos que se cambien los valores predeterminados, el programa de utilidad cliente MQTT se conecta con una sesión limpia; consulte Limpiar sesiones.

Para que le resulte más fácil realizar la tarea, escriba su propio `ClientIdentifier`, en lugar de utilizar el `ClientIdentifier` generado creado por el programa de utilidad cliente MQTT.

a) Pulse **Desconectar** para desconectar el programa de utilidad cliente MQTT del canal de telemetría.

El **Historial del cliente** registra un suceso de `Disconnected`

b) Cambie el valor de **Identificador de cliente** por `MyClient`.

c) Pulse **Conectar**.

El **Historial del cliente** registra un suceso de `Connected`

2. Compruebe que el programa de utilidad cliente MQTT ya no recibe la publicación para `MQTTExampleTopic`.

a) Pulse la carpeta `Queue Managers\`*`QmgrName`*`\Topics` en IBM MQ Explorer.

b) Pulse el botón derecho del ratón `MQTTExampleTopic` > **Probar publicación...**

c) Escriba `Hello World!` en el campo **Datos de mensaje** > **Publicar mensaje** > Vaya la ventana de programa de utilidad cliente de MQTT.

No se registra ningún suceso en el **Historial del cliente**.

3. Cree una definición de cola remota para el cliente.

Establezca `ClientIdentifier`, `MyClient`, como el nombre del gestor de colas remoto, en la definición de la cola remota. Utilice el nombre que desee como nombre de la cola remota. El nombre de la cola remota se pasa a un cliente MQTT como nombre de tema.

a) Pulse con el botón derecho del ratón en la carpeta `Queue Managers\`*`QmgrName`*`\Queues` > **Nuevo** > **Definición de cola remota**.

b) Llame a la definición, `MyClientRemoteQueue` > **Siguiente**.

c) Escriba `MQTTExampleQueue` en el campo **Cola remota**.

d) Escriba `MyClient` en el campo **Gestor de colas remoto**.

e) Escriba `SYSTEM.MQTT.TRANSMIT.QUEUE` en el campo **Cola de transmisión** > **Finalizar**.

4. Transfiera un mensaje de prueba a `MyClientRemoteQueue`.

a) Pulse el botón derecho del ratón en **MyClientRemoteQueue** > **Transferir mensaje de prueba...**

b) Escriba `Hello queue!` en el campo Datos de mensaje > **Transferir mensaje** > **Cerrar**

El **Historial del cliente** registra un suceso de `Received` .

5. Elimine `SYSTEM.MQTT.TRANSMIT.QUEUE` como la cola de transmisión predeterminada.

a) Pulse con el botón derecho del ratón en `Queue Managers\`*`QmgrName`* `folder` > **Propiedades...**

b) Pulse **Comunicación** en el navegador.

c) Quite `SYSTEM.MQTT.TRANSMIT.QUEUE` del campo **Cola de transmisión predeterminado** > **Aceptar**.

6. Vuelva a efectuar el paso "4" en la página 129.

`MyClientRemoteQueue` es una definición de cola remota que nombra explícitamente la cola de transmisión. No necesita una para definir la cola de transmisión predeterminada para enviar un mensaje a `MyClient`.

Qué hacer a continuación

Con la cola de transmisión predeterminada ya no establecida en `SYSTEM.MQTT.TRANSMIT.QUEUE`, el programa de utilidad cliente de MQTT no puede crear una nueva suscripción a menos que se haya definido un alias de gestor de colas para el `ClientIdentifier`, `MyClient`. Restaure la cola de transmisión predeterminada en `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Windows Linux AIX Publicación de un mensaje en un cliente MQTT v3 específico

Publique un mensaje de un cliente MQTT v3 a otro, utilizando `ClientIdentifier` como nombre de tema y IBM MQ como intermediario de publicación/suscripción.

Antes de empezar

Lleve a cabo la tarea "Publicación de un mensaje en el programa de utilidad cliente MQTT de IBM MQ Explorer" en la página 124. Deje el programa de utilidad cliente MQTT conectado.

Acerca de esta tarea

La tarea demuestra dos cosas:

1. La suscripción a un tema de un cliente MQTT y la recepción de una publicación de otro cliente MQTT.
2. La configuración de suscripciones "punto a punto" utilizando el `ClientIdentifier` como la serie de tema.

Procedimiento

1. Elimine las suscripciones existentes desconectando y volviendo a conectar el programa de utilidad cliente MQTT.

   La suscripción se elimina porque, a menos que se cambien los valores predeterminados, el programa de utilidad cliente MQTT se conecta con una sesión limpia; consulte Limpiar sesiones.

   Para que le resulte más fácil realizar la tarea, escriba su propio `ClientIdentifier`, en lugar de utilizar el `ClientIdentifier` generado creado por el programa de utilidad cliente MQTT.

   a) Pulse **Desconectar** para desconectar el programa de utilidad cliente MQTT del canal de telemetría.

   El **Historial del cliente** registra un suceso de `Disconnected`

   b) Cambie el valor de **Identificador de cliente** por `MyClient`.

   c) Pulse **Conectar**.

   El **Historial del cliente** registra un suceso de `Connected`
2. Cree una suscripción al tema `MyClient`

   `MyClient` es el `ClientIdentifier` de este cliente.

   a) Escriba `MyClient` en el campo **Suscripción\tema** > **Suscribir**.

   El **Historial del cliente** registra un suceso de `Subscribed` .
3. Inicie otro programa de utilidad MQTT.

   a) Abra la carpeta `Queue Managers\`*`QmgrName`*`\Telemetry\channels`.

   b) Pulse el botón derecho del ratón en el canal **PlainText** > **Ejecutar programa de utilidad cliente de MQTT...**

   c) Pulse **Conectar**.

   El **Historial del cliente** registra un suceso de `Connected`
4. Publique `Hello MyClient!` en el tema `MyClient`.

   a) Copie el tema de suscripción, `MyClient`, del programa de utilidad cliente de MQTT que se ejecuta con `ClientIdentifier`, `MyClient`.

   b) Pegue `MyClient` en el campo **Publicación\Tema** de cada una de las instancias del programa de utilidad cliente de MQTT.

   c) Escriba `Hello MyClient!` en el campo **Publicación\ mensaje**.

   d) Pulse **Publicar** en ambas instancias.

Resultados

El **Historial del cliente** en el programa de utilidad cliente de MQTT con `ClientIdentifier`, `MyClient`, registra dos sucesos **Recibidos** y un suceso **Publicado**. La otra instancia del programa de utilidad cliente MQTT registra un suceso **Publicado**.

Si solo ve un suceso del tipo **Recibido**, compruebe las posibles causas siguientes:

1. ¿Se ha establecido la cola de transmisión predeterminada del gestor de colas en `SYSTEM.MQTT.TRANSMIT.QUEUE`?
2. ¿Ha creado alias de gestor de colas o definiciones de colas remotas que hagan referencia a `MyClient` al realizar los otros ejercicios? En caso de que tenga un problema de configuración, suprima los recursos que hagan referencia a `MyClient`, como, por ejemplo, un alias de gestor de colas o colas de transmisión. Desconecte los programas de utilidad de cliente, detenga y reinicie el servicio de telemetría (MQXR).

Windows Linux AIX Envío de un mensaje a una aplicación de IBM MQ desde un cliente MQTT

Una aplicación de IBM MQ puede recibir un mensaje de un cliente MQTT v3 suscribiéndose a un tema. El cliente MQTT se conecta a IBM MQ utilizando un canal de telemetría y envía un mensaje a la aplicación de IBM MQ publicando en el mismo tema.

Realice la tarea, "Publicación de un mensaje en IBM MQ desde un cliente MQTT" en la página 131, para aprender a enviar una publicación de un cliente MQTT a una suscripción definida en IBM MQ.

Si el tema está en un clúster o se ha distribuido mediante la jerarquía de publicación/suscripción, la suscripción puede estar en un gestor de colas diferente del gestor de colas al que el cliente MQTT está conectado.

Windows Linux AIX Publicación de un mensaje en IBM MQ desde un cliente MQTT

Cree una suscripción a un tema utilizando IBM MQ Explorer y publique en el tema utilizando el programa de utilidad cliente de MQTT..

Antes de empezar

Lleve a cabo la tarea "Publicación de un mensaje en el programa de utilidad cliente MQTT de IBM MQ Explorer" en la página 124. Deje el programa de utilidad cliente MQTT conectado.

Acerca de esta tarea

En la tarea se muestra la publicación de un mensaje con un cliente MQTT y la recepción de la publicación utilizando una suscripción duradera no gestionada que se ha creado utilizando IBM MQ Explorer.

Procedimiento

1. Cree una suscripción duradera para la serie de tema `MQTT Example`.

   Realice los pasos siguientes para crear la cola y la suscripción utilizando IBM MQ Explorer.

   a) Pulse con el botón derecho del ratón en la carpeta `Queue Managers\`*`QmgrName`*`\Queues` en IBM MQ Explorer> **Nuevo** > **Cola local...**.

   b) Escriba `MQTTExampleQueue` como nombre de cola > **Finalizar**.

   c) Pulse con el botón derecho del ratón en la carpeta `Queue Managers\`*`QmgrName`*`\Subscriptions` en IBM MQ Explorer> **Nuevo** > **Suscripción...**.

   d) Escriba `MQTTExampleSubscription` como nombre de cola > **Siguiente**.

   e) Pulse **Seleccionar...** > `MQTTExampleTopic` > **Aceptar**.

Ya ha creado el tema MQTTExampleTopic en el paso “4” en la página 125 de “Publicación de un mensaje en el programa de utilidad cliente MQTT de IBM MQ Explorer” en la página 124.

f) Escribo MQTTExampleQueue como nombre de destino > **Finalizar**.

2. Como paso opcional, establezca la cola para que la utilice un usuario distinto, sin autorización de mqm.

   Si está realizando la configuración para usuarios con menos autorización que mqm, debe otorgar autorización put y get a MQTTExampleQueue. El acceso al tema y a la cola de transmisión se ha configurado en “Publicación de un mensaje en el programa de utilidad cliente MQTT de IBM MQ Explorer” en la página 124.

   a) Autorice a un usuario a colocar (put) y acceder a la cola (get) MQTTExampleQueue:

   ```
   setmqaut -m qMgrName -t queue -n MQTTExampleQueue -p User ID -all +put +get
   ```

3. Publique Hello IBM MQ! en el tema MQTT Example utilizando el programa de utilidad cliente MQTT .

   Si no ha dejado el programa de utilidad cliente MQTT conectado, pulse el botón derecho (del ratón) en el canal **PlainText** > **Ejecutar el programa de utilidad cliente MQTT...** > **Conectar**.

   a) Escriba MQTT Example en el campo **Publicación\Tema**.

   b) Escriba Hello IBM MQ! en el campo **Publicación\Mensaje** > **Publicar**.

4. Abra la carpeta Queue Managers*QmgrName*\Queues y busque MQTTExampleQueue.

   El valor del campo **Profundidad de cola actual** es 1.

5. Pulse con el botón derecho MQTTExampleQueue > **Examinar mensajes ...** y examine la publicación.

Windows Linux AIX Aplicaciones de publicación/suscripción MQTT

Utilice la publicación/suscripción basada en temas para escribir aplicaciones MQTT.

Cuando el cliente MQTT está conectado, las publicaciones fluyen en ambas direcciones entre el cliente y el servidor. Cuando la información se publica en el cliente, entonces se envían las publicaciones desde éste. El cliente recibe las publicaciones cuando se publica un mensaje en un tema que coincide con una suscripción que haya creado el cliente.

El intermediario de publicación/suscripción de IBM MQ gestiona los temas y suscripciones creados por los clientes MQTT. Los temas creados por los clientes MQTT comparten el mismo espacio de temas que los temas creados por las aplicaciones de IBM MQ.

Las publicaciones que coinciden con la serie de tema en una suscripción de cliente MQTT se colocan en SYSTEM.MQTT.TRANSMIT.QUEUE con el nombre del gestor de colas remoto establecido en el ClientIdentifier del cliente. El servicio de telemetría (MQXR) reenvía las publicaciones al cliente que ha creado la suscripción. Utiliza ClientIdentifier, que se ha establecido como el nombre del gestor de colas remoto para identificar al cliente.

Normalmente, se tiene que haber definido SYSTEM.MQTT.TRANSMIT.QUEUE como la cola de transmisión predeterminada. Es posible, aunque laborioso, para configurar MQTT no utilizar la cola de transmisión predeterminada; consulte Configurar gestión de colas distribuidas para enviar mensajes a clientes MQTT.

Un cliente MQTT puede crear una sesión persistente; consulte “Sesiones de MQTT con estado y sin estado” en la página 136. Las suscripciones que se crean en una sesión persistente son duraderas. Las publicaciones que llegan para un cliente que tiene una sesión persistente se almacenan en SYSTEM.MQTT.TRANSMIT.QUEUE y se reenvían al cliente cuando éste se vuelve a conectar.

Un cliente MQTT también puede publicar y suscribirse a las publicaciones retenidas; consulte Publicaciones retenidas y clientes MQTT. Un suscriptor de un tema de publicación retenida recibe la publicación más reciente al tema. El suscriptor recibe la publicación retenida cuando crea una suscripción, o cuando se vuelve a conectar a su sesión anterior.

Windows Linux AIX Aplicaciones de telemetría

Escriba aplicaciones de telemetría utilizando flujos de mensajes de IBM MQ o IBM Integration Bus.

Utilice JMS MQI u otras interfaces de programación de IBM MQ para programar aplicaciones de telemetría en IBM MQ.

El servicio de telemetría (MQXR) convierte entre mensajes MQTT v3 y mensajes IBM MQ. Crea suscripciones y publicaciones en nombre de los clientes MQTT y reenvía publicaciones a los clientes MQTT. Una publicación es la carga útil de un mensaje MQTT v3. La carga útil consta de las cabeceras del mensaje y una matriz de bytes en formato `jms-bytes`. El servidor de telemetría correlaciona las cabeceras entre un mensaje MQTTv3 y un mensaje IBM MQ; consulte "Integración de MQ Telemetry con gestores de colas" en la página 133.

Utilice la publicación, MQInput y los nodos Nodos de entrada de JMS para enviar y recibir publicaciones entre clientes de IBM Integration Bus y MQTT.

Utilizando flujos de mensajes puede integrar la telemetría con sitios web utilizando HTTP, y con otras aplicaciones utilizando IBM MQ y WebSphere Adapters.

Windows Linux AIX Integración de MQ Telemetry con gestores de colas

El cliente MQTT está integrado con IBM MQ como aplicación de publicación/suscripción. Puede publicar o suscribirse a temas en IBM MQ, crear nuevos temas o utilizar temas existentes. Recibe publicaciones de IBM MQ como resultado de los clientes MQTT, incluido él mismo u otras aplicaciones de IBM MQ que se publican en los temas de sus suscripciones. Las reglas se aplican para decidir los atributos de una publicación.

Muchos de los atributos asociados a temas, publicaciones, suscripciones y mensajes que proporciona IBM MQ no están soportados. En "Cliente de MQTT al intermediario de publicación/suscripción de IBM MQ" en la página 133 y en "IBM MQ a un cliente MQTT" en la página 135 se describe cómo se configuran los atributos de publicaciones. Los valores dependen de si la publicación va hacia o desde el intermediario de publicación/suscripción de IBM MQ.

En IBM MQ los temas de publicación/suscripción están asociados con objetos de tema administrativo. Los temas creados por clientes MQTT no son diferentes. Cuando un cliente MQTT crea una serie de tema para una publicación, el intermediario de publicación/suscripción de IBM MQ lo asocia a un objeto de tema administrativo. El intermediario correlaciona la serie de tema en la publicación con el padre de objeto de tema administrativo más cercano. La correlación es la misma que para las aplicaciones IBM MQ. Si no hay ningún tema que haya creado el usuario, la publicación de temas se correlaciona a `SYSTEM.BASE.TOPIC`. Los atributos que se aplican a la publicación se derivan del objeto de tema.

Cuando una aplicación de IBM MQ o un administrador crea una suscripción, la suscripción tiene un nombre. Listar suscripciones usandoIBM MQ Explorer , o mediante el uso**runmqsc** o comandos PCF. A todas las suscripciones de cliente MQTT se les asigna un nombre. Se les da un nombre de la forma:*`ClientIdentifier:Topic name`*

Cliente de MQTT al intermediario de publicación/suscripción de IBM MQ

Un cliente MQTT ha enviado una publicación a IBM MQ. El servicio de telemetría (MQXR) convierte la publicación en un mensaje de IBM MQ. El mensaje de IBM MQ contiene tres partes:

1. `MQMD`
2. `RFH2`
3. Mensaje

Las propiedades de `MQMD` se establecen en sus valores predeterminados, excepto cuando se indique lo contrario en la Tabla 9 en la página 134.

Tabla 9. Valores de las propiedades MQMD		
Campo MQMD	**Tipo**	**Valor**
Format	MQCHAR8	MQFMT_RF_HEADER_2
UserIdentifier	MQCHAR12	Establezca uno de los valores siguientes: MqttClient.ClientIdentifier MqttConnectOptions.UserName Un ID de usuario establecido por el administrador de IBM MQ para el canal de telemetría.
Priority	MQLONG	MQPRI_PRIORITY_AS_Q_DEF (Valor predeterminado para IBM MQ, que es diferente de JMS que tiene un valor predeterminado de 4).
Persistence	MQLONG	QoS=0→MQPER_NOT_PERSISTENT QoS=1→MQPER_PERSISTENT QoS=2→MQPER_PERSISTENT

La cabecera RFH2 no contiene una carpeta <msd> para definir el tipo del mensaje de JMS. El servicio de telemetría (MQXR) crea el mensaje de IBM MQ como un mensaje JMS predeterminado. El tipo de mensaje predeterminado de JMS es un mensaje `jms-bytes`. Una aplicación puede acceder a información de cabecera adicional como propiedades de mensaje; consulte Propiedades del mensaje.

Los valores de RFH2 se establecen tal como se muestra en la Tabla 10 en la página 134. El formato de la propiedad se establece en la cabecera fija RFH2, y los demás valores se establecen en las carpetas de RFH2.

Tabla 10. Valores de las propiedades RFH2		
Propiedad de RFH2	**Tipo/carpeta**	**Cabecera**
Formato	MQCHAR8	MQFMT_NONE
ClientIdentifier	mqtt/clientId	Copiar MqttClient.ClientIdentifier con una longitud de 1...23 bytes.
QoS	mqtt/qos	Copiar QoS del mensaje MQTT entrante.
ID de mensaje	mqtt/msgid	Copiar ID de mensaje del mensaje entrante MQTT, si QoS es 1 o 2.
MQIsRetained	mqps/Ret	Establecer si la publicación MQTT original se ha enviado con el conjunto de propiedades RETAIN y el mensaje se recibe como una publicación retenida.
MQTopicString	mqps/Top	El tema en el que se ha publicado el mensaje MQTT.

La carga útil de una publicación MQTT se correlaciona con el contenido de un mensaje de IBM MQ:

Tabla 11. Cómo la carga útil en una publicación de MQTT se correlaciona con el contenido del mensaje de IBM MQ		
Contenido del mensaje	**Tipo**	**Contenido del mensaje de IBM MQ**
`Buffer`	`MQBYTE` *n*	Copia de bytes del mensaje MQTT entrante. La longitud puede ser cero.

IBM MQ a un cliente MQTT

Un cliente se ha suscrito a un tema de publicación. Una aplicación de IBM MQ ha publicado en el tema, y en consecuencia, el intermediario de publicación/suscripción de MQTT envía una publicación al suscriptor de IBM MQ. De forma alternativa, una aplicación de IBM MQ ha enviado un mensaje no solicitado directamente a un cliente MQTT. En la Tabla 12 en la página 135 se describe cómo se establecen las cabeceras de mensajes fijos en el mensaje que se envía al cliente MQTT. Los demás datos de la cabecera de mensaje de IBM MQ o de cualquier otra cabecera, se eliminan. Los datos de mensaje del mensaje de IBM MQ se envían como carga útil del mensaje en el mensaje MQTT, sin ninguna alteración. El mensaje MQTT se envía al cliente MQTT a través del servicio de telemetría (MQXR).

Tabla 12. Cómo se establecen las cabeceras de mensajes fijas en un mensaje de IBM MQ enviado al cliente de MQTT		
Campo de MQTT	**Tipo**	**Valor**
DUP	`Boolean`	Establézcalo si `QoS = 1` o 2 y el mensaje se han enviado a este cliente en una transmisión anterior, y el mensaje no se ha reconocido transcurrido un tiempo.

QoS	`entero`	La forma en que se establece el valor de QoS en una publicación saliente desde el intermediario de publicación/suscripción en IBM MQ depende de la publicación entrante. Depende de si la publicación entrante se ha enviado desde un cliente MQTT o desde una aplicación IBM MQ.
		MQTT
		Valor inferior del QoS en la publicación entrante, y en el de QoS que ha solicitado el suscriptor.
		IBM MQ
		Valor inferior del QoS derivado de la publicación entrante:
		`MQPER_NOT_PERSISTENT→QoS=0`
		`MQPER_PERSISTENT→QoS=2`
		y el QoS que ha solicitado el suscriptor. Si el mensaje se envía al cliente sin una suscripción, QoS se establece, de forma predeterminada, en 2. Un cliente puede modificar este valor suscribiéndose a `DEFAULT.QoS` con un QoS diferente.

RETAIN	`Boolean`	Establecer si la publicación de entrada tiene establecida la propiedad retenida.

En la Tabla 13 en la página 136 se describe cómo las cabeceras de mensajes variables se establecen en el mensaje MQTT que se envía al cliente MQTT.

Tabla 13. Cómo se establecen las propiedades de cabecera variables de MQTT en un mensaje de MQTT que se envía al cliente de MQTT

Campo de MQTT	Tipo	Valor
Topic name	Serie	La serie de tema con la que se ha publicado el mensaje MQTT.
Message ID	Serie	Los 2 últimos bytes de la propiedad `MQMD.MsgId` de la publicación, cuando se coloca en la cola `SYSTEM.MQTT.TRANSMIT.QUEUE`.
Payload	byte[]	Copia directa de los bytes procedentes de la publicación entrante al intermediario de publicación/suscripción. La longitud puede ser cero.

Windows Linux AIX Sesiones de MQTT con estado y sin estado

Los clientes MQTT pueden crear una sesión con estado con el gestor de colas. Cuando un cliente MQTT con estado se desconecta, el gestor de colas mantiene las suscripciones creadas por el cliente y en los mensajes en curso. Cuando el cliente se vuelve a conectar, resuelve los mensajes en curso. Envía los mensajes que están en la cola de entrega para realizar su entrega, y recibe los mensajes publicados para sus suscripciones mientras estaba desconectado.

Cuando un cliente MQTT se conecta a un canal de telemetría, inicia una nueva sesión o bien reanuda una sesión antigua. Una sesión nueva no tiene ningún mensaje pendiente sin acuse de recibo, ninguna suscripción, ni publicación a la espera de ser entregada. Cuando un cliente se conecta, especifica si se empieza con una sesión limpia o si se reanuda una sesión existente; consulte Limpiar sesiones.

Si el cliente reanuda una sesión existente, continúa como si la conexión no se hubiera interrumpido. Las publicaciones que están esperando a ser entregadas se envían al cliente y las transferencias de mensajes no confirmadas se completan. Cuando un cliente de una sesión persistente se desconecta del servicio de telemetría (MQXR), todas las suscripciones que el cliente ha creado permanecen. Las publicaciones de las suscripciones se envían al cliente cuando éste vuelve a conectarse. Si se vuelve a conectar sin reanudar la sesión anterior, el servicio de telemetría (MQXR) descarta las publicaciones.

El gestor de colas guarda la información del estado de la sesión en la cola `SYSTEM.MQTT.PERSISTENT.STATE`.

El administrador de IBM MQ puede desconectar y depurar una sesión.

Windows Linux AIX Cuando un cliente MQTT no está conectado

Cuando un cliente no se conecta al gestor de colas, puede continuar recibiendo publicaciones en su nombre. Se reenvían al cliente cuando éste se vuelve a conectar. Un cliente puede crear una "Última voluntad y testamento" para que el gestor de colas realice publicaciones en su nombre cuando éste se desconecte de forma inesperada.

Si desea recibir notificaciones cuando el cliente se desconecte de forma inesperada, puede registrar una publicación de "Última voluntad y testamento"; consulte la publicación Última voluntad y testamento. Se envía a través del servicio de telemetría (MQXR), si éste detecta que la conexión con el cliente se ha interrumpido sin que este último lo haya solicitado.

Un cliente puede publicar una publicación retenida en cualquier momento; consulte Publicaciones retenidas y clientes MQTT. Puede solicitarse que una suscripción nueva a un tema se envíe con cualquier publicación retenida asociada al mismo. Si crea la última voluntad y testamento como una publicación retenida, puede utilizarla para supervisar el estado de un cliente.

Por ejemplo, el cliente publica una publicación retenida, cuando se conecta, informando de su disponibilidad. Al mismo tiempo, crea una publicación retenida de última voluntad y testamento que anuncia que no está disponible. Además, justo antes de realizar una desconexión planificada, publica

su no disponibilidad como una publicación retenida. Para averiguar si el cliente está disponible, debe suscribirse al tema de la publicación retenida. Siempre recibirá una de las tres publicaciones.

Si el cliente va a recibir mensajes publicados cuando se desconecta, vuelva a conectarlo a su sesión anterior; consulte “Sesiones de MQTT con estado y sin estado” en la página 136. Sus suscripciones quedan activas hasta que se supriman o hasta que el cliente cree una sesión limpia.

Windows Linux AIX Acoplamiento dinámico entre clientes MQTT y aplicaciones de IBM MQ

El flujo de publicaciones entre los clientes MQTT y las aplicaciones IBM MQ se acopla de forma dinámica. Las publicaciones pueden originarse en un cliente MQTT o en una aplicación de IBM MQ y sin ningún orden establecido. Los publicadores y los suscriptores son de acoplamiento dinámico. Interactúan unos con otros de forma indirecta a través de publicaciones y suscripciones. También puede enviar mensajes directamente a un cliente MQTT desde una aplicación de IBM MQ.

Los clientes MQTT y las aplicaciones de IBM MQ se pueden acoplar dinámicamente en dos sentidos:

1. Los publicadores y suscriptores tienen un acoplamiento dinámico por la asociación de una publicación y una suscripción con un tema. Los publicadores y suscriptores normalmente no tienen en cuenta ni la dirección ni la identidad del otro origen de una publicación o suscripción.
2. Los clientes MQTT publican, suscriben, reciben publicaciones y procesan reconocimientos de entrega en hebras separadas.

Una aplicación de cliente MQTT no espera hasta que se haya entregado una publicación. La aplicación transfiere un mensaje al cliente MQTT y a continuación, la aplicación continúa en su propia hebra. Para sincronizar la aplicación con la entrega de una publicación, se utiliza una señal de entrega; consulte Entrega de señales.

Después de pasar un mensaje al cliente MQTT, la aplicación tiene la opción de esperar en la señal de entrega. En lugar de esperar, el cliente puede proporcionar un método de devolución de llamada que se invoca cuando la publicación se entrega a IBM MQ. También Puede ignorar la señal de entrega.

En función de la calidad de servicio asociada al mensaje, se devuelve la señal de entrega de forma inmediata al método de devolución de llamada, o puede hacerse una vez transcurrido cierto tiempo. La señal de entrega puede devolverse incluso después de que el cliente se desconecte y vuelva a conectarse. Si la calidad de servicio es *enviar y olvidar*, la señal de entrega se devuelve inmediatamente. En los otros dos casos, la señal de entrega sólo se devuelve cuando el cliente recibe el acuse de recibo indicando que la publicación se ha enviado a los suscriptores.

Las publicaciones que se envían a un cliente MQTT como resultado de una suscripción cliente se entregan al método de devolución de llamada `messageArrived`. `messageArrived` se ejecuta en una hebra diferente a la de la aplicación principal.

Envío de mensajes directamente a un cliente MQTT

Puede enviar un mensaje a un cliente MQTT determinado de una de estas dos formas.

1. Una aplicación de IBM MQ puede enviar un mensaje directamente a un cliente MQTT sin una suscripción; consulte Envío de un mensaje a un cliente directamente.
2. Un enfoque alternativo es utilizar el convenio de denominación `ClientIdentifier`. Haga que todos los suscriptores MQTT creen suscripciones utilizando como tema el `ClientIdentifier` exclusivo. Publique en *Identificador_cliente*. La publicación se envía al cliente que esté suscrito al tema *Identificador_cliente*. Mediante esta técnica puede enviar una publicación a un suscriptor MQTT determinado.

Windows Linux AIX Seguridad de MQ Telemetry

La seguridad de los dispositivos de telemetría puede ser importante, ya que suelen ser portátiles y suelen utilizarse en lugares que no pueden controlarse de forma precisa. Puede utilizar VPN para proteger

la conexión del dispositivo MQTT en el servicio de telemetría (MQXR). MQ Telemetry proporciona dos mecanismos de seguridad adicionales, TLS y JAAS.

TLS se utiliza principalmente para cifrar las comunicaciones entre el dispositivo y el canal de telemetría y para autenticar que el dispositivo se está conectando al servidor correcto; consulte Autenticación del canal de telemetría utilizando TLS. También puede utilizar TLS para comprobar si el dispositivo cliente tiene permiso para conectarse al servidor; consulte Autenticación de cliente MQTT utilizando TLS.

JAAS se utiliza principalmente para comprobar si el usuario del dispositivo tiene permiso para utilizar una aplicación de servidor; consulte Autenticación de cliente MQTT utilizando una contraseña. JAAS puede utilizarse con LDAP para comprobar una contraseña que utiliza un directorio de inicio de sesión único.

TLS y JAAS pueden utilizarse conjuntamente para proporcionar dos factores de autenticación. Puede restringir los cifrados que utiliza TLS a cifrados que cumplan los estándares FIPS.

Con decenas de miles de usuarios, no siempre resulta práctico proporcionar perfiles individuales de seguridad. Tampoco resulta práctico utilizar los perfiles para autorizar a usuarios individuales a acceder a objetos de IBM MQ. En su lugar, es mejor agrupar los usuarios en clases para la autorización de publicaciones y suscripciones a temas y enviar publicaciones a los clientes.

Configure cada canal de telemetría para que se correlacione a clientes con ID de usuario de cliente comunes. Utilice un ID de usuario común para cada cliente que se conecta en un canal específico; consulte Identidad y autorización de cliente MQTT.

La autorización de grupos de usuarios no compromete la autenticación de cada individuo. Cada usuario puede autenticarse, en el servidor o en el cliente, con su valor de `Username` y de `Password`, y luego autorizarse en el servidor mediante un ID de usuario común.

Windows Linux AIX Globalización de MQ Telemetry

La carga útil del mensaje en el protocolo MQTT v3 se codifica como una matriz de bytes. Por lo general, las aplicaciones que manejan texto crean la carga útil del mensaje en formato UTF-8. El canal de telemetría describe la carga útil del mensaje como UTF-8, pero no realiza ninguna conversión de páginas de códigos. La serie del tema de publicación debe estar en UTF-8.

La aplicación se encarga de convertir los datos alfanuméricos a la página de códigos correcta y los datos numéricos a la codificación de números correcta.

El cliente MQTT Java tiene un cómodo método `MqttMessage.toString`. El método trata la carga útil del mensaje como si estuviera codificada en el conjunto de caracteres predeterminado de la plataforma que, por lo general, es UTF-8. Convierte la carga útil en una String de Java. Java cuenta con un método String, `getBytes` que convierte una serie en una matriz de bytes codificada utilizando el conjunto de caracteres predeterminado de la plataforma local. Dos programas MQTT Java que intercambien texto en la carga útil del mensaje entre plataformas con el mismo conjunto de caracteres predeterminado lo hacen de una forma fácil y eficaz en UTF-8.

Si el conjunto de caracteres predeterminado de una de las plataformas no es UTF-8, las aplicaciones deben establecer un convenio para el intercambio de mensajes. Por ejemplo, el publicador especifica la conversión de una serie a UTF-8 mediante el método `getBytes("UTF8")`. Para recibir el texto de un mensaje, el suscriptor presupone que el mensaje se ha codificado en el juego de caracteres UTF-8.

El servicio de telemetría (MQXR) describe la codificación de todas las publicaciones entrantes de mensajes de clientes MQTT como si fuera UTF-8. Establece `MQMD.CodedCharSetId` a UTF-8, y `RFH2.CodedCharSetId` a `MQCCSI_INHERIT` ; consulte “Integración de MQ Telemetry con gestores de colas” en la página 133. El formato de la publicación se establece en `MQFMT_NONE`, por lo que ni los canales, ni `MQGET`, pueden realizar la conversión.

Windows Linux AIX Rendimiento y escalabilidad de MQ Telemetry

Tenga en cuenta los siguientes factores al gestionar un gran número de clientes y mejorar la escalabilidad de MQ Telemetry.

Planificación de la capacidad

Para obtener información sobre los informes de rendimiento de MQ Telemetry, consulte los documentos de MQ Performance.

Conexiones

Los costes que conllevan las conexiones incluyen:

- El gasto de configuración de la misma conexión en términos de uso del procesador y de tiempo.
- Costes de la red.
- Memoria usada cuando se mantiene una conexión abierta que no se utilizada.

Existe una carga adicional que se produce cuando los clientes permanecen conectados. Si una conexión se mantiene abierta, los flujos de TCP/IP y los mensajes de MQTT utilizan la red para comprobar que la conexión aún está activa. Además, se utiliza memoria del servidor cada vez que la conexión de cliente se mantiene abierta.

Si se envía más de un mensaje por minuto, debe mantener la conexión abierta para evitar el coste que supone iniciar una nueva conexión. Si envía menos de un mensaje cada 10 ó 15 minutos, considere la posibilidad de interrumpir la conexión para evitar el gasto que supone mantenerla abierta. Es posible que desee mantener una conexión TLS abierta, pero desocupada, durante períodos de tiempo más largos, porque es más cara de configurar.

Tenga en cuenta, además, la capacidad del cliente. Si el cliente cuenta con una recurso de almacén y reenvío, puede configurar los mensajes por lotes, e interrumpir la conexión entre los envíos de los procesos por lotes. Sin embargo, si el cliente se desconecta, el cliente no podrá recibir ningún mensaje del servidor. Por lo tanto, el objetivo de la aplicación está relacionado con la decisión.

Si el sistema cuenta con un cliente que envía muchos mensajes, por ejemplo, transfiere archivos, es mejor no esperar la respuesta del servidor por mensaje. En su lugar, envíe todos los mensajes y compruebe al final que se han recibido. De forma alternativa, utilice Calidad de servicio (QoS).

Puede variar la QoS según el mensaje, suministrando mensajes sin importancia con QoS 0 y mensajes importantes utilizando una QoS de 2. La capacidad de proceso de mensajes puede ser aproximadamente el doble con una QoS de 0 que con una QoS de 2.

Convenios de denominación

Si diseña la aplicación para muchos clientes, implemente un convenio de denominación eficaz. Para correlacionar cada cliente al `ClientIdentifier` correcto, haga que el `ClientIdentifier` sea lo más significativo posible. Un buen convenio de denominación facilita al administrador el poder averiguar qué clientes se están ejecutando. Un convenio de denominación ayuda al administrador a filtrar una larga lista de clientes en IBM MQ Explorer y ayuda en la determinación de problemas; consulte Identificador de cliente.

Rendimiento

La longitud de los nombres de tema afecta al número de bytes que se transmiten por la red. Cuando se publica o se realiza una suscripción, el número de bytes de un mensaje puede ser importante. Por tanto, es recomendable que limite el número de caracteres del nombre de un tema. Cuando un cliente MQTT se suscribe a un tema IBM MQ le da un nombre del formulario:

```
ClientIdentifier: TopicName
```

Para ver todas las suscripciones para un cliente MQTT, puede utilizar el mandato IBM MQ MQSC **DISPLAY**:

```
DISPLAY SUB(' ClientID1:*')
```

Definición de recursos en IBM MQ para que los utilicen los clientes MQTT

Un cliente MQTT se conecta a un gestor de colas remoto de IBM MQ. Hay dos métodos básicos para que una aplicación IBM MQ envíe mensajes a un cliente MQTT : establecer la cola de transmisión predeterminada en `SYSTEM.MQTT.TRANSMIT.QUEUE` o utilizar alias de gestor de colas. Defina la cola de transmisión predeterminada de un gestor de colas; si existe un gran número de clientes MQTT. Si se utiliza el valor de cola de transmisión predeterminado se simplifica el esfuerzo de administración, consulte Configurar gestión de colas distribuidas para enviar mensajes a clientes MQTT.

Mejora de la escalabilidad evitando suscripciones.

Cuando un cliente de MQTT V3 se suscribe a un tema, el servicio de telemetría (MQXR) crea una suscripción en IBM MQ. La suscripción dirige las publicaciones del cliente a `SYSTEM.MQTT.TRANSMIT.QUEUE`. El nombre de gestor de colas remoto que aparece en la cabecera de transmisión de cada publicación se establece en el `ClientIdentifier` del cliente MQTT que realizó la suscripción. Si hay muchos clientes, cada uno de ellos realizando sus propias suscripciones, esto da como resultado que se mantengan muchas suscripciones de proxy en toda la jerarquía o clúster de publicación/suscripción de IBM MQ. Para obtener información sobre cómo no utilizar la publicación/suscripción pero, cómo utilizar una solución de tipo punto a punto, consulte Envío de un mensaje a un cliente directamente.

Gestión de grandes cantidades de clientes

Para dar soporte a muchos clientes conectados al mismo tiempo, aumente la memoria disponible para el servicio de telemetría (MQXR) definiendo los parámetros de JVM **-Xms** y **-Xmx**. Siga estos pasos:

1. Busque el archivo `java.properties` en el directorio de configuración del servicio de telemetría; consulte el directorio de configuración del servicio de telemetría (MQXR) en Windows o el directorio de configuración del servicio de telemetría en Linux.
2. Siga las instrucciones del archivo; un almacenamiento dinámico de 1 GB es suficiente para que haya 50.000 clientes conectados simultáneamente.

```
# Heap sizing options - uncomment the following lines to set the heap to 1G
#-Xmx1024m
#-Xms1024m
```

3. Añada otros argumentos de línea de mandatos para pasar a la JVM que ejecuta el servicio de telemetría (MQXR) en el archivo `java.properties`; consulte Cómo pasar parámetros de JVM al servicio de telemetría (MQXR).

Para aumentar el número de descriptores de archivo abiertos en Linux, añada las líneas siguientes a `/etc/security/limits.conf/` y vuelva a iniciar sesión.

```
@mqm soft nofile 65000
@mqm hard nofile 65000
```

Cada socket necesita un descriptor de archivo. El servicio de telemetría necesita algunos descriptores de archivo adicionales, por lo que este número debe ser mayor que el número de sockets abiertos necesarios.

El gestor de colas utiliza un manejador de objeto para cada suscripción no duradera. Para dar soporte a muchas suscripciones no duraderas activas, aumente el número máximo de manejadores activos en el gestor de colas; por ejemplo:

```
echo ALTER QMGR MAXHANDS(999999999) | runmqsc qMgrName
```

Figura 48. Modificar el número máximo de manejadores en Windows

```
echo "ALTER QMGR MAXHANDS(999999999)" | runmqsc qMgrName
```

Figura 49. Modificar el número máximo de manejadores en Linux

Otras consideraciones

Cuando planifique los requisitos del sistema, tenga en cuenta el tiempo que necesitará para reiniciar el sistema. El tiempo de inactividad planificado puede tener implicaciones para el número de mensajes que se acumulen en la cola, a la espera de ser procesados. Configure el sistema de modo que los mensajes se puedan procesar de forma satisfactoria en un tiempo aceptable. Revise el almacenamiento de disco, la memoria y la potencia de proceso. Con algunas aplicaciones cliente, es posible descartar mensajes cuando el cliente se vuelve a conectar. Para descartar mensajes, establezca `CleanSession` en los parámetros de conexión del cliente; consulte Borrado de sesiones. O bien, publique y suscriba utilizando el nivel máximo de calidad de servicio, `0`, en un cliente MQTT; consulte Calidad de servicio. Utilice mensajes no `persistentes` cuando envíe mensajes desde IBM MQ. Los mensajes que tienen estas calidades de servicio no se recuperan cuando se reinicia el sistema o la conexión.

Windows Linux AIX Dispositivos soportados por MQ Telemetry

Los clientes MQTT se pueden ejecutar en una gama de dispositivos, desde sensores y actuadores hasta dispositivos portátiles y sistemas para vehículos.

Los clientes MQTT son pequeños y se ejecutan en dispositivos con poca memoria y una potencia de proceso restringida. El MQTT protocol es fiable y tiene cabeceras pequeñas, que son ideales para redes limitadas por un ancho de banda reducido, un alto coste y una disponibilidad intermitente.

MQ Telemetry se comunica con dispositivos de telemetría a través de aplicaciones cliente MQTT. Estas aplicaciones utilizan los siguientes recursos, que implementan todos ellos el protocolo MQTT v3:

- Las siguientes bibliotecas cliente:
  - *MQTT client for Java* , que se utiliza para compilar aplicaciones nativas para (por ejemplo), dispositivos Android, OS X, Linux o Windows. Las aplicaciones que utilizan esta biblioteca de cliente se pueden ejecutar en todas las variaciones de Java desde CLDC (Connected Limited Device Configuration)/MIDP (Mobile Information Device Profile), a través de CDC (Connected Device Configuration)/Foundation, J2SE (Java Platform, Standard Edition) y J2EE (Java Platform, Enterprise Edition). La biblioteca de clases personalizadas jclRM de IBM también está soportada. La plataforma Java ME se suele utilizar en pequeños dispositivos, como por ejemplo, actuadores, sensores, teléfonos móviles y otros dispositivos integrados. La plataforma Java SE se instala por lo general en dispositivos de gama superior e integrados, como por ejemplo, sistemas y servidores.
  - *MQTT client for C*, que se utiliza para crear aplicaciones nativas para (por ejemplo) dispositivos iOS, OS X, Linux o Windows. Esta biblioteca de cliente proporciona una implementación de referencia C junto con el cliente nativo precompilado para sistemas Windows y Linux. La implementación de referencia C permite la portabilidad de MQTT a una amplia gama de dispositivos y plataformas. Algunos sistemas Windows en Intel, incluidos Windows 7, RedHat, Ubuntu y algunos sistemas Linux en plataformas ARM como Eurotech Viper, implementan versiones de Linux que ejecutan el cliente C, pero IBM no proporciona soporte de servicio para las plataformas. Deberá reducir los problemas con el cliente en una plataforma soportada si tiene previsto llamar al centro de soporte de IBM.
  - *MQTT client for Java* , que se utiliza para compilar aplicaciones web basadas en navegador.

MQTTLas bibliotecas cliente de están disponibles gratuitamente en Eclipse Paho y MQTT.org. Consulte Programas de ejemplo de IBM MQ Telemetry Transport.

Seguridad en IBM MQ

En IBM MQ, existen varios métodos para proporcionar seguridad: la interfaz del servicio de autorización; rutinas de salida de canal escritas por el usuario o por terceros; seguridad de canal utilizando la seguridad de la capa de transporte (TLS), registros de autenticación de canal y seguridad de mensajes.

Interfaz del servicio de autorización

La autorización para utilizar llamadas MQI, mandatos y acceso a objetos la proporciona el **gestor de autorizaciones sobre objetos** (OAM), que está habilitado de forma predeterminada. El acceso a las entidades de IBM MQ se controla a través de grupos de usuarios de IBM MQ y el OAM. Los administradores pueden utilizar una interfaz de línea de mandatos para otorgar o revocar las autorizaciones según sea necesario.

Para obtener más información sobre la creación de componentes del servicio de autorización, consulte Configuración de la seguridad en sistemas AIX, Linux, and Windows.

Salidas de canal escritas por el usuario o de otros fabricantes

Los canales pueden utilizar salidas de canal escritas por el usuario o de otros fabricantes. Para obtener más información, consulte Programas de salida de canal para canales de mensajería.

Seguridad de canal mediante TLS

El protocolo TLS (Transport Layer Security) proporciona la seguridad de canal estándar de la industria, con protección contra escuchas y manipulaciones no autorizadas y contra falsas identidades.

TLS utiliza técnicas de clave pública y simétricas para proporcionar confidencialidad e integridad así como autenticación mutua.

Para obtener una revisión completa de la seguridad en IBM MQ, incluida información detallada sobre TLS, consulte Seguridad. Para obtener una visión general de TLS, incluidos los punteros a los mandatos que se describen en este apartado, consulte Protocolos de seguridad de cifrado: TLS.

Registros de autenticación de canal

Utilice registros de autenticación de canal para ejercer un control preciso sobre el acceso otorgado a conectar sistemas en un nivel de canal. Para obtener más información, consulte Registros de autenticación de canal.

Seguridad de mensajes

Utilice Advanced Message Security, que es un componente con licencia y instalado por separado de IBM MQ, para proporcionar protección criptográfica a los mensajes enviados y recibidos utilizando IBM MQ. Consulte Advanced Message Security.

Tareas relacionadas

Seguridad

Planificación de los requisitos de seguridad

Soporte TLS de cliente gestionado de IBM MQ.NET

El cliente IBM MQ.NET completamente gestionado proporciona soporte de seguridad de la capa de transporte (TLS) que se basa en el kit Microsoft.NET SSLStreams. Esto es diferente de los otros clientes IBM MQ , que se basan en IBM Global Security Kit (GSKit).

Puede desarrollar aplicaciones IBM MQ.NET para que se ejecuten en modalidad gestionada o en modalidad no gestionada.

- En modalidad gestionada, las aplicaciones .NET funcionan en CLR (Common Language Rutime) .NET sin ninguna invocación entre plataformas al invocar la MQI C.
- En modalidad no gestionada, C MQI se invoca para las operaciones de MQI subyacentes. Básicamente, la interfaz en modalidad no gestionada consta de las clases de derivador .NET en la parte superior de C MQI.

El cliente de IBM MQ.NET gestionado utiliza las bibliotecas de Microsoft.NET Framework para implementar protocolos de sockets seguros TLS. El sistema.NET.Security.La clase SSLStream de Microsoft se utiliza para implementar la seguridad (TLS) en IBM MQ.NET.

La modalidad de cliente de IBM MQ.NET no gestionada ya da soporte a la característica TLS, que se basa en C MQI (y GSKit). Es decir, C MQI maneja las operaciones TLS. En este caso, GSKit implementa los protocolos de socket seguro TLS.

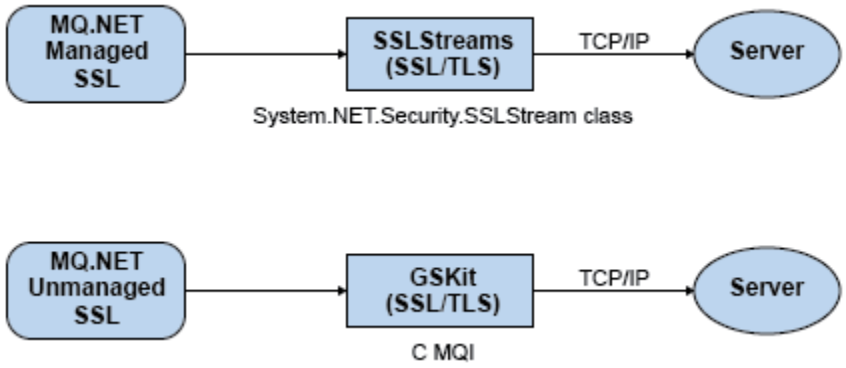


Figura 50. Comparación de TLS gestionada y no gestionada de IBM MQ.NET

En la tabla siguiente se resumen las diferencias entre las implementaciones gestionadas y no gestionadas:

Tabla 14. Diferencias entre implementaciones gestionadas y no gestionadas

Modalidad	Protocolos	Implementación	Comentarios
SSL gestionado de IBM MQ.NET	TLS	Clase System.NET.Security.SSLStream La clase SSLStream funciona como una corriente a través de un socket TCP conectado.	TLS 1.0 TLS 1.2 (solo con Microsoft.NET Framework v4.5)
SSL no gestionado de IBM MQ.NET	TLS	GSKit y C-MQI	Protocolos de sockets seguros TLS

Conceptos relacionados

Soporte de SSL (Secure Sockets Layer) y TLS (Transport Layer Security) para .NET

IBM MQ MQI clients

Un IBM MQ MQI client es un componente del producto IBM MQ que puede instalarse en un sistema en el que no se ejecuta ningún gestor de colas.

Un IBM MQMQI *cliente* es un componente que permite que una aplicación que se ejecuta en un sistema emita llamadas MQI a un gestor de colas que se ejecuta en otro sistema. La salida de la llamada se devuelve al cliente, que la devuelve a su vez a la aplicación.

Utilizando un IBM MQ MQI client, una aplicación que se ejecuta en el mismo sistema que el cliente puede conectarse a un gestor de colas que se ejecuta en otro sistema. La aplicación puede emitir llamadas MQI a ese gestor de colas. Dicha aplicación se denomina aplicación IBM MQ MQI client y el gestor de colas se denomina *gestor de colas servidor*.

Un IBM MQ *servidor* es un gestor de colas que proporciona servicios de gestión de colas a uno o más clientes. Todos los objetos de IBM MQ, por ejemplo colas, solamente existen en la máquina del gestor de

colas (la máquina del servidor de IBM MQ) y no en el cliente. Un servidor de IBM MQ también puede dar soporte a aplicaciones de IBM MQ locales.

La diferencia entre un servidor de IBM MQ y un gestor de colas ordinario es que un servidor tiene un enlace de comunicaciones dedicado con cada cliente. Puede obtener información adicional sobre la creación de canales para clientes y servidores consultando Configuración de colas distribuidas.

Una aplicación IBM MQ MQI client y un gestor de colas de servidor se comunican entre sí utilizando un *canal MQI*. Un canal MQI se inicia cuando la aplicación de cliente emite una llamada **MQCONN** o **MQCONNX** para conectarse al gestor de colas y finaliza cuando la aplicación de cliente emite una llamada **MQDISC** para desconectarse del gestor de colas. Los parámetros de entrada de una llamada MQI fluyen en una dirección de un canal MQI y los parámetros de salida fluyen en la dirección opuesta.

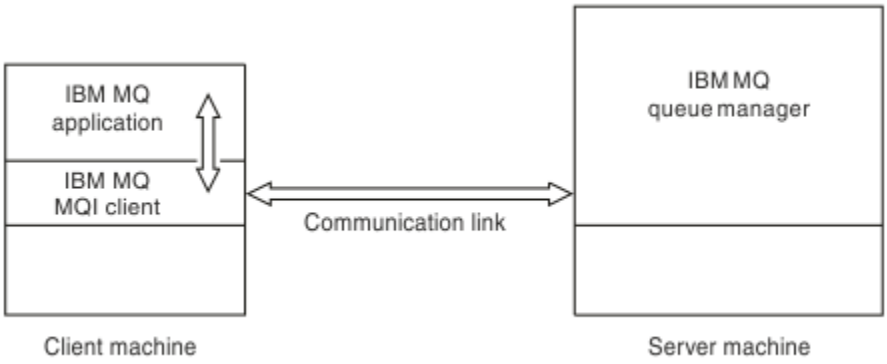


Figura 51. Enlace entre un cliente y un servidor

Se pueden utilizar las plataformas siguientes. Las combinaciones dependen del producto IBM MQ que esté utilizando y se describen en "Soporte de plataforma para clientes de IBM MQ" en la página 145.

IBM MQ MQI client	**Servidor de IBM MQ**
AIX and Linux Windows IBM i	AIX and Linux Windows IBM i z/OS

MQI está disponible para las aplicaciones que se ejecutan en la plataforma cliente; las colas y otros objetos de IBM MQ se mantienen en un gestor de colas que ha instalado en un servidor.

Una aplicación que se desea ejecutar en el entorno IBM MQ MQI client debe enlazarse primero con la biblioteca de cliente relevante. Cuando la aplicación emite una llamada MQI, el IBM MQ MQI client dirige la solicitud a un gestor de colas, donde se procesa y desde donde se envía una respuesta al IBM MQ MQI client.

El enlace entre la aplicación y el IBM MQ MQI client se establece dinámicamente en tiempo de ejecución.

También puede desarrollar aplicaciones cliente utilizando las IBM MQ classes for .NET, IBM MQ classes for Java o IBM MQ classes for Java Message Service (JMS). Puede utilizar clientes Java y JMS en las plataformas siguientes:

- IBM i
- AIX
- Linux
- Windows

El uso de Java y JMS no se describe aquí. Para obtener detalles completos acerca de la instalación, configuración y uso de IBM MQ classes for Java y IBM MQ classes for JMS, consulte Utilización de IBM MQ classes for Java y Utilización de IBM MQ classes for JMS.

Aplicaciones IBM MQ en un entorno cliente-servidor

Cuando se enlazan con un servidor, las aplicaciones cliente de IBM MQ pueden emitir la mayoría de llamadas MQI de la misma forma que las aplicaciones locales. La aplicación cliente emite una llamada `MQCONN` para conectarse a un gestor de colas especificado. Todas las llamadas MQI adicionales que especifiquen el manejador de conexión devuelto desde la petición de conexión serán procesadas entonces por este gestor de colas.

Debe enlazar sus aplicaciones con las bibliotecas de cliente adecuadas. Consulte Creación de aplicaciones para IBM MQ MQI clients.

Conceptos relacionados

"¿Por qué utilizar clientes IBM MQ" en la página 145
La utilización de clientes IBM MQ es una forma eficaz de implementar la mensajería y la transferencia a colas de IBM MQ.

"¿Qué es un cliente transaccional extendido?" en la página 147
Un cliente transaccional extendido de IBM MQ puede actualizar recursos gestionados por otro gestor de recursos, bajo el control de un gestor de transacciones externo.

"Cómo se conecta el cliente al servidor" en la página 148
Un cliente se conecta a un servidor utilizando MQCONN o MQCONNX y se comunica a través de un canal.

"Gestión y soporte de transacciones" en la página 149
Introducción a la gestión de transacciones y cómo IBM MQ da soporte a las transacciones.

"Extensión de recursos del gestor de colas" en la página 151
Puede ampliar los recursos del gestor de colas utilizando las salidas de usuario, las salidas de API, o los servicios instalables.

Información relacionada

Cómo configurar un IBM MQ MQI client

¿Por qué utilizar clientes IBM MQ

La utilización de clientes IBM MQ es una forma eficaz de implementar la mensajería y la transferencia a colas de IBM MQ.

Puede tener una aplicación que utilice la interfaz MQI en una máquina y el gestor de colas en ejecución en una máquina diferente (física o virtual). Las ventajas son las siguientes:

- No es necesario realizar una implementación completa de IBM MQen la máquina cliente.
- Los requisitos de hardware en el sistema cliente se reducen.
- Los requisitos de administración del sistema se reducen.
- Una aplicación de IBM MQ que se ejecuta en un cliente puede conectarse a varios gestores de colas en distintos sistemas.
- Se pueden utilizar canales alternativos que utilicen protocolos de transmisión diferentes.

Soporte de plataforma para clientes de IBM MQ

IBM MQ en todas las plataformas de servidor soportadas acepta conexiones de cliente de IBM MQ MQI clients en un número de plataformas.

IBM MQ instalado como *Producto base y servidor* en todas las plataformas de servidor soportadas puede aceptar conexiones de IBM MQ MQI clients en las siguientes plataformas:

- 

IBM i

- AIX
- Linux
- Windows

Las conexiones de cliente están sujetas a diferencias en el identificador de juego de caracteres codificados (CCSID) y el protocolo de comunicaciones.

¿Qué aplicaciones se ejecutan en un IBM MQ MQI client?

En el entorno cliente, la interfaz MQI completa está soportada. Esto permite que casi cualquier aplicación de IBM MQ se configure para ejecutarse en un sistema IBM MQ MQI client enlazando la aplicación en el IBM MQ MQI client a la biblioteca MQIC, en lugar de hacerlo a la biblioteca MQI. Existen las siguientes excepciones:

- MQGET con señal
- Una aplicación que necesita la coordinación de punto de sincronismo con otros gestores de recursos utilizan un cliente transaccional extendido.

Si se habilita la lectura anticipada, para mejorar el rendimiento de la mensajería no persistente no todas las opciones de MQGET están disponibles. La tabla muestra las opciones que están permitidas y si se pueden alterar entre llamadas MQGET.

Tabla 15. Opciones MQGET permitidas cuando la lectura anticipada está habilitada

Valores	**Permitidas cuando la lectura anticipada está habilitada y se pueden modificar entre llamadas MQGET**	**Permitidas cuando la lectura anticipada está habilitada, pero no se pueden modificar entre llamadas MQGET[1]**	**Opciones MQGET que no están permitidas cuando la lectura anticipada está habilitada[2]**
Valores MD de MQGET	MsgId[3] CorrelId [3]	Encoding CodedCharSetId	
Opciones MQGET MQGMO	MQGMO_WAIT MQGMO_NO_WAIT MQGMO_FAIL_IF_QUIESCING MQGMO_BROWSE_FIRST[4] MQGMO_BROWSE_NEXT[4] MQGMO_BROWSE_MESSAGE_UNDER_CURSOR [4]	MQGMO_SYNCPOINT_IF_PERSISTENT MQGMO_NO_SYNCPOINT MQGMO_ACCEPT_TRUNCATED_MSG MQGMO_CONVERT MQGMO_LOGICAL_ORDER MQGMO_COMPLETE_MSG MQGMO_ALL_MSGS_AVAILABLE MQGMO_ALL_SEGMENTS_AVAILABLE MQGMO_MARK_BROWSE_HANDLE MQGMO_MARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_HANDLE MQGMO_UNMARKED_BROWSE_MSG MQGMO_PROPERTIES_FORCE_MQRFH2 MQGMO_NO_PROPERTIES MQGMO_PROPERTIES_IN_HANDLE MQGMO_PROPERTIES_COMPATIBILITY	MQGMO_SET_SIGNAL MQGMO_SYNCPOINT MQGMO_MARK_SKIP_BACKOUT MQGMO_MSG_UNDER_CURSOR[4] MQGMO_LOCK MQGMO_UNLOCK
Valores MQGMO		MsgHandle	

1. Si estas opciones se modifican entre llamadas MQGET se devuelve el código de razón MQRC_OPTIONS_CHANGED.
2. Si estas opciones se especifican en la primera llamada MQGET, la lectura anticipada está inhabilitada. Si estas opciones se especifican en una llamada MQGET posterior, se devuelve el código de razón MQRC_OPTIONS_ERROR.

3. Las aplicaciones cliente han de tener presente que si los valores MsgId y CorrelId se modifican entre llamadas MQGET, es posible que los mensajes con los valores anteriores ya se hayan enviado al cliente y permanezcan en el almacenamiento intermedio de lectura anticipada del cliente hasta que se consuman (o se depuren automáticamente).
4. La primera llamada MQGET determina si los mensajes se han de examinar u obtener de una cola cuando la lectura anticipada está habilitada. Si la aplicación intenta utilizar una combinación de opciones de examen y de obtención, se devuelve el código de razón MQRC_OPTIONS_CHANGED.
5. MQGMO_MSG_UNDER_CURSOR no es posible con la lectura anticipada. Los mensajes se pueden examinar u obtener cuando la lectura anticipada está habilitada, pero no se puede hacer ambas cosas a la vez.

Una aplicación que se ejecuta en un IBM MQ MQI client se puede conectarse a más de un gestor de colas simultáneamente, o utilizar un nombre de gestor de colas con un asterisco (*) en una llamada MQCONN o MQCONNX (consulte los ejemplos en Conexión de aplicaciones IBM MQ MQI client a gestores de colas).

¿Qué es un cliente transaccional extendido?

Un cliente transaccional extendido de IBM MQ puede actualizar recursos gestionados por otro gestor de recursos, bajo el control de un gestor de transacciones externo.

Si no está familiarizado con los conceptos de gestión de transacciones, consulte “Gestión y soporte de transacciones” en la página 149.

Tenga en cuenta que el cliente transaccional XA ahora se suministra como parte de IBM MQ.

Una aplicación de cliente puede participar en una unidad de trabajo que está gestionada por un gestor de colas al que está conectado. Dentro de la unidad de trabajo, la aplicación de cliente puede transferir mensajes a las colas y obtener mensajes de las colas que son propiedad de ese gestor de colas. La aplicación de cliente puede entonces utilizar la llamada **MQCMIT** para confirmar la unidad de trabajo o la llamada **MQBACK** para restituir la unidad de trabajo. Sin embargo, dentro de la misma unidad de trabajo, la aplicación de cliente no puede actualizar los recursos de otro gestor de recursos, las tablas de una base de datos de Db2, por ejemplo. La utilización de un cliente transaccional extendido de IBM MQ elimina esta restricción.

Un cliente transaccional extendido de IBM MQ es un IBM MQ MQI client con alguna función adicional. Mediante esta función, una aplicación de cliente, dentro de la misma unidad de trabajo puede realizar las tareas siguientes:

- Transferir mensajes a las colas y obtener mensajes de las colas que son propiedad del gestor de colas al que está conectado
- Actualizar los recursos de un gestor de recursos que no sea un gestor de colas de IBM MQ

Esta unidad de trabajo la debe gestionar un gestor de transacciones externo que se ejecuta en el mismo sistema que la aplicación de cliente. La unidad de trabajo no la puede gestionar el gestor de colas al que está conectada la aplicación de cliente. Esto significa que el gestor de colas sólo puede actuar como un gestor de recursos y no como un gestor de transacciones. También indica que la aplicación de cliente puede confirmar o restituir la unidad de trabajo utilizando sólo la interfaz de programación de aplicaciones (API) que proporciona el gestor de transacciones externo. La aplicación de cliente no puede por lo tanto utilizar las llamadas de MQI, **MQBEGIN**, **MQCMIT** y **MQBACK**.

El gestor de transacciones externo se comunica con el gestor de colas como un gestor de recursos utilizando el mismo canal MQI que el que utiliza la aplicación de cliente que está conectada al gestor de colas. Pero en una situación de recuperación tras un error, cuando no hay ninguna aplicación en ejecución, el gestor de transacciones puede utilizar un canal MQI dedicado para recuperar las unidades incompletas de trabajo en las que el gestor de colas estaba participando en el momento de producirse la anomalía.

En esta sección, un IBM MQ MQI client que no tiene la función transaccional extendida se conoce como cliente base de IBM MQ. Por lo tanto, puede considerar que un cliente transaccional extendido de IBM MQ consta de un cliente base de IBM MQ y además incorpora la función transaccional extendida.

Nota: IBM i IBM MQ MQI client en IBM i no da soporte a la función transaccional extendida de IBM MQ.

Soporte de plataforma para clientes transaccionales extendidos

Multi

Los clientes transacciones extendidos están disponibles en todas las Multiplatforms que soportan un cliente base. Los clientes no están disponibles para z/OS.

Una aplicación cliente que utiliza un cliente transaccional extendido sólo puede conectarse a un gestor de colas de los siguientes productos de IBM MQ 9.0o posterior:

- AIX IBM MQ for AIX
- IBM i IBM MQ for IBM i
- Linux IBM MQ para Linux
- Windows IBM MQ for Windows

z/OS Aunque no hay clientes transaccionales extendidos que se ejecutan en z/OS, una aplicación de cliente que utiliza un cliente transaccional extendido se puede conectar a un gestor de colas que se ejecuta en z/OS.

Para cada plataforma, los requisitos de hardware y software para el cliente transaccional extendido son los mismos que los requisitos para el cliente base de IBM MQ. Un lenguaje de programación está soportado por un cliente transaccional extendido si está soportado por el cliente base de IBM MQ y por el gestor de transacciones que se está utilizando.

Para obtener información sobre los gestores de transacciones externas para todas las plataformas, consulte Requisitos del sistema para IBM MQ.

Cómo se conecta el cliente al servidor

Un cliente se conecta a un servidor utilizando MQCONN o MQCONNX y se comunica a través de un canal.

Una aplicación que se ejecuta en el entorno de cliente de IBM MQ debe mantener una conexión activa entre las máquinas cliente y servidor.

La conexión la establece una aplicación al emitir una llamada MQCONN o MQCONNX. Los clientes y los servidores se comunican mediante *canales MQI* o bien, cuando se utiliza el compartición de conversaciones, las conversaciones comparten cada una instancia de canal MQI. Cuando la llamada se realiza satisfactoriamente, la instancia de canal MQI o la conversación permanece conectada hasta que la aplicación emita una llamada MQDISC. Este es el caso de cada gestor de colas al que una aplicación necesita conectarse.

Un cliente y un gestor de colas en la misma máquina

También puede ejecutar una aplicación en el entorno IBM MQ MQI client cuando la máquina también tiene instalado un gestor de colas.

En esta situación, tiene la opción de enlazarse a las bibliotecas del gestor de colas o a las bibliotecas del cliente, pero recuerde que si se enlaza a las bibliotecas del cliente, sigue siendo necesario definir las conexiones de canal. Esto puede ser útil durante la fase de desarrollo de una aplicación. Puede probar el programa en su propia máquina, sin depender de otros, y tener confianza en que seguirá funcionando cuando lo traslade a un entorno IBM MQ MQI client independiente.

Clientes en plataformas diferentes

En este ejemplo, la máquina servidor se comunica con tres IBM MQ MQI clients en distintas plataformas.

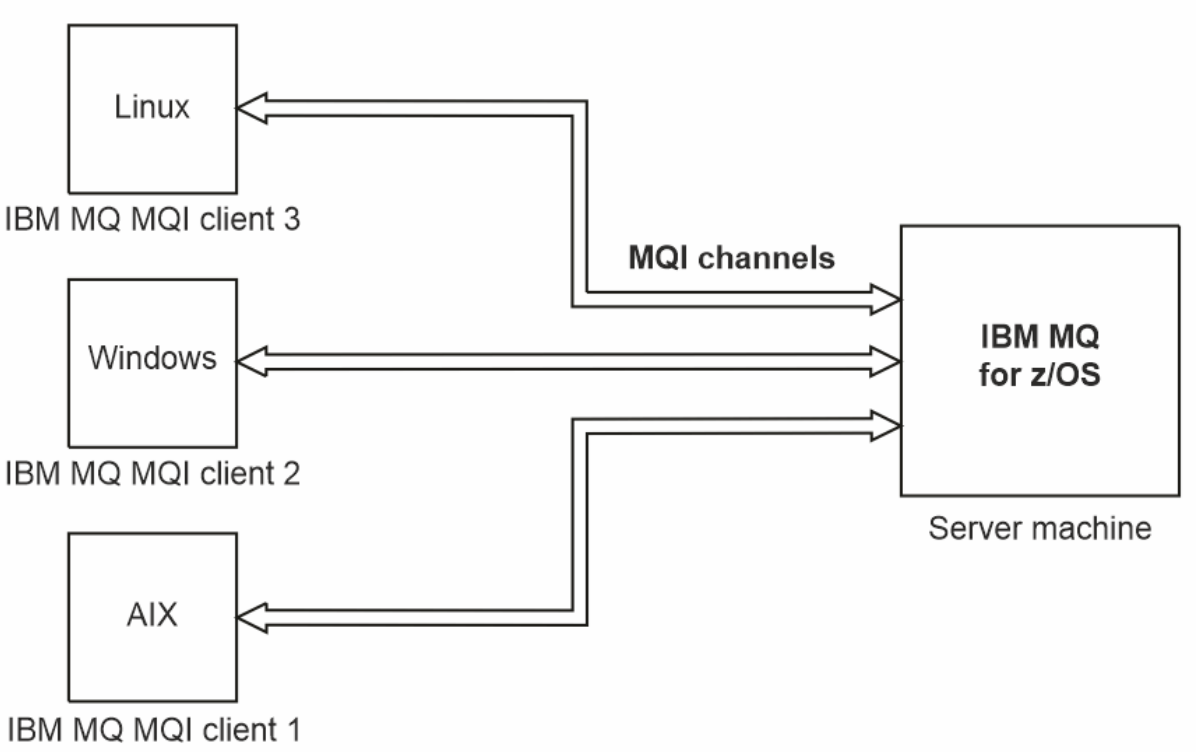


Figura 52. Servidor de IBM MQ conectado a clientes en distintas plataformas

Son posibles otros entornos más complejos. Por ejemplo, un cliente de IBM MQ se puede conectar a más de un gestor de colas o a cualquier número de gestores de colas conectados como parte de un grupo de compartición de colas.

Utilización de versiones diferentes de software de cliente y de servidor

Si utiliza versiones anteriores de productos IBM MQ, asegúrese de que el servidor da soporte a la conversión de código del CCSID del cliente.

Un cliente IBM MQ puede conectarse a todas las versiones soportadas del gestor de colas. Si se está conectando a un gestor de colas de versión anterior, no puede utilizar las características y estructuras de una versión posterior del producto en la aplicación IBM MQ en el cliente.

Un gestor de colas de IBM MQ puede comunicarse con clientes en diferentes versiones consigo mismo negociando hasta el nivel de protocolo más alto soportado mutuamente. Esto significa que los clientes más antiguos se pueden utilizar con niveles de gestor de colas posteriores. Se recomienda que tanto el cliente como el servidor estén en versiones de IBM MQ que estén actualmente en soporte para facilitar el diagnóstico de problemas y habilitar el soporte por parte de IBM.

Para obtener más información, consulte los lenguajes de programación soportados en Desarrollo de aplicaciones.

Gestión y soporte de transacciones

Introducción a la gestión de transacciones y cómo IBM MQ da soporte a las transacciones.

Un *gestor de recursos* es un subsistema informático que posee y gestiona recursos a los que las aplicaciones pueden acceder y que pueden actualizar. A continuación, se muestran algunos ejemplos de gestores de recursos:

- Un gestor de colas de IBM MQ, con recursos que son sus colas
- Una base de datos de Db2, con recursos que son las tablas

Cuando una aplicación actualiza los recursos de uno o varios gestores de recursos, es posible que haya un requisito empresarial para garantizar que determinadas actualizaciones se completan todas satisfactoriamente como un grupo o no se completará ninguna de ellas. El motivo de este tipo de requisito es que los datos empresariales podrían quedarse en un estado incoherente si sólo se completaran correctamente algunas de estas actualizaciones y otras no.

Las actualizaciones de los recursos que se gestionan de este modo se dice que ocurren dentro de una *unidad de trabajo* o una *transacción*. Un programa de aplicación puede agrupar un conjunto de actualizaciones en una unidad de trabajo.

Durante una unidad de trabajo, una aplicación emite peticiones a los gestores de recursos para actualizar sus recursos. La unidad de trabajo finaliza cuando la aplicación emite una petición para confirmar todas las actualizaciones. Hasta que las actualizaciones se confirmen, ninguna de ellas estará visible para las otras aplicaciones que acceden a los mismos recursos. De forma alternativa, si la aplicación decide que no puede completar la unidad de trabajo por algún motivo, puede emitir una petición para restituir todas las actualizaciones que haya solicitado hasta ese momento. En este caso, ninguna de las actualizaciones estará nunca visible para otras aplicaciones. Normalmente, estas actualizaciones están relacionadas lógicamente y deben ejecutarse todas satisfactoriamente para mantener la integridad de los datos. Si una actualización se ejecuta satisfactoriamente y otra no, se pierde la integridad de los datos.

Cuando una unidad de trabajo se ejecuta satisfactoriamente, se dice que se *confirma*. Una vez confirmada, todas las actualizaciones realizadas dentro de esa unidad de trabajo se convierten en persistentes e irreversibles. Sin embargo, si la unidad de trabajo no se ejecuta correctamente, todas las actualizaciones se *restituyen*. Este proceso, en el que las unidades de trabajo se confirman o se restituyen con integridad, se conoce como *coordinación del punto de sincronismo*.

El momento específico en el que todas las actualizaciones en una unidad de trabajo se han confirmado o se han restituido se denomina *punto de sincronismo*. Una actualización en una unidad de trabajo se dice que ocurre *dentro del control de punto de sincronismo*. Si una aplicación solicita una actualización que está *fuera del control de punto de sincronismo*, el gestor de recursos confirma inmediatamente la actualización, aunque haya una unidad de trabajo en curso, y la actualización no se puede restituir más tarde.

El subsistema informático que gestiona unidades de trabajo se denomina *gestión de transacciones* o *coordinador de puntos*.

Una unidad de trabajo de *local* es aquella en la que los únicos recursos actualizados son los del gestor de colas de IBM MQ. En ese caso, la coordinación del punto de sincronismo la proporciona el propio gestor de colas utilizando el proceso de confirmación en una sola fase.

Una unidad de trabajo *global* es aquella en la que los recursos que pertenecen a otros gestores de recursos, como bases de datos conformes a las normas XA, también se actualizan. En este caso, debe utilizarse un procedimiento de confirmación en dos fases y la unidad de trabajo la puede coordinar el propio gestor de colas, o bien externamente otro gestor de transacciones que cumpla las normas XA, como por ejemplo IBM TXSeries o BEA Tuxedo.

Un gestor de transacciones es el responsable de garantizar que todas las actualizaciones de recursos en una unidad de trabajo se completen correctamente o bien ninguna de ellas se completa. Es a un gestor de transacciones a quien la aplicación emite una petición para confirmar o restituir una unidad de trabajo. Entre los ejemplos de gestores de transacciones están CICS y WebSphere Application Server, aunque ambos poseen también otra función.

Algunos gestores de recursos tienen su propia función de gestión de transacciones. Por ejemplo, un gestor de colas de IBM MQ puede gestionar unidades de trabajo que impliquen actualizaciones de sus propios recursos y actualizaciones en las tablas de Db2. El gestor de colas no necesita un gestor de transacciones independiente para realizar esta función, aunque se puede utilizar uno, en caso de que sea requisito del usuario. Si se utiliza un gestor de transacciones independiente, se denominará *gestor de transacciones externo*.

Para que un gestor de transacciones externo gestione una unidad de trabajo, debe haber una interfaz estándar entre el gestor de transacciones y cada gestor de recursos que participa en la unidad de trabajo. Esta interfaz permite que el gestor de transacciones y un gestor de recursos se comuniquen entre sí. Una de estas interfaces es la *Interfaz XA*, que es una interfaz estándar soportada por una serie de gestores de

transacciones y gestores de recursos. La Interfaz XA está publicada por The Open Group en *Distributed Transaction Processing: The XA Specification*.

Cuando más de un gestor de recursos participa en una unidad de trabajo, un gestor de transacciones debe utilizar un protocolo de *confirmación en dos fases* para asegurarse de que todas las actualizaciones dentro de la unidad de trabajo se completan satisfactoriamente o bien ninguna de ellas se completará, aunque haya una anomalía en el sistema. Cuando una aplicación emite una petición a un gestor de transacciones para confirmar una unidad de trabajo, el gestor de transacciones realiza lo siguiente:

Fase 1 (preparar la confirmación)
El gestor de transacciones pregunta a cada gestor de recursos que participa en la unidad de trabajo para asegurarse de que toda la información sobre las actualizaciones previstas de sus recursos esté en un estado recuperable. Normalmente, un gestor de recursos es quien lo hace grabando la información en un registro y asegurándose de que la información se graba a través del disco duro. La Fase 1 se completa cuando el gestor de transacciones recibe notificación de cada gestor de recursos de que la información sobre las actualizaciones previstas de sus recursos se encuentran en un estado recuperable.

Fase 2 (confirmar)
Cuando la Fase 1 se ha completado, el gestor de transacciones toma la decisión irrevocable de confirmar la unidad de trabajo. Solicita a cada gestor de recursos que participa en la unidad de trabajo que confirme las actualizaciones de sus recursos. Cuando un gestor de recursos recibe esta petición, debe confirmar las actualizaciones. No tiene la opción de restituirlas en esta fase. La Fase 2 se completa cuando el gestor de transacciones recibe notificación de cada gestor de recursos de que ha confirmado las actualizaciones de los recursos.

La Interfaz XA utiliza un protocolo de confirmación en dos fases.

Para obtener más información, consulte Escenarios de soporte transaccional.

IBM MQ también proporciona soporte para Microsoft Transaction Server (COM+). Utilización de Microsoft Transaction Server (COM+) proporciona información sobre cómo configurar IBM MQ para aprovechar el soporte de COM+.

Extensión de recursos del gestor de colas

Puede ampliar los recursos del gestor de colas utilizando las salidas de usuario, las salidas de API, o los servicios instalables.

Salidas de usuario

Las salidas de usuario proporcionan un mecanismo para que los usuarios inserten su propio código en una función del gestor de colas. Las salidas de usuario soportadas son:

Salidas de canal
Estas salidas cambian el modo en que funcionan los canales. Las salidas de canal se describen en Programas de salida de canal para canales de mensajes.

Salidas de conversión de datos
Estas salidas crean fragmentos de código fuente que pueden transferirse a programas de aplicación para convertir datos de un formato a otro. Las salidas de conversiones de datos se describen en Escritura de salidas de conversión de datos.

La salida de carga de trabajo del clúster
La función realizada por esta salida la define el proveedor de la salida. Se proporciona información de definición de llamada en MQ_CLUSTER_WORKLOAD_EXIT - Descripción de llamada.

Salidas de API

Las salidas de API permiten escribir código que cambia el comportamiento de las llamadas de API de IBM MQ, como, por ejemplo, MQPUT y MQGET, e insertar ese código inmediatamente antes o inmediatamente después de esas llamadas. La inserción se realiza de forma automática. El gestor de colas dirige el código

de salida a los puntos registrados. Para obtener más información sobre las salidas de la API, consulte Utilización y escritura de salidas de API.

Servicios instalables

Los servicios instalables tienen interfaces formalizadas (una API) con varios puntos de entrada.

Una implementación de un servicio instalable se denomina *componente de servicio*. Puede utilizar los componentes suministrados con IBM MQ o puede escribir su propio componente para ejecutar las funciones que necesita.

Los servicios instalables que se proporcionan actualmente son los siguientes:

Servicio de autorización
El servicio de autorización le permite crear su propio recurso de seguridad.

El componente de servicio predeterminado que implementa el servicio es el gestor de autorizaciones sobre objetos (OAM). De forma predeterminada el OAM está activo, es decir, no tiene que hacer nada para configurarlo. Puede utilizar la interfaz del servicio de autorización para crear otros componentes que sustituyan o aumenten el OAM. Para obtener más información sobre el OAM, consulte Configuración de la seguridad en sistemas AIX, Linux, and Windows.

Servicio de nombres
El servicio de nombres permite a las aplicaciones compartir colas ya que se identifican las colas remotas como si fuesen colas locales.

Puede escribir su propio componente de servicio de nombres. Es posible que desee hacerlo si tiene la intención de utilizar el servicio de nombres con IBM MQ, por ejemplo. Para utilizar el servicio de nombres debe tener un componente escrito por el usuario o un componente proporcionado por un proveedor de software distinto. De forma predeterminada, el servicio de nombres está inactivo.

Conceptos relacionados
Salidas de usuario, salidas de API y servicios instalables de IBM MQ

Interfaces de lenguaje de IBM MQ Java

IBM MQ proporciona tres interfaces de programación de aplicaciones (API) para su uso en aplicaciones Java : IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMSy IBM MQ classes for Java.

IBM soporta, y es un participante activo en, estándares abiertos.

- A partir de IBM MQ 8.0, el producto implementa el estándar JMS 2.0 , que introdujo una nueva API simplificada junto con características como suscripciones compartidas.
- A partir de IBM MQ 9.3.0, también se da soporte a Jakarta Messaging 3.0 .
- Además, WebSphere Liberty da soporte a JMS 2.0 y Jakarta Messaging 3.0 con IBM MQ.

Dentro de IBM MQ hay tres API para su uso en aplicaciones Java :

JM 3.0 **IBM MQ classes for Jakarta Messaging**
IBM MQ classes for Jakarta Messaging es un proveedor de Jakarta Messaging que implementa las interfaces de Jakarta Messaging para IBM MQ como el sistema de mensajería. Jakarta Connectors Architecture proporciona una forma estándar de conectar aplicaciones que se ejecutan en un entorno de Jakarta EE a un sistema de información empresarial (EIS) como, por ejemplo, IBM MQ o Db2.

JMS 2.0 **IBM MQ classes for JMS**
IBM MQ classes for JMS es un proveedor de JMS que implementa las interfaces de JMS para IBM MQ como el sistema de mensajería. Java Platform, Enterprise Edition Connector Architecture (JCA) proporciona una forma estándar de conectar aplicaciones que se ejecutan en un entorno Java EE con un EIS (Enterprise Information System) como IBM MQ o Db2.

IBM MQ classes for Java

IBM MQ classes for Java permite utilizar IBM MQ en un entorno Java. IBM MQ classes for Java permite que una aplicación Java se conecte con IBM MQ como un cliente IBM MQ o se conecte directamente a un gestor de colas de IBM MQ.

Nota:

- JMS 2.0 ha sido reemplazado por Jakarta Messaging. IBM MQ classes for JMS sigue dando soporte al estándar JMS 2.0 , pero las futuras mejoras en la mensajería de Java solo surgirán en Jakarta Messaging, por lo tanto en IBM MQ classes for Jakarta Messaging. Los IBM MQ classes for JMS sólo se recomiendan para mantener y ampliar las aplicaciones JMS 2.0 existentes. IBM MQ classes for Jakarta Messaging debe ser la tecnología preferida para el nuevo desarrollo.
- Stabilized IBM MQ classes for Java se estabilizan funcionalmente en el nivel suministrado en IBM MQ 8.0. Las aplicaciones existentes que utilizan IBM MQ classes for Java seguirán estando totalmente soportadas, pero esta API se estabiliza, por lo que no se añadirán nuevas características y se rechazarán las solicitudes de mejoras. Soporte completo significa que los defectos se solucionarán junto con los cambios necesarios por los cambios en los requisitos del sistema IBM MQ.

JM 3.0 A partir de IBM MQ 9.3, IBM MQ classes for Java, IBM MQ classes for JMSy IBM MQ classes for Jakarta Messaging se crean con Java 8. Los entornos de ejecución de Java en o por encima de estos niveles deben utilizarse para ejecutar aplicaciones utilizando estas interfaces.

Conceptos relacionados

Acceso a IBM MQ desde Java -Opción de API

¿Por qué debo utilizar clases de IBM MQ para Jakarta Messaging?

¿Por qué debo utilizar IBM MQ classes for JMS?

¿Por qué debo utilizar IBM MQ classes for Java?

IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging son los proveedores de mensajería que se proporcionan con IBM MQ. Cada uno de estos proveedores también proporciona dos conjuntos de extensiones a la API de mensajería. Las aplicaciones Java Platform, Standard Edition (Java SE) y Java Platform, Enterprise Edition (Java EE) pueden utilizar estos proveedores de mensajería.

JM 3.0 IBM MQ 9.3.0 ha introducido soporte para Jakarta Messaging 3.0. JMS 2.0 sigue estando totalmente soportado.

Las especificaciones JMS y Jakarta Messaging definen un conjunto de interfaces que las aplicaciones pueden utilizar para realizar operaciones de mensajería. El producto da soporte a la versión JMS 2.0 del estándar JMS . Esta implementación ofrece todas las funcionalidades de la API clásica, pero requiere menos interfaces y es más sencilla de utilizar. Para obtener más información, consulte “Modelo JMS y Jakarta Messaging” en la página 157 y la especificación JMS 2.0 en Java .net. JM 3.0 A partir de IBM MQ 9.3.0, Jakarta Messaging también está soportado.

El paquete jakarta.jms (Jakarta Messaging 3.0) o javax.jms (JMS 2.0) especifica los detalles de las interfaces de mensajería y un proveedor de mensajería implementa estas interfaces para un producto de mensajería específico. Por ejemplo:

- IBM MQ classes for JMS son proveedores de JMS e implementan las interfaces JMS para IBM MQ y también proporcionan los dos conjuntos de extensiones siguiente a la API de JMS:
  - Extensiones de IBM MQJMS
  - Extensiones de IBMJMS
- Una fábrica de conexiones, cola u objeto de tema creado utilizando javax.dimso jakarta.jms, las interfaces o cualquier conjunto de extensiones de JMS se pueden dirigir utilizando cualquiera de estas API; es decir, se pueden convertir a cualquiera de las interfaces. Para mantener la portabilidad de las aplicaciones al máximo nivel, utilice las API más genéricas que se ajusten a sus requisitos.

Debido a que JMS y Jakarta Messaging comparten mucho en común, las referencias adicionales a JMS en este tema se pueden tomar como referencias a ambos. Las diferencias se resaltan según sea necesario.

Extensiones de IBM MQJMS

IBM MQ classes for JMS también proporciona extensiones a la API de JMS. IBM MQ classes for JMS contiene extensiones que se implementan en objetos MQConnectionFactory, MQQueue y MQTopic. Estos objetos tienen propiedades y métodos que son específicos de IBM MQ. Los objetos pueden ser objetos administrados, o una aplicación puede crearlos dinámicamente en tiempo de ejecución. Estas extensiones se denominan extensiones de IBM MQ JMS . Tenga en cuenta que, en esta documentación, los objetos creados dinámicamente por una aplicación en tiempo de ejecución no se consideran objetos administrados.

Extensiones de IBMJMS

Además de las extensiones de IBM MQ JMS , IBM MQ classes for JMS proporciona un conjunto más genérico de extensiones a la API de JMS o Java como el lenguaje de programación utilizado. Estas extensiones se denominan extensiones de IBM JMS y tienen los siguientes objetivos generales:

- Para proporcionar un mayor nivel de coherencia entre los proveedores de IBM JMS .
- Para facilitar la escritura de una aplicación puente entre dos sistemas de mensajería de IBM .
- Para facilitar el puerto de una aplicación de un proveedor de IBM JMS a otro.

El punto central de interés de estas extensiones es crear y configurar fábricas de conexiones y destinos de forma dinámica durante la ejecución, pero las extensiones también proporcionan una función que no está directamente relacionada con la mensajería como, por ejemplo, una función para la determinación de problemas.

Tareas relacionadas

Utilización de clases IBM MQ para JMS/Jakarta Messaging

Configuración de recursos JMS y Jakarta Messaging

JM 3.0 IBM MQ classes for Jakarta Messaging: una visión general

IBM MQ 9.3.0 introduce soporte para Jakarta Messaging. Para Jakarta Messaging 3.0, el control de la especificación JMS se ha movido de Oracle al proceso de comunidad de Java . Sin embargo, Oracle conserva el control del nombre "javax", que se utiliza en otras tecnologías Java . Por lo tanto, aunque Jakarta Messaging 3.0 es funcionalmente equivalente a JMS 2.0, existen algunas diferencias en la denominación. El nombre oficial de la versión 3.0 es Jakarta Messaging en lugar de Java Message Service, y los nombres de paquete y constante tienen el prefijo `jakarta` en lugar de `javax`.

En segundo plano

Durante muchos años, la plataforma Java se ha presentado en dos formatos: Standard Edition y Enterprise Edition.

Java Platform, Standard Edition (a veces abreviado como Java SE) es el lenguaje principal y las bibliotecas de clases, que pueden ejecutarse en un contexto autónomo. La mayoría de los paquetes de Java en Java SE tienen nombres que empiezan por "java.".

Java Platform, Enterprise Edition (Java EE) amplía esto, añadiendo funciones como Mensajería, varios beans, transaccionalidad, etc. Algunas de estas tecnologías también se pueden utilizar en un contexto Java SE . La mayoría de los paquetes de Java en Java EE históricamente tienen nombres que empiezan por "javax."-sin embargo, hay algún cruce, por lo que algunos paquetes de Java SE tienen "javax." como prefijo de su nombre.

Java Message Service (JMS) forma parte de Java Platform, Enterprise Edition. Java EE 7 incorpora JMS 2.0.

Hasta Java EE 7, las tecnologías estaban bajo la dirección de Oracle.

Las tecnologías Java EE han pasado recientemente de la administración de Oracle a un proceso comunitario supervisado por la Fundación Eclipse .

Como el "javax". no se ha podido mover el nombre al nuevo proyecto, se ha adoptado un nuevo nombre- todos los paquetes y nombres de propiedad tienen ahora el prefijo "jakarta". y Java Platform, Enterprise Edition se denominará "Jakarta EE" en el futuro. La numeración de la versión ha continuado: la versión 8 era una versión provisional que se puede ignorar en gran medida, y Jakarta EE 9 es el punto en el que el "jakarta". el prefijo entra en vigor.

La tecnología Jakarta EE principal que se aplica en el contexto IBM MQ es Jakarta Messaging 3.0 -el sucesor de Java Message Service (JMS) 2.0. Por lo tanto, Jakarta EE 9 incorpora Jakarta Messaging 3.0.

IBM MQ sigue dando soporte a Java EE 7 y JMS 2.0, al tiempo que introduce soporte para Jakarta EE 9 y Jakarta Messaging 3.0.

Qué se entrega: Java SE

Para Java Platform, Standard Edition, además del IBM MQ classes for JMS (que da soporte a las operaciones de JMS 2.0 con IBM MQ) IBM MQ 9.3.0 y versiones posteriores proporcionan IBM MQ classes for Jakarta Messaging. Estas clases proporcionan un proveedor de Jakarta Messaging 3.0 que se integra con IBM MQ, permitiendo el uso de gestores de colas de IBM MQ para facilitar las operaciones de Jakarta Messaging .

Estos se proporcionan como un archivo JAR estándar, `com.ibm.mq.jakarta.client.jar`, en el subdirectorio `java/lib` de la instalación de IBM MQ .

Para su uso en contenedores OSGi, como Apache Felix o Eclipse Equinox, IBM MQ también proporciona un par de paquetes OSGi:

- `com.ibm.mq.osgi.jms30.clientprereqs_`*`V.R.M.F`*`.jar`
- `com.ibm.mq.osgi.jms30.client_`*`V.R.M.F`*`.jar`

donde *V.R.M.F* representa la versión de IBM MQ, por ejemplo 9.3.0.0. Estos paquetes se pueden encontrar en el subdirectorio `java/lib/OSGi` de la instalación de IBM MQ .

Qué se entrega: Jakarta EE 9

Para dar soporte a la mensajería basada en IBM MQen un servidor de aplicaciones compatible con Jakarta EE 9 , IBM MQ proporciona un adaptador de recursos compatible con Jakarta EE 9: `wmq.jakarta.jmsra.rar`. Esto se puede encontrar en el subdirectorio `java/lib/jca` de la instalación de IBM MQ .

IBM MQ continúa proporcionando un adaptador de recursos compatible con Java EE 7 , `wmq.jmsra.rar`, en el subdirectorio `java/lib/jca` de la instalación de IBM MQ .

Cómo se entregan estos artefactos

Estos JAR y el archivo RAR para el adaptador de recursos se empaquetan con los artefactos preexistentes en el soporte de instalación habitual de IBM MQ , tanto el soporte de instalación específico de la plataforma, como los archivos "`.rpm`", como el soporte redistribuible, como los archivos JAR de cliente redistribuible autoextraíbles.

Qué ha cambiado entre JMS 2.0 y Jakarta Messaging 3.0

Jakarta EE 9 y Jakarta Messaging 3.0 no presentan ninguna funcionalidad nueva. Todo lo que cambia son nombres. Por ejemplo, si utiliza "javax.jms.Connection" en JMS 2.0, utilice "jakarta.jms.Connection" en Jakarta Messaging 3.0.

A medida que Eclipse Foundation reenvíe la plataforma Jakarta EE , se construirá sobre esta base y este convenio de denominación se utilizará para la nueva funcionalidad introducida en el futuro.

Qué ha cambiado entre IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging

Resumen

IBM MQ classes for JMS, que proporcionan soporte para JMS 2.0, permanecen disponibles y se recomiendan principalmente para mantener y ampliar las aplicaciones existentes. Están totalmente soportados.

IBM MQ classes for Jakarta Messaging, que proporcionan soporte para Jakarta Messaging 3.0, se recomiendan para el nuevo desarrollo.

En IBM MQ 9.3.0, estas dos ofertas eran funcionalmente equivalentes. Sólo la denominación difiere. Sin embargo, es más probable que surjan nuevas funciones de mensajería en IBM MQ classes for Jakarta Messaging que en IBM MQ classes for JMS.

Las dos ofertas son interoperables. Los mensajes producidos por IBM MQ classes for JMS pueden ser consumidos por IBM MQ classes for Jakarta Messagingy viceversa. Pero las dos ofertas no deben coexistir en una sola aplicación.

Cambios en la nomenclatura

Tabla 16. Cambios en los nombres de paquete

IBM MQ classes for JMS nombre de paquete	IBM MQ classes for Jakarta Messaging nombre de paquete
com.ibm.mq.jms[.*]	com.ibm.mq.jakarta.jms[.*]
com.ibm.jms	com.ibm.jakarta.jms
com.ibm.msg.client.jms.*	com.ibm.msg.client.jakarta.jms.*
com.ibm.msg.client.wmq.*	com.ibm.msg.client.jakarta.wmq.*

Los paquetes relacionados con servicios comunes (rastreo, registro, soporte de idioma nacional, etc.) y las implementaciones de JMQI (local y remota) son comunes a IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging, por lo que no es necesario realizar cambios en estas áreas.

Tenga en cuenta que los nombres de propiedad también han cambiado. Por ejemplo, la propiedad para habilitar las extensiones IBM MQ en IBM MQ classes for Jakarta Messaging es **`com.ibm.mq.jakarta.jms.SupportMQExtensions`**.

Los nombres de propiedad que son independientes de IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging, como las distintas propiedades de **`com.ibm.msg.client.commonservices.trace.*`** , se aplican por igual a ambas ofertas.

Programas de utilidad administrativos

Los programas de utilidad **`crtmqenv`** y **`setmqenv`** ahora aceptan una opción para especificar si la vía de acceso de clases debe configurarse para IBM MQ classes for JMS (`-j 2.0`) o IBM MQ classes for Jakarta Messaging (`-j 3.0`), y hay variantes IBM MQ classes for Jakarta Messaging de los programas de utilidad **`runjms`** , denominadas **`runjms30`** y nombres similares.

El programa de utilidad **`dspmqver`** , cuando se le solicite que informe sobre los componentes de Java , incluye IBM MQ classes for Jakarta Messaging en su salida.

Para configurar objetos IBM MQ classes for Jakarta Messaging para que se recuperen a través de JNDI, el nuevo programa de utilidad **`JMS30Admin`** es equivalente al programa de utilidad **`JMSAdmin`** para IBM MQ classes for JMS.

Tenga en cuenta que como los objetos subyacentes son de paquetes diferentes. Las definiciones JNDI creadas por **`JMSAdmin`** no pueden ser utilizadas por IBM MQ classes for Jakarta Messaging, ni tampoco las creadas por **`JMS30Admin`** pueden ser utilizadas por IBM MQ classes for JMS.

Nota: No hay soporte para objetos IBM MQ classes for Jakarta Messaging proporcionados por IBM MQ Explorer; su integración JNDI es solo para IBM MQ classes for JMS .

Conceptos relacionados

¿Por qué debo utilizar clases de IBM MQ para Jakarta Messaging?

Modelo JMS y Jakarta Messaging

El modelo JMS y Jakarta Messaging define un conjunto de interfaces que las aplicaciones Java pueden utilizar para realizar operaciones de mensajería. IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging son proveedores de mensajería que definen cómo están relacionados los objetos de mensajería de Java con los conceptos de IBM MQ . Las especificaciones JMS y Jakarta Messaging esperan que determinados objetos de mensajería sean objetos administrados.

A partir de IBM MQ 8.0, el producto da soporte a la versión JMS 2.0 del estándar JMS, que introdujo una API simplificada, al mismo tiempo que retiene la API clásica, de JMS 1.1.

JM 3.0 IBM MQ 9.3.0 ha introducido soporte para Jakarta Messaging 3.0. JMS 2.0 sigue estando totalmente soportado. Debido a que JMS y Jakarta Messaging comparten mucho en común, las referencias adicionales a JMS en este tema se pueden tomar como referencias a ambos. Las diferencias se resaltan según sea necesario.

API simplificada

JMS 2.0 ha introducido la API simplificada, al tiempo que retiene las interfaces específicas del dominio e independientes del dominio de JMS 1.1. La API simplificada disminuye el número de objetos necesarios para enviar y recibir mensajes y consta de las interfaces siguientes:

ConnectionFactory
ConnectionFactory es un objeto administrado que utiliza un cliente de JMS para crear una conexión. Esta interfaz también se utiliza en la API clásica.

Contexto de JMS
Este objeto combina los objetos Conexión y Sesión de la API clásica. Se pueden crear objetos JMSContext a partir de otros objetos JMSContext, duplicando la conexión subyacente.

JMSProductor
Se crea un JMSProducer mediante un JMSContext y se utiliza para enviar mensajes a una cola o tema. El objeto JMSProducer genera la creación de los objetos necesarios para enviar el mensaje.

JMSConsumidor
Se crea un JMSConsume mediante un JMSContext y se utiliza para recibir mensajes de un tema o una cola.

La API simplificada tiene varios efectos:

- El objeto JMSContext siempre inicia automáticamente la conexión subyacente.
- Ahora los JMSProducers y los JMSConsumers pueden trabajar directamente con los cuerpos de los mensajes, sin tener que obtener el objeto de mensaje completo, utilizando el método getBody del mensaje.
- Las propiedades de los mensajes se pueden establecer en el objeto JMSProducer mediante el encadenamiento de métodos, antes de enviar un 'body', un contenido de mensajes. El objeto JMSProducer manejará la creación de todos los objetos necesarios para enviar el mensaje. Utilizando JMS 2.0, se pueden establecer las propiedades y un envío de mensajes, como se indica a continuación:

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

JMS 2.0 también introdujo suscripciones compartidas donde los mensajes se pueden compartir entre varios consumidores. Todas las suscripciones de JMS 1.1 se tratan como suscripciones no compartidas.

API clásica

La lista siguiente resume las interfaces principales de JMS de la API clásica:

Destino
Un objeto Destination es la ubicación a la que una aplicación envía mensajes, o es el origen desde el que una aplicación recibe mensajes, o ambas cosas.

ConnectionFactory
Un objeto ConnectionFactory encapsula un conjunto de propiedades de configuración de una conexión. Una aplicación utiliza una fábrica de conexiones para crear una conexión.

Conexión
Un objeto Connection encapsula una conexión activa de una aplicación en un servidor de mensajería. Una aplicación utiliza una conexión para crear sesiones.

Session
Una sesión es un contexto de hebra única para enviar y recibir mensajes. Una aplicación utiliza una sesión para crear mensajes, productores de mensajes y consumidores de mensajes. Una sesión es transaccional o no transaccional.

Mensaje
Un objeto Message encapsula un mensaje que una aplicación envía o recibe.

MessageProducer
Una aplicación utiliza un productor de mensajes para enviar mensajes a un destino.

MessageConsumer
Una aplicación utiliza un consumidor de mensajes para recibir mensajes enviados a un destino.

Figura 53 en la página 158 muestras estos objetos y sus relaciones.

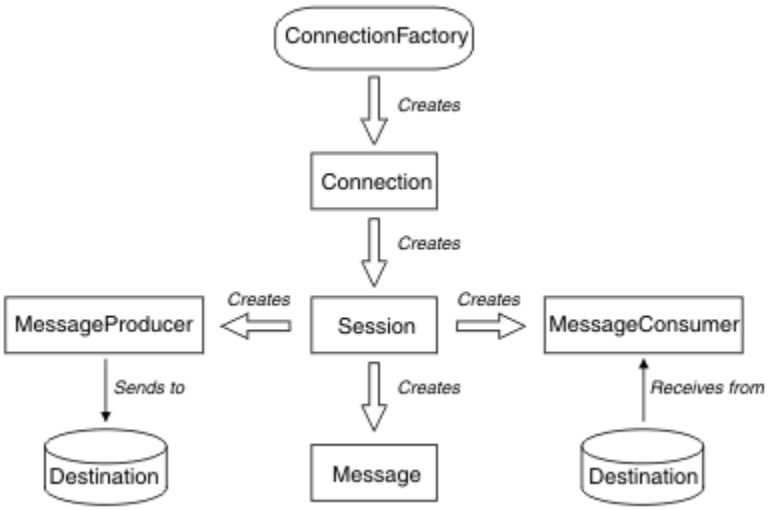


Figura 53. Objetos JMS y sus relaciones

El diagrama muestra las interfaces principales: ConnectionFactory, Connection, Session, MessageProducer, MessageConsumer, Message y Destination. Una aplicación utiliza una fábrica de conexiones para crear una conexión y utiliza una conexión para crear sesiones. La aplicación puede utilizar una sesión para crear mensajes, productores de mensajes y consumidores de mensajes. La aplicación utiliza un productor de mensajes para enviar mensajes a un destino, y utiliza un consumidor de mensajes para recibir mensajes enviados a un destino.

Un objeto Destination, ConnectionFactory o Connection lo pueden utilizar varias hebras al mismo tiempo, pero no pueden utilizar al mismo tiempo un objeto Session, MessageProducer o MessageConsumer. La

forma más simple de asegurarse de que un objeto Session, MessageProducer o MessageConsumer no se utiliza simultáneamente es crear un objeto Session separado para cada hebra.

JMS da soporte a dos tipos de mensajería:

- Mensajería punto a punto
- Mensajería de publicación/suscripción

También se hace referencia a estos tipos de mensajería como *dominios de mensajería* y puede combinar ambos tipos de mensajería en una aplicación. En el dominio punto a punto, un destino es una cola y, en el dominio de publicación/suscripción, un destino es un tema.

En las versiones de JMS anteriores a JMS 1.1, la programación para los dominios punto a punto utiliza un conjunto de interfaces y métodos y la programación para los dominios de publicación/suscripción utiliza otro conjunto. Los dos conjuntos son similares, pero independientes. A partir de JMS 1.1, puede utilizar un conjunto común de interfaces y métodos que dan soporte a ambos dominios de mensajería. Las interfaces comunes proporcionan una vista independiente del dominio para cada dominio de mensajería. La Tabla 17 en la página 159 muestra las interfaces independientes del dominio de JMS y sus interfaces específicas del dominio correspondientes.

Tabla 17. Las interfaces independientes del dominio de JMS y sus interfaces específicas del dominio correspondientes.

Interfaces independientes del dominio	**Interfaces específicas del dominio para el dominio punto a punto**	**Interfaces específicas del dominio para el dominio de publicación/suscripción**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Conexión	QueueConnection	TopicConnection
Destino	Cola	Tema
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 IBM MQ classes for JMS 2.0 da soporte a las interfaces específicas de dominio de JMS 1.1 anteriores y a la API simplificada de JMS 2.0. Por lo tanto, IBM MQ classes for JMS 2.0 se puede utilizar para mantener aplicaciones existentes, incluido el desarrollo de nuevas funciones en aplicaciones existentes.

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 da soporte a las versiones de Jakarta Messaging de las mismas interfaces y se recomienda para el desarrollo de nuevas aplicaciones.

En IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging, los objetos JMS están relacionados con los conceptos de IBM MQ de las maneras siguientes:

- Un objeto Connection tiene propiedades derivadas de las propiedades de la fábrica de conexiones utilizada para crear la conexión. Estas propiedades controlar cómo se conecta una aplicación a un gestor de colas. Los ejemplos de estas propiedades son el nombre del gestor de colas y, en el caso de una aplicación que se conecta al gestor de colas en modo cliente, el nombre de host o dirección IP del sistema en el que se ejecuta el gestor de colas.
- Un objeto Session encapsula un identificador de conexión IBM MQ, que por lo tanto define el ámbito transaccional de la sesión.
- Un objeto MessageProducer y un objeto MessageConsumer encapsula cada uno un manejador de objetos de IBM MQ.

Cuando se utiliza IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging, se aplican todas las reglas normales de IBM MQ . En concreto, tenga en cuenta que una aplicación puede enviar un mensaje a una cola remota pero solo puede recibir un mensaje de una cola propiedad del gestor de colas al que está conectada la aplicación.

La especificación JMS espera que los objetos ConnectionFactory y Destination sean objetos administrados. Un administrador crea y mantiene los objetos administrados en un repositorio central y una aplicación JMS recupera estos objetos utilizando la interfaz JNDI (Naming and Directory Interface) de Java.

En IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging, la implementación de la interfaz Destination es una superclase abstracta de Queue y Topic, por lo que una instancia de Destination es un objeto Queue o un objeto Topic. La interfaz independiente del dominio trata una cola o un tema como un destino. El dominio de mensajería para un objeto MessageProducer o MessageConsumer queda determinado por si el destino es una cola o un tema.

Por lo tanto, en IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging , los objetos de los tipos siguientes pueden ser objetos administrados:

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- Cola
- Tema
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

Arquitectura de IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging tienen una arquitectura en capas. La capa superior de código es una capa común que cualquier proveedor de mensajería de IBM Java puede utilizar.

JM 3.0 IBM MQ 9.3.0 ha introducido soporte para Jakarta Messaging 3.0. JMS 2.0 sigue estando totalmente soportado.

IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging tienen una arquitectura en capas tal como se muestra en el diagrama Figura 54 en la página 161. La capa superior de código es una capa común que puede utilizar cualquier proveedor de IBM JMS o Jakarta Messaging. Cuando una aplicación llama a un método JMS o Jakarta Messaging, cualquier proceso de la llamada que no sea específico de un sistema de mensajería lo realiza la capa común, que también proporciona una respuesta coherente a la llamada. Todo procesamiento de la llamada que sea específico de un sistema de mensajería se delega en una capa inferior. En el diagrama siguiente, el proveedor de mensajería de IBM MQ se muestra en la capa inferior, junto con otros dos proveedores de mensajería (proveedor de mensajería A y proveedor de mensajería B.)

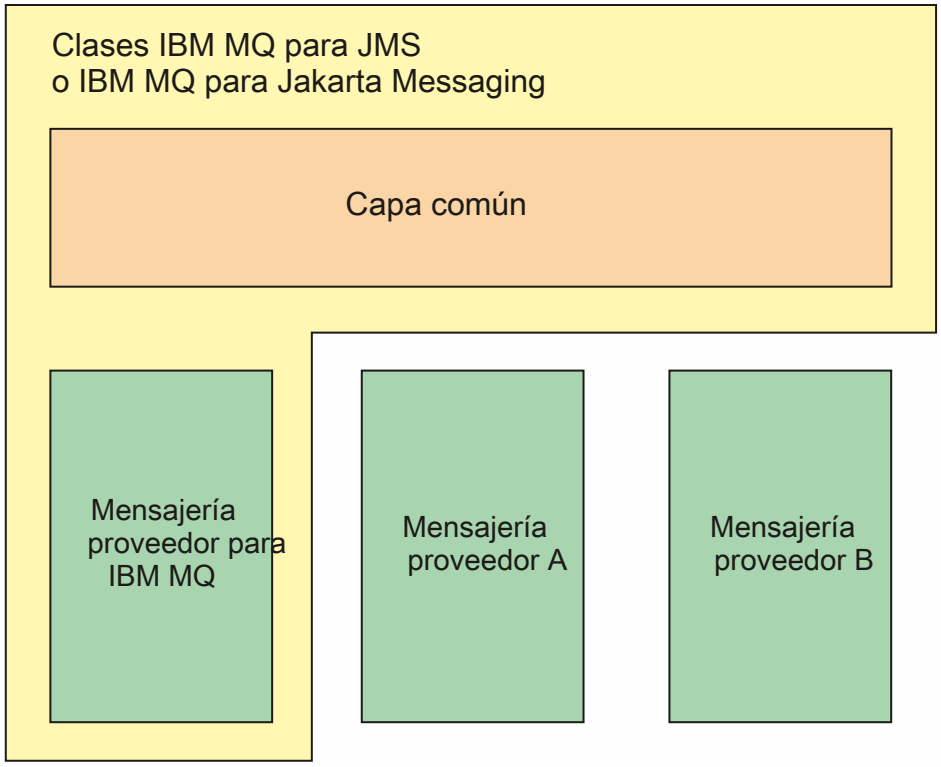


Figura 54. La arquitectura en capas para los proveedores de IBM JMS y Jakarta Messaging

Una arquitectura en capas cumple los siguientes objetivos:

- Para mejorar la coherencia del comportamiento de los diversos proveedores de IBM JMS y Jakarta Messaging
- Facilitar la escritura de una aplicación puente entre dos sistemas de mensajería de IBM
- Para facilitar el puerto de una aplicación de un proveedor de IBM JMS o Jakarta Messaging a otro

Tareas relacionadas

Utilización de clases IBM MQ para JMS/Jakarta Messaging

Soporte para objetos administrados

IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging dan soporte al uso de objetos administrados.

JM 3.0 A partir de IBM MQ 9.3.0, Jakarta Messaging 3.0 está soportado para desarrollar nuevas aplicaciones. IBM MQ 9.3.0 y posteriores siguen dando soporte a JMS 2.0 para las aplicaciones existentes. No está soportado utilizar tanto la API de Jakarta Messaging 3.0 como la API de JMS 2.0 en la misma aplicación. Para obtener más información, consulte Utilización de clases de IBM MQ para JMS/Jakarta Messaging.

El flujo de lógica dentro de una aplicación JMS o IBM MQ classes for Jakarta Messaging se inicia con ConnectionFactory y objetos Destination. La aplicación utiliza un objeto ConnectionFactory para crear un objeto Connection, que representa la conexión activa desde la aplicación hasta un servidor de mensajería. La aplicación utiliza el objeto Connection para crear un objeto Session, que es un contexto de hebra única para producir y consumir mensajes. A continuación, la aplicación puede utilizar el objeto Session y un objeto Destination para crear un objeto MessageProducer, que la aplicación utiliza para enviar mensajes al destino especificado. El destino es una cola o un tema del sistema de mensajería y está encapsulado por el objeto Destination. La aplicación también puede utilizar el objeto Session y un objeto

Destination para crear un objeto MessageConsumer, que la aplicación utiliza para recibir mensajes que se han enviado al destino especificado.

Las especificaciones JMS y Jakarta Messaging esperan que los objetos ConnectionFactory y Destination sean objetos administrados. Un administrador crea y mantiene objetos administrados en un repositorio central, y una aplicación JMS o Jakarta Messaging recupera estos objetos utilizando Java Naming Directory Interface (JNDI). El repositorio de objetos administrados puede ir desde un archivo simple a un directorio LDAP (Lightweight Directory Access Protocol).

IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging dan soporte al uso de objetos administrados. Una aplicación puede utilizar todas las características de IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging que se exponen a través de IBM MQ sin tener ninguna información específica de IBM MQcodificada en la propia aplicación. Este acuerdo proporciona a la aplicación un grado de independencia de la configuración de IBM MQ subyacente.

Para lograr esta independencia, la aplicación puede utilizar JNDI para recuperar fábricas de conexiones y destinos que se almacenan como objetos administrados, y utilizar sólo las interfaces definidas en el paquete `javax.jms` (JMS 2.0) o `jakarta.jms` (Jakarta Messaging 3.0) para realizar operaciones de mensajería.

JMS 2.0 Para JMS 2.0, un administrador puede utilizar la herramienta de administración de IBM MQ JMS **JMSAdmin**, o IBM MQ Explorer, para crear y mantener objetos administrados en un repositorio central.

JM 3.0 Para Jakarta Messaging 3.0, no puede administrar JNDI utilizando IBM MQ Explorer. La administración JNDI está soportada por la variante Jakarta Messaging 3.0 de **JMSAdmin**, que es **JMS30Admin**.

Normalmente, un servidor de aplicaciones proporciona su propio repositorio para objetos administrados y sus propias herramientas para crear y mantener los objetos. Por lo tanto, una aplicación Java EE JM 3.0 o Jakarta EE puede utilizar JNDI para recuperar objetos administrados del repositorio del servidor de aplicaciones o de un repositorio central.

Tareas relacionadas

Configuración de recursos JMS y Jakarta Messaging

Tipos de comunicación soportados en plataformas Java EE y Jakarta EE

En las plataformas Java EE y Jakarta EE , IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging dan soporte a dos tipos de comunicación entre un componente de una aplicación y un gestor de colas IBM MQ .

JM 3.0 IBM MQ 9.3.0 ha introducido soporte para Jakarta Messaging 3.0. JMS 2.0 sigue estando totalmente soportado. Debido a que JMS y Jakarta Messaging comparten mucho en común, las referencias adicionales a JMS en este tema se pueden tomar como referencias a ambos. Las diferencias se resaltan según sea necesario.

Se da soporte a los dos tipos de comunicación siguientes entre un componente de una aplicación y un gestor de colas de IBM MQ:

- Comunicación de salida
- Comunicación de entrada

Comunicación de salida

Utilizando la API JMS o Jakarta Messaging directamente, un componente de aplicación crea una conexión con un gestor de colas y, a continuación, envía y recibe mensajes.

Por ejemplo, el componente de la aplicación puede ser un cliente de aplicación, un servlet, una ASP (página de servidor activo) Java, un Enterprise Java Bean (EJB) o un bean controlado por mensaje (MDB). En este tipo de comunicación, el contenedor del servidor de aplicaciones solo proporciona funciones

de bajo nivel como soporte de las operaciones de mensajería como, por ejemplo, agrupaciones de conexiones y gestión de hilos.

Comunicación de entrada

En el caso de la comunicación de entrada, un mensaje que llega a un destino se entrega a un MDB, que a su vez procesa el mensaje.

Las aplicaciones Java EE JM 3.0 y Jakarta EE utilizan MDB para procesar mensajes de forma asíncrona. Un MDB actúa como escucha de mensajes JMS y se implementa con un método onMessage(), que define cómo se procesa un mensaje. Un MDB se despliega en el contenedor EJB de un servidor de aplicaciones. La forma precisa en que se configura un MDB depende del servidor de aplicaciones que se utilice, pero la información de configuración debe especificar a qué gestor de colas debe conectarse, cómo conectarse al mismo, qué destino debe supervisar mensajes y el comportamiento transaccional del MDB. El contenedor EJB utilizará esta información. Cuando un mensaje que cumple los criterios de selección del MDB llega al destino especificado, el contenedor EJB utiliza IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging para recuperar el mensaje del gestor de colas y, a continuación, entrega el mensaje al MDB llamando a su método onMessage().

Relación con las IBM MQ classes for Java

IBM MQ classes for Java, IBM MQ classes for Jakarta Messaging y IBM MQ classes for JMS son iguales que utilizan una interfaz Java común a la MQI.

La Figura 55 en la página 163 muestra la relación entre IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging y IBM MQ classes for Java.

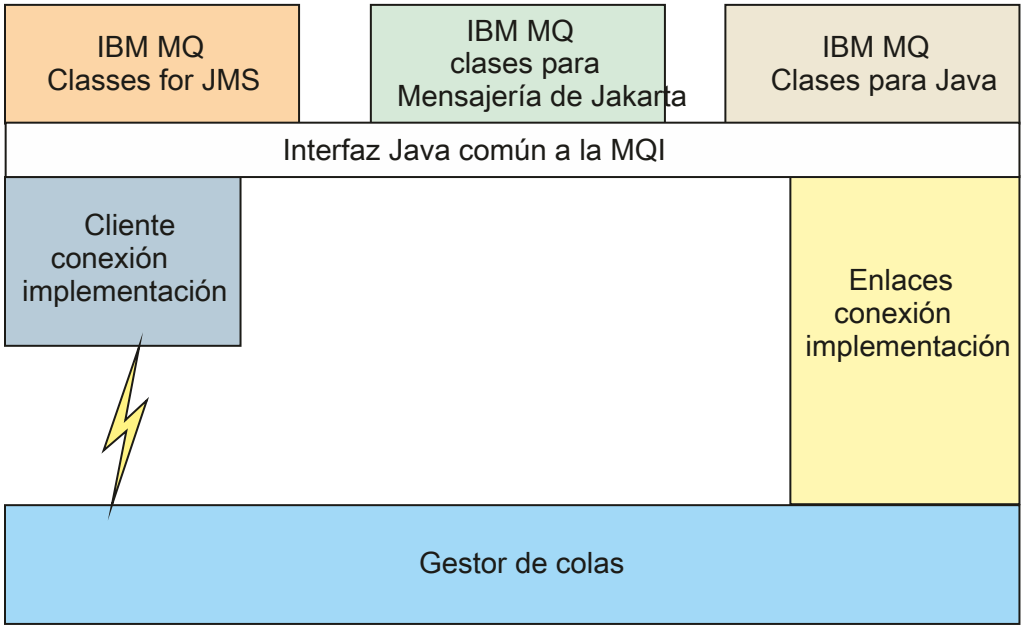


Figura 55. La relación entre IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging y IBM MQ classes for Java

En general, los programas Java sólo deben utilizar una interfaz para interactuar con IBM MQ - IBM MQ classes for Java, IBM MQ classes for Jakarta Messaging o IBM MQ classes for JMS. La combinación de interfaces no está soportada, con una excepción. Para mantener la compatibilidad con los releases anteriores a IBM WebSphere MQ 7.0, las clases de salida de canal escritas en Java todavía pueden utilizar las interfaces IBM MQ classes for Java, incluso si las clases de salida de canal se llaman desde las IBM MQ classes for JMS. Sin embargo, el uso de las interfaces IBM MQ classes for Java significa que las aplicaciones siguen dependiendo de:

- JMS 2.0 El archivo JAR de IBM MQ classes for Java , `com.ibm.mq.jar`. Si no desea `com.ibm.mq.jar` en la vía de acceso de clases, puede utilizar el conjunto de interfaces del paquete `com.ibm.mq.exits`.
- JM 3.0 Uso de `com.ibm.mq.jakarta.client.jar`, al interoperar con IBM MQ classes for Jakarta Messaging.

Conceptos relacionados

¿Por qué debo utilizar clases de IBM MQ para Jakarta Messaging?

¿Por qué debo utilizar las clases IBM MQ para JMS?

¿Por qué debo utilizar las clases de IBM MQ para Java?

Proveedor de mensajería de IBM MQ

El proveedor de mensajería de IBM MQ tiene tres modalidades de operación: modalidad normal, modalidad normal con restricciones y modalidad de migración.

El proveedor de mensajería de IBM MQ tiene tres modalidades de funcionamiento:

- Modalidad normal del proveedor de mensajería de IBM MQ
- Modalidad normal con restricciones del proveedor de mensajería de IBM MQ
- Modalidad de migración del proveedor de mensajería de IBM MQ

La modalidad normal del proveedor de mensajería de IBM MQ utiliza todas las características de un gestor de colas de IBM MQ para implementar JMS. Esta modalidad se optimiza para utilizar la API y la funcionalidad de JMS 2.0 JM 3.0 o Jakarta Messaging 3.0 .

If:

- El cliente especifica una versión de proveedor de 6 en un **`ConnectionFactory`**, el cliente se comporta de una forma compatible con el cliente proporcionado con IBM WebSphere MQ 6.0. Solo se da soporte a las interfaces JMS 1.1 y JMS 2, pero algunas funciones de JMS 2, como suscripciones compartidas, retardo de entrega y envío asíncrono, están inhabilitadas. No hay compartimiento de conexión.
- El cliente especifica una versión de proveedor de 7 en un **`ConnectionFactory`**, tanto las interfaces JMS 1.1 como JMS 2 están totalmente soportadas.
- No se ha especificado ninguna versión de proveedor, se ha intentado conectar con el proveedor versión 7. Si esto falla, se realiza un intento adicional con la versión 6 del proveedor.

Si desea conectarse a IBM Integration Bus utilizando IBM MQ Enterprise Transport, utilice la modalidad de migración. Si utiliza IBM MQ Real-Time Transport, la modalidad de migración se selecciona automáticamente porque se han seleccionado de manera explícita en el objeto de fábrica de conexiones. La conexión con IBM Integration Bus utilizando IBM MQ Enterprise Transport sigue las reglas generales para la selección de modalidad que se describen en Configuración de la propiedad JMS **`PROVIDERVERSION`**.

Tareas relacionadas

Configurar recursos de JMS

z/OS IBM MQ for z/OS concepts

Some of the concepts used by IBM MQ for z/OS are unique to the z/OS platform. For example, the logging mechanism, the storage management techniques, unit of recovery disposition, and queue sharing groups are provided only with IBM MQ for z/OS. Use this topic as an introduction to further information about these concepts.

The concepts include an overview of the objects that IBM MQ for z/OSuses, including:

- The queue manager
- The channel initiator

- Shared queues and queue sharing groups
- Intra-group queuing

The following topics also cover various procedures you need, including:

- System definitions on z/OS
- Storage management
- Recovery and restart
- Security concepts in IBM MQ for z/OS

Related concepts

"The queue manager on z/OS" on page 166
Before you can let your application programs use IBM MQ on your z/OS system, you must install the IBM MQ for z/OS product and start a queue manager. The queue manager owns and manages the set of resources that are used by IBM MQ.

"The channel initiator on z/OS" on page 167
The channel initiator provides and manages resources that enable IBM MQ distributed queuing. IBM MQ uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

"Terms and tasks for managing IBM MQ for z/OS" on page 168
Use this topic as an introduction to the terminology, and tasks that are specific to IBM MQ for z/OS.

"Shared queues and queue sharing groups" on page 171
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.

"Intra-group queuing" on page 215
This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

"Storage management on z/OS" on page 228
IBM MQ for z/OS requires permanent and temporary data structures and uses page sets and memory buffers to store this data. These topics give more details on how IBM MQ uses these page sets and buffers.

"Logging in IBM MQ for z/OS" on page 232
IBM MQ maintains *logs* of data changes and significant events as they occur. These logs can be used to recover data to a previous state if required.

"Recovery and restart on z/OS" on page 253
Use the links in this topic to find out about the features of IBM MQ for z/OS for restart and recovery.

"Security concepts in IBM MQ for z/OS" on page 269
Use this topic to understand the importance of security for IBM MQ, and the implications of not having adequate security settings on your system.

"Availability on z/OS" on page 275
IBM MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

"Unit of recovery disposition on z/OS" on page 280
Certain transactional applications can use a GROUP, rather than a QMGR, unit of recovery disposition when connected to a queue manager in a queue sharing group (QSG) by specifying the QSG name when they connect instead of the queue manager name. This allows transaction recovery to be more flexible and robust by removing the requirement to reconnect to the same queue manager in the QSG.

Related reference

"System definition on z/OS" on page 243
IBM MQ for z/OS uses many default object definitions, and provides sample JCL to create those default objects. Use this topic to understand these default objects, and the sample JCL.

"Monitoring and statistics on IBM MQ for z/OS" on page 279

IBM MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

z/OS The queue manager on z/OS

Before you can let your application programs use IBM MQ on your z/OS system, you must install the IBM MQ for z/OS product and start a queue manager. The queue manager owns and manages the set of resources that are used by IBM MQ.

The queue manager

A *queue manager* is a program that provides messaging services to applications. Applications that use the Message Queue Interface (MQI) can put messages on queues and get messages from queues. The queue manager ensures that messages are sent to the correct queue or are routed to another queue manager. The queue manager processes both the MQI calls that are issued to it, and the commands that are submitted to it (from whatever source). The queue manager generates the appropriate completion codes for each call or command.

The resources managed by the queue manager include:

- Page sets that hold the IBM MQ object definitions and message data
- Logs that are used to recover messages and objects in the event of queue manager failure
- Processor storage
- Connections through which different application environments (CICS, IMS, and Batch) can access the IBM MQ API
- The IBM MQ channel initiator, which allows communication between IBM MQ on your z/OS system and other systems

The queue manager has a name, and applications can connect to it using this name.

Figure 56 on page 166 illustrates a queue manager, showing connections to different application environments, and the channel initiator.

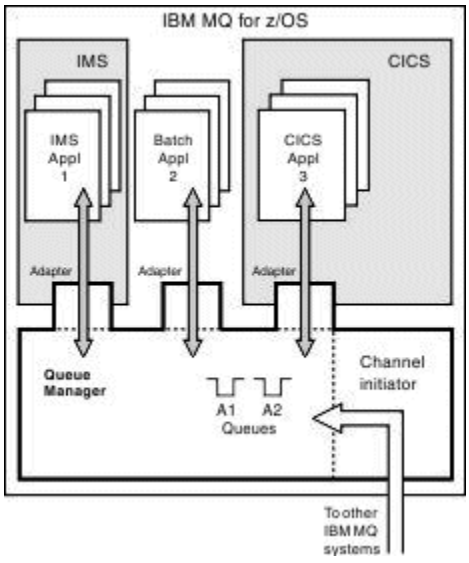


Figure 56. Overview of IBM MQ for z/OS

The queue manager subsystem on z/OS

On z/OS, IBM MQ runs as a z/OS subsystem that is started at IPL time. In the subsystem, the queue manager is started by executing a JCL procedure that specifies the z/OS data sets that contain information about the logs, and that hold object definitions and message data (the page sets). The

subsystem and the queue manager have the same name, of up to four characters. All queue managers in your network must have unique names, even if they are on different systems, sysplexes, or platforms.

z/OS The channel initiator on z/OS

The channel initiator provides and manages resources that enable IBM MQ distributed queuing. IBM MQ uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

To send messages from queue manager A to queue manager B, a *sending* MCA on queue manager A must set up a communications link to queue manager B. A *receiving* MCA must be started on queue manager B to receive messages from the communications link. This one-way path consisting of the sending MCA, the communications link, and the receiving MCA is known as a *channel*. The sending MCA takes messages from a transmission queue and sends them down a channel to the receiving MCA. The receiving MCA receives the messages and puts them on to the destination queues.

In IBM MQ for z/OS, the sending and receiving MCAs all run inside the channel initiator (the channel initiator is also known as the *mover*). The channel initiator runs as a z/OS address space under the control of the queue manager. There can be only a single channel initiator connected to a queue manager and it is run inside the same z/OS image as the queue manager. There can be thousands of MCA processes running inside the channel initiator concurrently.

Figure 57 on page 167 shows two queue managers within a sysplex. Each queue manager has a channel initiator and a local queue. Messages sent by queue managers on AIX and Windows are placed on the local queue, from where they are retrieved by an application. Reply messages are returned by a similar route.

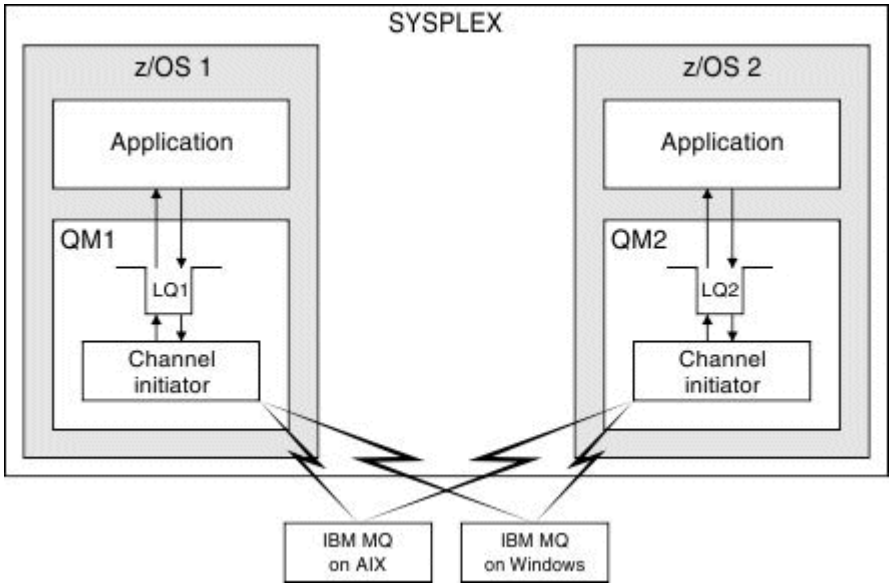


Figure 57. Communication between queue managers

The channel initiator also contains other processes concerned with the management of the channels. These processes include:

Listeners
These processes listen for inbound channel requests on a communications subsystem such as TCP, and start a named MCA when an inbound request is received.

Supervisor
This manages the channel initiator address space, for example it is responsible for restarting channels after a failure.

Name server
This is used to resolve TCP names into addresses.

TLS tasks
These are used to perform encryption and decryption and check certificate revocation lists.

z/OS SMF records for the channel initiator

The channel initiator (CHINIT) can produce SMF statistics records and accounting records with information on tasks and channels.

The CHINIT can produce SMF statistics records and accounting records with the following types of information:

- The tasks: dispatcher, adapter, Domain Name Server (DNS), and SSL. These tasks form what is called CHINIT statistics.
- Channels: provides accounting information similar to that available with the DIS CHSTATUS command. This is called channel accounting.

IBM MQ for Multiplatforms provides similar information by writing PCF messages to the SYSTEM.ADMIN.STATISTICS.QUEUE. See Channel statistics message data for further information on how statistics information is recorded on IBM MQ for Multiplatforms.

Statistics data

You can use this information to find out the following information:

- Whether you need more of the CHINIT tasks, such as number of SSL TCBs and how much CPU is used by these tasks.
- The average time for requests on these tasks.
- The longest duration request in the interval, and the time of day this occurred, for DNS and SSL tasks. You can correlate this time of day with problems you may experience with the channel.

Accounting data

You can use this information to monitor channel usage and find out the following information:

- The channels with the highest throughput.
- The rate at which messages were sent, and the rate of sending data in MB/second.
- The achieved batch size. If the achieved batch size is close to the batch size specified for the channel, the channel might be close to its limit for sending messages.

You use the START TRACE and STOP TRACE commands to control the collection of the accounting trace and the statistics trace. You can use the STATCHL and STATACLS options on the channel and queue manager to control whether channels produce SMF data.

z/OS Terms and tasks for managing IBM MQ for z/OS

Use this topic as an introduction to the terminology, and tasks that are specific to IBM MQ for z/OS.

Some of the terms and tasks required for managing IBM MQ for z/OS are specific to the z/OS platform. The following list contains some of these terms and tasks.

- Shared queues
- Page sets and buffer pools
- Logging
- Tailoring the queue manager environment
- Restart and recovery
- Security
- Availability
- Manipulating objects

- Monitoring and statistics
- Application environments

Shared queues

Queues can be *non-shared*, owned by and accessible to only one queue manager, or *shared*, owned by a *queue sharing group*. A queue sharing group consists of a number of queue managers, running within a single z/OS sysplex, that can access the same IBM MQ object definitions and message data concurrently. Within a queue sharing group, the shareable object definitions are stored in a shared Db2 database. The shared queue messages are held inside one or more coupling facility structures (CF structures). If the message data is too large to store directly in the structure (more than 63 KB in size), or if the message is large enough that installation-defined rules select it for offloading, the message control information is still stored in the coupling facility entry, but the message data is offloaded to a shared message data set (SMDS) or to a shared Db2 database. The shared message data sets, the shared Db2 database, and the coupling facility structures are resources that are jointly managed by all of the queue managers in the group.

Pages sets and buffer pools

When a message is put on to a non-shared queue, the queue manager stores the data on a page set in such a way that it can be retrieved when a subsequent operation gets a message from the same queue. If the message is removed from the queue, space in the page set that holds the data is later freed for reuse. As the number of messages held on a queue increases, so the amount of space used in the page set increases, and as the number of messages on a queue reduces, the space used in the page set reduces.

To reduce the performance cost of writing data to and reading data from the page sets, the queue manager buffers the updates into processor storage. The amount of storage used to buffer the page set access is controlled through IBM MQ objects called *buffer pools*.

For more information about page sets and buffer pools, see Storage management.

Logging

Any changes to objects held on page sets, and operations on persistent messages, are recorded as log records. These log records are written to a log data set called the *active log*. The name and size of the active log data set is held in a data set called the *bootstrap data set* (BSDS).

When the active log data set fills up, the queue manager switches to another log data set so that logging can continue, and copies the content of the full active log data set to an *archive log* data set. Information about these actions, including the name of the archive log data set, is held in the bootstrap data set. Conceptually, there is a ring of active log data sets that the queue manager cycles through; when an active log is filled, the log data is offloaded to an archive log, and the active log data set is available for reuse.

For more information about the log and bootstrap data sets, see “Logging in IBM MQ for z/OS” on page 232.

Tailoring the queue manager environment

When the queue manager is started, a set of initialization parameters that control how the queue manager operates are read. In addition, data sets containing IBM MQ commands are read, and the commands they contain are executed. Typically, these data sets contain definitions of the system objects required for IBM MQ to run, and you can tailor these to define or initialize the IBM MQ objects necessary for your operating environment. When these data sets have been read, any objects defined by them are stored, either on a page set or in Db2.

For more information about initialization parameters and system objects, see “System definition on z/OS” on page 243.

Recovery and restart

At any time during the operation of IBM MQ, there might be changes held in processor storage that have not yet been written to the page set. These changes are written out to the page set that is the least recently used by a background task within the queue manager.

If the queue manager terminates abnormally, the recovery phase of queue manager restart can recover the lost page set changes because persistent message data is held in log records. This means that IBM MQ can recover persistent message data and object changes right up to the point of failure.

If a queue manager that is a member of a queue sharing group encounters a coupling facility failure, the persistent messages on that queue can be recovered only if you have backed up your coupling facility structure.

For more information about recovery and restart, see “Recovery and restart on z/OS” on page 253.

Security

You can use an external security manager, such as Security Server (previously known as RACF) to protect the resources that IBM MQ owns and manages from access by unauthorized users. You can also use Transport Layer Security (TLS) for channel security. TLS is included as part of the IBM MQ product.

For more information about IBM MQ security, see “Security concepts in IBM MQ for z/OS” on page 269.

Availability

There are several features of IBM MQ that are designed to increase system availability in the event of queue manager or communications subsystem failure. For more information about these features, see “Availability on z/OS” on page 275.

Manipulating objects

When the queue manager is running, you can manipulate IBM MQ objects either through a z/OS console interface, or through an administration utility that uses ISPF services under TSO. Both mechanisms enable you to define, alter, or delete IBM MQ objects. You can also control and display the status of various IBM MQ and queue manager functions.

For more information about these facilities, see Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS.

You can also manipulate IBM MQ objects using the IBM MQ Explorer, a graphical user interface that provides a visual way of working with queues, queue managers, and other objects.

Monitoring and statistics

Several facilities are available to monitor your queue managers and channel initiators. You can also collect statistics for performance evaluation and accounting purposes.

For more information about these facilities, see “Monitoring and statistics on IBM MQ for z/OS” on page 279.

Application environments

When the queue manager has started, applications can connect to it and start using the IBM MQ API. These can be CICS, IMS, Batch, or WebSphere Application Server applications. IBM MQ applications can

also access applications on CICS and IMS systems that are not aware of IBM MQ, using the CICS and IMS bridges.

For more information about these facilities, see “IBM MQ and other z/OS products” on page 282.

For information about writing IBM MQ applications, see the following documentation:

- Developing applications
- Using C++
- Using IBM MQ classes for Java

z/OS Shared queues and queue sharing groups

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.

This section describes the attributes and benefits, and offers information about how several queue managers can share the same queues and the messages on those queues.

What is a shared queue?

A shared queue is a type of local queue. The messages on that queue can be accessed by one or more queue managers that are in a sysplex.

A queue sharing group

The queue managers that can access the same set of shared queues form a group called a *queue sharing group*.

Any queue manager can access messages

Any queue manager in the queue sharing group can access a shared queue. This means that you can put a message on to a shared queue on one queue manager, and get the same message from the queue from a different queue manager. This provides a rapid mechanism for communication within a queue sharing group that does not require channels to be active between queue managers.

IBM MQ supports the offloading of messages to Db2 or a shared message data set (SMDS). The offloading of messages of any size is configurable.

Figure 58 on page 172 shows three queue managers and a coupling facility, forming a queue sharing group. All three queue managers can access the shared queue in the coupling facility.

An application can connect to any of the queue managers within the queue sharing group. Because all the queue managers in the queue sharing group can access all the shared queues, the application does not depend on the availability of a specific queue manager; any queue manager in the queue sharing group can service the queue.

This gives greater availability because all the other queue managers in the queue sharing group can continue processing the queue if one of the queue managers has a problem.

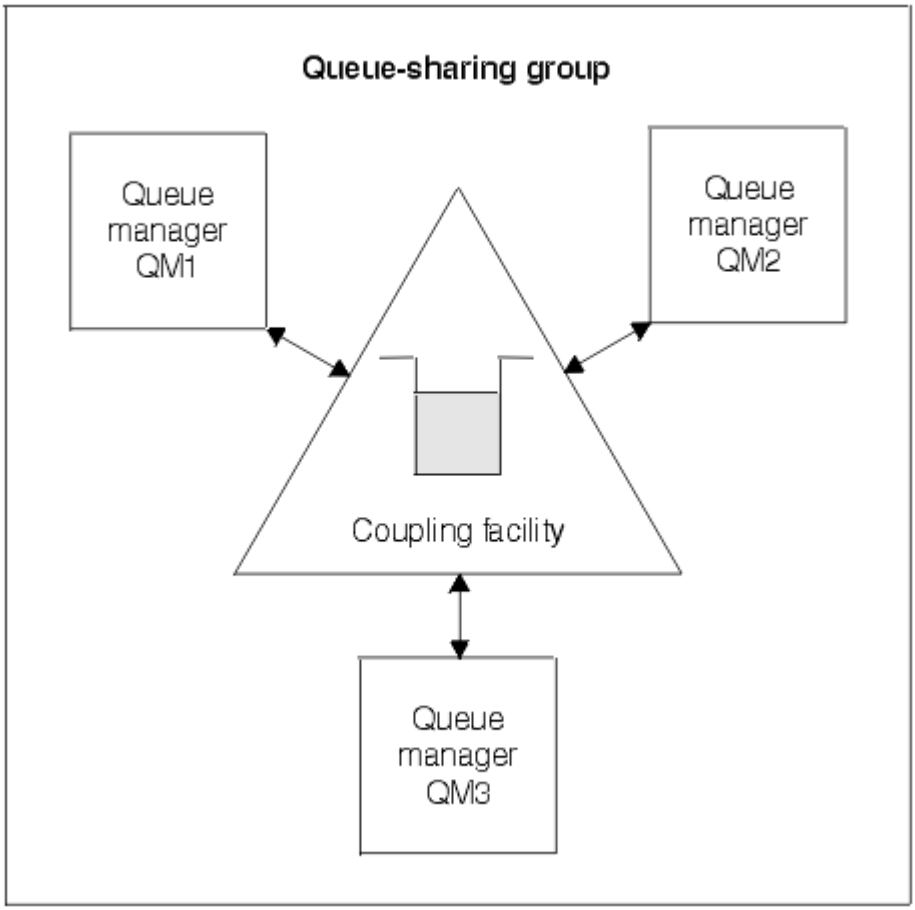


Figure 58. A queue sharing group

Queue definition is shared by all queue managers

Shared queue definitions are stored in the Db2 database table OBJ_B_QUEUE. Because of this, you need to define the queue only once and then it can be accessed by all the queue managers in the queue sharing group. This means that there are fewer definitions to make.

By contrast, the definition of a non-shared queue is stored on page set zero of the queue manager that owns the queue (as described in Page sets).

You cannot define a shared queue if a queue with that name has already been defined on the page sets of the defining queue manager. Likewise, you cannot define a local version of a queue on the queue manager page sets if a shared queue with the same name exists.

What is a queue sharing group?

A group of queue managers that can access the same shared queues is called a queue sharing group. Each member of the queue sharing group has access to the same set of shared queues.

Queue sharing groups have a name of up to four characters. The name must be unique in your network, and must be different from any queue manager names.

Figure 59 on page 173 illustrates a queue sharing group that contains two queue managers. Each queue manager has a channel initiator and its own local page sets and log data sets.

Each member of the queue sharing group must also connect to a Db2 system. The Db2 systems must all be in the same Db2 data-sharing group so that the queue managers can access the Db2 shared repository used to hold shared object definitions. These are definitions of any type of IBM MQ object (for example,

queues and channels) that are defined only once and then any queue manager in the group can use them. These are called *global* definitions and are described in Private and global definitions.

More than one queue sharing group can reference a particular data-sharing group. You specify the name of the Db2 subsystem and which data-sharing group a queue manager uses in the IBM MQ system parameters at startup.

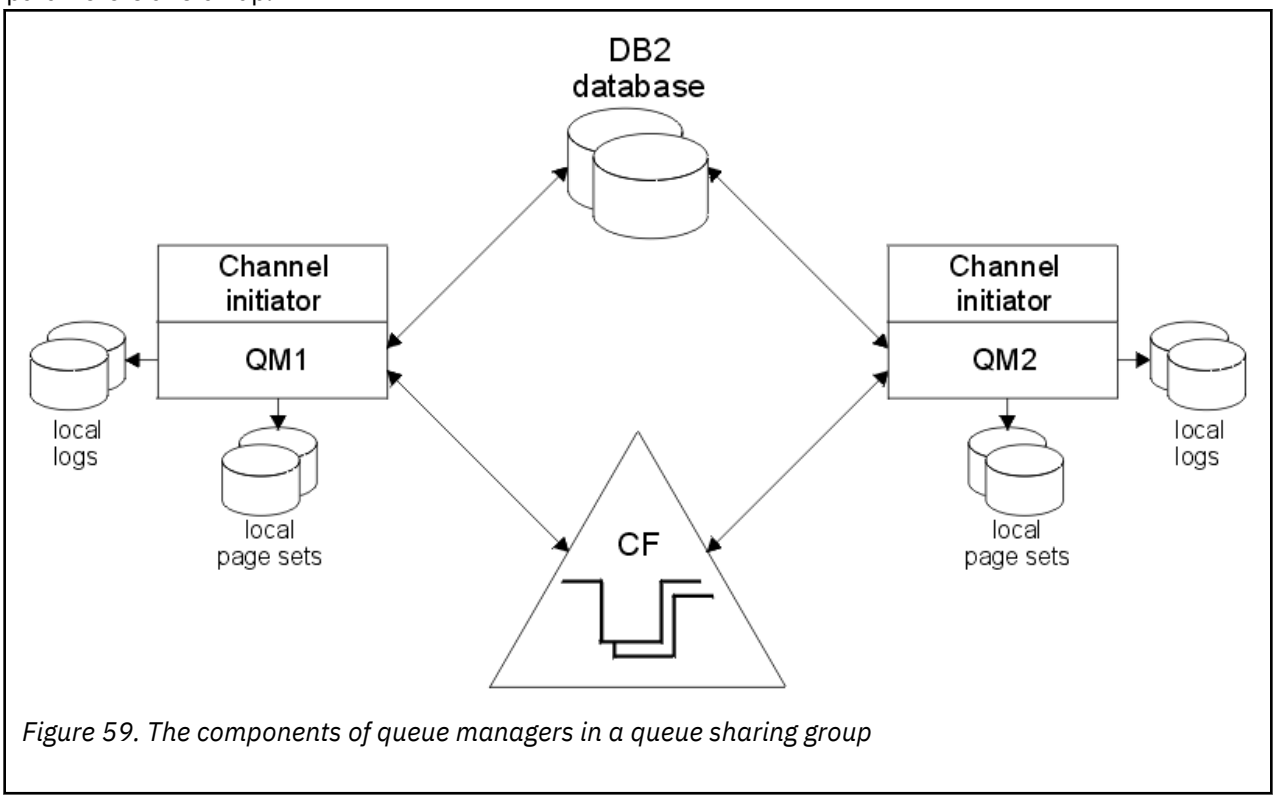


Figure 59. The components of queue managers in a queue sharing group

When a queue manager has joined a queue sharing group, it has access to the shared objects defined for that group, and you can use that queue manager to define new shared objects within the group. If shared queues are defined within the group, you can use this queue manager to put messages to and get messages from those shared queues. Any queue manager in the group can retrieve the messages held on a shared queue.

You can enter an MQSC command once, and have it executed on all queue managers within the queue sharing group as if it had been entered at each queue manager individually. The *command scope* attribute is used for this. This attribute is described in Directing commands to different queue managers.

When a queue manager runs as a member of a queue sharing group it must be possible to distinguish between IBM MQ objects defined privately to that queue manager and IBM MQ objects defined globally that are available to all queue managers in the queue sharing group. The *queue sharing group disposition* attribute is used for this. This attribute is described in Private and global definitions.

You can define a single set of security profiles that control access to IBM MQ objects anywhere within the group. This means that the number of profiles you have to define is greatly reduced.

A queue manager can belong to only one queue sharing group, and all queue managers in the group must be in the same sysplex. You specify which queue sharing group the queue manager belongs to in the system parameters at startup.

Related concepts

"Where are shared queue messages held?" on page 174
Each message on a shared queue is represented by an entry in a z/OS coupling facility list structure. If the message data is too large to fit in the same entry, it is offloaded either to a shared message data set (SMDS) or to Db2.

"Advantages of using shared queues" on page 190

Shared queue allows for IBM MQ applications to be scalable, highly available, and allows workload balancing to be implemented.

"Distributed queuing and queue sharing groups" on page 209
Distributed queuing and queue sharing groups are two techniques that you can use to increase the availability of your application systems. Use this topic to find further information about these techniques.

"Influencing workload distribution with shared queues" on page 213
Use this topic to understand the factors that affect workload distribution with shared queues in a queue sharing group.

Related reference

"Where to find more information about shared queues and queue sharing groups" on page 214
Use the table in this topic to find more information about how IBM MQ for z/OS uses shared queues and queue sharing groups.

z/OS Where are shared queue messages held?

Each message on a shared queue is represented by an entry in a z/OS coupling facility list structure. If the message data is too large to fit in the same entry, it is offloaded either to a shared message data set (SMDS) or to Db2.

If the CF structure has been configured to use System Class Memory (SCM), IBM MQ can use this with no additional configuration.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

Shared queue message storage

Messages that are put onto shared queues are not stored on page sets and do not use buffer pools.

The messages in shared queues have entries on list structures in the z/OS coupling facility (CF). Many queue managers in the same sysplex can access those messages using the CF list structure.

The message data for small shared queue messages is normally included in the coupling facility entry. For larger messages, the message data can be stored either in a shared message data set (SMDS), or as one or more binary large objects (BLOBs) in a Db2 table which is shared by a Db2 data sharing group. Message data exceeding 63 KB is always offloaded to SMDS or Db2. Smaller messages can also optionally be offloaded in the same way to save space in the coupling facility structure. See "Specifying offload options for shared messages" on page 176 for more details.

Messages put on a shared queue are referenced in a coupling facility structure until they are retrieved by an MQGET. Coupling facility operations are used to:

- Search for the next retrievable message
- Lock uncommitted messages on shared queues
- Notify interested queue managers about the arrival of committed messages

MQPUT and MQGET operations on persistent messages are recorded on the log of the queue manager performing that operation. This minimizes the risk of data loss in the event of a coupling facility failure.

The coupling facility

The messages held on shared queues are referenced inside a coupling facility. The coupling facility lies outside any of the z/OS images in the sysplex and is typically configured to run on a different power supply. The coupling facility is therefore resilient to software failures and you can configure it so that it is resilient to hardware failures or power-outages. This means that messages stored in the coupling facility are highly available.

Each coupling facility list structure used by IBM MQ is dedicated to a specific queue sharing group, but a coupling facility can hold structures for more than one queue sharing group. Queue managers in different queue sharing groups cannot share data. Up to 32 queue managers in a queue sharing group can connect to a coupling facility list structure at the same time.

A single coupling facility list structure can contain up to 512 shared queues. The total amount of message data stored in the structure is limited by the structure capacity. However, with **CFLEVEL(5)** you can use the offload parameters to offload data for messages less than 63 KB thereby increasing the number of messages which can be stored in the structure, although each message still requires at least a coupling facility entry plus at least 768 bytes of data, made up of 256 bytes for the entry and 512 bytes for the two elements of header and descriptor.

The size of the list structure is restricted by the following factors:

- It must lie within a single coupling facility.
- It might share the available coupling facility storage with other structures for IBM MQ and other products.

Coupling facility list structures can have storage class memory associated with them. In certain situations this storage class memory can be useful when used with shared queues. See "Use of storage class memory with shared queues" on page 191 for more information.

Planning the CF structure size

If you require guidance on the sizing of your CF structures you can use the MP16: IBM MQ for z/OS Capacity planning and tuning supportpac. You can also use the web-based tool CFSizer, which is provided by IBM to assist with CF sizes.

The CF structure object

The queue manager's use of a coupling facility structure is specified in a CF structure (CFSTRUCT) IBM MQ object.

These structure objects are stored in Db2.

When using z/OS commands or definitions relating to a coupling facility structure, the first four characters of the name of the queue sharing group are required. However, an IBM MQ CFSTRUCT object always exists within a single queue sharing group, and so its name does not include the first four characters of the name of the queue sharing group. For example, CFSTRUCT(MYDATA) defined in queue sharing group starting with SQ03 would use coupling facility list structure SQ03MYDATA.

CF structures have a CFLEVEL attribute that determines their functional capability:

- 1, 2 - can be used for nonpersistent messages less than 63 KB
- 3 - can be used for persistent and nonpersistent messages less than 63 KB
- 4 - can be used for persistent and nonpersistent messages up to 100 MB
- 5 - can be used for persistent and nonpersistent messages up to 100 MB and selectively offloaded to shared message data sets (SMDS) or Db2.

Note: When using IBM MQ you can encrypt a coupling facility structure. See Encrypting coupling facility structure data for more information.

Backup and recovery of the coupling facility

You can back up coupling facility list structures using the IBM MQ command BACKUP CFSTRUCT. This puts a copy of the persistent messages currently within the CF structure onto the active log data set of the queue manager making the backup, and writes a record of the backup to Db2.

If coupling facility fails, you can use the IBM MQ command RECOVER CFSTRUCT. This uses the backup record from Db2 to locate and restore persistent messages from the backup of the CF structure. Any activity since the last backup is replayed using the logs of all the queue managers in the queue sharing group, and the CF structure is then restored up to the point before the failure.

See the BACKUP CFSTRUCT and RECOVER CFSTRUCT commands for more details.

Related concepts

"Specifying offload options for shared messages" on page 176
You can choose where the message data for a shared queue message is stored, either in a Db2 table or a shared message data set (SMDS). You can also select which messages are offloaded, based on the size of the message and the current usage of the coupling facility structure (CF).

"Managing your shared message data set (SMDS) environment" on page 178
If you select shared message data sets to offload large messages then you must also be aware of the information that IBM MQ uses to manage these data sets and the commands used to work with this information. Use this topic to understand how to manage shared message data sets.

z/OS *Specifying offload options for shared messages*

You can choose where the message data for a shared queue message is stored, either in a Db2 table or a shared message data set (SMDS). You can also select which messages are offloaded, based on the size of the message and the current usage of the coupling facility structure (CF).

The message data for shared queues can be offloaded from the coupling facility and stored in either a Db2 table or in an IBM MQ managed data set called a *shared message data set* (SMDS).

For messages larger than the coupling facility entry size of 63 KB, offloading message data to a SMDS can have a significant performance improvement compared with offloading to Db2.

Every shared queue message is still managed using a list entry in a coupling facility structure, but when the message data is offloaded to the SMDS, the coupling facility entry only contains some control information and a list of references to the relevant disk blocks where the message is stored. Using this mechanism means the amount of coupling facility element storage required for each message is only a fraction of the actual size of the message.

Selecting where the shared queue messages are stored

The selection of SMDS or Db2 shared message storage is controlled with the **OFFLOAD(SMDS|DB2)** parameter on the **CFSTRUCT** definition. **OFFLOAD(SMDS)** is the default value.

This parameter also requires the **CFSTRUCT** to use **CFLEVEL(5)** or greater.

The **OFFLOAD** parameter is only valid from **CFLEVEL(5)**. See DEFINE CFSTRUCT for more details.

OFFLOAD(DB2) is supported primarily for migration purposes.

Selecting which shared queue messages are offloaded

Message data is offloaded to SMDS or Db2 based on the size of the message data, and the current usage of the coupling facility structure. There are three rules, and each rule specifies a matching pair of parameters. These parameters are a corresponding coupling facility structure usage threshold percentage (**OFFLDnTH**) and a message size limit (**OFFLDnSZ**).

The current implementation of the three rules is specified using the following pairs of keywords:

- OFFLD1TH and OFFLD1SZ
- OFFLD2TH and OFFLD2SZ
- OFFLD3TH and OFFLD3SZ

Rule pair	Default value	Description
Rule pair 1	OFFLD1TH(70) and OFFLD1SZ(32K)	If the coupling facility structure is more than 70% full offload data for messages exceeding 32 KB
Rule pair 2	OFFLD2TH(80) and OFFLD2SZ(4K)	If the coupling facility structure is more than 80% full offload data for messages exceeding 4 KB
Rule pair 3	OFFLD3TH(90) and OFFLD3SZ(0K)	If the coupling facility structure is more than 90% full offload data for messages exceeding 0 KB (all messages)

If an offload rule has the OFFLD *x* SZ value of 64K this indicates that the rule is not in effect. In this case messages will only be offloaded if another offload rule is in effect, or if the message is greater than 63.75 KB and so, too large to store in the structure.

Each message which is offloaded still requires 0.75 KB of storage in the coupling facility.

The three offload rules which can be specified for each structure are intended to be used as follows.

- Performance
  - When there is plenty of space in the application structure, message data should only be offloaded if it is too large to store in the structure, or if it exceeds some lower message size threshold such that the performance value of storing it in the structure is not worth the amount of structure space that it would need.
  - If a specific message size threshold is required, it is conventionally specified using the first offload rule.
- Capacity
  - When there is very little space in the application structure, the maximum amount of message data should be offloaded so as to make the best use of the remaining space.
  - The third offload rule is conventionally used to indicate that when the structure is nearly full, most messages should be offloaded, so the entries in the application structure will be typically of the minimum size (requiring about 0.75K bytes).
  - The usage threshold parameter should be chosen based on the application structure size and the maximum anticipated backlog. For example, if the maximum anticipated backlog is 1M messages, then the amount of structure storage required for this number of messages is about 0.75G bytes. This means for example that if the structure is about 10G bytes, the usage threshold for offloading all messages must be set to 92% or lower.
  - Structure space is divided into elements and entries, and even though there may be enough space overall, one of these may run out before the other. The system provides AUTOALTER capabilities to adjust the ratio when necessary, but this is not very sensitive, so the amount of space actually available may be somewhat less. It may be better therefore to aim to use not more than 90% of the maximum structure space, so in the previous example, the usage threshold for offloading all messages would be better set around 80%.
- Cushioned transition:
  - As the amount of space left in the coupling facility structure decreases, it would be undesirable to have a large sudden change in the performance characteristics. It is also undesirable for coupling facility management to have a sudden threshold change in the typical ratio of entries to elements being used.
  - The second offload rule is conventionally used to provide some intermediate cushion between the performance and capacity biased offload rules. It can be set to cause a significant increase

in offload activity when the space used in the coupling facility structure exceeds an intermediate threshold. This means that the remaining space is used up more slowly, and gives the coupling facility automatic alter processing more time to adapt to the higher usage levels.

If the coupling facility structure cannot be expanded, and there is a need to store at least some predetermined number of messages, the third rule can be modified as necessary to ensure that offloading of data for all messages starts at an appropriate threshold to ensure space is reserved for that predetermined number of messages.

For example, if the coupling facility structure size is 4 GB, and the predetermined number of messages is 1 million, then 1,000,000 * 0.75 KB are needed, which is 768 MB, 18.75% of 4 GB. In this case the threshold for offloading all messages needs to be set around 80% rather than 90%. This gives parameters OFFLD3TH(80) and OFFLD3SZ(0K) . The other offload parameters would also need to be adjusted.

If it is found that offloading very small messages has a significant performance impact, but the relative impact is less for larger messages, then the usage thresholds for the other rules can be reduced to offload larger messages earlier, leaving more space in the structure for small messages before they need to be offloaded.

For example, if messages exceeding 32KB occur frequently but the elapsed time performance for offloading them (as determined from RMF statistics or application performance) is very similar to that for keeping them in the coupling facility, then the threshold for the first rule could be set to 0% to offload all such messages. This gives parameters OFFLD1TH(0) and OFFLD1SZ(32K). Again the other offload parameters would need to be adjusted.

If there are many messages around specific intermediate sizes, such as 16 KB and 6 KB, then it might be useful to change the message size option for the second rule so that the larger ones get offloaded at a fairly low usage threshold, saving a significant amount of space, but the smaller ones still get stored only in the coupling facility.

z/OS *Managing your shared message data set (SMDS) environment*

If you select shared message data sets to offload large messages then you must also be aware of the information that IBM MQ uses to manage these data sets and the commands used to work with this information. Use this topic to understand how to manage shared message data sets.

SMDS objects

The properties and status of each shared message data set are tracked in a shared SMDS object which can be updated through any queue manager in the queue sharing group.

There is one shared message data set for each queue manager that can access each coupling facility application structure. The shared message data set is identified by the owning queue manager name, specified using the SMDS keyword, and by the application structure name, specified using the CFSTRUCT keyword.

Note: When defining SMDS data sets for a structure, you must have one for each queue manager.

The SMDS object is stored in an array (with one entry per queue manager in the group) which forms an extension of the corresponding CFSTRUCT object stored in Db2.

There is no command to DEFINE or DELETE the SMDS object because it is created or deleted as part of the CFSTRUCT object, but there is a command to ALTER it to change settings for an individual owning queue manager.

For further information on SMDS commands, see “SMDS related commands” on page 189

SMDSCONN information

It is possible for a shared message data set to be in a normal state, but for one or more queue managers to be unable to connect to it, for example because of a problem with a security definition or with direct access device connectivity. It is therefore necessary for each queue manager to keep track of connection

status, and availability information for each shared message data set, indicating for example whether it can currently connect to it, and if not why not.

The SMDSCONN information represents a queue manager connection to a shared message data set. As for the shared message data set itself, it is identified by the queue manager which owns the shared message data set (as specified on the SMDS keyword for the shared object itself) combined with the CFSTRUCT name.

There is no parameter to identify the connecting queue manager because commands addressed to a specific queue manager can only refer to SMDSCONN information for that same queue manager.

The SMDSCONN information entries are maintained in main storage in the owning queue manager, and are re-created when the queue manager is restarted. However, if a connection from an individual queue manager has been explicitly stopped, this information is also stored as a flag in a connection array in the corresponding CFSTRUCT or SMDS object, so that it persists across a queue manager restart.

Status and availability information

Status information indicates the state of a resource or connection (for example, whether it is not yet being used, is in normal use or is in need of recovery). It is usually described using the STATUS keyword. The possible values depend on the type of object.

Status information is normally updated automatically, for example when an error is detected while using the resource or connection. However, in some cases a command can also be used to update the status, to allow for cases when it is not possible for a queue manager to determine the correct status automatically.

Availability information indicates whether the resource or connection can be used, and is usually primarily determined by the status information. For the resource or connection types used in shared message data set support, three levels of availability are implemented:

Available
This means that the resource is available to be used normally. This does not necessarily mean that it is in use at present (which can be determined instead from the STATUS value). For a data set, if it requires restart processing, this allows the owning queue manager to open it, but other queue managers must wait until the data set is back in the ACTIVE state.

Unavailable because of error
This means that the resource has been made unavailable automatically because of an error and is not expected to be available again until some form of repair or recovery processing has been performed. However, attempts to make it available again are permitted without operator intervention. Such an attempt can also be triggered by a command to mark the resource as enabled, or a command which changes the status in such a way as to indicate that recovery processing has been completed.

The reason that the resource has been made unavailable is normally obvious from the related STATUS value, but in some cases there may be other reasons to make the resource unavailable, in which case a separate REASON value is provided to indicate the reason.

Unavailable because of operator command
This means that access to the resource has been explicitly disabled by a command. It can only be made available by using a command to enable it again.

SMDS availability

For the shared SMDS object, the availability is described by the ACCESS keyword, with the possible values ENABLED, SUSPENDED and DISABLED.

The availability can be updated using a **RESET SMDS** command for the relevant shared object from any queue manager in the group to set ACCESS(ENABLED) or ACCESS(DISABLED).

If the availability was previously ACCESS(SUSPENDED), changing it to ACCESS(ENABLED) will trigger a new attempt to use the shared message data set, but if the previous error is still present, the availability will be reset back to ACCESS(SUSPENDED).

SMDSCONN availability

For a local SMDSCONN information entry, the availability is described by the AVAIL keyword, with the possible values NORMAL, ERROR or STOPPED. The availability can be updated using a **START SMDSCONN** or **STOP SMDSCONN** command addressed to a specific queue manager to enable or disable its connection.

If the availability was previously AVAIL(ERROR), changing it to AVAIL(NORMAL) will trigger a new attempt to use the shared message data set, but if the previous error is still present, the availability will be reset back to AVAIL(ERROR).

Shared message data set shared status and availability

The availability of each shared message data set is managed within the group using shared status information, which can be displayed using the **DISPLAY CFSTATUS** command with TYPE(SMDS). This displays status information for each queue manager that has activated a data set for each structure. Each data set can be in one of the following states:

NOTFOUND
This means that the corresponding data set has not yet been activated. This status only appears when a specific queue manager is specified, as data sets which have not been activated are skipped when all queue managers are selected.

NEW
The data set is being opened and initialized for the first time, ready to be made active.

ACTIVE
This means that the data set is fully available and should be allocated and opened by all active queue managers for the structure.

FAILED
This means the data set is not available at all (except for recovery processing) and must be closed and deallocated by all queue managers.

INRECOVER
This means that media recovery (using RECOVER CFSTRUCT) is in progress for this data set.

RECOVERED
This indicates that a command has been issued to switch a failed data set back to the active state, but further restart processing is required which is not yet complete, so the data set can only be opened by the owning queue manager for restart processing.

EMPTY
The data set contains no messages. The data set is put into this state if it is closed normally by the owning queue manager, at a time when it does not contain any messages. It can also be put into EMPTY state when the previous data set contents are to be discarded because the application structure has been emptied (using **RECOVER CFSTRUCT** with TYPE PURGE or, for a nonrecoverable structure only, by deleting the previous instance of the structure). The next time the data set is opened by its owning queue manager, the space map is reset to empty, and the status is changed to ACTIVE. As the previous data set contents are no longer required, a data set in this state can be replaced with a newly allocated data set, for example to change the space allocation or move it to another volume.

The command output includes the date and time at which recovery logging was enabled, if any, and the date and time at which the data set failed, if it is not currently active.

A shared message data set can be put into a FAILED state either by a **RESET SMDS** command or automatically when any of the following types of error are detected:

- The data set cannot be allocated or opened by the owning queue manager.
- Validation of the data set header fails after it has been successfully opened by any queue manager.
- A permanent I/O error occurs when the owning queue manager is reading or writing data.
- A permanent I/O error occurs when another queue manager is reading data from a data set which had successfully completed open processing and validation.

When a data set is in the FAILED or INRECOVER state, it not available for normal use, so if the availability state is ACCESS(ENABLED) it is changed to ACCESS(SUSPENDED).

If a data set has been put into the the FAILED state but no media recovery is required, for example because the data was still valid but the storage device was temporarily offline, then the **RESET SMDS** command can be used to request changing the status directly to the RECOVERED state.

When the data set enters the RECOVERED state, either on completion of recovery processing or as a result of the **RESET SMDS** command, then it is ready to be used again once restart processing has been completed. If it was in the ACCESS(SUSPENDED) state, it is automatically switched back to the ACCESS(ENABLED) state, which allows the owning queue manager to perform restart processing. When restart processing completes, the state is changed to ACTIVE and all other queue managers can then connect to the data set again.

Shared message data set connection status and availability

Each queue manager maintains local status and availability information for its connection to each shared message data set owned by itself and by other queue managers in the group. This information can be displayed using the **DISPLAY SMDSCONN** command.

If it is unable to access a shared message data set in the ACTIVE state which belongs to another queue manager it flags the connection as being unavailable from its own point of view.

If the error definitely indicates a problem with the data set itself, the queue manager also automatically changes the shared status to indicate that the data set is now in a FAILED state. However, if the error could be caused by an environmental problem, such as not being authorized to open the data set, the queue manager issues error messages and treats the data set as being unavailable, but it does not modify the shared data set status. If the environmental error turns out to be a problem with the data set anyway (for example it has been allocated on a device which cannot be accessed by some of the queue managers) then an operator can use the RESET SMDS command specifying STATUS(FAILED) to allow the data set to be recovered or repaired as necessary.

If a connection to a shared message data set could not be established but the data set appears to be valid, a new attempt to use it can be triggered by issuing a **START SMDSCONN** command for the owning queue manager.

If there is an operational need to terminate the connection between a specific queue manager and a data set temporarily, but the data set itself is not damaged, then the data set can be closed and deallocated using the **STOP SMDSCONN** command. If the data set is in use, the queue manager will close it normally (although any requests for data in that data set will be rejected with a return code). If it is the owned data set, the queue manager will save the space map during CLOSE processing, avoiding the need for restart processing.

If a data set needs to be taken out of service temporarily from all queue managers (for example to move it) but is not damaged, then it is best to use **STOP SMDSCONN** for the relevant data set with the option CMDSCOPE(*) to stop the queue managers using it first, as this will avoid the need for restart processing when the data set is brought back into service. In contrast, if the data set is marked as FAILED this tells queue managers that they must stop using it immediately, which means that the space map will not be saved and will need to be rebuilt by restart processing.

Access to any shared message data sets previously in the ACCESS(SUSPENDED) state will be retried if the queue manager is restarted.

Shared message data set recovery logging

Persistent shared messages are logged for media recovery purposes. This means that the messages can be recovered after any failure of coupling facility structures or shared message data sets, provided that the recovery logs are still intact. Persistent messages can also be re-created from the recovery logs at another site for disaster recovery purposes.

When the message data is written to a shared message data set, each block written to the data set is logged separately followed by the message entry (including the data map) as written to the coupling

facility. The recovery process always recovers the coupling facility structure, but it does not need to recover individual shared message data sets except when the data set status is FAILED, or when the status is ACTIVE but the data set header record is no longer valid, indicating that the data set has been re-created. A data set is not selected for recovery if its status is ACTIVE and the data set header is still valid, nor if its status is EMPTY, indicating that no messages were stored in it at the time of the failure.

Shared message data set backups

When BACKUP CFSTRUCT is used to make a backup of the shared messages in an application structure, any data for persistent messages stored in shared message data sets is backed up at the same time, as for persistent shared messages previously stored in DB.

Shared message data set recovery

If a shared message data set is corrupted or lost, then it needs to be put into the FAILED state to stop the queue managers from using it until it has been repaired. This normally happens automatically, but can also be done using the **RESET SMDS** command specifying STATUS(FAILED).

If the shared message data set contained any persistent messages, these can be recovered using the RECOVER CFSTRUCT command. This command first restores any persistent message data for that shared message data set from the most recent BACKUP CFSTRUCT command, then applies all logged changes since that time. If no **BACKUP CFSTRUCT** command has been performed since the time that the data set was first activated, it is reset to empty then all changes since activation are applied.

If the CFSTRUCT contents and all of the shared message data sets are unavailable, for example in a disaster recovery situation, they can all be recovered in a single **RECOVER CFSTRUCT** command.

If a shared message data set is damaged but recovery was not active for the CFSTRUCT, or the log containing the latest BACKUP CFSTRUCT is unavailable or unusable, then the messages offloaded to that data set cannot be recovered. In this case, the **RECOVER CFSTRUCT** command with the parameter TYPE(PURGE) can be used to mark the shared message data set as empty and delete any messages from the structure which had data stored in that data set.

When the **RECOVER CFSTRUCT** command is issued, the shared message data set status is changed from FAILED to INRECOVER. If recovery completes successfully, the status is automatically changed to RECOVERED, otherwise it changes back to FAILED.

When the data set is changed to the RECOVERED state, this tells the owning queue manager that it can now try to open the data set and perform restart processing.

Shared message data set recovery and syncpoints

The shared message data set recovery process reapplies the changes for all complete log records up to the end of the log, regardless of syncpoints.

If changes were made within syncpoint, restart or recovery processing for the CFSTRUCT may result in backing out of uncommitted requests, so some of the recovered changes may not actually be used, but there is no harm in recovering them anyway.

It is also possible that an uncommitted MQPUT message may have been written to the structure but the corresponding data may not have been written to the data set or the log (as I/O completion is only forced at the start of syncpoint processing). This is harmless because restart processing will back out the message entry in the structure, so the fact that it refers to unrecovered data does not matter.

Shared message data set restart processing

If a queue manager connection to a CFSTRUCT terminates normally, the queue manager writes out the free block space map for each shared message data set to a checkpoint area within the data set, just before the data set is closed. The space map can then be read in again at connection restart time, provided that neither the CFSTRUCT nor the shared message data set require any recovery processing before the next restart.

However, if a queue manager terminates abnormally, or the structure or data set require any recovery processing, then additional processing is required to rebuild the space map dynamically when the queue manager connection to the structure is restarted.

Provided that the data set itself did not need to be recovered, queue manager restart simply scans the current contents of the structure to locate references to message data owned by the current queue manager, and marks the relevant data blocks as owned in the space map. Other queue managers can continue to use the structure and read the data owned by the restarting queue manager while the space map is being rebuilt.

Shared message data set restart after recovery

If a shared message data set had to be recovered from a backup, then all nonpersistent messages stored in the data set will have been lost, and if the data set was recovered using TYPE(PURGE) then all messages stored in the data set will have been lost. Until recovery has completed, the data set will be marked as FAILED or INRECOVER so any attempt to read one of the affected messages from another queue manager returns an error code indicating that the data set is temporarily unavailable.

When the data set has been recovered, the status is changed to RECOVERED, which allows the owning queue manager to open it for restart processing, but the data set remains unavailable to other queue managers. Queue manager restart scans the structure to rebuild the space map for any remaining messages. The scan also checks for messages for which the data has been lost, and deletes them from the structure (or if necessary flags them as lost, to be deleted later).

The data set status is automatically changed from RECOVERED to ACTIVE when this restart scan completes, at which point other queue managers can start using it again.

Shared message data set usage information

The DISPLAY USAGE command now also shows information about shared message data set space and buffer pool usage for any currently open shared message data sets. This information is displayed if either the new option TYPE(SMDS) or the existing option TYPE(ALL) is specified.

Shared message data performance and capacity considerations

Monitoring data set usage

The current percentage full of each owned shared message data set can be displayed by the **DISPLAY USAGE** command with the option **TYPE(SMDS)**.

The queue manager will normally automatically expand a shared message data set when it reaches 90% full, provided that the option **DSEXPAND(YES)** is in effect for the SMDS definition. This applies when either the SMDS option is set to **DSEXPAND(YES)** or the SMDS option is set to **DSEXPAND(DEFAULT)** and the CFSTRUCT default option is set to **DSEXPAND(YES)**.

If the expansion attempt fails because no secondary allocation size was specified when the data set was created (giving message IEC070I with reason code 203) the queue manager repeats the expansion request using an override secondary allocation of approximately 20% of the current size.

When a data set is expanded, the new data set extents are formatted as part of the expansion processing, which can take tens of seconds, or even minutes for very large extents. The new space becomes available for use after formatting is complete and the catalog has been updated to show the new high used control interval.

If new messages are being created very rapidly, it is possible for the existing data set to become full before expansion processing completes. In this case, any request which could not allocate space is temporarily suspended until the expansion attempt completes and the new space becomes available for use. If the expansion was successful the request is retried automatically.

If an expansion attempt fails, because of a lack of available space or because the maximum extents have already been reached, a message is issued giving the reason for the failure, then the override option for the affected SMDS is automatically altered to **DSEXPAND(NO)** to prevent further expansion

attempts. In this case, there is a risk that the data set may become full, in which case further action may be needed as described in Data set becomes full.

Monitoring application structure usage

The usage level of an application structure can be displayed using the MVS **DISPLAY XCF,STRUCTURE** command specifying the full name of the application structure (including the queue sharing group prefix). The IXC360I response message shows current usage of elements and entries.

When the structure usage exceeds the **FULLTHRESHOLD** value specified in the CFRM policy, the system issues message IXC585E and may perform automatic **ALTER** actions if specified, which may either alter the entry to element ratio or increase the structure size.

Optimising buffer pool sizes

Each buffer in a shared buffer pool is used to read or write a contiguous range of pages for one message of up to the logical block size. If the message spills over into further blocks, each range of pages in a separate block requires a separate buffer.

Buffers containing message data after a write or read operation are retained in storage and reused using a least-recently-used (LRU) cache scheme so that a request to read the same data again shortly afterwards will not need to go to disk. This provides a significant optimization when shared messages are written and then read back soon afterwards by applications running on the same system. If messages owned by another queue manager are browsed for selection purposes then retrieved, this also avoids the need to reread the message from disk.

This means that the number of buffers required for each application structure is one for each concurrent API request which reads or writes large messages for that application structure plus some number of additional buffers which will be used to save recently accessed data in order to optimize subsequent read accesses.

For shared buffer pools, if there are insufficient buffers, API requests will simply wait if a buffer is not immediately available. However, this situation should be avoided as it can cause significantly degraded performance.

The statistics from the **DISPLAY USAGE** command for shared buffer pools show whether there have been any buffer waits within the current statistics interval, and also shows the lowest number of free buffers (or a negative value indicating the maximum number of threads which waited for a buffer at any time), the number of buffers which have saved data, and the percentage of the times that a buffer request has successfully found saved data on the LRU chain ("LRU hits") instead of having to read it ("LRU misses") [1] .

- If there have been any waits, the number of buffers should be increased.
- If there are many unused buffers, the number of buffers may be reduced to make more storage available in the region for other purposes.
- If there are many buffers containing saved data but the proportion of reads which were hits against that saved data is very small, the number of buffers may be reduced if the storage could be better used for other purposes. The number of buffers should not however be reduced by more than the lowest number of free buffers, as that could trigger waits, and it should preferably be high enough that the lowest free buffer count is normally well above zero.

Deleting shared message data sets

The DELETE CFSTRUCT command (which is only allowed when all shared queues in the structure are empty and closed) does not delete the shared message data sets themselves, but they can be deleted in the usual way after this command has completed. If the same data set is to be reused as a shared message data set, it must be reformatted first to reset it to the empty state.

[1] `(Hits / (Hits+Misses))* 100`

Exception situations for shared message data sets

There are a number of exception situations which can occur during normal use, even when no software or hardware error is present.

Data set becomes full

If a data set becomes full but cannot be expanded, or the expansion attempt fails, applications using the corresponding queue manager to write large messages to the corresponding application structure will receive error 2192, `MQRC_STORAGE_MEDIUM_FULL` (also known as `MQRC_PAGESET_FULL`).

A data set could become full because of a failure in the application which is supposed to process the data, causing a large backlog of messages to accumulate. If so, expanding the data set any further will only be a temporary solution, and it is important to get the processing application going again as soon as possible.

If more space can be made available the **ALTER SMDS** command can then be used to set **DSEXPAND(YES)** or **DSEXPAND(DEFAULT)** (assuming that YES has been set or assumed as the **DSEXPAND** default for the CFSTRUCT definition) to trigger a retry. If the reason for the failure was however that maximum extents had been reached, the new expansion attempt will be rejected with a message and **DSEXPAND(NO)** will be set again. In this case, the only way to expand it any further is to reallocate it, which involves making it temporarily unavailable, as described next.

Data set needs to be moved or reallocated

If a data set needs to be moved or expanded but is otherwise in normal use, it can be taken out of use temporarily to allow it to be moved or reallocated. Any API request which attempts to use the data set while it is unavailable will receive the reason code `MQRC_DATA_SET_NOT_AVAILABLE.`

1. Use the **RESET SMDS** command to mark the data set as **ACCESS(DISABLED)**. This will cause it to be closed normally and deallocated by all currently connected queue managers.
2. Move or reallocate the data set as necessary, copying the old contents to the newly allocated data set, for example using the Access Method Services (AMS) **REPRO** command.

   Do not attempt to preformat the new data set before copying the old data into it, as this would result in the copied data being appended to the end of the formatted data set.
3. Use the **RESET SMDS** command to mark the data set as **ACCESS(ENABLED)** again, to bring it back into use.

   If the old contents are smaller than the size of the new data set, the rest of the space will be preformatted automatically when the new data set is opened.

   If the old contents were larger than the size of the new data set then the queue manager has to scan the messages in the coupling facility structure and rebuild the space map to ensure that none of the active data has been lost. If any reference is found to a data block which is outside the new extents, the data set is marked as **STATUS(FAILED)** and must be repaired by replacing the data set with one of the correct size and either copying the old data set into it again or using **RECOVER CFSTRUCT** to recover any persistent messages.

Coupling facility structure is low on space

If the coupling facility structure is running out of space, causing message IXC585E, it is worth checking whether the offload rules have been set to ensure that the maximum amount of data is being offloaded in this case. If not, the offload rules can be modified using the **ALTER CFSTRUCT** command.

Error situations for shared message data sets

There are a number of problems to be aware of, which can only be caused by errors and not occur in normal operational situations.

Owned data set cannot be opened

If the queue manager which owns a shared message data set cannot allocate it or open it, or the data set attributes are not supported, the queue manager sets an appropriate **SMDSCONN** status value of **ALLOCFAIL** or **OPENFAIL** and sets the **SMDSCONN** availability to **AVAIL(ERROR)**. It also sets the SMDS availability to **ACCESS(SUSPENDED)**. When the error has been corrected, use the **RESET SMDS** command to set **ACCESS(ENABLED)** to trigger a retry, or issue the **START SMDSCONN** command to the owning queue manager.

Read-only data set cannot be opened

If a queue manager cannot allocate or open a shared message data set owned by another queue manager and marked as **STATUS(ACTIVE)**, it assumes that this is probably due to a specific problem with its connection to the data set (represented by the **SMDSCONN** object) rather than a problem with the data set itself.

It marks the **SMDSCONN** as **STATUS(ALLOCFAIL)** or **STATUS(OPENFAIL)** as appropriate and marks the **SMDSCONN** availability as AVAIL(ERROR) to prevent further attempts to use it.

If the problem can been corrected without affecting the status of the data set itself, use the **START SMDSCONN** command to trigger a retry.

If the problem turns out to be a problem with the data set itself, then the **RESET SMDS** command can be used to mark the data set as **STATUS(FAILED)** until it it has been recovered. When the data set has been recovered, the action of changing the status back to **STATUS(ACTIVE)** will cause other queue managers to be notified. If the **SMDSCONN** is marked as **AVAIL(ERROR)**, it will automatically be changed back to **AVAIL(NORMAL)** to trigger a new attempt to open the data set.

Data set header is corrupt

If the data set was successfully opened but the format of the header information is incorrect, the queue manager closes and deallocates the data set and sets the status set to **STATUS(FAILED)** and the availability to **ACCESS(SUSPENDED)**. This allows **RECOVER CFSTRUCT** to be used to recover the contents.

If the error arose because the data set contained residual data from another use and had not been subsequently preformatted, then preformat the data set and use the **RESET SMDS** command to change the status to **STATUS(RECOVERED)**.

Otherwise, the data set must be recovered.

Data set is unexpectedly empty

If the queue manager opens a data set which is marked as **STATUS(ACTIVE)** but finds that it is uninitialized or newly preformatted but otherwise valid, the queue manager closes and deallocates the shared message data set then sets the status to **STATUS(FAILED)** and the availability to **ACCESS(SUSPENDED)**.

Data set has permanent I/O errors

If a data set has permanent I/O errors after successful **OPEN** processing, it probably needs recovery. The queue manager will mark the data set as **STATUS(FAILED)** so that all currently connected queue managers will close and deallocate it.

Data set has recoverable I/O errors

If there are hardware problems with the data set, it is possible that this might result in recoverable I/O errors which are not reflected back to the queue manager but which cause significant performance degradation, and also indicate a risk of permanent I/O errors in the near future.

In this case, the data set may be taken off line for recovery by using the **RESET SMDS** command to mark it as **STATUS(FAILED)**. This will cause it to be closed and deallocated by all queue managers, so for example it could be moved to a new volume before being made available again.

When a data set is made unavailable in this way, the space map is not saved so the queue manager connection restart processing will need to scan the coupling facility structure to locate messages in the data set and rebuild the space map before the data set can be made available again. As an

alternative, if the shared message data set is still usable, it set can be made unavailable more gently by using the **RESET SMDS** command to mark the data set **ACCESS(DISABLED)** until it is ready to be made available again.

Data set contents are incorrect

The queue manager cannot detect directly that a data set contains incorrect data or is not up to date, for example because a volume including that data set had to be restored from backups. However, it performs integrity checks which make it very unlikely that any such errors could result in incorrect message data being seen by application programs.

For integrity checking purposes, each message block in the data set is prefixed with a copy of the corresponding coupling facility entry ID, including a unique time stamp, which is checked whenever the message block is read, before the message data is passed to the user program. If the message block prefix does not match the entry ID (and the coupling facility entry was not deleted in the mean time) the message block is assumed to be damaged and unusable.

If the damaged message was persistent, the data set is marked as **STATUS(FAILED)** and the structure contents must be recovered using the **RECOVER CFSTRUCT** command. If the damaged message was non-persistent, there is no way to recover it, so a diagnostic message is issued and the corresponding coupling facility message entry is deleted.

If no saved space map is available when the data set is opened, it is rebuilt by scanning the coupling facility structure for references to data in the data set. During this scan, the queue manager performs a number of actions:

1. The queue manager determines the location of the most recent message (if any) currently remaining in the data set.
2. The queue manager then reads that message from the data set to ensure that the block prefix matches the message entry id

These actions ensure that the queue manager detects any case where the data set is down-level, and marks the data set as FAILED. This check does however tolerate the case where the data set was restored from a previous copy and either no new messages had been added since then or all messages added since that copy had been subsequently read and deleted.

To protect against down-level data in the case where the data set was closed normally, the queue manager performs a number of actions:

1. The queue manager saves a copy of the space map time stamp in the SMDS object within Db2 when the data set is closed normally.
2. The queue manager then checks the space map time stamp is the same, when the data set is opened again

If the time stamp does not match, this suggests that a down-level copy of the data set might have been used, so the queue manager ignores the existing space map and rebuilds it, which will succeed only if no message data was actually lost.

Note: These integrity checks do not guarantee to detect a down-level or damaged data set in all theoretically possible cases. For example, they will not detect a case where the start of a message block is valid but the rest of the data has been partly overwritten.

Recovery scenarios for shared message data sets

This section described shared message data set recovery scenarios.

Data set recovery where no data was lost

In some cases, the correct contents of a failed data set can be restored without needing actual recovery. One example is where a data set contains residual data from a previous use and has not been preformatted again, which can be fixed by preformatting it. Another case is when a data set has been moved, but there was an error in the process of copying the data across, which can be fixed by copying the data again correctly.

In such cases, the corrected data set can be made available again by using the **RESET SMDS** command to set **STATUS(RECOVERED)**. If the availability is currently **ACCESS(SUSPENDED)** this will automatically set it back to **ACCESS(ENABLED)**.

When the owning queue manager is notified that the data set has been recovered, it scans the structure contents to reconstruct the space map, then changes the status to **STATUS(ACTIVE)**. The other queue managers can then start reading the data set again.

Data set recovery with TYPE(NORMAL)

If the contents of a data set have been lost, but the application structure was defined with **RECOVER(YES)** and the appropriate recovery logs are available, the **RECOVER CFSTRUCT** command can be used to recover any persistent messages stored in the structure including persistent message data offloaded to shared message data sets. This command restores the current state using information logged by the **BACKUP CFSTRUCT** command plus all logged changes to persistent messages since the backup time.

The **RECOVER CFSTRUCT** command always recovers all persistent messages in the coupling facility structure together with offloaded message data stored in Db2. For offloaded data stored in shared message data sets, each data set is only selected for recovery processing if it is already marked as **STATUS(FAILED)** or if it is found to be unexpectedly empty or otherwise invalid when opened by recovery processing. Any shared message data set which is marked as active and which passes the validation checks does not need to be recovered, as the existing message data is already correct, but the header is updated to indicate that any saved space map will need to be rebuilt after recovery.

Recovery processing is only possible when the structure has been marked as failed, as the complete contents of the structure need to be reconstructed by recovery processing. However, if at least one shared message data set has been marked as failed the **RECOVER CFSTRUCT** command will automatically mark the structure as failed if necessary to allow recovery processing to proceed.

Recovery may be performed from any queue manager in the queue sharing group, provided that it has been given write access to the relevant data sets.

Only persistent messages are backed up and logged, so normal recovery processing will restore all persistent messages, but will cause any non-persistent messages in the structure to be lost.

When recovery has completed, any data set which was selected for recovery is automatically changed to **STATUS(RECOVERED)**, and if the availability was **ACCESS(SUSPENDED)** it is changed to **ACCESS(ENABLED)**. The queue manager rebuilds the space map for each data set by scanning the messages in the coupling facility, then marks the data set as **STATUS(ACTIVE)** so that it can be used again.

Data set recovery with TYPE(PURGE)

For a recoverable structure, if the data set contents have been lost, but recovery is not possible for some reason, for example because recovery logs are not available or recovery would take too long, the **RECOVER CFSTRUCT** command can be used with **TYPE(PURGE)** to get the structure back to a usable state. This resets the structure to the empty state and marks all of the associated data sets as **STATUS(EMPTY)**.

Deleting the application structure

If a non-recoverable application structure is deleted using the MVS **SETXCF FORCE** command, or as a result of structure failure, then the next time the structure is connected, message CSQE028I is issued to say that the structure has been reset and all existing messages have been discarded, and any existing data sets are automatically reset to **STATUS(EMPTY)** as well. This action makes a non-recoverable structure usable again after loss of data either in the structure or in any of the associated data sets.

If a recoverable application structure is deleted, it will be treated in the same way as if the structure had failed.

Data set recovery fails

If **RECOVER CFSTRUCT** cannot complete for some reason, for example because a log data set is no longer available, or because the queue manager terminated while recovery was in progress, then any data set for which recovery was at least started will be marked in the header to show that partial recovery has been attempted, and the data set will be left in the **STATUS(FAILED)** state.

In this case, the options are to repeat the original recovery request or to recover with **TYPE(PURGE)** instead, discarding the existing data.

If an attempt is made to mark the data set as **STATUS(RECOVERED)** without actually recovering it, then the next time it is opened the queue manager will see that the header indicates incomplete recovery and mark it as **STATUS(FAILED)** again.

Off site disaster recovery

For off site disaster recovery, persistent shared messages can be re-created using only the logs and the Db2 shared objects containing the CFSTRUCT definitions and associated SMDS status information.

After setting up the Db2 tables containing the definitions, the application structure and the shared message data sets can be set up as empty. When a queue manager connects to them and finds that they are unexpectedly empty, it will mark them as failed, after which a single **RECOVER CFSTRUCT** command can be used to recover all persistent messages for all affected structures.

z/OS *SMDS related commands*

This topic describes and provides access to the commands relating to shared message data sets.

Display and alter the **CFSTRUCT** options relating to large message offload (**OFFLOAD** and offload rules) and shared message data sets (**DSGROUP, DSBLOCK, DSBUFS, DSEXPAND)**:

- DISPLAY CFSTRUCT
- DEFINE CFSTRUCT
- ALTER CFSTRUCT
- DELETE CFSTRUCT

Display **CFSTRUCT** status relating to large message offload **(OFFLDUSE)**:

- DISPLAY CFSTATUS

Display and alter override data set options (**DSEXPAND** and **DSBUFS**) for individual queue managers:

- DISPLAY SMDS
- ALTER SMDS

Display or modify the status and availability of the data sets within the queue sharing group:

- DISPLAY CFSTATUS TYPE(SMDS)
- RESET SMDS

Display SMDS data set space usage and buffer usage information for a queue manager:

- DISPLAY USAGE TYPE(SMDS)

Display or modify the status and availability of the connections (**SMDSCONN**) to the data sets from an individual queue manager:

- DISPLAY SMDSCONN
- START SMDSCONN
- STOP SMDSCONN

Backup and recover shared messages, including large message data in SMDS when necessary:

- BACKUP CFSTRUCT
- RECOVER CFSTRUCT

z/OS *Advantages of using shared queues*

Shared queue allows for IBM MQ applications to be scalable, highly available, and allows workload balancing to be implemented.

The advantages of shared queues

The shared queue architecture, where cloned servers pull work from a single shared queue, has some useful properties:

- It is scalable, by adding new instances of the server application, or even adding a new z/OS image with a queue manager (in the queue sharing group) and a copy of the application.
- It is highly available.
- It naturally performs *pull* workload balancing, based on the available processing capacity of each queue manager in the queue sharing group.

Using shared queues for high availability

The following examples illustrate how you can use a shared queue to increase application availability.

Consider an IBM MQ scenario where client applications running in the network want to make requests of server applications running on z/OS. The client application constructs a request message and places it on a request queue. The client then waits for a reply from the server, sent to the reply-to queue named in the message descriptor of the request message.

IBM MQ manages the transportation of the request message from the client machine to the server's input queue on z/OS and of the response from the server back to the client. By defining the server's input queue as a shared queue, any messages put to the queue can be retrieved on any queue manager in the queue sharing group. This means that you can configure a queue manager on each z/OS image in the sysplex and, by connecting them all to the same queue sharing group, any one of them can access messages on the server's input queue.

Messages on the input queue of the server are still available, even if one of the queue managers terminates abnormally or you have to stop it for administrative reasons. You can take an entire z/OS image offline and the messages will still be available.

To take advantage of this availability of messages on a shared queue, run an instance of the server application on each z/OS image in the sysplex to provide higher server application capacity and availability, as shown in Figure 60 on page 191.

One instance of the server application retrieves a request message from the shared queue and, based on the content, performs its processing, producing a result that is sent back to the client as an IBM MQ message. The response message is destined for the reply-to queue and reply-to queue manager named in the message descriptor of the request message.

There are a number of options that you can use to configure the return path. For more information about these options, see “Distributed queuing and queue sharing groups” on page 209.

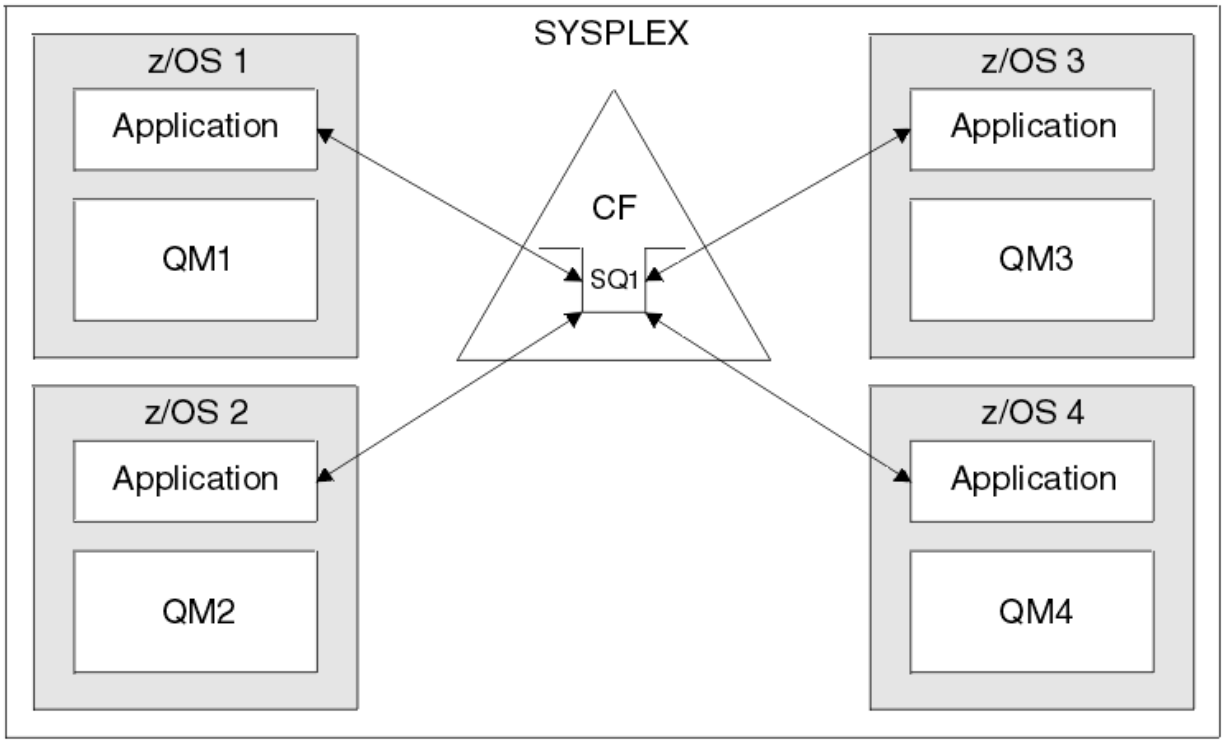


Figure 60. Multiple instances of an application servicing a shared queue

Peer recovery

To further enhance the availability of messages in a queue sharing group, IBM MQ detects if another queue manager in the group disconnects from the coupling facility abnormally and completes units of work for that queue manager that are still pending, where possible. This feature is known as *peer recovery.*

Suppose a queue manager terminates abnormally at a point where an application has retrieved a request message from a queue in sync point, but has not yet put the response message or committed the unit of work. Another queue manager in the queue sharing group detects the failure, and backs out the in-flight units of work being performed on the failed queue manager. This means that the request message is put back on to the request queue and is available for one of the other server instances to process, without waiting for the failed queue manager to restart.

If IBM MQ cannot resolve a unit of work automatically, you can resolve the shared portion manually to enable another queue manager in the queue sharing group to continue processing that work.

z/OS *Use of storage class memory with shared queues*

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

The z13, zEC12, and zBC12 machines allow the installation of Flash Express cards. These cards contain flash solid-state drives (SSD). After installation, flash storage from the cards can be allocated to one or more LPARs where it is typically known as SCM.

SCM sits between real storage and direct access storage device (DASD) in terms of both I/O latency and cost. Because SCM has no moving parts, it exhibits much lower I/O latencies than DASD.

SCM is also much cheaper than real storage. As a result, a large amount of storage can be installed for a relatively low cost; for example, a pair of Flash Express cards contains 1424 GB of usable storage.

These characteristics mean that SCM is useful when a large amount of data must be taken from real storage in a short period of time, because that data can be written to SCM much quicker than it can be written to DASD. This specific point can be very useful when using coupling facility (CF) list structures containing IBM MQ shared queues.

Why list structures fill up

When a CF structure is defined, it is configured with a SIZE attribute that describes the maximum size of the structure. Because CF structures are always permanently resident in real storage, the sum of the SIZE attributes of the structures that are defined on a CF should be less than the amount of the real storage allocated to the CF.

As a result, there is a constant pressure to keep the SIZE value of any given structure to the minimum value possible, so that more structures can fit into the CF. However, ensuring structures are large enough to achieve their purpose can result in a conflicting pressure, because making a structure too small means that it might fill up, disrupting the applications or subsystems making use of it.

There is a strong need to accurately size a structure based on its expected usage. However, this task is difficult to do because workloads can change over time and accounting for their fluctuations is not easy.

IBM MQ shared queues use CF list structures to store messages. IBM MQ calls CF structures, which contain messages and application structures.

Application structures are referenced using the information stored in IBM MQ CFSTRUCT objects. When a message that is smaller than 63 KB is put on a shared queue, the message is stored entirely in an application structure as a single list entry, and zero or more list elements.

Because IBM MQ shared queues use list structures, the pressures described also affect shared queues. In this case, the maximum number of messages that can be stored on a shared queue is a function of the:

- Size of the messages on the queue
- Maximum size of the structure
- Number of entries and elements available in the structure

Because up to 512 shared queues can use the same structure, and effectively compete for entries and elements, this complicates matters even further

IBM MQ queues are used for the transfer of data between applications, so a common situation is an application putting messages to a queue, when the partner application, which should be getting those messages, is not running.

When this happens the number of messages on the queue increase over time until one or more of the following situations occur:

- The putting application stops putting messages.
- The getting application starts getting messages.
- Existing messages on the queue start expiring, and are removed from the queue.
- The queue reaches its maximum depth in which case an MQRC_Q_FULL reason code is returned to the putting application.
- The structure containing the shared queue reaches its maximum size, or the CF containing the structure runs out of available storage. In either case, an MQRC_STORAGE_MEDIUM_FULL reason code is returned to the putting application.

In the last three cases the queue is full. At this point the putting application has a problem because there is nowhere for its messages to go. The putting application typically solves this problem by using one or more of the following solutions:

- Repeatedly retry putting the message, optionally with a delay between retries.
- Put the messages somewhere else, such as a database or a file. The messages can be accessed later and put to the queue as normal.
- Discard the message if it is nonpersistent.

However, for some classes of applications, for example those with a large volume of incoming messages, or no access to a file system, these solutions are not practical. There is a real need to ensure that queues never, or are extremely unlikely to, fill up in the first place and this is especially pertinent to shared queues.

SMDS and offload rules

The offload rules introduced in IBM WebSphere MQ 7.1 provide a way of reducing the likelihood of an application structure filling up.

Each application structure has three rules associated with it, specified using three pairs of keywords:

- OFFLD1SZ and OFFLD1TH
- OFFLD2SZ and OFFLD2TH
- OFFLD3SZ and OFFLD3TH

Each rule specifies the conditions that must be met for message data to be offloaded to the storage mechanism that is associated with the application structure. Two types of storage mechanisms are currently available:

- Db2
- Group of Virtual Storage Access Method (VSAM) linear data sets, which IBM MQ calls a shared message data set (SMDS).

The following example shows the MQSC command to create an application structure named LIST1, using the DEFINE CFSTRUCT command.

This structure has the default offload rules in place, and uses SMDS as the offload mechanism. This means that when the structure is 70% full (OFFLD1TH), all messages that are 32 KB or larger (OFFLD1SZ) are offloaded to SMDS.

Similarly, when the structure is 80% full (OFFLD2TH) all messages that are 4 KB or larger (OFFLD2SZ) are offloaded. When the structure is 90% full (OFFLD3TH) all messages (OFFLD3SZ) are offloaded.

```
DEFINE CFSTRUCT(LIST1)
CFLEVEL(5)
OFFLOAD(SMDS)
OFFLD1SZ(32K) OFFLD1TH(70)
OFFLD2SZ(4K) OFFLD2TH(80)
OFFLD3SZ(0K) OFFLD3TH(90)
```

An offloaded message is stored in the offload medium, and a pointer to the message is stored in the structure. While the offload rules reduce the chance of the structure filling up, by putting less message data in the structure as it runs out of storage, some data is still written to the structure for each message. That is, the pointer to the offloaded message.

Additionally, the offload rules come with a performance cost. Writing a message to a structure is relatively quick and is largely dominated by the time spent to send the request for the write to the CF. The actual writing to the structure is fast, happening at real storage speeds.

Writing a message to SMDS is much slower because it includes writing to the structure for the message pointer, and writing the message data to SMDS. This second write operation is done at DASD speed and has the potential to add latency. If Db2 is used as the offload mechanism the performance cost is much greater.

z/OS *How storage class memory works with IBM MQ for z/OS*
An overview of the use of storage class memory (SCM) with IBM MQ for z/OS shared queues.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

A coupling facility (CF) that is at CFLEVEL 19, or greater, can have SCM allocated to it. The structures defined in that CF can then be configured to make use of SCM to reduce the chances of the structures filling up (known as a structure full condition). When a structure configured to make use of SCM fills up past a system-determined point, the CF starts moving data from the structure into SCM, which frees space in the structure for new data.

Note: Because SCM itself can fill up, allocating SCM to a structure reduces only the likelihood of a structure full condition, but does not entirely remove the chance of one occurring.

A structure is configured to use SCM by specifying both the **SCMALGORITHM** and **SCMMAXSIZE** keywords in the coupling facility resource manager (CFRM) policy, containing the definition of that structure.

Note that after these keywords are specified, and the CFRM policy is applied, the structure must be rebuilt, or deallocated so that they can take effect.

SCMALGORITHM keyword

Because the input/output speed of SCM is slower than that of real storage, the CF uses an algorithm that is tailored to the expected use of the structure in order to reduce the impact of writing to, or reading from, SCM.

The algorithm is configured by the **SCMALGORITHM** keyword in the CFRM policy for the structure, using the *KEYPRIORITY1* value. Note that you should use the *KEYPRIORITY1* value only with list structures used by IBM MQ shared queues.

The *KEYPRIORITY1* algorithm works by assuming that most applications will get messages from a shared queue in priority order; that is, when an application gets a message, it gets the oldest message with the highest priority.

When a structure starts to fill past the system-defined threshold of 90%, the CF starts asynchronously migrating messages that are least likely to be got next. These are messages with lower priorities that were more recently put on the queue.

This asynchronous migration of messages from the structure into SCM is known as "pre-staging".

Pre-staging reduces the performance cost of using SCM because it reduces the chance of an application being blocked during the occurrence of synchronous input/output to SCM.

In addition to pre-staging, the *KEYPRIORITY1* algorithm also asynchronously brings back messages from SCM and into the structure when sufficient free space is available. For the *KEYPRIORITY1* algorithm, this means that when the structure is less than or equal to 70% full.

The act of bringing messages from SCM into the structure is known as "pre-fetching".

Pre-fetching reduces the likelihood of an application trying to get a message that has been pre-staged to SCM and having to wait while the CF synchronously brings back the message into the structure.

SCMMAXSIZE keyword

The **SCMMAXSIZE** keyword defines the maximum amount of SCM that can be used by a structure. Because SCM is allocated to the structure by the CF when it is required, it is possible to specify a **SCMMAXSIZE** that is greater than the total amount of free SCM available. This is known as "over-committing".

Important: Never over-commit SCM. If you do, the applications that are relying on it will not obtain the behavior that they expect. For example, IBM MQ applications using shared queues might get unexpected MQRC_STORAGE_MEDIUM_FULL reason codes.

The CF uses various data structures to track its use of SCM. These data structures reside in the real storage that is allocated to the CF and, as a result, reduce the amount of real storage that can be used by structures. The storage used by these data structures is known as "augmented space".

When a structure is configured with SCM, a small amount of real storage is allocated from the CF to the structure known as fixed augmented space. This is allocated even if the structure never actually uses any SCM. As data from the structure is stored into SCM, extra dynamic augmented space will be allocated from the spare real storage in the CF.

When the data is removed from SCM, the dynamic augmented space is returned to the CF. Augmented space, either fixed or dynamic, is never taken from the real storage that is allocated to a structure.

In addition to augmented storage, when a structure is configured to use SCM, the amount of control storage used by that structure increases. This means that a list structure configured with SCM can contain fewer entries and elements than a structure of the same size without SCM being configured.

To understand the impact of SCM on new or existing structures, use the CFSizer tool.

A final important point to note is that after data is moved from the structure into SCM, and dynamic augmented space has been used, the structure cannot be altered either manually or automatically.

That is, the amount of storage allocated to the structure cannot be increased or decreased, the entry-to-element ratio that is used by the structure cannot be changed, and so on. To make the structure alterable again, the structure must not have any data stored in SCM and must not be making use of dynamic augmented storage.

z/OS *Why use SCM*

Emergency storage and improved performance are two use cases for using SCM with IBM MQ for z/OS.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This section introduces the theory behind the two possible scenarios. For further details on how you set up the scenarios, see:

- “Emergency storage - basic configuration” on page 198
- “Improved performance - basic configuration” on page 204

Important: The use of SCM with CF structures is not dependent on any specific version of IBM MQ. However the emergency storage scenario works only with IBM WebSphere MQ 7.1 and later, because it requires SMDS and the offload rules.

Emergency storage

SMDS and message offloading can be used in conjunction with SCM to reduce the likelihood of an MQRC_STORAGE_MEDIUM_FULL reason code being returned to an IBM MQ application during an extended outage.

Overview

A single shared queue is configured on an application structure. The putting application puts messages onto the shared queue; the getting application gets messages from the shared queue.

During normal running, the queue depth is expected to be close to zero, but a business requirement indicates that the system must be able to tolerate a two-hour outage of the getting application. This means that the shared queue must be able to contain two hours of messages from the putting application.

Currently, this process is achieved by using the default offload rules, and SMDS, so that the size of the structure is minimized, while reducing the performance cost that is associated with offloading.

The rate of messages being sent to the shared queue is expected to double in the short to medium term. Although the requirement that the system be able to tolerate a two-hour outage still exists, not enough real storage is available in the CF to double the size of the structure.

Because the CF, which contains the application structure, resides on a zEC12 machine, the possibility exists to associate sufficient SCM with the structure to store enough messages so that a two-hour outage can be tolerated.

Consider what happens over a period of time:

1. Initially, the system is in its steady state. Both the putting and getting application are running normally and the queue depth is close to, or at, zero. The result is that the application structure is largely empty.
2. At a certain time, the getting application suffers an unexpected failure and stops. The putting application continues to put messages to the queue and the application structure starts to fill up.
3. After the structure reaches 70% full, the conditions of the first offload rule are met and all messages with a size greater than or equal to 32 KB are offloaded to SMDS.

   See "SMDS and offload rules" on page 193 for an overview of the offload rules.
4. As messages continue to be put to the shared queue, the structure continues to fill (either because of the message data being stored in the structure, or as a result of the pointers to the offloaded messages being stored in the structure).

   When the structure reaches 80% full, the second offload rule starts to apply and messages that are 4 KB or greater are offloaded to SMDS.
5. When the structure is past 90% full, all messages are offloaded to SMDS and only the message pointers are being placed in the structure.

   About this time, the pre-staging algorithm starts to run, and begins moving data from the structure into SCM. Assuming all messages on the queue are the same priority, the newest messages are pre-staged.

   Because all messages are now being offloaded to SMDS, the data being moved into SCM is not actual message data, but instead the pointers to the messages on SMDS.

   As a result, the number of messages that can be stored on the combination of the structure, and the SCM and SMDS associated with the structure, is very large.

   Performance: During this stage of the outage, the putting application can suffer a degree of performance degradation because of having to write to SMDS. In this case, the use of SCM should not be a limiting factor on the putting application in terms of performance. SCM provides extra space to prevent the structure filling up.
6. Eventually the getting application is available again and the outage is over.

   However SCM is still being used by the structure. The getting application starts reading messages off the queue, getting the oldest, highest priority messages first.

   Because these messages were written before the structure started to fill up, they come out entirely from the real storage portion of the structure.
7. As the structure starts to empty, it goes below the threshold at which pre-staging is active, and so pre-staging stops.
8. The structure usage reduces below the point at which the offload rules take effect, so messages are no longer offloaded to SMDS unless they are more than 63 KB.

   At about this time, the pre-fetch algorithm starts moving data from SCM in to the structure. Because the getting application gets messages from the queue in the order expected by the SCM algorithms, messages are brought in before the getting application needs them.

   The result is that the getting application never needs to wait for messages to be brought in synchronously from SCM.

9. As the getting application continues to move down the queue, it starts retrieving messages that were offloaded to SMDS.
10. Finally, the system is in a steady state again. No messages are stored in SCM or SMDS, and the queue depth is close to zero.

Improved performance

This scenario describes using SCM to increase the number of messages that can be stored on a shared queue without incurring the performance cost of using SMDS.

Description

For this scenario, a putting and getting application communicate through a shared queue which is stored in application structure.

The putting application tends to run in bursts, when it puts a large number of messages in a short amount of time. Then, in an extended period of time, it produces no messages at all.

The getting application sequentially processes each message, and performs complex processing on each one. As a result, most of the time the queue depth is zero, except for when the putting application starts to run, where the queue depth starts increasing as messages are being put faster than they are being got.

The queue depth increases until the putting application stops, and the getting application has enough time to process all the messages on the queue.

Notes:

1. In this scenario, the key factor is performance. The messages being sent to the queue are always less than 63 KB and so never need to be offloaded to SMDS.
2. The application structure has been sized so that it is large enough to contain all of the messages that will be placed on it by the putting application in a single "burst."
3. The offload rules must all be disabled so that, even when the structure starts to fill, the messages are not offloaded to SMDS. This is because the performance costs that are associated with writing messages to, and reading messages from, SMDS are deemed unacceptable.

Over time, the number of messages that the putting application send in a burst must increase by several orders of magnitude. Because the getting application must process each message sequentially, the number of messages on the queue increases to the point where the structure fills up.

At this point, the putting application receives a reason code (MQRC_STORAGE_MEDIUM_FULL) when putting a message, and the put operation fails. The putting application can only briefly tolerate periods when it is unable to put messages to the queue. If the period is too long, the application ends.

Assuming that you do not have the time, or skills available, to rewrite either the putting application or getting application, this problem has three possible solutions:

1. Increase the size of the application structure.
2. Add offload rules to the application structure so that messages are offloaded to SMDS as the queue starts to fill up.
3. Associate SCM with the structure.

The first solution is quick to implement, but not enough real storage is available on the CF.

The second solution might also be quick to implement, but the performance impact of offloading to SMDS is considered too significant to use this option.

The third solution, associating SCM with the structure, provides an acceptable balance of cost and performance.

Associating SCM with a structure results in a higher use of real storage in the CF because of the augmented storage that get operations used. However, the actual amount of real storage will be less than the amount used in the first option.

Another consideration is the cost of SCM. However this cost is much cheaper than real storage. These factors combine to make the third option cheaper than the first option.

Although the third option, potentially, might not perform as well as the first option, the pre-fetch and pre-staging algorithms used by the CF can combine to make the differences in performance acceptable, or in some cases negligible.

Certainly the performance can be much better than using SMDS to offload messages.

Consider what happens over a period of time:

1. Initially, the getting application is active and waiting for messages to be delivered to the shared queue. The putting application is not active and the shared queue is empty.
2. At a certain time, the putting application becomes active, and starts putting a large number of messages to the shared queue. The getting application starts getting the messages, but the queue depth rapidly starts to increase because the getting application is slower than the putting application.

   As a result, the application structure starts to fill up.
3. As the time increases, the putting application is still active. The application structure fills up to approximately 90%.

   This is when the SCM pre-staging algorithm starts to move messages from the structure into SCM, freeing space in the structure.

   Because the getting application gets the oldest, highest priority messages from the queue first, it is always getting messages from the structure and does not need to wait for messages to be brought synchronously from SCM in to the structure.
4. The putting application is still active and putting messages to the shared queue. However, the application never receives an MQRC_STORAGE_MEDIUM_FULL reason code, because enough space exists in SCM to store all the messages that do not fit in the structure.
5. Eventually, the putting application stops because it has no more messages to put.

   The pre-staging algorithm stops because the structure falls below 90% in use, and the getting application continues processing the messages in the queue.
6. As the getting application starts to free space in the structure, the pre-fetch algorithm starts to bring messages back from SCM in to the structure.

   Because the getting application processes messages in the order expected by the pre-fetch algorithm, the getting application never becomes blocked waiting for message data to be brought synchronously from SCM in to the structure.
7. Finally, the getting application processes all the messages on the shared queue, and waits until the next message is available. The structure and SCM are empty of messages.

z/OS *Emergency storage - basic configuration*

How you set up a basic scenario for emergency storage on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

SMDS and message offloading can be used in conjunction with SCM to reduce the likelihood of an MQRC_STORAGE_MEDIUM_FULL reason code being returned to an IBM MQ application during an extended outage.

For example, your enterprise has an application that puts messages onto the queue and an application that gets messages from the queue. During normal running, you expect the queue depth to be close to

zero, but a business requirement indicates that the system be able to tolerate a two-hour outage of the application that gets the messages.

This means that the shared queue being used must be able to contain two hours of messages from the putting application. Currently you achieve this by using the default offload rules, and SMDS.

You expect the rate of messages being sent to the shared queue to double in the short to medium term. Although your requirement that the system be able to tolerate a two-hour outage still exists, not enough real storage is available in the CF to double the size of the structure. Because the CF containing the application structure resides on a zEC12 machine, you have the capability of associating sufficient SCM with the structure to store enough messages, so that a two-hour outage can be tolerated

This initial scenario uses a:

- Queue sharing group, IBM1, that contains a single queue manager, CSQ3. In addition to the administration structure, the queue sharing group defined a single application structure, SCEN1.
- Coupling facility (CF) CF01, in which the SCEN1 application structure is stored as the IBM1SCEN1 structure. This structure has a maximum size of 1 GB.
- Single shared queue, SCEN1.Q that the application structure uses.

This configuration is illustrated in Figure 61 on page 199.

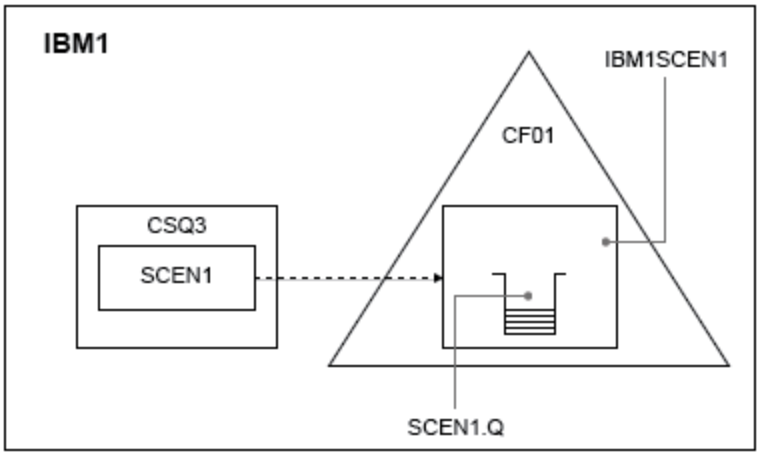


Figure 61. Basic configuration

Furthermore, assume that queue manager CSQ3 is already the only member of queue sharing group IBM1.

You must add the definition for structure IBM1SCEN1 to the coupling facility resource manager (CFRM) policy. For simplicity, the structure is defined so that it can be created in only a single coupling facility, CF01, by specifying PREFLIST(CF01).

Attention: To allow for high availability in your production system, you should include at least two CFs in the PREFLIST for any structures that are used by IBM MQ.

Procedure

1. Refresh the CFRM policy by using the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN1
```

   Sample CFRM policy for structure IBM1SCEN1:

```
STRUCTURE
NAME(IBM1SCEN1)
SIZE(1024M)
INITSIZE(512M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
```

```
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
```

2. Verify that the structure has been created correctly, by using the following command:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

   At this point, your structure has not been allocated, shown by the STATUS line, to the queue sharing group.

3. Configure IBM MQ to make use of the structure defined in the CFRM policy.

   a. Use the DEFINE CFSTRUCT command, with the structure name of SCEN1 to create an IBM MQ CFSTRUCT object:

```
DEFINE CFSTRUCT(SCEN1)
CFCONLOS(TOLERATE)
CFLEVEL(5)
DESCR('Structure for SCM scenario 1')
RECOVER(NO)
RECAUTO(YES)
OFFLOAD(DB2)
OFFLD1SZ(64K) OFFLD1TH(70)
OFFLD2SZ(64K) OFFLD2TH(80)
OFFLD3SZ(64K) OFFLD3TH(90)
```

   .

   b. Validate the structure, using the DISPLAY CFSTRUCT command.

   c. Define the SCEN1.Q shared queue, to use the SCEN1 structure, using the following MQSC command:

```
DEFINE QLOCAL(SCEN1.Q) QSGDISP(SHARED) CFSTRUCT(SCEN1) MAXDEPTH(999999999)
```

4. Use IBM MQ Explorer to put a single message to the queue SCEN1.Q and take the message off again.

5. Issue the following command to check that the structure is now allocated:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

   Check in the output from the command, that the STATUS line shows ALLOCATED.

Results

You have created the basic configuration. You can now obtain an idea of the baseline performance of your configuration using whatever method you select.

What to do next

Add SMDS and SCM to the initial structure

Related concepts

“Use of storage class memory with shared queues” on page 191

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

z/OS *Adding SMDS and SCM to the initial structure*

How you add SMDS and SCM for emergency storage on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This part of the task uses the basic configuration described in “Emergency storage - basic configuration” on page 198. The scenario describes the addition of shared message data sets (SMDS), and then of SCM to the initial structure.

This final configuration is illustrated in Figure 62 on page 201.

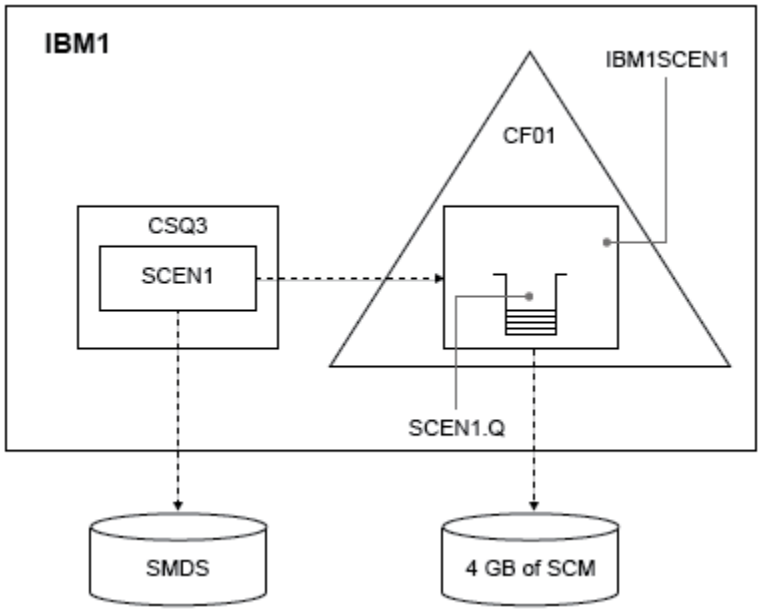


Figure 62. Configuration adding SMDS and SCM for emergency storage

Procedure

1. Create the SMDS data set that the SCEN1 application structure uses, by editing the **CSQ4SMDS** sample JCL, as shown:

```
//CSQ4SMDS JOB NOTIFY=&SYSUID
//*
//* Allocate SMDS
//*
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER           -
(NAME(CSQSMDS.SCEN1.CSQ3.SMDS) -
MEGABYTES(5000 3000)     -
LINEAR             -
SHAREOPTIONS(2 3) )      -
DATA                -
(NAME(CSQSMDS.SCEN1.CSQ3.SMDS.DATA) )
/*
//*
//* Format the SMDS
//*
//FORM   EXEC PGM=CSQJUFMT,COND=(0,NE),REGION=0M
//STEPLIB DD DSN=MQ800.SCSQANLE,DISP=SHR
//     DD DSN=MQ800.SCSQAUTH,DISP=SHR
//SYSUT1  DD DISP=OLD,DSN=CSQSMDS.SCEN1.CSQ3.SMDS
//SYSPRINT DD SYSOUT=*
```

2. Issue the ALTER CFSTRUCT command to change the SCEN1 application structure, to use SMDS for offloading, implementing the default offload rules:

```
ALTER CFSTRUCT(SCEN1) OFFLOAD(SMDS) OFFLD1SZ(32K) OFFLD2SZ(4K) OFFLD3SZ(0K)
DSGROUP('CSQSMDS.SCEN1.*.SMDS') DSBLOCK(1M)
```

   Note the following:

   - Because SCEN1.Q is the only shared queue on the SCEN1 application structure, the **DSBLOCK** value has been set to 1M, the largest value possible. This should be the most efficient setting for our scenario.
   - Because the messages sent by the putting application are 30 KB, offloading to SMDS does not start until the second offload rule is met, when the structure is 80% full.

3. Run your test application again.

   Note the increased storage of messages on the queue.

4. Add 4 GB of SCM to structure IBM1SCEN1 by carrying out the following procedure:

   a) Check how much SCM is installed, and allocated to CF01, by issuing the following command:

```
D CF,CFNAME=CF01
```

   b) Check the STORAGE-CLASS MEMORY figures in the STORAGE CONFIGURATION section of the displayed output to see the available storage.

   c) Update the CFRM policy with the SCMMAXSIZE and SCMALGORITHM keywords as shown:

```
STRUCTURE
NAME(IBM1SCEN1)
SIZE(1024M)
INITSIZE(512M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
SCMMAXSIZE(4G)
SCMALGORITHM(KEYPRIORITY1)
```

5. Activate the CFRM policy by issuing the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=polname
```

6. Rebuild the IBM1SCEN1 structure.

   You must carry out this procedure because the structure was allocated when you made the previous changes.

   Issue the following command to rebuild the structure:

```
SETXCF START,REBUILD,STRNM=IBM1SCEN1
```

Results

You have successfully added SCM to your configuration.

What to do next

Optimize the performance of your system. See “Optimizing storage class memory usage” on page 203 for more information.

z/OS *Optimizing storage class memory usage*

How you improve your use of storage class memory (SCM).

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

Run the following command:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

As the structure was already full with message data, because of the previous tests, part of the rebuild involved pre-staging some of the messages from the structure into SCM. This process was initiated by using the previous command.

The output from this command produces, for example:

```
ACTIVE STRUCTURE
----------------
ALLOCATION TIME: 06/17/2014 09:28:50
CFNAME : CF01
COUPLING FACILITY: 002827.IBM.02.00000000B8D7
PARTITION: 3B CPCID: 00
STORAGE CONFIGURATION ALLOCATED MAXIMUM %
ACTUAL SIZE: 1024 M 1024 M 100
AUGMENTED SPACE: 3 M 142 M 2
STORAGE-CLASS MEMORY: 88 M 4096 M 2
ENTRIES: 120120 1089536 11
ELEMENTS: 240240 15664556 1
SPACE USAGE IN-USE TOTAL %
ENTRIES: 84921 219439 38
ELEMENTS: 2707678 3149050 85
EMCS: 2 282044 0
LOCKS: 1024
SCMHIGHTHRESHOLD : 90
SCMLOWTHRESHOLD : 70
ACTUAL SUBNOTIFYDELAY: 5000
PHYSICAL VERSION: CD5186A0 2BD8B85C
LOGICAL VERSION: CD515C50 CE2ED258
SYSTEM-MANAGED PROCESS LEVEL: 9
XCF GRPNAME : IXCLO053
DISPOSITION : KEEP
ACCESS TIME : NOLIMIT
MAX CONNECTIONS: 32
# CONNECTIONS : 1
CONNECTION NAME ID VERSION  SYSNAME JOBNAME  ASID STATE
--------------- -- -------  ------- -------  ---- ----------------
CSQEIBM1CSQ301  01 00010059 SC61    CSQ3MSTR 0091 ACTIVE
```

Note the following from the output of the command:

- That STORAGE_CLASS MEMORY provides confirmation that a **MAXIMUM** of 4096 MB of SCM has been added to the structure.
- The ALLOCATED figure for the amount of STORAGE-CLASS MEMORY used for pre-staging. There is now free space in the structure where there was none before SCM was added.
- The amount of AUGMENTED SPACE used to track SCM usage.
- The point at which the pre-staging algorithm starts to move data from the structure into SCM is when the structure is 90% full. This is indicated by the non-configurable **SCMHIGHTHRESHOLD** property.
- The point below which the prefetching algorithm starts to move data from SCM into the structure is when the structure is 70% full. This is indicated by the non-configurable **SCMLOWTHRESHOLD** property.

You can now test various ways to optimize the use of SCM. Note the following:

- After SCM is used to store messages, you cannot alter the structure until you have removed all the data from SCM.

  In this case, that means that the entry-to-element ratio is frozen at the value that was in place when SCM was first used. You must carefully ensure that the structure is in the state you want, before the pre-staging algorithm starts moving data into SCM.

- Is the current structure size correct before using SCM?

  For example, have you increased **INITSIZE** from 512 MB to a SIZE of 1 GB?

  If you do not do this, it is possible that although you enabled your structure for auto-alteration, the pre-staging algorithm will start to move data into SCM before the alteration has a chance to start. As a result, the structure is frozen using 512 MB of real storage.
- Is the entry-to-element ratio correct before using SCM?

  The goal of this scenario is to increase the number of offloaded message pointers that can be stored in the structure and SCM as a whole, as well as keeping as many messages entirely in structure storage as possible. Accessing these messages is faster than accessing messages on SMDS.

  Therefore, you need to have a structure that starts with an entry-to-element ratio that is good for storing messages, and then transitions to a ratio that is good for storing message pointers before the prestage algorithm first starts. This transition can be achieved, in part, by making use of the IBM MQ offload rules.

Change the offload rules by issuing the following command:

```
ALTER CFSTRUCT(SCEN1) OFFLD1SZ(0K)
```

You might have to carry out several runs to optimize the entry-to-element ratios.

The following table shows possible improvements in the number of messages put on the queue during the different phases of the emergency storage scenario.

Table 18. Comparison of results for the emergency storage scenario

Test description	Number of messages	Time to fill queue (seconds)
Basic configuration	27,850	3.2
SMDS with default offload rules	205,000	158
SCM with default offload rules	828,610	469
SCM with adjusted offload rules	1,135,775	679

The last row in the table shows that adjusting the offload rules had the required effect.

You need to examine your system to see if you can improve on these figures in any way. For example, you might run out of available SMDS storage. If you can allocate more SMDS storage you should be able to increase the number of messages on the queue quite significantly.

z/OS *Improved performance - basic configuration*

How you set up a basic scenario for improved performance using shared queues on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This scenario describes the use of SCM to increase the number of messages that can be stored on a shared queue without incurring the performance cost of using SMDS.

This initial scenario is very similar to that used for emergency storage and uses a:

- Queue sharing group, IBM1, that contains a single queue manager, CSQ3. In addition to the administration structure, the queue sharing group defined a single application structure, SCEN2.

- Coupling facility (CF) CF01, in which the SCEN2 application structure is stored as the IBM1SCEN2 structure. This structure has a maximum size of 2 GB.
- Single shared queue, SCEN2.Q, which is configured to use the application structure.

This configuration is illustrated in Figure 63 on page 205.

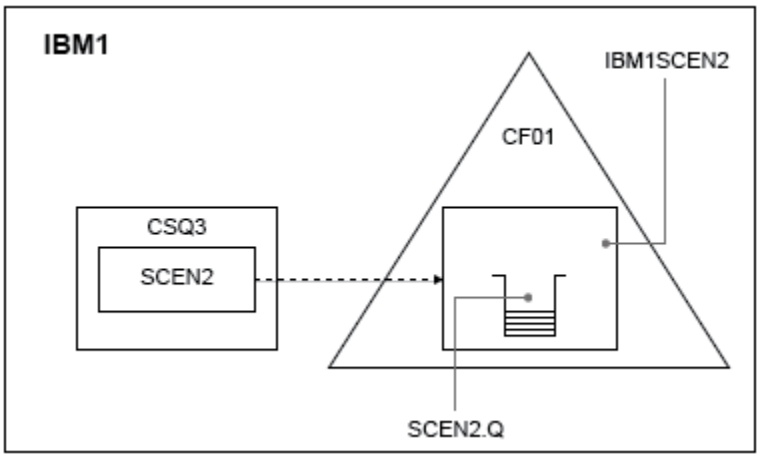


Figure 63. Basic configuration

Furthermore, assume that queue manager CSQ3 is already the only member of queue sharing group IBM1.

You must add the definition for structure IBM1SCEN2 to the coupling facility resource manager (CFRM) policy. For simplicity, the structure is defined so that it can be created in only a single coupling facility, CF01, by specifying PREFLIST(CF01).

Sample CFRM policy for structure IBM1SCEN2:

```
STRUCTURE
NAME(IBM1SCEN2)
SIZE(2048M)
INITSIZE(2048M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
```

Both the **INITSIZE** and **SIZE** keywords have the value 2048M so that the structure cannot resize.

Procedure

1. Refresh the CFRM policy by using the following command:

   ```
   SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN2
   ```

2. Verify that the structure has been created correctly, by using the following command:

   ```
   D XCF,STR,STRNAME=IBM1SCEN2
   ```

   Issuing the preceding command gives the following output:

   ```
   RESPONSE=SC61
   IXC360I 07.58.51 DISPLAY XCF 581
   STRNAME: IBM1SCEN2
   STATUS: NOT ALLOCATED
   POLICY INFORMATION:
   POLICY SIZE : 2048 M
   POLICY INITSIZE: 2048 M
   ```

```
POLICY MINSIZE : 1536 M
FULLTHRESHOLD : 85
ALLOWAUTOALT : YES
REBUILD PERCENT: N/A
DUPLEX : DISABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF01
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY

EVENT MANAGEMENT: MESSAGE-BASED   MANAGER SYSTEM NAME: SC53
MANAGEMENT LEVEL : 01050107
```

At this point, your structure has not been allocated, shown by the STATUS line, to the queue sharing group.

3. Configure IBM MQ to make use of the structure defined in the CFRM policy.
   a. Use the DEFINE CFSTRUCT command, with the structure name of SCEN2 to create an IBM MQ CFSTRUCT object.

```
DEFINE CFSTRUCT(SCEN2)
CFCONLOS(TOLERATE)
CFLEVEL(5)
DESCR('Structure for SCM scenario 2')
RECOVER(NO)
RECAUTO(YES)
OFFLOAD(DB2)
OFFLD1SZ(64K) OFFLD1TH(70)
OFFLD2SZ(64K) OFFLD2TH(80)
OFFLD3SZ(64K) OFFLD3TH(90)
```

   b. Check the structure, using the DISPLAY CFSTRUCT command.
   c. Define the SCEN2.Q shared queue, to use the SCEN2 structure, using the following MQSC command:

```
DEFINE QLOCAL(SCEN2.Q) QSGDISP(SHARED) CFSTRUCT(SCEN2) MAXDEPTH(999999999)
```

4. Use IBM MQ Explorer to put a single message to the queue SCEN2.Q and take the message off again.
5. Issue the following command to check that the structure is now allocated:

```
D XCF,STR,STRNAME=IBM1SCEN2
```

Review the output from the command, a portion of which is shown, and ensure that the STATUS line shows ALLOCATED.

```
RESPONSE=SC61
IXC360I 08.31.27 DISPLAY XCF 703
STRNAME: IBM1SCEN2
STATUS: ALLOCATED
EVENT MANAGEMENT: MESSAGE-BASED
TYPE: SERIALIZED LIST
POLICY INFORMATION:
POLICY SIZE : 2048 M
POLICY INITSIZE: 2048 M
POLICY MINSIZE : 1536 M
FULLTHRESHOLD : 85
ALLOWAUTOALT : YES
REBUILD PERCENT: N/A
DUPLEX : DISABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF01
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY
```

Additionally, note the values of the fields in the SPACE USAGE section:

- ENTRIES
- ELEMENTS

- EMCS
- LOCKS

An example of the values follows:

```
SPACE USAGE   IN-USE      TOTAL    %
ENTRIES:      344686     345242   99
ELEMENTS:    6548455    6548467   99
EMCS:              2     780318    0
LOCKS:          1024
```

Results

You have created the basic configuration. You can now obtain an idea of the baseline performance of your configuration using whatever method you select.

What to do next

You should test the basic scenario. As an example, you can use the following three applications, starting the applications in the order shown and, running them concurrently.

1. Use a PCF application to request the current depth (**CURDEPTH**) value for SCEN2.Q every five seconds. The output can be used to plot the depth of the queue over time.
2. A single threaded getting application repeatedly gets messages from SCEN2.Q, using a get with an infinite wait. To simulate processing of the messages that were removed, the getting application pauses for four milliseconds for every ten messages that it removed.
3. A single threaded putting application puts a total of one million 4 KB non-persistent messages to SCEN2.Q. This application does not pause between putting each message so messages are put on SCEN2.Q faster than the getting application can get them.

   As a result, when the putting application is running, the depth of SCEN2.Q increases.

   When structure IBM1SCEN2 is filled, and the putting application receives a MQRC_STORAGE_MEDIUM_FULL reason code, the putting application sleeps for five seconds before attempting to put the next message to the queue.

You can plot the results of the CURDEPTH application over a period of time. You obtain some form of saw-tooth wave output as the putting application pauses to allow the queue to partially empty.

Go to “Adding SCM to the initial structure” on page 207.

Related concepts

“Use of storage class memory with shared queues” on page 191

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

z/OS *Adding SCM to the initial structure*

How you add SCM for improved performance on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This part of the task uses the basic configuration described in “Improved performance - basic configuration” on page 204. The scenario describes the addition of SCM to the initial structure.

This final configuration is illustrated in Figure 64 on page 208.

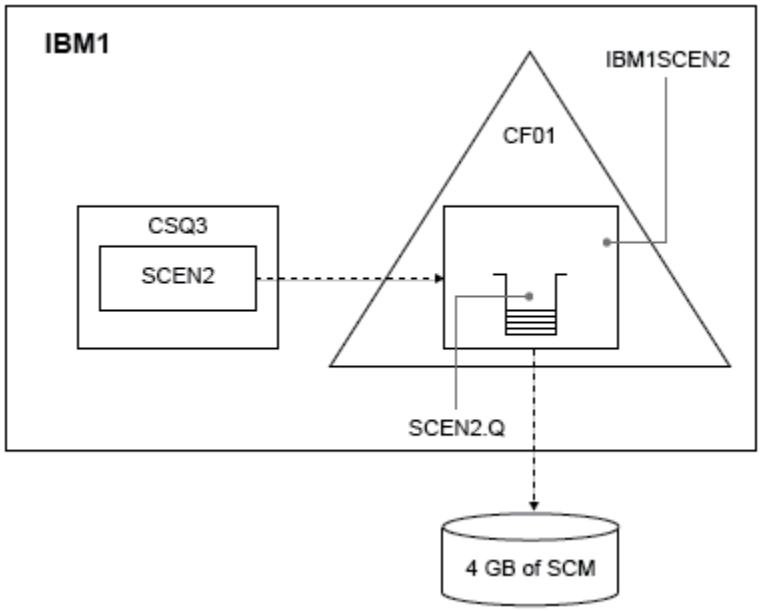


Figure 64. Configuration adding SCM for improved performance

Procedure

1. Add 4 GB of SCM to structure IBM1SCEN2 by carrying out the following procedure:
   a) Check how much SCM is installed, and allocated to CF01, by issuing the following command:

```
D CF,CFNAME=CF01
```

   b) Check the STORAGE-CLASS MEMORY figures in the STORAGE CONFIGURATION section of the displayed output to see the available storage.
   c) Update the CFRM policy with the SCMMAXSIZE and SCMALGORITHM keywords as shown:

```
STRUCTURE
NAME(IBM1SCEN2)
SIZE(2048M)
INITSIZE(2048M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
SCMMAXSIZE(4G)
SCMALGORITHM(KEYPRIORITY1)
```

2. Activate the CFRM policy by issuing the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN2
```

3. Rebuild the IBM1SCEN2 structure.

   You must carry out this procedure because the structure was allocated when you made the previous changes.

   Issue the following command to rebuild the structure:

```
SETXCF START,REBUILD,STRNM=IBM1SCEN2
```

4. Issue the following command to confirm the new configuration of the structure:

```
D XCF,STR,STRNAME=IBM1SCEN2
```

Review the output of the command, a portion of which follows:

```
SPACE USAGE     IN-USE       TOTAL      %
ENTRIES:          33         342684     0
ELEMENTS:         48        6503697     0
EMCS:              2         575600     0
LOCKS:                         1024
```

Results

Calculate the change in the use of real storage by the increase in control storage required to use SCM.

- Before SCM is added to the structure, the structure has these totals as shown in "Improved performance - basic configuration" on page 204:
  - 345,242 entries
  - 6,548,467 elements
  - 780,318 EMCS
- After SCM is added to the structure, the structure has these totals:
  - 342,684 entries
  - 6,503,697 elements
  - 575,600 EMCS

Using these figures, after the SCM was added, the structure is reduced in size by:

- 2558 entries
- 44,770 elements
- 204,718 EMCS

The amount of structure storage that is used to manage SCM, is as follows for a 2 GB structure with 4 GB of SCM allocated:

```
(2558 + 44,770 + 204,718) * 256 = 61.5 MB
```

Note that adding more SCM is likely to achieve only a marginal reduction of the size of the structure, because the amount of control storage used to track SCM increases, both as the structure size, and the amount of allocated SCM increases.

What to do next

Repeat the tests described in the final section of "Improved performance - basic configuration" on page 204.

You can plot the results of the revised application over a period of time. Comparing the plot to the one obtained previously, you now obtain an output without a saw-tooth wave, as the putting application no longer has to wait for the queue to partially empty.

For more information, refer to MP16: WebSphere MQ for z/OS - Capacity planning & tuning.

z/OS Distributed queuing and queue sharing groups

Distributed queuing and queue sharing groups are two techniques that you can use to increase the availability of your application systems. Use this topic to find further information about these techniques.

To complement the high availability of messages on shared queues, the distributed queuing component of IBM MQ has additional functions to provide the following:

- Higher availability to the network.
- Increased capacity for inbound network connections to the queue sharing group.

Figure 65 on page 210 illustrates distributed queuing and queue sharing groups. It shows two queue managers within a sysplex, both of which belong to the same queue sharing group. They can both access shared queue SQ1. Queue managers in the network (on AIX and Windows for example) can put messages onto this queue through the channel initiator of either queue manager. Cloned applications on both queue managers service the queue.

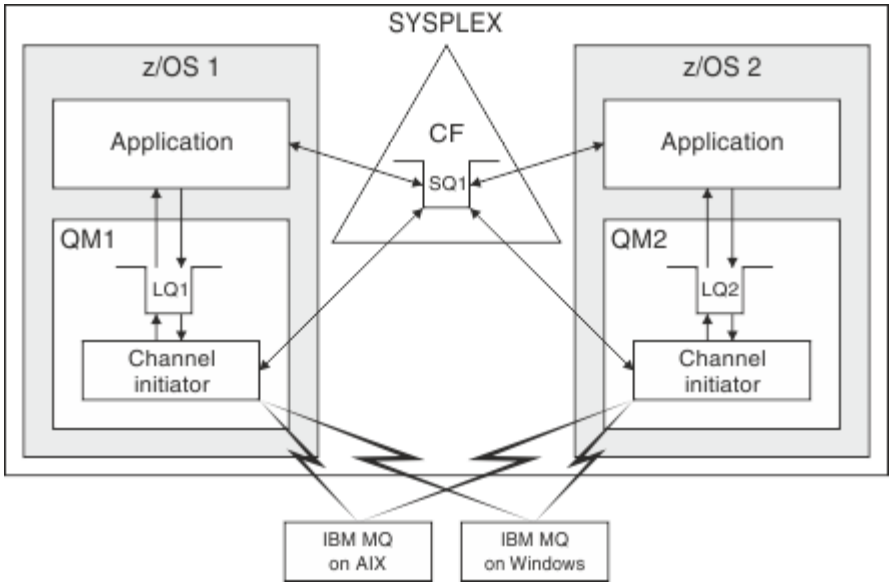


Figure 65. Distributed queuing and queue sharing groups

Related concepts

"Shared channels" on page 210
Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

"Intra-group queuing" on page 215
This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

"Clusters and queue sharing groups" on page 212
Use this topic to understand how you can use queue sharing groups with clusters.

z/OS *Shared channels*

Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

A number of networking products provide a mechanism to hide server failures from the network, or to balance inbound network requests across a set of eligible servers. The network products make a *generic port* available for inbound network connection requests, and the inbound request can be satisfied by connecting to one of the eligible servers.

These networking products include:

- VTAM generic resources
- SYSPLEX Distributor

The channel initiator takes advantage of these products to use the capabilities of shared queues

There are two types of shared channels, *shared inbound channel*, and the *shared outbound channel*.

- Shared inbound channels
- Shared outbound channels

For further information about channels see

- Shared channel summary
- Shared channel status

Shared inbound channels

Each channel initiator in the queue sharing group starts an additional listener task to listen on a *generic port*. This generic port is made available to the network by one of the supporting technologies (VTAM, TCP/IP). Inbound network attach requests to the generic port are dispatched by the network technology, to any one of the listeners in the queue sharing group (QSG) that are listening on the generic port.

You can start a channel on the channel initiator to which the inbound attach is directed if the channel initiator has access to a channel definition for a channel with that name. You can define a channel definition to be private to a queue manager or stored on the shared repository and so available anywhere (a global definition). This means that you can make a channel definition available on any channel initiator in the queue sharing group by defining it as a global definition.

There is an additional difference when starting a channel through the generic port; channel synchronization is with the queue sharing group and not with an individual queue manager. For example, consider a remote queue manager starting a channel through the generic port. When the channel first starts, it might start on queue manager QM1 and messages flow. If the channel stops and is restarted on queue manager QM2, information about the number of messages that have flowed is still correct because the synchronization is with the queue sharing group.

You can use an inbound channel started through the generic port to put messages to any queue. The remote queue manager does not know whether the target queue is shared or not. If the target queue is a shared queue, the remote queue manager connects through any available channel initiator in a load-balanced fashion and the messages are put to the shared queue.

If the target queue is a private queue, the messages are put to the private queue owned by which ever queue manager the current instance of the channel is connected to. In this environment, known as *replicated local queues*, each queue manager must have the same set of private queues defined.

Configuring SVRCONN channels for a queue sharing group

The optimal configuration for SVRCONN channels in a queue sharing group is to set up private listeners in each CHINIT which use a different port number from the point to point channels. These listener ports are then used as the 'back-end' resources for a new workload distribution mechanism such as Sysplex Distributor using Virtual IP addresses (VIPA). The external VIPA address is then used as the target address for the CLNTCONN definitions in the network. The SVRCONN channel can be defined with QSGDISP(GROUP) so the same definition is available to all queue managers in the QSG. This configuration avoids using a shared listener, and therefore reduces the performance effect of the queue sharing group maintaining shared channel state, which is not needed for client/server channels.

Shared outbound channels

An outbound channel is considered to be a shared channel if it is taking messages from a shared transmission queue. If it is shared, it holds synchronization information at queue sharing group level. This means that the channel can be restarted on a different queue manager and channel initiator instance within the queue sharing group if the communications subsystem, channel initiator, or queue manager fails. Restarting failed channels in this way is a feature of shared channels called *peer channel recovery*.

Workload balancing for shared outbound channels

An outbound shared channel is eligible for starting on any channel initiator within the queue sharing group, if you have not specified that you want it to be started on a particular channel initiator. The channel initiator selected by IBM MQ is determined using the following criteria:

- Is the communications subsystem required currently available to the channel initiator?
- Is a Db2 connection available to the channel initiator?
- Which channel initiator has the lowest current workload? The workload includes channels that are active and retrying.

Shared channel summary

Shared channels differ from private channels in the following ways:

Private channel
Tied to a single channel initiator.

- Outbound channel uses a local transmission queue.
- Inbound channel started through a local port.
- Synchronization information held in SYSTEM.CHANNEL.SYNCQ queue.

Shared Channel
Workload balanced with high availability.

- Outbound channel uses a shared transmission queue.
- Inbound channel started through a generic port.
- Synchronization information held in SYSTEM.QSG.CHANNEL.SYNCQ queue.

You specify whether a channel is private or shared when you start the channel by using CHLDISP options with the START CHANNEL command. A shared channel can be started by triggering in the same way as a private channel. However, when a shared channel is started, IBM MQ performs workload balancing and starts the channel on the most appropriate channel initiator within the queue sharing group. (If required, you can specify that a shared channel is to be started on a particular channel initiator.)

Shared channel status

The channel initiators in a queue sharing group maintain a shared channel-status table in Db2. This records which channels are active on which channel initiators. The shared channel-status table is used if there is a channel initiator or communications system failure. It indicates which channels need to be restarted on a different channel initiator in the queue sharing group.

z/OS *Clusters and queue sharing groups*

Use this topic to understand how you can use queue sharing groups with clusters.

You can make your shared queues available to a cluster in a single definition. To do this you specify the name of the cluster when you define the shared queue.

Users in the network see the shared queue as being hosted by each queue manager within the queue sharing group (the shared queue is not advertised as being hosted by the queue sharing group). Clients can start sessions with any members of the queue sharing group to put messages to the same shared queue.

Figure 66 on page 213 shows how members of a cluster can access a shared queue through any member of the queue sharing group.

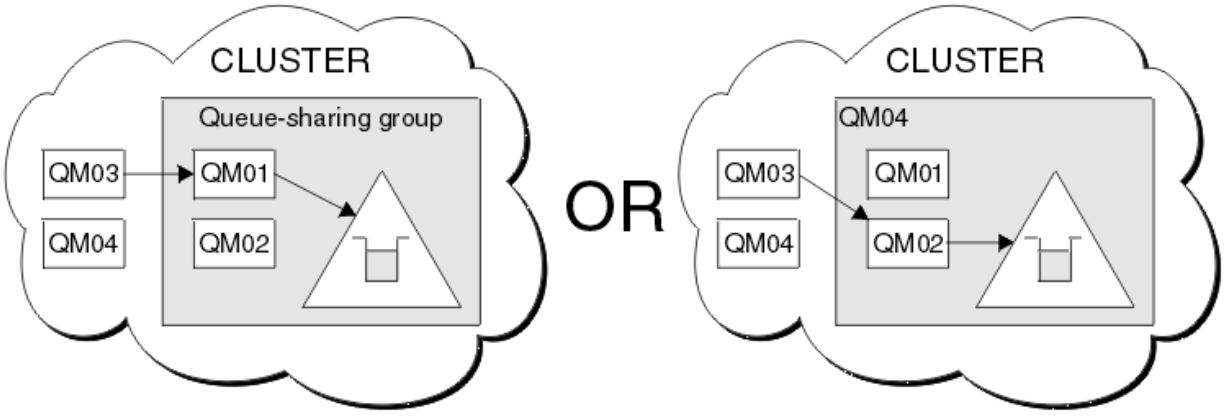


Figure 66. A queue sharing group as part of a cluster

z/OS Influencing workload distribution with shared queues

Use this topic to understand the factors that affect workload distribution with shared queues in a queue sharing group.

IBM MQ does not provide workload balancing for shared queues. However, workload distribution in a queue sharing group (QSG) can be influenced in *a pull based fashion*. The choice of which queue manager services a queue (receives a message written to a shared queue) is affected by the available processing capacity of each queue manager in the queue sharing group and the workload management goals defined across the sysplex.

However, it is important to appreciate, the queue manager that performs the MQPUT of a message can also have a large influence in deciding which queue manager gets the message.

A local queue manager is more likely to perform the MQGET

For an application performing an MQPUT, the local queue manager is said to be the queue manager to which the application is connected.

Exactly which queue manager services an MQPUT of a message by performing an MQGET on behalf of a getting application is influenced by the following considerations.

When a message is put to an empty shared queue, the local queue manager is typically posted before any of the other queue manager in the queue sharing group is notified. If the local queue manager is in a position to process the message, it receives a list transition notification from the coupling facility (CF) before any other queue manager in the QSG. (A list transition notification is a notification that the shared queue has changed state from empty to non-empty.)

The possible scenarios, in this case, are as follows:

1. MQPUT of nonpersistent message out of sync point and *fast put to waiting getter*.

   If there is an application with an *MQGET with wait* on the local queue manager for the queue, then the MQPUT of the message is passed directly to the getting application's buffer and not written to the queue. This is true for shared and non-shared queues. This feature is often called *fast put to a waiting getter* mechanism. In the case of shared queues, no other queue manager in the QSG is notified as there is no transition from empty to non-empty of the queue. This means, for example, that provided this queue manager can service all the puts from this application and assuming that no other applications are putting messages to the queue, then no other queue manager in the queue sharing group assists in draining this queue. If however there is no MQGET with wait on the local queue manager, and a message is put to the shared queue then the CF will notify other queue managers in the queue sharing group according to its rules for notifications of list transitions.
2. MQPUT of a persistent or in-syncpoint message.

In this case, if there is an application with an *MQGET with wait* on the local queue manager, then the message is put to the shared queue and the CF notifies other queue managers in the queue sharing group according to its rules for notifications of list transitions. However, the local queue manager does not wait for a transition notification from the CF but honors any local *MQGET with wait* first and usually performs the get of this message on behalf of the application before any other queue manager in the queue sharing group can respond to a CF notification. This is dependent on how busy the local queue manager is. Otherwise, any queue manager notified by the CF due to the arrival of the message on the empty queue will try to service the get first. The first queue manager to respond processes the new message.

3. Finally, if the queue is not drained of messages, where the CF has sent a notification of a state change from empty to non-empty for the queue, all connected queue managers will have an opportunity to assist in the processing of the queue. In this event, the workload is said to be *pull based*.

This design allows for the improved performance over a purely pull based workload distribution. The aim is to take advantage of the high availability offered by queues held in the CF while allowing the queue manager, where possible, to perform the MQGET without needing to reference the CF and so to process the message workload as efficiently as possible.

Alternative approaches can be adopted where emphasis on balance of the workload is more important than the previously described performance enhancements. For example, ensuring that none of the getting applications are connected to the same queue manager that the putting application is connected to. Using this design all messages are put to the queue and all queue managers in the QSG are notified when the queue moves from empty to non-empty, in accordance with the CF algorithm for handling such transitions. In addition, the *fast put to waiting getter* mechanism is not applicable.

z/OS Where to find more information about shared queues and queue sharing groups

Use the table in this topic to find more information about how IBM MQ for z/OS uses shared queues and queue sharing groups.

Table 19. Where to find more information about shared queues and queue sharing groups

Topic	Where to look
Queue sharing group recovery	“Recovery and restart on z/OS” on page 253
Queue sharing group security	“Security concepts in IBM MQ for z/OS” on page 269
Private and global object definitions Directing commands to different queue managers	Sources from which you can issue commands on z/OS
Planning your coupling facility environment	Defining coupling facility resources
Planning your SMDS environment	Planning your shared message data set (SMDS) environment
Planning your Db2 environment	Planning your Db2 environment
Setting up your shared queues System parameters	“Shared queues and queue sharing groups” on page 171
Utility programs Migrating queues	IBM MQ utilities on z/OS reference

Table 19. Where to find more information about shared queues and queue sharing groups (continued)	
Topic	**Where to look**
Console messages	Messages for IBM MQ for z/OS
MQSC commands	MQSC commands
IBM MQ clusters	Configuring a queue manager cluster
IBM MQ distributed queuing Channel names	Introduction to distributed queue management
Writing applications	Overview of application design
MQCONNX call	MQCONNX

z/OS Intra-group queuing

This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

For information about queue sharing groups, see “Shared queues and queue sharing groups” on page 171.

Intra-group queuing concepts

You can perform fast message transfer between queue managers in a queue sharing group without defining channels. This uses a system queue called the SYSTEM.QSG.TRANSMIT.QUEUE, which is a shared transmission queue. Each queue manager in the queue sharing group starts a task called the intra-group queuing agent, which waits for messages to arrive on this queue that are destined for their queue manager. When such a message is detected, it is removed from the queue and placed on the correct destination queue.

Standard name resolution rules are used but, if intra-group queuing (IGQ) is enabled and the target queue manager is within the queue sharing group, the SYSTEM.QSG.TRANSMIT.QUEUE is used to transfer the message to the correct destination queue manager instead of using a transmission queue and channel.

You enable intra-group queuing through a queue manager attribute. Intra-group queuing moves nonpersistent messages outside sync point, and persistent messages within sync point. If it finds a problem delivering messages to the target queue, intra-group queuing tries to put them to the dead-letter queue. If the dead-letter queue is full or undefined, nonpersistent messages are discarded, but persistent messages are backed out and returned to the SYSTEM.QSG.TRANSMIT.QUEUE, and the IGQ agent tries to deliver the messages until it is successful.

An inbound shared channel that receives a message destined for a queue on a different queue manager in the queue sharing group can use intra-group queuing to *hop* the message to the correct destination.

There might be times when you want the local queue manager to put a message directly to the target queue if the target queue is a shared queue, rather than the message first being transferred to the target queue manager. You can use the queue manager attribute SQQMNAME to control this. If you set the value of SQQMNAME to USE, the MQOPEN command is performed on the queue manager specified by the **ObjectQMgrName**.

However, if the target queue is a shared queue and you set the value of SQQMNAME to IGNORE, and the **ObjectQMgrName** is that of another queue manager in the queue sharing group, the shared queue is opened on the local queue manager. If the local queue manager cannot open the target queue, or put a

message to the queue, the message is transferred to the specified **ObjectQMgrName** through either IGQ or an IBM MQ channel.

Intra-group queuing can be used to deliver, more efficiently, small messages to queues residing on remote queue managers within a queue sharing group. Intra-group queuing also supports large messages, the largest being 100 MB *minus* the length of the transmission queue header.

Note: If you use this feature, users must have the same access to the queues on each queue manager in the queue sharing group.

The following diagram shows a typical example of intra-group queuing.

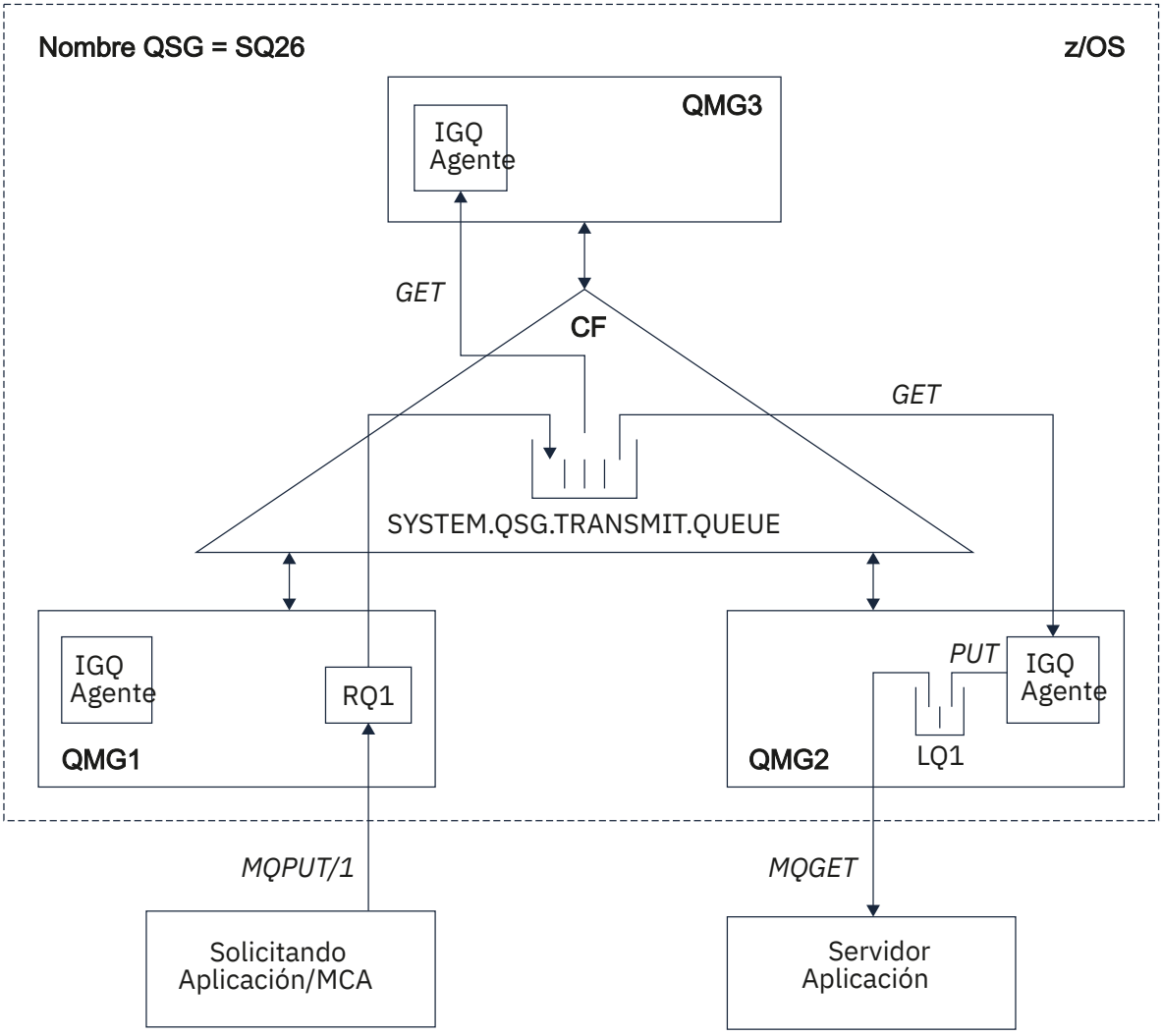


Figure 67. An example of intra-group queuing

The diagram shows:

- IGQ agents running on three queue managers (QMG1, QMG2, and QMG3) that are defined to a queue sharing group called SQ26.
- Shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE that is defined in the coupling facility (CF).
- A remote queue definition that is defined in queue manager QMG1.
- A local queue that is defined in queue manager QMG2.
- A requesting application (this application could be a Message Channel Agent (MCA)) that is connected to queue manager QMG1.

- A server application that is connected to queue manager QMG2.
- A request message being placed on to the SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing and the intra-group queuing agent

An IGQ agent is started during queue manager initialization. When applications open and put messages to remote queues, the local queue manager determines whether intra-group queuing is used for message transfer. If intra-group queuing is to be used, the local queue manager places the message on to the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent on the target remote queue manager retrieves the message and places it on to the destination queue.

Intra-group queuing terminology

Explanations of the terminology: intra-group queuing, shared transmission queue for use by intra-group queuing, and intra-group queuing agent.

Intra-group queuing

Intra-group queuing can effect potentially fast and less expensive message transfer between queue managers in a queue sharing group, without the need to define channels.

Shared transmission queue for use by intra-group queuing

Each queue sharing group has a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE for use by intra-group queuing. If intra-group queuing is enabled, SYSTEM.QSG.TRANSMIT.QUEUE appears in the name resolution path when opening remote queues. When applications (including Message Channel Agents (MCAs)) put messages to a remote queue, the local queue manager determines the eligibility of messages for fast transfer and places them on SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing agent

The IGQ agent is the task, started at queue manager initialization, that waits for suitable messages to arrive on the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent retrieves suitable messages from this queue and delivers them to the destination queues.

The IGQ agent for each queue manager is always started because intra-group queuing is used by the queue manager itself for its own internal processing.

z/OS Benefits of intra-group queuing

The benefits of intra-group queuing are: reduced system definitions, reduced system administration, improved performance, supports migration, and delivery of messages when multi-hopping between queue managers in a queue sharing group.

The benefits of intra-group queuing are:

Reduced system definitions
Intra-group queuing removes the need to define channels between queue managers in a queue sharing group.

Reduced system administration
Because there are no channels defined between queue managers in a queue sharing group, there is no requirement for channel administration.

Improved performance
Because there is only one IGQ agent needed for the delivery of a message to a target queue (instead of two intermediate sender and receiver agents), the delivery of messages using intra-group queuing can be less expensive than the delivery of messages using channels. In intra-group queuing there is only a receiving component, because the need for the sending component has been removed. This saving is because the message is available to the IGQ agent at the destination queue manager for

delivery to the destination queue once the put operation at the local queue manager has completed and, in the case of messages put in sync point scope, committed.

Supports migration

Applications external to a queue sharing group can deliver messages to a queue residing on any queue manager in the queue sharing group, while being connected only to a particular queue manager in the queue sharing group. This is because messages arriving on a receiver channel, destined for a queue on a remote queue manager, can be transparently sent to the destination queue using intra-group queuing. This facility allows applications to be deployed among the queue sharing group without the need to change any systems that are external to the queue sharing group.

A typical configuration is illustrated by the following diagram, in which:

- A requesting application connected to queue manager QMG1 needs to send a message to a local queue on queue manager QMG3.
- Queue manager QMG1 is connected only to queue manager QMG2.
- Queue managers QMG2 and QMG3, which were previously connected using channels, are now members of queue sharing group SQ26.

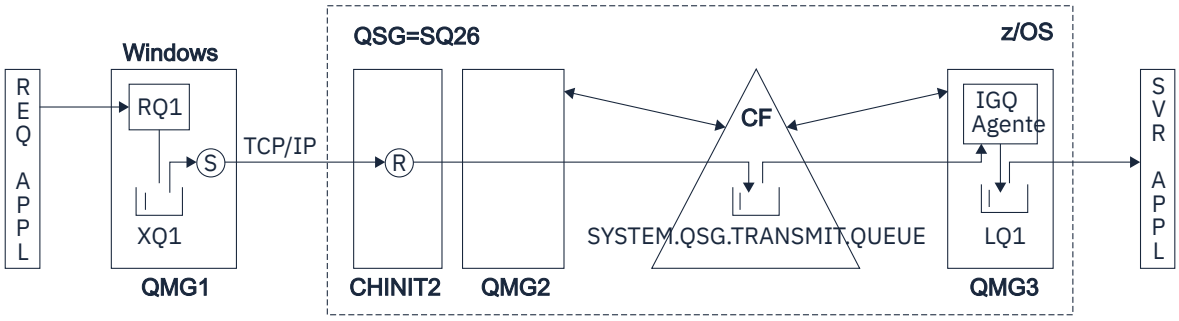


Figure 68. An example of migration support

The flow of operations is as follows:

1. The requesting application puts a message, destined for local queue LQ1 at remote queue manager QMG3, on to remote queue definition RQ1.
2. Queue manager QMG1, running on a Windows NT workstation, places the message on to the transmission queue XQ1.
3. Sender MCA (S) on QM1 transmits the message, using TCP/IP, to the receiver MCA (R) on channel initiator CHINIT2.
4. Receiver MCA (R) on channel initiator CHINIT2 places the message on to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
5. IGQ agent on queue manager QMG3 retrieves the message from the SYSTEM.QSG.TRANSMIT.QUEUE and places it on to the target local queue LQ1.
6. The server application retrieves the message from the target local queue and processes it.

Delivery of messages when multi-hopping between queue managers in a queue sharing group

The previous diagram in Supports migration also illustrates the delivery of messages when multi-hopping between queue managers in a queue sharing group. Messages arriving on a queue manager within the queue sharing group, but destined for a queue on another queue manager in the queue sharing group, can be easily transmitted to the destination queue on the destination queue manager, using intra-group queuing.

z/OS Limitations of intra-group queuing

The limitations of intra-group queuing are: messages eligible for transfer using intra-group queuing, number of intra-group queuing agents per queue manager, and starting and stopping the intra-group queuing agent.

This topic describes the limitations of intra-group queuing.

Messages eligible for transfer using intra-group queuing
Because intra-group queuing uses a shared transmission queue that is defined in the coupling facility (CF), intra-group queuing is limited to delivering messages of the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Number of intra-group queuing agents per queue manager
Only one IGQ agent is started per queue manager in a queue sharing group.

Starting and stopping the intra-group queuing agent
The IGQ agent is started during queue manager initialization and terminated during queue manager shut-down. It is designed to be a long running, self recovering (in the event of abnormal termination), task. If there is an error with the definition of the SYSTEM.QSG.TRANSMIT.QUEUE (for example, if this queue is Get inhibited) the IGQ agent keeps retrying. If the IGQ agent encounters an error that results in normal termination of the agent while the queue manager is still active, it can be restarted by issuing an ALTER QMGR IGQ(ENABLED) command. This command avoids the need to recycle the queue manager.

Setting the queue manager attribute IGQ to ENABLED or DISABLED
If the queue manager attribute IGQ is set to ENABLED or DISABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See Getting started with intra-group queueing for more information.

z/OS Getting started with intra-group queuing

You can enable, disable, and use intra-group queuing as described in this topic.

Enabling intra-group queuing
To enable intra-group queuing on your queue managers, you need to do the following:

- Define a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE. The definition of this queue can be found in thlqual.SCSQPROC(CSQ4INSS), the CSQINP2 sample for SYSTEM objects for queue sharing groups. This queue must be defined with the correct attributes, as stated in thlqual.SCSQPROC(CSQ4INSS), for intra-group queuing to work properly.
- Because the IGQ agent is always started at queue manager initialization, intra-group queuing is always available for inbound message processing. The IGQ agent processes any messages that are placed on the SYSTEM.QSG.TRANSMIT.QUEUE. However, to enable intra-group queuing for outbound processing, the queue manager attribute IGQ must be set to ENABLED.

  Important: If the queue manager attribute IGQ is set to ENABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See “Specific properties of intra-group queuing” on page 227 for more information. As described in the 'Programmer response' section for this reason code, applications need to be coded to handle this situation (see 2041 (07F9) (RC2041): MQRC_OBJECT_CHANGED for more details).

Additionally, as IGQ is designed as a long running and self-recovering task, which starts during initialization and terminates with shutdown, see “Limitations of intra-group queuing” on page 219 for further information.

Disabling intra-group queuing

To disable intra-group queuing for outbound message transfer, set the queue manager attribute IGQ to DISABLED. If intra-group queuing is disabled for a particular queue manager, the IGQ agent on that queue manager can still process inbound messages that have been placed on the SYSTEM.QSG.TRANSMIT.QUEUE by a queue manager that does have intra-group queuing enabled for outbound transfer.

Important: If the queue manager attribute IGQ is set to ENABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See “Specific properties of intra-group queuing” on page 227 for more information. As described in the 'Programmer response' section for this reason code, applications need to be coded to handle this situation (see 2041 (07F9) (RC2041): MQRC_OBJECT_CHANGED for more details).

Additionally, as IGQ is designed as a long running and self-recovering task, which starts during initialization and terminates with shutdown, see “Limitations of intra-group queuing” on page 219 for further information.

Using intra-group queuing

Once intra-group queuing is enabled, it is available for use and a queue manager uses it whenever possible. That is, when an application puts a message to a remote queue definition, to a fully qualified remote queue, or to a cluster queue, the queue manager determines if the message is eligible to be delivered using intra-group queuing and if it is, places the message on to SYSTEM.QSG.TRANSMIT.QUEUE. There is no need to change user applications, or to application queues, because for eligible messages the queue manager uses the SYSTEM.QSG.TRANSMIT.QUEUE, in preference to any other transmission queue.

z/OS Intra-group queuing configurations

In addition to the typical intra-group queuing configuration, other configurations are possible.

“Intra-group queuing concepts” on page 215 describes the typical configuration.

Related concepts

“Distributed queuing with intra-group queuing (multiple delivery paths)” on page 220
For applications that process short messages it might be feasible to configure intra-group queuing only for delivering messages between queue managers in a queue sharing group.

“Clustering with intra-group queuing (multiple delivery paths)” on page 222
It is possible to configure queue managers so that they are in a cluster as well as in a queue sharing group.

“Clustering, intra-group queuing and distributed queuing” on page 224
It is possible to configure a queue manager that is a member of a cluster as well as a queue sharing group and is connected to a distributed queue manager using a sender/receiver channel pair.

z/OS *Distributed queuing with intra-group queuing (multiple delivery paths)*

For applications that process short messages it might be feasible to configure intra-group queuing only for delivering messages between queue managers in a queue sharing group.

The choice of intra-group queuing over channel communications can be controlled by the CFSTRUCT type level. (3 instead of 4 or 5). The maximum message length as set on the SYSTEM.QSQ.TRANSMIT.QUEUE.

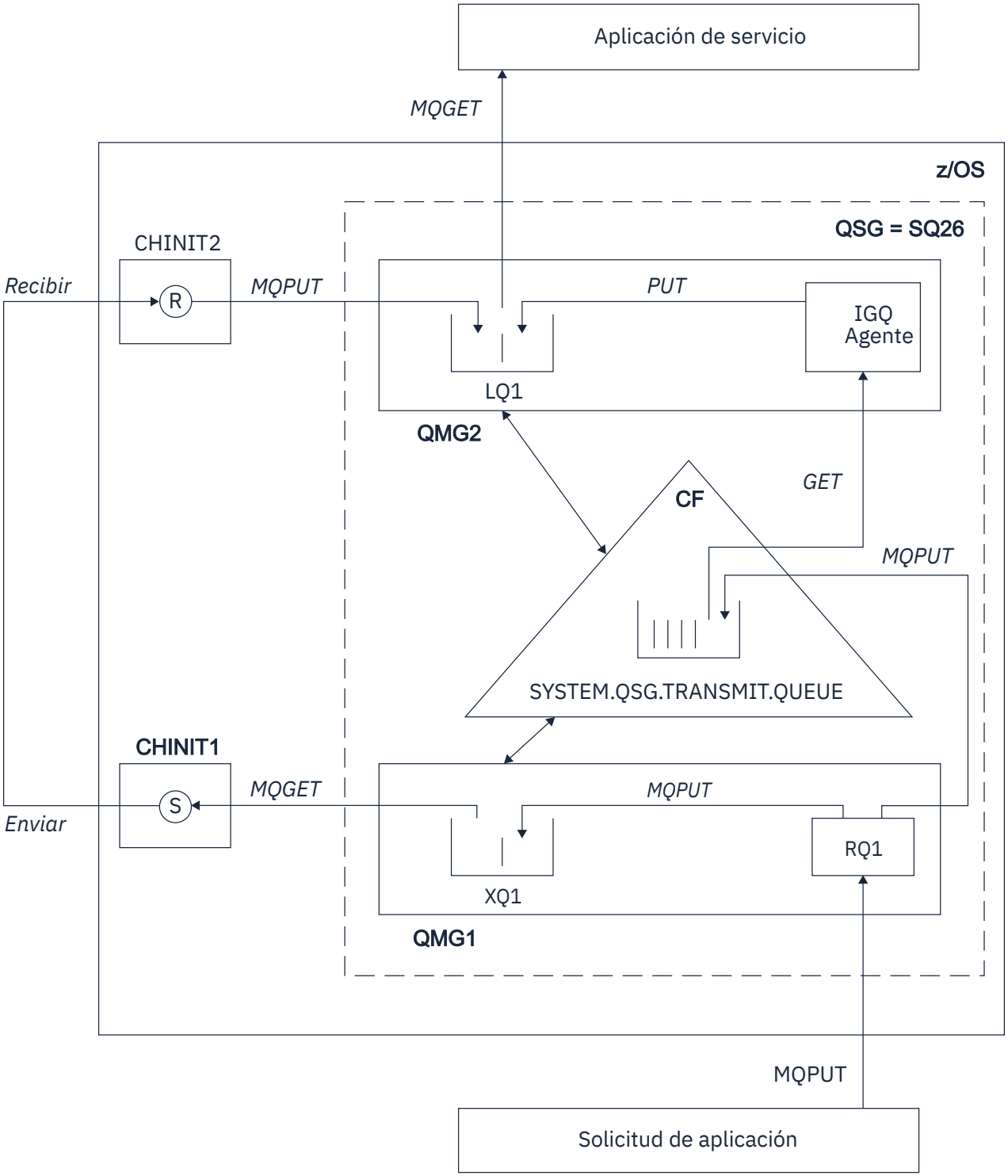


Figure 69. An example configuration

Open/Put processing

1. It is important to note that when the requesting application opens remote queue RQ1, name resolution occurs for both the non-shared transmission queue XQ1 and the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
2. When the requesting application puts a message on to the remote queue, based on whether intra-group queuing is enabled for outbound transfer on the queue manager and on the message characteristics, the message is put to transmission queue XQ1, or to transmission queue

SYSTEM.QSG.TRANSMIT.QUEUE. The queue manager places all large messages on to transmission queue XQ1, and all small messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.

3. If transmission queue XQ1 is full, or is not available, put requests for large messages fail synchronously with a suitable return and reason code. However, put requests for small messages continue to succeed and are placed on transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
4. If transmission queue SYSTEM.QSG.TRANSMIT.QUEUE is full, or cannot be put to, put requests for small messages fail synchronously with a suitable return and reason code. However, put requests for large messages continue to succeed and are placed on transmission queue XQ1. In this case, no attempt is made to put the small messages on to a transmission queue.

Flow for large messages

1. The requesting application puts large messages to remote queue RQ1.
2. Queue manager QMG1 puts the messages on to transmission queue XQ1.
3. Sender MCA (S) on queue manager QMG1 retrieves the messages from transmission queue XQ1 and sends them to queue manager QMG2.
4. Receiver MCA (R) on queue manager QMG2 receives the messages and places them on to destination queue LQ1.
5. The serving application retrieves and then processes the messages from queue LQ1.

Flow for small messages

1. The requesting application puts small messages on to remote queue RQ1.
2. Queue manager QMG1 puts the messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
3. IGQ on queue manager QMG2 retrieves the messages and places them on to the destination queue LQ1.
4. The serving application retrieves the messages from queue LQ1.

Points to note

1. The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
2. A potentially faster message delivery mechanism can be achieved for small messages.
3. Multiple paths are available for message delivery (that is, the normal channel route and the intra-group queuing route).
4. The intra-group queuing route, being potentially faster, is selected in preference to the normal channel route. Depending on the message characteristics, message delivery might be divided across the two paths. Hence, messages might be delivered out of sequence (though this delivery is also possible if messages are delivered using only the normal channel route).
5. When a route has been selected, and messages have been placed on to the transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to transmission queue XQ1.

z/OS *Clustering with intra-group queuing (multiple delivery paths)*

It is possible to configure queue managers so that they are in a cluster as well as in a queue sharing group.

When messages are sent to a cluster queue and the local and remote destination queue managers are in the same queue sharing group, intra-group queuing is used for the delivery of small messages (using the SYSTEM.QSG.TRANSMIT.QUEUE), and the delivery of large messages if intra-group queuing supports the size of the message. Also, the SYSTEM.CLUSTER.TRANSMIT.QUEUE is used for the delivery of messages to any queue manager that is in the cluster, but outside the queue sharing group. The following diagram illustrates this configuration (the channel initiators are not shown).

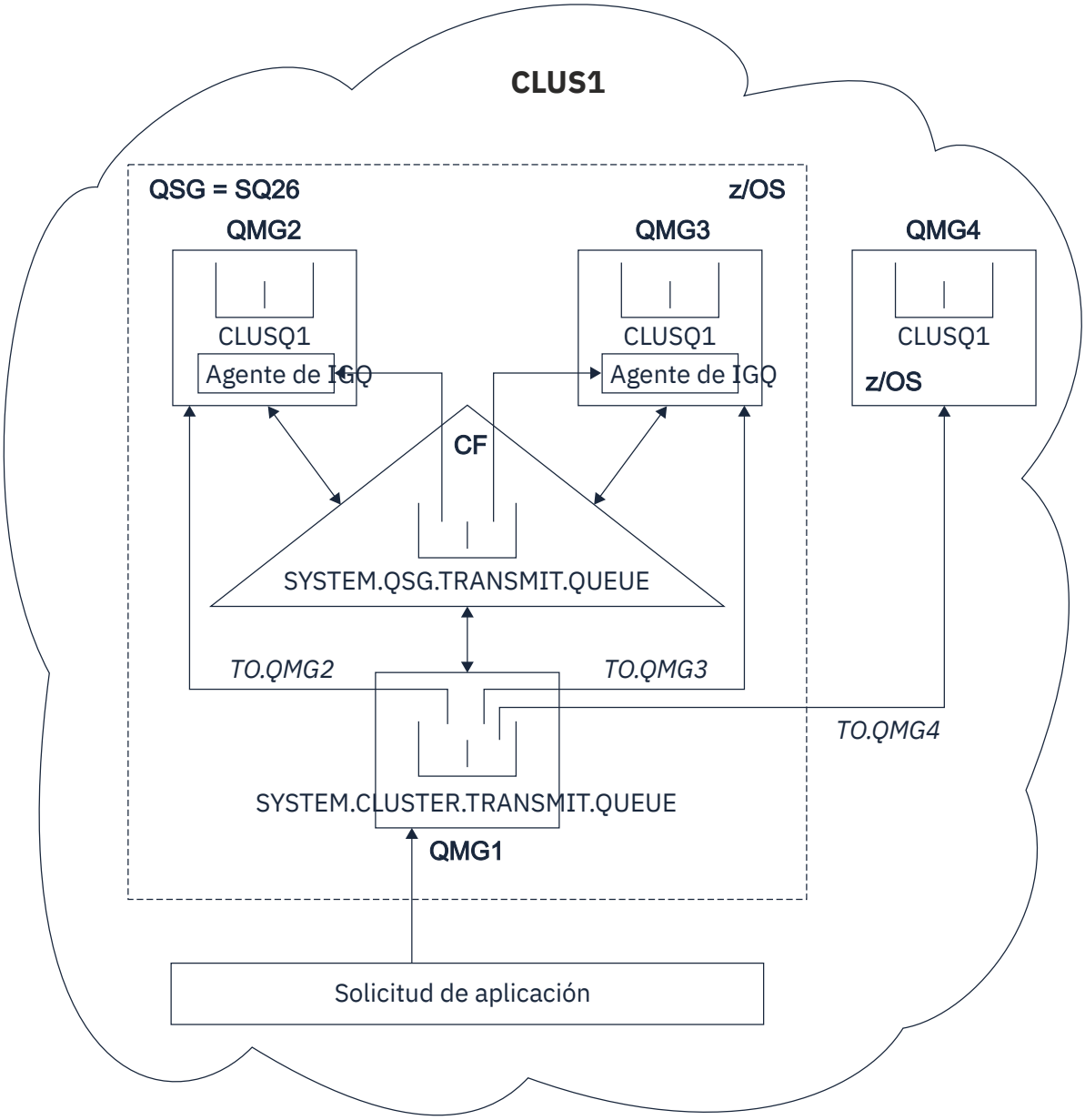


Figure 70. An example of clustering with intra-group queuing

The diagram shows:

- Four z/OS queue managers QMG1, QMG2, QMG3, and QMG4 configured in a cluster CLUS1.
- Queue managers QMG1, QMG2, and QMG3 configured in a queue sharing group SQ26.
- IGQ agents running on queue managers QMG2 and QMG3.
- The local SYSTEM.CLUSTER.TRANSMIT.QUEUE defined in QMG1.

  Note: For clarity, the SYSTEM.CLUSTER.TRANSMIT.QUEUE on the other queue managers not shown.
- The shared SYSTEM.QSG.TRANSMIT.QUEUE defined in the CF, which is in an IBM MQ structure configured with the CFLEVEL(3) RECOVER(YES) attribute.
- Cluster channels TO.QMG2 (connecting QMG1 to QMG2), TO.QMG3 (connecting QMG1 to QMG3), and TO.QMG4 (connecting QMG1 to QMG4).
- Cluster queue CLUSQ1 being hosted on queue managers QMG2, QMG3, and QMG4.

Assume that the requesting application opens the cluster queue with the MQOO_BIND_NOT_FIXED option, so that the target queue manager for the cluster queue is selected at put time.

If the selected target queue manager is QMG2:

- All large messages put by the requesting application are:
  - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1, because SYSTEM.QSG.TRANSMIT.QUEUE is in a CFLEVEL(3) structure; therefore supports messages only up to 63 KB in size.
  - Transferred to cluster queue CLUSQ1 on QMG2 using cluster channel TO.QMG2
- All small messages put by the requesting application are
  - Put to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE. This queue is in a structure configured with the RECOVER(YES) attribute, so is used for both persistent, and non-persistent, small messages.
  - Retrieved by the IGQ agent on QMG2
  - Put to the cluster queue CLUSQ1 on QMG2

If the selected target queue manager is QMG4:

- Because QMG4 is not a member of queue sharing group SQ26, all messages put by the requesting application are
  - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1
  - Transferred to cluster queue CLUSQ1 on QMG4 using cluster channel TO.QMG4

Points to note

- The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
- A potentially faster delivery mechanism is achieved for the transfer of small non-persistent messages between queue managers in a queue sharing group (even if the same queue managers are in a cluster).
- Multiple paths are available for message delivery (that is, both the cluster route and the intra-group queuing route).
- The intra-group queuing route, being potentially faster, is selected in preference to the cluster route. Depending on the message characteristics, message delivery might be divided across the two paths. Hence, messages might be delivered out of sequence. It is important to note that this delivery is possible without regard to the MQOO_BIND_* option specified by the application. Intra-group queuing distributes messages in the same way as clustering does, depending on whether the MQOO_BIND_NOT_FIXED, MQOO_BIND_ON_OPEN, MQOO_BIND_ON_GROUP, or MQOO_BIND_AS_Q_DEF is specified on open.
- When a route has been selected, and messages have been placed on to the transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

z/OS *Clustering, intra-group queuing and distributed queuing*

It is possible to configure a queue manager that is a member of a cluster as well as a queue sharing group and is connected to a distributed queue manager using a sender/receiver channel pair.

This configuration is a combination of distributed queuing with intra-group queuing and clustering with intra-group queuing.

Intra-group queuing is described in “Distributed queuing with intra-group queuing (multiple delivery paths)” on page 220.

Clustering with intra-group queuing is described in “Clustering with intra-group queuing (multiple delivery paths)” on page 222.

z/OS Intra-group queuing messages

This section describes the messages put to the SYSTEM.QSG.TRANSMIT.QUEUE.

Message structure

Like all other messages that are put to transmission queues, messages that are put to the SYSTEM.QSG.TRANSMIT.QUEUE are prefixed with the transmission queue header (MQXQH).

Message persistence

In IBM WebSphere MQ 5.3 and above, shared queues support both persistent and non-persistent messages.

If the queue manager terminates while the IGQ agent is processing non-persistent messages, or if the IGQ agent terminates abnormally while in the middle of processing messages, non-persistent messages being processed might be lost. Applications must make arrangements for the recovery of non-persistent messages if their recovery is required.

If a put request for a non-persistent message, issued by the IGQ agent, fails unexpectedly, the message being processed is lost.

Delivery of messages

The IGQ agent retrieves and delivers all nonpersistent messages outside of sync point scope, and all persistent messages within sync point scope. In this case, the IGQ agent acts as the sync point coordinator. The IGQ agent therefore processes nonpersistent messages like the way fast, nonpersistent messages are processed on a message channel. See Fast, nonpersistent messages.

Batching of messages

The IGQ agent uses a fixed batch size of 50 messages. Any persistent messages retrieved within a batch are committed at intervals of 50 messages. The agent commits a batch consisting of persistent messages when there are no more messages available for retrieval on the SYSTEM.QSG.TRANSMIT.QUEUE.

Message size

The maximum size of message that can be put to the SYSTEM.QSG.TRANSMIT.QUEUE is the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Default message persistence and default message priority

If the SYSTEM.QSG.TRANSMIT.QUEUE is in the queue name resolution path established at open time, then for messages that are put with default persistence and default priority (or with default persistence or default priority), the normal rules are applied in the selection of the queue that has default priority and persistence values that are used. (See the IBM MQ messages section for more information about the rules of queue selection).

Related concepts

“Undelivered/unprocessed messages” on page 225
This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

“Report messages - Intra Group Queuing” on page 226
This topic describes the report messages: Confirmation of arrival, confirmation of delivery, expiry report, and exception report.

z/OS *Undelivered/unprocessed messages*

This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

If an IGQ agent cannot deliver a message to the destination queue, the IGQ agent:

- Honors the MQRO_DISCARD_MSG report option (if the Report options field of the MQMD for the undelivered message indicates that it must) and discards the undelivered message.

- Attempts to place the undelivered message on to the dead letter queue for the destination queue manager, if the message has not already been discarded. The IGQ agent prefixes the message with a dead letter queue header (MQDLH).

If a dead letter queue is not defined, or if an undelivered message cannot be put to the dead letter queue, and if the undelivered message is:

- persistent, the IGQ agent backs out the current batch of persistent messages that it is processing, and enters a state of retry. For more information, see "Specific properties of intra-group queuing" on page 227.
- non-persistent, the IGQ agent discards the message and continues to process the next message.

If a queue manager in a queue sharing group is terminated before its associated IGQ agent has had time to process all its messages, the unprocessed messages remain on the SYSTEM.QSG.TRANSMIT.QUEUE until the queue manager is next started. The IGQ agent then retrieves and delivers the messages to the destination queues.

If the coupling facility fails before all the messages on the SYSTEM.QSG.TRANSMIT.QUEUE have been processed, any unprocessed non-persistent messages are lost.

IBM recommends that applications do not put messages directly to transmission queues. If an application does put messages directly to the SYSTEM.QSG.TRANSMIT.QUEUE, the IGQ agent might not be able to process these messages and they remain on the SYSTEM.QSG.TRANSMIT.QUEUE. Users then have to use their own methods to deal with these unprocessed messages.

z/OS *Report messages - Intra Group Queuing*

This topic describes the report messages: Confirmation of arrival, confirmation of delivery, expiry report, and exception report.

Confirmation of arrival (COA)/confirmation of delivery (COD) report messages
COA and COD messages are generated by the queue manager, when intra-group queuing is used.

Expiry report messages
Expiry report messages are generated by the queue manager.

Exception report messages

Depending on the MQRO_EXCEPTION_* report option specified in the *Report options* field of the message descriptor for the undelivered message, the IGQ agent generates the required exception report and places it on the specified reply-to queue. Intra-group queuing can be used to deliver the exception report to the destination reply-to queue.

The persistence of the report message is the same as the persistence of the undelivered message. If the IGQ agent fails to resolve the name of the destination reply-to queue, or if it fails to put the reply message to a transmission queue (for subsequent transfer to the destination reply-to queue) it attempts to put the exception report to the dead letter queue of the queue manager on which the report message is generated. If it is not possible, then if the undelivered message is:

- persistent, the IGQ agent discards the exception report, backs out the current batch of messages, and enters a state of retry. For more information, see "Specific properties of intra-group queuing" on page 227.
- non-persistent, the IGQ agent discards the exception report and continues processing the next message on the SYSTEM.QSG.TRANSMIT.QUEUE.

z/OS Security for intra-group queuing

This topic describes the security arrangements for intra-group queuing.

Queue manager attributes IGQAUT (IGQ authority) and IGQUSER (IGQ agent user ID) can be set to control the level of security checking that is performed when the IGQ agent opens destination queues.

Intra-group queuing authority (IGQAUT)

The IGQAUT attribute can be set to indicate the type of security checks to be performed, and hence to determine the userids to be used by the IGQ agent when it establishes the authority to put messages on to the destination queue.

The IGQAUT attribute is analogous to the PUTAUT attribute that is available on channel definitions.

Intra-group queuing user identifier (IGQUSER)

The IGQUSER attribute can be used to nominate a user ID to be used by the IGQ agent when it establishes the authority to put messages on to a destination queue.

The IGQUSER attribute is analogous to the MCAUSER attribute that is available on channel definitions.

z/OS Specific properties of intra-group queuing

This section describes the specific properties of intra-group queuing including Invalidation of object handles, self recovery and retry capability of the intra-group queuing agent, and the intra-group queuing agent and serialization.

Invalidation of object handles (MQRC_OBJECT_CHANGED)

If the attributes of an object are found to have changed after the object is opened, the queue manager invalidates the object handle with MQRC_OBJECT_CHANGED on its next use.

Intra-group queuing introduces the following rules for object handle invalidation:

- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was ENABLED at open time, but intra-group queuing is found to be DISABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was not included in the name resolution path during open processing because intra-group queuing was DISABLED at open time, but intra-group queuing is found to be ENABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was enabled at open time, but the SYSTEM.QSG.TRANSMIT.QUEUE definition is found to have changed by put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.

Self recovery of the intra-group queuing agent

If the IGQ agent terminates abnormally, message CSQM067E is issued and the IGQ agent starts again.

Retry capability of the intra-group queuing agent

If the IGQ agent encounters a problem accessing the SYSTEM.QSG.TRANSMIT.QUEUE (because it is not defined, for example, or is defined with incorrect attributes, or is inhibited for Gets, or for some other reason), the IGQ agent goes into the state of retry.

The IGQ agent observes short and long retry counts and intervals. The values for these counts and intervals, which cannot be changed, are as follows:

Table 20. Short and long retry counts and intervals values

Constant	Value
Short retry count	10
Short retry interval	60 seconds = 1 min
Long retry count	999,999,999

Table 20. Short and long retry counts and intervals values (continued)	
Constant	**Value**
Long retry interval	1200 seconds = 20 min

The intra-group queuing agent and serialization

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail.

If there is a failure of a queue manager in a queue sharing group while the IGQ agent is dealing with uncommitted messages on a shared queue or queues, the IGQ agent ends, and shared queue peer recovery takes place for the failing queue manager. Because shared queue peer recovery is an asynchronous activity, it leaves the possibility for the failing queue manager, and also the IGQ agent for that queue manager, to restart before shared queue peer recovery is complete. Which in turn leaves the possibility for any committed messages to be processed ahead of and out of sequence with the messages still being recovered. To ensure that messages are not processed out of sequence, the IGQ agent serializes access to shared queues by issuing the MQCONNX API call.

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail. An error message is issued and the IGQ agent is put into retry state. When queue manager peer recovery is complete, for example at the time of the next retry, the IGQ agent can start.

z/OS Storage management on z/OS

IBM MQ for z/OS requires permanent and temporary data structures and uses page sets and memory buffers to store this data. These topics give more details on how IBM MQ uses these page sets and buffers.

Related concepts

"Page sets for IBM MQ for z/OS" on page 228
Use this topic to understand how IBM MQ for z/OS uses pages sets to store messages.

"Storage classes for IBM MQ for z/OS" on page 229
A storage class is an IBM MQ for z/OS concept that allows the queue manager to map queues to page sets. You can use storage classes to control which data sets are used by which queues.

"Buffers and buffer pools for IBM MQ for z/OS" on page 231
IBM MQ for z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

Related reference

"Where to find more information about storage management for IBM MQ for z/OS" on page 232
Use this topic as a reference to find further information about storage management for IBM MQ for z/OS.

z/OS Page sets for IBM MQ for z/OS

Use this topic to understand how IBM MQ for z/OS uses pages sets to store messages.

A *page set* is a VSAM linear data set that has been specially formatted to be used by IBM MQ. Page sets are used to store most messages and object definitions.

The exceptions to this are global definitions, which are stored in a shared repository on Db2, and the messages on shared queues. These are not stored on the queue manager page sets. For more information about shared queues, see "Shared queues and queue sharing groups" on page 171, and for more information about global definitions, see Private and global definitions.

IBM MQ page sets can be up to 64 GB in size. Each page set is identified by a page set identifier (PSID), an integer in the range 00 through 99. Each queue manager must have its own page sets.

IBM MQ uses page set zero (PSID=00) to store object definitions and other important information relevant to the queue manager. For normal operation of IBM MQ it is essential that page set zero does not become full, so do not use it to store messages.

To improve the performance of your system, you should also separate short-lived messages from long-lived messages by placing them on different page sets.

You must format page sets, and IBM MQ provides a FORMAT utility for this; see Formatting page sets (FORMAT). Page sets must also be defined to the IBM MQ subsystem.

IBM MQ for z/OS can be configured to expand a page set dynamically if it becomes full. IBM MQ continues to expand the page set if required until 123 logical extents exist, if there is sufficient disk storage space available. The extents can span volumes if the linear data set is defined in this way, however, IBM MQ cannot expand the page sets beyond 64 GB.

You cannot use page sets from one IBM MQ queue manager on a different IBM MQ queue manager, or change the queue manager name. If you want to transfer the data from one queue manager to another, you must unload all the objects and messages from the first queue manager and reload them onto another.

It is not possible to use page sets greater than 4 GB in a queue manager running a release earlier than V6. During the migration period, when it is likely that you might need to fall back to a previous release of code:

- Do not change page set 0 to be greater than 4 GB.
- Other page sets greater than 4 GB will be left offline when restarting a queue manager with a previous release.

For further information about migrating existing page sets capable of expanding beyond 4 GB, see Defining a page set to be larger than 4 GB.

It is possible for an administrator to dynamically add page sets to a running queue manager, or remove page sets from a running queue manager (except for page set zero). The DEFINE PSID command can run after the queue manager restart has completed, only if the command contains the DSN keyword.

z/OS Storage classes for IBM MQ for z/OS

A storage class is an IBM MQ for z/OS concept that allows the queue manager to map queues to page sets. You can use storage classes to control which data sets are used by which queues.

Introducing storage classes

A *storage class* maps one or more queues to a page set. This means that messages for that queue are stored on that page set.

Storage classes allow you to control where non-shared message data is stored for administrative, data set space and load management, or application isolation purposes. You can also use storage classes to define the XCF group and member name of an IMS region if you are using the IMS bridge (described in “IBM MQ and IMS” on page 284).

Shared queues do not use storage classes to obtain a page set mapping because the messages on them are not stored on page sets.

How storage classes work

- You define a storage class, using the DEFINE STGCLASS command, specifying a page set identifier (PSID).
- When you define a queue, you specify the storage class in the STGCLASS attribute.

In the following example, the local queue QE5 is mapped to page set 21 through storage class ARC2.

```
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE QLOCAL(QE5) STGCLASS(ARC2)
```

This means that messages that are put on the queue QE5 are stored on page set 21 (if they stay on the queue long enough to be written to DASD).

More than one queue can use the same storage class, and you can define as many storage classes as you like. For example, you can extend the previous example to include more storage class and queue definitions, as follows:

```
DEFINE STGCLASS(ARC1) PSID(05)
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE STGCLASS(MAXI) PSID(05)
DEFINE QLOCAL(QE1) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE2) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE3) STGCLASS(MAXI) ...
DEFINE QLOCAL(QE4) STGCLASS(ARC2) ...
DEFINE QLOCAL(QE5) STGCLASS(ARC2) ...
```

In Figure 71 on page 230, both storage classes ARC1 and MAXI are associated with page set 05. Therefore, the queues QE1, QE2, and QE3 are mapped to page set 05. Similarly, storage class ARC2 associates queues QE4 and QE5 with page set 21.

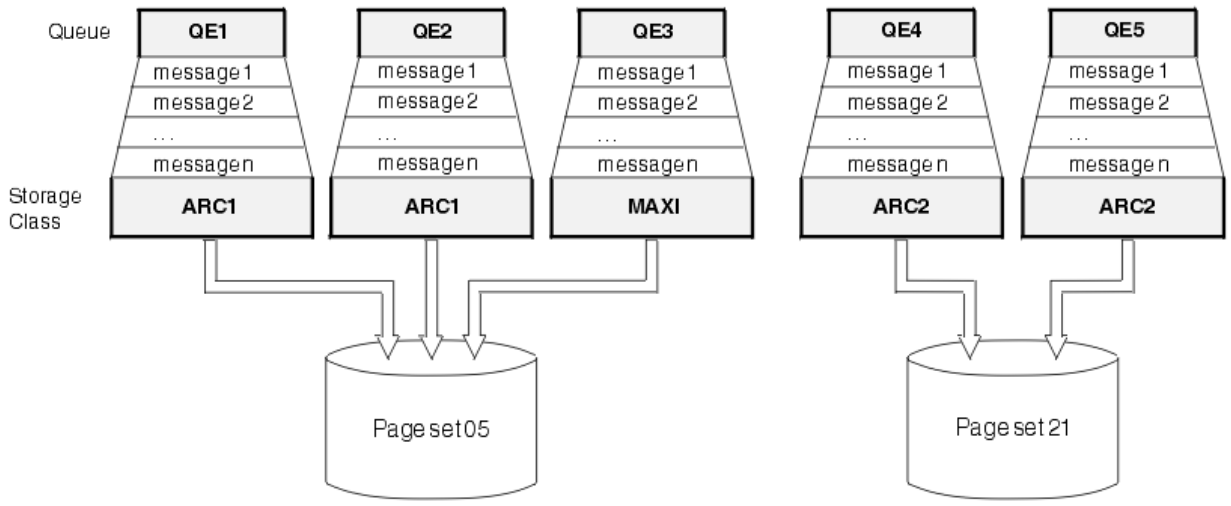


Figure 71. Mapping queues to page sets through storage classes

If you define a queue without specifying a storage class, IBM MQ uses a default storage class.

If a message is put on a queue that names a nonexistent storage class, the application receives an error. You must alter the queue definition to give it an existing storage class name, or create the storage class named by the queue.

You can change a storage class only when:

- All queues that use this storage class are empty, and have no uncommitted activity.
- All queues that use this storage class are closed.

z/OS Buffers and buffer pools for IBM MQ for z/OS

IBM MQ for z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

For efficiency, IBM MQ uses a form of caching whereby messages (and object definitions) are stored temporarily in buffers before being stored in page sets on DASD. Short-lived messages, that is, messages that are retrieved from a queue shortly after they are received, might only ever be stored in the buffers. This caching activity is controlled by a buffer manager, which is a component of IBM MQ.

The buffers are organized into *buffer pools*. You can define up to 100 buffer pools (0 through 99) for each queue manager.

You are recommended to use the minimal number of buffer pools consistent with the object and message type separation outlined in Figure 72 on page 231, and any data isolation requirements your application might have. Each buffer is 4 KB long. Buffer pools use 31 bit storage by default, in this mode, the maximum number of buffers is determined by the amount of 31 bit storage available in the queue manager address space; do not use more than about 70% for buffers. Alternatively, buffer pool storage allocation can be made from 64 bit storage (use the LOCATION attribute of the **DEFINE BUFFPOOL** command). Using LOCATION(ABOVE) so that 64 bit storage is used has two benefits. Firstly, there is much more 64 bit storage available so buffer pools can be much bigger, and secondly, 31 bit storage is made available for use by other functions. Typically, the more buffers you have, the more efficient the buffering and the better the performance of IBM MQ.

Figure 72 on page 231 shows the relationship between messages, buffers, buffer pools, and page sets. A buffer pool is associated with one or more page sets; each page set is associated with a single buffer pool.

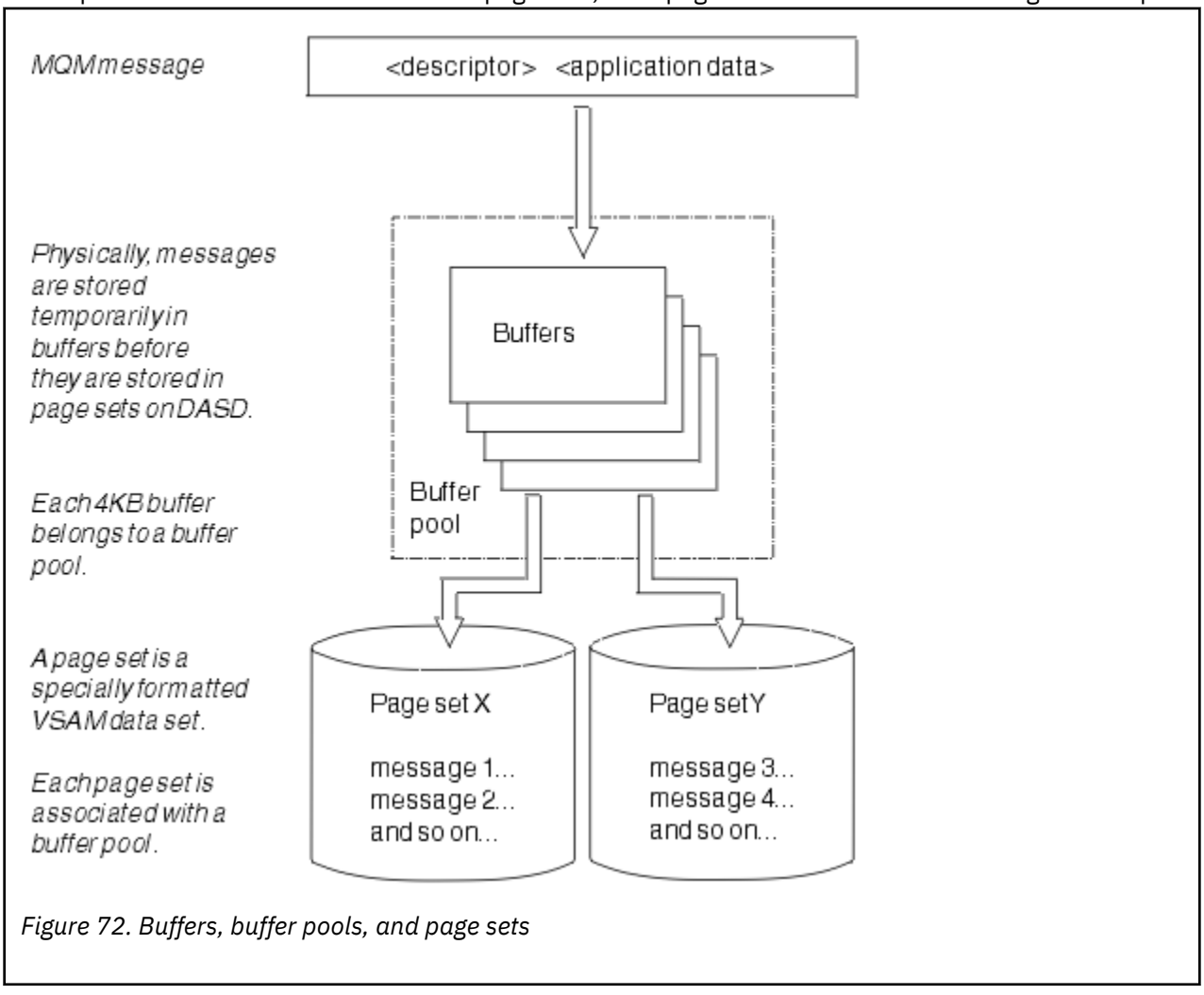


Figure 72. Buffers, buffer pools, and page sets

You can dynamically issue commands to modify buffer pool size, and location, using the **ALTER BUFFPOOL** command. Page sets can be dynamically added by using the **DEFINE PSID** command, or deleted by using the **DELETE PSID** command.

If a buffer pool is too small, IBM MQ issues message CSQP020E. You can then dynamically add more buffers to the affected buffer pool (note that you may have to remove buffers from other buffer pools to do this).

You specify the number of buffers in a pool with the **DEFINE BUFFPOOL** command, and you can dynamically resize buffer pools with the **ALTER BUFFPOOL** command. You determine the current number of buffers in a pool dynamically by displaying a page set that uses the buffer pool, using the **DISPLAY USAGE** command.

For performance reasons, do not put messages and object definitions in the same buffer pool. Use one buffer pool (say number zero) exclusively for page set zero, where the object definitions are kept. Similarly, keep short-lived messages and long-lived messages in different buffer pools and therefore on different page sets, and in different queues.

The **DEFINE BUFFPOOL** command cannot be used after restart to create a new buffer pool. Instead, if a **DEFINE PSID** command uses the DSN keyword, it can explicitly identify a buffer pool that is not currently defined. That new buffer pool will then be created.

z/OS Where to find more information about storage management for IBM MQ for z/OS

Use this topic as a reference to find further information about storage management for IBM MQ for z/OS.

You can find more information about the topics in this section from the following sources:

Table 21. Where to find more information about storage management

Topic	Where to look
How much storage you need	Planning your storage and performance requirements on z/OS
How large to make your page sets and buffer pools	Plan your page sets and buffer pools
Managing page sets	Managing page sets
MQSC commands	The MQSC commands

z/OS Logging in IBM MQ for z/OS

IBM MQ maintains *logs* of data changes and significant events as they occur. These logs can be used to recover data to a previous state if required.

The *bootstrap data set* (BSDS) stores information about the data sets that contain the logs.

The log does not contain information for statistics, traces, or performance evaluation. For further details about the statistical and monitoring information that IBM MQ collects, see Monitoring and statistics.

For more information about logging, see the following topics:

- “Log files in IBM MQ for z/OS” on page 233
- “How the log is structured” on page 236
- “How the IBM MQ for z/OS logs are written” on page 237
- “Larger log Relative Byte Address” on page 240
- “The bootstrap data set” on page 241

Related tasks

Planning your logging environment

Setting logs using the system parameter module

z/OS Administering z/OS

Related reference

z/OS Messages for IBM MQ for z/OS

z/OS Log files in IBM MQ for z/OS

Log files contain information needed for transaction recovery. Active log files can be archived so that you can keep log data for a long period.

What is a log file

IBM MQ records all significant events as they occur in an *active log*. The log contains the information needed to recover:

- Persistent messages
- IBM MQ objects, such as queues
- The IBM MQ queue manager

The active log comprises a collection of data sets (up to 310) which are used cyclically.

You can enable log archiving so that when an active log fills a copy is made in an archive data set. Using archiving allows you to keep log data for an extended period. If you do not use archiving, the logs wrap and earlier data is overwritten. To recover a page set, or recover data in a CF structure, you need log data from when the backup of the page set or structure was taken. An archive log can be created on disk or on tape.

Archiving

Because the active log has a fixed size, IBM MQ copies the contents of each log data set periodically to an *archive log*, which is normally a data set on a direct-access storage device (DASD) or a magnetic tape. If there is a subsystem or transaction failure, IBM MQ uses the active log and, if necessary, the archive log for recovery.

The archive log can contain up to 1000 sequential data sets. You can catalog each data set using the z/OS integrated catalog facility (ICF).

Archiving is an essential component of IBM MQ recovery. If a unit of recovery is a long-running one, log records within that unit of recovery might be found in the archive log. In this case, recovery requires data from the archive log. However, if archiving is disabled, the active log with new log records wraps, overwriting earlier log records. This means that IBM MQ might not be able to back out the unit of recovery and messages might be lost. The queue manager then terminates abnormally.

Therefore, in a production environment, **never switch archiving off**. If you do, you run the risk of losing data after a system or transaction failure. Only if you are running in a test environment can you consider switching archiving off. If you need to do this, use the CSQ6LOGP macro, which is described in Using CSQ6LOGP.

To help prevent problems with unplanned long-running units of work, IBM MQ issues a message (CSQJ160I or CSQJ161I) if a long-running unit of work is detected during active log offload processing.

Dual logging

In dual logging, each log record is written to two different active log data sets to minimize the likelihood of data loss problems during restart.

You can configure IBM MQ to run with either *single logging* or *dual logging*. With single logging, log records are written once to an active log data set. Each active log data set is a single-extent VSAM linear data set (LDS). With dual logging, each log record is written to two different active log data sets. Dual logging minimizes the likelihood of data loss problems during restart.

Log shunting

Log shunting causes the log records for some units of work to be written further down the log. This reduces the amount of log data that must be read at queue manager restart, or backout, for long running or long term in-doubt units of work.

When a unit of work is considered to be long, a representation of each log record is written further down the log. This technique is known as *shunting*. When the whole of the unit of work has been processed, the unit of work is in a *shunted* state. Any backout or restart activity relating to the shunted unit of work can use the shunted log records instead of using the original unit of work log records.

Detecting a long-running unit of work is a function of the checkpoint process. At checkpoint time, each active unit of work is checked to establish whether it needs to be shunted. If the unit of work has been through two prior checkpoints since it was created, or since it was last shunted, the unit of work is suitable to be shunted. This means that a single unit of work might be shunted more than once. This is known as a *multi-shunted* unit of work.

A unit of work is shunted every three checkpoints. However the checkpoint is performed asynchronously to the log-switch (or the writing of the log record which caused LOGLOAD to be exceeded).

There is only a single checkpoint taking place at a time, so there might be multiple log-switches before a checkpoint completes.

This means that if there are not enough active logs, or if they are too small, then shunting of a large unit of work might not complete before all the logs are filled.

Message CSQR027I results if shunting is unable to complete.

If log archiving is turned off, ABEND 5C6 with reason 00D1032A occurs if there is an attempt to back out the unit of work for which shunting failed. To avoid this problem you should use OFFLOAD=YES.

Log shunting is always active, and runs whether log archiving is enabled or not.

Note: Although all log records for a unit of work are shunted, the entire content of each record is not shunted, only the part that is necessary for backout. This means that the amount of log data written is kept to a minimum, and that shunted records cannot be used if a page set failure occurs. A long running unit of work is one that has been running for more than three queue manager checkpoints.

For more information about log shunting, see Managing the logs.

Log compression

You can configure IBM MQ for z/OS to compress and decompress log records as they are written and read from the log data set.

Log compression can be used to reduce the amount of data written to the log for persistent messages on private queues. The amount of compression that is achieved depends on the type of data contained within messages. For example, Run Length Encoding (RLE) works by compacting repeated instances of bytes which can give good results efficiently for structured or record oriented data.

Attention: Persistent messages that are being put to a shared queue are not subject to log compression.

You can use fields within the Log manager section of the System Management Facility 115 (SMF) records to monitor how much data compression is achieved. For more information about SMF, see Using the System Management Facility and Accounting and statistics messages.

Log compression increases the processor utilization of the system. You should only consider using compression if throughput of your queue manager is constrained by the IO bandwidth writing to the

log data sets or you are constrained by the disk storage needed to hold log data sets. If you are using shared queues then IO bandwidth constraints can be relieved by adding additional queue managers to the queue sharing group and distributing the workload across more queue managers.

The log compression option can be enabled and disabled as required without the need to stop and restart the queue manager. The queue manager can read any compressed log records regardless of the current log compression setting.

The queue manager supports 3 settings for log compression.

NONE
No log data compression is used. This is the default value.

RLE
Log data compression is performed using run-length encoding (RLE).

ANY
Enable the queue manager to select the compression algorithm that gives the greatest degree of log record compression. This option results in RLE compression.

You can control the compression of log records using one of the following:

- The SET and DISPLAY LOG commands in MQSC; see SET LOG and DISPLAY LOG
- The Set Log and Inquire Log functions in the PCF interface; see Set log and Inquire log
- The CSQ6LOGP macro in the system parameter module; see Using CSQ6LOGP

In addition the Log Print utility CSQ1LOGP has support for expanding any compressed log records.

Log data

The log can contain up to 18 million million million ($1.8*10^{19}$) bytes. Each byte can be addressed by its offset from the beginning of the log, and that offset is known as its *relative byte address* (RBA).

The RBA is referenced by a 6-byte or 8-byte field giving a total addressable range of 2^{48} bytes, or 2^{64} bytes, depending on whether 6-byte or 8-byte log RBAs are in use.

However, when IBM MQ detects that the used range is beyond F00000000000 (if 6-byte RBAs are in use) or FFFF800000000000 (if 8-byte log RBAs are in use), messages CSQI045, CSQI046, CSQI047, and CSQJ032 are issued, warning you to reset the log RBA.

If the RBA value reaches FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFC000000000 (if 8-byte log RBAs are in use) the queue manager terminates with reason code 00D10257.

Once the warning messages about the used log range are being issued, you should plan a queue manager outage during which the queue manager can be converted to use 8-byte log RBAs, or the log can be reset. The procedure to reset the log is documented in Resetting the queue manager's log.

If your queue manager is using 6-byte log RBAs, consider converting the queue manager to use 8-byte log RBAs rather than resetting the queue manager's log, following the procedure documented in Implementing the larger log Relative Byte Address.

The log consists of *log records*, each of which is a set of log data treated as a single unit. A log record is identified either by the RBA of the first byte of its header, or by its log record sequence number (LRSN). The RBA or LRSN uniquely identifies a record that starts at a particular point in the log.

Whether you use the RBA or LRSN to identify log points depends on whether you are using queue sharing groups. In a queue-sharing environment, you cannot use the relative byte address to uniquely identify a log point, because multiple queue managers can update the same queue at the same time, and each has its own log. To solve this, the log record sequence number is derived from a timestamp value, and does not necessarily represent the physical displacement of the log record within the log.

Each log record has a header that gives its type, the IBM MQ subcomponent that made the record, and, for unit of recovery records, a unit of recovery identifier.

There are four types of log record, described under the following headings:

- Unit of recovery log records
- Checkpoint records
- Page set control records
- CF structure backup records

Unit of recovery log records

Most of the log records describe changes to IBM MQ queues. All such changes are made within units of recovery.

IBM MQ uses special logging techniques involving *undo/redo* and *compensating log records* to reduce restart times and improve system availability.

One effect of this is that the restart time is bounded. If a failure occurs during a restart so that the queue manager has to be restarted a second time, all the recovery activity that completed to the point of failure in the first restart does not need to be reapplied during a second restart. This means that successive restarts do not take progressively longer times to complete.

Checkpoint records

To reduce restart time, IBM MQ takes periodic checkpoints during normal operation. These occur as follows:

- When a predefined number of log records has been written. This number is defined by the checkpoint frequency operand called LOGLOAD of the system parameter macro CSQ6SYSP, described in Using CSQ6SYSP.
- At the end of a successful restart.
- At normal termination.
- Whenever IBM MQ switches to the next active log data set in the cycle.

At the time a checkpoint is taken, IBM MQ issues the DISPLAY CONN command (described in DISPLAY CONN) internally so that a list of connections currently in doubt is written to the z/OS console log.

Page set control records

These records register the page sets and buffer pools known to the IBM MQ queue manager at each checkpoint, and record information about the log ranges required to perform media recovery of the page set at the time of the checkpoint.

Certain dynamic changes to page sets and buffer pools are also written as page set control records, so that the changes can be recovered and automatically reinstated at the next queue manager restart.

CF structure backup records

These records hold data read from a coupling facility list structure in response to a BACKUP CFSTRUCT command. In the unlikely event of a coupling facility structure failure, these records are used, together with unit of recovery records, by the RECOVER CFSTRUCT command to perform media recovery of the coupling facility structure to the point of failure.

Related tasks

Implementing the larger log Relative Byte Address

z/OS How the log is structured

Use this topic to understand the terminology used to describe log records.

Each active log data set must be a VSAM linear data set (LDS). The physical output unit written to the active log data set is a 4 KB control interval (CI). Each CI contains one VSAM record.

Physical and logical log records

One VSAM CI is a *physical* record. The information logged at a particular time forms a *logical* record, with a length that varies independently of the space available in the CI. So one physical record might contain:

- Several logical records
- One or more logical records and part of another logical record
- Part of one logical record only

The term *log record* refers to the *logical* record, regardless of how many *physical* records are needed to store it.

z/OS How the IBM MQ for z/OS logs are written

Use this topic to understand how IBM MQ processes log file records.

IBM MQ writes each log record to a DASD data set called the *active log*. When the active log is full, IBM MQ copies its contents to a DASD or tape data set called the *archive log*. This process is called *offloading*.

Figure 73 on page 238 illustrates the process of logging. Log records typically go through the following cycle:

1. IBM MQ notes changes to data and significant events in recovery log records.
2. IBM MQ processes recovery log records and breaks them into segments, if necessary.
3. Log records are placed sequentially in *output log buffers*, which are formatted as VSAM Controls Intervals (CI). Each log record is identified by a relative byte address in the range zero through 2^{64} -1.
4. The CIs are written to a set of predefined DASD active log data sets, which are used sequentially and recycled.
5. If archiving is active, as each active log data set becomes full, its contents are automatically offloaded to a new archive log data set.

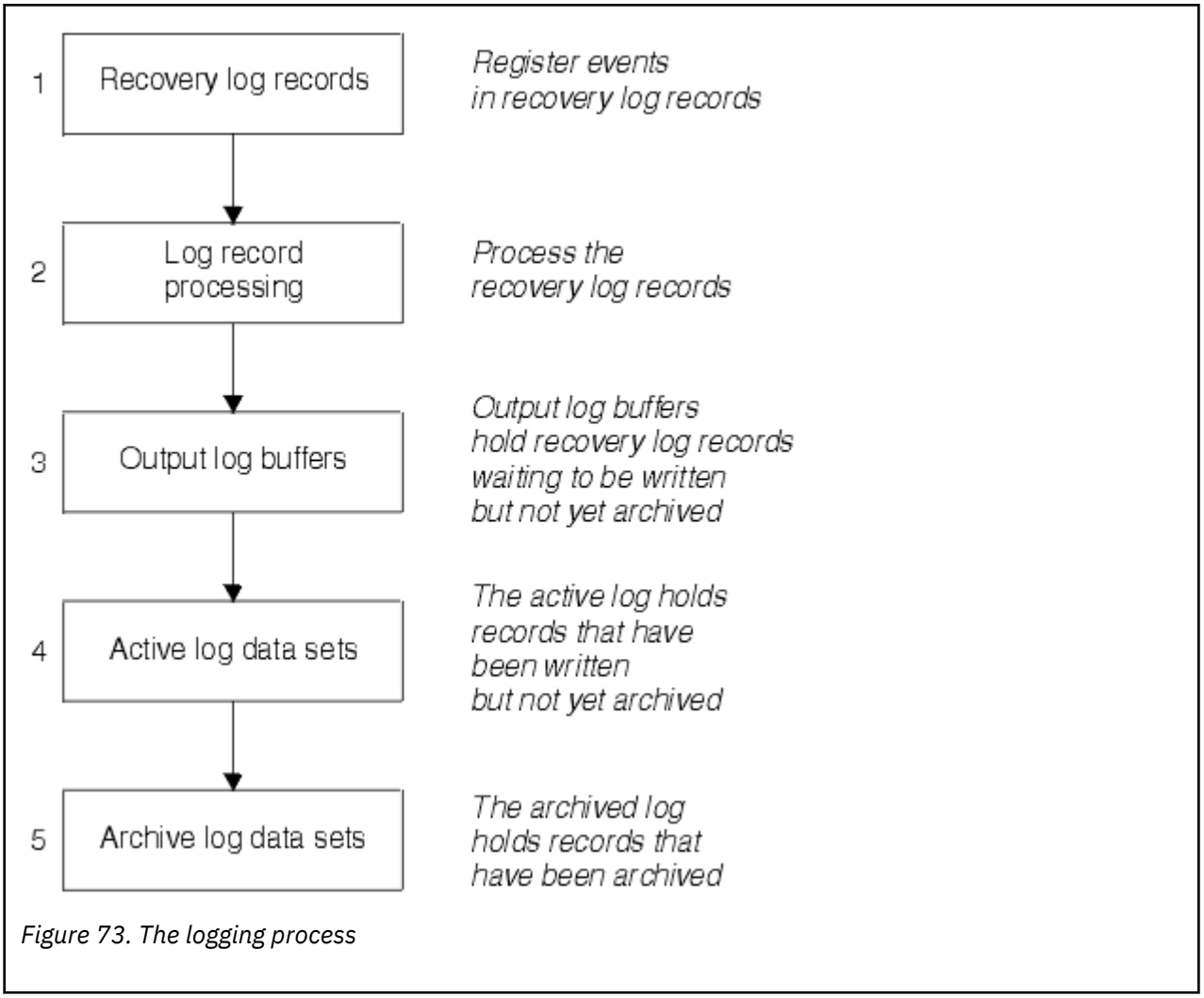


Figure 73. The logging process

When the active log is written

The in-storage log buffers are written to an active log data set whenever any of the following occur:

- The log buffers become full.
- The write threshold is reached (as specified in the CSQ6LOGP macro).
- Certain significant events occur, such as a commit point, or when an IBM MQ BACKUP CFSTRUCT command is issued.

When the queue manager is initialized, the active log data sets named in the BSDS are dynamically allocated for exclusive use by the queue manager and remain allocated exclusively to IBM MQ until the queue manager terminates.

Dynamically adding log data sets

It is possible to dynamically define new active log data sets while the queue manager is running. This feature alleviates the problem of a queue manager hang when archiving is not able to offload active logs due to a transient problem. See the DEFINE LOG command for more information.

Note: To redefine or remove active logs you must terminate and restart the queue manager.

IBM MQ and Storage Management Subsystem

IBM MQ parameters enable you to specify Storage Management Subsystem (MVS™/DFP SMS) storage classes when allocating IBM MQ archive log data sets dynamically. IBM MQ initiates the archiving of log data sets, but you can use SMS to perform allocation of the archive data set.

Related reference

“When the IBM MQ for z/OS archive log is written” on page 239
Use this topic to understand the process of copying active logs to archive logs, and when the process occurs.

z/OS *When the IBM MQ for z/OS archive log is written*

Use this topic to understand the process of copying active logs to archive logs, and when the process occurs.

The process of copying active logs to archive logs is called *offloading*. The relation of offloading to other logging events is shown schematically in Figure 74 on page 239.

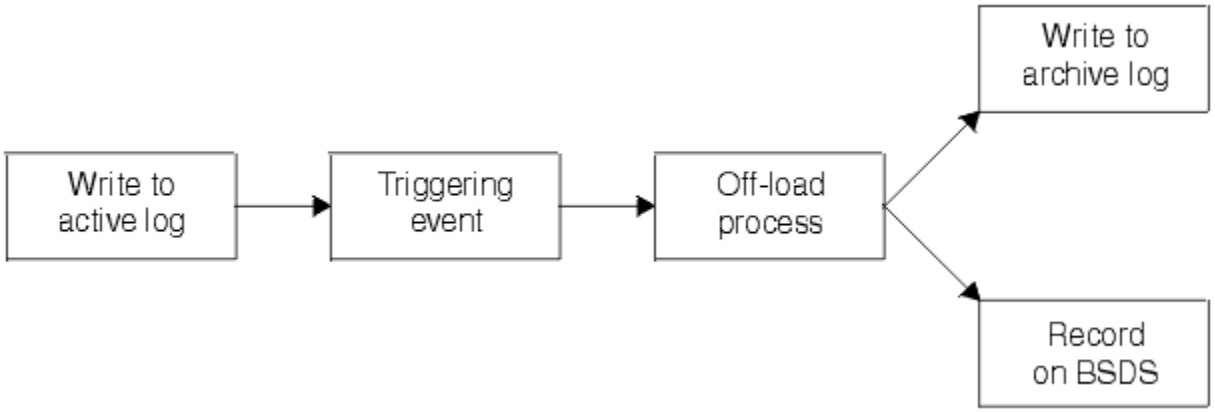


Figure 74. The offloading process

Triggering the offloading process

The offload process of an active log to an archive log can be triggered by several events. For example:

- Filling an active log data set.
- Using the MQSC ARCHIVE LOG command.
- An error occurring while writing to an active log data set.

The data set is truncated before the point of failure, and the record that was not written becomes the first record of the new data set. Offloading is triggered for the truncated data set as for a normal full log data set. If there are dual active logs, both copies are truncated so that the two copies remain synchronized.

Message CSQJ110E is issued when the last available active log is 5% full and at 5% increments thereafter, stating the percentage of the log's capacity that is in use. If all the active logs become full, IBM MQ stops processing, until offloading occurs, and issues this message:

```
CSQJ111A +CSQ1 OUT OF SPACE IN ACTIVE LOG DATA SETS
```

The offload process

When all the active logs become full, IBM MQ runs the offloading process and halts processing until the offloading process has been completed. If the offload processing fails when the active logs are full, IBM MQ abends.

When an active log is ready to be offloaded, a request is sent to the z/OS console operator to mount a tape or prepare a DASD unit. The value of the ARCWTOR logging option (for further information, see Using CSQ6ARVP) determines whether the request is received. If you are using tape for offloading, specify ARCWTOR=YES. If the value is YES, the request is preceded by a WTOR (message number CSQJ008E) telling the operator to prepare an archive log data set to be allocated.

The operator need not respond to this message immediately. However, delaying the response delays the offload process. It does not affect IBM MQ performance unless the operator delays the response for so long that IBM MQ runs out of active logs.

The operator can respond by canceling the offload process. In this case, if the allocation is for the first copy of dual archive data sets, the offload process is merely delayed until the next active log data set becomes full. If the allocation is for the second copy, the archive process switches to single copy mode, but for this data set only.

Interruptions and errors while offloading

A request to stop the queue manager does not take effect until offload processing has finished. If IBM MQ fails while offloading is in progress, offloading begins again when the queue manager is restarted.

Messages during offload processing

Offloaded messages are sent to the z/OS console by IBM MQ and the offloading process. You can use these messages to find the RBA ranges in the various log data sets.

z/OS Larger log Relative Byte Address

This function improves the availability of the queue manager by increasing the period of time before you have to reset the log.

Recovery data is written to the log so that persistent messages are available when the queue manager is restarted. The term log Relative Byte Address (log RBA) is used to refer to the location of data as an offset from the beginning of the log.

Before IBM MQ 8.0, the 6 byte log RBA could address up to 256 terabytes of data. Before this quantity of log records has been written, you have to reset the queue manager's log by following the procedure documented in Resetting the queue manager's log.

Resetting the logs of queue managers is not a quick process, and can require an extended outage, due to the need to reset the page sets as part of the process. For a high use queue manager this operation might typically be done once a year.

From IBM MQ 8.0, the log RBA can be 8 bytes long and the queue manager can now address over 64,000 times as much data (16 exabytes) before the log RBA needs to be reset. The impact of using the larger log RBA is that the size of the log data written increases by a few bytes.

When is this function enabled?

Queue managers created at IBM MQ 9.3.0 or later already have this function enabled.

If the current log RBA is approaching the end of the log RBA range, consider converting the queue manager to use an 8 byte log RBA rather than resetting the queue manager's log. Converting a queue manager to use 8 byte log RBAs requires a shorter outage than resetting the log, and significantly increases the period of time before you have to reset the log.

Message CSQJ034I, issued during queue manager initialization, indicates the end of the log RBA range for the queue manager as configured, and can be used to determine whether 6 byte or 8 byte log RBAs are in use.

How is this function enabled?

8 byte log RBA is enabled by starting the queue manager with a version 2 format BSDS. In summary, this is achieved by:

1. Ensuring that all queue managers in the queue sharing group meet the requirements for enabling 8 byte log RBA
2. Shutting down the queue manager cleanly
3. Running the BSDS conversion utility to create a copy of the BSDS in version 2 format.
4. Restarting the queue manager with the converted BSDS.

Once a queue manager has been converted to use 8 byte log RBAs, it cannot go back to using 6 byte log RBA.

See Implementing the larger log Relative Byte Address for the detailed procedure on how to enable 8 byte log RBAs.

Related tasks

Planning to increase the maximum addressable log range

Related reference

The BSDS conversion utility (CSQJUCNV)

z/OS The bootstrap data set

The bootstrap data set is required by IBM MQ as a mechanism to reference log data sets, and log records. This information is required during normal processing, and restart recovery.

What the bootstrap data set is for

The *bootstrap data set* (BSDS) is a VSAM key-sequenced data set (KSDS) that holds information needed by IBM MQ. It contains the following:

- An inventory of all active and archived log data sets known to IBM MQ. IBM MQ uses this inventory to:
  - Track the active and archived log data sets
  - Locate log records so that it can satisfy log read requests during normal processing
  - Locate log records so that it can handle restart processing

  IBM MQ stores information in the inventory each time an archive log data set is defined or an active log data set is reused. For active logs, the inventory shows which are full and which are available for reuse. The inventory holds the relative byte address (RBA) of each portion of the log held in that data set.
- A *wrap-around* inventory of all recent IBM MQ activity. This is needed if you have to restart the queue manager.

The BSDS is required if the queue manager has an error and you have to restart it. IBM MQ **must** have a BSDS. To minimize the likelihood of problems during a restart, you can configure IBM MQ with dual BSDSs, each recording the same information. Using dual BSDSs is known as running in *dual mode*. If possible, place the copies on separate volumes. This reduces the risk of them both being lost if the volume is corrupted or destroyed. Use dual BSDSs rather than dual write to DASD.

The BSDS is set up when IBM MQ is customized and you can manage the inventory using the change log inventory utility (CSQJU003). For more information about this utility, see Administración IBM MQ for z/OS. It is referenced by a DD statement in the queue manager startup procedure.

Normally, IBM MQ keeps duplicate copies of the BSDS. If an I/O error occurs, it deallocates the failing copy and continues with a single BSDS. You can restore dual-mode operation, this is described in the Administración IBM MQ for z/OS.

The active logs are first registered in the BSDS when IBM MQ is installed. You cannot replace the active logs without terminating and restarting the queue manager.

Archive log data sets are allocated dynamically. When one is allocated, the data set name is registered in the BSDS. The list of archive log data sets expands as archives are added, and wraps when a user-determined number of entries has been reached. The maximum number of entries is 1000 for single archive logging and 2000 for dual logging.

You can use a tape management system to delete the archive log data sets (IBM MQ does not have an automated method). Therefore, the information about an archive log data set can be in the BSDS long after the archive log data set has been deleted by the system administrator.

Conversely, the maximum number of archive log data sets could have been exceeded, and the data from the BSDS dropped long before the data set has reached its expiry date.

You can use the following MQSC command to determine the extent of the log, and the name of the active or archive log data set holding the earliest log RBA, required for various types of media or queue manager recovery:

```
DISPLAY USAGE TYPE(DATASET)
```

If the system parameter module specifies that archive log data sets are cataloged when allocated, the BSDS points to the integrated catalog facility (ICF) catalog for the information needed for later allocations. Otherwise, the BSDS entries for each volume register the volume serial number and unit information that is needed for later allocations.

The BSDS version

The format of the BSDS varies according to its version. Increasing the version of the BSDS allows new features to be used. The following BSDS versions are supported by IBM MQ:

Version 1
Supported by all releases of IBM MQ. A version 1 BSDS supports 6-byte log RBA values.

Version 2
Supported by IBM MQ 8.0 and later. A version 2 BSDS enables 8-byte log RBA values, and up to 310 data sets in each active log copy.

Enabled by default for queue managers created at IBM MQ 9.3.0 or later.

Version 3
Supported by IBM MQ 8.0 and later. The BSDS is automatically converted to version 3, from version 2, when more than 31 data sets are added to either active log copy.

You can determine the version of a BSDS by running the print log map utility (CSQJU004). To convert a BSDS from Version 1 to Version 2, run the BSDS conversion utility (CSQJUCNV).

See “Larger log Relative Byte Address” on page 240 for more information on 6-byte and 8-byte log RBAs.

Archive log data sets and BSDS copies

Each time a new archive log data set is created, a copy of the BSDS is also created. If the archive log is on tape, the BSDS is the first data set on the first output volume. If the archive log is on DASD, the BSDS is a separate data set.

The data set names of the archive log and the BSDS copy are the same, except that the lowest-level qualifier of the archive log name begins with A and the BSDS copy begins with B, for example:

Archive log name
CSQ.ARCHLOG1.E00186.T2336229. *A* 0000001

BSDS copy name
CSQ.ARCHLOG1.E00186.T2336229. *B* 0000001

If there is a read error while copying the BSDS, the copy is not created, message CSQJ125E is issued, and the offloading to the new archive log data set continues without the BSDS copy.

z/OS System definition on z/OS

IBM MQ for z/OS uses many default object definitions, and provides sample JCL to create those default objects. Use this topic to understand these default objects, and the sample JCL.

Setting system parameters

In IBM MQ for z/OS, a system parameter module controls the logging, archiving, tracing, and connection environments that IBM MQ uses in its operation. The system parameters are specified by three assembler macros, as follows:

CSQ6SYSP
System parameters, including setting the connection and tracing environment.

CSQ6LOGP
Logging parameters.

CSQ6ARVP
Log archive parameters.

Default parameter modules are supplied with IBM MQ for z/OS. If these do not contain the values that you want to use, you can create your own parameter modules using the sample supplied with IBM MQ. The sample is `thlqual.SCSQPROC(CSQ4ZPRM)`.

You can alter some system parameters while a queue manager is running. See the SET SYSTEM, SET LOG, and SET ARCHIVE commands in The MQSC commands.

For more information about defining , see the following topics:

- “Defining system objects for IBM MQ for z/OS” on page 243
- “Tuning your queue manager on IBM MQ for z/OS” on page 248
- “Sample definitions supplied with IBM MQ for z/OS” on page 249

Related concepts
Customize the sample initialization input data sets
Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS
Related tasks
Administering z/OS
Configuring clusters
Monitoring IBM MQ

z/OS Defining system objects for IBM MQ for z/OS

IBM MQ for z/OS requires additional predefined objects for publish/subscribe applications, cluster, and channel control and other system administration functions.

The system objects required by IBM MQ for z/OS can be divided into the following categories:

- Publish/subscribe objects
- System default objects
- System command objects
- System administration objects
- Channels queues
- Cluster queues

- Queue sharing group queues
- Storage classes
- Defining the system object dead-letter queue
- Default transmission queue
- Internal queues
- “Channel authentication queue” on page 247

Publish/subscribe objects

There are several system objects that you need to define before you can use publish/subscribe applications with IBM MQ for z/OS. Sample definitions are supplied with IBM MQ to help you define these objects. These samples are described in CSQ4INSG.

To use publish/subscribe you need to define the following objects:

- A local queue called SYSTEM.RETAINED.PUB.QUEUE, which is used to hold a copy of each retained publication in the queue manager. Each full topic name could have up to one retained publication stored on this queue. If your applications will make use of retained publications on many different topics, or if your retained publication messages are large messages, the requirements for storage for this queue should be carefully planned, including assigning it to its own page set if the storage requirements for it are large. To improve performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definition).
- A local queue called SYSTEM.DURABLE.SUBSCRIBER.QUEUE, which is used to hold a persistent copy of the durable subscriptions in the queue manager. To improve performance, you should define this queue with an index type of CORRELID (as shown in the supplied sample queue definition).
- A local queue called SYSTEM.DURABLE.MODEL.QUEUE, which is used as a model for managed durable subscriptions.
- A local queue called SYSTEM.NDURABLE.MODEL.QUEUE, which is used as a model for managed non-durable subscriptions.
- A namelist called SYSTEM.QPUBSUB.QUEUE.NAMELIST, which contains a list of queue names monitored by the queued publish/subscribe interface.
- A namelist called SYSTEM.QPUBSUB.SUBPOINT.NAMELIST, which contains a list of topic objects used by the queued publish/subscribe interface to match topic objects to subscription points.
- A topic called SYSTEM.BASE.TOPIC, which is used as a base topic for resolving attributes.
- A topic called SYSTEM.BROKER.DEFAULT.STREAM, which is the default stream used by the queued publish/subscribe interface.
- A topic called SYSTEM.BROKER.DEFAULT.SUBPOINT, which is the default RFH2 subscription point used by the queued publish/subscribe interface.
- A topic called SYSTEM.BROKER.ADMIN.STREAM, which is the admin stream used by the queued publish/subscribe interface.
- A subscription called SYSTEM.DEFAULT.SUB, which is a default subscription object used to provide default values on DEFINE SUB commands.

System default objects

System default objects are used to provide default attributes when you define an object and do not specify the name of another object to base the definition on.

The names of the default system object definitions begin with the characters "SYSTEM.DEFAULT" or "SYSTEM.DEF." For example, the system default local queue is named SYSTEM.DEFAULT.LOCAL.QUEUE.

These objects define the system defaults for the attributes of these IBM MQ objects:

- Local queues

- Model queues
- Alias queues
- Remote queues
- Processes
- Namelists
- Channels
- Storage classes
- Authentication information

Shared queues are a special type of local queue, so when you define a shared queue, the definition is based on the SYSTEM.DEFAULT.LOCAL.QUEUE. You need to remember to supply a value for the Coupling Facility structure name because one is not specified in the default definition. Alternatively, you could define your own default shared queue definition to use as a basis for shared queues so that they all inherit the required attributes. Remember that you need to define a shared queue on one queue manager in the queue sharing group only.

System command objects

The names of the system command objects begin with the characters SYSTEM.COMMAND. You must define these objects before you can use the IBM MQ operations and control panels to issue commands to an IBM MQ subsystem.

There are two system command objects:

1. The system-command input queue is a local queue on which commands are put before they are processed by the IBM MQ command processor. It must be called SYSTEM.COMMAND.INPUT. An alias named SYSTEM.ADMIN.COMMAND.QUEUE should also be defined, for compatibility with IBM MQ for Multiplatforms, and for use by the IBM MQ Console and administrative REST API.
2. SYSTEM.COMMAND.REPLY.MODEL is a model queue that defines the system-command reply-to queue.

There are two extra objects for use by the IBM MQ Explorer:

- SYSTEM.MQEXPLORER.REPLY.MODEL queue
- SYSTEM.ADMIN.SVRCONN channel

SYSTEM.REST.REPLY.QUEUE is the reply queue used by the IBM MQ administrative REST API.

Commands are normally sent using nonpersistent messages so both the system command objects should have the DEFPSIST(NO) attribute so that applications using them (including the supplied applications like the utility program and the operations and control panels) get nonpersistent messages by default. If you have an application that uses persistent messages for commands, set the DEFTYPE(PERMDYN) attribute for the reply-to queue, because the reply messages to such commands are persistent.

System administration objects

The names of the system administration objects begin with the characters SYSTEM.ADMIN.

There are seven system administration objects:

- The SYSTEM.ADMIN.CHANNEL.EVENT queue
- The SYSTEM.ADMIN.COMMAND.EVENT queue
- The SYSTEM.ADMIN.CONFIG.EVENT queue
- The SYSTEM.ADMIN.PERFM.EVENT queue
- The SYSTEM.ADMIN.QMGR.EVENT queue

- The SYSTEM.ADMIN.TRACE.ROUTE.QUEUE queue
- The SYSTEM.ADMIN.ACTIVITY.QUEUE queue

Channels queues

To use distributed queuing, you need to define the following objects:

- A local queue with the name SYSTEM.CHANNEL.SYNCQ, which is used to maintain sequence numbers and logical units of work identifiers (LUWID) of channels. To improve channel performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definition).
- A local queue with the name SYSTEM.CHANNEL.INITQ, which is used for channel commands.

You cannot define these queues as shared queues.

Cluster queues

To use IBM MQ clusters, you need to define the following objects:

- A local queue called the SYSTEM.CLUSTER.COMMAND.QUEUE, which is used to communicate repository changes between queue managers. Messages written to this queue contain updates to the repository data to be applied to the local copy of the repository, or requests for repository data.
- A local queue called SYSTEM.CLUSTER.REPOSITORY.QUEUE, which is used to hold a persistent copy of the repository.
- A local queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE, which is the transmission queue for all destinations in the cluster. For performance reasons, you should define this queue with an index type of CORRELID (as shown in the sample queue definition).

These queues typically contain large numbers of messages.

You cannot define these queues as shared queues.

Queue sharing group queues

To use shared channels and intra-group queuing, you need to define the following objects:

- A shared queue with the name SYSTEM.QSG.CHANNEL.SYNCQ, which is used to hold synchronization information for shared channels.
- A shared queue with the name SYSTEM.QSG.TRANSMIT.QUEUE, which is used as the transmission queue for intra-group queuing. If you are running in a queue sharing group, you must define this queue, even if you are not using intra-group queuing.

Storage classes

You are recommended to define the following six storage classes. You must define four of them because they are required by IBM MQ. The other storage class definitions are recommended because they are used in the sample queue definitions.

DEFAULT (required)
This storage class is used for all message queues that are not performance critical and that don't fit in to any of the other storage classes. It is also the supplied default storage class if you do not specify one when defining a queue.

NODEFINE (required)
This storage class is used if the storage class specified when you define a queue is not defined.

REMOTE (required)
This storage class is used primarily for transmission queues, that is, system related queues with short-lived performance-critical messages.

SYSLNGLV
This storage class is used for long-lived, performance-critical messages.

SYSTEM (required)
This storage class is used for performance critical, system related message queues, for example the SYSTEM.CHANNEL.SYNQ and the SYSTEM.CLUSTER.* queues.

SYSVOLAT
This storage class is used for short-lived, performance-critical messages.

You can modify their attributes and add other storage class definitions as required.

Defining the system object dead-letter queue

The dead-letter queue is used if the message destination is not valid. IBM MQ puts such messages on a local queue called the dead-letter queue. Although having a dead-letter queue is not mandatory, you should regard it as essential, especially if you are using distributed queuing or one of the IBM MQ bridges.

Do **not** define the dead-letter queue as a shared queue. A put to a local queue on one queue manager might get put to the dead letter queue. If the dead letter queue was a shared queue, a dead letter queue handler on a different system could process the message and put it on a queue with the same name, but because this is on a different queue manager, it would be the wrong queue, or have a different security profile. If the queue did not exist, it would fail to reprocess it.

If you decide to define a dead-letter queue, you must also tell the queue manager its name. To do this use the ALTER QMGR DEADQ(*queue-name*) command. For more information see Displaying and altering queue manager attributes.

Default transmission queue

The default transmission queue is used when no other suitable transmission queue is available for sending messages to another queue manager. If you define a default transmission queue, you must also define a channel to serve the queue. If you do not do this, messages that are put on to the default transmission queue are not transmitted to the remote queue manager and remain on the queue.

If you decide to define a default transmission queue, you must also tell the queue manager its name. To do this use the ALTER QMGR command.

Internal queues

- **Pending data queue**
  - A queue defined for internal use, SYSTEM.PENDING.DATA.QUEUE, supports the use of durable subscriptions in a JMS publish/subscribe environment.
- **JMS 2.0 delivery delay staging queue**
  - If the delivery delay functionality provided by JMS 2.0 is used then an internal staging queue, SYSTEM.DDELAY.LOCAL.QUEUE, must be defined. This queue is used by the queue manager to temporarily store messages sent with a non-zero delivery delay until the delivery delay is completed, and the message is put to its target destination. A sample definition for this queue is provided, commented out, in CSQ4INSG.
  - When you define the SYSTEM.DDELAY.LOCAL.QUEUE queue, you must set the STGCLASS, MAXMSGL and MAXDEPTH attributes for the anticipated number of messages that will be sent with a delivery delay. Additionally when defining the SYSTEM.DDELAY.LOCAL.QUEUE queue ensure that only the queue manager can put messages to this queue. Care should be taken to ensure that no user identifier has the authority to put messages to this queue.

Channel authentication queue

For internal use of channel authentication the SYSTEM.CHLAUTH.DATA.QUEUE queue is required. Sample definitions are supplied with IBM MQ to help you define these objects. This sample is described in CSQ4INSA, which also defines some default rules.

z/OS Tuning your queue manager on IBM MQ for z/OS

There are few simple steps that you can take to ensure that your queue manager is tuned to avoid basic performance problems.

There are a number of ways in which you can improve the performance of your queue manager, which are controlled by queue manager attributes set by the ALTER QMGR command. This section contains information about how you can do this by setting the maximum number of messages allowed on the queue manager, or by performing 'housekeeping' on the queue manager. IBM MQ SupportPac MP16 - IBM MQ for z/OS Planificación de capacidad y ajuste de gives more information on performance and tuning.

Syncpoints

One of the roles of the queue manager is syncpoint control within an application. An application constructs a unit of work containing any number of MQPUT or MQGET calls terminated with an MQCMIT call.

As the number of MQPUT or MQGET calls within the scope of one MQCMIT increases, the performance cost of the commit increases significantly. Applications, in general, should be designed to not MQPUT/MQGET a large number of messages in a single synchpoint.

You can administratively limit the number of messages within any single syncpoint by using the MAXUMSGS queue manager attribute. If an application exceeds this limit it receives MQRC_SYNCPOINT_LIMIT_REACHED on the MQPUT,MQPUT1, or MQGET call which exceeds the limit. The application should then issue MQCMIT or MQBACK as appropriate.

The default value of MAXUMSGS is 10000. This value can be lowered if you want to enforce a lower limit, which can also help protect against looping applications. Before reducing MAXUMSGS make sure you understand your existing applications to ensure they do not exceed the limit, or can tolerate the MQRC_SYNCPOINT_LIMIT_REACHED return code

Expired messages

Messages that have expired are discarded by the next appropriate MQGET call. However, if no such call occurs, the expired messages are not discarded, and, for some queues, particularly those where message retrieval is done by MessageId, CorrelId, or GroupId and the queue is indexed for performance, many expired messages can accumulate. The queue manager can periodically scan any queue for expired messages, which are then deleted. You can choose how often this scanning takes place, if at all. There are two ways of doing this:

Explicit request
: You can control which queues are scanned and when. Issue the REFRESH QMGR TYPE(EXPIRY) command, specifying the queue or queues that you want to be scanned.

Periodic scan
: You can specify an expiry interval in the queue manager object by using the EXPRYINT attribute. The queue manager maintains information about the expired messages on each queue, and knows at what time a scan for expired messages is worthwhile. Each time that the EXPRYINT interval is reached, the queue manager looks for candidate queues that are worth scanning for expired messages, and scans only those queues that it deems to be worthwhile. It does not scan all queues. This avoids any processor time being wasted on unnecessary scans.

  Shared queues are only scanned by one queue manager in the queue sharing group. Generally, the first queue manager to restart or the first to have the EXPRYINT set performs the scan.

Note: You must set the same EXPRYINT value for all queue managers within a queue sharing group.

z/OS Sample definitions supplied with IBM MQ for z/OS

Use this topic as a reference for the sample JCL, and code supplied with IBM MQ for z/OS.

The following sample definitions are supplied with IBM MQ in the thlqual.SCSQPROC library. You can use them to define the system objects and to customize your own objects. You can include some of them in the initialization input data sets (described in Initialization commands).

Table 22. IBM MQ sample definitions for system objects

Initialization input data set	Sample name
CSQINP1	CSQ4INP1 CSQ4INPR
CSQINP2	CSQ4INSA CSQ4INYS [1] CSQ4INSX CSQ4INSG CSQ4INSR CSQ4INSS CSQ4INSJ CSQ4INSM CSQ4INYG CSQ4INYR CSQ4INYC CSQ4INYD CSQ4INSC
CSQINPT	CSQ4INST CSQ4INYT
Other	CSQ4DISP CSQ4INPX CSQ4IVPQ CSQ4IVPG CSQ4MSTR CSQ4MSRR CSQ4QMIN

Note:

1. The order of these sample definitions is important: an error occurs if INYS, INSX, and INSG are ordered incorrectly.

CSQINP1 samples

Use the sample CSQINP1 data set thlqual.SCSQPROC(CSQ4INP1) when you are using one page set for each class of message, or thlqual.SCSQPROC(CSQ4INPR) when using multiple page sets for the major classes of message. It contains definitions of buffer pools, page set to buffer pool associations, and an ALTER SECURITY command. Include the sample in the CSQINP1 concatenation of your queue manager started task procedure.

CSQINP2 samples

CSQ4INSG system object sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSG) contains definitions for the following system objects for general use:

- System default objects
- System command objects
- System administration objects
- Other objects for system use

You must define the objects in this sample, but you need to do it only once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across queue manager shutdown and restart. You must not change the object names, but you can change their attributes if required.

When the following conditions are met, one message is put to the SYSTEM.DURABLE.SUBSCRIBER.QUEUE queue (even if publish subscribe is not active):

- The QMGR attribute PSMODE is set to DISABLED
- The sample object CSQ4INST statement `DEFINE SUB('SYSTEM.DEFAULT.SUB')` is present.

To avoid this, delete or comment out the `DEFINE SUB('SYSTEM.DEFAULT.SUB')` statement.

The JMS 2.0 delivery delay staging queue, SYSTEM.DDELAY.LOCAL.QUEUE only need be defined if JMS 2.0 delivery delay is used. By default, the queue definition is commented out, which you can uncomment if required.

CSQ4INSA system object and authentication sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSA) contains the channel authentication system queue definition. This queue holds the channel authentication records. It also contains the default channel authentication rules.

You must define the objects in this sample if CHLAUTH is ENABLED on the queue manager and you want to run channels, or you want to SET or DISPLAY CHLAUTH record. You only need to define them once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across a queue manager shutdown and restart, you must not change the queue name.

CSQ4INSS system object sample

You can define additional system objects if you are using queue sharing groups.

Sample data set thlqual.SCSQPROC(CSQ4INSS) contains sample commands for use with CF structures and a set of definitions for the system objects required for shared channels and intra-group queuing.

You cannot use this sample as is; you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required commands.

When you are defining group or shared objects, you need to include them in the CSQINP2 DD concatenation for only one queue manager in the queue sharing group.

CSQ4INSX system object sample

You must define additional system objects if you are using distributed queuing and clustering.

Sample data set thlqual.SCSQPROC(CSQ4INSX) contains the queue definitions required. You can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or

you can use it as input to the COMMAND function in CSQUTIL utility to issue the required DEFINE commands.

There are two types of object definitions:

- SYSTEM.CHANNEL.xx, needed for any distributed queuing
- SYSTEM.CLUSTER.xx, needed for clustering

CSQ4INSJ system JMS object sample

Defines queues used in the JMS publish/subscribe domain.

CSQ4INSM system object sample

If you are using advanced message security you must define additional system objects. Sample data set thlqual.SCSQPROC(CSQ4INSM) contains the queue definitions required.

CSQ4INSR object sample

Defines queues used by WebSphere Application Server and brokers.

CSQ4INYD object sample

If you are using distributed queuing and you need to set up your own queues, processes, and channels.

Sample data set thlqual.SCSQPROC(CSQ4INYD) contains sample definitions that you can use for customizing your distributed queuing objects. It comprises:

- A set of definitions for the sending end
- A set of definitions for the receiving end
- A set of definitions for using clients

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. (This is preferable because it means that you don't have to redefine these objects each time you restart the queue manager).

CSQ4INYC object sample

If you are using clustering, definitions equivalent to the channel definitions and remote queue definitions of distributed queuing are created automatically, when needed. However, some manual channel definitions are needed - a cluster-receiver channel for the cluster and a cluster-sender definition to at least one cluster repository queue manager.

The sample data set: thlqual.SCSQPROC(CSQ4INYC) contains the following sample definitions that you can use for customizing your clustering objects:

- Definitions for the queue manager
- Definitions for the receiving channel
- Definitions for the sending channel
- Definitions for cluster queues
- Definitions for lists of clusters

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. This is preferable because it means that you don't have to redefine these objects each time that you restart IBM MQ.

CSQ4INYG object sample

The sample data set: thlqual.SCSQPROC(CSQ4INYG) contains the following sample definitions that you can use for customizing your own objects for general use:

- Dead-letter queue
- Default transmission queue
- CICS adapter objects

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. This is preferable because it means that you don't have to redefine these objects each time that you restart IBM MQ.

In addition to the sample definitions here, you can use the system object definitions as the basis for your own resource definitions. For example, you can make a working copy of SYSTEM.DEFAULT.LOCAL.QUEUE and name it MY.DEFAULT.LOCAL.QUEUE. You can then change any of the parameters in this copy as required. You could then issue a DEFINE command by whichever method you choose, provided you have the authority to create resources of that type.

Default transmission queue
Read the Default transmission queue description before you decide whether you want to define a default transmission queue.

- If you decide that you do want to define a default transmission queue, remember that you must also define a channel to serve it.
- If you decide that you do not want to define one, remember to remove the DEFXMITQ statement from the ALTER QMGR command in the sample.

CICS adapter objects
The sample defines an initiation queue named CICS01.INITQ. This queue is used by the IBM MQ -supplied CKTI transaction. You can change the name of this queue; however it must match the name specified in the CICS system initialization table (SIT) or SYSIN override in the INITPARM statement.

CSQ4INYS/CSQ4INYR object samples

Storage class definitions for using:

- one page set for each class of message
- multiple page sets for major classes of message

For example, SYSTEM.COMMAND.INPUT uses STGCLASS('SYSVOLAT'), and SYSTEM.CLUSTER.TRANSMIT.QUEUE uses STGCLASS('REMOTE'). In CSQ4INYS, both of those storage classes use the same page set. In CSQ4INYR, those storage classes use different page sets in order to lessen the impact of the transmission queue filling.

CSQINPT samples

CSQ4INST
The sample data set: thlqual.SCSQPROC(CSQ4INST) contains the definition for the system default subscription.

You must define the object in this sample, but you need to do it only once when the publish/subscribe engine is first started. Including the definition in the CSQINPT data set is the best way to do this. It is maintained across queue manager shutdown and restart. You must not change the object name, but you can change their attributes if required.

CSQ4INYT
The sample data set: thlqual.SCSQPROC(CSQ4INYT) contains a set of commands that you might want to run when the publish/subscribe engine is started. This sample displays Topic and Subscription information.

Other

CSQ4DISP display sample

The sample data set: thlqual.SCSQPROC(CSQ4DISP) contains a set of generic DISPLAY commands that display all the defined resources on your queue manager. This includes the definitions for all IBM MQ objects and definitions such as storage classes and trace. These commands can generate a large amount of output. You can use this sample in the CSQINP2 data set or as input to the COMMAND function of the CSQUTIL utility.

CSQ4INPX sample

The sample data set: thlqual.SCSQPROC(CSQ4INPX) contains a set of commands that you might want to execute each time the channel initiator starts. You must customize this sample before use; you can then include it in the CSQINPX data set for the channel initiator.

CSQ4IVPQ and CSQ4IVPG samples

The sample data sets: thlqual.SCSQPROC(CSQ4IVPQ) and thlqual.SCSQPROC(CSQ4IVPG) contain sets of DEFINE commands that are required to run the installation verification programs (IVPs).

You can include these samples in the CSQINP2 data set. When you have run the IVPs successfully, you do not need to run them again each time the queue manager is restarted. Therefore, you do not need to keep these samples in the CSQINP2 concatenation permanently.

CSQ4MSTR and CSQ4MSRR samples

These are sample started task procedures for the queue manager: thlqual.SCSQPROC(CSQ4MSTR) and thlqual.SCSQPROC(CSQ4MSRR).

CSQ4MSRR uses CSQ4INYR in the CSQINP2 concatenation so that important queues are spread across different page sets.

You can remove the comments, so that you can use the CSQMINI card for newly created queue managers if required.

CSQ4QMIN sample

A sample QMINI data set, thlqual.SCSQPROC(CSQ4QMIN).

See QMINI data set for details of the QMINI data set and the **TransportSecurity** stanza.

z/OS Recovery and restart on z/OS

Use the links in this topic to find out about the features of IBM MQ for z/OS for restart and recovery.

IBM MQ for z/OS has robust features for restart and recovery. For information about how a queue manager recovers after it has stopped, and what happens when it is restarted, see the following subtopics:

- “How changes are made to data in IBM MQ for z/OS” on page 254
- “How consistency is maintained in IBM MQ for z/OS” on page 255
- “What happens during termination in IBM MQ for z/OS” on page 257
- “What happens during restart and recovery in IBM MQ for z/OS” on page 258
- “How in-doubt units of recovery are resolved” on page 260
- “Shared queue recovery” on page 263

Related concepts

z/OS IBM MQ for z/OS recovery actions

Related tasks

Planning for backup and recovery

z/OS Administering z/OS

Related reference

z/OS Messages for IBM MQ for z/OS

z/OS How changes are made to data in IBM MQ for z/OS

IBM MQ for z/OS must interact with other subsystems to keep all the data consistent. This topic contains information about *units of recovery*, what they are and how they are used in *back outs*.

Units of recovery

A *unit of recovery* is the processing done by a single queue manager for an application program, that changes IBM MQ data from one point of consistency to another. A *point of consistency* - also called a *syncpoint* or *commit point* - is a point in time when all the recoverable data that an application program accesses is consistent.

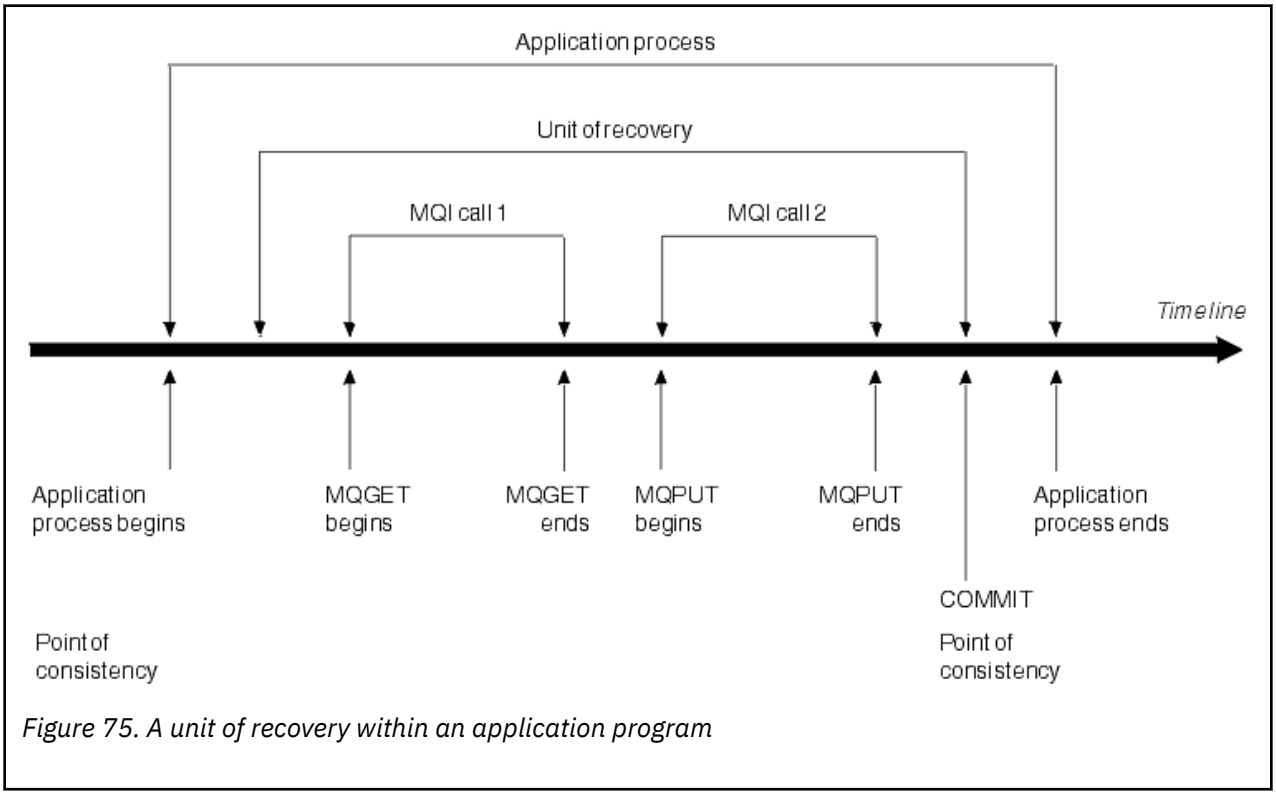


Figure 75. A unit of recovery within an application program

A unit of recovery begins with the first change to the data after the beginning of the program or following the previous point of consistency; it ends with a later point of consistency. Figure 75 on page 254 shows the relationship between units of recovery, the point of consistency, and an application program. In this example, the application program makes changes to queues through MQI calls 1 and 2. The application program can include more than one unit of recovery or just one. However, any complete unit of recovery ends in a commit point.

For example, a bank transaction transfers funds from one account to another. First, the program subtracts the amount from the first account, account A. Then, it adds the amount to the second account, B. After subtracting the amount from A, the two accounts are inconsistent and IBM MQ cannot commit. They become consistent when the amount is added to account B. When both steps are complete, the program can announce a point of consistency through a commit, making the changes visible to other application programs.

Normal termination of an application program automatically causes a point of consistency. Some program requests in CICS and IMS programs also cause a point of consistency, for example, EXEC CICS SYNCPOINT.

Backing out work

If an error occurs within a unit of recovery, IBM MQ removes any changes to data, returning the data to its state at the start of the unit of recovery; that is, IBM MQ backs out the work. The events are shown in Figure 76 on page 255.

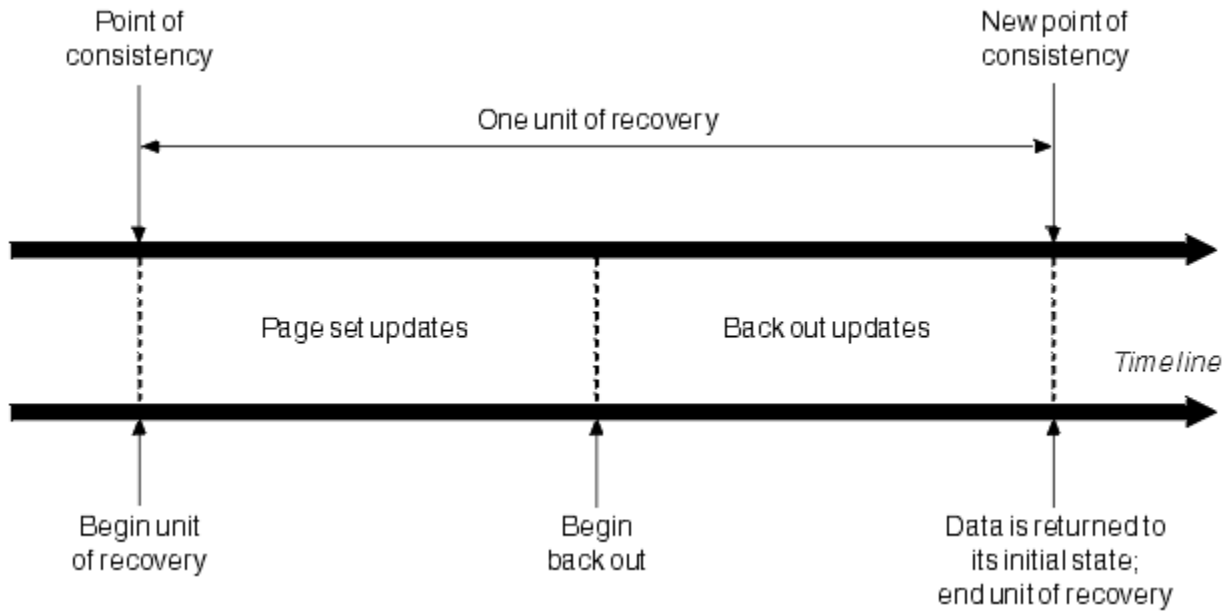


Figure 76. A unit of recovery showing back out

z/OS How consistency is maintained in IBM MQ for z/OS

Data in IBM MQ for z/OS must be consistent with batch, CICS, IMS, or TSO. Any data changed in one must be matched by a change in the other.

Before one system commits the changed data, it must know that the other system can make the corresponding change. So, the systems must communicate.

During a *two-phase commit* (for example under CICS), one subsystem coordinates the process. That subsystem is called the *coordinator*; the other is the *participant*. CICS or IMS is always the coordinator in interactions with IBM MQ, and IBM MQ is always the participant. In the batch or TSO environment, IBM MQ can participate in two-phase commit protocols coordinated by z/OS RRS.

During a *single-phase commit* (for example under TSO or batch), IBM MQ is always the coordinator in the interactions and completely controls the commit process.

In a WebSphere Application Server environment, the semantics of the JMS session object determine whether single-phase or two-phase commit coordination is used.

Consistency with CICS or IMS

The connection between IBM MQ and CICS or IMS supports the following syncpoint protocols:

- Two-phase commit - for transactions that update resources owned by more than one resource manager.

This is the standard distributed syncpoint protocol. It involves more logging and message flows than a single-phase commit.

- Single-phase commit - for transactions that update resources owned by a single resource manager (IBM MQ).

  This protocol is optimized for logging and message flows.

- Bypass of syncpoint - for transactions that involve IBM MQ but which do nothing in the queue manager that requires a syncpoint (for example, browsing a queue).

In each case, CICS or IMS acts as the syncpoint manager.

The stages of the two-phase commit that IBM MQ uses to communicate with CICS or IMS are as follows:

1. In phase 1, each system determines independently whether it has recorded enough recovery information in its log, and can commit its work.

   At the end of the phase, the systems communicate. If they agree, each begins the next phase.
2. In phase 2, the changes are made permanent. If one of the systems abends during phase 2, the operation is completed by the recovery process during restart.

Illustration of the two-phase commit process

Figure 77 on page 256 illustrates the two-phase commit process. Events in the CICS or IMS coordinator are shown on the upper line, events in IBM MQ on the lower line.

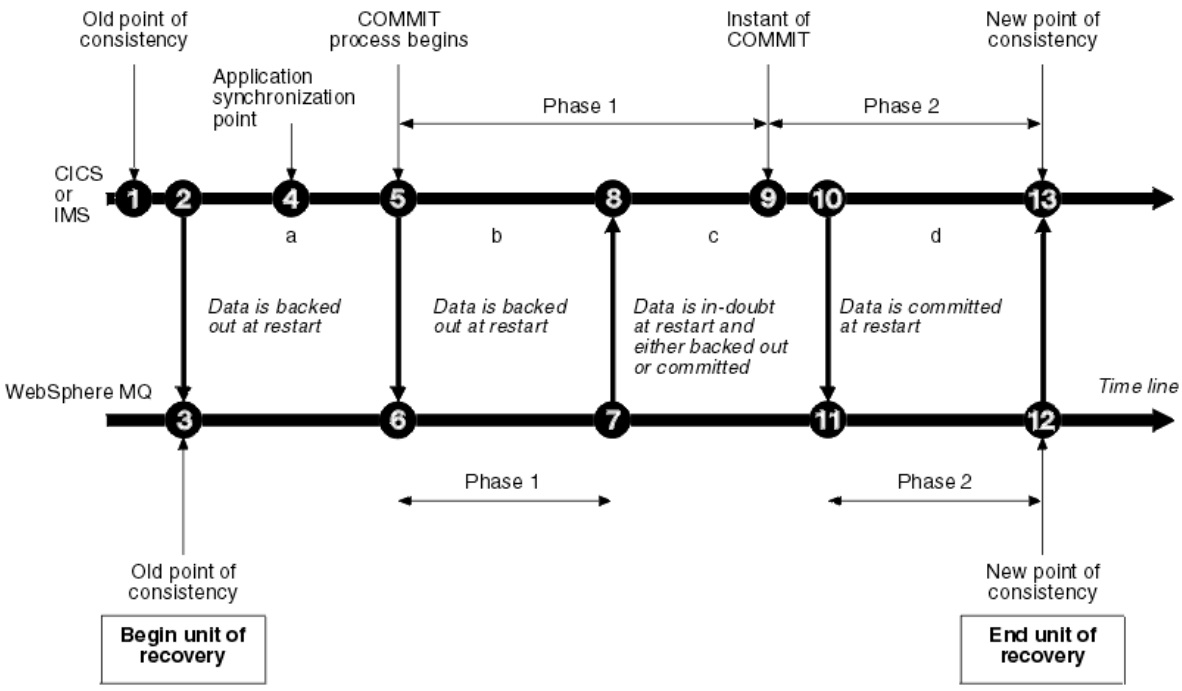


Figure 77. The two-phase commit process

The numbers in the following section are linked to those shown in the figure.

1. The data in the coordinator is at a point of consistency.
2. An application program in the coordinator calls IBM MQ to update a queue by adding a message.
3. This starts a unit of recovery in IBM MQ.
4. Processing continues in the coordinator until an application synchronization point is reached.
5. The coordinator then starts commit processing. CICS programs use a SYNCPOINT command or a normal application termination to start the commit. IMS programs can start the commit by using

a CHKP call, a SYNC call, a GET UNIQUE call to the IOPCB, or a normal application termination. Phase 1 of commit processing begins.

6. As the coordinator begins phase 1 processing, so does IBM MQ.
7. IBM MQ successfully completes phase 1, writes this fact in its log, and notifies the coordinator.
8. The coordinator receives the notification.
9. The coordinator successfully completes its phase 1 processing. Now both subsystems agree to commit the data changes, because both have completed phase 1 and could recover from any errors. The coordinator records in its log the instant of commit - the irrevocable decision of the two subsystems to make the changes.

   The coordinator now begins phase 2 of the processing - the actual commitment.
10. The coordinator notifies IBM MQ to begin its phase 2.
11. IBM MQ logs the start of phase 2.
12. Phase 2 is successfully completed, and this is now a new point of consistency for IBM MQ. IBM MQ then notifies the coordinator that it has finished its phase 2 processing.
13. The coordinator finishes its phase 2 processing. The data controlled by both subsystems is now consistent and available to other applications.

How consistency is maintained after an abnormal termination

When a queue manager is restarted after an abnormal termination, it must determine whether to commit or to back out units of recovery that were active at the time of termination. For some units of recovery, IBM MQ has enough information to make the decision. For others, it does not, and must get information from the coordinator when the connection is reestablished.

Figure 77 on page 256 shows four periods within the two phases: a, b, c, and d. The status of a unit of recovery depends on the period in which termination happened. The status can be one of the following:

In flight
The queue manager ended before finishing phase 1 (period a or b); during restart, IBM MQ backs out the updates.

In doubt
The queue manager ended after finishing phase 1 and before starting phase 2 (period c); only the coordinator knows whether the error happened before or after the commit (point 9). If it happened before, IBM MQ must back out its changes; if it happened after, IBM MQ must make its changes and commit them. At restart, IBM MQ waits for information from the coordinator before processing this unit of recovery.

In commit
The queue manager ended after it began its own phase 2 processing (period d); it makes committed changes.

In backout
The queue manager ended after a unit of recovery began to be backed out but before the process was complete (not shown in the figure) during restart, IBM MQ continues to back out the changes.

z/OS What happens during termination in IBM MQ for z/OS

A queue manager terminates normally in response to the STOP QMGR command. If a queue manager stops for any other reason, the termination is abnormal.

Note, that during queue manager termination, IBM MQ internally issues the command

```
DISPLAY CONN(*) TYPE(CONN) ALL WHERE (APPLTYPE NE SYSTEMAL)
```

so that you are aware of what threads might prevent the queue manager from completing shutdown.

SYSTEMAL matches APPLTYPES of either SYSTEM or CHINIT, so the DISPLAY CONN command filtering application types not matching SYSTEMAL, returns to the joblog information about threads that could be preventing normal shutdown.

Normal termination

In a normal termination, IBM MQ stops all activity in an orderly way. You can stop IBM MQ using either quiesce, force, or restart mode. The effects are given in Table 23 on page 258.

Table 23. Termination using QUIESCE, FORCE, and RESTART

Thread type	QUIESCE	FORCE	RESTART
Active threads	Run to completion	Back out	Back out
New threads	Can start	Not permitted	Not permitted
New connections	Not permitted	Not permitted	Not permitted

Batch applications are notified if a termination occurs while the application is still connected.

With CICS, a current thread runs only to the end of the unit of recovery. With CICS, stopping a queue manager in quiesce mode stops the CICS adapter, and so if an active task contains more than one unit of recovery, the task does not necessarily run to completion.

If you stop a queue manager in force or restart mode, no new threads are allocated, and work on connected threads is rolled back. Using these modes can create in-doubt units of recovery for threads that are between commit processing phases. They are resolved when IBM MQ is reconnected with the controlling CICS, IMS, or RRS subsystem.

When you stop a queue manager, in any mode, the steps are:

1. Connections are ended.
2. IBM MQ ceases to accept commands.
3. IBM MQ ensures that any outstanding updates to the page sets are completed.
4. The DISPLAY USAGE command is issued internally by IBM MQ so that the restart RBA is recorded on the z/OS console log.
5. The shutdown checkpoint is taken and the BSDS is updated.

Terminations that specify quiesce mode do not affect in-doubt units of recovery. Any unit that is in doubt remains in doubt.

Abnormal termination

An abnormal termination can leave data in an inconsistent state, for example:

- A unit of recovery has been interrupted before reaching a point of consistency.
- Committed data has not been written to page sets.
- Uncommitted data has been written to page sets.
- An application program has been interrupted between phase 1 and phase 2 of the commit process, leaving the unit of recovery in doubt.

IBM MQ resolves any data inconsistencies arising from abnormal termination during restart and recovery.

z/OS What happens during restart and recovery in IBM MQ for z/OS

IBM MQ uses its recovery log and the bootstrap data set (BSDS) to determine what to recover when it restarts. The BSDS identifies the active and archive log data sets, and the location of the most recent IBM MQ checkpoint in the log.

Introduction to restart and recovery

After IBM MQ has been initialized, the queue manager restart process takes place as follows:

- Log initialization
- Current status rebuild
- Forward log recovery
- Backward log recovery
- Queue index rebuilding

When recovery has been completed:

- Committed changes are reflected in the data.
- In-doubt activity is reflected in the data. However, the data is locked and cannot be used until IBM MQ recognizes and acts on the in-doubt decision.
- Interrupted in-flight and in-abort changes have been removed from the queues. The messages are consistent and can be used.
- A new checkpoint has been taken.
- New indexes have been built for indexed queues containing persistent messages (described in “Rebuilding queue indexes” on page 260).

If dual BSDSs are in use, IBM MQ checks the consistency of the time stamps in the BSDS:

- If both copies of the BSDS are current, IBM MQ tests whether the two time stamps are equal. If they are not, IBM MQ issues message CSQJ120E and terminates. This can happen when the two copies of the BSDS are maintained on separate DASD volumes and one of the volumes was restored while the queue manager was stopped. IBM MQ detects the situation at restart.
- If one copy of the BSDS was de-allocated, and logging continued with a single BSDS, a problem could arise. If both copies of the BSDS are maintained on a single volume, and the volume was restored, or if both BSDS copies were restored separately, IBM MQ might not detect the restoration. In that case, log records not noted in the BSDS would be unknown to the system.

Batch applications are not notified when restart occurs *after* the application has requested a connection.

Understanding the log range required for recovery

During restart, the range of log data which must be read is dependent on many factors:

- At the time of an abnormal termination, there are typically many incomplete units of work in the system. As described earlier, restart processing will bring the system to a state of consistency, which may involve backing out inflight units of work, or recovering locks on indoubt units of work. Unit of work recovery requires that all unit of work log records for inflight, in-backout, and in-doubt units of work are available. IBM MQ will 'shunt' old units of work, so that unit of work recovery can be performed using a much smaller range of log data.
- At the time of an abnormal termination, there are typically many persistent updates which are only held in the buffer pool cache. They have not yet been written to disk. These changes must be read from the log, and reapplied to the data held in page sets. Page set recovery RBAs in the checkpoint describe the lowest log RBA required for updating the page sets to a consistent state.
- If old page sets have been introduced into the system, for example, a page set backup has been introduced to recover from a media failure, all the changes must be read from the log from the time the backup was taken. These changes are reapplied to the data held in the page set being recovered. Page set recovery RBAs held in page 0 of the page set describe the lowest log RBA required for media recovery of a page set.
- If using persistent messages on shared queues, a range of log data is required to recover CFSTRUCTs which are holding persistent messages. The earliest log data that would be required to perform a CFSTRUCT recovery, is from around the time of the old CFSTRUCT BACKUP.

During normal running, the DISPLAY USAGE TYPE(DATASET) command can be used to view the recovery log range associated with these factors (it is unable to provide information due to reintroducing old page sets, of course). To avoid any issues that might prolong a queue manager restart in the event of an abnormal termination, regularly monitor the values output from DISPLAY USAGE TYPE(DATASET).

In addition, the queue manager issues informational messages relating to these factors:

- CSQJ160I and CSQJ161I warn of long running units of work.
- CSQR026I and CSQR027I provide information about whether these long running units of work have been successfully shunted.
- CSQE040I and CSQE041E warn that structure backups are getting old, and consequently a RECOVER CFSTRUCT operation would take a long time.

Determining which application has a long running unit of work

It is possible to determine the application with the long-running unit of work. To do this, you use the DISPLAY CONN command.

The DISPLAY CONN command returns connection information for all the applications connected to the queue manager, together with additional information that helps you determine which application(s) currently have a long-running unit of work. The information returned by the DISPLAY CONN command is similar to the information returned by the DISPLAY QSTATUS command, but the main difference is that DISPLAY CONN displays information about objects, and transactional information for a particular connection, rather than details of which connections are associated with a particular object.

For each connected application, the DISPLAY CONN command returns the following information:

- Basic information including the Connection Id and PID.
- Transactional information for that connection, including the time and date when the transaction was created (that is, when the first MQGET/PUT was made under syncpoint), and when the transaction first wrote to the log.
- Log time information indicating which application still has a long running unit of work.
- A list of all objects that the connection currently has open. Details for each object are returned as a separate message, with the Connection Id used as a key. Because there are different types of object such as queues and queue managers, the information displayed with the object is specific to its object type.

Rebuilding queue indexes

To increase the speed of MQGET operations on a queue where messages are not retrieved sequentially, you can specify that you want IBM MQ to maintain an index of the message or correlation identifiers or groupid for all the messages on that queue.

When a queue manager is restarted, these indexes are rebuilt for each queue. This applies only to persistent messages; nonpersistent messages are deleted at restart. If your indexed queues contain large numbers of persistent messages, this increases the time taken to restart the queue manager.

You can choose to have indexes rebuilt asynchronously to queue manager startup by using the QINDXBLD parameter of the CSQ6SYSP macro. If you set QINDXBLD=NOWAIT, IBM MQ restarts without waiting for indexes to rebuild.

z/OS How in-doubt units of recovery are resolved

If IBM MQ loses its connection to another resource manager, it typically attempts to recover all inconsistent objects at restart.

If IBM MQ loses its connection to CICS, IMS, or RRS, it normally attempts to recover all inconsistent objects at restart. The information required to resolve in-doubt units of recovery must come from the coordinating system. The next sections describe the process of resolution for different environments.

- How in-doubt units of recovery are resolved from CICS
- How in-doubt units of recovery are resolved from IMS
- How in-doubt units of recovery are resolved from RRS
- How in-doubt units of recovery with a GROUP unit of recovery disposition are resolved

How in-doubt units of recovery are resolved from CICS

Under some circumstances, CICS cannot run the IBM MQ process to resolve in-doubt units of recovery. When this happens, IBM MQ sends one of the following messages:

- CSQC404E
- CSQC405E
- CSQC406E
- CSQC407E

followed by the message CSQC408I.

For details of what these messages mean, see the IBM MQ for z/OS mensajes, finalización, y códigos de razón manual.

The resolution of in-doubt units does not effect CICS resources. CICS is in control of recovery coordination and, when it restarts, automatically commits or backs out each unit, depending on whether there was a log record marking the beginning of the commit. The existence of in-doubt objects does not lock CICS resources while IBM MQ is being reconnected.

One of the functions of the CICS adapter is to keep data synchronized between CICS and IBM MQ. If a queue manager abends while connected to CICS, it is possible for CICS to commit or back out work without IBM MQ being aware of it. When the queue manager restarts, that work is termed *in doubt*.

IBM MQ cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to IBM MQ resources) until the connection to CICS is restarted or reconnected.

A process to resolve in-doubt units of recovery is initiated during startup of the CICS adapter. The process starts when the adapter requests a list of in-doubt units of recovery. Then:

- The adapter receives a list of in-doubt units of recovery for this connection ID from IBM MQ, and passes them to CICS for resolution.
- CICS compares entries from this list with entries in its own log. CICS determines from its own list what action it took for each in-doubt unit of recovery.

For all resolved units, IBM MQ updates the queues as necessary and releases the corresponding locks. Unresolved units can remain after restart. Resolve them by the methods described in the Administración IBM MQ for z/OS.

How in-doubt units of recovery are resolved from IMS

Resolving in-doubt units of recovery in IMSdoes not effect DL/I resources. IMS is in control of recovery coordination and, when it restarts, automatically commits or backs out incomplete DL/I work. The decision to commit or back out for online regions (non-fast-path) is on the presence or absence of IMS log record types X'3730' and X'3801'. The existence of in-doubt units of recovery does not imply that DL/I records are locked until IBM MQ connects.

During queue manager restart, IBM MQ makes a list of in-doubt units of recovery. IMS builds its own list of residual recovery entries (RREs). The RREs are logged at IMS checkpoints until all entries are resolved.

During reconnection of an IMS region to IBM MQ, IMS indicates to IBM MQ whether to commit or back out units of work marked in IBM MQ as in doubt.

When in-doubt units are resolved:

1. If IBM MQ recognizes that it has marked an entry for commit and IMS has marked it to be backed out, IBM MQ issues message CSQQ010E. IBM MQ issues this message for all inconsistencies of this type between IBM MQ and IMS.
2. If IBM MQ has any remaining in-doubt units, the adapter issues message CSQQ008I.

For all resolved units, IBM MQ updates queues as necessary and releases the corresponding locks.

IBM MQ maintains locks on in-doubt work that was not resolved. This can cause a backlog in the system if important locks are being held. The connection remains active so you can resolve the IMS RREs. Recover the in-doubt threads by the methods described in the Administración IBM MQ for z/OS.

All in-doubt work should be resolved unless there are software or operating problems, such as with an IMS cold start. In-doubt resolution by the IMS control region takes place in two circumstances:

1. At the start of the connection to IBM MQ, during which resolution is done synchronously.
2. When a program abends, during which the resolution is done asynchronously.

How in-doubt units of recovery are resolved from RRS

One of the functions of the RRS adapter is to keep data synchronized between IBM MQ and other RRS-participating resource managers. If a failure occurs when IBM MQ has completed phase one of the commit and is waiting for a decision from RRS (the commit coordinator), the unit of recovery enters the in-doubt state.

When communication is reestablished between RRS and IBM MQ, RRS automatically commits or backs out each unit of recovery, depending on whether there was a log record marking the beginning of the commit. IBM MQ cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to IBM MQ resources) until the connection to RRS is reestablished.

Under some circumstances, RRS cannot resolve in-doubt units of recovery. When this happens, IBM MQ sends one of the following messages to the z/OS console:

- CSQ3011I
- CSQ3013I
- CSQ3014I
- CSQ3016I

For details of what these messages mean, see the IBM MQ for z/OS mensajes, finalización, y códigos de razón manual.

For all resolved units of recovery, IBM MQ updates the queues as necessary and releases the corresponding locks. Unresolved units of recovery can remain after restart. Resolve them by the method described in the Administración IBM MQ for z/OS.

How in-doubt units of recovery with a GROUP unit of recovery disposition are resolved

In-doubt transactions that have a GROUP unit of recovery disposition can be resolved by the transaction coordinator by any queue manager in the queue sharing group (QSG) where the GROUPUR queue manager attribute is enabled. Whenever a transaction coordinator reconnects it typically requests a list of any outstanding in-doubt transactions and then resolves them using information from its logs.

When a transaction coordinator, that has connected with a GROUP unit of recovery disposition, requests the list of in-doubt transactions, the list returned comprises all in-doubt transactions with a GROUP unit of recovery disposition that exist throughout the queue sharing group. This list is not dependent on which queue manager those in-doubt transactions were started on. A queue manager processing such a request compiles the list by communicating with all other active queue managers in the queue sharing group using the SYSTEM.QSG.UR.RESOLUTION.QUEUE. The queue manager then reads the logs of any inactive queue

managers, from their last checkpoint, to identify any additional in-doubt transactions that they would have reported had they been active.

When a transaction coordinator requests the resolution of an in-doubt transaction, the queue manager to which it is connected identifies whether the transaction was originated on itself and if so resolves it in the same way as transactions with a QMGR unit of recovery disposition. If the transaction was originated on another active queue manager in the QSG, a request to complete the resolution is routed to that queue manager using the SYSTEM.QSG.UR.RESOLUTION.QUEUE. In the case where the transaction was originated on an inactive queue manager in the QSG, any shared-queue work is resolved immediately, and a request to resolve any remaining private queue work is placed on the SYSTEM.QSG.UR.RESOLUTION.QUEUE. The inactive queue manager processes this request upon start-up before accepting new work. In this scenario, the original queue manager's logs still reflect that the unit of recovery is in doubt until it has restarted and processed the request.

z/OS Shared queue recovery

Use this topic to understand IBM MQ recovery and resilience of various components in the queue sharing group environment.

Transactional recovery

When an application issues a MQBACK call or terminates abnormally (for example, because of an EXEC CICS ROLLBACK or an IMS abend) thread-level information stored in the queue manager ensures that the in-flight unit of work is rolled back. MQPUT and MQGET operations within syncpoint on shared queues are rolled back in the same way as updates to non-shared queues.

Peer recovery

If a queue manager fails, it disconnects abnormally from the coupling facility structures that it is currently connected to. If the connection between the z/OS instance and the coupling facility fails (for example, physical link failure or power-off of a coupling facility or partition) this is also detected as an abnormal termination of the connection between the queue manager and the coupling facility structures involved. Other queue managers in the same queue sharing group that remain connected to that structure detect the abnormal disconnection and all attempt to initiate *peer recovery* for the failed queue manager on that structure. Only one of these queue managers initiates peer recovery successfully, but all the other queue managers cooperate in the recovery of units of work that were owned by the queue manager that failed.

If a queue manager fails when there are no peers connected to a structure, recovery is performed when another queue manager connects to that structure, or when the queue manager that failed restarts.

Peer recovery, often referred to as Peer Level Recovery (PLR), is performed on a structure by structure basis and it is possible for a single queue manager to participate in the recovery of more than one structure at the same time. However, the set of peers cooperating in the recovery of different structures might vary depending on which queue managers were connected to the different structures at the time of failure.

When the failed queue manager restarts, it reconnects to the structures that it was connected to at the time of failure, and recovers any remaining unresolved units of work that were not recovered by peer recovery.

Peer recovery is a multi-phase process. During the first phase, units of work that had progressed beyond the in-flight phase are recovered; this might involve committing messages for units of work that are in-commit and locking messages for units of work that are in-doubt. During the second phase, queues that had threads active against them in the failing queue manager are checked, uncommitted messages related to in-flight units of work are rolled back, and information about active handles on shared queues in the failed queue manager are reset. This means that IBM MQ resets any indicators that the failing queue manager had a shared queue open for input-exclusive, allowing other active queue managers to open the queue for input.

Shared queue definitions

The queue objects that represent the attributes of a shared queue are held in the shared Db2 repository used by the queue sharing group. Ensure that adequate procedures are in place for the backup and recovery of the Db2 tables used to hold IBM MQ objects. You can also use the IBM MQ CSQUTIL utility to create MQSC commands for replay into a queue manager to redefine IBM MQ objects, including shared queue and group definitions stored in Db2.

Logging

Queue sharing groups can support persistent messages, because the messages on shared queues can be logged in the queue manager logs.

Coupling facility and structure failures

There are two types of failure that can be reported for a coupling facility (CF) structure: structure failure and loss of connectivity. Sysplex services for data sharing (XES) inform IBM MQ of a CF structure failure or a CF failure with a structure failure event. If XES creates a loss of connectivity event this does not necessarily indicate that there is a problem with the structure, it might be that there is no connection available to communicate with the structure. It is possible that not all of the queue managers receive a loss of connectivity event for the structure; it depends on the configuration of connections to the CF. A loss of connectivity event can also be received because of operator commands, for example VARY PATH OFFLINE or CONFIG CHP OFFLINE.

The CF structures that are used by IBM MQ can be configured to use system-managed duplexing. This means that if there is a single failure, system-managed failover processing hides the failure of a structure or the loss of connectivity, and the queue manager is not informed of the failure. If there is a failure of both instances of a duplexed structure or connection, the queue manager receives the appropriate event and handles it in the same way as a failure event for a simplex structure. Details of how the queue manager handles the events are described in Scenarios.

In the unlikely event of a CF or structure failure, any nonpersistent messages stored in the affected application structures are lost. You can recover persistent messages using the RECOVER CFSTRUCT command. If a recoverable application structure has failed, any further application activity to this structure is prevented until the structure has been recovered.

To ensure that you can recover a CF structure in a reasonable period of time, take frequent backups, using the BACKUP CFSTRUCT command. You can choose to perform the backups on any queue managers in the queue sharing group or dedicate one queue manager to perform all the backups. Automate the process of taking backups to ensure that they are taken on a regular basis.

Each backup is written to the active log data set of the queue manager taking the backup. The shared queue Db2 repository records the name of the CF structure being backed up, the name of the queue manager doing the backup, the RBA range for this backup on that queue manager's log, and the backup time.

The administration structure contains information about incomplete units of work on shared queues at the time of any application structure failure so the administration structure must be available during RECOVER CFSTRUCT processing. If the administration structure has failed, all the queue managers in the queue sharing group must have rebuilt their administration structure entries before you can issue the RECOVER CFSTRUCT command.

Queue managers rebuild their administration structure entries automatically and without terminating. If a queue manager is not running at the time of the failure, its administration structure entries can be rebuilt by another queue manager in the queue sharing group that is running at the same or higher level.

To recover an application structure, issue a RECOVER CFSTRUCT command to the queue manager that you want to perform the recovery. You can recover a single CF structure or you can recover several CF structures simultaneously. You can recover using any queue manager in the queue sharing group, it does not have to be the one that performed the backup, or one that has been connected previously to the failed structure.

The RECOVER CFSTRUCT command uses the backup, located through the Db2 repository information (Db2 must therefore be available on the queue manager where recovery is being carried out), and recovers this to the point of failure.

The RECOVER CFSTRUCT command does this by applying log records from every queue manager in the queue sharing group that has performed an MQPUT or MQGET between the start of the backup and the time of failure, to any shared queue that maps to the CF structure. The resulting merging of the logs might require reading a considerable amount of log data because all the log data written by participating queue managers since the backup is read. You are strongly recommended to make frequent (for example, hourly) backups, especially if there are large messages within the backup.

Structure failure scenarios

Scenarios

If a failure is reported for a CF structure, the action taken by connected queue managers depends on the following:

- The type of failure reported by the XES component of z/OS to IBM MQ.
- The structure type (application or administration)
- The queue manager level
- The CFLEVEL of the IBM MQ CFSTRUCT object (2, 3, 4 or 5. This is not the CFLEVEL of the CFCC microcode)
- The RECAUTO attribute of an IBM MQ CFSTRUCT object at CFLEVEL(5)

The following scenarios describe what happens when a failure is reported for the administration structure:

- If a structure failure event is received for the administration structure, the structure is reallocated and rebuilt automatically without the queue manager terminating. A new instance of the structure is allocated by XES when a queue manager attempts to connect to it.

  When the queue manager has connected to the new instance of the structure, the queue manager writes the entries for itself into the structure. This processing is carried out by the queue manager and is not part of XES rebuild processing.

  If a queue manager was not running at the time of the failure, or terminates before recovery of its part of the administration structure has been completed, its administration structure entries can be rebuilt by another queue manager in the queue sharing group.

  Administration structure entries of a queue manager can only be rebuilt by another queue manager running at the same level or higher. If administration structure entries of a queue manager cannot be rebuilt by another queue manager in the queue sharing group, restart the queue manager so that it can complete the rebuild of its part of the structure.

Certain actions are suspended until the administration structure entries for all queue managers have been rebuilt. The suspended actions include the following:

- Opening and closing of shared queues.
- Committing or backing out units of recovery.
- Serialized applications connecting to or disconnecting from the queue manager.
- Backing up or recovering an application structure.

Any serialized applications that have already connected to the queue manager can continue processing. Any serialized application attempting to connect with the MQCNO_SERIALIZE_CONN_TAG_QSG or MQCNO_RESTRICT_CONN_TAG_QSG parameters receive the MQRC_CONN_TAG_NOT_USABLE return code.

When the administration structure entries for the queue manager have been rebuilt, the suspended actions are resumed.

The following scenarios describe what happens when a failure is reported for an application structure:

- If a structure failure event is received for an application structure, and the CFLEVEL is 1 or 2, the queue manager terminates. Restart the queue manager. The first queue manager to attempt to connect to the structure again causes XES to allocate a new instance of the structure.
- If a structure failure event is received for an application structure, and the CFLEVEL is 3, 4, or 5 the queue managers connected to the structure continue to run. Applications that do not use the queues in the failed structure can continue normal processing.

  However, applications that attempt operations on queues in the failed structure receive an MQRC_CF_STRUC_FAILED error until the structure has been successfully rebuilt, at which point the application can open the queues again.

  Structure rebuild is started automatically for CFLEVEL(5) application structures defined with RECAUTO(YES). Otherwise, the structure will be rebuilt when the RECOVER CFSTRUCT command is issued.

Resilience to coupling facility connectivity failures

What is resilience to coupling facility connectivity failures?

Resilience to coupling facility connectivity failures refers to the ability of queue managers in a queue sharing group to tolerate loss of connectivity to a coupling facility structure without terminating. This function also attempts to rebuild the structure in another coupling facility with better connectivity in order to regain access to shared queues as soon as possible.

What is partial loss of connectivity?

IBM MQ defines partial loss of connectivity as a situation where one or more systems in the sysplex lose connectivity to the coupling facility where the structure being accessed by the system is allocated, but at least one system in the sysplex maintains connectivity to the same coupling facility.

What is total loss of connectivity?

IBM MQ defines a total loss of connectivity as a situation where no systems in the sysplex have connectivity to the coupling facility and the structure allocated within it.

Why would you enable this function?

Resilience to coupling facility connectivity failures improves the availability of IBM MQ, allowing non-shared queues to remain available after a queue manager has lost connectivity to one or more coupling facility structures. Additionally, queue managers that lose connectivity to a coupling facility structure automatically attempt to rebuild the structure in another available coupling facility, improving the availability of the shared queues within the queue sharing group.

Considerations when enabling this function

A queue manager that tolerates loss of connectivity to coupling facility structures without terminating may not be able to reconnect to a coupling facility structure for some time if there is no alternative coupling facility available. Shared queues defined on a structure that has suffered loss of connectivity remain unavailable until connectivity to the structure is restored. In this situation, applications that connect into members of the queue sharing group in order to perform shared queue work may find that the shared queues they need to access are not available. To avoid this situation it is recommended that queue managers should be configured to terminate when connectivity to a coupling facility structure is lost. This termination forces applications to connect to another member of the queue sharing group that has connectivity to the coupling facility structures where the shared queues the application requires are defined.

Managing Resilience to coupling facility connectivity failures

How do I enable this functionality?

The following steps must be performed in order to enable resilience to coupling facility connectivity

1. Ensure that the CFRM couple data set has been formatted to support system-managed rebuild. This allows queue managers to initiate a system-managed rebuild to re-create a structure into an available coupling facility. Use the **DISPLAY XCF,COUPLE,TYPE=CFRM** command to determine the format of the CFRM couple data set. To support system-managed rebuild, the CFRM couple data set should be formatted by specifying:

   ```
   "ITEM NAME(SMREBLD) NUMBER(1)"
   ```

   Refer to the z/OS MVS Setting Up a Sysplex documentation for more information on formatting a CFRM couple data set.
2. Ensure that an alternative coupling facility is available and is in the CFRM preference list for all IBM MQ coupling facility structures. This enables the queue managers to attempt to rebuild structures into an alternative available coupling facility to restore access to the structures as soon as possible.

   IBM MQ structures must be defined with ENFORCEORDER(NO) in CFRM policy, so that XCF is able to choose the optimum CF in the configuration if IBM MQ needs to reallocate the structure.

   Refer to the z/OS MVS Setting Up a Sysplex documentation for more information about structure preference lists.
3. Alter all application coupling facility structures that need to tolerate loss of connectivity to CFLEVEL(5). This is the minimum level that can tolerate a loss of connectivity.
4. Determine the values required for the **QMGR CFCONLOS** and the **CFSTRUCT CFCONLOS** attributes and alter these accordingly. The **QMGR CFCONLOS** attribute controls whether loss of connectivity to the administration structure is tolerated, and the **CFSTRUCT CFCONLOS** attribute controls whether loss of connectivity is tolerated by each application coupling facility structure. If the default values for these attributes are retained, the queue manager terminates following loss of connectivity to any coupling facility structure.
5. Determine the values required for the **CFSTRUCT RECAUTO** attribute for each application coupling facility structure, and alter these accordingly. This attribute controls whether coupling facility structures should be automatically recovered using logged data following total loss of connectivity. If the default value for this attribute is retained, no automatic recovery is performed for application structures following total loss of connectivity.

Scenario 1 - Loss of connectivity to the administration structure

Queue managers can tolerate loss of connectivity to the administration structure without terminating.

When connectivity to the administration structure is lost by any queue manager that has been configured to tolerate loss of connectivity to the administration structure, all members of the queue sharing group disconnect from the administration structure. All active queue managers in the queue

sharing group then attempt to reconnect to the administration structure, causing it to be reallocated in the coupling facility with the best connectivity to all systems in the sysplex, and rebuild the administration structure data.

Note: This may not necessarily be the coupling facility which has the best connectivity to all systems that have active queue managers.

If a queue manager cannot reconnect to the administration structure, for example because none of the coupling facilities in the CFRM preference list for the administration structure are available, some shared queue operations remain unavailable until the queue manager can successfully reconnect to the administration structure and rebuild its administration structure data. Reconnection occurs automatically when a suitable coupling facility becomes available on the system.

Failure to connect to the administration structure during queue manager startup as a result of a lack of connectivity to the coupling facility, or no suitable coupling facility available to allocate the structure, is not tolerated. All active queue managers in the queue sharing group then attempt to reconnect to the administration structure, causing it to be reallocated in another coupling facility if one is available, and rebuild the administration structure data.

Scenario 2- Loss of connectivity to the application structure

Loss of connectivity to application structures at **CFLEVEL(5)** or higher can be tolerated without the queue manager terminating. Queue managers connected to application structures at **CFLEVEL(4)** or lower, or structures at **CFLEVEL(5)** that have not been configured to tolerate loss of connectivity, abend with reason code 00C510AB when connectivity to the structure is lost.

When connectivity is lost to an application structure that has been configured to tolerate loss of connectivity, all queue managers that lost connectivity to the structure disconnect. The subsequent behavior of the queue manager depends on whether the loss of connectivity is partial or total.

Partial loss of connectivity to an application structure

If the loss of connectivity is determined to be partial, queue managers that have lost connectivity to the structure attempt to initiate a system-managed rebuild in order to move the structure to another coupling facility with improved connectivity. If this rebuild is successful, both persistent and non-persistent messages in the structure are copied to the other coupling facility, and access to queues on the structure is restored. Queue managers that did not lose connectivity remain connected to the structure, however, operations that access the structure are delayed during the system-managed rebuild process.

If an application structure cannot be rebuilt to another coupling facility with improved connectivity, or some queue managers still do not have connectivity to the structure after it has been rebuilt in another coupling facility, queues defined on the structure remain unavailable on the queue managers that do not have connectivity to the structure until connectivity is restored to the coupling facility. Queue managers automatically reconnect to the structure when it becomes available and access to the shared queues defined on the structure are restored.

Total loss of connectivity to an application structure

If all MVS systems in the sysplex have lost connectivity to the coupling facility that the application structure is allocated in, z/OS deallocates the structure from the coupling facility whenever an attempt is made to reconnect to the structure. It is possible for the queue manager to attempt to reconnect to the structure for several reasons, such as an attempt by an application to open a shared queue, or a notification from the system that new coupling facility resources may have become available. It is therefore likely that all non-persistent messages in the affected structure are lost following total loss of connectivity to an application structure.

Recoverable application structures are automatically recovered following total loss of connectivity, if they have been defined with **RECAUTO(YES)**. The recovery starts almost immediately if an alternative coupling facility is available to allocate the structure in, or whenever such a coupling facility becomes available. If a structure has not been defined with **RECAUTO(YES)**, recovery can be started by issuing the **RECOVER CFSTRUCT** command. This recovers all persistent messages in the structure, but all non-persistent messages are lost. As this process involves reading the queue manager log it can take

some time to complete, therefore it is recommended that structure backups be taken regularly to reduce the time until access to the shared queues on the structure is restored.

Queue managers attempt to reconnect to non-recoverable application structures as soon as an application attempts to open a shared queue that is defined on the structure or a notification is received from the system that new coupling facility resources have become available. If a suitable coupling facility is available to allocate the structure in, a new structure is allocated and access to the shared queues defined on the structure is restored. As persistent messages cannot be put to queues defined in non-recoverable structures, all messages on the shared queues are lost.

Operational behavior

If an IBM WebSphere MQ 7.1, or later, queue manager, configured to tolerate loss of connectivity to a particular coupling facility structure loses connectivity, the members of the queue sharing group attempt to automatically recover from the failure and reconnect to the structure. This activity may involve reallocating the structure in another coupling facility with better connectivity if one is available. However, operator intervention may still be required to recover from the loss of connectivity.

Typically the required operator action is to:

1. Resolve the cause of the failure that resulting in the loss of connectivity.
2. Ensure that a coupling facility where the IBM MQ structures can be allocated is available on all systems in the sysplex

Any structures that have been automatically reallocated in another coupling facility after the loss of connectivity event, can be moved to the coupling facility with the optimal connectivity to all queue managers in the queue sharing group. If required, this can be done by initiating the system-managed rebuild command **SETXCF START,REBUILD** as documented in z/OS MVS System Commands Reference.

In the case of a partial loss of connectivity to an application structure, the queue managers that lost connectivity to the structure attempt to initiate a system-managed rebuild. This process only allocates the structure in another coupling facility if that coupling facility has connectivity to all active queue managers currently connected to the structure. Therefore, it is possible that where the majority of queue managers in a queue sharing group have lost connectivity to an application structure, they are unable to rebuild the structure into another coupling facility due to the queue managers that are still connected to the original structure. In this situation the queue managers that are still connected to the original structure can be shut down to allow the structure to be rebuilt, or the **RESET CFSTRUCT ACTION(FAIL)** command can be issued to fail the structure. Recovery can be initiated on applicable structures by issuing the **RECOVER CFSTRUCT** command.

Note: When failing and recovering the structure, all non-persistent messages on the structure are lost.

z/OS Security concepts in IBM MQ for z/OS

Use this topic to understand the importance of security for IBM MQ, and the implications of not having adequate security settings on your system.

Why you must protect IBM MQ resources

IBM MQ handles the transfer of information that is potentially valuable. Applying security ensures that the resources IBM MQ owns and manages are protected from unauthorized access. Such access might lead to the loss or disclosure of the information.

You should ensure that none of the following resources are accessed or changed by any unauthorized user or process:

- Connections to IBM MQ
- IBM MQ objects such as queues, processes, and namelists
- IBM MQ transmission links, that is, IBM MQ channels
- IBM MQ system control commands

- IBM MQ messages
- Context information associated with messages

To provide the necessary security, IBM MQ uses the z/OS system authorization facility (SAF) to route authorization requests to an External Security Manager (ESM), for example Security Server (previously known as RACF). IBM MQ does no security verification of its own. Where distributed queuing or clients are being used, you might require additional security measures, for which IBM MQ provides channel authentication records, channel exits, the MCAUSER channel attribute, and TLS.

The decision to allow access to an object is made by the ESM and IBM MQ follows that decision. If the ESM cannot make a decision, IBM MQ prevents access to the object.

What happens if you do not protect IBM MQ resources

If you do nothing about security, the most likely effect is that *all* users can access and change *every* resource. This includes not only local users, but also those on remote systems using distributed queuing or clients, where the logon security controls might be less strict than is normally the case for z/OS.

To enable security checking you must do the following:

- Install and activate an ESM (for example, Security Server).
- Define the MQADMIN class if you are using an ESM other than Security Server.
- Activate the MQADMIN class.

You must consider whether using mixed-case resource names would be beneficial to your enterprise. If you do use mixed-case resource names in your ESM profiles you must define and activate the MXADMIN class.

z/OS Data Set Encryption

Data Set Encryption (DSE) provides the capability to encrypt z/OS data sets, so that the data they contain can only be viewed or modified by user IDs granted the specific permission. This provides encryption of data at rest in the file system, and prevents inadvertent disclosure of sensitive information to users who have a legitimate business need and permissions to manage the data sets themselves.

Prior to IBM MQ for z/OS 9.1.4, IBM MQ for z/OS does not support use of DSE with the active logs, page sets, and shared message data sets (SMDS) that provide the primary persistence mechanisms for IBM MQ messages.

Instead, Advanced Message Security provides an end-to-end encryption solution for IBM MQ messaging, which encompasses the entire IBM MQ network, encryption of data in flight, at rest, and even inside the runtime IBM MQ processes.

Other VSAM and sequential data sets used in an IBM MQ subsystem can be encrypted using DSE. For example:

- Bootstrap data set (BSDS)
- Sequential files holding system configuration (MQSC) commands read at startup using CSQINPx DDNAMEs
- IBM MQ archive logs, often used for long term archival of IBM MQ log data for audit purposes.

You can encrypt using DSE by allocating a dataclass that is defined with a data set key label. For more information, see Planning your log archive storage.

From IBM MQ for z/OS 9.1.4, IBM MQ for z/OS supports use of DSE with the active logs and page sets in addition to the support provided in earlier releases.

IBM MQ for z/OS does not support use of DSE for shared message data sets (SMDS).

See the section, confidentiality for data at rest on IBM MQ for z/OS with data set encryption. for more information.

Related concepts
Security concepts
Channel authentication records
Authority to work with IBM MQ objects on z/OS
Cryptographic security protocols: TLS
Related tasks
Setting up security on z/OS
Comparing link level security and application level security
Related reference
Messages for IBM MQ for z/OS

z/OS Security controls and options in IBM MQ for z/OS

You can specify whether security is turned on for the whole IBM MQ subsystem, and whether you want to perform security checks at queue manager or queue sharing group level. You can also control the number of user IDs checked for API-resource security.

Subsystem security

Subsystem security is a control that specifies whether any security checking is done for the whole queue manager. If you do not require security checking (for example, on a test system), or if you are satisfied with the level of security on all the resources that can connect to IBM MQ (including clients and channels), you can turn security checking off for the queue manager or queue sharing group so that no further security checking takes place.

This is the only check that can turn security off completely and determine whether any other security checks are performed or not. That is, if you turn off checking for the queue manager or queue sharing group, no other IBM MQ checking is done; if you leave it turned on, IBM MQ checks your security requirements for other IBM MQ resources.

You can also turn security on or off for particular sets of resources, such as commands.

Queue manager or queue sharing group level checking

You can implement security at queue manager level or at queue sharing group level. If you implement security at queue sharing group level, all the queue managers in the group share the same profiles. This means that there are fewer profiles to define and maintain, making security management easier. It also makes it easy to add a new queue manager to the queue sharing group because it inherits the existing security profiles.

It is also possible to implement a combination of both if your installation requires it, for example, during migration or if you have one queue manager in the queue sharing group that requires different levels of security to the other queue managers in the group.

Queue sharing group level security

Queue sharing group level security checking is performed for the entire queue sharing group. It enables you to simplify security administration because it requires you to define fewer security profiles. The authorization of a user ID to use a particular resource is handled at the queue sharing group level, and is independent of which queue manager that user ID is using to access the resource.

For example, say a server application runs under user ID SERVER and wants access to a queue called SERVER.REQUEST, and you want to run an instance of SERVER on each z/OS image in the sysplex. Rather than permitting SERVER to open SERVER.REQUEST on each queue manager individually (queue manager level security), you can permit access only at the queue sharing group level.

You can use queue sharing group level security profiles to protect all types of resource, whether local or shared.

Queue manager level security
You can use queue manager level security profiles to protect all types of resource, whether local or shared.

Combination of both levels
You can use a combination of both queue manager and queue sharing group level security.

You can override queue sharing group level security settings for a particular queue manager that is a member of that group. This means that you can perform a different level of security checks on an individual queue manager to those performed on the other queue managers in the group.

For more information, see Profiles to control queue sharing group or queue manager level security.

Controlling the number of user IDs checked

RESLEVEL is a Security Server profile that controls the number of user IDs checked for IBM MQ resource security. Normally, when a user attempts to access an IBM MQ resource, Security Server checks the relevant user ID or IDs to see if access is allowed to that resource. By defining a RESLEVEL profile you can control whether zero, one or, where applicable, two user IDs are checked.

These controls are done on a connection by connection basis, and last for the life of the connection.

There is only one RESLEVEL profile for each queue manager. Control is implemented by the access that a user ID has to this profile.

Mixed case or uppercase IBM MQ RACF classes

You can now use mixed case RACF profile support, which allows you to use mixed case resource names and define IBM MQ RACF profiles to protect them.

You can choose to either:

- Continue using uppercase only IBM MQ RACF Classes as in previous releases, or
- Use the new mixed case IBM MQ RACF classes.

Without the use of mixed case RACF profiles, you can still use mixed case resource names in IBM MQ for z/OS ; however, these resource names can only be protected by generic RACF profiles in the uppercase IBM MQ classes. When using mixed case IBM MQ RACF profile support you can provide a more granular level of protection by defining IBM MQ RACF profiles in the mixed case IBM MQ classes.

z/OS Resources you can protect in IBM MQ for z/OS

When a queue manager starts, or when instructed by an operator command, IBM MQ for z/OS determines which resources you want to protect.

You can control which security checks are performed for each individual queue manager. For example, you can implement a number of security checks on a production queue manager, but none on a test queue manager.

Connection security

Connection security checking is carried out either when an application program tries to connect to a queue manager. It is done by issuing an `MQCONN` or `MQCONNX` request, or when the channel initiator, or CICS or IMS adapter issues a connection request.

If you are using queue manager level security, you can turn connection security checking off for a particular queue manager. However, if you do this any user can connect to that queue manager.

For the CICS adapter, only the CICS address space user ID is used for the connection security check, not the individual CICS terminal user ID. For the IMS adapter, when the IMS control or dependent regions connect to IBM MQ, the IMS address space user ID is checked. For the channel initiator, the user ID used by the channel initiator address space is checked.

You can turn connection security checking on or off at either queue manager or queue sharing group level.

Command security

Command security checking is carried out when a user issues an MQSC command from any of the sources described in Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS. You can make a separate check on the resource specified by the command as described in “Command resource security” on page 273.

If you turn off command checking, issuers of commands are not checked to see whether they have the authority to issue the command.

If MQSC commands are entered from a console, the console must have the z/OS SYS console authority attribute. Commands that are issued from the CSQINP1 or CSQINP2 data sets, or internally by the queue manager, are exempt from all security checking while those for CSQINPX use the user ID of the channel initiator address space. You must control who is allowed to update these data sets through normal data set protection.

You can turn command security checking on or off at either queue manager or queue sharing group level.

Command resource security

Some MQSC commands, for example defining a local queue, involve the manipulation of IBM MQ resources. When command resource security is active, each time a command involving a resource is issued, IBM MQ checks to see if the user is allowed to change the definition of that resource.

You can use command resource security to help enforce naming standards. For example, a payroll administrator might be allowed to delete and define only queues with names beginning "PAYROLL".
If command resource security is inactive, no security checks are made on the resource that is being manipulated by the command. Do not confuse command resource security with command security; the two are independent.

Turning off command resource security checking does not affect the resource checking that is done specifically for other types of processing that do not involve commands.

You can turn command resource security checking on or off at either queue manager or queue sharing group level.

Channel security considerations

Channel security

When you are using channels, the security features available depend on which communications protocol you are going to use. If you use TCP, there are no security features provided with the communications protocol, although you can use TLS. If you are using APPC, you can flow user ID information from the sending MCA through the network to the destination MCA for verification.

For both protocols, you can specify which user IDs you want to check for security purposes, and how many. Again, the choices available to you depend on which protocol you are using, what you specify when you define the channel, and the RESLEVEL settings for the channel initiator.

For more information about the types of channel security available see Channel authentication records and Security exit overview

Related reference

"API-resource security in IBM MQ for z/OS" on page 274
Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on what open options are specified when the queue is opened.

z/OS API-resource security in IBM MQ for z/OS

Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on what open options are specified when the queue is opened.

API-resource security is subdivided into the following checks:

- Queue
- Process
- Namelist
- Alternate user
- Context

No security checks are performed when opening the queue manager object or when accessing storage class objects.

Queue

Queue security checking controls who is allowed to open which queue, and what options they are allowed to open it with. For example, a user might be allowed to open a queue called PAYROLL.INCREASE.SALARY to browse the messages on the queue (using the MQOO_BROWSE option), but not to remove messages from the queue (using one of the MQOO_INPUT_* options). If you turn checking for queues off, any user can open any queue with any valid open option (that is, any valid MQOO_* option on an MQOPEN or MQPUT1 call).

You can turn queue security checking on or off at either queue manager or queue sharing group level.

Process

Process security checking is carried out when a user opens a process definition object. If you turn checking for processes off, any user can open any process.

You can turn process security checking on or off at either queue manager or queue sharing group level.

Namelist

Namelist security checking is carried out when a user opens a namelist. If you turn checking for namelists off, any user can open any namelist.

You can turn namelist security checking on or off at either queue manager or queue sharing group level.

Alternate user

Alternate user security controls whether one user ID can use the authority of another user ID to open an IBM MQ object.

For example:

- A server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1.
- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.

- Instead of using its own user ID (PAYSERV) to authorize opening the reply-to queue, the server can specify some other user ID, in this case, USER1. In this example, alternate user security would control whether user ID PAYSERV is allowed to specify user ID USER1 as an alternative user ID when opening the reply-to queue.

The alternative user ID is specified in the *AlternateUserId* field of the object descriptor (MQOD).

You can use alternative user IDs on any IBM MQ object, for example, processes or namelists. It does not affect the user ID used by any other resource managers, for example, for CICS security or for z/OS data set security.

If alternate user security is not active, any user can use any other user ID as an alternative user ID.

You can turn alternate user security checking on or off at either queue manager or queue sharing group level.

Context

Context is information that is applicable to a particular message and is contained in the message descriptor (MQMD) that is part of the message. The context information comes in two sections:

Identity section
The user of the application that first put the message to a queue. It consists of the following fields:

- *UserIdentifier*
- *AccountingToken*
- *ApplIdentityData*

Origin section
The application that put the message on the queue where it is currently stored. It consists of the following fields:

- *PutApplType*
- *PutApplName*
- *PutDate*
- *PutTime*
- *ApplOriginData*

Applications can specify the context data when either an MQPUT or an MQPUT1 call is made. The application might generate the data, the data might be passed on from another message, or the queue manager might generate the data by default. For example, server programs can use context data to check the identity of the requester, that is, did this message come from the correct application? Typically, the *UserIdentifier* field is used to determine the user ID of an alternative user.

You use context security to control whether the user can specify any of the context options on any MQOPEN or MQPUT call. For information about the context options, see the MQOPEN options relating to message context. For descriptions of the message descriptor fields relating to context, see MQMD - Message descriptorMQMD - Message descriptor.

If you turn context security checking off, any user can use any of the context options that the queue security allows.

You can turn context security checking on or off at either queue, queue manager or queue sharing group level.

z/OS Availability on z/OS

IBM MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

Several features of IBM MQ can increase system availability if the queue manager or channel initiator fails. For more information about these features, see the following sections:

- Sysplex considerations

- Shared queues
- Shared channels
- IBM MQ network availability
- Using the z/OS Automatic Restart Manager (ARM)
- Using the z/OS Extended Recovery Facility (XRF)
- Using the z/OS GROUPUR attribute for recovery in a queue sharing group
- Where to find more information about availability

Sysplex considerations

In a *sysplex*, a number of z/OS operating system images collaborate in a single system image and communicate using a coupling facility. IBM MQ can use the facilities of the sysplex environment for enhanced availability.

Removing the affinities between a queue manager and a particular z/OS image allows a queue manager to be restarted on a different z/OS image in the event of an image failure. The restart mechanism can be manual, use ARM, or use system automation, if you ensure the following:

- All page sets, logs, bootstrap data sets, code libraries, and queue manager configuration data sets are defined on shared volumes.
- The subsystem definition has sysplex scope and a unique name within the sysplex.
- The level of *early code* installed on every z/OS image at IPL time is at the same level.
- TCP virtual IP addresses (VIPA) is available on each TCP stack in the sysplex, and you have configured IBM MQ TCP listeners and inbound connections to use VIPAs rather than default host names.

For more information about using TCP in a sysplex, see *TCP/IP in a sysplex*, SG24-5235, an IBM Redbooks® publication.

You can additionally configure multiple queue managers running on different operating system images in a sysplex to operate as a queue sharing group, which can take advantage of shared queues and shared channels for higher availability and workload balancing.

Shared queues

In the queue sharing group environment, an application can connect to any of the queue managers within the queue sharing group. Because all the queue managers in the queue sharing group can access the same set of shared queues, the application does not depend on the availability of a particular queue manager; any queue manager in the queue sharing group can service any queue. This gives greater availability if a queue manager stops because all the other queue managers in the queue sharing group can continue processing the queue. For information about high availability of shared queues, see “Advantages of using shared queues” on page 190.

To further enhance the availability of messages in a queue sharing group, IBM MQ detects if another queue manager in the group disconnects from the coupling facility abnormally, and completes units of work for that queue manager that are still pending, where possible. This is known as *peer recovery* and is described in “Peer recovery” on page 263.

Peer recovery cannot recover units of work that were in doubt at the time of the failure. You can use the Automatic Restart Manager (ARM) to restart all the systems involved in the failure (CICS, Db2, and IBM MQ for example), and to ensure that they are all restarted on the same new processor. This means that they can resynchronize, and gives rapid recovery of in-doubt units of work. This is described in “Using the z/OS Automatic Restart Manager (ARM)” on page 277.

Shared channels

In the queue sharing group environment, IBM MQ provides functions that give high availability to the network. The channel initiator enables you to use networking products that balance network requests across a set of eligible servers and hide server failures from the network (for example, VTAM generic resources). IBM MQ uses a generic port for inbound requests so that attach requests can be routed to any available channel initiator in the queue sharing group. This is described in “Shared channels” on page 210.

Shared outbound channels take the messages they send from a shared transmission queue. Information about the status of a shared channel is held in one place for the whole queue sharing group level. This means that a channel can be restarted automatically on a different channel initiator in the queue sharing group if the channel initiator, queue manager, or communications subsystem fails. This is called *peer channel recovery* and is described in Shared outbound channels.

IBM MQ network availability

IBM MQ messages are carried from queue manager to queue manager in an IBM MQ network using channels. You can change the configuration at a number of levels to improve the network availability of a queue manager, and the ability of an IBM MQ channel to detect a network problem and to reconnect.

TCP *Keepalive* is available for TCP/IP channels. It causes TCP to send packets periodically between sessions to detect network failures. The KAINT channel attribute determines the frequency of these packets for a channel.

AdoptMCA allows a channel, blocked in receive processing as a result of a network outage, to be terminated and replaced by a new connection request. You control AdoptMCA using the ADOPTMCA queue manager property with the MQSC utility or the AdoptNewMCAType property with the Programmable Command Formats interface.

ReceiveTimeout prevents a channel from being permanently blocked in a network receive call. The RCVTIME and RCVTMIN channel initiator parameters, determine the receive timeout characteristics for channels, as a function of their heartbeat interval. See Queue manager parameter for more details.

Using the z/OS Automatic Restart Manager (ARM)

You can use IBM MQ for z/OS in conjunction with the z/OS automatic restart manager (ARM). If a queue manager or a channel initiator has failed, ARM restarts it on the same z/OS image. If z/OS fails, a whole group of related subsystems and applications also fail. ARM can restart all the failed systems automatically, in a predefined order, on another z/OS image within the sysplex. This is called a cross-system restart.

ARM enables rapid recovery of in-doubt transactions in the shared queue environment. It also gives higher availability if you are not using queue sharing groups.

You can use ARM to restart a queue manager on a different z/OS image within the sysplex in the event of z/OS failure.

To enable automatic restart, you must do the following:

1. Set up an ARM coupling data set.
2. Define the automatic restart actions that you want z/OS to perform in an *ARM policy*.
3. Start the ARM policy.

If you want to restart queue managers in different z/OS images automatically, every queue manager in each z/OS image on which that queue manager might be restarted must be defined with a sysplex-wide unique 4-character subsystem name.

Using ARM with IBM MQ is described in Using ARM in an IBM MQ network.

Using the z/OS Extended Recovery Facility (XRF)

You can use IBM MQ in an extended recovery facility (XRF) environment. All IBM MQ-owned data sets (executable code, BSDSs, logs, and page sets) must be on DASD shared between the active and alternative XRF processors.

If you use XRF for recovery, you must stop the queue manager on the active processor and start it on the alternative processor. For CICS, you can do this using the command list table (CLT) provided by CICS, or the system operator can do it manually. For IMS, this is a manual operation and you must do it after the coordinating IMS system has completed the processor switch.

IBM MQ utilities must be completed or terminated before the queue manager can be switched to the alternative processor. Consider the effect of this potential interruption carefully when planning your XRF recovery plans.

Take care to prevent the queue manager starting on the alternative processor before the queue manager on the active processor terminates. A premature start can cause severe integrity problems in data, the catalog, and the log. Using global resource serialization (GRS) helps avoid the integrity problems by preventing simultaneous use of IBM MQ on the two systems. You must include the BSDS as a protected resource, and you must include the active and alternative XRF processors in the GRS ring.

Using the z/OS GROUPUR attribute for recovery in a queue sharing group

Queue sharing groups (QSG) allow additional transactional facilities which are described in this topic. The GROUPUR attribute allows XA client applications to have any in-doubt transaction recovery that may be required, performed on any member of the QSG.

If an XA client application connects to a queue sharing group (QSG) through a Sysplex it cannot guarantee which specific queue manager it connects to. Use of the GROUPUR attribute by queue managers within the queue sharing group can enable any in-doubt transaction recovery that may be necessary to occur on any member of the QSG. Even if the queue manager to which the application was initially connected is not available, transaction recovery can take place.

This feature frees the XA client application from any dependency on specific members of the QSG and thus extends the availability of the queue manager. The queue sharing group appears to the transactional application as a single entity providing all the IBM MQ features and without a single queue manager point of failure.

This functionality is not apparent to the transactional application.

Where to find more information about availability

You can find more information about these topics from the following sources:

Table 24. Where to find more information about availability

Topic	**Where to look**
Queue sharing groups	“Shared queues and queue sharing groups” on page 171
System parameters	Configuring system parameters
Using the Automatic Restart Manager Utility programs	Using ARM in an IBM MQ network
MQSC commands	MQSC commands

z/OS Monitoring and statistics on IBM MQ for z/OS

IBM MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

IBM MQ supplies facilities for monitoring the system and collecting statistics. For further information about these facilities, see the following sections:

- "Online monitoring" on page 279
- "IBM MQ trace" on page 279
- "Events" on page 279

Online monitoring

IBM MQ includes the following commands for monitoring the status of IBM MQ objects:

- DISPLAY CHSTATUS displays the status of a specified channel.
- DISPLAY QSTATUS displays the status of a specified queue.
- DISPLAY CONN displays the status of a specified connection.

For more information about these commands, see The MQSC commands.

IBM MQ trace

IBM MQ supplies a trace facility that you can use to gather the following information while the queue manager is running:

Performance statistics

The statistics trace gathers the following information to help you monitor performance and tune your system:

- Counts of different MQI requests (message manager statistics)
- Counts of different object requests (data manager statistics)
- Information about Db2 usage (Db2 manager statistics)
- Information about Coupling Facility usage (Coupling Facility manager statistics)
- Information about SMDS usage (shared message data set statistics)
- Information about buffer pool usage (buffer manager statistics)
- Information about logging (log manager statistics)
- Information about storage usage (storage manager statistics)
- Information about lock requests (lock manager statistics)

Accounting data

- The accounting trace gathers information about the processor time spent processing MQI calls and about the number of MQPUT and MQGET requests made by a particular user.
- IBM MQ can also gather information about each task using IBM MQ. This data is gathered as a thread-level accounting record. For each thread, IBM MQ also gathers information about each queue used by that thread.

The data generated by the trace is sent to the System Management Facility (SMF) or the generalized trace facility (GTF).

Events

IBM MQ events provide information about errors, warnings, and other significant occurrences in a queue manager. By incorporating these events into your own system management application, you can monitor the activities across many queue managers, for multiple IBM MQ applications. In particular, you can monitor all the queue managers in your system from a single queue manager.

Events can be reported through a user-written reporting mechanism to an administration application that supports the presentation of the events to an operator. Events also enable applications acting as agents for other administration networks, for example NetView®, to monitor reports and create the appropriate alerts.

Related tasks

Using IBM MQ trace

Using IBM MQ events

z/OS Unit of recovery disposition on z/OS

Certain transactional applications can use a GROUP, rather than a QMGR, unit of recovery disposition when connected to a queue manager in a queue sharing group (QSG) by specifying the QSG name when they connect instead of the queue manager name. This allows transaction recovery to be more flexible and robust by removing the requirement to reconnect to the same queue manager in the QSG.

Transactions started by applications that have connected using the queue sharing group name also have a GROUP unit of recovery disposition.

When a transactional application connects with a GROUP unit of recovery disposition it is logically connected to the queue sharing group and does not have an affinity to any specific queue manager. Any 2-phase commit transactions that it has started that have completed phase-1 of the commit process, that is, they are in doubt, can be inquired and resolved, when connected to any queue manager within the QSG. In a recovery scenario this means that the transaction coordinator does not have to reconnect to the same queue manager, which may be unavailable at that time.

Applications that connect with a QMGR unit of recovery disposition have a direct affinity to the queue manager to which they are connected. In a recovery scenario the transaction coordinator must reconnect to the same queue manager to resolve any in-doubt transactions, irrespective of whether the queue manager belongs to a queue sharing group.

When applications specify a queue sharing group name, and thus connect to a queue manager in a QSG with a GROUP unit of recovery disposition, the queue sharing group is logically a separate resource manager. This means that in-doubt transactions are only visible to an application if it reconnects with the same unit of recovery disposition. In-doubt transactions with a QMGR unit of recovery disposition are not visible to applications that have connected with a GROUP unit of recovery disposition and vice versa.

Related concepts

“Enabling GROUP units of recovery” on page 280
A queue sharing group can configure and enable support for GROUP units of recovery.

“Application support” on page 281
Use this page to determine which applications can connect with a GROUP unit of recovery disposition.

z/OS Enabling GROUP units of recovery

A queue sharing group can configure and enable support for GROUP units of recovery.

To use GROUP units of recovery on a queue manager within a QSG, enable the GROUPUR queue manager attribute. For more information about this concept, see “Unit of recovery disposition on z/OS” on page 280 before reading the rest of this topic.

When the GROUPUR queue manager attribute is enabled, the queue manager accepts new connections with a GROUP unit of recovery disposition. If you disable this attribute new connections with this disposition are not accepted, although applications already connected is unaffected until they disconnect.

When an application connects with a GROUP unit of recovery disposition and either inquires what transactions are in doubt or attempts to resolve a transaction that was started elsewhere in the

queue sharing group (QSG), the queue manager to which it is now connected must be able to communicate with the other members of the queue sharing group so that it can process the request. To do this it uses a shared queue called SYSTEM.QSG.UR.RESOLUTION.QUEUE. This queue must be on a recoverable application structure called CSQSYSAPPL. The structure must be recoverable because persistent messages are stored on this queue when processing resolution requests.

Before you can enable GROUP units of recovery, you must ensure that the coupling facility structure and the shared queue are defined. You can use the definitions in the CSQ4INSS sample. When the queue is defined, or detected during startup, each queue manager in the queue sharing group opens the queue so that it can receive incoming requests. If you want to delete or move the queue because it has been defined incorrectly you can request that the queue managers close their open handles on it by updating the queue object to inhibit MQGET requests. When you have made the necessary corrections, permitting applications to get messages from the queue once more directs each queue manager to reopen it. Use the DISPLAY QSTATUS command to identify what handles are open on a queue.

When you have completed this setup you can then enable GROUP units of recovery on each queue manager that you want transactional applications to be able to connect to with a GROUP unit of recovery disposition. This need not be all of the queue managers within the queue sharing group but if you choose to only enable this functionality on a subset of the queue sharing group you must ensure that your applications only attempt to connect to queue managers on which you have enabled it. For more information, see “Application support” on page 281.

When you attempt to enable the GROUPUR queue manager attribute, a number of configuration checks are performed. The queue manager checks that:

- It belongs to a queue sharing group.
- The shared-queue called SYSTEM.QSG.UR.RESOLUTION.QUEUE has been defined, according to the the definition in CSQ4INSS.
- The SYSTEM.QSG.UR.RESOLUTION.QUEUE is on a recoverable CF structure called CSQSYSAPPL.

If any of the above checks fail, the GROUPUR attribute remains disabled and a message code is returned.

These configuration checks are also performed at queue manager startup if the queue manager attribute is enabled. If any of the checks fail during startup GROUP units of recovery is disabled and the queue manager issues a message identifying which check failed. When you have performed the necessary corrective action you must then re-enable the queue manager attribute.

z/OS Application support

Use this page to determine which applications can connect with a GROUP unit of recovery disposition.

Support for the GROUP unit of recovery disposition is limited to certain types of transactional applications for which IBM MQ for z/OS is a resource manager but not the transaction coordinator. Currently supported transactional applications are:

- IBM MQ extended transactional client applications
- IBM MQ classes for JMS applications running in an application server, such as WebSphere Application Server.
- CICS applications running in CICS Transaction Server 4.2 or later, when the CICS MQCONN resource definition is configured with RESYNCMEMBER(GROUPRESYNC).

Related concepts

“ IBM MQ extended transactional client applications” on page 282
Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

“CICS applications” on page 282
Use this page to determine how CICS can use the GROUP unit of recovery disposition.

z/OS *IBM MQ extended transactional client applications*

Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

An example of an IBM MQ extended transactional client application is one that uses JMS and runs in WebSphere Application Server, connecting to IBM MQ over TCP/IP, rather than local bindings. These client applications connect to IBM MQ for z/OS over network connections, such as via TCP/IP. For these applications, it is the value specified for the QMNAME parameter of the xa_info string passed in the xa_open call that specifies whether a QMGR or GROUP unit of recovery disposition is used. For more information about xa_open, see The format of an xa_open string and Additional error processing for xa_open. For JMS applications this is done by specifying the name of the queue sharing group (QSG) in the ConnectionFactory instead of the name of a specific queue manager.

For XA client applications to take advantage of using the GROUP unit of recovery disposition you must configure your TCP/IP setup to allow your client applications to be routed to the queue managers in the queue sharing group that have the GROUPUR attribute enabled, rather than a specific queue manager. One of the dynamic virtual IP address technologies that you can use to do this is the z/OS SysPlex Distributor. See z/OS Communications Server and z/OS Conocimientos básicos: Direccionamiento virtual dinámico for more details. If you want to enable GROUP units of recovery on a subset of the queue managers in your queue sharing group, ensure that your client applications cannot be routed to those on which it is not enabled.

Your client applications do not have to connect to the queue sharing group using shared channels.

z/OS *CICS applications*

Use this page to determine how CICS can use the GROUP unit of recovery disposition.

CICS 4.2 and later provides the group resynchronization option, RESYNCMEMBER(GROUPRESYNC) in an MQCONN resource definition. A CICS configured with this option can connect to any suitable queue manager in a queue sharing group which is running on the same LPAR as that CICS region. To support the CICS GROUPRESYNC option, a queue manager must be running at MQ V7.1 or later, and be enabled for GROUPUR support.

Transactions running within a CICS region connected to MQ using GROUPRESYNC create units of work with GROUP unit of recovery disposition.

You can use RESYNCMEMBER(GROUPRESYNC) to enable faster recovery after a queue manager failure as it enables the CICS region to immediately connect to an alternative eligible queue manager running on the same LPAR, resolving any indoubt transactions as necessary, without waiting for queue manager restart.

RESYNCMEMBER(GROUPRESYNC) also enables more flexible restart options for CICS. A CICS region with its MQ connection configured to use GROUPRESYNC and MQ shared queues can be restarted on any LPAR where there is a queue manager running as a member of the same queue sharing group.

z/OS IBM MQ and other z/OS products

Use this topic to understand how IBM MQ can work with other z/OS products.

Related concepts

“IBM MQ and CICS” on page 283
All the CICS versions supported by IBM MQ 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

“IBM MQ for z/OS and WebSphere Application Server” on page 289
Use this topic to understand the use of IBM MQ for z/OS by the WebSphere Application Server.

Related reference

“IBM MQ and IMS” on page 284
Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

“IBM MQ and the z/OS Batch, TSO, and RRS adapters” on page 287

Use this topic to understand how IBM MQ works with the z/OS Batch, TSO, and RRS adapters.

z/OS IBM MQ and CICS

All the CICS versions supported by IBM MQ 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

For more information about configuring the IBM MQ CICS adapter, and the IBM MQ CICS bridge components, see the Configuring connections to IBM MQ section of the CICS documentation.

Related tasks

Using IBM MQ with CICS

z/OS CICS group attach

CICS group attach provides the ability for a CICS region to connect to any active member of an IBM MQ queue sharing group on the same LPAR rather than specifying an individual queue manager. CICS still connects to a single queue manager at a time.

You require at least two queue managers on the LPAR to support CICS group attach. Using group attach provides higher availability as you do not need a particular queue manager to be active. CICS connects to any queue manager in the queue sharing group on the LPAR.

For more information, see the CICS documentation on the MQCONN resource.

CICS attempts to connect to MQNAME passed as if it were a queue manager:

- If the queue manager exists and is active, the connection will work.
- If the connection fails, CICS queries the status of queue managers in the group to ascertain which are active on same LPAR.
- If multiple queue managers are active, CICS checks for RESYNCMEMBER(YES) and the UOW status to determine whether CICS needs to connect, or should connect, to a particular member, or wait if not active.
- If there is no need to connect to a particular member, CICS selects a queue manager (using a randomizing algorithm).
- CICS attempts to connect to chosen queue manager.
- If the attempt fails then, depending upon the return code, CICS chooses the next member, then goes through the selection loop again.
- If no queue managers are active, CICS issues multiple connections to the list of queue managers and waits on ECBLIST until the first queue manager becomes available.

Related concepts

“Group units of recovery (GROUPUR) for CICS” on page 283
The IBM MQ GROUPUR for CICS provides peer recovery for in-doubt units of work in a queue sharing group (QSG). One IBM MQ queue manager can resolve in-doubt units of work on behalf of another queue manager in the queue sharing group. This means that if CICS reconnects through group attach to a different queue manager in the QSG, it can resolve indoubt transactions from a previous IBM MQ connection.

Related information

Support for IBM MQ queue sharing groups

z/OS Group units of recovery (GROUPUR) for CICS

The IBM MQ GROUPUR for CICS provides peer recovery for in-doubt units of work in a queue sharing group (QSG). One IBM MQ queue manager can resolve in-doubt units of work on behalf of another queue manager in the queue sharing group. This means that if CICS reconnects through group attach to a different queue manager in the QSG, it can resolve indoubt transactions from a previous IBM MQ connection.

If a CICS region is working with a queue manager, and the queue manager ends abnormally, then any indoubt transactions are recovered. This eliminates the need for the CICS region to wait for the queue manager that it was working with to restart, and then resolve any in doubt units of work. This means that you need at least two queue managers on the LPAR, so that CICS can connect to another queue manager in the event of an abnormal termination of the first queue manager.

The new RESYNCMEMBER(GROUPRESYNC) setting on the CICS MQCONN definition:

- Uses the IBM MQ group attach function and peer recovery.
- Requires a queue manager with the GROUPUR attribute enabled.
- Still supports the existing CICS MQCONN RESYNCMEMBER settings (YES and NO):
  - Uses the existing CICS group attach function and no peer recovery.
  - Changing RESYNCMEMBER settings takes effect next time CICS connects to IBM MQ.

Related concepts

"Enabling GROUP units of recovery" on page 280

A queue sharing group can configure and enable support for GROUP units of recovery.

z/OS IBM MQ and IMS

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

The optional additional IBM MQ - IMS bridge enables applications to run an IMS application that does not use the MQI. This means that you can use your legacy applications with IBM MQ, without the need to rewrite them.

For more information about these components, see the following subtopics:

Related concepts

IMS and IMS bridge applications on IBM MQ for z/OS

Related tasks

Setting up the IMS adapter

Setting up the IMS bridge

Operating the IMS adapter

Related reference

MQIIH - IMS information header

z/OS The IMS adapter

The IMS adapter is an interface between IMS application programs and an IBM MQ subsystem.

The IBM MQ adapters enable different application environments to send and receive messages through a message queuing network. The IMS adapter is the interface between IMS application programs and an IBM MQ subsystem. It makes it possible for IMS application programs to use the MQI.

The IMS adapter receives and interprets requests for access to IBM MQ using the External Subsystem Attach Facility (ESAF) provided by IMS. Usually, IMS connects to IBM MQ automatically without operator intervention.

The IMS adapter provides access to IBM MQ resources for programs running in the following modes or states:

- Task (TCB) mode
- Problem state
- Non-cross-memory mode
- Non-access register mode

The adapter provides a connection thread from an application task control block (TCB) to IBM MQ.

The adapter supports a two-phase commit protocol for changes made to resources owned by IBM MQ with IMS acting as the syncpoint coordinator. Conversations where IMS is not the syncpoint coordinator, for example APPC- protected (SYNCLVL=SYNCPT) conversations, are not supported by the IMS adapter.

The adapter also provides a trigger monitor transaction (CSQQTRMN). This is described in "The IMS trigger monitor" on page 285.

You can use IBM MQ with the IMS Extended Recovery Facility (XRF) to aid recovery from a IMS error.

Note: As of IMS 15.2 Extended Recovery Facility (XRF) is no longer supported. See the IMS documentation for more information.

Using the adapter

The application programs and the IMS adapter run in the same address space. The queue manager is separate, in its own address space.

You must link-edit each program that issues one or more MQI calls to a suitable IMS language interface module, and, unless it uses dynamic MQI calls, the IBM MQ-supplied API stub program, CSQQSTUB. When the application issues an MQI call, the stub transfers control to the adapter through the IMS external subsystem interface, which manages the processing of the request by the message queue manager.

System administration and operation with IMS

An authorized IMS terminal operator can issue IMS commands to control and monitor the connection to IBM MQ. However, the IMS terminal operator has no control over the IBM MQ address space. For example, the operator cannot shut down IBM MQ from an IMS address space.

Restrictions

The following IBM MQ API calls are not supported within an application using the IMS adapter:

- MQCB
- MQCB_FUNCTION
- MQCTL

The IMS trigger monitor

The IMS trigger monitor (**CSQQTRMN**) is an IBM MQ-supplied IMS application that starts an IMS transaction when an IBM MQ event occurs, for example, when a message is put onto a specific queue.

How it works

When a message is put onto an application message queue, a trigger is generated if the trigger conditions are met. The queue manager then writes a message (containing some user-defined data), known as a *trigger message*, to the initiation queue that has been specified for that message queue. In an IMS environment, you can start an instance of CSQQTRMN to monitor an initiation queue and to retrieve the trigger messages from it as they arrive. Typically, CSQQTRMN schedules another IMS transaction by an INSERT (ISRT) to the IMS message queue. The started IMS application reads the message from the application message queue and then processes it. CSQQTRMN must run as a non-message BMP.

Each copy of CSQQTRMN services a single initiation queue. When it has started, the trigger monitor runs until IBM MQ or IMS ends.

The APPLCTN macro for CSQQTRMN must specify SCHDTYP=PARALLEL.

Because the trigger monitor is a batch-oriented BMP, IMS transactions that are started by the trigger monitor contain the following:

- Blanks in the LTERM field of the IOPCB
- The PSB name of the trigger monitor BMP in the Userid field of the IOPCB

If the target IMS transaction is protected by Security Server (previously known as RACF), you might need to define CSQQTRMN as a user ID to Security Server.

z/OS The IBM MQ - IMS bridge

The IBM MQ - IMS bridge is the component of IBM MQ for z/OS that allows direct access from IBM MQ applications to applications on your IMS system.

The IBM MQ - IMS bridge enables *implicit MQI support*. This means that you can re-engineer legacy applications that were controlled by 3270-connected terminals to be controlled by IBM MQ messages, without having to rewrite, recompile, or re-link them. The bridge is an IMS *Open Transaction Manager Access* (OTMA) client.

In bridge applications there are no IBM MQ calls within the IMS application. The application gets its input using a GET UNIQUE (GU) to the IOPCB and sends its output using an ISRT to the IOPCB. IBM MQ applications use the IMS header (the MQIIH structure) in the message data to ensure that the applications can execute as they did when driven by nonprogrammable terminals. If you are using an IMS application that processes multi-segment messages, note that all segments should be contained within one IBM MQ message.

The IMS bridge is illustrated in Figure 78 on page 286.

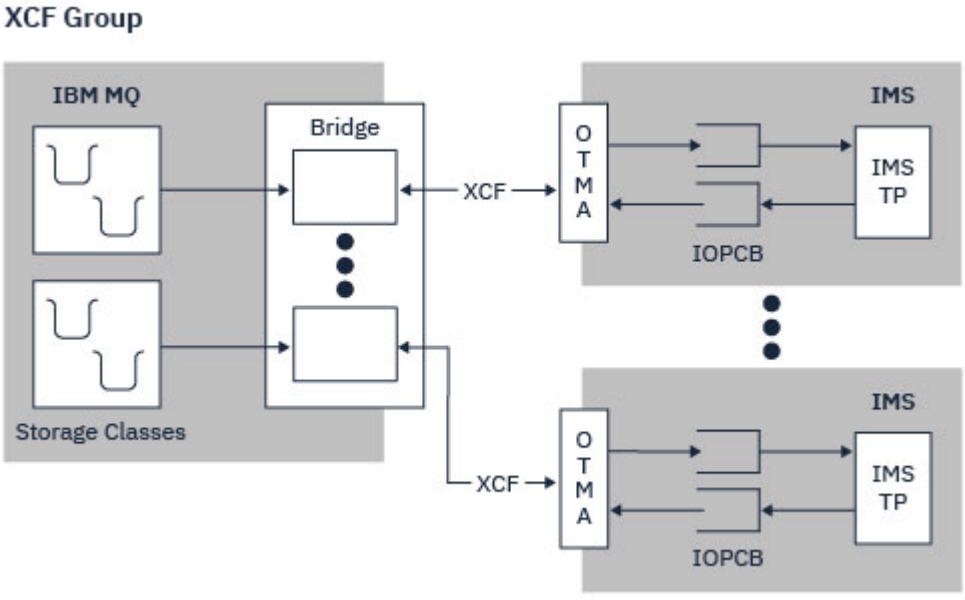


Figure 78. The IBM MQ - IMS bridge

A queue manager can connect to one or more IMS systems, and more than one queue manager can connect to one IMS system. The only restriction is that they must all belong to the same XCF group and must all be in the same sysplex.

See Setting up the IMS bridge for information on setting up an IMS bridge and adding an additional IMS connection to the same queue manager.

What is OTMA?

The IMS OTMA facility is a transaction-based connectionless client/server protocol that runs on IMS. It functions as an interface for host-based communications servers accessing IMS TM applications through the z/OS Cross Systems coupling facility (XCF).

OTMA enables clients to connect to IMS to provide high performance for interactions between clients and IMS for a large network or large number of sessions. OTMA is implemented in a z/OS sysplex environment. Therefore, the domain of OTMA is restricted to the domain of XCF.

OTMA Resource Monitoring

Support for the x'3C' OTMA protocol messages, available in IMS v10 or higher, is included in the IBM MQ - IMS bridge in IBM MQ for z/OS. These messages are sent to OTMA clients by IMS to report its health status.

If an IMS partner is unable to process the volume of transaction requests being sent then it will notify IBM MQ that a flood warning has occurred. In response IBM MQ will slow down the rate at which requests are sent over the bridge.

If IMS is still unable to process the transaction requests and a full flood condition occurs all TPIPEs to the IMS partner are suspended. Upon notification from the IMS partner that the flood or flood-warning condition has been relieved IBM MQ will resume all suspended TPIPEs, if appropriate, and gradually increase the rate at which transaction requests are sent until the maximum rate is achieved. Console messages are issued by IBM MQ in response to a change in the status of IMS partners.

If IMS v10 partners are being used you should ensure that PTF UK45082 has been applied.

Submitting IMS transactions from IBM MQ

To submit an IMS transaction that uses the bridge, applications put messages on an IBM MQ queue as usual. The messages contain IMS transaction data; they can have an IMS header (the MQIIH structure) or allow the IBM MQ - IMS bridge to make assumptions about the data in the message.

IBM MQ then puts the message to an IMS queue (it is queued in IBM MQ first to enable the use of syncpoints to assure data integrity). The storage class of the IBM MQ queue determines whether the queue is an *OTMA queue* (that is, a queue used to transmit messages to the IBM MQ - IMS bridge) and the particular IMS partner to which the message data is sent.

Remote queue managers can also start IMS transactions by writing to these OTMA queues on IBM MQ for z/OS.

Data returned from the IMS system is written directly to the IBM MQ reply-to queue specified in the message descriptor structure (MQMD). (This might be a transmission queue to the queue manager specified in the **ReplyToQMgr** field of the MQMD.)

Related concepts

IMS and IMS bridge applications on IBM MQ for z/OS

Related tasks

Customizing the IMS bridge

Related reference

“IBM MQ and IMS” on page 284

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

z/OS IBM MQ and the z/OS Batch, TSO, and RRS adapters

Use this topic to understand how IBM MQ works with the z/OS Batch, TSO, and RRS adapters.

Introduction to the Batch adapters

The Batch/TSO adapters are the interface between IBM MQ and z/OS application programs running under JES, TSO, or z/OS UNIX System Services. These adaptors enable z/OS application programs to use the MQI.

The adapters provide access to IBM MQ resources for programs running in the following modes or states:

- Task (TCB) mode
- Problem or supervisor state
- Non-cross-memory mode
- Non-access register mode

Connections between application programs and IBM MQ are at the task level. The adapters provide a connection thread from an application task control block (TCB) to IBM MQ.

The Batch/TSO adapter supports a single-phase commit protocol for changes made to resources owned by IBM MQ. It does not support multi-phase commit protocols. The RRS adapter enables IBM MQ applications to participate in two-phase commit protocols with other RRS-enabled products, coordinated by z/OS Resource Recovery Services (RRS).

The adapters use the z/OS STIMERM service to schedule an asynchronous event every second. This event runs an interrupt request block (IRB) that does not involve any waiting by the batch application's task. This IRB checks to see if the IBM MQ termination ECB has been posted. If the termination ECB has been posted, the IRB posts any application ECBs that are waiting on an event in IBM MQ (for example, a signal or a wait).

The Batch/TSO adapter

The IBM MQ Batch/TSO adapter provides IBM MQ support for z/OS Batch and TSO applications. All application programs that run under z/OS Batch or TSO must have the API stub program CSQBSTUB link-edited with them. The stub provides the application with access to all MQI calls. You use single-phase commit and backout for applications by issuing the MQI calls **MQCMIT** and **MQBACK**.

The RRS adapter

Resource Recovery Services (RRS) is a subcomponent of z/OS that provides a system-wide service for coordinating two-phase commit across z/OS products. The IBM MQ Batch/TSO RRS adapter (the RRS adapter) provides IBM MQ support for z/OS Batch and TSO applications that want to use these services. The RRS adapter enables IBM MQ to become a full participant in RRS coordination. Applications can participate in two-phase commit processing with other products that support RRS (for example, Db2).

The RRS adapter provides two stubs; you must link-edit application programs that want to use RRS with one of these stubs.

CSQBRSTB
: This stub allows you to use two-phase commit and backout for applications by using the RRS callable resource recovery services instead of the MQI calls **MQCMIT** and **MQBACK**.

  You must also link-edit module ATRSCSS from library SYS1.CSSLIB with your application. If you use the MQI calls **MQCMIT** and **MQBACK**, you will receive return code MQRC_ENVIRONMENT_ERROR.

CSQBRRSI
: This stub allows you to use MQI calls **MQCMIT** and **MQBACK** ; IBM MQ actually implements these calls as the **SRRCMIT** and **SRRBACK** RRS calls.

For information about building application programs that use the RRS adapter, see The RRS batch adapter.

Where to find more information about the z/OS Batch, TSO, and RRS adapters

You can find more information about the topics in this section in the following sources:

Table 25. Where to find more information about using z/OS Batch with IBM MQ	
Topic	**Where to look**
Setting up the Batch adapters	Task 19: Set up Batch, TSO, and RRS adapters
RRS callable resource recovery services	*MVS Programming: Callable Services for High Level Languages*

z/OS IBM MQ for z/OS and WebSphere Application Server

Use this topic to understand the use of IBM MQ for z/OS by the WebSphere Application Server.

Applications written in Java that are running under WebSphere Application Server can use the Java Message Service (JMS) specification to perform messaging. Point-to-point messaging in this environment can be provided by an IBM MQ for z/OS queue manager.

A benefit of using an IBM MQ for z/OS queue manager to provide the messaging is that connecting JMS applications can participate fully in the functionality of an IBM MQ network. For example, they can use the IMS bridge, or exchange messages with queue managers running on other platforms.

Connection between WebSphere Application Server and a queue manager

See Using IBM MQ and WebSphere Application Server together for more information.

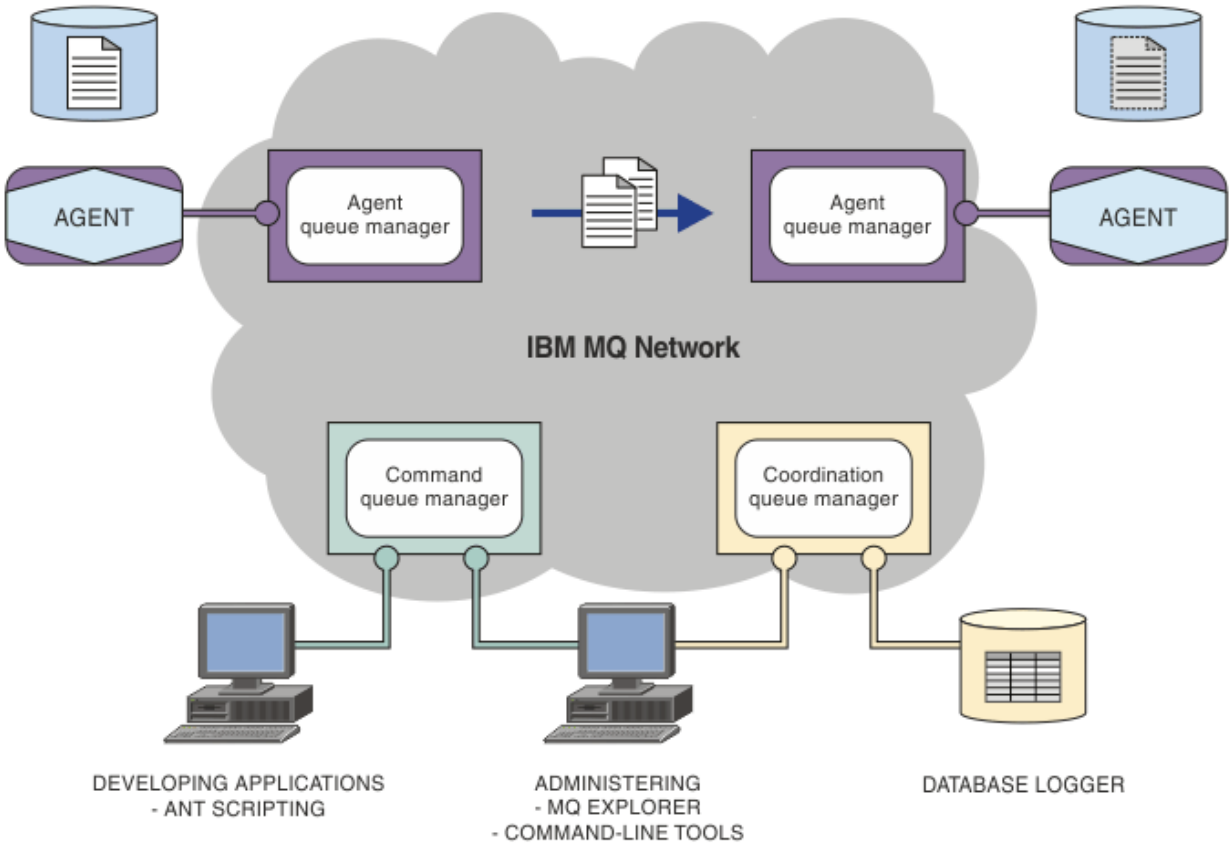
Using IBM MQ functions from JMS applications

By default, JMS messages held on IBM MQ queues use an MQRFH2 header to hold some of the JMS message header information. Many legacy IBM MQ applications cannot process messages with these headers, and require their own characteristic headers, for example the MQCIH for CICS Bridge, or MQWIH for IBM MQ Workflow applications. For more details about these special considerations, see Mapping JMS messages onto IBM MQ messages.

Managed File Transfer

Managed File Transfer transfiere archivos entre sistemas de forma gestionada y auditable, independientemente del tamaño de archivo o de los sistemas operativos utilizados.

Puede utilizar Managed File Transfer para construir una solución personalizada, escalable y automatizada que le permita gestionar, confiar en, y proteger las transferencias de archivos. Managed File Transfer elimina costosas redundancias, reduce los costes de mantenimiento y maximiza las inversiones existentes en TI.



El diagrama muestra una topología simple de Managed File Transfer. Hay dos agentes, cada uno de ellos se conecta a su propio gestor de colas de agente en una red IBM MQ. Se transfiere un archivo desde el agente en un lado del diagrama, a través de la red IBM MQ, hasta el agente en el otro lado del diagrama. También en la red IBM MQ están el gestor de colas de coordinación y un gestor de colas de mandatos. Las aplicaciones y herramientas se conectan a estos gestores de colas para configurar, administrar, realizar operaciones y registrar la actividad de Managed File Transfer en la red.IBM MQ

Managed File Transfer se puede instalar como cuatro opciones diferentes, en función del sistema operativo y de la configuración general. Estas opciones son Managed File Transfer Agent, Managed File Transfer Logger, Managed File Transfer Service y Managed File Transfer Tools. Para obtener más información, consulte Opciones del producto Managed File Transfer.

Puede utilizar Managed File Transfer para realizar las siguientes tareas:

- Crear transferencias de archivos gestionadas
  - Windows Linux Crear nuevas transferencias de archivos desde IBM MQ Explorer en plataformas Linux o Windows.
  - Crear nuevas transferencias de archivos desde la línea de mandatos en todas las plataformas soportadas.
  - Integrar la función de transferencia de archivos a la herramienta Apache Ant.
  - Escribir aplicaciones que controlen Managed File Transfer colocando mensajes en colas de mandatos de agente.
  - Planificar transferencias de archivos para que se realicen con posterioridad. Además, puede desencadenar transferencias de archivos planificadas basándose en una serie de sucesos de sistema de archivos, por ejemplo un nuevo archivo que se crea.

  - Supervisar continuamente un recurso, por ejemplo un directorio, y cuando el contenido de dicho recurso cumpla una determinada condición predefinida, iniciar una tarea. Esta tarea puede ser una transferencia de archivos, un script Ant o un trabajo JCL.
  - Transferir archivos a y de las colas de IBM MQ
  - Transferir archivos a y desde servidores FTP, FTPS o SFTP.
  - Transferir archivos a y desde nodos Connect:Direct.
  - Transferir archivos de texto y binarios. Los archivos de texto se convierten automáticamente entre las páginas de códigos y las convenciones de fin de línea de los sistemas de origen y destino.
  - Las transferencias se pueden proteger, mediante los estándares del sector para conexiones basadas en SSL (Socket Layer Layer).
- Ver las transferencias en curso y registrar información sobre todas las transferencias de la red
  - 

Ver el estado de la transferencias en cursos desde IBM MQ Explorer en plataformas Linux o Windows.
  - 

Comprobar el estado de las transferencias que se han completado utilizando IBM MQ Explorer en plataformas Linux o Windows.
  - Utilizar la característica de registrador de base de datos de Managed File Transfer para guardar los mensajes de registro en una base de datos Db2 u Oracle.

Managed File Transfer se basa en IBM MQ, que proporciona una entrega segura y única de mensajes entre aplicaciones. Puede aprovechar las diversas características de IBM MQ. Por ejemplo, puede utilizar la compresión de canales para comprimir los datos que envíe entre agentes a través de canales de IBM MQ y utilizar canales SSL para proteger los datos que envíe entre agentes. Los archivos se transfieren de forma fiable y pueden tolerar que falle la infraestructura a través de la cual se lleva a cabo la transferencia de archivos. Si experimenta una interrupción de la red, la transferencia de archivos se reinicia desde el punto en que se quedó cuando se restaura la conectividad.

Gracias a la consolidación de la transferencia de archivos con la red IBM MQ existente, puede evitar el gasto de los recursos necesarios de mantenimiento de dos infraestructuras distintas. Si todavía no es un cliente de IBM MQ, al crear una red IBM MQ de soporte de Managed File Transfer, está creando la red troncal para una futura implementación SOA. Si ya es cliente de IBM MQ, Managed File Transfer puede aprovechar la infraestructura existente de IBM MQ, incluidos IBM MQ Internet Pass-Thru e IBM Integration Bus.

Puede aprovechar las soluciones de alta disponibilidad de IBM MQ para mejorar la resiliencia de la configuración de Managed File Transfer . Si los agentes utilizan gestores de colas de datos replicados (RDQM), debe configurarlos para utilizar la característica de dirección IP flotante. Esto significa que los agentes utilizan la misma dirección IP para comunicarse con cualquiera de las tres instancias RDQM que se están ejecutando actualmente y se reconectan automáticamente en la migración tras error (consulte Alta disponibilidad RDQM y Creación y supresión de una dirección IP flotante). Si utiliza la solución de gestor de colas de varias instancias, las aplicaciones utilizan una dirección IP diferente para comunicarse con cada instancia, que gestiona la reconexión de cliente en la migración tras error (consulte Gestores de colas de varias instancias y Reconexión de canal y cliente).

Managed File Transfer se integra con una serie de otros productos IBM:

IBM Integration Bus

Procese archivos que han sido transferidos por Managed File Transfer como parte de un flujo de IBM Integration Bus. Para obtener más información, consulte Trabajar con MFT desde IBM Integration Bus.

IBM Sterling Connect:Direct

Transfiera archivos a y desde una red Connect:Direct existente utilizando el puente Managed File TransferConnect:Direct. Para obtener más información, consulte El puente Connect:Direct bridge.

IBM Tivoli Composite Application Manager

IBM Tivoli Composite Application Manager proporciona un agente que puede utilizar para supervisar la información que se publica en el gestor de colas de coordinación.

Conceptos relacionados

Opciones de producto de Managed File Transfer

"Visión general de la topología de MFT" en la página 292

Una visión general de cómo se conectan los agentes de Managed File Transfer con el gestor de colas de coordinación en una red IBM MQ.

"¿Cómo funciona MFT con IBM MQ?" en la página 292

Managed File Transfer interactúa de varias maneras con IBM MQ.

¿Cómo funciona MFT con IBM MQ?

Managed File Transfer interactúa de varias maneras con IBM MQ.

- Managed File Transfer transfiere archivos entre procesos de agente dividiendo cada archivo en uno o varios mensajes y transmitiendo los mensajes a través de la red IBM MQ.
- El agente procesa los datos de archivo utilizando mensajes no persistentes para minimizar el impacto en los registros de IBM MQ. La intercomunicación entre agentes hace que los procesos de agente regulen el flujo de los mensajes que contienen datos de archivo. Esto impide que los mensajes que contienen datos de archivo se acumulen en las colas de transmisión de IBM MQ y garantiza que si alguno de los mensajes no persistentes no se entrega, los datos de archivo se envíen de nuevo.
- Los agentes de Managed File Transfer utilizan un número de colas de IBM MQ.. Para obtener más información, consulte Colas del sistema MFT y el tema del sistema.
- Si bien algunas de estas colas son estrictamente para uso interno, un agente puede aceptar solicitudes en forma de mensajes de mandato con formato especial que se envían a una cola específica que el agente lee. Tanto los mandatos de línea de mandatos como el conector de IBM MQ Explorer envían mensajes de IBM MQ al agente para indicarle que realice la acción deseada. Puede escribir aplicaciones de IBM MQ que interactúen con el agente de este modo. Para obtener más información, consulte Control de MFT colocando mensajes en la cola de mandatos del agente .
- Los agentes de Managed File Transfer envían información sobre su estado, progreso y resultado de las transferencias a un gestor de colas de MQ designado como gestor de colas de coordinación. El gestor de colas de coordinación publica esta información a la que se pueden suscribir las aplicaciones que desean supervisar el progreso de las transferencias o mantener un registro de las transferencias que se han producido. Tanto los mandatos de línea de mandatos como el conector de IBM MQ Explorer pueden utilizar la información que se ha publicado. Puede escribir aplicaciones de IBM MQ que utilicen esta información. Para obtener más información sobre el tema en el que se publica la información, consulte SYSTEM.FTE TemaFTE.
- Los componentes clave de Managed File Transfer aprovechan la capacidad de los gestores de colas de IBM MQ para almacenar y reenviar mensajes. Esto significa que si sufre una interrupción, las partes de la infraestructura que no se ven afectadas pueden continuar transfiriendo archivos. Esto se aplica por extensión al gestor de colas de coordinación, en el que una combinación de suscripciones duraderas de almacenamiento y envío permite que el gestor de colas de coordinación acepte dejar de estar disponible sin perder información clave sobre las transferencias de archivos que se han realizado.

Visión general de la topología de MFT

Una visión general de cómo se conectan los agentes de Managed File Transfer con el gestor de colas de coordinación en una red IBM MQ.

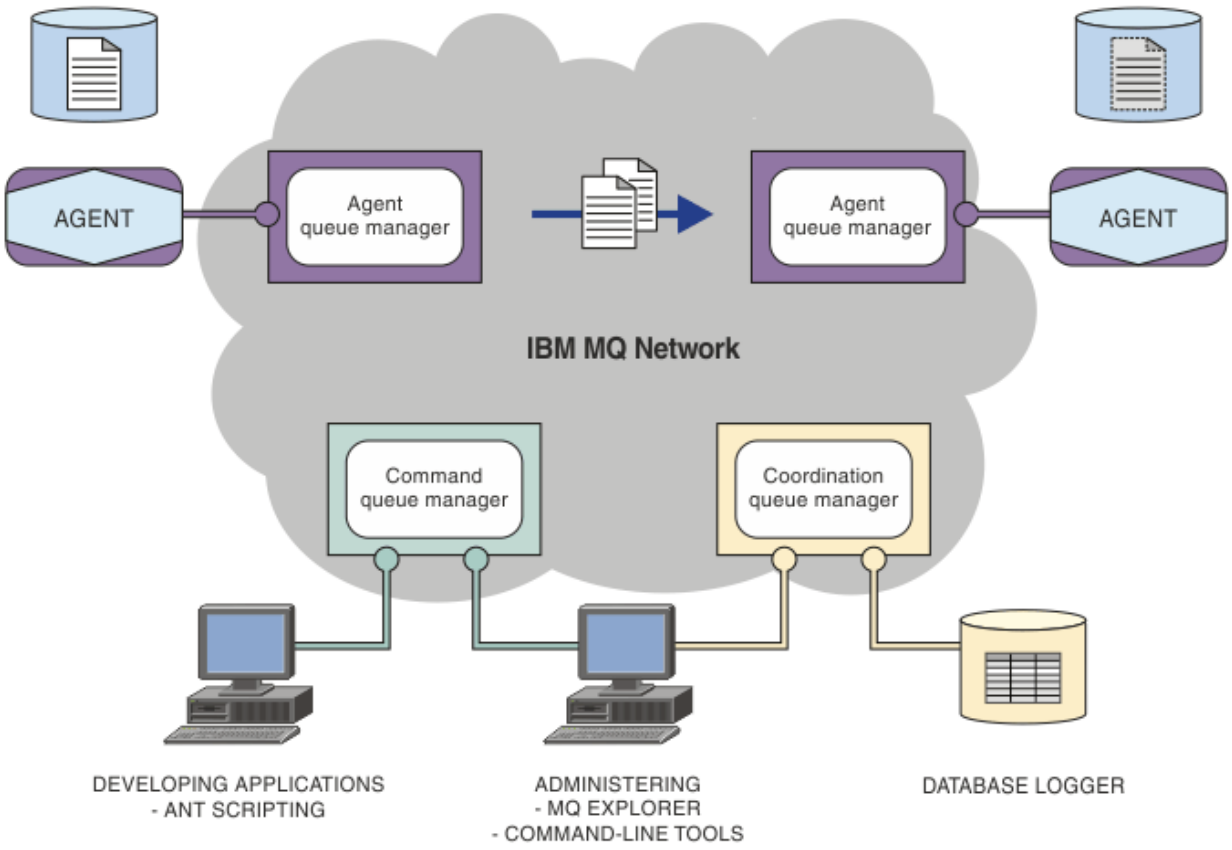
Los agentes de Managed File Transfer envían y reciben los archivos que se transfieren. Cada agente tiene su propio conjunto de colas en el gestor de colas asociado y el agente está vinculado al gestor de colas en modalidad de enlaces o de cliente. Un agente también puede utilizar el gestor de colas de coordinación como gestor de colas.

El gestor de colas de coordinación difunde información de auditoría y de archivos. El gestor de colas de coordinación representa un único punto para la recopilación de información de agente, de estado de transferencias y de auditoría de transferencias. No es necesario que el gestor de colas de coordinación esté disponible para que se produzcan transferencias. Si el gestor de colas de coordinación queda no disponible temporalmente, las transferencias continúan con toda normalidad. Los mensajes de auditoría y de estado se almacenan en los gestores de colas de agente hasta que el gestor de colas de coordinación queda disponible, y luego se pueden procesar con normalidad.

Los agentes se registran con el gestor de colas de coordinación y publicar los detalles en dicho gestor de colas. Esta información de agente la utiliza el conector de Managed File Transfer para habilitar el inicio de transferencias desde IBM MQ Explorer. La información de agente recopilada en el gestor de colas de coordinación también es utilizada por los mandatos para mostrar información sobre agentes y estado de agentes.

La información de estado de transferencias y de auditoría de transferencias se publica en el gestor de cola de coordinación. El estado de la transferencia y la información de auditoría de transferencia de Managed File Transfer utiliza el estado de transferencia y la información de auditoría de transferencia para supervisar el progreso de las transferencias desde IBM MQ Explorer. La información de auditoría de transferencias almacenada en el gestor de colas de coordinación se puede conservar para facilitar la auditabilidad.

El gestor de colas de mandatos se utiliza para conectarse a la red IBM MQ y es el gestor de colas al que se ha conectado al emitir mandatos de Managed File Transfer.



Conceptos relacionados

“Managed File Transfer” en la página 289
Managed File Transfer transfiere archivos entre sistemas de forma gestionada y auditable, independientemente del tamaño de archivo o de los sistemas operativos utilizados.

“¿Cómo funciona MFT con IBM MQ?” en la página 292
Managed File Transfer interactúa de varias maneras con IBM MQ.

Escenario de Managed File Transfer

Descripción general de MFT REST API

La REST API da soporte a determinados mandatos de Managed File Transfer, incluidos el listado de transferencias y detalles sobre los agentes de transferencia de archivos.

La REST API incluye opciones para listar todas las transferencias de Managed File Transfer y consultar el estado de los agentes de Managed File Transfer. Para obtener más información, consulte Iniciación a la REST API MFT.

IBM MQ Internet Pass-Thru

IBM MQ Internet Pass-Thru (MQIPT) es un componente opcional de IBM MQ que se puede utilizar para implementar soluciones de mensajería entre sitios remotos a través de Internet.

Para obtener los archivos de instalación de MQIPT para IBM MQ 9.4.x, vaya a IBM Fix Central para IBM MQ.

Puede utilizar MQIPT para conectar cualquier versión soportada de IBM MQ. No tiene que instalar ningún otro componente de IBM MQ en la misma versión que MQIPT.

Si ha adquirido la titularidad para IBM MQ, puede instalar tantas copias de MQIPT como sea necesario. Las instancias de MQIPT no se contarán en relación con la titularidad de IBM MQ que ha adquirido. Para obtener más información sobre las licencias de IBM MQ , consulte Información de licencia deIBM MQ.

Nota: Esta documentación está relacionada con MQIPT en IBM MQ 9.4. Para la documentación del paquete de soporte de MQIPT (versión 2.1) en IBM Documentation, consulte MQIPT (SupportPac MS81) en la documentación de IBM MQ 9.0.

Nota: Si utiliza MQIPT 2.1 o anterior, se recomienda actualizar a MQIPT para IBM MQ 9.4, ya que la fecha de fin de soporte para el paquete de soporte de MQIPT era el 30th de septiembre de 2020.

IBM MQ Internet Pass-Thru se ejecuta como un servicio autónomo que puede recibir y reenviar flujos de mensajes de IBM MQ, ya sea entre dos gestores de colas de IBM MQ o entre un cliente de IBM MQ y un gestor de colas de IBM MQ.

MQIPT permite esta conexión cuando el cliente y el servidor no están en la misma red física.

Una o más instancias de MQIPT se pueden colocar en la vía de comunicación entre dos gestores de colas de IBM MQ o entre un cliente de IBM MQ y un gestor de colas de IBM MQ. Las instancias de MQIPT permiten que los dos sistemas IBM MQ intercambien mensajes sin necesidad de una conexión TCP/IP directa entre los dos sistemas. Esta arquitectura es útil si la configuración del cortafuegos prohíbe una conexión TCP/IP directa entre los dos sistemas.

MQIPT escucha en uno o más puertos TCP/IP las conexiones entrantes. Estas conexiones pueden transportar mensajes IBM MQ normales, mensajes IBM MQ túneles dentro de HTTP o mensajes cifrados utilizando TLS (Transport Layer Security) o SSL (Secure Sockets Layer). MQIPT puede manejar varias conexiones simultáneas.

El canal IBM MQ que realiza la solicitud de conexión TCP/IP inicial recibe el nombre de *linterlocutor*, el canal al que se intenta conectarse es el *respondedor* y el gestor de colas al que finalmente intenta contactar es el *gestor de colas de destino*.

MQIPT conserva los datos en la memoria a medida que los envía desde su origen a su destino. No se salvan datos en el disco (excepto para la memoria que el sistema operativo transfiera al disco). La única vez que MQIPT accede al disco de forma explícita es para leer su archivo de configuración y para escribir registros de anotaciones cronológicas y de rastreo de conexión.

El rango completo de tipos de canal IBM MQ se puede conectar a través de una o más instancias de MQIPT. La presencia de MQIPT en una vía de acceso de comunicación no tiene ningún efecto en las características funcionales de los componentes de IBM MQ conectados. Sin embargo, el rendimiento de la transferencia de mensajes puede verse afectado.

MQIPT se puede utilizar con IBM MQ tal como se describe en “Configuraciones posibles de MQIPT” en la página 298.

Para instalar MQIPT, consulte Instalación de MQIPT.

Tareas relacionadas

Configuración de IBM MQ Internet Pass-Thru

Administración y configuración de IBM MQ Internet Pass-Thru

Referencia relacionada

Referencia de configuración de IBM MQ Internet Pass-Thru

usos de MQIPT

Hay una serie de usos potenciales para IBM MQ Internet Pass-Thru (MQIPT).

MQIPT se puede utilizar como un concentrador de canales

Si se utiliza MQIPT de este modo, los canales hacia o desde varios hosts separados pueden mostrarse para un cortafuegos como si todos fueran hacia o desde el host MQIPT. Esto hace que resulte más fácil definir y gestionar las reglas de filtro del cortafuegos.

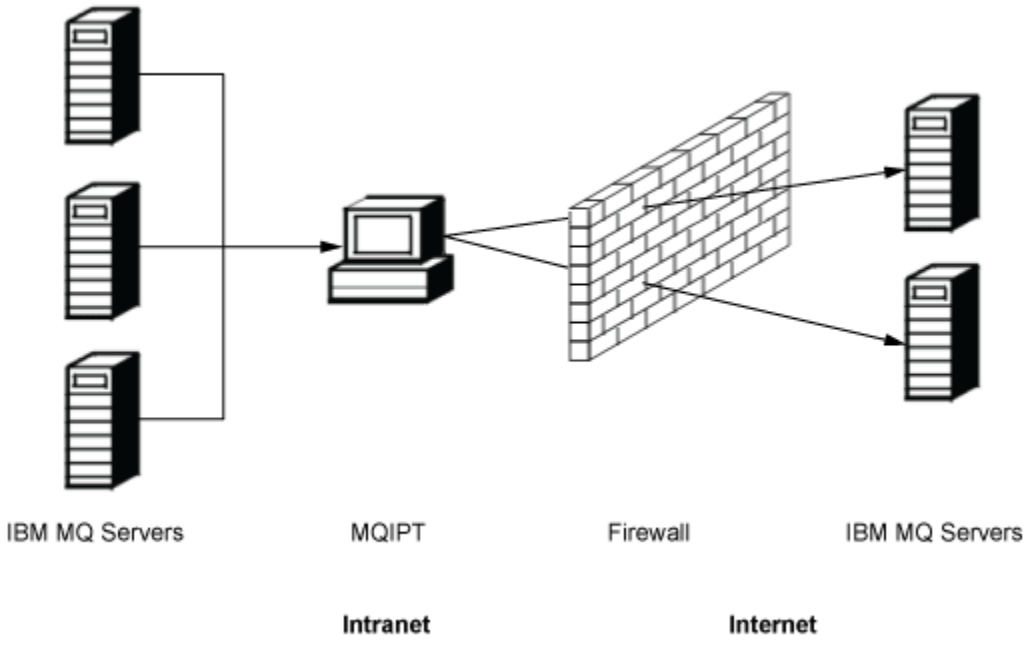


Figura 79. Ejemplo de MQIPT como un concentrador de canales.

MQIPT se puede colocar en un DMZ para proporcionar un único punto de acceso

Si MQIPT se coloca dentro de un cortafuegos DMZ (una configuración de cortafuegos para proteger redes de área local), en un sistema con una dirección de protocolo de Internet (IP) conocida y de confianza, se puede utilizar MQIPT para escuchar las conexiones de canal de IBM MQ entrantes que a continuación, puede reenviar a la intranet de confianza; el cortafuegos interno debe permitir que este sistema fiable realice conexiones de entrada. En esta configuración, MQIPT impide que las solicitudes de acceso externas reciban las direcciones IP verdaderas de los sistemas en la intranet de confianza. De este modo, MQIPT proporciona un único punto de acceso. Si es necesario, MQIPT se puede configurar para aceptar conexiones TLS y reenviar datos al destino utilizando una conexión TLS separada, por lo tanto, terminando la sesión TLS en la DMZ.

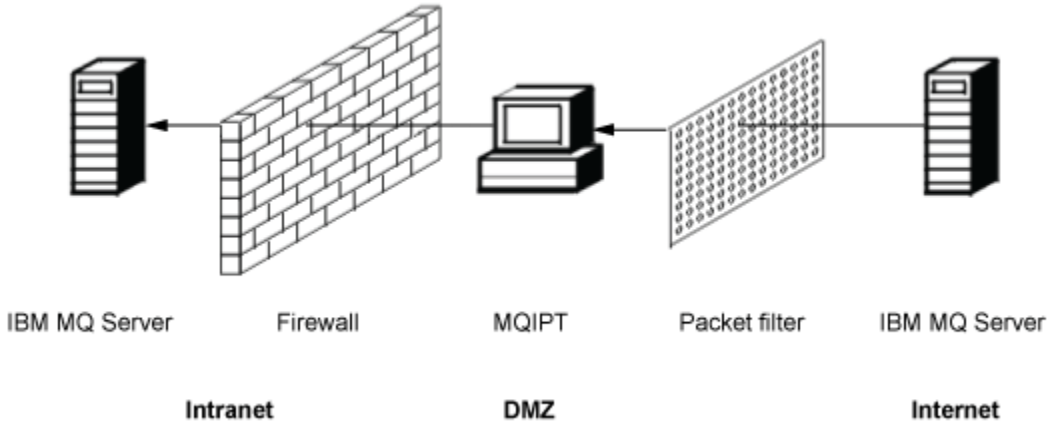


Figura 80. Ejemplo de MQIPT en un cortafuegos DMZ

MQIPT se puede comunicar mediante el túnel HTTP

Si se despliegan dos instancias de MQIPT en línea, se pueden comunicar utilizando HTTP. La función de túnel HTTP permite transmitir las solicitudes a través de cortafuegos, utilizando los proxys HTTP existentes. El primer MQIPT inserta el protocolo IBM MQ en HTTP y el segundo extrae el protocolo IBM MQ de su derivador HTTP y lo reenvía al gestor de colas de destino.

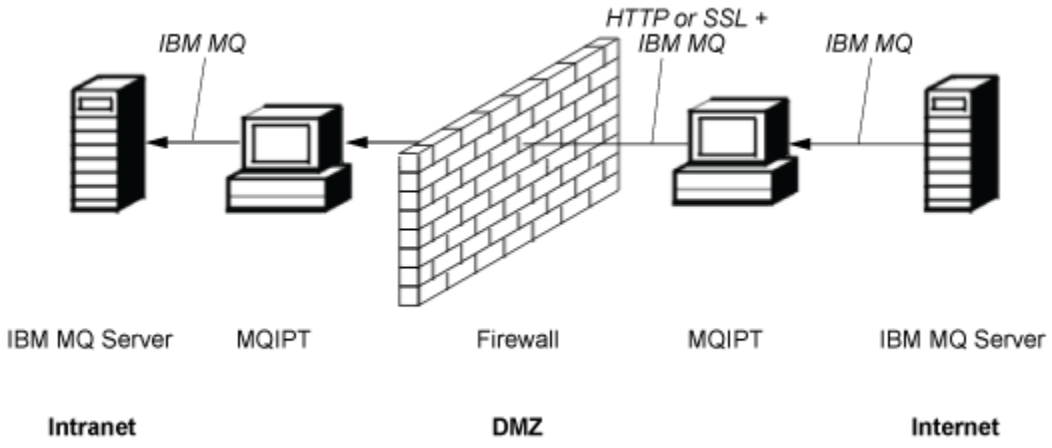


Figura 81. Ejemplo de MQIPT y túnel HTTP

MQIPT puede cifrar mensajes

Si MQIPT se configura como en el ejemplo anterior, se pueden cifrar las solicitudes antes de la transmisión a través de cortafuegos. El primer MQIPT cifra los datos y el segundo los descifra utilizando SSL/TLS antes de enviarlos al gestor de colas de destino.

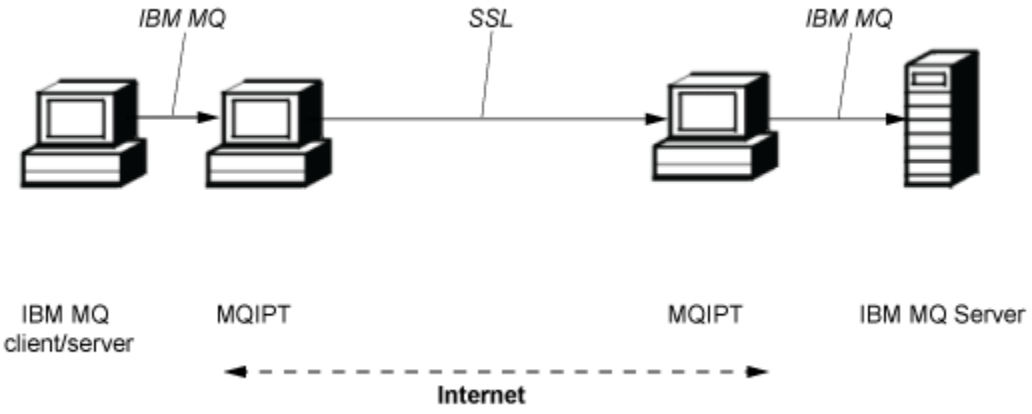


Figura 82. Ejemplo de MQIPT y SSL/TLS

Cómo funciona MQIPT

En su configuración más sencilla, MQIPT actúa como un reenviador de protocolo de IBM MQ. Permanece a la escucha en un puerto TCP/IP y acepta solicitudes de conexión de los canales de IBM MQ.

Si se recibe una solicitud con el formato correcto, MQIPT establece una conexión TCP/IP adicional entre él y el gestor de colas de IBM MQ de destino. A continuación, pasa todos los paquetes de protocolo que recibe desde su conexión de entrada al gestor de colas de destino, y devuelve los paquetes de protocolos desde el gestor de colas de destino a la conexión de entrada original.

No hay ningún cambio en el protocolo de IBM MQ (cliente/servidor o gestor de colas para el gestor de colas) porque ninguno de los dos extremos conoce directamente la presencia del intermediario. No se necesitan nuevas versiones del cliente o código de servidor de IBM MQ.

Para utilizar MQIPT, el canal emisor debe configurarse para que utilice el nombre de host y puerto de MQIPT y no el nombre de host y puerto del gestor de colas de destino. Se define con la propiedad **CONNAME** del canal IBM MQ. MQIPT lee los datos de entrada y simplemente los pasa al gestor de colas de destino. Otros campos de configuración, tales como el ID de usuario y la contraseña de un canal de cliente/servidor, se pasan del mismo modo al gestor de colas de destino.

Varios gestores de colas

Se puede utilizar MQIPT para permitir el acceso a más de un gestor de colas de destino. Para que esto funcione, debe haber un mecanismo que indique a MQIPT con qué gestor de colas se ha de conectar, por lo que MQIPT utiliza el número de puerto TCP/IP para determinar a qué gestor de colas se ha conectar.

Por lo tanto, puede configurar MQIPT para que escuche en varios puertos TCP/IP. Cada puerto de escucha se correlaciona con un gestor de colas de destino a través de una *ruta*de MQIPT . Puede definir un máximo de 100 rutas de este tipo, que asocian un puerto TCP/IP de escucha con el nombre de host y el puerto del gestor de colas de destino. Esto significa que el nombre de host (dirección IP) del gestor de colas de destino nunca está visible para el canal de origen. Cada ruta puede manejar varias conexiones entre su puerto de escucha y el destino, y cada conexión actúa de forma independiente.

Archivo de configuración de MQIPT

MQIPT utiliza un archivo de configuración denominado `mqipt.conf`. Este archivo contiene las definiciones de todas las rutas y sus propiedades asociadas. Consulte Administración y configuración de IBM MQ Internet Pass-Thru para obtener más información sobre `mqipt.conf`.

Cuando se inicia MQIPT, inicia cada una de las rutas que se listan en el archivo de configuración. Los mensajes se graban en la consola del sistema y muestran el estado de cada ruta. Cuando se muestra

el mensaje MQCPI078 para una ruta, significa que esta ruta está preparada para aceptar solicitudes de conexión.

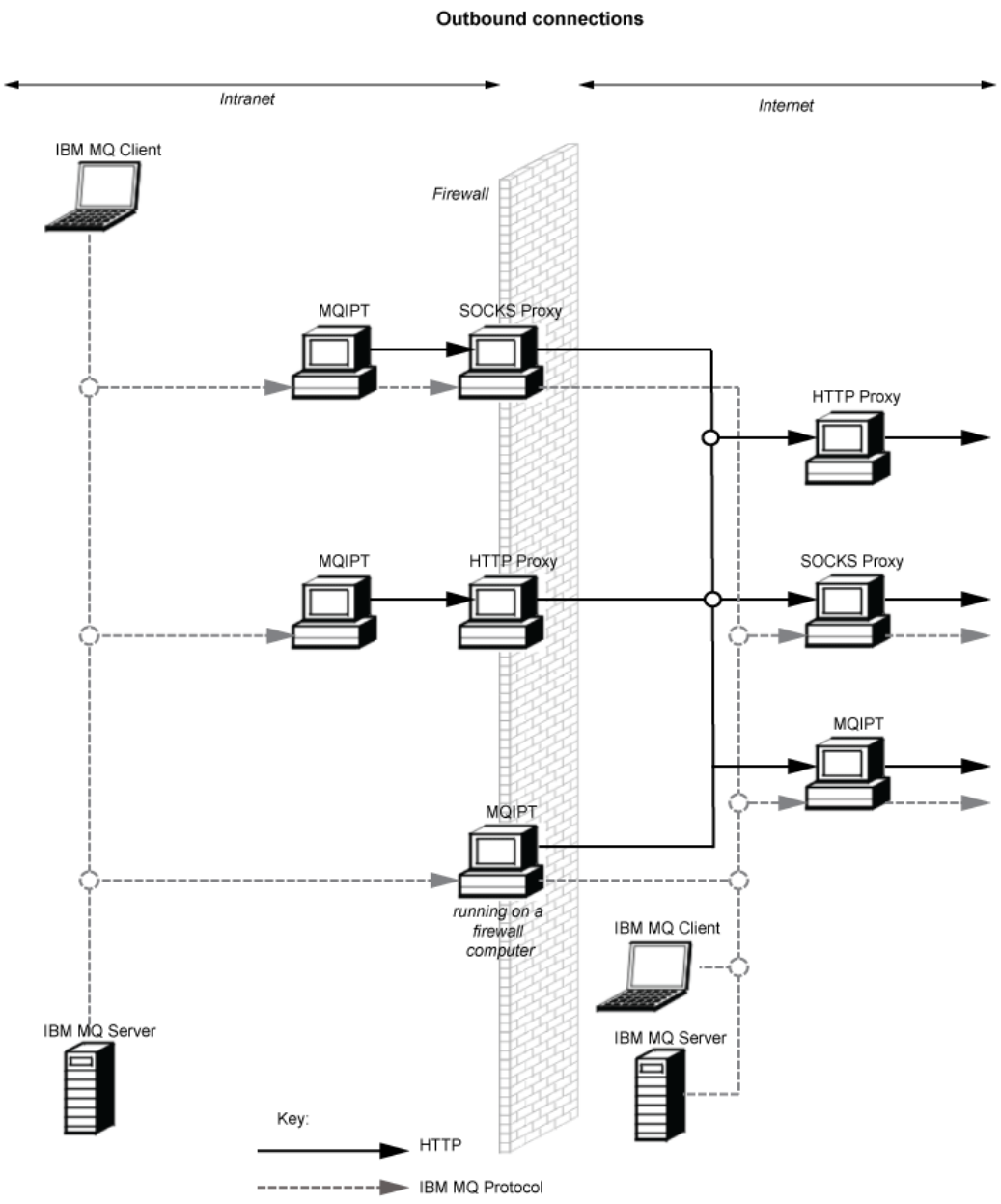
Configuraciones posibles de MQIPT

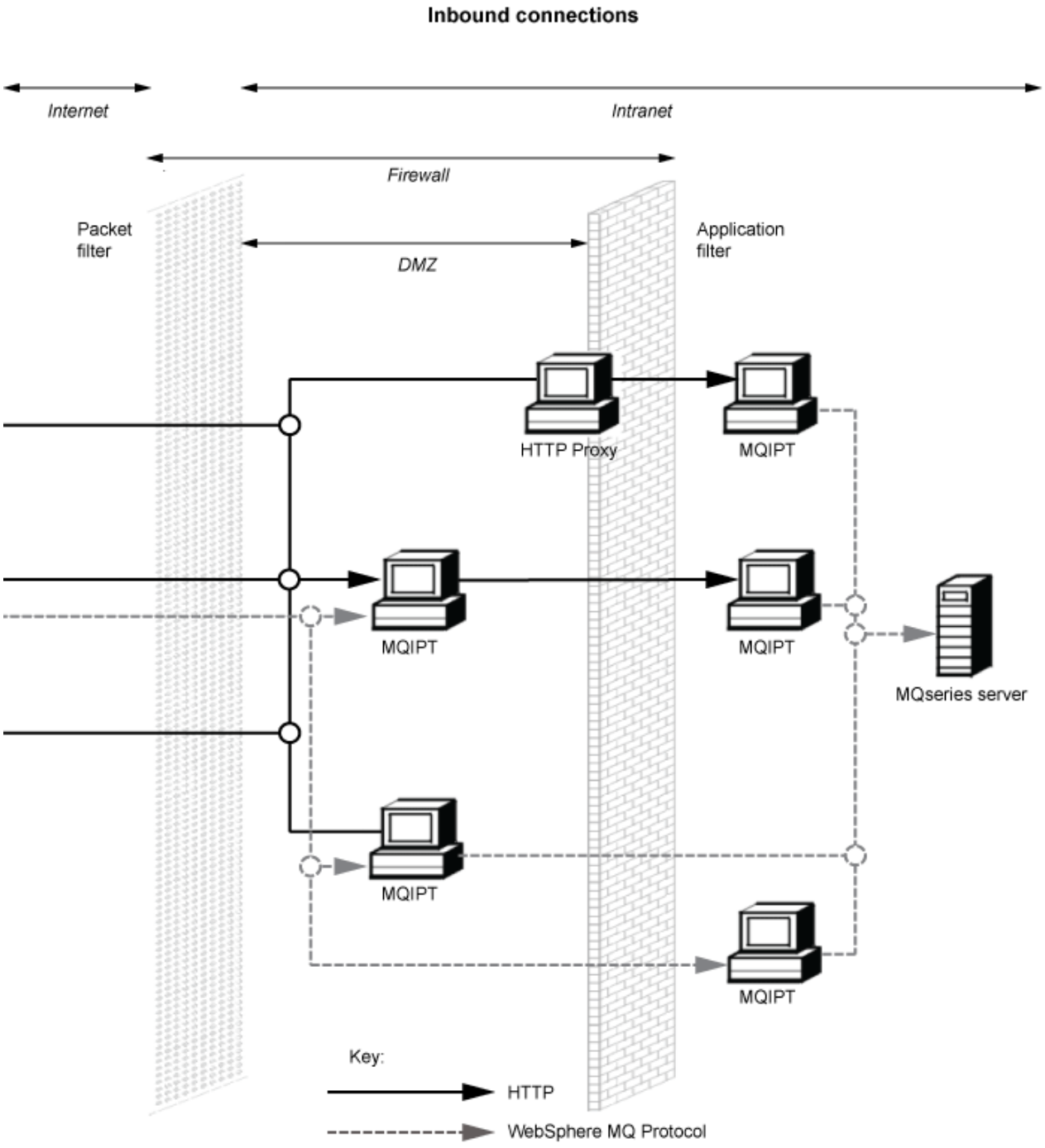
MQIPT se puede utilizar junto con IBM MQ y IBM Integration Bus.

En la siguiente figura de varias partes se muestran muchas de las posibles configuraciones para MQIPT en una topología de IBM MQ. Ilustra los diferentes modos en que MQIPT puede enviar mensajes. Muestra clientes y servidores en una intranet, en un cortafuegos y en internet fuera del cortafuegos, pasando mensajes a MQIPT, el proxy HTTP o el proxy SOCKS, que los reenvía.

Los mensajes los recibe un proxy MQIPT o un proxy HTTP en un DMZ antes de pasar el mensaje a través del cortafuegos de entrada a un servidor.

Tenga en cuenta que el proxy HTTP, el proxy SOCKS y los sistemas MQIPT del extremo de la intranet del cortafuegos representan la posibilidad de varios sistemas encadenados conjuntamente en internet. Por ejemplo, un sistema MQIPT se puede comunicar a través de uno o varios sistemas proxy SOCKS o HTTP, o sistemas MQIPT adicionales, antes de llegar a su destino.


Outbound connections
Intranet
Internet
IBM MQ Client
Firewall
MQIPT
SOCKS Proxy
HTTP Proxy
MQIPT
HTTP Proxy
SOCKS Proxy
MQIPT
MQIPT
running on a firewall computer
IBM MQ Client
IBM MQ Server
IBM MQ Server
Key:
HTTP
IBM MQ Protocol



Configuraciones compatibles

Escenarios de conexión compatibles en los que un cliente o gestor de colas de IBM MQ se comunica con MQIPT. Se utiliza la misma ruta de MQIPT, o una segunda ruta, para la comunicación con el gestor de colas de destino.

Configuraciones compatibles con una única ruta de MQIPT

Puede utilizar una única ruta de MQIPT para comunicarse con IBM MQ.

Las columnas de la Tabla 26 en la página 301 contienen la información siguiente:

1. El protocolo utilizado entre IBM MQ y la ruta MQIPT. La conexión puede ser creada por un cliente de IBM MQ o un gestor de colas, y puede utilizar IBM MQ Formatos y Protocolos (FAP) o un protocolo SSL/TLS.
2. La modalidad en que opera la ruta de MQIPT. El formato de la comunicación a través de Internet entre MQIPT y IBM MQ, viene determinado por la configuración de la ruta MQIPT. Tenga en cuenta que donde la tabla menciona SSL, también puede utilizar TLS.
3. El protocolo utilizado entre la ruta de MQIPT y el gestor de colas de destino.

Tabla 26. Configuraciones válidas con una única instancia de MQIPT

1. IBM MQ Protocolo de origen	**2. Modalidad de la ruta de MQIPT**	**3. IBM MQ Protocolo de destino**
FAP	FAP-proxy (valor predeterminado)	FAP
	FAP-server y SSL-client	SSL/TLS
SSL/TLS	SSL-proxy	SSL/TLS
	SSL-server y FAP-client	FAP
	SSL-server and SSL-client	SSL/TLS

Configuraciones compatibles con más de una ruta de MQIPT

Puede optar por utilizar más de una ruta, en una o varias instancias de MQIPT, para comunicarse con IBM MQ.

Las columnas de la Tabla 27 en la página 302 contienen la información siguiente:

1. El protocolo utilizado entre IBM MQ y la primera ruta de MQIPT. La conexión puede ser creada por un cliente de IBM MQ o un gestor de colas, y puede utilizar IBM MQ Formatos y Protocolos (FAP) o un protocolo SSL/TLS.
2. La modalidad en que opera la primera ruta de MQIPT. El formato de la comunicación a través de Internet entre MQIPT y IBM MQ, viene determinado por la configuración de la ruta MQIPT. Tenga en cuenta que donde la tabla menciona SSL, también puede utilizar TLS.
3. La modalidad en que opera la segunda ruta de MQIPT.
4. El protocolo utilizado entre la segunda ruta de MQIPT y el gestor de colas de destino.

Tabla 27. Configuraciones válidas con varias instancias de MQIPT

1. IBM MQ Protocolo de origen	2. Modalidad de la primera ruta de MQIPT	3. Modalidad de la segunda ruta de MQIPT	4. IBM MQ Protocolo de destino
FAP (valor predeterminado)	FAP-proxy (valor predeterminado)	FAP-proxy (valor predeterminado)	FAP
	FAP-server y SSL-client	SSL-proxy	SSL/TLS
		SSL-server y FAP-client	FAP
		SSL-server and SSL-client	SSL/TLS
	HTTP-client	HTTP-server y SSL-client	SSL/TLS
	HTTPS-client	HTTPS-server y SSL-client	SSL/TLS
	HTTP-client	HTTP-server	FAP
	HTTPS-client	HTTPS-server	FAP
SSL/TLS	SSL-proxy	SSL-proxy	SSL/TLS
		SSL-server y FAP-client	FAP
		SSL-server and SSL-client	SSL/TLS
	HTTP-client	HTTP-server	FAP
	HTTPS-client	HTTPS-server	SSL/TLS
	HTTP-client	HTTP-server y SSL-client	FAP
	HTTPS-client	HTTPS-server y SSL-client	SSL/TLS

Configuraciones de canales soportadas

Se da soporte a todos los tipos de canal IBM MQ, pero la configuración está restringida a las conexiones TCP/IP. En un cliente o gestor de colas de IBM MQ, MQIPT aparece como si fuera el gestor de colas de destino. Donde la configuración del canal requiere un host de destino y un número de puerto, se especifican el nombre de host de MQIPT y el número de puerto de escucha.

Canales de cliente/servidor

MQIPT permanece a la escucha de las solicitudes de conexión de cliente entrantes y, a continuación, las reenvía utilizando el túnel HTTP, SSL/TLS o como paquetes de protocolo estándar de IBM MQ. Si MQIPT utiliza el túnel HTTP o SSL/TLS, las envía en una conexión a un segundo MQIPT. Si no está utilizando el túnel HTTP, las envía en una conexión a lo que ve como el gestor de colas de destino (aunque esto pueda ser a su vez un MQIPT adicional). Cuando el gestor de colas de destino ha aceptado la conexión del cliente, transmite los paquetes entre el cliente y el servidor.

Canales de emisor/receptor del clúster

Si MQIPT recibe una solicitud de entrada desde un canal emisor del clúster, presupone que el gestor de colas está habilitado para SOCKS y la dirección de destino verdadera se obtiene durante el proceso de reconocimiento de SOCKS. Envía la solicitud al siguiente MQIPT o gestor de colas de destino, exactamente del mismo modo que para los canales de conexión de cliente. Esto también incluye los canales de emisor de clúster definidos automáticamente.

Emisor/receptor
Si MQIPT recibe una solicitud de entrada desde un canal emisor, la envía al siguiente MQIPT o gestor de colas de destino, exactamente del mismo modo que para los canales de conexión de cliente. El gestor de colas de destino valida la solicitud de entrada e inicia el canal receptor, si resulta adecuado. Todas las comunicaciones entre el canal emisor y receptor (incluidos los flujos de seguridad) se transmiten en relé.

Solicitante/servidor
Esta combinación se maneja de la misma manera que las configuraciones anteriores. La validación de la solicitud de conexión la realiza el canal del servidor en el gestor de colas de destino.

Solicitante/emisor
La configuración de "devolución de llamada" puede resultar útil si los dos gestores de colas no pueden establecer conexiones directas entre sí, pero ambos pueden conectar con MQIPT y aceptar conexiones desde el mismo.

Servidor/solicitante y servidor/receptor
MQIPT los maneja de la misma forma que maneja la configuración de `Sender/Receiver` .

Terminación del canal y condiciones de anomalía

Cuando MQIPT detecta el cierre (normal o anormal) de un canal de IBM MQ, propaga su cierre. Si cierra una ruta utilizando MQIPT, se cierran todos los canales que transitan esta ruta.

MQIPT proporciona un recurso de tiempo de espera desocupado opcional. Si MQIPT detecta que un canal ha estado desocupado durante un periodo de tiempo que supera el tiempo de espera, realiza una conclusión inmediata de las dos conexiones específicas.

Los sistemas IBM MQ en cualquiera de ambos extremos del canal observan estas condiciones anómalas de cierre, ya sea como anomalías de red o como terminación del canal por parte de su socio. A continuación, se puede reiniciar y recuperar el canal (si el error se produce durante un periodo dudoso del protocolo), como si no se estuviera utilizando MQIPT.

Seguridad de mensajes

La gestión de colas distribuidas de IBM MQ garantiza que los mensajes se entreguen correctamente. Esto continúa siendo el caso cuando está presente MQIPT entre los dos extremos del canal. MQIPT no almacena datos de mensajes ni participa en el procedimiento de punto de sincronización que garantiza la entrega correcta de los mensajes.

Cuando se utilizan mensajes de IBM MQ rápidos y no persistentes, si la ruta de MQIPT falla o se reinicia cuando un mensaje de IBM MQ está en tránsito, es posible que se pierda el mensaje. Antes de reiniciar la ruta, asegúrese de que todos los canales IBM MQ que utilizan la ruta MQIPT estén inactivos.

Para obtener más información sobre la seguridad de los mensajes en IBM MQ, consulte Seguridad de mensajes.

Gestores de colas de varias instancias y alta disponibilidad

MQIPT se puede utilizar con gestores de colas de varias instancias en entornos de alta disponibilidad.

MQIPT no tiene ningún estado persistente y, por lo tanto, no supone ninguna ventaja realizar la migración tras error de MQIPT a otro sistema. En su lugar, tenga varias instancias de MQIPT con archivos de configuración `mqipt.conf` idénticos que se ejecutan en sistemas diferentes. Supervise la disponibilidad de cada instancia de MQIPT y reiníciela (en el mismo sistema) si es necesario. Esto proporciona un conjunto de instancias de MQIPT idénticas que se pueden utilizar para direccionar las conexiones. A continuación, debe asegurarse de que IBM MQ puede direccionar las conexiones a MQIPT y de que MQIPT puede reenviar esas conexiones al gestor de colas de destino.

Los canales de IBM MQ de salida se pueden dirigir a una instancia de MQIPT disponible de varias maneras, por ejemplo:

- Utilice un equilibrador de carga o un direccionador de alta disponibilidad, tal como IBM Network Dispatcher, desde el producto WebSphere Edge Components.
- Especifique varios nombres de conexión en la definición del canal IBM MQ utilizando una lista separada por comas. A continuación, IBM MQ intenta conectar con cada dirección de MQIPT por orden, hasta que encuentra una instancia de MQIPT disponible.

También debe direccionar las conexiones desde MQIPT al gestor de colas de destino. Si la configuración de alta disponibilidad garantiza que la dirección migra tras error con el gestor de colas de destino, no es necesaria ninguna configuración especial de MQIPT: Especifique la dirección IP de destino en la propiedad de ruta **Destination** y permita la operación de migración tras error para mover la dirección IP con el gestor de colas.

No obstante, si la dirección IP del gestor de colas cambia después de una migración tras error, debe establecer que MQIPT envíe la conexión al destino correcto. Esto se puede hacer de una de las maneras siguientes:

- Escriba una salida de direccionamiento que compruebe qué dirección IP y número de puerto están accesibles y, a continuación, reemplace el destino de ruta de cada conexión. Existen algunas salidas de direccionamiento de ejemplo que se proporcionan con MQIPT; se pueden adaptar para este fin.
- Utilice un equilibrador de carga de alta disponibilidad para redireccionar la conexión.
- Defina varias rutas de MQIPT, una para cada dirección IP y puerto donde es posible que se esté ejecutando el gestor de colas. A continuación, dirija las conexiones de IBM MQ a las distintas rutas de MQIPT, por ejemplo, listando todas las direcciones IP de ruta y los números de puerto en una lista separada por comas en el nombre de conexión del canal de salida.

También es importante ajustar todos los componentes de un extremo a otro en la vía de acceso de red:

1. Los intentos de conexión con sistemas no disponibles deben fallar rápidamente, de modo que los intentos de reconexión puedan pasar al primer destino disponible.

   Para las rutas SSL de MQIPT, ajuste la propiedad de ruta **SSLClientConnectTimeout** para asegurar un error de conexión rápido para destinos no disponibles. Consulte la documentación de IBM MQ para obtener más detalles sobre los parámetros de ajuste de IBM MQ. Asimismo, consulte la documentación del sistema operativo para obtener detalles sobre cómo ajustar TCP/IP para el sistema operativo. En todos los casos, los intentos de conexión erróneos deben devolver rápidamente un error de red (por ejemplo, un paquete de restablecimiento TCP) o deben agotar el tiempo de espera sin ningún retardo indebido.
2. Las conexiones activas con un sistema anómalo se deben interrumpir rápidamente para que se puedan establecer nuevas conexiones.

   También debe tener en cuenta el impacto de una migración tras error en un periodo en que las conexiones están utilizando MQIPT activamente. Es probable que las conexiones de red se interrumpan durante una migración tras error. En el caso de las aplicaciones cliente, puede utilizar la característica de reconexión automática de cliente de IBM MQ para restablecer las conexiones interrumpidas. Para los canales de mensajes, puede especificar un intervalo de reintento corto, de modo que el canal se vuelva a conectar rápidamente. Consulte la documentación de IBM MQ para obtener más información sobre la reconexión automática del cliente y la configuración del reintento del canal de mensajes.

IBM MQ Console y REST API

Puede utilizar IBM MQ Console y REST API para administrar IBM MQy realizar operaciones de mensajería utilizando HTTP.

- Puede utilizar IBM MQ Console para realizar tareas de administración básicas desde un navegador web. Si desea más información, consulte Administración utilizando IBM MQ Console.
- Puede utilizar la administrative REST API para administrar objetos IBM MQ como, por ejemplo, gestores de colas y colas, y agentes y transferencias Managed File Transfer. Si desea más información, consulte Administración utilizando REST API.

- Puede utilizar messaging REST API para realizar una mensajería de punto a punto simple y de publicación. Para obtener más información, consulte Mensajería utilizando REST API.

Opciones de instalación

IBM MQ Console y REST API se ejecutan en un servidor WebSphere Liberty , denominado mqweb. A partir de IBM MQ 9.3.5, puede instalar el servidor mqweb como un componente opcional en una instalación de IBM MQ o como una instalación autónoma de IBM MQ Web Server .

Linux V 9.4.0 **Instalación de IBM MQ Web Server autónomo**

A partir de IBM MQ 9.4.0, el servidor mqweb se puede ejecutar en una instalación autónoma de IBM MQ Web Server. Una instalación autónoma de IBM MQ Web Server le permite instalar y ejecutar el servidor mqweb en sistemas independientes de las instalaciones de IBM MQ . La instalación de un IBM MQ Web Server autónomo proporciona una mayor flexibilidad en cuanto a qué sistemas, y el número de sistemas, en los que elige ejecutar los servidores mqweb. Si es necesario, se pueden ejecutar varias instancias del servidor mqweb en distintas máquinas para proporcionar la escalabilidad y la disponibilidad que necesita.

Si ha adquirido la titularidad de IBM MQ , puede instalar tantas copias como sea necesario del IBM MQ Web Serverautónomo. Las instancias de IBM MQ Web Server no se contarán en relación con la titularidad de IBM MQ que ha adquirido. Para obtener más información sobre la licencia de IBM MQ, consulte Información de licencia de IBM MQ.

Las restricciones siguientes se aplican en una instalación autónoma de IBM MQ Web Server :

- IBM MQ Console se puede utilizar para administrar sólo gestores de colas remotos.
- El messaging REST API sólo se puede utilizar con gestores de colas remotos.
- El administrative REST API no está disponible.

El IBM MQ Web Server autónomo solo está soportado en plataformas Linux .

Para obtener más información sobre la instalación del IBM MQ Web Serverautónomo, consulte Instalación del IBM MQ Web Serverautónomo.

Componente opcional de una instalación de IBM MQ

Puede optar por instalar el componente IBM MQ Console y REST API como parte de una instalación de IBM MQ .

Todas las características IBM MQ Console y REST API están disponibles cuando el servidor mqweb se ejecuta en una instalación de IBM MQ .

- IBM MQ Console se puede utilizar para administrar gestores de colas locales y remotos.
- El messaging REST API se puede utilizar con gestores de colas locales y remotos.
- administrative REST API se puede utilizar para administrar gestores de colas locales y remotos.

Para utilizar el componente IBM MQ Console y REST API , instale el componente siguiente como parte de la instalación de IBM MQ :

- AIX En AIX, instale el conjunto de archivos `mqm.web.rte`.
- IBM i En IBM i, instale el componente WEB.
- Linux En Linux, instale el componente `MQSeriesWeb`.
- Windows En Windows, instale la característica `Web Administration`.
- z/OS En z/OS, instale la característica `IBM MQ for z/OS UNIX System Services Web Components`.

Avisos

Esta información se ha desarrollado para productos y servicios ofrecidos en los Estados Unidos.

Es posible que IBM no ofrezca los productos, servicios o las características que se tratan en este documento en otros países. Consulte al representante local de IBM para obtener información sobre los productos y servicios que actualmente pueden adquirirse en su zona. Las referencias a programas, productos o servicios de IBM no pretenden establecer ni implicar que sólo puedan utilizarse dichos productos, programas o servicios de IBM. En su lugar podrá utilizarse cualquier producto, programa o servicio equivalente que no infrinja ninguno de los derechos de propiedad intelectual de IBM. No obstante, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio no IBM.

IBM puede tener patentes o solicitudes de patentes pendientes que cubran el tema principal descrito en este documento. El suministro de este documento no le otorga ninguna licencia sobre estas patentes. Puede enviar consultas sobre licencias, por escrito, a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Para consultas sobre licencias relacionadas con información de doble byte (DBCS), póngase en contacto con el Departamento de propiedad intelectual de IBM de su país o envíe las consultas por escrito a:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokio 103-8510, Japón

El párrafo siguiente no se aplica al Reino Unido ni a ningún otro país donde estas disposiciones contradigan la legislación vigente: INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL" SIN NINGÚN TIPO DE GARANTÍA, YA SEA EXPLÍCITA O IMPLÍCITA, INCLUYENDO, PERO SIN LIMITARSE A, LAS GARANTÍAS IMPLÍCITAS DE NO INCUMPLIMIENTO, COMERCIABILIDAD O IDONEIDAD PARA UNA FINALIDAD DETERMINADA. Algunas legislaciones no contemplan la exclusión de garantías, ni implícitas ni explícitas, en determinadas transacciones, por lo que puede haber usuarios a los que no les afecte dicha norma.

Esta información puede contener imprecisiones técnicas o errores tipográficos. La información aquí contenida está sometida a cambios periódicos; tales cambios se irán incorporando en nuevas ediciones de la publicación. IBM puede realizar mejoras y/o cambios en los productos y/o programas descritos en esta publicación en cualquier momento sin previo aviso.

Las referencias hechas en esta publicación a sitios web que no son de IBM se proporcionan sólo para la comodidad del usuario y no constituyen de modo alguno un aval de esos sitios web. Los materiales de estos sitios web no forman parte de los materiales para este producto IBM, por lo que la utilización de dichos sitios web es a cuenta y riesgo del usuario.

IBM puede utilizar o distribuir cualquier información que el usuario le proporcione del modo que considere apropiado sin incurrir por ello en ninguna obligación con respeto al usuario.

Los titulares de licencias de este programa que deseen información del mismo con el fin de permitir: (i) el intercambio de información entre los programas creados de forma independiente y otros programas (incluido este) y (ii) el uso mutuo de la información intercambiada, deben ponerse en contacto con:

IBM Corporation
Software Interoperability Coordinator, Department 49XA

3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluyendo, en algunos casos, el pago de una cantidad.

El programa bajo licencia que se describe en esta información y todo el material bajo licencia disponible para el mismo lo proporciona IBM bajo los términos del Acuerdo de cliente de IBM, el Acuerdo de licencia de programas internacional de IBM o cualquier acuerdo equivalente entre las partes.

Los datos de rendimiento incluidos en este documento se han obtenido en un entorno controlado. Por consiguiente, los resultados obtenidos en otros entornos operativos pueden variar de manera significativa. Es posible que algunas mediciones se hayan realizado en sistemas en nivel de desarrollo y no existe ninguna garantía de que estas mediciones serán las mismas en sistemas disponibles generalmente. Además, es posible que algunas mediciones se hayan estimado mediante extrapolación. Los resultados reales pueden variar. Los usuarios de este documento deben verificar los datos aplicables a su entorno específico.

La información relativa a productos que no son de IBM se obtuvo de los proveedores de esos productos, sus anuncios publicados u otras fuentes de disponibilidad pública. IBM no ha comprobado estos productos y no puede confirmar la precisión de su rendimiento, compatibilidad o alguna reclamación relacionada con productos que no sean de IBM. Todas las preguntas sobre las prestaciones de productos que no son de IBM deben dirigirse a los proveedores de dichos productos.

Todas las declaraciones relacionadas con una futura intención o tendencia de IBM están sujetas a cambios o se pueden retirar sin previo aviso y sólo representan metas y objetivos.

Este documento contiene ejemplos de datos e informes que se utilizan diariamente en la actividad de la empresa. Para ilustrar los ejemplos de la forma más completa posible, éstos incluyen nombres de personas, empresas, marcas y productos. Todos estos nombres son ficticios y cualquier similitud con los nombres y direcciones utilizados por una empresa real es puramente casual.

LICENCIA DE DERECHOS DE AUTOR:

Esta información contiene programas de aplicación de ejemplo en lenguaje fuente que ilustran técnicas de programación en diversas plataformas operativas. Puede copiar, modificar y distribuir estos programas de ejemplo de cualquier forma sin pagar ninguna cuota a IBM para fines de desarrollo, uso, marketing o distribución de programas de aplicación que se ajusten a la interfaz de programación de aplicaciones para la plataforma operativa para la que se han escrito los programas de ejemplo. Los ejemplos no se han probado minuciosamente bajo todas las condiciones. IBM, por tanto, no puede garantizar la fiabilidad, servicio o funciones de estos programas.

Puede que si visualiza esta información en copia software, las fotografías e ilustraciones a color no aparezcan.

Información acerca de las interfaces de programación

La información de interfaz de programación, si se proporciona, está pensada para ayudarle a crear software de aplicación para su uso con este programa.

Este manual contiene información sobre las interfaces de programación previstas que permiten al cliente escribir programas para obtener los servicios de IBM MQ.

Sin embargo, esta información puede contener también información de diagnóstico, modificación y ajustes. La información de diagnóstico, modificación y ajustes se proporciona para ayudarle a depurar el software de aplicación.

Importante: No utilice esta información de diagnóstico, modificación y ajuste como interfaz de programación porque está sujeta a cambios.

Marcas registradas

IBM, el logotipo de IBM , ibm.com, son marcas registradas de IBM Corporation, registradas en muchas jurisdicciones de todo el mundo. Hay disponible una lista actual de marcas registradas de IBM en la web en "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Otros nombres de productos y servicios pueden ser marcas registradas de IBM o de otras empresas.

Microsoft y Windows son marcas registradas de Microsoft Corporation en Estados Unidos y/o otros países.

UNIX es una marca registrada de Open Group en Estados Unidos y en otros países.

Linux es una marca registrada de Linus Torvalds en Estados Unidos y en otros países.

Este producto incluye software desarrollado por Eclipse Project (https://www.eclipse.org/).

Java y todas las marcas registradas y logotipos son marcas registradas de Oracle o sus afiliados.

Número Pieza:

(1P) P/N: