9.4

Visão geral técnica do IBM MQ

Nota

Antes de usar estas informações e o produto que elas suportam, leia as informações em “Avisos” na página 307.

Esta edição se aplica à versão 9, liberação 4 do IBM® MQ e a todas as liberações e modificações subsequentes, até que seja indicado de outra forma em novas edições

Ao enviar informações para a IBM, você concede à IBM um direito não exclusivo de usar ou distribuir as informações da maneira que julgar apropriada, sem incorrer em qualquer obrigação para com você

Índice

Visão Geral Técnica do IBM MQ

Use o IBM MQ para conectar seus aplicativos e gerenciar a distribuição de informações em sua organização.

O IBM MQ permite que programas se comuniquem uns com os outros por meio de uma rede de componentes diferentes (processadores, sistemas operacionais, subsistemas e protocolos de comunicação) usando uma interface de programação de aplicativos consistente. Os aplicativos projetados e gravados usando esta interface são conhecidos como aplicativos de enfileiramento de mensagens.

Use os subtópicos a seguir para descobrir sobre o enfileiramento de mensagens e outros recursos fornecidos pelo IBM MQ.

Conceitos relacionados

Introdução ao IBM MQ

Onde localizar informações de requisitos e suporte do produto

Tarefas relacionadas

Planejando uma arquitetura do IBM MQ

Referências relacionadas

“Principais Recursos e Benefícios do Enfileiramento de Mensagens” na página 7

Essas informações destacam alguns recursos e benefícios do enfileiramento de mensagens. Descreve recursos como segurança e integridade de dados do enfileiramento de mensagens.

Introdução ao Enfileiramento de Mensagens

Os produtos IBM MQ permitem que os programas se comuniquem entre si em uma rede de componentes diferentes (processadores, sistemas operacionais, subsistemas e protocolos de comunicação) usando uma interface de programação de aplicativos consistente.

Os aplicativos designados e gravados usando esta interface são mostrados como aplicativos de *enfileiramento de mensagens*, porque usam o estilo de *sistema de mensagens* e de *enfileiramento*:

- Sistema de mensagens significa que os programas se comunicam enviando uns aos outros dados em mensagens, em vez de chamar uns aos outros diretamente.
- Enfileiramento significa que as mensagens são colocadas em filas no armazenamento, permitindo que os programas sejam executados independentemente uns dos outros, em velocidades e horários diferentes, em locais diferentes e sem que haja uma conexão lógica entre eles.

O enfileiramento de mensagens tem sido usado no processamento de dados por vários anos. Hoje ele é frequentemente usado no correio eletrônico. Sem o enfileiramento, o envio de uma mensagem eletrônica a longas distâncias requer que cada nó na rota fique disponível para o encaminhamento de mensagens e que os endereços sejam registrados e reconheçam que você está tentando enviar uma mensagem a eles. Em um sistema de enfileiramento, as mensagens são armazenadas nos nós intermediários até que o sistema fique pronto para encaminhá-las. Em seus destinos finais, elas são armazenadas em uma caixa de correio eletrônico até que o destinatário esteja pronto para lê-las.

Mesmo assim, muitas transações de negócios complexas são processadas atualmente sem o enfileiramento. Em uma rede ampla, o sistema pode estar mantendo milhares de conexões em um estado pronto para uso. Se uma parte do sistema sofrer um problema, várias partes do sistema podem ficar inutilizáveis.

É possível imaginar um enfileiramento de mensagens como sendo o correio eletrônico para os programas. Em um ambiente de enfileiramento de mensagens, cada programa que faz parte de um conjunto de aplicativos executa uma função autocontida e bem definida em resposta a uma solicitação específica. Para comunicar-se com outro programa, um programa deve colocar uma mensagem em uma fila predefinida. O outro programa recupera a mensagem da fila e processa as solicitações e as informações contidas na mensagem. Portanto, o enfileiramento de mensagens é um estilo de comunicação programa-a-programa.

O enfileiramento é o mecanismo pelo qual as mensagens são retidas até que um aplicativo esteja pronto para processá-las. O enfileiramento permite:

- Comunicar-se entre os programas (que podem estar em execução em ambientes diferentes) sem ter que gravar o código de comunicação.
- Selecionar a ordem na qual um programa processa as mensagens.
- Equilibrar os carregamentos em um sistema organizando mais de um programa para atender uma fila quando o número de mensagens exceder um limite.
- Aumentar a disponibilidade dos seus aplicativos organizando um sistema alternativo para atender as filas se o seu sistema primário estiver indisponível.

O que é uma fila de mensagens?

Uma fila de mensagens, conhecida simplesmente como uma fila, é um destino nomeado para o qual as mensagens podem ser enviadas. As mensagens se acumulam em filas até que sejam recuperadas pelos programas que atendem essas filas.

Filas residem em, e são gerenciadas por, um gerenciador de filas (consulte "Terminologia de Enfileiramento de Mensagens" na página 9). A natureza física de uma fila depende do sistema operacional no qual o gerenciador de filas está sendo executado. Uma fila pode ser área de buffer volátil na memória de um computador ou um conjunto de dados em um dispositivo de armazenamento permanente (como um disco). O gerenciamento físico de filas é a responsabilidade do gerenciador de filas e não se torna aparente aos programas de aplicativo participantes.

Os programas acessam as filas apenas por meio dos serviços externos do gerenciador de filas. Eles podem abrir uma fila, colocar as mensagens nela, obter as mensagens a partir dela e fechar a fila. Eles também podem configurar, e perguntar sobre, os atributos das filas.

Diferentes Estilos de Enfileiramento de Mensagens

Ponto a ponto

Uma mensagem é colocada na fila e um aplicativo recebe essa mensagem.

No sistema de mensagens ponto a ponto, um aplicativo de envio deve ter informações sobre o aplicativo de recebimento antes de poder enviar uma mensagem para ele. Por exemplo, o aplicativo de envio precisará saber o nome da fila para a qual enviar as informações e também poderá especificar um nome do gerenciador de filas.

Publicação/Assinatura

Uma cópia de cada mensagem publicada por um aplicativo de publicação é entregue para cada aplicativo interessado. Pode haver vários, um ou nenhum aplicativo interessado. Na publicação/assinatura, um aplicativo interessado é conhecido como assinante, e as mensagens são enfileiradas em uma fila identificada por uma assinatura.

O sistema de mensagens de publicação/assinatura permite separar o provedor de informações dos consumidores dessas informações. O aplicativo de envio e o aplicativo de recebimento não precisam saber nada um do outro para que as informações sejam enviadas e recebidas. Para obter mais informações, consulte "Sistema de Mensagens de Publicação/Assinatura" na página 64.

Benefícios do Enfileiramento de Mensagens para o Editor de Telas e Desenvolvedor

O IBM MQ permite que os programas de aplicativo usem o *enfileiramento de mensagens* para participar no processamento orientado à mensagem. Os programas de aplicativo podem se comunicar entre diferentes plataformas usando os produtos de software de enfileiramento de mensagens apropriados. Por exemplo, os aplicativos z/OS podem se comunicar por meio de IBM MQ for z/OS. Os aplicativos são blindados a partir das mecânicas das comunicações subjacentes. Alguns outros benefícios do enfileiramento de mensagens são:

- É possível projetar os aplicativos usando pequenos programas que podem ser compartilhados entre vários aplicativos.
- É possível construir novos aplicativos reutilizando esses blocos de construção.
- Os aplicativos gravados para usar as técnicas de enfileiramento de mensagens não são afetados por mudanças na maneira como esses gerenciadores de filas trabalham.
- Você não precisa usar qualquer protocolo de comunicação. O gerenciador de filas lida com todos os aspectos da comunicação.
- Os programas que recebem as mensagens não precisam estar em execução no momento em que as mensagem são enviadas. As mensagens são retidas nas filas.

Os designers podem reduzir o custo de seus aplicativos porque o desenvolvimento é mais rápido, menos desenvolvedores são necessários e as demandas na qualificação de programação são menores do que para os aplicativos que não usam o enfileiramento de mensagens.

IBM MQ implementa uma interface de programação de aplicativos comum conhecida como a *interface de fila de mensagens* (ou MQI) sempre que os aplicativos são executados. Isso facilita portar os programas de aplicativos de uma plataforma a outra.

Para obter detalhes sobre o MQI, consulte Visão geral do Message Queue Interface.

Principais Recursos e Benefícios do Enfileiramento de Mensagens

Essas informações destacam alguns recursos e benefícios do enfileiramento de mensagens. Descreve recursos como segurança e integridade de dados do enfileiramento de mensagens.

Os principais recursos de aplicativos que usam as técnicas de enfileiramento de mensagens são:

Nota: Ao considerar clientes e servidores IBM MQ, não é necessário alterar um aplicativo do servidor para suportar IBM MQ MQI clients adicionais em novas plataformas. Da mesma forma, o IBM MQ MQI client pode, sem mudança, funcionar com tipos adicionais de servidores.

Sem conexões diretas entre os programas

O enfileiramento de mensagem é uma técnica para comunicação indireta programa a programa. Ele pode ser usado em qualquer aplicativo no qual os programas se comunicam entre si. A comunicação ocorre por um programa que coloca as mensagens em uma fila (de propriedade de um gerenciador de filas) e outro programa que recebe as mensagens da fila.

Os programas podem receber as mensagens que foram colocadas em uma fila por outros programas. Os outros programas podem ser conectados no mesmo gerenciador de filas como o programa de recebimento ou em outro gerenciador de filas. Este outro gerenciador de filas pode estar em outro sistema, um sistema de computador diferente ou até em um negócio ou empreendimento diferente.

Não existem conexões físicas entre os programas que se comunicam usando as filas de mensagens. Um programa envia mensagens em uma fila de propriedade de um gerenciador de filas e outro programa recupera as mensagens da fila (consulte Figura 1 na página 8).

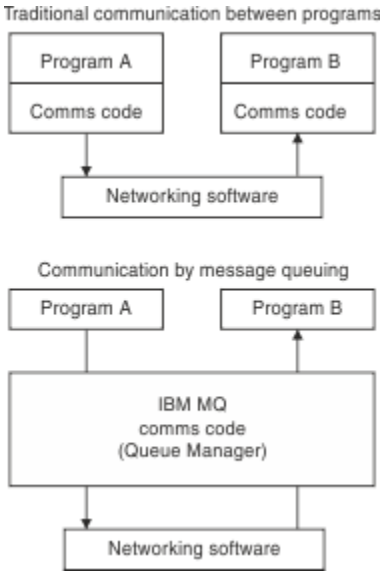


Figura 1. Enfileiramento de Mensagens Comparado à Comunicação Tradicional

Assim como no correio eletrônico, as mensagens individuais que fazem parte de uma transação viajam por meio de uma rede com base em um armazenamento e encaminhamento. Se um link entre os nós falhar, a mensagem será mantida até que o link seja restaurado ou o operador ou o programa redirecione a mensagem.

O mecanismo pelo qual uma mensagem move de fila para fila é oculto dos programas. Portanto, os programas são mais simples.

Comunicação independente de horário

Os programas que precisam de outros para executar o trabalho não precisam aguardar pela resposta para uma solicitação. Eles podem realizar outros trabalhos e processar a resposta quando chegar ou mais recentemente. Ao gravar um aplicativo de mensagens, não é necessário saber (ou se preocupar) quando um programa enviar uma mensagem ou quando o destino conseguir receber a mensagem. A mensagem não é perdida; ela é retida pelo gerenciador de filas até que o destino esteja pronto para processá-la. A mensagem permanece na fila até que seja removida por um programa. Isso significa que os programas de aplicativo de envio e recebimento são separados; o emissor pode continuar processando sem aguardar que o receptor reconheça o recebimento da mensagem. O aplicativo de destino não precisa estar em execução quando a mensagem for enviada. Ele pode recuperar a mensagem depois que ela tiver sido iniciada.

Programas pequenos

O enfileiramento de mensagens permite as vantagens de usar programas pequenos e autocontidos. Em vez de um único programa grande executando doas as partes de uma tarefa sequencialmente, é possível espalhar a tarefa em diversos programas independentes menores. O programa de solicitação envia as mensagens a cada um dos programas separados, pedindo que executem suas funções; quando cada programa for concluído, os resultados serão enviados de volta como uma ou mais mensagens.

Processamento orientado a mensagens

Quando as mensagens chegarem a uma fila, elas poderão iniciar automaticamente um aplicativo usando *triggering*. Se necessário, os aplicativos podem ser interrompidos quando a mensagem (ou mensagens) tiverem sido processadas.

Processamento Direcionado a Eventos

Os programas podem ser controlados de acordo com o estado das filas. Por exemplo, é possível organizar um programa para que seja iniciado assim que uma mensagem chegar a uma fila ou é possível especificar que o programa não seja iniciado até que existam, por exemplo, 10 mensagens acima de uma certa prioridade na fila ou 10 mensagens de qualquer prioridade na fila.

Prioridade da mensagem

Um programa pode designar uma prioridade a uma mensagem quando colocar a mensagem em uma fila. Isso determina a posição na fila na qual a nova mensagem é incluída.

Os programas podem receber mensagens de uma fila na ordem em que as mensagens ficam na fila ou recebendo uma mensagem específica. (Talvez um programa queira obter uma mensagem específica, se estiver procurando pela resposta a uma solicitação que enviou anteriormente.)

Segurança

São fornecidos recursos de segurança, incluindo autenticação de aplicativos quando eles usam um gerenciador de filas, verificações de autorização quando eles usam recursos, como uma fila no gerenciador de filas e criptografia de dados da mensagem conforme eles circulam pela rede e enquanto residem em filas. Para obter informações adicionais sobre segurança, consulte Visão geral de segurança.

Integridade de dados

A integridade de dados é fornecida por unidades de trabalho. A sincronização do início e do término das unidades de trabalho é totalmente suportada como uma opção em cada MQGET ou MQPUT, permitindo que os resultados da unidade de trabalho sejam confirmados ou retrocedidos. O suporte de ponto de sincronização opera interna ou externamente no IBM MQ, dependendo da forma de coordenação de ponto de sincronização selecionada para o aplicativo.

Suporte à recuperação

Para que a recuperação seja possível, todas as atualizações persistentes do IBM MQ são registradas. Se a recuperação for necessária, todas as mensagens persistentes serão restauradas, todas as transações em trânsito serão retrocedidas e qualquer confirmação de ponto de sincronização e restaurações serão tratas da maneira normal no gerenciador de ponto de sincronização em controle. Para obter informações adicionais sobre mensagens persistentes, consulte Mensagens persistentes.

Terminologia de Enfileiramento de Mensagens

Essas informações fornecem um insight em alguns termos usados no enfileiramento de mensagens.

São eles:

- Canais
- Cluster
- IBM MQ MQI client
- z/OS Enfileiramento intragrupo
- Mensagem
- Agente do canal de mensagens
- Descritor de mensagens
- Ponto a ponto
- Publicação/Assinatura
- Fila
- Gerenciador de Filas

- z/OS Grupo de filas compartilhadas
- z/OS Fila compartilhada
- Assinatura
- Tópico

Canais

Os canais são usados para mover as mensagens de um gerenciador de filas para outro e protegem os aplicativos dos protocolos de comunicações subjacentes. Os gerenciadores de filas podem existir no mesmo sistema ou em sistemas diferentes na mesma plataforma ou ainda em plataformas diferentes. As mensagens que são enviadas podem se originar de vários locais:

- Programas de aplicativo gravados pelo usuário que transferem dados de um nó para outro.
- Aplicativos de administração gravados pelo usuário que usam comandos PCF ou a MQAI.
- O IBM MQ Explorer.
- Gerenciadores de filas que enviam mensagens de evento de instrumentação para outro gerenciador de filas.
- Gerenciadores de filas que enviam comandos de administração remota para outro gerenciador de filas. Por exemplo, usando comandos MQSC ou o administrative REST API.

Para obter mais informações sobre os canais, consulte “Definições de canal” na página 32

Cluster

Um *cluster* é uma rede de gerenciadores de filas que estão associados logicamente de alguma maneira.

Em uma rede do IBM MQ que usa enfileiramento distribuído sem cluster, cada gerenciador de filas é independente. Se um gerenciador de filas precisar enviar as mensagens para outro, ele deve ter definir uma fila de transmissão e um canal para outro gerenciador de filas remotas.

Existem duas razões diferentes para usar os clusters: reduzir a administração do sistema e melhorar a disponibilidade e o balanceamento de carga de trabalho.

Assim que estabelecer até mesmo o menor de todos os clusters, você será beneficiado pela administração de sistema simplificada. Os gerenciadores de fila que fazem parte de um cluster precisarão de menos definições e assim o risco de cometer um erro nas suas definições é reduzido.

Para obter informações adicionais sobre armazenamento em cluster, consulte Clusters.

IBM MQ MQI client

Os IBM MQ clientes MQI são componentes instaláveis independentemente do IBM MQ Um cliente do MQI permite que você execute aplicativos IBM MQ com um protocolo de comunicação, para interagir com um ou mais servidores do Message Queue Interface (MQI) em outras plataformas e para conectar-se a seus gerenciadores de filas.

Para obter detalhes completos sobre como instalar e usar os componentes do IBM MQ MQI client, veja os tópicos a seguir:

- AIX Instalando um cliente do IBM MQ no AIX
- Linux Instalando um cliente do IBM MQ no Linux®
- Windows Instalando um cliente do IBM MQ no Windows
- IBM i Instalando um cliente do IBM MQ no IBM i

e Configurando conexões entre o servidor e o cliente.

Fila do intra-grupo

z/OS

Os gerenciadores de filas em um grupo de filas compartilhadas podem se comunicar usando canais normais ou é possível usar uma técnica chamada *enfileiramento intragrupo* (IGQ), o que permite executar a transferência de mensagem rápida sem definir os canais. Isso é aplicável apenas ao IBM MQ for z/OS.

Para obter mais informações sobre o enfileiramento intragrupo, consulte “Intra-group queuing” na página 216

Mensagem

No enfileiramento de mensagens, uma mensagem é uma coleta de dados enviados por um programa e pretendido para outro programa. Consulte mensagens do IBM MQ.

Para obter informações sobre os tipos de mensagens, consulte Tipos de mensagem.

Agente do canal de mensagens

Um agente do canal de mensagens é uma extremidade de um canal. Um par de agentes do canal de mensagens, um envio e um recebimento formam um canal e movem mensagens de um gerenciador de filas para outro.

Para obter informações sobre como os agentes do canal de mensagens são usados, consulte Introdução ao gerenciamento de filas distribuídas.

Descritor de Mensagens

Uma mensagem do IBM MQ consiste em informações de controle e dados de aplicativos.

As informações de controle são definidas em uma estrutura de descritor de mensagens (MQMD) e contêm coisas como:

- O tipo da mensagem
- Um identificador para a mensagem
- A prioridade para a entrega da mensagem

A estrutura e o conteúdo dos dados do aplicativo são determinados pelos programas participantes, não pelo IBM MQ.

Para obter mais informações, consulte MQMD..

Sistema de mensagens ponto a ponto

No sistema de mensagens ponto a ponto, cada mensagem passa de um aplicativo produtor para um aplicativo consumidor. As mensagens são transferidas por meio do aplicativo produtor, que as coloca em uma fila da qual o aplicativo consumidor as retira.

Sistema de Mensagens de Publicação/Assinatura

No sistema de mensagens de publicação/assinatura, uma cópia de cada mensagem publicada por um aplicativo de publicação é entregue a cada aplicativo interessado. Pode haver vários, um ou nenhum aplicativo interessado. Na publicação/assinatura, um aplicativo interessado é conhecido como assinante, e as mensagens são enfileiradas em uma fila identificada por uma assinatura.

Para obter informações adicionais, consulte “Sistema de Mensagens de Publicação/Assinatura” na página 64.

Fila

Um destino nomeado para o qual mensagens podem ser enviadas. As mensagens se acumulam em filas até que sejam recuperadas pelos programas que atendem essas filas.

Para obter informações adicionais, consulte “Filas” na página 20.

Gerenciador de filas

Um *gerenciador de filas* é um programa do sistema que fornece serviços de enfileiramento para os aplicativos.

Fornece uma interface de programação de aplicativos para que os programas possam colocar as mensagens em, e obter as mensagens de, filas. Um gerenciador de filas fornece funções adicionais para que os administradores possam criar novas filas, alterar as propriedades das filas existentes e controlar a operação do gerenciador de filas.

Para que serviços de enfileiramento de mensagens do IBM MQ estejam disponíveis em um sistema, deve haver um gerenciador de filas em execução. É possível ter mais de um gerenciador de filas em execução em um único sistema (por exemplo, para separar um sistema de teste de um sistema de *produção*). Para um aplicativo, cada gerenciador de filas é identificado por uma *manipulação de conexões* (*Hconn*).

Muitos aplicativos diferentes podem usar os serviços do gerenciador de filas ao mesmo tempo, e esses aplicativos podem estar inteiramente desvinculados. Para que um programa use os serviços de um gerenciador de filas, ele deve estabelecer uma conexão a esse gerenciador de filas.

Para que os aplicativos enviem as mensagens aos aplicativos que estão conectados a outros gerenciadores de fila, os gerenciadores de vila devem conseguir se comunicar entre si. IBM MQ implementa um protocolo de *armazenamento e encaminhamento* para assegurar a entrega segura de mensagens entre esses aplicativos.

Para obter informações adicionais, consulte “Gerenciadores de filas” na página 29.

Grupo de filas compartilhadas

z/OS

Os gerenciadores de filas que podem acessar o mesmo conjunto de filas compartilhadas formam um grupo denominado *grupo de filas compartilhadas* (QSG). Eles se comunicam uns com os outros com o recurso de acoplamento (CF) que armazena as filas compartilhadas. Isso é aplicável apenas ao IBM MQ for z/OS.

Para obter informações adicionais, consulte “Shared queues and queue sharing groups” na página 172.

Fila compartilhada

z/OS

Uma *fila compartilhada* é um tipo de fila local com mensagens que podem ser acessadas por um ou mais gerenciadores de filas que estão em um sysplex. Isso não é igual a uma fila sendo compartilhada por mais de um aplicativo, usando o mesmo gerenciador de filas. Isso é aplicável apenas ao IBM MQ for z/OS.

Assinatura

Um aplicativo de publicação/assinatura pode registrar um interesse nas mensagens sobre tópicos específicos. Quando faz isso, o aplicativo fica conhecido como assinante, e o termo assinatura define como as mensagens correspondentes são enfileiradas para o processamento.

Uma assinatura contém informações sobre a identidade do assinante e a identidade da fila de destino na qual as publicações devem ser colocadas. Ela também contém informações sobre como uma publicação deve ser colocada na fila de destino.

Para obter informações adicionais, consulte “Assinantes e assinaturas” na página 68.

Tópico

Um tópico é uma cadeia de caracteres que descreve o assunto das informações que são publicadas em uma mensagem de publicação/assinatura.

Os tópicos são a chave para a entrega de mensagens bem-sucedida em um sistema de Publicação/Assinatura. Em vez de incluir um endereço de destino específico em cada mensagem, um publicador designa um tópico a cada mensagem. O gerenciador de filas corresponde ao tópico com uma lista de assinantes que assinaram esse tópico, e entrega a mensagem para cada um desses assinantes.

Para obter informações adicionais, consulte “tópicos” na página 70.

Mensagens e Filas

As mensagens e as filas são componentes básicos de um sistema de enfileiramento de mensagens.

O que é uma mensagem?

Um *mensagem* é uma sequência de bytes que é significativa para os aplicativos que a usam. As mensagens são usadas para transferir as informações de um programa de aplicativo a outro (ou entre diferentes partes do mesmo aplicativo). Os aplicativos podem estar em execução na mesma plataforma ou em diferentes plataformas.

Uma mensagem do IBM MQ é composta de:

- *Os dados do aplicativo.*O conteúdo e a estrutura dos dados do aplicativo são definidos pelos programas de aplicativo que o usam.
- *Um descritor de mensagens.*O descritor de mensagens identifica a mensagem e contém informações adicionais de controle, como o tipo de mensagem e a prioridade designada à mensagem enviando o aplicativo.

  O formato do descritor de mensagens é definido por IBM MQ. Para obter uma descrição completa do descritor de mensagens, consulte MQMD - Descritor de mensagens.
- *Propriedades de mensagem.* Metadados sobre a mensagem. O conteúdo das propriedades de mensagem é definido pelos programas de aplicativo que os utilizam. Para obter informações adicionais, consulte Propriedades de mensagem.

Comprimentos de Mensagens

O comprimento padrão máximo da mensagem é 4 MB, embora seja possível aumentar isso para um comprimento máximo de 100 MB (em que 1 MB é igual a 1 048 576 bytes). Na prática, o comprimento da mensagem pode ser limitado por:

- Comprimento máximo da mensagem definido para a fila de recepção
- Comprimento máximo da mensagem definido para o gerenciador de fila
- Comprimento máximo da mensagem definido pela fila
- Comprimento máximo da mensagem definido pelo envio ou pela recepção do aplicativo
- A quantidade de armazenamento disponível para a mensagem

Pode haver várias mensagens para enviar todas as informações necessárias para um aplicativo.

Como os Aplicativos Enviam e Recebem as Mensagens?

Os programas de aplicativo enviam e recebem mensagens usando **chamadas MQI**.

Por exemplo, para colocar uma mensagem em uma fila, um aplicativo:

1. Abre a fila necessária, emitindo uma chamada MQI `MQOPEN`
2. Emite uma chamada MQI `MQPUT` para colocar a mensagem na fila

Outro aplicativo pode recuperar a mensagem a partir da mesma fila emitindo uma chamada MQI `MQGET`

Para obter informações adicionais sobre as chamadas MQI, consulte Chamadas MQI.

O que É uma Fila?

Uma *fila* é uma estrutura de dados usada para armazenar as mensagens.

Cada fila é de propriedade de um *gerenciador de fila*. O gerenciador de filas é responsável por manter as filas que possui e por armazenar todas as mensagens que recebe nas filas apropriadas. As mensagens podem ser colocadas na fila pelos programas aplicativos ou um gerenciador de filas como parte de sua operação normal.

Filas Predefinidas e Filas Dinâmicas

As filas podem ser caracterizadas pela maneira como são criadas:

- As **filas predefinidas** são criadas por um administrador usando comandos MQSC ou PCF apropriados.As filas predefinidas são permanentes; elas existem independentemente dos aplicativos que as usam e sobrevivem aos reinícios do IBM MQ.
- As **filas dinâmicas** são criadas quando um aplicativo emite uma solicitação de MQOPEN especificando o nome de uma *fila modelo*. A fila criada é baseada em uma *definição de fila modelo*, que é denominada fila modelo.É possível criar uma fila modelo usando o comando DEFINE QMODEL do MQSC. Os atributos de uma fila modelo (por exemplo, o número máximo de mensagens que podem ser armazenadas) são herdados por qualquer fila dinâmica que tenha sido criada para ela.

  As filas modelos possuem um atributo que especifica se a fila dinâmica deve ser permanente ou temporária. As filas permanentes sobrevivem aos reinício do aplicativo e do gerenciador de filas; as filas temporárias são perdidas no reinício.

Recuperando Mensagens das Filas

Os aplicativos devidamente autorizados podem recuperar as mensagens de uma fila de acordo com os seguintes algoritmos de recuperação:

- Primeiro a entrar, primeiro a sair (FIFO).
- Prioridade da mensagem, conforme definido no descritor de mensagens. As mensagens que possuem a mesma prioridade são recuperadas com base em FIFO.
- Uma solicitação de programa para uma mensagem específica.

A solicitação MQGET do aplicativo determina o método usado.

Objetos do IBM MQ

Gerenciadores de Filas definem as propriedades de objetos do IBM MQ. Os valores dessas propriedades afetam a maneira como o IBM MQ processa esses objetos. É possível criar e gerenciar objetos usando os comandos e interfaces do IBM MQ. A partir de seus aplicativos, você usa a Message Queue Interface (MQI) para controlar objetos. Os objetos são identificados por um IBM MQ *descritor de objeto* (MQOD) quando direcionados de um programa.

Administração de objeto

A administração de objetos inclui as tarefas a seguir:

- Iniciar e parar os gerenciadores de filas.
- Criar objetos, especialmente as filas, para os aplicativos.
- Exibindo ou alterando os atributos de objetos.
- Excluindo objetos.
- Trabalhar com canais para criar caminhos de comunicação para os gerenciadores de fila em outros sistemas (remotos).

- Criar *clusters* dos gerenciadores de fila para simplificar o processo geral de administração e para equilibrar a carga de trabalho.

Com a exceção de filas dinâmicas, objetos devem ser definidos para um gerenciador de filas antes que se possa trabalhar com eles.

Ao utilizar um comando do IBM MQ para transportar uma operação de administração de objeto, o gerenciador de filas verifica se você tem o nível necessário de autoridade para executar a operação. Da mesma forma, quando um aplicativo usa a chamada MQOPEN para abrir um objeto, o gerenciador de filas verifica se o aplicativo possui o nível necessário de autoridade antes que conceda acesso a esse objeto. As verificações são feitas no nome do objeto sendo aberto.

É possível definir e gerenciar objetos usando os seguintes métodos:

- Os comandos PCF descritos em Referência de formatos de comandos programáveis e Automatizando tarefas de administração
- Os comandos MQSC descritos em Os comandos MQSC
- z/OS As operações IBM MQ for z/OS e os painéis de controle, descritos em Administrando IBM MQ for z/OS
- Windows Linux O IBM MQ Explorer (somente sistemas Windows e Linux for Intel). Para obter mais informações, veja Introdução ao MQ Explorer.

Você também pode gerenciar objetos usando os seguintes métodos:

- Os comandos de controle, que são digitado a partir de um teclado. Consulte Administração IBM MQ for Multiplatforms usando comandos de controle.
- IBM MQ As chamadas da Interface de Administração (MQAI) em um programa. Consulte IBM MQ Administration Interface (MQAI).

ALW Para sequências de comandos do IBM MQ no AIX, Linux, and Windows, é possível usar o recurso do MQSC para executar uma série de comandos retidos em um arquivo. Para obter mais informações, consulte Administrando IBM MQ usando os comandos MQSC

IBM i Para sequências de comandos do IBM MQ for IBM i que você usa regularmente, é possível gravar comandos de CL. Para obter mais informações, veja Gerenciando o IBM MQ for IBM i usando comandos de CL.

z/OS Para sequências de comandos do IBM MQ for z/OS que você usa regularmente, é possível gravar programas de administração que criam mensagens que contêm comandos e que colocam essas mensagens na fila de entrada de comando do sistema. O gerenciador de filas processa as mensagens nesta fila da mesma maneira que processa os comandos inseridos a partir da linha de comandos ou das operações e dos painéis de controle. Esta técnica está descrita em Gravando programas para administrar o IBM MQ, e é demonstrada no aplicativo de amostra do Gerenciador de correio entregue com o IBM MQ for z/OS. Para obter uma descrição desta amostra, consulte Programas de amostra para o IBM MQ for z/OS .

Atributos do Objeto

As propriedades de um objeto são definidas por seus atributos. Algumas pessoas podem especificar e outras podem apenas visualizar.

Por exemplo, o comprimento máximo da mensagem que uma fila pode acomodar é definido por seu atributo **MaxMsgLength**; é possível especificar este atributo ao criar uma fila. O atributo **DefinitionType** especifica como a fila foi criada; é possível exibir apenas este atributo.

No IBM MQ, existem duas maneiras de se referir a um atributo:

- Usando seu nome PCF, por exemplo, **MaxMsgLength**.
- usando seu nome de comando MQSC, por exemplo, MAXMSGL.

Grupos de compartilhamento de filas

z/OS

Os gerenciadores de filas que podem acessar o mesmo conjunto de filas compartilhadas formam um grupo denominado *grupo de filas compartilhadas* (QSG) e eles se comunicam entre si usando um recurso de acoplamento (CF) que armazena as filas compartilhadas. Observe que um QSG não é estritamente um objeto..

Uma fila compartilhada é um tipo de fila local com mensagens que podem ser acessadas por um ou mais gerenciadores de filas que estão em um grupo de filas compartilhadas. Isso não é igual a uma fila sendo compartilhada por mais de um aplicativo, usando o mesmo gerenciador de filas.

Os grupos de filas compartilhadas têm um nome de até quatro caracteres. O nome deve ser exclusivo em sua rede e deve ser diferente de qualquer nome de gerenciador de filas.

Importante: As filas compartilhadas e os grupos de filas compartilhadas são suportados apenas no IBM MQ for z/OS.

Consulte “Shared queues and queue sharing groups” na página 172 para obter mais informações.

Objetos Padrão do Sistema

Os *objetos padrão do sistema* são um conjunto de definições de objetos que são criadas automaticamente sempre que um gerenciador de filas é criado. É possível copiar e modificar quaisquer dessas definições de objeto para uso nos aplicativos em sua instalação.

Os nomes de objeto padrão possuem a raiz SYSTEM; por exemplo, a fila local padrão é SYSTEM.DEFAULT.LOCAL.QUEUE e o canal receptor padrão é SYSTEM.DEF.RECEIVER. Não é possível renomear esses objetos; os objetos padrão desses nomes são necessários.

Ao definir um objeto, quaisquer atributos que você não especificar explicitamente serão copiados a partir do objeto padrão apropriado. Por exemplo, se definir uma fila local, esses atributos que você não especificar são extraídos da fila padrão SYSTEM.DEFAULT.LOCAL.QUEUE.

Consulte Objetos do Sistema e Padrão para obter informações adicionais

Tipos de objeto

Várias das tarefas de administração envolvem a manipulação de vários tipos de IBM MQ *objetos*.

Para obter informações sobre a nomeação de objetos IBM MQ, consulte “Nomeando os Objetos IBM MQ” na página 37.

Para obter informações sobre os objetos padrão criados em um gerenciador de filas, consulte “Objetos Padrão do Sistema” na página 16.

Para obter informações sobre os diferentes tipos de objetos IBM MQ , consulte o seguinte:

Objetos das Informações sobre Autenticação

O objeto de informações sobre autenticação fornece as definições necessárias para executar a verificação de revogação de certificado.

O objeto de informações sobre autenticação do gerenciador de filas faz parte do suporte do IBM MQ para Segurança da Camada de Transporte (TLS). Ele fornece as definições necessárias para verificar os certificados revogados. As Autoridades de Certificação revogam os certificados que não podem mais ser confiáveis.

É possível usar o comando MQSC **DEFINE AUTHINFO** para definir um objeto de informações sobre autenticação. Para obter mais informações sobre os atributos de objetos de informação de autenticação, consulte **DEFINE AUTHINFO**

É possível usar os comandos de controle do IBM MQ a seguir com um objeto de informações sobre autenticação:

- **setmqaut** (conceder ou revogar autoridade)
- **dspmqaut** (exibir autorização de objeto)
- **dmpmqaut** (fazer dump de autorizações)
- **rcrmqobj** (recriar objeto)
- **rcdmqimg** (registrar imagem de mídia)
- **dspmqfls** (exibir nomes de arquivos)

Para obter uma visão geral de TLS e o uso dos objetos de informações sobre autenticação, consulte Conceitos de Segurança da Camada de Transporte (TLS) e Protocolos de segurança TLS no IBM MQ.

Canais

Os canais são objetos que fornecem um caminho de comunicação de um gerenciador de filas para outro.

Consulte “Canais” na página 30 para obter mais informações.

Objetos de Informações de Comunicação

O IBM MQ Multicast oferece baixa latência, alto fanout e mensagem multicast confiável. Um objeto de informações de comunicação (COMMINFO) é necessário para usar a transmissão Multicast.

Consulte “IBM MQ Multicast” na página 109 para obter mais informações.

Um objeto COMMINFO é um objeto do IBM MQ que contém os atributos associados à transmissão multicast. Para obter informações adicionais sobre estes atributos, consulte DEFINE COMMINFO. Para obter informações adicionais sobre a criação de um objeto COMMINFO, consulte Introdução ao multicast.

Listeners

Os *listeners* são processos que aceitam as solicitações de rede de outros gerenciadores de fila ou aplicativos clientes e iniciam os canais associados.

Os processos do listener podem ser iniciados usando o comando de controle **runmqlsr**

Objetos de listener são objetos do IBM MQ que permitem gerenciar o início e a parada dos processos de listener de dentro do escopo de um gerenciador de filas. Ao definir os atributos de um objeto listener, faça o seguinte:

- Configure o processo listener.
- Especifique se o processo listener inicia e para automaticamente quando o gerenciador de filas é iniciado e parado.

Importante: z/OS Os objetos de listener não são suportados no IBM MQ for z/OS. Para obter mais informações sobre como o IBM MQ for z/OS implementa o atendimento, usando o inicializador de canais, consulte “The channel initiator on z/OS” na página 168.

Listas de Nomes

Uma *lista de nomes* é um objeto do IBM MQ que contém uma lista de nomes de cluster, nomes de fila ou nomes de objeto de informações de autenticação. Em um cluster, isso pode ser usado para identificar uma lista de clusters para a qual o gerenciador de filas retém os repositórios.

Uma lista de nomes é um objeto do IBM MQ que contém uma lista de outros objetos do IBM MQ. Geralmente, as listas de nomes são usadas por aplicativos como monitores acionadores, em que são usadas para identificar um grupo de filas. A vantagem de usar uma lista de nomes é que ela é mantida independentemente dos aplicativos; ela pode ser atualizada sem parar qualquer um dos aplicativos que a usam. Além disso, se um aplicativo falhar, a lista de nomes não será afetada e outros aplicativos poderão continuar usando-a.

As listas de nomes também são usadas com clusters do gerenciador de filas para manter uma lista de clusters referidos por mais de um objeto IBM MQ

É possível definir e modificar listas de nomes usando os comandos MQSC DEFINE NAMELIST e ALTER NAMELIST .

Nota: z/OS Como alternativa, no z/OS, é possível usar as operações IBM MQ for z/OS e os painéis de controle

Os programas podem usar o MQI para localizar quais filas estão incluídas nessas listas de nomes. A organização das listas de nomes é responsabilidade do editor de telas e do administrador do sistema.

Para obter uma lista de atributos de lista de nomes disponíveis para uso, consulte Atributos para listas de nomes,

Definições de Processo

Os objetos de definição de processo permitem que os aplicativos sejam iniciados sem a necessidade da intervenção do operador, definindo os atributos do aplicativo para uso pelo gerenciador de filas.

O objeto de definição de processo define um aplicativo que é iniciado em resposta a um evento acionador em um gerenciador de filas do IBM MQ. Os atributos de definição de processo incluem o ID do aplicativo, o tipo de aplicativo e os dados específicos para o aplicativo. Para obter mais informações, veja *Filas de inicialização* em "Filas usadas para propósitos específicos por IBM MQ" na página 27.

Para permitir que um aplicativo seja iniciado sem a necessidade de intervenção do operador, conforme descrito em Iniciando aplicativos IBM MQ usando acionadores, os atributos do aplicativo devem ser conhecidos para o gerenciador de filas. Esses atributos são definidos em um *objeto de definição de processo*.

O atributo **ProcessName** é corrigido quando o objeto é criado. No entanto, é possível mudar outros atributos usando os comandos do IBM MQ.

Nota: z/OS Como alternativa, em z/OSé possível usar as operações IBM MQ for z/OS e os painéis de controle.

É possível consultar sobre os valores de todos os atributos usando MQINQ - Consultar atributos do objeto.

Para obter uma lista de atributos de definição de processo disponíveis para uso, consulte Atributos para definições de processo

Filas

Uma IBM MQ *fila* é um objeto nomeado no qual os aplicativos podem colocar mensagens e por meio do qual os aplicativos podem obter mensagens.

Consulte "Filas" na página 20 para obter mais informações.

Gerenciadores de filas

IBM MQ O gerenciador de filas fornece serviços de enfileiramento aos aplicativos e gerencia as filas que pertencem a eles.

Consulte "Gerenciadores de filas" na página 29 para obter mais informações.

Serviços

Os objetos de *serviço* são uma maneira de definir os programas para que sejam executados quando um gerenciador de filas é iniciado ou parado.

Os programas podem ser de um dos tipos a seguir:

Servidores

Um *servidor* é um objeto de serviço que possui o parâmetro SERVTYPE especificado como SERVER. Um objeto de serviço do servidor é a definição de um programa que será executado quando um gerenciador de filas especificado for iniciado. Apenas uma instância de um processo do servidor pode ser executada simultaneamente. Durante a execução, o status de um processo do servidor pode ser monitorado usando o comando MQSC, DISPLAY SVSTATUS. Geralmente os objetos de serviço do servidor são definições de programas como manipuladores de devolução ou monitores acionadores; no entanto, os programas que podem ser executados não são limitados aos fornecidos com IBM MQ. Além disso, um objeto de serviço do servidor pode ser definido para incluir um comando que será executado quando o gerenciador de filas especificado for encerrado para finalizar o programa.

Comandos

Um *comando* é um objeto de serviço que possui o parâmetro SERVTYPE especificado como COMMAND. Um objeto de serviço do comando é a definição de um programa que será executado quando um gerenciador de filas especificado for iniciado ou parado. Diversas instâncias de um processo de comando podem ser executadas simultaneamente. Os objetos de serviço de comando são diferentes dos objetos de serviço do servidor porque depois que o programa é executado o gerenciador de filas não monitorará o programa. Geralmente, os objetos de serviço de comando são definições de programas que são de curta duração e executarão uma tarefa específica como iniciar uma ou mais tarefas diferentes.

Importante: z/OS Os objetos de serviço não são suportados no IBM MQ for z/OS.

Para obter mais informações, consulte Trabalhando com serviços..

Classes de Armazenamento

z/OS

Uma classe de armazenamento mapeia uma ou mais filas para um conjunto de páginas.

Isso significa que as mensagens para essa fila são armazenadas (sujeito ao buffer) nesse conjunto de páginas.

As classes de armazenamento são suportadas somente no IBM MQ for z/OS.

Para obter mais informações sobre as classes de armazenamento, consulte Planejando seu ambiente do IBM MQ no z/OS

Objetos Topic

Um *objeto do tópico* é um objeto do IBM MQ que permite que você a designar atributos específicos não padrão aos tópicos.

Um *tópico* é definido por um aplicativo publicando ou assinando uma *sequência de tópicos* específica. Uma sequência de tópicos pode especificar uma hierarquia de tópicos, separando-os com um caractere de barra (/). Isso pode ser visualizado por uma *árvore de tópicos*. Por exemplo, se um aplicativo publica nas sequências de tópicos `/Sport/American Football` e `/Sport/Soccer`, uma árvore de tópicos é criada que possui um nó pai `Sport` com dois filhos, `American Football` e `Soccer`.

Os tópico herdam seus atributos do primeiro nó administrativo pai localizado em suas árvores de tópicos. Se não houver nenhum nó de tópico administrativo em uma determinada árvore de tópicos, todos os tópicos herdarão seus atributos do objeto do tópico base, SYSTEM.BASE.TOPIC.

É possível criar um objeto do tópico em qualquer nó em uma árvore de tópicos especificando a sequência de tópicos do nó no atributo TOPICSTR do objeto de tópico. Também é possível definir outros atributos para o nó de tópico administrativo. Para obter mais informações sobre estes atributos, veja Os comandos MQSC ou Automatizando a administração usando comandos PCF. Cada objeto do tópico, por padrão, herda seus atributos de seu nó de tópico administrativo pai mais próximo.

Os objetos do tópico também podem ser usados para ocultar a árvore de tópicos integral dos desenvolvedores de aplicativos. Se um objeto de tópico denominado `F00TBALL.US` for criado para o

tópico `/Sport/American Football`, um aplicativo poderá publicar ou assinar o objeto denominado `FOOTBALL.US` em vez da cadeia `/Sport/American Football` com o mesmo resultado.

Se você inserir um caractere #, +, / ou * dentro de uma sequência de tópicos em um objeto do tópico, o caractere é tratado como um caractere normal na sequência e será considerado parte da sequência de tópicos associada a um objeto de tópico.

Para obter mais informações sobre objetos do tópico, consulte "Sistema de Mensagens de Publicação/Assinatura" na página 64.

Conceitos relacionados

"Introdução ao Enfileiramento de Mensagens" na página 5
Os produtos IBM MQ permitem que os programas se comuniquem entre si em uma rede de componentes diferentes (processadores, sistemas operacionais, subsistemas e protocolos de comunicação) usando uma interface de programação de aplicativos consistente.

Referências relacionadas

Os Comandos MQSC

Filas

Introdução às filas e atributos de filas do IBM MQ.

As mensagens são armazenadas em uma fila, para que se o aplicativo de colocação estiver esperando uma resposta para sua mensagem, ele fique livre para executar outro trabalho enquanto aguarda pela resposta. Os aplicativos acessam uma fila usando a Message Queue Interface (MQI), descrita em Visão geral do Message Queue Interface.

Antes que uma mensagem ser colocada em uma fila, ela já deve ter sido criada. Uma fila é de propriedade de um gerenciador de filas e esse gerenciador de filas pode ter várias filas. No entanto, cada fila deve ter um nome que seja exclusivo nesse gerenciador de filas.

Uma fila é mantida por meio de um gerenciador de filas. Na maioria dos casos, cada fila é fisicamente gerenciada por seu gerenciador de filas, mas isso não é aparente para um programa de aplicativo. As filas compartilhadas do IBM MQ for z/OS podem ser gerenciadas por qualquer gerenciador de filas no grupo de filas compartilhadas.

Para criar uma fila, você pode utilizar os comandos do IBM MQ (MQSC), comandos PCF ou interfaces específicas de plataforma. Por exemplo, os painéis de operações e de controle do IBM MQ for z/OS são específicos da plataforma.

É possível criar filas locais para tarefas temporárias *dinamicamente* a partir do seu aplicativo. Por exemplo, é possível criar filas de *resposta* (que não são necessárias após o término de um aplicativo). Para obter mais informações, consulte "Filas Dinâmicas e de Modelo" na página 25.

Antes de usar uma fila, você deve abri-la, especificando o que deseja fazer com ela. Por exemplo, é possível abrir uma fila para:

- Procurar apenas as mensagens (sem recuperá-las)
- Recuperar as mensagens (e compartilhando o acesso com outros programas ou com acesso exclusivo)
- Colocar as mensagens na fila
- Pergunta sobre os atributos da fila
- Configurar os atributos da fila

Para obter uma lista completa das opções que podem ser especificadas quando você abre uma fila, consulte MQOPEN - Objeto de Abertura.

Atributos das Filas

Alguns dos atributos de uma fila são especificados quando a fila é definida e não podem ser mudados mais tarde (por exemplo, o tipo da fila). Outros atributos de filas podem ser agrupados naqueles que podem ser mudados:

- Pelo gerenciador de filas durante o processamento da fila (por exemplo, a profundidade atual de uma fila)
- Apenas por comandos (por exemplo, a descrição de texto da fila)
- Por aplicativos, usando a chamada MQSET (por exemplo, se as operações de entrada forem permitidas na fila)

É possível localizar os valores de todos os atributos usando a chamada MQINQ.

Os atributos que são comuns a mais de um tipo de fila são:

QName
Nome da fila.

QType
Tipo da fila.

QDesc
A descrição de texto da fila.

InhibitGet
Se os programas têm permissão para obter mensagens da fila. No entanto, nunca é possível obter mensagens de filas remotas.

InhibitPut
Se os programas têm permissão para colocar mensagens na fila.

DefPriority
Prioridade padrão para as mensagens colocadas na fila;

DefPersistence
Persistência padrão para as mensagens colocadas na fila

Escopo
Controla se uma entrada para esta fila também existe em um serviço de nomes.

z/OS O atributo **Scope** não é suportado no z/OS

Para obter uma descrição completa destes atributos, consulte Atributos para Filas.

Maneiras de definir filas

É possível definir filas para o IBM MQ usando o comando MQSC DEFINE ou o comando PCF Criar fila. Os comandos especificam o tipo de fila e seus atributos. Por exemplo, um objeto de fila local possui atributos que especificam o que acontece quando os aplicativos fazem referência a essa fila nas chamadas MQI. Os exemplos de atributos são:

- Indica se os aplicativos podem recuperar as mensagens da fila (GET ativado)
- Indica se os aplicativos podem colocar as mensagens na fila (PUT ativado)
- Indica se o acesso à fila é exclusivo a um aplicativo ou compartilhado entre os aplicativos
- O número máximo de mensagens que podem ser armazenadas na fila ao mesmo tempo (profundidade máxima da fila)
- O comprimento máximo de mensagens que podem ser colocadas na fila

Há também várias interfaces específicas de plataforma que é possível usar para definir filas.

Conceitos relacionados

“Filas de Clusters” na página 60
Uma fila de clusters é uma fila que é hospedada por um gerenciador de filas do cluster e disponibilizada para outros gerenciadores de filas no cluster.

“Filas de Devoluções” na página 51
A fila de devoluções (ou fila de mensagens não entregues) é a fila para a qual as mensagens são enviadas se não puderem ser roteadas para seus destinos corretos. Cada gerenciador de filas geralmente possui uma fila de devoluções.

Automatizando a administração usando comandos PCF

IBM MQ Console: Trabalhando com filas

Tarefas relacionadas

Administrando o IBM MQ usando comandos MQSC

Criando e configurando gerenciadores de filas e objetos com o MQ Explorer

IBM i Gerenciando IBM MQ for IBM i usando os comandos de CL

z/OS Origens das quais é possível emitir comandos MQSC e PCF no IBM MQ for z/OS

Referências relacionadas

“Comparison between shared queues and cluster queues” na página 60

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

Informações relacionadas

“What is a shared queue?” na página 172

Filas locais

Filas de transmissão, inicialização, devoluções, comandos, padrões, canais e eventos são tipos de filas locais.

Uma fila é conhecida em um programa como *local* se for de propriedade do gerenciador de filas ao qual o programa está conectado. É possível obter mensagens de filas locais e colocar as mensagens nelas.

O objeto de definição de fila contém as informações de definição da fila, bem como as mensagens físicas colocadas na fila.

Cada gerenciador de filas pode ter algumas filas locais que usa para propósitos especiais:

Filas de transmissão

Quando um aplicativo envia uma mensagem para uma fila remota, o gerenciador de filas locais armazena a mensagem em uma fila local especial, denominada *fila de transmissão*. Os aplicativos podem colocar mensagens diretamente em uma fila de transmissão ou indiretamente por meio de uma definição de fila remota.

Quando um gerenciador de filas envia mensagens para um gerenciador de filas remotas, ele identifica a fila de transmissão utilizando a seguinte sequência:

1. A fila de transmissão nomeada no atributo XMITQ da definição local de uma fila remota.
2. Uma fila de transmissão com o mesmo nome que o gerenciador de filas remotas. Esse valor é o valor padrão no XMITQ da definição local de uma fila remota.
3. A fila de transmissão nomeada no atributo DEFXMITQ do gerenciador de fila local.

Um *agente do canal de mensagens* é um programa do canal associado à fila de transmissão e ele entrega a mensagem ao seu próximo destino. O próximo destino é o gerenciador de filas ao qual o canal de mensagens está conectado. Ele não é necessariamente o mesmo gerenciador de filas que o destino final da mensagem. Quando a mensagem é entregue para seu próximo destino, ela é excluída da fila de transmissão. A mensagem pode precisar passar por muitos gerenciadores de filas em sua jornada até seu destino final. Você deve definir uma fila de transmissão em cada gerenciador de filas ao longo da rota, cada uma contendo mensagens aguardando para serem transmitidas para o próximo destino. Uma fila de transmissão normal contém mensagens para o próximo destino, embora as mensagens possam ter diferentes destinos eventuais. Uma fila de transmissão do cluster contém mensagens para diversos destinos. O `correlID` de cada mensagem identifica o canal no qual a mensagem é colocada para transferi-la para seu próximo destino.

É possível definir várias filas de transmissão em um gerenciador de filas. Você pode definir várias filas de transmissão para o mesmo destino, com cada um sendo usado para uma classe de serviço diferente. Por exemplo, talvez você deseje criar filas de transmissão diferentes para mensagens pequenas e mensagens grandes indo para o mesmo destino. É possível, então, transferir as mensagens usando diferentes canais de mensagens, para que as mensagens grandes não atrapalhem

as mensagens menores. Todas as mensagens nas filas de clusters ou em tópicos de cluster são colocadas em uma única fila de transmissão do cluster SYSTEM.CLUSTER.TRANSMIT.QUEUE, por padrão. Como uma opção, é possível alterar o padrão e separar o tráfego de mensagens que vai para diferentes gerenciadores de filas de clusters para diferentes filas de transmissão do cluster. Se você configurar o atributo do gerenciador de filas DEFCLXQ para CHANNEL, cada canal do emissor de clusters criará uma fila de transmissão do cluster separada. Como uma alternativa, é possível definir manualmente as filas de transmissão do cluster para serem usadas por canais do emissor de clusters.

As filas de transmissão podem acionar um agente do canal de mensagens para enviar mensagens para a frente; consulte Iniciando aplicativos IBM MQ usando acionadores.

z/OS No IBM MQ for z/OS, se você estiver usando enfileiramento dentro do grupo, a fila de transmissão será atendida por um *agente de enfileiramento dentro do grupo*. Uma fila de transmissão compartilhada é usada ao usar o enfileiramento entre grupos no IBM MQ for z/OS.

Filas de inicialização

Uma *fila de inicialização* é uma fila local na qual o gerenciador de filas coloca uma mensagem do acionador quando um evento acionador ocorre em uma fila do aplicativo.

Um evento acionador é um evento destinado a fazer com que um programa inicie o processamento de uma fila. Por exemplo, um evento pode ter mais de 10 mensagens chegando. Para obter mais informações sobre como o acionamento funciona, consulte Iniciando aplicativos IBM MQ usando acionadores.

Fila de Devoluções (mensagem não entregue)

Uma *fila de devoluções (mensagem não entregue)* é uma fila local na qual o gerenciador de filas coloca as mensagens que não pode entregar.

Quando o gerenciador de filas coloca uma mensagem no fila de devoluções, inclui um cabeçalho na mensagem. As informações do cabeçalho incluem a razão pela qual o gerenciador de filas coloca a mensagem na fila de devoluções. Elas também contêm o destino da mensagem original, a data e o horário em que o gerenciador de filas colocou a mensagem na fila de devoluções.

Os aplicativos também podem usar a fila para as mensagens que não podem ser entregues. Para obter informações adicionais, consulte Usando a fila de devolução (mensagem não entregue).

Fila de Comando do Sistema

A *fila de comando do sistema* é uma fila para a qual os aplicativos devidamente autorizados podem enviar os comandos do IBM MQ . Essas filas recebem os comandos PCF, MQSC e CL, conforme suportado em sua plataforma, na prontidão para o gerenciador de filas para agir neles.

z/OS Em IBM MQ for z/OS a fila é chamada SYSTEM.COMMAND.INPUT ; em outras plataformas é chamado de SYSTEM.ADMIN.COMMAND.QUEUE. Os comandos aceitos variam por plataforma. Consulte Referência de formatos de comando programáveis para obter detalhes.

Filas Padrão do Sistema

As *filas padrão do sistema* contêm as definições iniciais das filas para o seu sistema. Quando você cria uma definição de fila, o gerenciador de filas copia a definição da fila padrão do sistema apropriada. A criação de uma definição de fila é diferente de criar uma fila dinâmica. A definição da fila dinâmica é baseada na fila modelo escolhida como o modelo para a fila dinâmica.

Filas de Eventos

As *filas de eventos* retêm as mensagens do evento. Essas mensagens são relatadas pelo gerenciador de filas ou por um canal.

Filas Remotas

Para um programa, uma fila é *remota* caso seja de propriedade de um gerenciador de filas diferente daquele ao qual o programa está conectado.

Quando um link de comunicação tiver sido estabelecido, um programa pode enviar uma mensagem a uma fila remota. Um programa nunca pode receber uma mensagem de uma fila remota.

O objeto de definição de fila, criado quando você define uma fila remota, retém apenas as informações necessárias para que o gerenciador de filas locais localize a fila à qual deseja que a sua mensagem acesse. Este objeto é conhecido como a *definição local de uma fila remota*. Todos os atributos da fila remota são retidos pelo gerenciador de filas que o possui, porque é uma fila local para esse gerenciador de filas.

Ao abrir uma fila remota, para identificar a fila, deve-se especificar um dos seguintes:

- O nome da definição local que define a fila remota. Do ponto de vista de um aplicativo, isso é igual a abri uma fila local. Um aplicativo não precisa saber se uma fila é local ou remota.

  Para criar uma definição local de uma fila remota em todas as plataformas exceto IBM i, use o comando DEFINE QREMOTE .

  IBM i No IBM i, use o comando CRTMQMQ.
- O nome do gerenciador de filas remotas e o nome da fila como é conhecido nesse gerenciador de filas remotas.

As definições locais das filas remotas possuem três atributos além dos atributos comuns descritos em “Atributos das Filas” na página 20. Esses três atributos são:

RemoteQName
O nome pelo qual o gerenciador de filas proprietário da fila a conhece.

RemoteQMgrName
O nome do gerenciador de filas proprietário.

XmitQName
O nome da fila de transmissão local que é usada ao encaminhar mensagens para outros gerenciadores de filas.

Para obter mais informações sobre esses atributos, veja Atributos para filas.

Se você usar a chamada MQINQ na definição local de uma fila remota, o gerenciador de filas retornará os atributos da definição local apenas, que é o nome da fila remota, o nome do gerenciador de filas remotas e o nome da fila de transmissão, não os atributos da fila local correspondente no sistema remoto.

Consulte também Filas de Transmissão.

Filas de Alias

Uma *fila de alias* é um objeto do IBM MQ que você pode usar para acessar outra fila ou um tópico. Isso significa que mais de um programa pode funcionar com a mesma fila, acessando-a usando diferentes nomes.

A fila que resulta da resolução de um nome de alias, conhecido como a fila base, pode ser qualquer um dos seguintes tipos de filas, conforme suportado pela plataforma:

- Uma fila local
- A definição local de uma fila remota.
- z/OS Uma fila compartilhada, que é um tipo de fila local disponível somente no IBM MQ for z/OS.
- Uma fila predefinida
- Uma fila dinâmica

Um nome alternativo também pode ser resolvido em um tópico. Se um aplicativo colocar atualmente as mensagens em uma fila, ele pode estar preparado para publicar em um tópico, tornando o nome da fila um alias para o tópico. Não é necessário nenhum código do aplicativo.

Nota: Um alias não pode resolver para outro alias diretamente no mesmo gerenciador de filas.

Um exemplo do uso das filas de alias é para que um administrador do sistema forneça diferentes autoridades de acesso ao nome da fila base (ou seja, a fila na qual o alias é resolvido) e ao nome da fila de alias. Isso significa que um programa ou usuário pode estar autorizado a usar a fila de alias, mas não a fila base.

Como alternativa, a autorização pode ser configurada para inibir as operações de entrada para o nome alternativo, mas permiti-las para a fila base.

Em alguns aplicativos, o uso de filas de alias significa que os administradores do sistema podem alterar facilmente a definição de um objeto da fila de alias sem precisar ter o aplicativo mudado.

IBM MQ faz verificações de autorização no nome alternativo quando os programas tentam usar esse nome. Ele não verifica se o programa está autorizado a acessar o nome para o qual o alias é resolvido. Portanto, um programa pode estar autorizado a acessar um nome de fila de alias, mas não o nome de fila resolvido.

Além dos atributos da fila gerais descritos em “Filas” na página 20, as filas de alias possuem um atributo **BaseQName**. Este é o nome da fila base para a qual o nome alternativo é resolvido. Para obter uma descrição mais completa deste atributo, consulte BaseQName (MQCHAR48).

Os atributos *InhibitGet* e **InhibitPut** (consulte “Filas” na página 20) das filas de alias pertencem ao nome alternativo. Por exemplo, se o nome de fila de alias ALIAS1 for resolvido para o nome de fila base BASE, as inibições no ALIAS1 afetam o ALIAS1 apenas e o BASE não é inibido. No entanto, as inibições no BASE também afetam ALIAS1.

Os atributos *DefPriority* e **DefPersistence** também pertencem ao nome alternativo. Portanto, por exemplo, você pode designar diferentes prioridades padrão a diferentes alias da mesma fila base. Além disso, é possível alterar essas prioridades sem ter que alterar os aplicativos que usam os alias.

Filas Dinâmicas e de Modelo

Essas informações fornecem um insight em filas dinâmicas, propriedades de filas dinâmicas permanentes e temporárias, usos de filas dinâmicas, algumas considerações ao uso de filas dinâmicas e filas de modelo.

Quando um programa de aplicativo emite uma chamada MQOPEN para abrir uma fila de modelo, o gerenciador de filas cria dinamicamente uma instância de uma fila local com os mesmos atributos que a fila de modelo. Dependendo do valor do campo *DefinitionType* da fila modelo, o gerenciador de filas cria uma fila dinâmica permanente ou temporária (consulte Criando filas dinâmicas).

Propriedades das Fila Dinâmicas Temporárias

As *filas dinâmicas temporárias* possuem as seguintes propriedades:

- z/OS Elas não podem ser filas compartilhadas, acessíveis por meio de gerenciadores de filas em um grupo de filas compartilhadas.

  Observe que os grupos de filas compartilhadas estão disponíveis apenas no IBM MQ for z/OS.
- Elas retém apenas as mensagens não persistentes.
- Elas são irrecuperáveis.
- Elas são excluídas quando o gerenciador de filas é iniciado.
- Elas são excluídas quando o aplicativo que emitiu a chamada MQOPEN que criou a fila fecha a fila ou termina.
  - Se houver alguma mensagem confirmada na fila, ela será excluída.
  - Se houver alguma chamada MQGET, MQPUT ou MQPUT1 não confirmada pendente na fila neste momento, a fila será marcada como sendo logicamente excluída e será excluída apenas

fisicamente (depois que essas chamadas forem confirmadas) como parte do processo de fechamento ou quando o aplicativo for terminado.

- Se a fila estiver em uso neste momento (pela criação ou outro aplicativo), a fila será marcada como sendo logicamente excluída e será excluída fisicamente apenas quando fechada pelo último aplicativo que usa a fila.
- As tentativas de acessar uma fila logicamente excluída (em vez de fechá-la) falham com o código de razão MQRC_Q_DELETED.
- MQCO_NONE, MQCO_DELETE e MQCO_DELETE_PURGE são todos tratados como MQCO_NONE quando especificado em uma chamada MQCLOSE para a chamada MQOPEN correspondente que criou a fila.

Propriedades de Filas Dinâmicas Permanente

As *filas dinâmicas permanente* possuem as seguintes propriedades:

- Elas retém mensagens persistentes ou não persistentes.
- Elas são recuperáveis no caso de falhas do sistema.
- Elas são excluídas quando um aplicativo (não necessariamente aquele que emitiu a chamada MQOPEN que criou a fila) fechar com êxito a fila usando a opção MQCO_DELETE ou MQCO_DELETE_PURGE.
  - Uma solicitação de fechamento com a opção MQCO_DELETE falhará se houver alguma mensagem (confirmada ou não confirmada) ainda na fila. Uma solicitação de fechamento com a opção MQCO_DELETE_PURGE é bem-sucedida mesmo se houver mensagens confirmadas na fila (as mensagens sendo excluídas como parte do fechamento), mas falha se houver alguma chamada MQGET, MQPUT ou MQPUT1 não confirmada pendente na fila.
  - Se a solicitação de exclusão for bem-sucedida, mas a fila tiver que ficar em uso (pela criação ou outro aplicativo), a fila será marcada como sendo logicamente excluída e será excluída fisicamente apenas quando fechada pelo último aplicativo que usa a fila.
- Elas não são excluídas se fechadas por um aplicativo que não está autorizado a excluir a fila, a menos que o aplicativo de fechamento tenha emitido a chamada MQOPEN que criou a fila. As verificações de autorização são executadas no identificador de usuários (ou identificador de usuários alternativo se MQOO_ALTERNATE_USER_AUTHORITY tiver sido especificado) que foi usado para validar a chamada MQOPEN correspondente.
- Elas podem ser excluídas da mesma maneira que uma fila normal.

Usos das filas dinâmicas

É possível usar as finas dinâmicas para:

- Aplicativos que não requerem que as filas sejam retidas depois que o aplicativo tiver sido terminado.
- Aplicativos que requerem que as respostas para as mensagens sejam processadas por outro aplicativo. Esses aplicativos podem criar dinamicamente uma fila de resposta abrindo uma fila modelo. Por exemplo, um aplicativo cliente pode:
  1. Criar uma fila dinâmica.
  2. Forneça seu nome no campo **ReplyToQ** da estrutura do descritor de mensagens da mensagem de solicitação.
  3. Coloque a solicitação em uma fila sendo processada por um servidor.

O servidor então pode colocar a mensagem de resposta na fila de resposta. Finalmente, o cliente poderia processar a resposta e fechar a fila de resposta com a opção de exclusão.

Considerações ao usar as filas dinâmicas

Considere os seguintes pontos ao usar as filas dinâmicas:

- Em um modelo cliente-servidor, cada cliente deve criar e usar sua própria fila de resposta dinâmica. Se uma fila de resposta dinâmica for compartilhada entre mais de um cliente, a exclusão da fila de

resposta pode ser atrasada porque existe uma atividade não confirmada pendente na fila ou porque a fila está em uso por outro cliente. Além disso, a fila pode ser marcada como sendo logicamente excluída e inacessível para as solicitações de API subsequente (que não seja MQCLOSE).

- Se o seu ambiente de aplicativos precisar que as filas dinâmicas sejam compartilhadas entre os aplicativos, certifique-se de que a fila seja apenas fechada (com a opção de exclusão) quando toda a atividade na fila tiver sido confirmada. Isso deve acontecer com o último usuário. Isso assegura que a exclusão da fila não seja atrasada e minimiza o período em que a fila fica inacessível porque foi marcada como sendo logicamente excluída.

Filas Modelo

Uma *fila modelo* é um modelo de uma definição de fila que você usa ao criar uma fila dinâmica.

É possível criar uma fila local dinamicamente a partir de um IBM MQ, nomeando a fila modelo que deseja usar como modelo para os atributos de fila. Nesse ponto, é possível alterar alguns atributos da nova fila. No entanto, não é possível alterar o **DefinitionType**. Se, por exemplo, você precisar de uma fila permanente, selecione uma fila modelo com o tipo de definição configurado como permanente. Alguns aplicativos de conversação podem usar as filas dinâmicas para reter suas consultas porque provavelmente não precisam manter essas filas depois que tiverem processado as respostas.

Especifique o nome de uma fila modelo no *descritor de objeto* (MQOD) de sua chamada MQOPEN. Usando os atributos da fila modelo, o gerenciador de filas cria dinamicamente uma fila local para você.

É possível especificar um nome (por inteiro) para a fila dinâmica ou raiz de um nome (por exemplo, ABC) e deixar que o gerenciador de filas inclua uma parte exclusiva nisso ou é possível permitir que o gerenciador de filas designe um nome exclusivo completo. Se o gerenciador de filas designar o nome, ele o colocará na estrutura MQOD.

Não é possível emitir uma chamada MQPUT1 diretamente para uma fila de modelo, mas é possível emitir um MQPUT1 para uma fila dinâmica que foi criada, abrindo uma fila modelo.

MQSET e MQINQ não podem ser emitidos em relação a uma fila modelo. Abrir uma fila modelo com MQOO_INQUIRE ou MQOO_SET resulta em chamadas MQINQ e MQSET subsequentes que estão sendo feitas com relação à fila criada dinamicamente.

Os atributos de uma fila modelo são um subconjunto daqueles de uma fila local. Para obter uma descrição integral, consulte Atributos para Filas.

Filas usadas para propósitos específicos por IBM MQ

O IBM MQ usa algumas filas locais para propósitos específicos relacionados à sua operação.

Você deve definir estas filas antes de o IBM MQ poder usá-las.

Filas de inicialização

As filas de inicialização são aquelas usadas no acionamento. Um gerenciador de filas coloca uma mensagem do acionador em uma fila de inicialização quando ocorrer um evento acionador. Um evento acionador é uma combinação lógica das condições detectadas por um gerenciador de filas. Por exemplo, um evento acionador pode ser gerado quando o número de mensagens em uma fila atingir uma profundidade predefinida. Este evento faz com que o gerenciador de filas coloque uma mensagem do acionador em uma fila de inicialização especificada. Esta mensagem do acionador é recuperado por um *monitor acionador*, um aplicativo especial que monitora uma fila de inicialização. O monitor acionador então inicia o programa de aplicativo que foi especificado na mensagem do acionador.

Se um gerenciador de filas tiver que usar o acionamento, pelo menos, uma fila de inicialização deve ser definida para esse gerenciador de filas. Consulte Gerenciando objetos para acionamento, runmqtrm e Iniciando aplicativos IBM MQ usando acionadores

Filas de transmissão

As filas de transmissão são filas que armazenam temporariamente as mensagens que são destinadas para um gerenciador de filas remotas. Você deve definir pelo menos uma fila de transmissão para cada gerenciador de filas remotas para a qual o gerenciador de filas locais deve enviar as mensagens

diretamente. Essas filas também são usadas na administração remota; consulte Administração remota a partir de um gerenciador de filas local. Para obter informações sobre o uso de filas de transmissão no enfileiramento distribuído, consulte Técnicas de enfileiramento distribuído do IBM MQ.

Cada gerenciador de filas pode ter uma fila de transmissão padrão. Se um gerenciador de filas que não faz parte de um cluster colocar uma mensagem em uma fila remota, a ação padrão é usar a fila de transmissão padrão. Se houver uma fila de transmissão com o mesmo nome que o gerenciador de filas de destino, a mensagem será colocada nessa fila de transmissão. Se houver uma definição de alias do gerenciador de filas, na qual o parâmetro **RQMNAME** corresponde ao gerenciador de filas de destino, e o parâmetro **XMITQ** é especificado, a mensagem é colocada na fila de transmissão nomeada por **XMITQ**. Se não houver o parâmetro **XMITQ**, a mensagem será colocada na fila local nomeada na mensagem.

Filas de Transmissão de Cluster

Cada gerenciador de filas dentro de um cluster possui uma fila de transmissão do cluster chamada SYSTEM.CLUSTER.TRANSMIT.QUEUE e uma fila de transmissão do cluster de modelo, SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. As definições destas filas são criadas por padrão quando você define um gerenciador de filas. Se o atributo do gerenciador de filas, **DEFCLXQ**, é configurado como CHANNEL, uma fila de transmissão de cluster dinâmico permanente é criada automaticamente para cada canal do emissor de clusters que é criado. As filas são chamadas de SYSTEM.CLUSTER.TRANSMIT. *ChannelName* Também é possível definir filas de transmissão do cluster manualmente.

Um gerenciador de filas que faz parte do cluster envia mensagens em uma destas filas para outros gerenciadores de filas que estão no mesmo cluster.

Durante a resolução do nome, uma fila de transmissão do cluster tem precedência sobre a fila de transmissão padrão e uma fila de transmissão do cluster específica tem precedência sobre SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Filas de Devoluções

Uma fila de devoluções (mensagens não entregues) é uma fila que armazena as mensagens que não podem ser roteadas para seus destinos corretos. Uma mensagem não pode ser roteada quando, por exemplo, a fila de destino está cheia. A fila de devoluções fornecida é chamada de SYSTEM.DEAD.LETTER.QUEUE.

Para o enfileiramento distribuído, defina uma fila de devoluções em cada gerenciador de filas envolvido.

Filas de Comando

A fila de comandos, SYSTEM.ADMIN.COMMAND.QUEUE, é uma fila local para a qual aplicativos autorizados de conformidade podem enviar comandos do MQSC para processamento. Esses comandos são então recuperados por um componente do IBM MQ denominado servidor de comandos. O servidor de comandos valida os comandos, encaminha os válidos para processamento pelo gerenciador de filas e retorna qualquer resposta para a fila de resposta apropriada.

Uma fila de comandos é criada automaticamente para cada gerenciador de filas quando este for criado.

Filas de Resposta

Quando um aplicativo envia uma mensagem de solicitação, o aplicativo que recebe a mensagem pode enviar de volta uma mensagem de resposta para o aplicativo de envio. Esta mensagem é colocada em uma fila, denominada fila de resposta, que normalmente é uma fila local para o aplicativo de envio. O nome do fila de resposta é especificado pelo aplicativo de envio como parte do descritor de mensagens.

Filas de Eventos

Os eventos de instrumentação podem ser usados para monitorar os gerenciadores de filas independentemente dos aplicativos MQI.

Quando ocorrer um evento de instrumentação, o gerenciador de filas colocará uma mensagem de evento em uma fila de eventos. Esta mensagem pode então ser lida por um aplicativo de

monitoramento, que pode informar um administrador ou iniciar alguma ação reparatória se o evento indicar um problema.

Nota: Eventos acionadores são diferentes de eventos de instrumentação. Os eventos acionadores não são causados pelas mesmas condições e não geram mensagens do evento.

Para obter informações adicionais sobre os eventos de instrumentação, consulte Eventos de instrumentação.

Gerenciadores de filas

Uma instrução para os *gerenciadores de filas* e os serviços de enfileiramento que eles fornecem aos aplicativos.

Um programa deve ter uma conexão a um gerenciador de filas antes que possa usar os serviços desse gerenciador de filas. Um programa pode estabelecer esta conexão explicitamente (usando a chamada MQCONN ou MQCONNX) ou a conexão pode ser estabelecida implicitamente (isso depende da plataforma e do ambiente no qual o programa está sendo executado).

Um gerenciador de filas do IBM MQ assegura as ações a seguir:

- Os atributos do objeto são mudados de acordo com os comandos recebidos.
- Eventos especiais como eventos acionadores ou eventos de instrumentação são gerados quando as condições apropriadas são atendidas.
- As mensagens são colocadas na fila correta, conforme solicitado pelo aplicativo que está fazendo a chamada MQPUT. O aplicativo será informado, se isso não puder ser feito e um código de razão apropriado for fornecido.

Cada fila pertence a um único gerenciador de filas e é considerado como uma *fila local* para esse gerenciador de filas. O gerenciador de filas ao qual um aplicativo está conectado é considerado o *gerenciador de filas locais* para esse aplicativo. Para o aplicativo, as filas que pertencem a seu gerenciador de filas locais são filas locais.

Uma *fila remota* é uma fila que pertence a outro gerenciador de filas. Um *gerenciador de filas remotas* é qualquer gerenciador de fila que não seja o gerenciador de filas locais. Um gerenciador de filas remotas pode existir em uma máquina remota na rede ou pode existir na mesma máquina que o gerenciador de filas locais. IBM MQ suporta diversos gerenciadores de filas na mesma máquina.

Um objeto gerenciador de filas pode ser usado em algumas chamadas MQI. Por exemplo, você pode questionar sobre os atributos do objeto gerenciador de filas usando a chamada MQI MQINQ.

Atributos dos Gerenciadores de Filas

Associado a cada gerenciador de filas está um conjunto de atributos (ou propriedades) que definem suas características.Alguns dos atributos de um gerenciador de filas são corrigidos quando é criado; é possível alterar outros usando os comandos do IBM MQ . É possível consultar os valores de todos os atributos, exceto os usados para a criptografia de Segurança da Camada de Transporte (TLS), usando a chamada MQINQ.

Os atributos corrigidos incluem:

- O nome do gerenciador de filas
- A plataforma na qual o gerenciador de filas é executado (por exemplo, Windows)
- O nível dos comandos de controle do sistema que o gerenciador de filas suporta
- A prioridade máxima que você pode designar às mensagens processadas pelo gerenciador de filas
- O nome da fila para o qual os programas podem enviar os comandos do IBM MQ
- O comprimento máximo de mensagens que o gerenciador de filas pode processar z/OS (corrigido apenas no IBM MQ for z/OS)
- Indica se o gerenciador de filas suporta a indicação de sincronização quando os programas colocam e obtêm as mensagens

Os atributos *alteráveis* incluem:

- Uma descrição do texto do gerenciador de filas
- O identificador do conjunto de caracteres que o gerenciador de filas usa para as sequências de caracteres quando processa chamadas MQI
- O intervalo de tempo que o gerenciador de filas usa para restringir o número de mensagens do acionador
- z/OS O intervalo de tempo que o gerenciador de filas usa para determinar com que freqüência as filas devem ser varridas para as mensagens expiradas (IBM MQ for z/OS apenas)
- O nome da fila de devolução do gerenciador de filas (mensagem não entregue)
- O nome da fila de transmissão padrão do gerenciador de filas
- O número máximo de manipulações abertas para qualquer conexão
- A ativação e desativação de várias categorias da geração de relatórios do evento
- O número máximo de mensagens não confirmadas em uma unidade de trabalho

Gerenciadores de Filas e Gerenciamento de Carga de Trabalho

É possível configurar um cluster dos gerenciadores de fila que possui mais de uma definição para a mesma fila (por exemplo, os gerenciadores de fila no cluster deve ser clones uns dos outros). As mensagens para uma determinada fila podem ser tratadas por qualquer gerenciador de filas que hospede uma instância da fila. Um algoritmo de gerenciamento de carga de trabalho decide qual gerenciador de filas manipula a mensagem e, portanto, difunde a carga de trabalho entre seus gerenciadores de filas; consulte O Algoritmo de Gerenciamento de Carga de Trabalho do Cluster para obter informações adicionais.

Canais

Um *canal* é um link de comunicação lógica, usado por gerenciadores de filas distribuídas, entre um IBM MQ MQI client e um servidor IBM MQ ou entre dois servidores IBM MQ.

Os canais são usados para mover as mensagens de um gerenciador de filas para outro e protegem os aplicativos dos protocolos de comunicações subjacentes. Os gerenciadores de filas podem existir no mesmo sistema ou em sistemas diferentes na mesma plataforma ou ainda em plataformas diferentes. As mensagens que são enviadas podem se originar de vários locais:

- Programas de aplicativo gravados pelo usuário que transferem dados de um nó para outro.
- Aplicativos de administração gravados pelo usuário que usam comandos PCF ou a MQAI.
- O IBM MQ Explorer.
- Gerenciadores de filas que enviam mensagens de evento de instrumentação para outro gerenciador de filas.
- Gerenciadores de filas que enviam comandos de administração remota para outro gerenciador de filas. Por exemplo, usando comandos MQSC ou o administrative REST API.

Um canal possui duas definições: um em cada extremidade da conexão. Para que os gerenciadores de filas se comuniquem entre si, você deve definir um objeto do canal no gerenciador de filas que deve enviar as mensagens e outro, completar no gerenciador de filas que deve recebê-las. O mesmo *nome de canal* deve ser usado em cada extremidade da conexão e o *tipo de canal* usado deve ser compatível.

Existem três categorias de canal no IBM MQ, com diferentes tipos de canal dentro dessas categorias:

- Canais de mensagens, que são unidirecionais e transferem mensagens de um gerenciador de filas para outro.
- Canais MQI, que são bidirecionais e transferem chamadas MQI de um IBM MQ MQI client para um gerenciador de filas e respostas de um gerenciador de filas para um cliente IBM MQ.

- Os canais do AMQP, que são bidirecionais e conectam um cliente do AMQP a um gerenciador de filas em uma máquina servidor. O IBM MQ usa canais do AMQP para transferir chamadas e respostas do AMQP entre os aplicativos e gerenciadores de filas do AMQP

Canais de mensagens

O propósito de um canal de mensagens é transferir as mensagens de um gerenciador de filas para outro. Os canais de mensagens não são necessários para o ambiente do servidor do cliente.

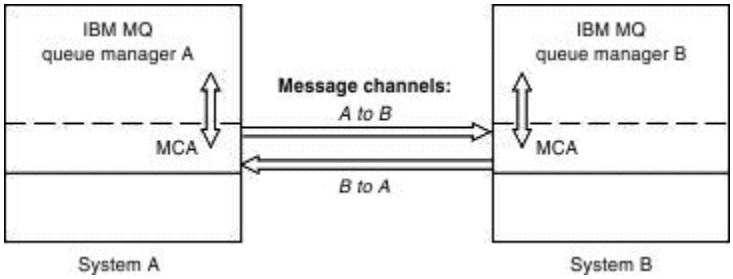


Figura 2. Canais de Mensagens entre Dois Gerenciadores de Filas

Um canal de mensagem é um link unidirecional. Se desejar remover um gerenciador de filas remotas para responder às mensagens enviadas por um gerenciador de filas locais, deve-se configurar um segundo canal para enviar as respostas de volta para o gerenciador de filas locais.

Um canal de mensagens conecta dois gerenciadores de filas usando *agentes do canal de mensagens* (MCAs). Há um agente do canal de mensagens em cada extremidade de um canal. É possível permitir que um MCA transfira mensagens utilizando diversos encadeamentos. Esse processo é conhecido como *enfileiramento*. O enfileirando permite que o MCA transfira mensagens com mais eficiência, aprimorando o desempenho do canal. Para obter mais informações sobre enfileiramento, consulte Atributos de canais.

Para obter mais informações sobre canais, consulte Chamadas de saída do canal e estruturas de dados e “Componentes de Enfileiramento Distribuído” na página 48.

Canais da MQI

Um canal Message Queue Interface (MQI) conecta um IBM MQ MQI client a um gerenciador de filas em uma máquina servidor e é estabelecido quando você emite uma chamada `MQCONN` ou `MQCONNX` de um aplicativo IBM MQ MQI client.

É um link de duas vias e é usado para a transferência de chamadas e respostas de MQI apenas, incluindo chamadas `MQPUT` que contêm dados da mensagem e chamadas `MQGET` que resultam no retorno de dados da mensagem. Há maneiras diferentes de criar e usar as definições de canal (consulte Definindo canais de MQI).

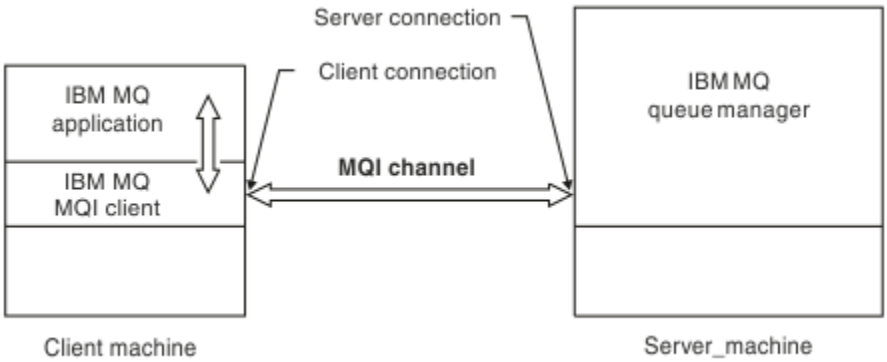


Figura 3. Conexão do Cliente e Conexão do Servidor em um Canal MQI

z/OS Um canal MQI pode ser usado para conectar um cliente a um único gerenciador de filas ou a um gerenciador de filas que faz parte de um grupo de filas compartilhadas (consulte Conectando um cliente a um grupo de filas compartilhadas).

Existem dois tipos de canais para definições de canal MQI. Eles definem o canal MQI bidirecional.

Canal de conexão do cliente
Esse tipo é para o IBM MQ MQI client.

Canal de Conexão do Servidor
Esse tipo é para o servidor executando o gerenciador de filas, com o qual o IBM MQ aplicativo, em execução em um ambiente do IBM MQ MQI client, deve se comunicar.

Canais AMQP

Multi

Existe somente um tipo de canal AMQP.

É usado um canal para conectar um aplicativo de sistema de mensagens AMQP com um gerenciador de filas, ativando o aplicativo para troca de mensagens com os aplicativos do IBM MQ. Um canal AMQP permite que você desenvolva um aplicativo usando MQ Light e, em seguida, implemente-o como um aplicativo corporativo, tirando vantagem das instalações de nível corporativo fornecidas pelo IBM MQ.

Canais de Conexão do Cliente

Canais de conexão do cliente são objetos que fornecem um caminho de comunicação de um IBM MQ MQI client para um gerenciador de filas.

Os canais de conexão do cliente são usados no enfileiramento distribuído para mover as mensagens entre um gerenciador de filas e um cliente. Eles protegem os aplicativos dos protocolos de comunicação subjacentes. O cliente pode existir em plataformas iguais ou diferentes do gerenciador de filas.

Definições de canal

Consulte “Definições de canal” na página 32 para obter as descrições de cada tipo de canal.

Conceitos relacionados

“Enfileiramento distribuído e clusters” na página 44
Enfileiramento distribuído significa enviar mensagens de um gerenciador de filas para outro. O gerenciador de filas de recebimento pode estar na mesma máquina ou em outra; perto ou do outro lado do mundo. Ele pode estar sendo executado na mesma plataforma que o gerenciador de filas locais ou pode estar em qualquer uma das plataformas suportadas pelo IBM MQ. É possível definir manualmente todas as conexões em um ambiente de enfileiramento distribuído ou você pode criar um cluster e permitir que o IBM MQ defina a maioria dos detalhes da conexão para você.

Visão geral do Message Queue Interface

Tarefas relacionadas

Administrando objetos remotos do IBM MQ

Parando canais MQI

Configurando as Conexões entre o Servidor e o Cliente

Referências relacionadas

Chamadas de Saída do Canal e Estrutura de Dados

“Comunicações” na página 36
IBM MQ MQI clients usam canais MQI para se comunicar com o servidor.

Definições de canal

Tabelas que descrevem os diferentes tipos de canais de mensagens e canais de MQI que o IBM MQ usa.

Ao referir-se aos canais de mensagens, a palavra canal é normalmente usada como um sinônimo de definição de canal. Fica claro, pelo contexto, se estamos falando de um canal completo, que tem duas extremidades, ou de uma definição de canal, que tem apenas uma extremidade.

Canais de mensagens

As definições de canal de mensagens podem ser um dos seguintes tipos:

Tipo de definição de canal de mensagens	Descrição
Emissor	Um canal emissor é um canal de mensagens que o gerenciador de filas utiliza para enviar mensagens a outros gerenciadores de filas. Para enviar mensagens utilizando um canal emissor, é necessário criar também, no outro gerenciador de filas, um canal receptor com o mesmo nome do canal emissor. É possível também utilizar canais emissores com canais solicitantes se estiver implementando um mecanismo de "retorno de chamada".
Servidor	Um canal servidor é um canal de mensagens que o gerenciador de filas utiliza para enviar mensagens a outros gerenciadores de filas. Para enviar mensagens utilizando um canal servidor, é necessário criar também, no outro gerenciador de filas, um canal receptor com o mesmo nome do canal servidor. É possível também utilizar os canais do servidor com canais de solicitante. Nesse caso, a definição de canal do solicitante na outra extremidade do canal solicita o início da definição de canal do servidor. O servidor envia mensagens para o solicitante. O servidor também pode iniciar a comunicação assim que o servidor souber o nome da conexão do canal do parceiro.
Receptor	Um canal de receptor é um canal de mensagens que o gerenciador de filas utiliza para receber mensagens de outros gerenciadores de filas. Para receber mensagens utilizando um canal receptor, é necessário criar também, no outro gerenciador de filas, um canal emissor ou servidor com o mesmo nome desse canal receptor.

Tipo de definição de canal de mensagens	Descrição
Solicitante	Um canal do Solicitante é um canal de mensagens que o gerenciador de filas usa para receber mensagens de outros gerenciadores de filas. Um canal do solicitante pode solicitar a inicialização do canal do parceiro que está definido na extremidade remota. Se o canal do parceiro for um canal do servidor, ele aceitará a solicitação de inicialização e começará a enviar mensagens da fila de transmissão identificada na definição do canal do servidor para o canal do solicitante. Se o canal do parceiro for um canal emissor, ele aceitará a solicitação de inicialização. No entanto, ele encerrará a conexão com o solicitante em seguida. Em seguida, o canal emissor é iniciado, negocia uma sessão com o canal do solicitante do parceiro e começa a enviar mensagens da fila de transmissão que é identificada na definição do canal emissor. Este último caso, essencialmente, fornece um mecanismo de retorno de chamada no qual o canal do Solicitante solicita que o canal Emissor retorne a chamada.
Emissor de cluster	Uma definição de canal emissor do cluster (CLUSSDR) define a extremidade de envio de um canal no qual um gerenciador de filas do cluster pode enviar informações sobre o cluster para um dos repositórios completos. O canal emissor do cluster é utilizado para notificar o repositório de todas as mudanças no status do gerenciador de filas, por exemplo, a inclusão ou remoção de uma fila. É usado também para transmitir mensagens. Os próprios gerenciadores de filas de repositório completo têm canais do emissor de clusters que apontam um para o outro. Eles os usam para comunicar uns aos outros alterações no status do cluster. É de menor importância para qual repositório completo uma definição de canal CLUSSDR do gerenciador de filas aponta. Depois que o contato inicial foi feito, novos objetos do gerenciador de filas do cluster são definidos automaticamente conforme requerido para que o gerenciador de filas possa enviar informações sobre o cluster a cada repositório completo e mensagens a cada gerenciador de filas.
Receptor de cluster	Uma definição de canal receptor do cluster (CLUSRCVR) define a extremidade de recebimento de um canal no qual um gerenciador de filas do cluster pode receber mensagens de outros gerenciadores de filas no cluster. Um canal receptor do cluster pode também transportar informações sobre o cluster destinadas ao repositório. Definindo o canal receptor do cluster, o gerenciador de filas indica aos outros gerenciadores de filas do cluster que ele está disponível para receber mensagens. É necessário pelo menos um canal receptor de clusters para cada gerenciador de filas do cluster.

Para cada canal, você deve definir ambas as extremidades para que haja uma definição de canal para cada extremidade do canal. As duas extremidades do canal devem ser de tipos compatíveis.

Você pode ter as seguintes combinações de definições de canal:

- Emissor-Receptor
- Servidor-Receptor
- Solicitante-Servidor
- Solicitante-Servidor (retorno de chamada)
- Cluster-emissor-Cluster-receptor

Agente do canal de mensagens

Cada definição de canal criada pertence a um determinado gerenciador de filas. Um gerenciador de filas pode ter vários canais de tipos iguais ou diferentes. Em cada extremidade do canal, há um programa, o MCA (Message Channel Agent). Em uma extremidade do canal, o MCA responsável pela chamada pega as mensagens da fila de transmissão e as envia por meio do canal. Na outra extremidade do canal, o MCA responsável pelo atendimento recebe as mensagens e as entrega ao gerenciador de filas remotas.

Um MCA responsável pela chamada pode estar associado a um canal de emissor, servidor ou solicitante. Um MCA responsável pelo atendimento pode estar associado a qualquer tipo de canal de mensagens.

O IBM MQ suporta as seguintes combinações de tipo de canal nas duas extremidades de uma conexão:

Responsável pela Chamada		**Direcionamento do Fluxo de Mensagens**	**Respondente**	
Tipo de canal	**Listener Exigido?**		**Listener Exigido?**	**Tipo de canal**
Emissor	No	Responsável pela Chamada para Responsável pelo Atendimento	Sim	Receptor
Servidor	No	Responsável pela Chamada para Responsável pelo Atendimento	Sim	Receptor
Servidor	No	Responsável pela Chamada para Responsável pelo Atendimento	Sim	Solicitante
Solicitante	No	Responsável pelo Atendimento para Responsável pela Chamada	Sim	Servidor
Solicitante	Sim	Responsável pelo Atendimento para Responsável pela Chamada	Sim	Emissor

Canais da MQI

Os canais MQI podem ser de um dos seguintes tipos:

Tipo de Canal MQI	Descrição
Conexão do servidor	Um canal de conexão do servidor é um canal MQI bidirecional que é usado para conectar um cliente do IBM MQ a um servidor do IBM MQ. O canal de conexão do servidor é a extremidade de servidor do canal.
Conexão do cliente	Um canal de conexão do cliente é um canal MQI bidirecional que é usado para conectar um cliente do IBM MQ a um servidor do IBM MQ. O IBM MQ Explorer também usa conexões do cliente para se conectar a gerenciadores de filas remotas. O canal de conexão do cliente é o final do cliente do canal. Quando você cria um canal de conexão do cliente, é criado um arquivo no computador que hospeda o gerenciador de filas. Deve-se copiar o arquivo de conexão do cliente no computador cliente do IBM MQ.

Multi ***Suporte de encadeamento múltiplo-pipelining***

Opcionalmente, é possível permitir que um agente do canal de mensagens (MCA) transfira mensagens usando vários encadeamentos. Esse processo, chamado *pipeline*, permite que o MCA transfira mensagens de forma mais eficiente, com menos estados de espera, o que melhora o desempenho do canal. Cada MCA é limitado a um máximo de dois encadeamentos

Você controla o pipeline com o parâmetro *PipeLineLength* no arquivo qm.ini . Esse parâmetro é incluído na Sub-rotina de Canais

Nota: A definição de pipeline é eficaz somente em canais TCP/IP.

Ao usar pipelining, os gerenciadores de filas em ambas as extremidades do canal devem ser configurados para ter um *PipeLineLength* maior que 1.

Considerações de saída do canal

O pipelining pode fazer com que alguns programas de saída falhem, porque:

- As saídas podem não ser chamadas em série.
- As saídas podem ser chamadas alternadamente de diferentes encadeamentos.

Verifique o design de seus programas de saída antes de usar pipelining:

- As saídas devem ser reentrantes em todos os estágios de sua execução
- Ao usar chamadas MQI, lembre-se de que não é possível usar a mesma manipulação MQI quando a saída é chamada de diferentes encadeamentos.

Considere uma saída de mensagens que abre uma fila e usa seu identificador para chamadas MQPUT em todas as chamadas subsequentes da saída Isso falha no modo de pipelining porque a saída é chamada a partir de diferentes encadeamentos. Para evitar essa falha, mantenha um identificador de fila para cada encadeamento e verifique o identificador de encadeamento toda vez que a saída for chamada

Comunicações

IBM MQ MQI clients usam canais MQI para se comunicar com o servidor.

Uma definição de canal deve ser criada em ambas as extremidades do IBM MQ MQI client e servidor da conexão. Como criar definições de canal é explicado em in Definindo canais MQI.

Os possíveis protocolos de transmissão são mostrados na seguintes tabela:

Tabela 1. Protocolos de Transmissão para Canais MQI				
Plataforma do cliente	**LU6.2**	**TCP/IP**	**NetBIOS**	**SPX**
IBM i		Sim		
Linux AIX Sistemas AIX and Linux	Sim [1]	Sim		
Windows	Sim	Sim	Sim	Sim

Nota:

1. Linux O LU 6.2 não é suportado nas plataformas a seguir:
   - Linux (plataforma POWER)
   - Linux (plataforma x86-64)
   - Linux (plataforma zSeries s390x)

Protocolos de transmissão – combinação de plataformas de IBM MQ MQI client e do servidor mostra as possíveis combinações de plataformas de IBM MQ MQI client e do servidor, usando esse protocolos de transmissão.

Um aplicativo IBM MQ em um IBM MQ MQI client pode usar todas as chamadas MQI da mesma maneira que quando o gerenciador de filas é local. **MQCONN** ou **MQCONNX** associa o aplicativo IBM MQ com o gerenciador de filas selecionado, criando um *identificador de conexões*. Outras chamadas usando esse identificador de conexões são então processadas pelo gerenciador de filas conectado. A comunicação do IBM MQ MQI client requer uma conexão ativa entre o cliente e o servidor, contrastando com a comunicação entre gerenciadores de filas, que é independente de conexão e de tempo.

O protocolo de transmissão é especificado usando a definição de canal e não afeta o aplicativo. Por exemplo, um aplicativo Windows pode conectar-se a um gerenciador de filas sobre TCP/IP e a outro gerenciador de filas sobre NetBIOS.

Considerações sobre Desempenho

O protocolo de transmissão que você usa pode afetar o desempenho do sistema do cliente e servidor IBM MQ. Em determinadas situações em que a transmissão é lenta, é possível usar a compactação de canal do IBM MQ

Nomeando os Objetos IBM MQ

A convenção de nomenclatura adotada para objetos do IBM MQ depende do objeto. o nome das máquinas e os ID de usuário que você usa com o IBM MQ também estão sujeitos a algumas restrições de nomenclatura.

Cada instância de um gerenciador de filas é conhecida por seu nome. Este nome deve ser exclusivo na rede de gerenciadores de fila interconectados, para que um gerenciador de filas possa identificar precisamente o gerenciador de filas de destino para o qual qualquer mensagem fornecida é enviada.

Para outros tipos de objeto, cada objeto possui um nome associado e pode ser referido por esse nome. Esses nomes devem ser exclusivos em um gerenciador de filas e tipo de objeto. Por exemplo, é possível ter uma fila e um processo com o mesmo nome, mas não ter duas filas com o mesmo nome.

No IBM MQ, os nomes podem ter no máximo 48 caracteres, com exceção dos *canais* que possuem um máximo de 20 caracteres. Para obter informações adicionais sobre a nomeação dos objetos IBM MQ, consulte “Regras para Nomear os Objetos IBM MQ” na página 38.

O nome das máquinas e os ID de usuário que você usa com o IBM MQ também estão sujeitos a algumas restrições de nomenclatura:

- Certifique-se de que o nome da máquina não contenha nenhum espaço. O IBM MQ não oferece suporte a nomes de máquinas que incluam espaços. Se você instalar o IBM MQ nesse tipo de máquina, não poderá criar nenhum gerenciador de filas.
- Para obter autorizações do IBM MQ, IDs de nomes de usuários e grupos não devem ter mais de 20 caracteres (espaços não são permitidos).
- Windows Um servidor do IBM MQ for Windows não suporta a conexão de um IBM MQ MQI client se o cliente estiver executando sob um ID de usuário que contém o caractere @, por exemplo, abc@d.

Conceitos relacionados

“Nomes de arquivos do IBM MQ” na página 41
Cada gerenciador de filas, fila, definição de processo, lista de nomes, canal, canal de conexão do cliente, listener, serviço e objeto de informações sobre autenticação do IBM MQ é representado por um arquivo. Como os nomes de objeto não são necessariamente nomes de arquivo válidos, o gerenciador de filas converte o nome do objeto em um nome de arquivo válido onde necessário.

Referências relacionadas

“Regras para Nomear os Objetos IBM MQ” na página 38
Os nomes de objeto IBM MQ possuem comprimentos máximos e fazem distinção entre maiúsculas e minúsculas. Nem todos os caracteres são suportados para cada tipo de objeto e vários objetos possuem regras a respeito da exclusividade de nomes.

Regras para Nomear os Objetos IBM MQ

Os nomes de objeto IBM MQ possuem comprimentos máximos e fazem distinção entre maiúsculas e minúsculas. Nem todos os caracteres são suportados para cada tipo de objeto e vários objetos possuem regras a respeito da exclusividade de nomes.

Existe vários tipos diferentes de objeto IBM MQ e todos os objetos de cada tipo possuem o mesmo nome porque existem nos espaços de nome de objeto separados: Por exemplo, um nome local e um canal emissor podem ter o mesmo nome. No entanto, um objeto não pode ter o mesmo nome que outro objeto no mesmo namespace: Por exemplo, uma fila local não pode ter o mesmo nome que uma fila de modelo e um canal emissor não pode ter o mesmo nome que um canal receptor.

Os seguintes objetos IBM MQ existem em namespaces de objeto separados:

- Informações sobre Autenticação
- Canal
- Canal do cliente
- Listener
- Lista de Nomes
- Processo
- Fila
- Serviço
- Classe de armazenamento
- Assinatura
- Tópico

Comprimento do Caractere de Nomes de Objeto

Em geral, os nomes de objetos do IBM MQ podem ter até 48 caracteres de comprimento. Essa regra se aplica aos seguintes objetos:

- Informações sobre Autenticação
- Cluster
- Listener

- Lista de Nomes
- Definição de processo
- Fila
- Gerenciador de filas
- Serviço
- Assinatura
- Tópico

Existem restrições:

1. z/OS Nos sistemas z/OS, os gerenciadores de fila devem ter, no máximo, 4 caracteres e devem estar em maiúsculas e caracteres numéricos apenas.
2. O comprimento máximo de nomes de objeto de canal e os nomes de canal de conexão do cliente é 20 caracteres. Consulte Definindo os canais para obter informações adicionais sobre canais.
3. As sequências de tópicos podem ter no máximo 10240 bytes. Todos os nomes de objeto IBM MQ fazem distinção entre maiúsculas e minúsculas.
4. Os nomes de assinatura podem ter um máximo de 10240 bytes e podem conter espaços.
5. O comprimento máximo de nomes de classe de armazenamento é de 8 caracteres.
6. O comprimento máximo de nomes de estrutura CF é de 12 caracteres.

Caracteres nos Nomes do Objeto

Os caracteres válidos para os nomes de objeto IBM MQ são:

Caracteres	Restrições
A - Z maiúscula	• Nenhum
a - z minúscula	• Em scripts do MQSC, nomes com caracteres minúsculos devem ser colocados entre aspas simples. Isso evita que caracteres minúsculos sejam convertidos em maiúsculos. • Os sistemas que usam EBCDIC Katakana não podem usar caracteres a- z minúsculos nos nomes do objeto. • z/OS Podem haver restrições ao usar caracteres minúsculos em sistemas z/OS, por exemplo, os nomes do gerenciador de filas não podem conter caracteres minúsculos. • IBM i Nos sistemas IBM i, durante o uso de comandos da CL, nomes com caracteres minúsculos devem ser colocados entre aspas simples. Isso evita que caracteres minúsculos sejam convertidos em maiúsculos.
Numéricos 0 - 9	• Nenhum
Ponto (.)	• Nenhum

Caracteres	Restrições
Sublinhado (_)	• Multi Nenhum • z/OS Evite usar os nomes com sublinhados iniciais ou finais porque eles não podem ser manipulados pelas operações IBM MQ for z/OS e pelos painéis de controle.
Barra (/)	• Windows Em sistemas Windows, o primeiro caractere em um nome do gerenciador de filas não pode ser uma barra. • IBM i Nos sistemas IBM i, durante o uso de comandos da CL, nomes contendo uma barra devem ser colocados entre aspas simples. • z/OS Nenhum
Sinal de percentual (%)	• ALW Nenhum • z/OS Se você estiver usando o RACF como o gerenciador de segurança externa para IBM MQ for z/OS, não use % em nomes de objetos porque os nomes não são incluídos nas verificações de segurança quando perfis genéricos do RACF são utilizados. • IBM i Nos sistemas IBM i, durante o uso de comandos da CL, nomes contendo um sinal de porcentagem devem ser colocados entre aspas simples.

Também há algumas regras gerais a respeito dos caracteres nos nomes do objeto:

1. Os espaços em branco integrado ou iniciais não são permitidos.
2. Os caracteres do idioma nacional não são permitidos.
3. Qualquer nome que seja inferior ao comprimento integral do campo pode ser preenchido à direita com espaços em branco. Todos os nomes abreviados que são retornados pelo gerenciador de filas são sempre preenchidos à direita com espaços em branco.

Nomes da Fila

O nome de uma fila tem duas partes:

- O nome de um gerenciador de filas
- O nome local da fila como é conhecido para esse gerenciador de filas

Cada parte do nome da fila tem 48 caracteres de comprimento.

Para consultar uma fila local, é possível omitir o nome do gerenciador de filas (substituindo-o pelos caracteres em branco ou usando um caractere nulo inicial). No entanto, todos os nomes de fila retornados a um programa pelo IBM MQ contêm o nome do gerenciador de filas.

z/OS Uma fila compartilhada, acessível a qualquer gerenciador de filas em seu grupo de filas compartilhadas, não pode ter o mesmo nome que qualquer fila local não compartilhada no mesmo grupo de filas compartilhadas. Esta restrição evita a possibilidade de um aplicativo abrindo

equivocamente uma fila compartilhada quando destinada a abrir uma fila local ou vice-versa. As filas compartilhadas e os grupos de filas compartilhadas estão disponíveis apenas no IBM MQ for z/OS.

Para consultar uma fila remota, um programa deve incluir o nome do gerenciador de filas no nome de fila integral ou deve haver uma definição local da fila remota.

Quando um aplicativo usa um nome da fila, esse nome pode ser o nome de uma fila local (ou um alias para um) ou o nome de uma definição local de uma fila remota, mas o aplicativo não precisa saber qual, a menos que precise obter uma mensagem da fila (quando a fila tiver que ser local). Quando o aplicativo abre o objeto da fila, a chamada MQOPEN executa uma função de resolução de nome para determinar em qual fila executar as operações subsequentes. O significado disso é que o aplicativo não possui dependência integrada nas filas especificas sendo definidas em locais específicos em uma rede de gerenciadores de fila. Portanto, se um administrador do sistema realocar as filas na rede e alterar suas definições, os aplicativos que usam essas filas não precisarão ser alteradas.

Nomes do Objeto Reservado

Nomes de objetos que começam com SYSTEM. são reservados para objetos definidos pelo gerenciador de filas. É possível usar os comandos **Alter**, **Define**e **Replace** para alterar essas definições de objeto para adequar sua instalação. Os nomes que são definidos para IBM MQ são listados por completo em Nomes da Fila.

z/OS No IBM MQ for z/OS, o nome da estrutura do aplicativo do recurso de acoplamento CSQSYSAPPL é reservado.

Conceitos relacionados
Nome de instalação no AIX, Linux, and Windows

Nomes de arquivos do IBM MQ

Cada gerenciador de filas, fila, definição de processo, lista de nomes, canal, canal de conexão do cliente, listener, serviço e objeto de informações sobre autenticação do IBM MQ é representado por um arquivo. Como os nomes de objeto não são necessariamente nomes de arquivo válidos, o gerenciador de filas converte o nome do objeto em um nome de arquivo válido onde necessário.

O caminho padrão para um diretório do gerenciador de filas é o seguinte:

- Um prefixo, que é definido nas informações de configuração do IBM MQ:
  - Linux AIX No AIX and Linux, o prefixo padrão é `/var/mqm`. Isso é configurado na sub-rotina `DefaultPrefix` do arquivo de configuração mqs.ini.
  - Windows Em sistemas Windows de 32 bits, o prefixo padrão é `C:\Program Files (x86)\IBM\WebSphere MQ`. Em sistemas Windows de 64 bits, o prefixo padrão é `C:\Program Files\IBM\MQ`. Para instalações de 32 e 64 bits, os diretórios de dados são instalados em `C:\ProgramData \IBM \MQ`. Isso é configurado na sub-rotina `DefaultPrefix` do arquivo de configuração mqs.ini.

  Quando disponível, o prefixo pode ser mudado usando a página de propriedades do IBM MQ no IBM MQ Explorer, caso contrário, edite o arquivo de configuração do `mqs.ini` manualmente.
- O nome do gerenciador de filas é transformado em um nome de diretório válido. Por exemplo, o gerenciador de filas:

```
queue.manager
```

  seria representando como:

```
queue!manager
```

  Este processo é conhecido como *transformação de nomes*.

No IBM MQ, é possível fornecer a um gerenciador de filas um nome contendo até 48 caracteres.

Por exemplo, você poderia nomear um gerenciador de filas:

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

No entanto, cada gerenciador de filas é representado por um arquivo e existem limitações no comprimento máximo de um nome de arquivo e nos caracteres que podem ser usados no nome. Como resultado, os nomes de arquivos que representam os objetos são automaticamente transformados para atenderem aos requisitos do sistema de arquivos.

As regras que regem a transformação de um nome do gerenciador de filas são as seguintes:

1. Transformar caracteres individuais:
   - De ... para!
   - De / para &
2. Se o nome ainda não for válido:
   a. Truncá-lo para oito caracteres
   b. Anexar um sufixo numérico de três caracteres

Por exemplo, assumindo o prefixo padrão e um gerenciador de filas com o nome `queue.manager`:

- Windows No Windows com NTFS ou FAT32, o nome do gerenciador de filas torna-se:

  ```
  C:\Program Files\IBM\MQ\qmgrs\queue!manager
  ```

- Windows No Windows com FAT, o nome do gerenciador de filas torna-se:

  ```
  C:\Program Files\IBM\MQ\qmgrs\queue!ma
  ```

- Linux AIX No AIX and Linux, o nome do gerenciador de filas torna-se:

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

O algoritmo de transformação também se distingue entre os nomes que diferem apenas no caso nos sistemas de arquivos que não fazem distinção entre maiúsculas e minúsculas.

Transformação de Nome de Objeto

Os nomes de objeto não são necessariamente nomes do sistema de arquivos válidos. Talvez você precise transformar seus nomes de objeto. O método usado é diferente daquele para nomes do gerenciador de filas porque, embora existam apenas alguns nomes do gerenciador de filas em cada máquina, existe um grande número de outros objetos para cada gerenciador de filas. Filas, definições de processo, listas de nomes, canais, canais de conexão do cliente, listeners, serviços e objetos de informações de autenticação são representados no sistema de arquivos.

Quando um novo nome for gerado pelo processo de transformação, não haverá nenhum relacionamento simples com o nome do objeto original. É possível usa o comando **dspmqfls** para converter entre nomes de objeto reais e transformados.

Referências relacionadas
dspmqfls (exibir nomes de arquivos)
Informações relacionadas
Sub-rotina AllQueueManagers do arquivo mqs.ini

IBM i Nomes de objetos no IBM i

Um gerenciador de filas tem uma biblioteca de gerenciador de filas associada que tem um nome exclusivo. Os nomes de gerenciadores de filas e os nomes de objetos podem precisar ser transformados para atenderem aos requisitos do Sistema de Arquivos Integrado (IFS) do IBM i.

Quando um gerenciador de filas é criado, o IBM MQ associa uma biblioteca do gerenciador de filas a ele. Esta biblioteca do gerenciador de filas recebe um nome exclusivo, que não ultrapassa 10 caracteres de comprimento, amplamente baseado no nome do gerenciador de filas definido pelo usuário. O gerenciador de filas e a biblioteca do gerenciador de filas são colocados em um diretório que também é baseado no nome do gerenciador de filas com o prefixo `/QIBM/UserData/mqm`. Um exemplo de um gerenciador de filas, biblioteca do gerenciador de filas e diretório é o seguinte:

Nome do gerenciador de filas	LARANJA
Nome da Biblioteca do Gerenciador de Filas	QMORANGE
Diretório	`/QIBM/UserData/mqm/ORANGE`

Todos os nomes de gerenciadores de filas e nomes de bibliotecas de gerenciador de filas são gravados em sub-rotinas no arquivo `/QIBM/UserData/mqm/mqs.ini`.

Diretórios e arquivos IFS do IBM MQ

O IBM i Integrated File System (IFS) é usado extensivamente pelo IBM MQ para armazenar os dados. Para obter informações adicionais sobre o IFS consulte *Introdução ao Sistema de Arquivos Integrado*.

Cada objeto do IBM MQ, por exemplo, um canal ou um gerenciador de filas, é representado por um arquivo. Como os nomes de objeto não são necessariamente nomes de arquivo válidos, o gerenciador de filas converte o nome do objeto em um nome de arquivo válido onde necessário.

O caminho para um diretório do gerenciador de filas é formado a partir do seguinte:

- Um prefixo, que é definido no arquivo de configuração do gerenciador de filas, `qm.ini`. O prefixo padrão é `/QIBM/UserData/mqm`.
- Um literal, `qmgrs`.
- Um nome do gerenciador de filas codificado, que é o nome do gerenciador de filas transformado em um nome do diretório válido. Por exemplo, o gerenciador de filas `queue/manager` é representado por `queue&manager`.

Este processo é referido como a transformação do nome.

Transformação de Nome do Gerenciador de Filas IFS

No IBM MQ, é possível fornecer a um gerenciador de filas um nome contendo até 48 caracteres.

Por exemplo, é possível nome um gerenciador de filas `QUEUE/MANAGER/ACCOUNTING/SERVICES`. Da mesma maneira que uma biblioteca é criada para cada gerenciador de filas, cada gerenciador de filas também é representado por um arquivo. Por causa dos pontos de código variantes em EBCDIC, existem limitações aos caracteres que podem ser usados no nome. Como resultado, os nomes de arquivos IFS que representam os objetos são automaticamente transformados para atenderem aos requisitos do sistema de arquivos.

Usando o exemplo de um gerenciador de filas com o nome `queue/manager`, transformando o caractere `/` a `&` e assumindo o prefixo padrão, o nome do gerenciador de filas no IBM MQ for IBM i se torna `/QIBM/UserData/mqm/qmgrs/queue&manager`.

Transformação de Nome de Objeto

Os nomes de objeto não são necessariamente nomes do sistema de arquivos válidos; portanto, os nomes do objeto podem precisar ser transformados. O método usado é diferente daquele para os nomes do gerenciador de filas porque, embora existam apenas alguns nomes do gerenciador de filas para cada

máquina, pode haver um grande número de outros objetos para cada gerenciador de filas. Apenas as definições de processo, filas e listas de nomes são representadas no sistema de arquivos; os canais não são afetados por essas considerações.

Quando um novo nome for gerado pelo processo de transformação, não haverá nenhum relacionamento simples com o nome do objeto original. É possível usar o comando DSPMQMOBJN para visualizar os nomes transformados para objetos do IBM MQ.

Enfileiramento distribuído e clusters

Enfileiramento distribuído significa enviar mensagens de um gerenciador de filas para outro. O gerenciador de filas de recebimento pode estar na mesma máquina ou em outra; perto ou do outro lado do mundo. Ele pode estar sendo executado na mesma plataforma que o gerenciador de filas locais ou pode estar em qualquer uma das plataformas suportadas pelo IBM MQ. É possível definir manualmente todas as conexões em um ambiente de enfileiramento distribuído ou você pode criar um cluster e permitir que o IBM MQ defina a maioria dos detalhes da conexão para você.

Enfileiramento distribuído

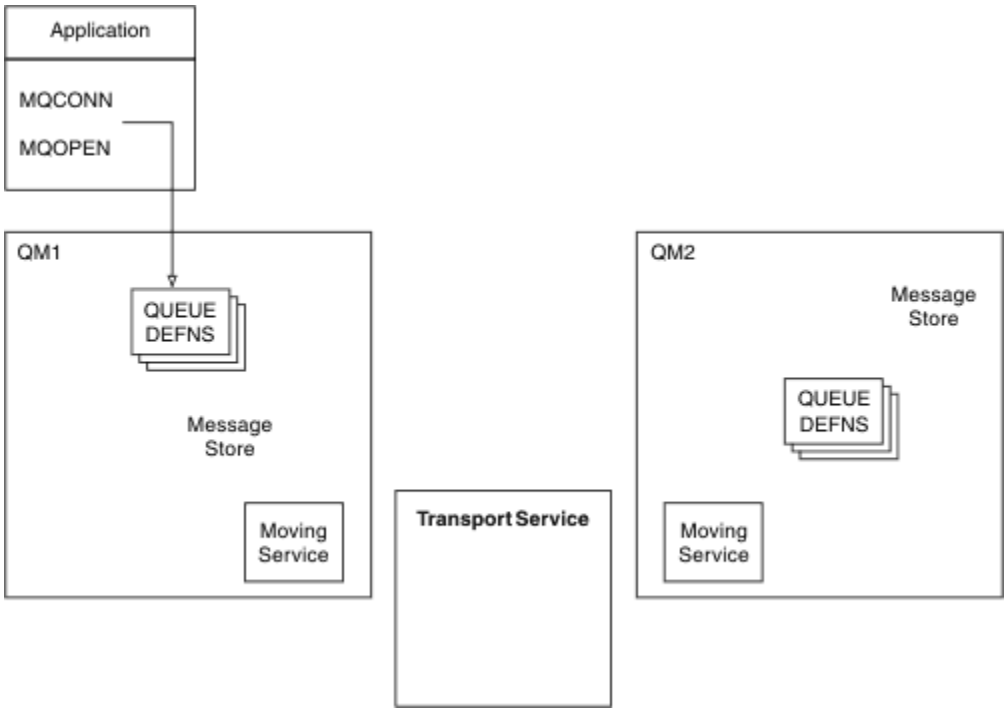


Figura 4. Visão Geral dos Componentes de Enfileiramento Distribuído

Na figura anterior:

- Um aplicativo usa a chamada MQCONN para conectar-se a um gerenciador de filas. O aplicativo usa então a chamada MQOPEN para abrir uma fila para que possa colocar mensagens na fila.
- Cada gerenciador de filas possui uma definição para cada uma de suas filas. Ele pode ter definições de *filas locais* (ou seja, hospedadas por este gerenciador de filas) e definições de *filas remotas* (ou seja, hospedadas por outros gerenciadores de filas).
- Se as mensagens forem destinadas a uma fila remota, o gerenciador de filas locais as manterá em uma *fila de transmissão*, que as persiste em um armazenamento de mensagem, até que elas possam ser encaminhadas para o gerenciador de filas remotas.
- Cada gerenciador de filas contém o software de comunicações, conhecido como o *serviço de movimentação*, que o gerenciador de filas usa para se comunicar com outros gerenciadores de filas.
- O *serviço de transporte* é independente do gerenciador de filas e pode ser qualquer um dos seguintes (dependendo da plataforma):

- Systems Network Architecture Advanced Program-to Program Communication (SNA APPC)
- TCPIP (Transmission Control Protocol/Internet Protocol)
- Sistema BIOS de Rede (NetBIOS)
- Sequenced Packet Exchange (SPX)

Componentes necessários para enviar uma mensagem

Se uma mensagem tiver que ser enviada para um gerenciador de filas remotas, o gerenciador de filas locais precisará de definições para uma *fila de transmissão* e um *canal*. Um canal é um link de comunicação unidirecional entre dois gerenciadores de fila. Ele pode transportar as mensagens destinadas para qualquer número de filas no gerenciador de filas remotas.

Cada extremidade de canal possui uma definição separada, definindo-a, por exemplo, como a extremidade de envio ou extremidade de recebimento. Um canal simples consiste em uma definição de canal *emissor* no gerenciador de filas locais e definição de canal *receptor* no gerenciador de filas remotas. Essas duas definições devem ter o mesmo nome e juntas elas constituem um canal.

O software que manipula o envio e o recebimento de mensagens é chamado *Message Channel Agent* (MCA). Há um *agente do canal de mensagens* (MCA) em cada extremidade de um canal.

Cada gerenciador de filas deve ter uma *fila de devoluções* (também conhecida como a *fila de mensagens não entregues*). As mensagens são colocadas nesta fila se não puderem ser entregues a seus destinos.

A figura a seguir mostra o relacionamento entre os gerenciadores de filas, as filas de transmissão, os canais e os MCAs:

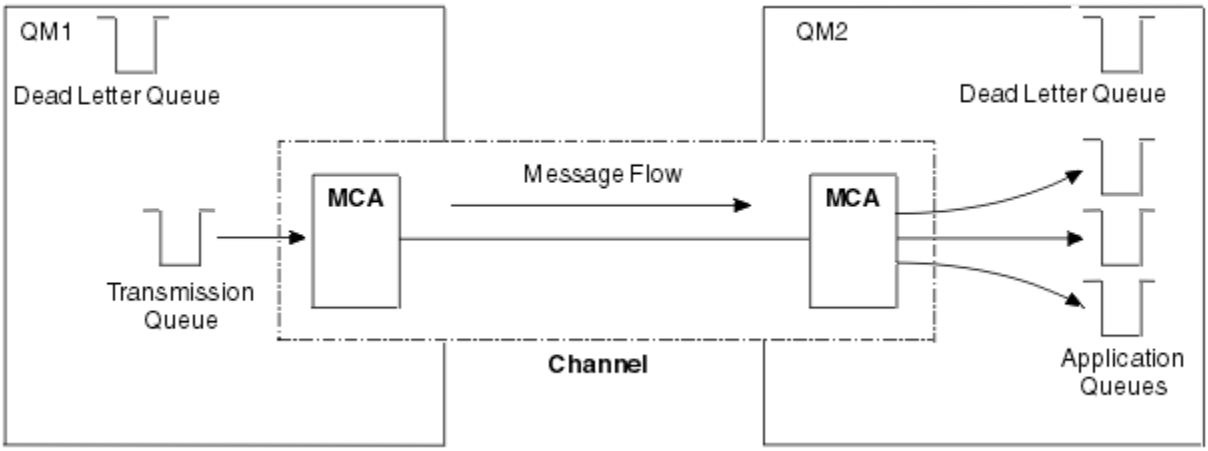


Figura 5. Enviando mensagens

Componentes necessários para retornar uma mensagem

Se o seu aplicativo requer que as mensagens sejam retornadas do gerenciador de filas remotas, é necessário definir outro canal para execução na direção oposta entre os gerenciadores de fila, conforme mostrado na figura a seguir:

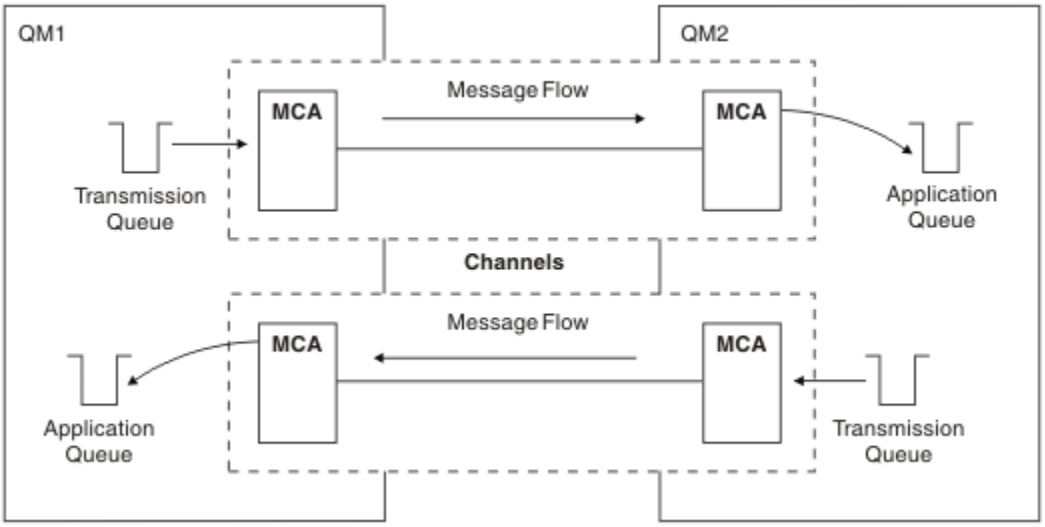


Figura 6. Enviando as Mensagens em Ambas as Direções

Clusters

Em vez de definir manualmente todas as conexões em um ambiente de enfileiramento distribuído, é possível agrupar um conjunto de gerenciadores de filas em um cluster. Ao fazer isso, os gerenciadores de filas podem disponibilizar as filas que eles hospedam para outros gerenciadores de filas no cluster, sem a necessidade de definições de canal explícitas, definições de filas remotas ou filas de transmissão para cada destino. Cada gerenciador de filas em um cluster possui uma única fila de transmissão que transmite as mensagens a qualquer outro gerenciador de filas no cluster. Para cada gerenciador de filas, você só precisa definir um canal do receptor de clusters e um canal do emissor de clusters; quaisquer canais adicionais serão automaticamente gerenciados pelo cluster.

Um cliente IBM MQ pode se conectar a um gerenciador de filas que faz parte de um cluster, assim como ele pode se conectar a qualquer outro gerenciador de filas. Como com o enfileiramento distribuído configurado manualmente, você usa a chamada MQPUT para colocar uma mensagem em uma fila em qualquer gerenciador de filas. Use a chamada MQGET para recuperar as mensagens de uma fila local.

Os gerenciadores de filas nas plataformas que suportam clusters não precisam fazer parte de um cluster. É possível continuar configurando manualmente o enfileiramento distribuído bem como ou em vez de, usar clusters.

Benefícios do uso de clusters

O armazenamento em cluster fornece dois benefícios principais:

- Os clusters simplificam a administração de redes do IBM MQ, que geralmente requerem que muitas definições de objetos para canais, filas de transmissão e filas remotas sejam configuradas. Esta situação é especialmente verdadeira em redes grandes, de mudança em potencial, nas quais muitos gerenciadores de filas precisam estar interconectados. Esta arquitetura é especialmente difícil para configurar e ativamente manter.
- Clusters podem ser usados para distribuir a carga do tráfego de mensagens em filas e gerenciadores de filas no cluster. Esta distribuição permite que a carga de mensagens de uma fila única seja distribuída entre as instâncias equivalentes dessa fila localizadas em vários gerenciadores de filas. Esta distribuição da carga de trabalho pode ser usada para atingir maior resiliência para falhas do sistema e melhorar o desempenho de ajuste de escala de fluxos de mensagens particularmente ativas em um sistema. Em tal ambiente, cada uma das instâncias das filas distribuídas possuem aplicativos consumidores processando as mensagens. Para obter informações adicionais, consulte Usando clusters para gerenciamento de carga de trabalho.

Como as mensagens são roteadas em um cluster

É possível considerar um cluster como uma rede de gerenciadores de filas mantida por um administrador de sistemas consciencioso. Sempre que definir uma fila de clusters, o administrador de sistemas cria automaticamente as definições de fila remota correspondentes conforme necessário nos outros gerenciadores de filas.

Você não precisa criar definições de fila de transmissão porque o IBM MQ fornece uma fila de transmissão em cada gerenciador de filas no cluster. Esta única fila de transmissão pode ser usada para transportar mensagens para qualquer outro gerenciador de filas no cluster. Você não está limitado ao uso de uma fila de transmissão única. Um gerenciador de filas pode usar diversas filas de transmissão para separar as mensagens que vão para cada gerenciador de filas em um cluster. Geralmente, um gerenciador de filas usa uma fila de transmissão do cluster única. É possível alterar o atributo do gerenciador de filas DEFCLXQ, de forma que um gerenciador de filas use uma fila de transmissão de cluster diferente para cada gerenciador de filas em um cluster. Também é possível definir filas de transmissão do cluster manualmente.

Todos os gerenciadores de filas que se unem a um cluster concordam em trabalhar desta maneira. Eles enviam informações sobre eles mesmos e sobre as filas que eles hospedam e recebem informações sobre os outros membros do cluster.

Para assegurar que nenhuma informação seja perdida quando um gerenciador de filas se tornar indisponível, especifique dois gerenciadores de filas no cluster para agirem como *repositórios completos*. Esses gerenciadores de filas armazenam um conjunto completo de informações sobre todos os gerenciadores de filas e filas no cluster. Todos os outros gerenciadores de filas no cluster armazenam informações somente sobre os gerenciadores de filas e filas com os quais eles trocam mensagens. Esses gerenciadores de filas são conhecidos como *repositórios parciais*. Para obter mais informações, consulte “Repositório de Cluster” na página 58.

Para se tornar parte de um cluster, um gerenciador de filas deve ter dois canais; um canal do emissor de clusters e um canal do receptor de clusters:

- Um canal do emissor de clusters é um canal de comunicação semelhante a um canal emissor. Você deve criar manualmente um canal do emissor de clusters em um gerenciador de filas para conectá-lo a um repositório completo que já é um membro do cluster.
- Um canal do receptor de clusters é um canal de comunicação semelhante a um canal receptor. Você deve criar manualmente um canal do receptor de clusters. O canal age como o mecanismo para o gerenciador de filas para receber comunicações do cluster.

Todos os outros canais que são necessários para a comunicação entre este gerenciador de filas e outros membros do cluster são então criados automaticamente.

A figura a seguir mostra os componentes de um cluster chamado CLUSTER:

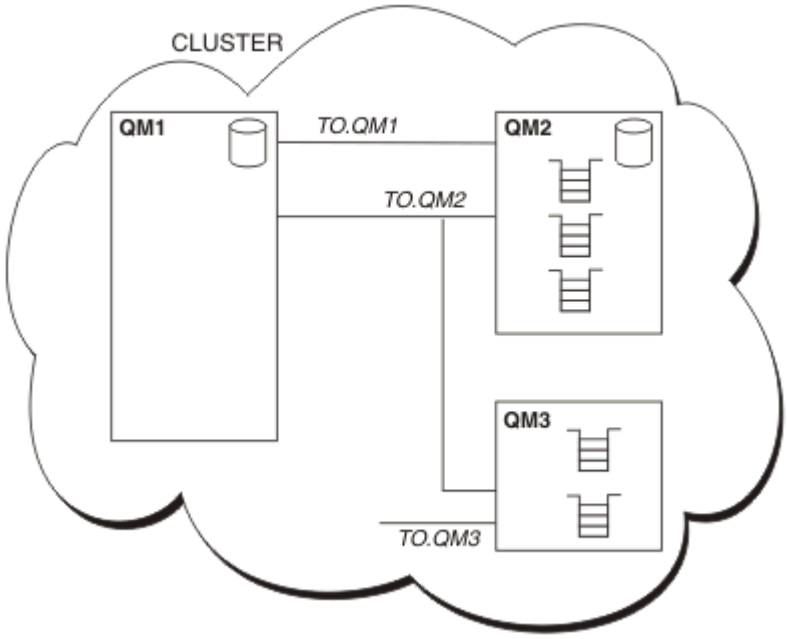


Figura 7. Um Cluster de Gerenciadores de Filas

- CLUSTER contém três gerenciadores de filas, QM1, QM2 e QM3.
- QM1 e QM2 hospedam repositórios das informações sobre os gerenciadores de filas e as filas no cluster.
- QM2 e QM3 hospedam algumas filas de cluster ou seja, filas que estão acessíveis a qualquer outro gerenciador de filas no cluster.
- Cada gerenciador de filas possui um canal do receptor de clusters denominado TO.qmgr no qual pode receber as mensagens.
- Cada gerenciador de filas também possui um canal do emissor de clusters no qual pode enviar as informações a um dos gerenciadores de fila do repositório.
- QM1 e QM3 enviam ao repositório em QM2, e QM2 envia ao repositório em QM1.

Componentes de Enfileiramento Distribuído

Os componentes de enfileiramento distribuído são canais de mensagens, agentes do canal de mensagens, filas de transmissão, inicializadores e listeners de canais e programas de saída do canal. A definição de cada extremidade de um canal de mensagens pode ser um entre vários tipos.

Canais de mensagens são os canais que transportam as mensagens de um gerenciador de filas a outro. Não confunda canais de mensagens com canais MQI. Há dois tipos de canal MQI, conexão do servidor (SVRCONN) e conexão do cliente (CLNTCONN). Para obter mais informações, consulte Canais.

A definição de cada extremidade de um canal de mensagens pode ser um dos seguintes tipos:

- Emissor (SDR)
- Receptor (RCVR)
- Servidor (SVR)
- Solicitante (RQSTR)
- Emissor de cluster (CLUSSDR)
- Receptor de cluster (CLUSRCVR)

Um canal de mensagens é definido usando um desses tipos definidos em uma extremidade e um tipo compatível na outra extremidade. As combinações possíveis são:

- Emissor-receptor
- Solicitante-servidor
- Solicitante-emissor (retorno de chamada)
- Servidor-receptor
- Emissor de cluster-receptor de cluster

Instruções detalhadas para criar um canal emissor-receptor estão incluídas em Definindo os canais. Para obter exemplos dos parâmetros necessários para configurar canais do emissor-receptor, consulte Informações de Configuração de Exemplo aplicável à sua plataforma. Para os parâmetros necessários para definir um canal de qualquer tipo, consulte DEFINE CHANNEL.

Canais Emissor-Receptor

Um emissor em um sistema inicia o canal para que possa enviar as mensagens a outro sistema. O emissor solicita que o receptor na outra extremidade do canal seja iniciado. O emissor envia as mensagens de sua fila de transmissão para o receptor. O receptor coloca as mensagens na fila de destino. Figura 8 na página 49 ilustra isso.

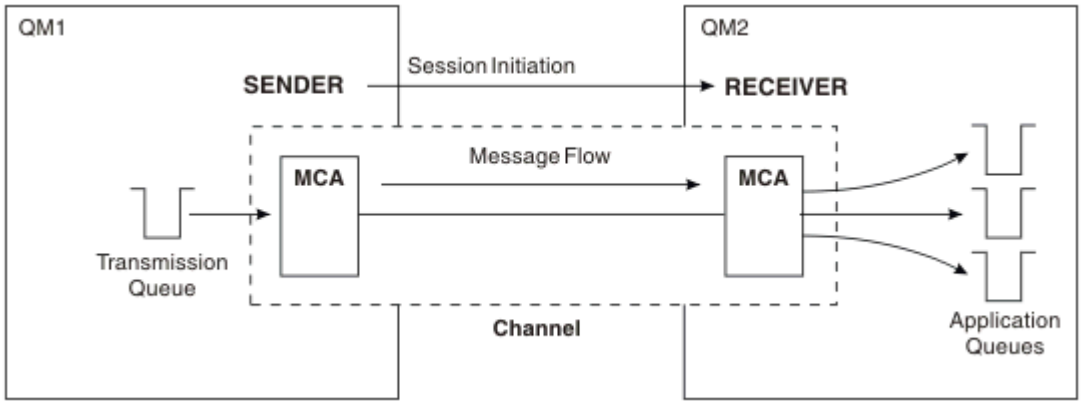


Figura 8. Um Canal Emissor-Receptor

Canais solicitante-servidor

Um solicitante em um sistema inicia o canal para que possa receber as mensagens do outro sistema. O solicitante solicita que o servidor na outra extremidade do canal seja iniciado. O servidor envia as mensagens ao solicitante da fila de transmissão definida em sua definição de canal.

Um canal do servidor também inicia a comunicação e envia as mensagens a um solicitante. Isso se aplica apenas a servidores *completos*, que sejam canais de servidor que tenham o nome de conexão do parceiro especificado na definição de canal. Um servidor completo pode ser iniciado por um solicitante ou pode iniciar uma comunicação com um solicitante.

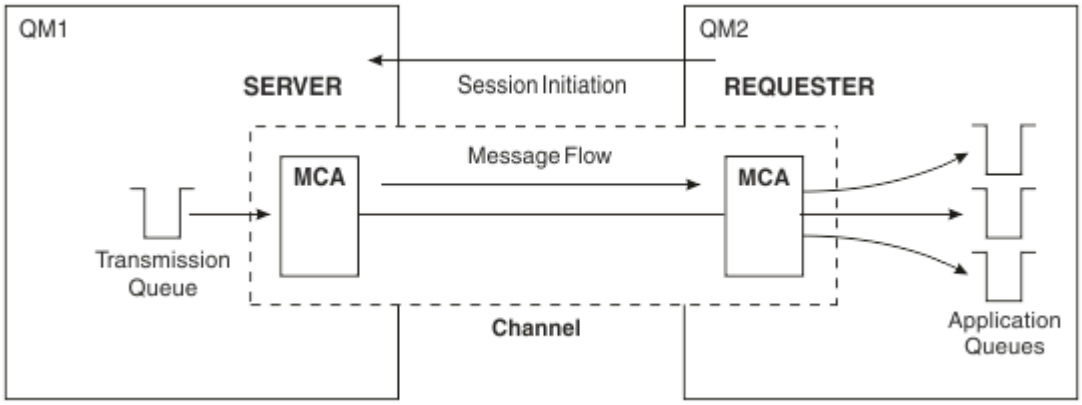


Figura 9. Um Canal Solicitante-Servidor

Canais solicitante-emissor

O solicitante inicia o canal e o emissor termina o canal. O emissor então reinicia a comunicação de acordo com as informações em sua definição de canal (conhecida como *retorno de chamada*). Ele envia as mensagens da fila de transmissão para o solicitante.

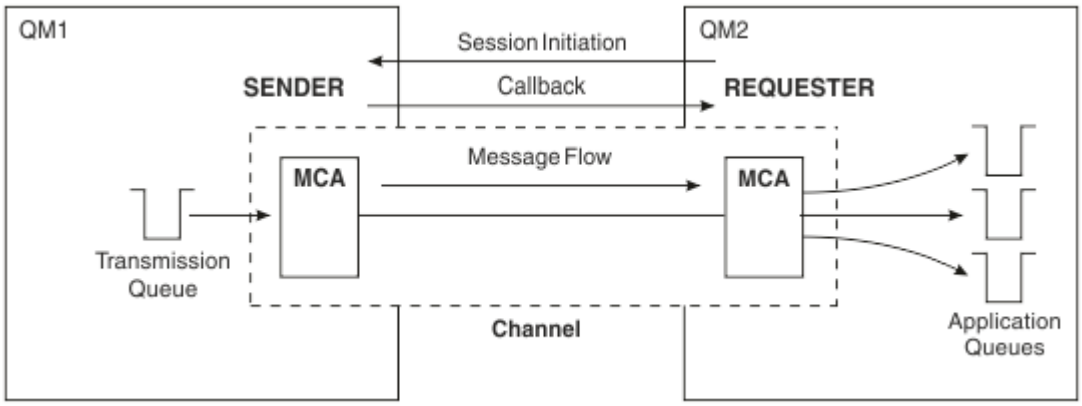


Figura 10. Um Canal Solicitante-Emissor

Canais Servidor-Receptor

Isso é como o emissor-receptor mas se aplica apenas a servidores *completos*, que sejam canais de servidor que tenham o nome de conexão do parceiro especificado na definição de canal. A inicialização do canal deve ser iniciada na extremidade do servidor do link. A ilustração disso é como a ilustração no Figura 8 na página 49.

Canais do Emissor de Clusters

Em um cluster, cada gerenciador de filas tem um canal do emissor de clusters no qual pode enviar as informações de cluster a um dos gerenciadores de filas de repositório completo. Os gerenciadores de filas também podem enviar as mensagem a outros gerenciadores de filas nos canais do emissor de clusters.

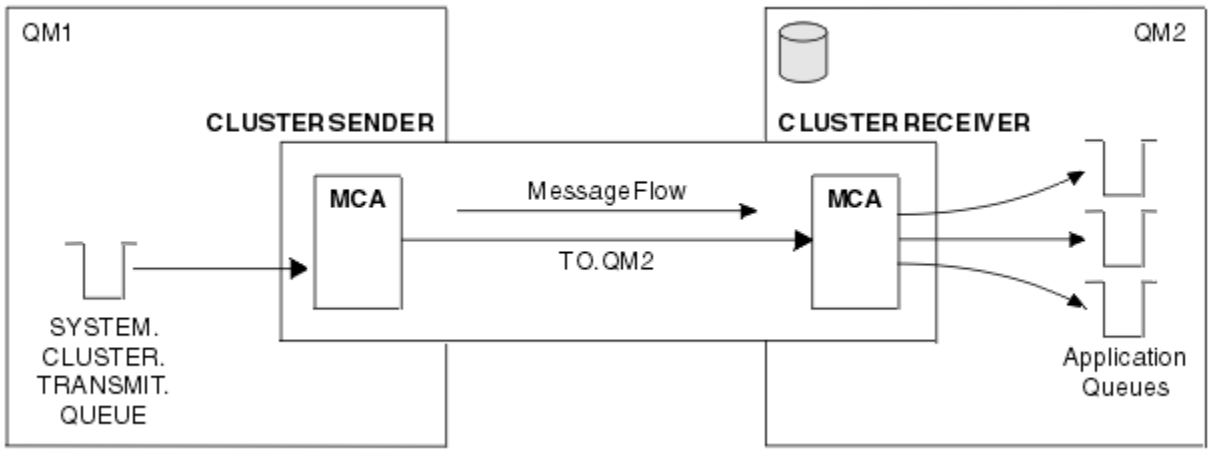


Figura 11. Um Canal do Emissor de Clusters

Canais do Receptor de Clusters

Em um cluster, cada gerenciador de filas tem um canal do receptor de clusters no qual pode receber as mensagens e informações sobre o cluster. A ilustração disso é como a ilustração no Figura 11 na página 51.

Filas de Devoluções

A fila de devoluções (ou fila de mensagens não entregues) é a fila para a qual as mensagens são enviadas se não puderem ser roteadas para seus destinos corretos. Cada gerenciador de filas geralmente possui uma fila de devoluções.

Uma *fila de devoluções* (DLQ), às vezes referida como uma *fila de mensagens não entregues*, é uma fila de contenção para mensagens que não podem ser entregues em suas filas de destino, por exemplo, porque a fila não existe ou está cheia. As filas de devoluções também são usadas na extremidade de envio de um canal, para erros de conversão de dados. Cada gerenciador de filas em uma rede geralmente tem uma fila local para ser usada como uma fila de devoluções de forma que mensagens que não puderem ser entregues ao seu destino correto possam ser armazenadas para recuperação posterior.

As mensagens podem ser colocadas no DLQ pelos gerenciadores de fila, agentes do canal de mensagem (MCAs) e aplicativos. Todas as mensagens no DLQ devem ser prefixadas com uma estrutura de *cabeçalho de devoluções*, MQDLH. O campo *Razão* da estrutura MQDLH contém um código de razão que identifica o motivo pelo qual a mensagem está no DLQ.

Geralmente é necessário definir uma fila de devoluções para cada gerenciador de filas. Se não fizer isso e o MCA não conseguir colocar uma mensagem, ela será deixada na fila de transmissão e o canal será parado. Além disso, se as mensagens rápidas e não persistentes (consulte mensagens rápidas e não persistentes) não puderem ser entregues e não existir nenhuma fila de devolução no sistema de destino, essas mensagens serão descartadas.

No entanto, usar as filas de devoluções pode afetar a sequência na qual as mensagens são entregues e assim você pode optar por não usá-las.

Tarefas relacionadas

Trabalhando com filas de mensagens não entregues

Resolução de problemas de mensagens não entregues

Referências relacionadas

runmqdlq (executar manipulador da fila de devoluções)

Definições de fila remota

As definições de fila remotas são definições para as filas que são de propriedade de outro gerenciador de filas.

Embora os aplicativos possam recuperar as mensagens apenas de filas locais, elas podem colocar as mensagens em filas locais ou filas remotas. Portanto, assim como uma definição para cada uma de suas filas locais, um gerenciador de filas pode ter *definições de fila remota*. A vantagem das definições de fila remota é que elas permitem que um aplicativo coloque uma mensagem em uma fila remota sem ter que especificar o nome da fila remota ou o gerenciador de filas remotas ou o nome da fila de transmissão. As definições de fila remota fornecem independência de local.

Existem outros usos para as definições de fila remota, que serão descritas posteriormente.

Como Chegar ao Gerenciador de Filas Remotas

Talvez nem sempre você tenha um canal entre cada gerenciador de filas de origem e de destino. Existem inúmeras outras maneiras de se vincular entre as duas, incluindo canais de compartilhamento de diversos hops, usando diferentes canais e clusters.

Diversos Hops

Se não houver nenhum link de comunicação direta entre o gerenciador de filas de origem e o gerenciador de filas de destino, é possível passar um ou mais *gerenciadores de fila intermediários* no caminho para o gerenciador de filas de destino. Isso é conhecido como *diversos hops*.

Você precisa definir os canais entre todos os gerenciadores de fila e as filas de transmissão nos gerenciadores de fila intermediários. Isso é mostrado no Figura 12 na página 52.

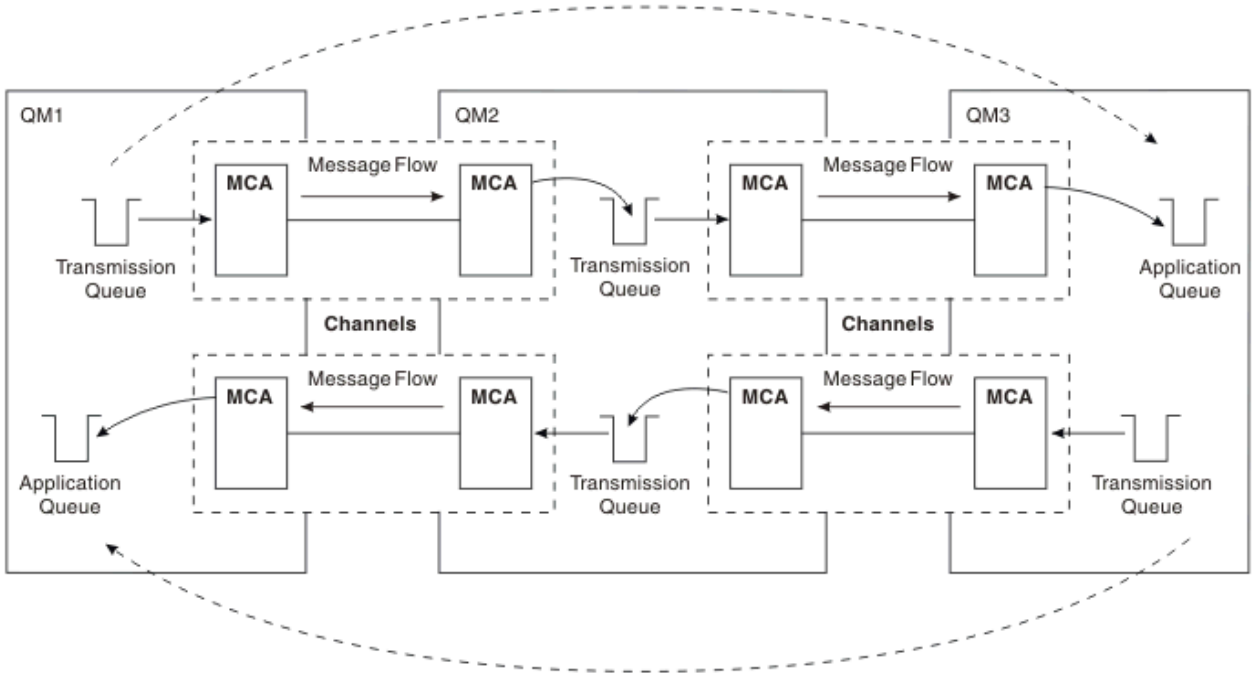


Figura 12. Passando pelos Gerenciadores de Fila Intermediários

Canais de Compartilhamento

Como um editor de telas, você tem a opção de forçar os seus aplicativos a especificar o nome do gerenciador de fila remoto juntamente com o nome da fila ou criar uma *definição de fila remota* para cada fila remota. Esta definição retém o nome do gerenciador de filas remotas, o nome da fila e o nome da fila de transmissão. De qualquer maneira, todas as mensagens de todos os aplicativos que endereçam as

filas no mesmo local remoto possuem suas mensagens enviadas por meio da mesma fila de transmissão. Isso é mostrado no Figura 13 na página 53.

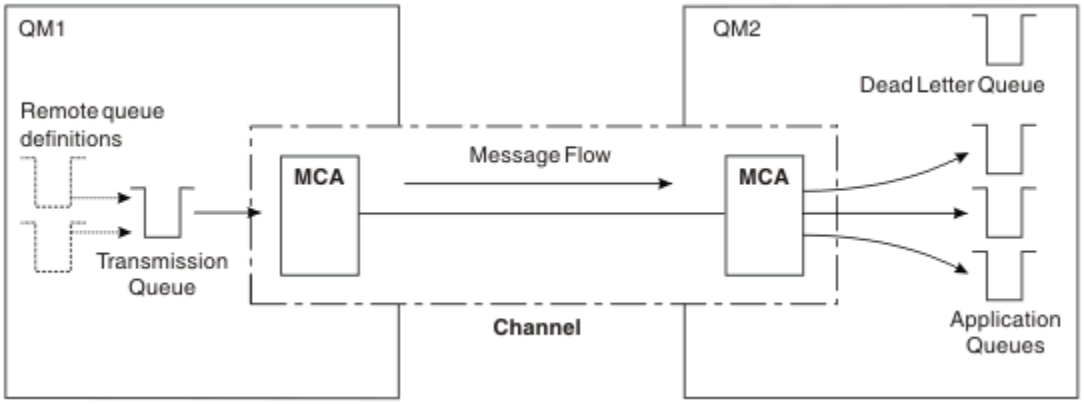


Figura 13. Compartilhando uma Fila de Transmissão

Figura 13 na página 53 ilustra quais mensagens dos diversos aplicativos para diversas filas remotas podem usar o mesmo canais.

Usando Canais Diferentes

Se você tiver mensagens de diferentes tipos a serem enviados entre dois gerenciadores de fila, poderá definir mais de um canal entre os dois. Há momento em que são necessários canais alternativos, talvez por propósitos de segurança ou para negociar a velocidade de entrega no montante absoluto do tráfego da mensagem.

Para configurar um segundo canal, você precisa definir outro canal e outra fila de transmissão e criar uma definição de fila remota que especifica o local e o nome da fila de transmissão. Seus aplicativos podem então usar o canal, mas as mensagens ainda são entregues nas mesmas filas de destino. Isso é mostrado no Figura 14 na página 53.

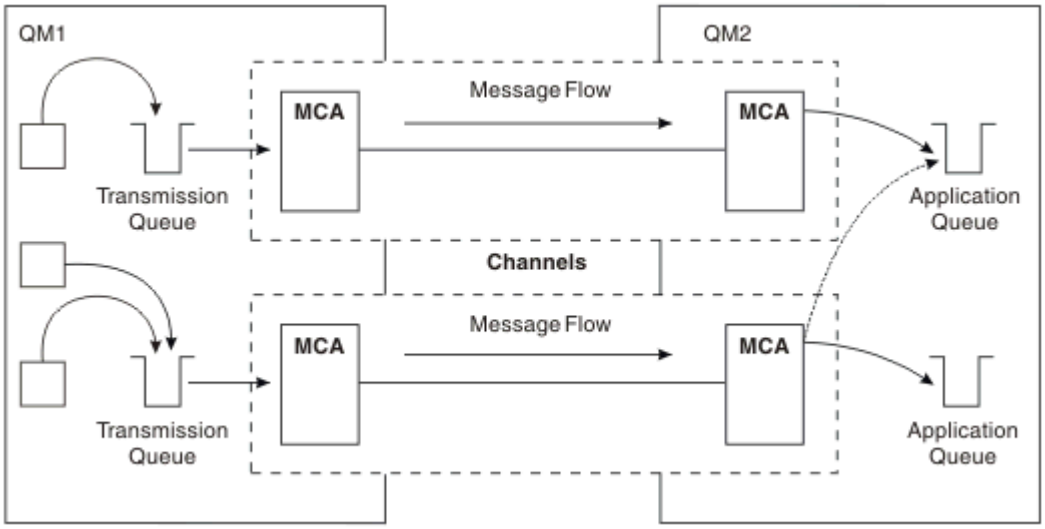


Figura 14. Usando Diversos Canais

Ao usar as definições de fila remota para especificar uma fila de transmissão, seus aplicativos **não** devem especificar o local (ou seja, o gerenciador da fila de destino) sozinhos. Se o fizerem, o gerenciador de filas não usará as definições de fila remota. As definições de fila remota fornecem independência de local. Os aplicativos podem colocar as mensagens em uma fila *lógica* sem saber onde a fila está localizada e você pode alterar a fila *física* sem ter que alterar os seus aplicativos.

Usando o Armazenamento em Cluster

Cada gerenciador de filas em um cluster define um canal do receptor de clusters. Quando outro gerenciador de filas desejar enviar uma mensagem para esse gerenciador de filas, ele definirá o canal do emissor de clusters correspondente automaticamente. Por exemplo, se houver mais de uma instância de uma fila em um cluster, o canal do emissor de clusters poderia ser definido em qualquer um dos gerenciadores de filas que hospede a fila. IBM MQ usa o algoritmo de gerenciamento de carga que usa uma rotina round-robin para selecionar um gerenciador de filas disponível para o qual rotear uma mensagem. Para obter informações adicionais, consulte Clusters.

Informações de Endereçamento

Quando um aplicativo coloca as mensagens que são destinadas para um gerenciador de filas remotas, o gerenciador de filas locais inclui um cabeçalho de transmissão nelas antes de colocá-las na fila de transmissão. Este cabeçalho contém o nome da fila de destino e o gerenciador de filas ou seja, as *informações de endereçamento*.

Em um ambiente de gerenciador de filas únicas, o endereço de uma fila de destino é estabelecido quando um aplicativo abre uma fila na qual colocar as mensagens. Como a fila de destino está no mesmo gerenciador de filas, não há necessidade de qualquer informação de endereçamento.

Em um ambiente de enfileiramento distribuído, o gerenciador de filas precisa conhecer não somente o nome da fila de destino, mas também o local dessa fila (ou seja, o nome do gerenciador de filas), e a rota para esse local remoto (ou seja, a fila de transmissão). As informações de endereçamento estão contidas no cabeçalho de transmissão. O canal de recebimento remove o cabeçalho de transmissão e usa as informações contidas nele para localizar a fila de destino.

É possível evitar a necessidade para que os seus aplicativos especifiquem o nome do gerenciador de fila de destino, se usar uma definição de fila remota. Esta definição especifica o nome da fila remota, o nome do gerenciador de filas remotas para o qual as mensagens estão destinadas e o nome da fila de transmissão usada para transportar as mensagens.

O que São Aliases?

Aliases são usados para fornecer a qualidade de serviço para as mensagens. O alias do gerenciador de filas permite que um administrador do sistema altere o nome de um gerenciador de filas de destino sem que tenha que alterar os seus aplicativos. Ele também permite que o administrador do sistema altere a rota para um gerenciador de filas de destino ou configure uma rota que envolva a passagem por inúmeros outros gerenciadores de fila (diversos hops). O alias da fila de resposta fornece a qualidade de serviço para as respostas.

Os aliases do gerenciador de filas e os aliases de fila de resposta são criados usando uma definição de fila remota que tem um RNAME em branco. Essas definições não definem as filas reais; elas são usadas pelo gerenciador de filas para resolver os nomes de fila física, os nomes do gerenciador de filas e as filas de transmissão.

As definições de alias são caracterizadas tendo um RNAME em branco.

Resolução de Nome da Fila

A resolução de nome da fila ocorre a cada gerenciador de filas sempre que uma fila é aberta. Seu propósito é identificar a fila de destino, o gerenciador de filas de destino (que pode ser local) e a rota para esse gerenciador de filas (que pode ser nulo). O nome resolvido tem três partes: o nome do gerenciador de filas, o nome da fila e, se o gerenciador de filas for remoto, a fila de transmissão.

Quando existir uma definição de fila remota, nenhuma definição de alias será referenciada. O nome da fila fornecido pelo aplicativo é resolvido no nome da fila de destino, gerenciador de filas remotas e fila de transmissão especificados na definição de fila remota. Para obter informações mais detalhadas sobre a resolução do nome da fila, consulte Resolução do Nome da Fila.

Se não houver definição de fila remota e um nome do gerenciador de filas for especificado ou resolvido pelo serviço de nomes, o gerenciador de filas verificará se existe uma definição de alias de gerenciador

de filas que corresponda ao nome do gerenciador de filas fornecido. Se houver, as informações contidas nelas serão usadas para resolver o nome do gerenciador de filas para o nome do gerenciador de filas de destino. A definição de alias do gerenciador de filas também pode ser usada para determinar a fila de transmissão para o gerenciador de filas de destino.

Se o nome da fila resolvido não for uma fila local, o nome do gerenciador de filas e o nome da fila serão incluídos no cabeçalho de transmissão de cada mensagem colocada pelo aplicativo para a fila de transmissão.

A fila de transmissão geralmente usada tem o mesmo nome que o gerenciador de filas resolvido, a menos que mudado por uma definição de fila remota ou uma definição de alias de gerenciador de filas. Se você não definiu essa fila de transmissão, mas definiu uma fila de transmissão padrão, isso será usado.

z/OS Nomes dos gerenciadores de fila sendo executados no z/OS são limitados a quatro caracteres.

Definições de Alias do Gerenciador de Filas

As definições de alias do gerenciador de filas se aplicam quando um aplicativo, que abre uma fila para colocar uma mensagem, especifica o nome da fila **e** o nome do gerenciador de filas.

As definições de alias do gerenciador de filas possuem três usos:

- Ao enviar as mensagens, remapeando o nome do gerenciador de filas
- Ao enviar as mensagens, alterando ou especificando a fila de transmissão
- Ao receber as mensagens, determinando se o gerenciador de filas locais é o destino desejado para essas mensagens

Mensagens de Saída - Remapeando o Nome do Gerenciador de Filas

As definições de alias do gerenciador de filas pode ser usado para remapear o nome do gerenciador de filas especificado em uma chamada MQOPEN. Por exemplo, uma chamada MQOPEN especifica um nome da fila igual a THISQ e um nome do gerenciador de filas igual a YOURQM. No gerenciador de filas locais, existe uma definição de alias do gerenciador de filas como o seguinte exemplo:

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

Isto mostra que o gerenciador de filas real a ser usado, quando um aplicativo coloca mensagens no gerenciador de filas YOURQM, é REALQM. Se o gerenciador de filas locais é REALQM, ele coloca as mensagens na fila THISQ, a qual é uma fila local. Se o gerenciador de filas locais não for chamado de REALQM, ele roteará a mensagem para uma fila de transmissão chamada REALQM. O gerenciador de filas altera o cabeçalho de transmissão para informar REALQM em vez de YOURQM.

Mensagens de Saída - Alterando ou Especificando a Fila de Transmissão

Figura 15 na página 56 mostra um cenário no qual as mensagens chegam no gerenciador de filas QM1 com cabeçalhos de transmissão mostrando nomes da fila no gerenciador de filas QM3. Neste cenário, QM3 é acessível por saltos múltiplos por meio do QM2.

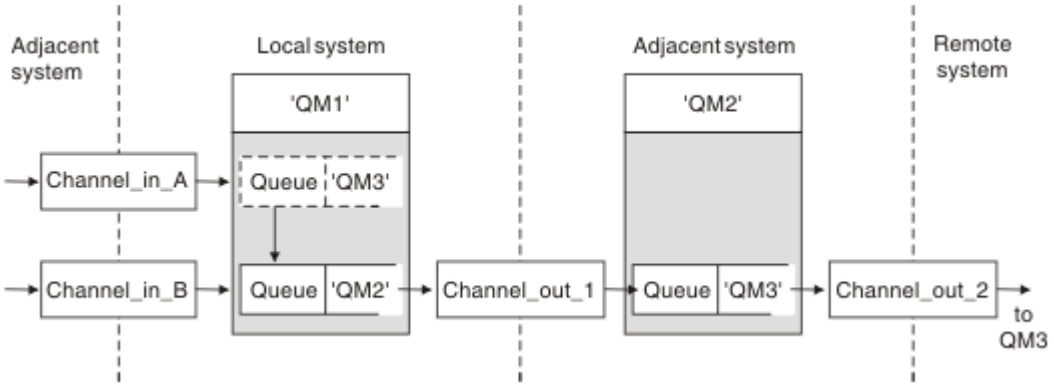


Figura 15. Alias do gerenciador de filas

Todas as mensagens para QM3 são capturadas em QM1 com um alias do gerenciador de filas. O alias do gerenciador de filas é denominado QM3 e contém a definição QM3 por meio da fila de transmissão QM2. A definição se parece com o seguinte exemplo:

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

O gerenciador de filas coloca as mensagens na fila de transmissão QM2 mas não altera o cabeçalho da fila de transmissão porque o nome do gerenciador de filas de destino, QM3, não é mudado.

Todas as mensagens chegam em QM1 e mostrando um cabeçalho de transmissão contendo um nome da fila em QM2 também são colocadas na fila de transmissão QM2. Desta maneira, as mensagens com diferentes destinos são coletadas em uma fila de transmissão comum para um sistema adjacente apropriado, para transmissões adicionais em seus destinos.

Mensagens de Entrada - Determinando o Destino

Um MCA de recebimento abre a fila referenciada no cabeçalho de transmissão. Se uma definição de alias do gerenciador de filas existir com o mesmo nome que o gerenciador de filas referenciado, o nome do gerenciador de filas recebido no cabeçalho de transmissão será substituído pelo RQMNAME a partir dessa definição.

Este processo tem dois usos:

- Direcionar as mensagens para outro gerenciador de filas
- Alterar o nome do gerenciador de filas para ser igual ao gerenciador de filas locais

Definições de Alias da Fila de Resposta

Uma definição de alias da fila de resposta especifica nomes alternativos para informações de resposta no descritor de mensagem. A vantagem disso é que você pode alterar o nome de uma fila ou o gerenciador de filas sem ter que alterar os seus aplicativos.

Resolução de Nome da Fila

Quando um aplicativo responde a uma mensagem, ele usa os dados no *descritor de mensagens* da mensagem recebida para descobrir o nome da fila para a qual responder. O aplicativo de envio indica para onde as respostas são enviadas e anexa essas informações a suas mensagens. Esse conceito deve ser coordenado como parte do design do aplicativo.

A resolução de nome de fila acontece na extremidade de envio do aplicativo antes de a mensagem ser colocada em uma fila. A resolução de nome de fila ocorre antes da interação com o aplicativo remoto para o qual a mensagem está sendo enviada. Esta é a única situação em que ocorre a resolução do nome no momento em que uma fila não estiver sendo aberta.

Resolução de Nome de Fila Usando um Alias de Gerenciador de Filas

Normalmente, um aplicativo especifica uma fila de resposta e deixa o nome do gerenciador de fila de resposta em branco. O gerenciador de filas conclui seu próprio nome no momento da inserção. Este método funciona bem, exceto quando você deseja que um canal alternativo seja usado para respostas, por exemplo, um canal que usa a fila de transmissão `QM1_relief` em vez do canal de retorno padrão que usa a fila de transmissão `QM1`. Nessa situação, os nomes de gerenciador de filas especificados em cabeçalhos de fila de transmissão não correspondem aos nomes de gerenciador de filas "reais", mas eles são especificados novamente usando definições de alias do gerenciador de filas. Para retornar as respostas juntamente com as rotas alternativas, é necessário mapear os dados da fila de resposta também, usando as definições de alias de fila de resposta.

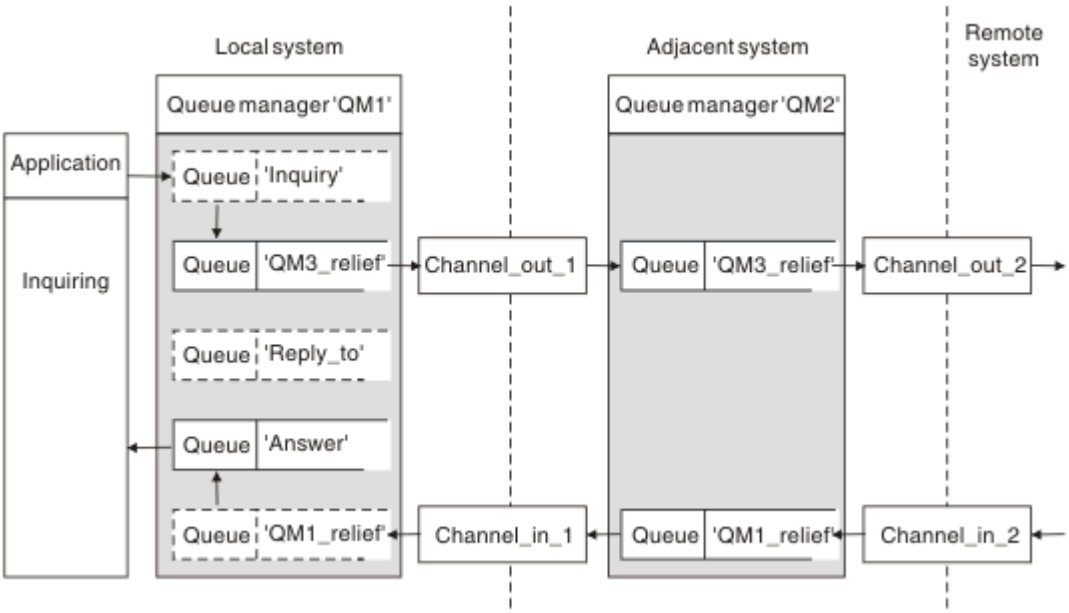


Figura 16. Alias da Fila de Resposta Usados para Alterar o Local de Resposta

No exemplo em Figura 16 na página 57:

1. O aplicativo coloca uma mensagem que usa a chamada MQPUT e que especifica as seguintes informações no descritor de mensagens:

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

   ReplyToQMgr deve estar em branco para que o alias da fila de resposta seja usado.
2. Você cria uma definição de alias de fila de resposta chamada `Reply_to`, a qual contém o nome `Answer`, e o nome do gerenciador de filas `QM1_relief`.

   ```
   DEFINE QREMOTE ('Reply_to') RNAME ('Answer')
   RQMNAME ('QM1_relief')
   ```
3. As mensagens são enviadas com um descritor de mensagens que mostra `ReplyToQ='Answer'` e `ReplyToQMgr='QM1_relief'`.
4. A especificação do aplicativo deve incluir a informação de que as respostas devem ser localizadas na fila `Answer` em vez de `Reply_to`.

Para preparar-se para as respostas, você tem que criar o canal de retorno paralelo, definindo:

- Em QM2, a fila de transmissão denominada `QM1_relief`

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

- Em QM1, o alias do gerenciador de filas `QM1_relief`

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

  Este alias do gerenciador de filas termina a cadeia de canais de retorno paralelo e captura as mensagens para QM1.

Se você acha que talvez queira fazer isso em algum momento no futuro, certifique-se que de os aplicativos usam o nome de alias do início. Por enquanto, esse é um alias de fila normal para a fila de resposta, mas posteriormente, poderá ser mudado para um alias do gerenciador de filas.

Nome da Fila de Resposta

É necessário cuidado com a nomeação das filas de resposta. A razão pela qual um aplicativo coloca um nome da fila de resposta na mensagem é que ela pode especificar a fila para a qual suas respostas são enviadas. Ao criar uma definição de alias da fila de resposta com este nome, você não pode ter a fila de resposta real (ou seja, uma definição de fila local) com o mesmo nome. Portanto, a definição de alias da fila de resposta deve conter um novo nome de fila bem como o nome do gerenciador de filas e a especificação do aplicativo deve incluir as informações nas quais suas respostas são encontradas nesta outra fila.

Os aplicativos agora tem que recuperar as mensagens de uma fila diferente daquela que nomearam como fila de resposta quando colocaram a mensagem original.

Componentes do Cluster

Os clusters são compostos de gerenciadores de fila, repositórios do cluster, canais de cluster e filas de cluster.

Consulte os subtópicos a seguir para obter informações sobre cada um dos componentes de cluster:

Conceitos relacionados

Comparação de Armazenamento em Cluster e Enfileiramento Distribuído

Tarefas relacionadas

Configurando um cluster do gerenciador de filas

Configurando um novo cluster

Repositório de Cluster

Um repositório é uma coleção de informações sobre os gerenciadores de filas que são membros de um cluster.

As informações do repositório incluem os nomes dos gerenciadores de filas, seus locais, seus canais, quais filas eles hospedam e outras informações. As informações são armazenadas na forma de mensagens em uma fila chamada `SYSTEM.CLUSTER.REPOSITORY.QUEUE`. Essa fila é um dos objetos padrão. Multi No Multiplataformas, ela é definida quando você cria um gerenciador de filas do IBM MQ. z/OS No IBM MQ for z/OS, ela é definida como parte da customização do gerenciador de filas.

Repositório Completo e Repositório Parcial

Geralmente, dois gerenciadores de filas em um cluster contêm um repositório completo. Todos os gerenciadores de filas restantes contêm um repositório parcial.

Um gerenciador de filas que hospeda um conjunto completo de informações sobre cada gerenciador de filas no cluster tem um repositório completo. Outros gerenciadores de filas no cluster têm repositórios parciais que contém um subconjunto das informações nos repositórios completos.

Um repositório parcial contém informações apenas sobre os gerenciadores de filas com os quais o gerenciador de filas precisa trocar mensagens. Os gerenciadores de filas solicitam atualizações nas informações que eles precisam, de forma que, se elas forem alteradas, o gerenciador de filas do repositório completo enviará as novas informações. Na maior parte do tempo, um repositório parcial contém todas as informações que um gerenciador de filas precisa para executar no cluster. Quando um gerenciador de filas precisa de alguma informação adicional, ele faz consultas do repositório completo e atualiza seu repositório parcial. Os gerenciadores de filas usam a fila `SYSTEM.CLUSTER.COMMAND.QUEUE` para solicitar e receber atualizações para os repositórios.

Ao migrar os gerenciadores de filas que são membros de um cluster, migre os repositórios completos antes dos repositórios parciais. Isso é porque um repositório mais antigo não pode armazenar atributos mais novos introduzidos em uma liberação mais recente. Ele os tolera, mas não os armazena.

Gerenciador de filas do cluster

Um gerenciador de filas do cluster é um gerenciador de filas que é membro de um cluster.

Um gerenciador de filas pode ser membro de mais de um cluster. Cada gerenciador de filas do cluster deve ter um nome que seja exclusivo em todos os clusters nos quais ele é um membro.

Um gerenciador de filas do cluster pode hospedar filas, que ele anuncia aos outros gerenciadores de filas no cluster. No entanto, ele não tem de fazer isso. Em vez disso, ele pode alimentar mensagens em filas hospedadas em outro lugar no cluster e receber somente respostas que são direcionadas explicitamente a ele.

z/OS No IBM MQ for z/OS, um gerenciador de filas do cluster pode ser um membro de um grupo de filas compartilhadas. Nesse caso, ele compartilha suas definições de fila com outros gerenciadores de filas no mesmo grupo de filas compartilhadas.

Os gerenciadores de filas do cluster são autônomos. Eles possuem controle total sobre filas e canais que eles definem. Suas definições não podem ser modificadas por outros gerenciadores de filas (diferentes dos gerenciadores de filas no mesmo grupo de filas compartilhadas). Os gerenciadores de filas do repositório não controlam as definições em outros gerenciadores de filas no cluster. Elas retém um conjunto completo de todas as definições, para uso quando necessário. Um cluster é uma federação de gerenciadores de filas.

Após criar ou alterar uma definição em um gerenciador de filas do cluster, as informações serão enviadas para o gerenciador de filas do repositório completo. Outros repositórios no cluster são atualizados posteriormente.

Gerenciador de Filas do Repositório Completo

Um gerenciador de filas do repositório completo é um gerenciador de filas do cluster que contém uma representação completa dos recursos do cluster. Para assegurar a disponibilidade, configure dois ou mais gerenciadores de fila do repositório completo em cada cluster. Os gerenciadores de filas do repositório completo recebem informações enviadas por outros gerenciadores de filas no cluster e atualizam seus repositórios. Eles enviam mensagens uns aos outros para assegurarem-se de que ambos serão mantidos atualizados com as novas informações sobre o cluster.

Repositórios e Gerenciadores de Filas

Cada cluster possui pelo menos um (preferivelmente dois) gerenciador de filas contendo repositórios completos de informações sobre os gerenciadores de filas, filas e canais em um cluster. Estes repositórios também contêm solicitações de outros gerenciadores de filas no cluster de atualizações das informações.

Os outros gerenciadores de filas contêm, cada um, um repositório parcial, contendo informações sobre o subconjunto de filas e gerenciadores de filas com os quais eles precisam se comunicar. Os gerenciadores de filas constroem seus repositórios parciais fazendo consultas quando eles precisam acessar pela primeira vez outra fila ou o gerenciador de filas. Eles solicitam ser notificados de qualquer nova informação concernente a essa fila ou ao gerenciador de filas.

Cada gerenciador de filas armazena suas informações de repositório em mensagens em uma fila chamada `SYSTEM.CLUSTER.REPOSITORY.QUEUE`. Os gerenciadores de filas trocam informações do repositório em mensagens em uma fila chamada `SYSTEM.CLUSTER.COMMAND.QUEUE`.

Cada gerenciador de filas que se associa a um cluster define um canal emissor de cluster, `CLUSSDR`, para um dos repositórios. Ele aprende imediatamente quais outros gerenciadores de filas no cluster contêm repositórios completos. Daí em diante, o gerenciador de filas pode solicitar informações a partir de qualquer um dos repositórios. Quando o gerenciador de filas enviar informações para o repositório escolhido, ele também enviará informações para outro repositório (se houver um).

Um repositório completo será atualizado quando o gerenciador de filas que o hospeda receber novas informações de um dos gerenciadores de filas que estão vinculados a ele. As novas informações também são enviadas a outro repositório, para reduzir o risco delas serem atrasadas se um gerenciador de filas de repositório estiver fora de serviço. Como todas as informações são enviadas duas vezes, os repositórios precisam descartar as duplicatas. Cada item de informações transporta um número de sequência, o qual os repositórios usam para identificar as duplicatas. Todos os repositórios são mantidos em etapa para troca de mensagens.

Filas de Clusters

Uma fila de clusters é uma fila que é hospedada por um gerenciador de filas do cluster e disponibilizada para outros gerenciadores de filas no cluster.

Uma definição de fila de clusters é divulgada para outros gerenciadores de filas no cluster. Os outros gerenciadores de filas no cluster podem colocar mensagens em uma fila de clusters sem a necessidade de uma definição de fila remota correspondente. Uma fila de clusters pode ser divulgada em mais de um cluster usando uma lista de nomes de cluster.

Quando uma fila é divulgada, qualquer gerenciador de filas no cluster pode colocar mensagens nela. Para colocar uma mensagem, o gerenciador de filas deve descobrir, a partir dos repositórios completos, onde a fila está hospedada. Em seguida, ele inclui algumas informações de roteamento na mensagem e coloca a mensagem em uma fila de transmissão do cluster.

Uma fila de clusters pode ser uma fila que é compartilhada por membros de um grupo de filas compartilhadas no IBM MQ for z/OS.

Tarefas relacionadas

Definindo filas de clusters

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.

Canais de cluster

Em cada repositório completo, você define manualmente um canal do receptor de clusters, e um conjunto de canais do emissor de clusters para se conectar a qualquer outro repositório completo no cluster. Ao incluir um repositório parcial, você define manualmente um canal do receptor de clusters, e um canal do emissor de clusters único que se conecta a um dos repositórios completos. Canais do emissor de clusters adicionais são definidos automaticamente pelo cluster quando necessário. Canais do emissor de clusters definidos automaticamente obtém seus atributos a partir da definição de canal do receptor de clusters correspondente no gerenciador de filas de recebimento.

Canal do Receptor de Clusters: CLUSRCVR

Uma definição de canal CLUSRCVR define o término de um canal no qual um gerenciador de filas do cluster pode receber mensagens de outros gerenciadores de filas no cluster.

Deve-se definir pelo menos um canal CLUSRCVR para cada gerenciador de filas do cluster. Definindo o canal CLUSRCVR, o gerenciador de filas mostra aos outros gerenciadores de filas do cluster que está disponível para receber mensagens.

Uma definição de canal CLUSRCVR também permite que outros gerenciadores de filas definam automaticamente definições do canal do emissor de clusters correspondentes. Consulte a seção “Canais do emissor de clusters definidos automaticamente” na página 62 deste artigo.

Canal do Emissor de Clusters: CLUSSDR

Você define manualmente um canal CLUSSDR a partir de cada gerenciador de filas de repositório completo para qualquer outro gerenciador de filas de repositório completo no cluster. Todas as

atualizações trocadas pelos repositórios completos fluem exclusivamente nesses canais. Definindo manualmente esses canais, você controla a rede de repositórios completos explicitamente.

Ao incluir um gerenciador de filas de repositório parcial em um cluster, você define manualmente um único canal CLUSSDR para se conectar a um dos repositórios completos. Faz pouca diferença qual repositório completo é escolhido, porque, após o contato inicial ter sido feito, objetos do gerenciador de filas do cluster adicionais para seu gerenciador de filas, incluindo canais CLUSSDR, são definidos automaticamente conforme necessário. Isso permite que o gerenciador de filas envie informações do cluster para qualquer repositório completo, e envie mensagens para qualquer gerenciador de filas no cluster.

Conforme está explicado na seção deste artigo, canais do emissor definidos automaticamente são baseados na configuração do canal do receptor de clusters. Portanto, qualquer propriedade de canal que você configure em canais de cluster deve ser configurada de forma idêntica em canais do receptor de clusters CLUSSDR correspondentes ou configuradas somente nos canais do receptor de clusters.

Você deve definir canais CLUSSDR manualmente somente pelos motivos descritos anteriormente. Ou seja, para conectar inicialmente um repositório parcial a um repositório completo ou para conectar dois repositórios completos juntos. Configurar manualmente um canal CLUSSDR que se conecta a um repositório parcial ou a um gerenciador de filas não no cluster, causa a emissão de mensagens de erro como AMQ9427 e AMQ9428. Embora isto, às vezes, possa ser inevitável como uma situação temporária, por exemplo ao modificar o local de um repositório completo, a definição manual deve ser excluída o mais rápido possível.

Canais do emissor de clusters definidos automaticamente

Geralmente, quando você inclui um gerenciador de filas de repositório parcial em um cluster, você define manualmente somente dois canais de cluster no gerenciador de filas:

- Um canal do emissor de clusters (CLUSSDR) para um gerenciador de filas de repositório completo para o cluster.
- Um canal do receptor de cluster (CLUSRCVR).

O canal CLUSSDR que você define permite que o gerenciador de filas faça o contato inicial com o cluster. Depois que o contato inicial foi feito, canais CLUSSDR adicionais são definidos automaticamente pelo cluster quando necessário.

Um canal CLUSSDR definido automaticamente obtém seus atributos da definição de canal CLUSRCVR correspondente no gerenciador de filas de recebimento. Mesmo se houver um canal CLUSSDR definido manualmente, os atributos do canal CLUSSDR definido automaticamente são usados. Suponha, por exemplo, que você define um canal CLUSRCVR sem especificar um número de porta no parâmetro **CONNAME**, e define manualmente um canal CLUSSDR que não especifica um número de porta. Quando o canal CLUSSDR definido automaticamente substitui o definido manualmente, o número da porta (obtido do canal CLUSRCVR) se torna em branco. O número da porta padrão é usado e o canal falha.

Onde houver diferenças de configuração entre um canal CLUSSDR definido manualmente e a definição de canal CLUSRCVR correspondente, algumas diferenças entram em vigor imediatamente (por exemplo, os parâmetros de balanceamento de carga de trabalho) e algumas entram em vigor somente na reinicialização do canal (por exemplo, a configuração do TLS).

Para evitar confusão, tanto quanto possível observe as diretrizes a seguir:

- Defina manualmente os canais CLUSSDR somente para apontar para repositórios completos.
- Se você tiver definido manualmente canais CLUSSDR, configure-os para corresponder de forma idêntica à respectiva definição de canal CLUSRCVR no gerenciador de filas de recebimento.

Consulte também Trabalhando com canais definidos automaticamente.

Conceitos relacionados

Trabalhando com canais autodefinidos

Trabalhando com filas de transmissão do cluster e canais do emissor de clusters

Tarefas relacionadas

Configurando um novo cluster

Incluindo um Gerenciador de Filas em um Cluster

Tópicos em Cluster

Os tópicos de cluster são tópicos administrativos com o atributo de **`cluster`** definido Informações sobre tópicos de cluster são enviadas por push para todos os membros de um cluster, e combinadas com tópicos locais para criar partes de um espaço de tópico que se estende por vários gerenciadores de filas. Isso permite que mensagens publicadas em um tópico em um gerenciador de filas sejam entregues às assinaturas de outros gerenciadores de filas no cluster.

Ao definir um tópico de cluster em um gerenciador de filas, a definição de tópico de cluster é enviada para os gerenciadores de filas do repositório completo. Os repositórios completos, em seguida, propagam a definição de tópico de cluster para todos os gerenciadores de filas no cluster, tornando o mesmo tópico de cluster disponível para publicadores e assinantes em qualquer gerenciador de filas no cluster. O gerenciador de filas no qual você cria um tópico de cluster é conhecido como um host de tópico de cluster. O tópico de cluster pode ser usado por qualquer gerenciador de filas no cluster, mas quaisquer modificações em um tópico de cluster deverão ser feitas no gerenciador de filas no qual esse tópico está definido (o host), nesse ponto a modificação é propagada para todos os membros do cluster por meio dos repositórios completos.

Para obter informações sobre como configurar tópicos de cluster para usar *roteamento direto* ou *roteamento de host de tópico*, e sobre herança de tópico em cluster e assinaturas curingas, consulte Definindo tópicos de cluster.

Para obter informações sobre os comandos a serem usados para exibir tópicos de cluster, consulte as informações relacionadas.

Conceitos relacionados

Trabalhando com tópicos administrativos

Trabalhando com assinaturas

Referências relacionadas

DISPLAYTOPIC

DISPLAYTPSTATUS

DISPLAYSUB

Objetos de Cluster Padrão

Multi Em Multiplataformas, os objetos de cluster padrão são incluídos no conjunto de objetos padrão criado automaticamente quando você define um gerenciador de filas. z/OS No z/OS, as definições de objeto do cluster padrão podem ser localizadas nas amostras de customização.

Nota: É possível alterar as definições de canal padrão da mesma maneira que qualquer outra definição de canal, executando comandos MQSC ou PCF. Não altere as definições de fila padrão, exceto para `SYSTEM.CLUSTER.HISTORY.QUEUE`.

`SYSTEM.CLUSTER.COMMAND.QUEUE`
: Cada gerenciador de filas em um cluster possui uma fila local chamada `SYSTEM.CLUSTER.COMMAND.QUEUE` que é usada para transferir mensagens para o repositório completo. A mensagem contém quaisquer informações novas ou alteradas sobre o gerenciador de filas ou quaisquer solicitações por informações sobre outros gerenciadores de filas. `SYSTEM.CLUSTER.COMMAND.QUEUE` normalmente está vazia.

`SYSTEM.CLUSTER.HISTORY.QUEUE`
: Cada gerenciador de filas em um cluster tem uma fila local chamada `SYSTEM.CLUSTER.HISTORY.QUEUE`. O `SYSTEM.CLUSTER.HISTORY.QUEUE` é usado para armazenar o histórico de informações do estado do cluster para fins de serviço.

Nas configurações do objeto padrão, SYSTEM.CLUSTER.HISTORY.QUEUE é configurado como PUT (ENABLED). Para suprimir a coleção histórica, mude a configuração para PUT (DISABLED).

SYSTEM.CLUSTER.REPOSITORY.QUEUE
Cada gerenciador de filas em um cluster tem uma fila local chamada SYSTEM.CLUSTER.REPOSITORY.QUEUE. Esta fila é usada para armazenar todas as informações de repositório completo. Esta fila normalmente não está vazia.

SYSTEM.CLUSTER.TRANSMIT.QUEUE
Cada gerenciador de filas tem uma definição para uma fila local chamada SYSTEM.CLUSTER.TRANSMIT.QUEUE. SYSTEM.CLUSTER.TRANSMIT.QUEUE é a fila de transmissão padrão para todas as mensagens de todas as filas e gerenciadores de filas que estão nos clusters. É possível mudar a fila de transmissão padrão para cada canal do emissor de clusters para SYSTEM.CLUSTER.TRANSMIT. *ChannelName*, mudando o atributo do gerenciador de filas DEFCLXQ . Não é possível excluir SYSTEM.CLUSTER.TRANSMIT.QUEUE. Ele também é usado para definir verificações de autorização se a fila de transmissão padrão usada é SYSTEM.CLUSTER.TRANSMIT.QUEUE ou SYSTEM.CLUSTER.TRANSMIT. *ChannelName*.

SYSTEM.DEF.CLUSRCVR
Cada cluster possui uma definição de canal CLUSRCVR padrão denominada SYSTEM.DEF.CLUSRCVR. O SYSTEM.DEF.CLUSRCVR é usado para fornecer valores padrão para quaisquer atributos que não forem especificados ao criar um canal do receptor de clusters em um gerenciador de filas no cluster.

SYSTEM.DEF.CLUSSDR
Cada cluster possui uma definição de canal CLUSSDR padrão chamada SYSTEM.DEF.CLUSSDR. SYSTEM.DEF.CLUSSDR é usado para fornecer valores padrão para quaisquer atributos que não forem especificados ao criar um canal do emissor de clusters em um gerenciador de filas no cluster.

Conceitos relacionados

Trabalhando com objetos do cluster padrão

Sistema de Mensagens de Publicação/Assinatura

O sistema de mensagens de publicação/assinatura permite separar o provedor de informações dos consumidores dessas informações. O aplicativo de envio e o aplicativo de recebimento não precisam saber nada um do outro para que as informações sejam enviadas e recebidas.

Antes que um aplicativo IBM MQ ponto a ponto possa enviar uma mensagem para outro aplicativo, ele precisa saber algo sobre esse aplicativo. Por exemplo, precisará saber o nome da fila para a qual enviar as informações, e também pode especificar um nome do gerenciador de filas.

O IBM MQ de publicação/assinatura remove a necessidade do aplicativo saber alguma coisa sobre o aplicativo de destino. Todos os aplicativos de envio precisam fazer o seguinte:

- *Coloque* uma mensagem do IBM MQ que contenha as informações que o aplicativo deseja.
- Designe a mensagem em um tópico que denote o assunto das informações.
- Deixe que o IBM MQ manipule a distribuição dessas informações.

Semelhantemente, o aplicativo de destino não precisa saber nada sobre a origem das informações recebidas.

A figura a seguir mostra o sistema publicar/assinar mais simples. Existe um publicador, um gerenciador de filas e um assinante. Uma assinatura é feita pelo assinante em um gerenciador de filas, uma publicação é enviada do publicador para o gerenciador de filas, e a publicação é então encaminhada pelo gerenciador de filas para o assinante.

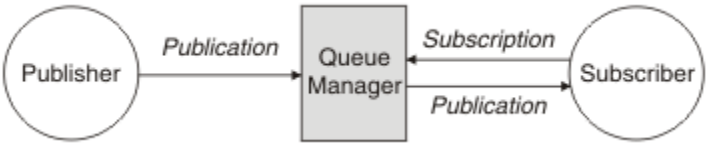


Figura 17. Configuração Simples de Publicação/Assinatura

Um sistema de publicação/assinatura típico tem mais de um publicador e mais de um assinante em muitos tópicos diferentes, e geralmente, mais de um gerenciador de filas. Um aplicativo pode ser tanto um publicador quanto um assinante.

Outra diferença importante entre o sistema de mensagens publicar/assinar e ponto a ponto é que uma mensagem enviada para uma fila ponto a ponto é processada somente por um único aplicativo consumidor. Uma mensagem publicada em um tópico publicar/assinar, no qual mais de um assinante registrou interesse, é processada por cada assinante interessado.

Componentes publicar/assinar

Publicação/Assinatura é o mecanismo pelo qual os assinantes podem receber informações na forma de mensagens, de publicadores. As interações entre os publicadores e assinantes são controladas por gerenciadores de filas, usando os recursos padrão do IBM MQ.

Um sistema de publicação/assinatura típico tem mais de um publicador e mais de um assinante em muitos tópicos diferentes, e geralmente, mais de um gerenciador de filas. Um aplicativo pode ser tanto um publicador quanto um assinante.

O provedor de informações é chamado de *publicador*. Os publicadores fornecem informações sobre um assunto, sem precisar saber nada sobre os aplicativos que estão interessados nessa informação. Os publicadores geram essas informações na forma de mensagens, chamadas *publicações* que desejam publicar e definir o tópico dessas mensagens.

O consumidor das informações é denominado *assinante*. Assinantes criam *assinaturas* que descrevem o tópico em que o assinante está interessado. Assim, a assinatura determina quais publicações são encaminhadas ao assinante. Os assinantes podem fazer várias assinaturas e podem receber informações de vários publicadores diferentes.

Informações publicadas são enviadas em uma mensagem do IBM MQ e o assunto das informações é identificado por seu *tópico*. O publicador especifica o tópico quando ele publica as informações e o assinante especifica os tópicos sobre os quais ele deseja receber publicações. O assinante recebe informações apenas sobre os tópicos que ele assina.

É a existência de tópicos que permite que os provedores e consumidores das informações sejam dissociados no sistema de mensagens de publicação/assinatura, removendo a necessidade de incluir um destino específico em cada mensagem, conforme necessário para o sistema de mensagens ponto a ponto.

As interações entre os publicadores e os assinantes são todas controladas por um gerenciador de filas. O gerenciador de filas recebe mensagens dos publicadores e assinaturas dos assinantes (para um intervalo de tópicos). A tarefa do gerenciador de filas é rotear as mensagens publicadas aos assinantes que registraram interesse no tópico das mensagens.

Os recursos do IBM MQ são usados para distribuir mensagens, portanto, seus aplicativos podem usar todos os recursos que estão disponíveis para aplicativos existentes do IBM MQ. Isso significa que você pode usar mensagens persistentes para obter a entrega única garantida, e que suas mensagens possam ser parte de uma unidade transacional de trabalho para assegurar que as mensagens sejam entregues ao assinante apenas se forem confirmadas pelo publicador.

Publicadores e Publicações

No IBM MQ, um publicador é um aplicativo que disponibiliza informações sobre um tópico especificado disponíveis para um gerenciador de filas na forma de um padrão do IBM MQ mensagem de chamada de publicação. Um publicador pode publicar informações sobre mais de um tópico.

Os publicadores usam o verbo MQPUT para colocar uma mensagem em um tópico aberto anteriormente; esta mensagem é uma publicação. Em seguida, o gerenciador de filas locais roteia a publicação para todos os assinantes que têm assinaturas do tópico da publicação. Uma mensagem publicada pode ser consumida por mais de um assinante.

Além de distribuir publicações para todos os assinantes locais que têm assinaturas apropriadas, um gerenciador de filas também pode distribuir a publicação para quaisquer outros gerenciadores de filas conectados a ele, diretamente ou através de uma rede de gerenciadores de filas que possuem assinantes do tópico.

Em um IBM MQ de rede, um aplicativo de publicação também pode ser um assinante.

Publicações no Ponto de Sincronização

Os publicadores podem emitir chamadas MQPUT ou MQPUT1 no ponto de sincronização para incluir todas as mensagens entregues a assinantes em uma unidade de trabalho. Se a opção MQPMO_RETAIN ou as opções de entrega de tópico NPMSGDLV ou PMSGDLV com valores ALL ou ALLDUR forem especificadas, o gerenciador de filas usará chamadas MQPUT ou MQPUT1 internas no ponto de sincronização, dentro do escopo da chamada MQPUT ou MQPUT1 do publicador.

Informações de Evento e de Estado

As publicações podem ser categorizadas como publicações de estado, como o preço atual de uma ação ou publicações de evento, como um comércio dessa ação.

Publicações de Estado

Publicações de estado contêm informações sobre o estado atual de algo, como o preço de ações ou a pontuação atual em um jogo de futebol. Quando acontece algo (por exemplo, o preço das ações ou a pontuação da partida de futebol são mudados), as informações de estado anteriores não são mais necessárias porque são substituídas pelas novas informações.

Um assinante vai querer receber a versão atual das informações de estado quando for iniciado, e serão enviadas novas informações sempre que houver mudanças no estado.

Se uma publicação contiver informações sobre estado, ela muitas vezes será publicada como uma publicação retida. Geralmente, um novo assinante deseja as informações de estado atual imediatamente. Ele não quer esperar por um evento que faça com que as informações sejam publicadas novamente. Os assinantes irão receber automaticamente uma publicação retida do tópico quando assinar a menos que o assinante use as opções MQSO_PUBLICATIONS_ON_REQUEST ou MQSO_NEW_PUBLICATIONS_ONLY.

Publicações de Evento

Publicações de evento contêm informações sobre eventos individuais que ocorrem, como o comércio de alguma ação ou a pontuação de um determinado jogo. Cada evento é independente do outro.

Um assinante vai querer receber informações sobre eventos assim que acontecem.

Publicações Retidas

Por padrão, após uma publicação ser enviada a todos os assinantes interessados, ela é descartada. No entanto, um publicador pode especificar que uma cópia de uma publicação seja retida para que ela possa ser enviada aos assinantes futuros que registrarem um interesse no tópico.

A exclusão de publicações depois de terem sido enviadas a todos os assinantes interessados é adequada para informações de evento, mas nem sempre é adequada para informações de estado. Ao reter uma mensagem, novos assinantes não precisam esperar até que as informações sejam publicadas novamente antes que eles recebam as informações de estado iniciais. Por exemplo, um assinante com uma assinatura para um preço de ação receberia o preço atual imediatamente, sem aguardar que o preço da ação fosse mudado (e portanto, republicado).

O gerenciador de filas pode reter apenas uma publicação de cada tópico, portanto, a publicação retida existente de um tópico é excluída quando uma nova publicação retida chega no gerenciador de filas. No entanto, a exclusão da publicação existente não pode ocorrer de maneira síncrona com a chegada da nova publicação retida. Portanto, sempre que possível, não têm mais de um publicador enviando publicações retidas em qualquer tópico.

Os assinantes podem especificar que eles não desejam receber publicações retidas usando a opção de assinatura MQSO_NEW_PUBLICATIONS_ONLY. Os assinantes existentes podem pedir que cópias duplicadas das publicações retidas sejam enviadas a eles.

Há momentos em que você pode não desejar reter publicações, mesmo que para informações de estado:

- Se todas as assinaturas para um tópico forem feitas antes que seja feita qualquer publicação sobre esse tópico, e você não espera ou não permite, novas assinaturas, não haverá necessidade de reter publicações porque elas serão entregues ao conjunto completo de assinantes na primeira vez que forem publicadas.
- Se as publicações ocorrem frequentemente, tal como a cada segundo, um novo assinante (ou um assinante que recupera de uma falha) recebe o estado atual quase imediatamente após a sua primeira assinatura, portanto, não haverá necessidade de reter essas publicações.
- Se as publicações forem grandes, você poderá acabar necessitando de uma quantidade considerável de espaço de armazenamento para armazenar a publicação retida para cada tópico. Em um ambiente do gerenciador de filas múltiplo, as publicações retidas são armazenadas por todos os gerenciadores de filas na rede que possuem uma assinatura correspondente.

Ao decidir se deve usar publicações retidas, considere como os aplicativos de assinatura se recuperam de uma falha. Se o assinante não usar publicações retidas, o aplicativo assinante pode ter que armazenar seu estado atual localmente.

Para assegurar que uma publicação seja retida, use a opção colocar mensagem MQPMO_RETAIN. Se essa opção for usada e a publicação não puder ser retida, a mensagem não será publicada e a chamada falhará com MQRC_PUT_NOT_RETAINED.

Se uma mensagem for uma publicação retida, isso será indicado pela propriedade de mensagem MQIsRetained. A persistência de uma mensagem é conforme ele era quando foi originalmente publicada.

Conceitos relacionados

Considerações de design para publicações retidas em clusters de publicação/assinatura

Publicações no Ponto de Sincronização

Na publicação/assinatura do IBM MQ, o ponto de sincronização pode ser usado por publicadores ou internamente pelo gerenciador de filas.

Publicadores usam sincronização quando emitem chamadas MQPUT/MQPUT1 com a opção MQPMO_SYNCPOINT. Todas as mensagens entregues aos assinantes contam no número máximo de mensagens não confirmadas em uma unidade de trabalho. O atributo do gerenciador de filas MAXUMSGS especifica esse limite. Se o limite for atingido, então o publicador receberá o código de razão 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED.

Quando um publicador emite chamadas MQPUT/MQPUT1 usando MQPMO_NO_SYNCPOINT com a opção MQPMO_RETAIN ou as opções de entrega de tópico NPMSGDLV/PMSGDLV com valores ALL ou ALLDUR, o gerenciador de filas usa pontos de sincronização internos para garantir que as mensagens sejam entregues conforme solicitado. O publicador poderá receber o código de razão 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED se o limite for atingido dentro do escopo da chamada do publicador MQPUT/MQPUT1.

Assinantes e assinaturas

Na publicação/assinatura do IBM MQ, um assinante é um aplicativo que solicita informações sobre um tópico específico a partir de um gerenciador de filas em uma rede de publicação/assinatura. Um assinante pode receber mensagens sobre o mesmo tópico ou tópicos diferentes, de mais de um publicador.

As assinaturas podem ser criadas manualmente usando um comando MQSC ou por aplicativos. Essas assinaturas são emitidas para o gerenciador de filas locais e contêm informações sobre as publicações que o assinante deseja receber:

- O tópico em que o assinante está interessado; isto pode ser resolvido para vários tópicos se curingas forem usados.
- Uma cadeia de seleção opcional a ser aplicada a mensagens publicadas.
- Um manipulador para uma fila (conhecida como a *fila de assinante*), na qual as publicações selecionadas devem ser colocada, e o CorrelId opcional.

O gerenciador de filas locais armazena informações de assinatura e quando ele recebe uma publicação, varre as informações para determinar se há uma assinatura que corresponde ao tópico da publicação e sequência de seleção. Para cada assinatura correspondente, o gerenciador de filas direciona a publicação à fila de assinante do assinante. As informações que um gerenciador de filas armazenam sobre assinaturas podem ser visualizadas através do comandos DIS SUB DIS SBSTATUS.

Uma assinatura é excluída apenas quando um dos seguintes eventos ocorre:

- O assinante cancela usando a chamada MQCLOSE (se a assinatura foi feita como não durável).
- A assinatura expira.
- A assinatura é excluída pelo administrador do sistema usando o comando DELETE SUB.
- O aplicativo do assinante termina (se a assinatura foi feita como não durável).
- O gerenciador de filas está interrompido ou reiniciado (se a assinatura foi feita como não durável).

Ao obter mensagens, use opções apropriadas na chamada MQGET. Se o aplicativo processar apenas mensagens para uma assinatura, então, no mínimo, será necessário usar `get-by-correlid`, conforme demonstrado no programa de amostra C `amqssbxa.c` e no assinante MQ não gerenciado. O **`CorrelId`** a ser usado é retornado de MQSUB no MQSD.campo **`SubCorrelId`** .

Conceitos relacionados

Assinaturas clonadas e compartilhadas

Referências relacionadas

Exemplos de como definir a propriedade sharedSubscription

Filas Gerenciadas e Publicação/Assinatura

Quando criar uma assinatura, você poderá optar por usar o enfileiramento gerenciado. Se você usar o enfileiramento gerenciado, uma fila de assinaturas será criada automaticamente ao criar uma assinatura. As filas gerenciadas são organizadas automaticamente de acordo com a durabilidade da assinatura. Usar filas gerenciadas significa que você não tem que se preocupar com a criação de filas para receber publicações e quaisquer publicações não consumidas são removidas das filas de assinante automaticamente se uma conexão de assinatura não durável for fechada.

Se um aplicativo não tem necessidade de usar uma fila específica como sua fila de assinante, o destino para as publicações que ele recebe, ele pode fazer uso das *assinaturas gerenciadas* usando a opção de assinatura MQSO_MANAGED. Se criar uma assinatura gerenciada, o gerenciador de filas retornará uma manipulação de objetos ao assinante para uma fila de assinantes que o gerenciador de filas cria na qual as publicações serão recebidas. Isso ocorre porque é por meio das *assinaturas gerenciadas* que o IBM MQ manipula a assinatura. A manipulação de objetos da fila será retornada, permitindo procurar, receber ou consultar na fila (não é possível colocar ou configurar atributos de uma fila gerenciada, a menos que você tenha ganho explicitamente o acesso às filas dinâmicas temporárias).

A durabilidade da assinatura determina se a fila gerenciada permanece quando a conexão do aplicativo de assinatura com o gerenciador de filas é interrompida.

As assinaturas gerenciados são particularmente úteis quando usadas com assinaturas não duráveis, porque quando a conexão do aplicativo é encerrada, as mensagens não consumidas são, em vez disso, permanecer na fila de assinantes, assumindo espaço em seu gerenciador de filas indefinidamente. Se você estiver usando uma assinatura gerenciada, a fila gerenciada será uma fila dinâmica temporária e, como tal, será excluída junto com todas as mensagens não consumidas quando a conexão for interrompida por qualquer um dos seguintes motivos:

- MQCLOSE com MQCO_REMOVE_SUB é usado e o Hobj gerenciado é fechado.
- Uma conexão é perdida para um aplicativo usando uma assinatura não durável (MQSO_NON_DURABLE).
- Uma assinatura é removida porque ela expirou e o Hobj gerenciado é fechado.

As assinaturas gerenciadas também pode ser usadas com assinaturas duráveis, mas é possível que você deseje de deixar as mensagens não consumidas na fila de assinantes, para que eles possam ser recuperados quando a conexão for reaberta. Por essa razão, filas gerenciada para assinaturas duráveis assumem a forma de uma fila dinâmica permanente e permanecem quando a conexão do aplicativo de assinatura com o gerenciador de filas é interrompida.

É possível configurar uma expiração em sua assinatura se desejar usar uma fila gerenciada dinâmica permanente de forma que, embora a fila ainda exista depois que a conexão for interrompida, não continue a existir indefinidamente.

Se excluir a fila gerenciada receberá uma mensagem de erro.

As filas gerenciadas que são criadas são nomeadas com números no final (registros de data e hora), assim, cada uma é exclusiva.

Durabilidade da assinatura

Assinaturas podem ser configuradas para serem duráveis ou não duráveis. A durabilidade da assinatura determina o que acontece com as assinaturas quando os aplicativos de assinatura se desconectam de um gerenciador de filas.

Assinaturas duráveis

As assinaturas duráveis continuarão existindo quando a conexão do aplicativo de assinatura com o gerenciador de filas for encerrada. Se uma assinatura for durável, quando o aplicativo de assinatura for desconectado, a assinatura permanecerá no local e poderá ser usada pelo aplicativo de assinatura quando ele for reconectado solicitando a assinatura novamente usando o **SubName** que foi retornado quando a assinatura foi criada.

Quando assinada como durável, um nome de assinatura (**SubName**) é necessário. Os nomes devem ser exclusivos em um gerenciador de filas para que eles possam ser usados para identificar uma assinatura. Esse meio de identificação é necessário ao especificar uma assinatura que você deseja retomar, se você tiver fechado deliberadamente a conexão com a assinatura (usando a opção MQCO_KEEP_SUB) ou tiver sido desconectado do gerenciador de filas. É possível retomar uma assinatura existente usando a chamada MQSUB com a opção MQSO_RESUME. Os nomes de assinatura também serão exibidos se você usar o comando DISPLAY SBSTATUS com SUBTYPE ALL ou ADMIN

Quando um aplicativo não requer mais uma assinatura durável, ela pode ser removida usando a chamada de função MQCLOSE com a opção MQCO_REMOVE_SUB ou pode ser excluída manualmente usando o comando DELETE SUB MQSC.

É possível usar o atributo de tópico **DURSUB** para especificar se assinaturas duráveis de um tópico podem ou não ser feitas.

No retorno de uma chamada MQSUB usando a opção MQSO_RESUME, a expiração da assinatura é configurada para a expiração original da assinatura e não o tempo de expiração restante.

Um gerenciador de filas continua a enviar publicações para satisfazer uma assinatura durável mesmo se esse aplicativo assinante não está conectado. Isso leva a um acúmulo de mensagens na fila de assinantes. A maneira mais fácil para evitar esse problema é usando uma assinatura não durável sempre que apropriado. Entretanto, em que é necessário usar assinaturas duráveis, um acúmulo de mensagens

pode ser evitado se o assinante assina usando a opção Publicações retidas. Um assinante pode então, controlar quando recebe publicações usando a chamada MQSUBRQ.

Assinaturas Não Duráveis

As assinaturas não duráveis existem apenas enquanto a conexão do aplicativo de assinatura com o gerenciador de filas permanecer aberta. A assinatura é removida quando o aplicativo de assinatura se desconecta do gerenciador de filas deliberadamente ou pela perda de conexão. Quando a conexão é fechada, as informações sobre a assinatura são removidas do gerenciador de filas, e não é mais mostrado se você exibe assinaturas usando o comando DISPLAY SBSTATUS. Mais nenhuma mensagem é colocada na fila de assinantes.

O que acontece com todas as publicações não consumidas na fila de assinantes para assinaturas não duráveis é determinado conforme a seguir.

- Se um aplicativo de assinatura estiver usando um destino gerenciado, quaisquer publicações que não tiverem sido consumidas serão removidas automaticamente.
- Se o aplicativo de assinatura fornece uma manipulação para sua própria fila de assinantes quando o assina, as mensagens não consumidas não são removidas automaticamente. É responsabilidade do aplicativo para limpar a fila, se isso for apropriado. Se a fila é compartilhada por mais de um assinante ou outros aplicativos ponto a ponto, pode não ser apropriado limpar a fila completamente.

Embora não requerido para assinaturas não duráveis, um nome de assinatura, se fornecido, é usado pelo gerenciador de filas. Nomes de assinatura devem ser exclusivos dentro do gerenciador de filas, assim, podem ser usados para identificar uma assinatura.

Conceitos relacionados

Assinaturas clonadas e compartilhadas

Tarefas relacionadas

Usando assinaturas compartilhadas do JMS 2.0

Referências relacionadas

Exemplos de como definir a propriedade sharedSubscription

Sequências de seleção

Uma *sequência de seleção* é uma expressão aplicada a uma publicação para determinar se ela corresponde a uma assinatura. Sequências de seleção podem incluir caracteres curinga.

Ao assinar, além de especificar um tópico, você pode especificar uma sequência de seleção para selecionar as publicações de acordo com suas propriedades de mensagem.

A sequência de seleção é avaliada em relação à mensagem como colocadas pelo publicador antes de ser modificada para entrega para cada assinante. Tome cuidado ao usar campos na sequência de seleção que possam ser modificados como parte da operação de publicação. Por exemplo, os campos MQMD `UserIdentifier`, `MsgId`, e `CorrelId`.

Sequências de seleção não devem fazer referência a campos de propriedade de mensagem incluídos pelo gerenciador de filas como parte da operação de publicação (consulte Publicar/assinar propriedades de mensagem), exceto para a propriedade de mensagem `MQTopicString`, que contém a sequência de tópicos para a publicação.

Conceitos relacionados

Regras e restrições de sequência de seleção

tópicos

Um tópico é o assunto das informações que são publicadas em uma mensagem de publicação/assinatura.

Mensagens em sistemas ponto a ponto são enviadas para um endereço de destino específico. Mensagens em sistemas de publicação/assinatura baseados em assunto são enviadas para assinantes com base no assunto que descreve o conteúdo da mensagem. Em sistemas baseados em conteúdo, mensagens são enviadas para assinantes com base no conteúdo da própria mensagem.

O sistema de publicação/assinatura do IBM MQ é um sistema de publicação/assinatura baseado em assunto. Um publicador cria uma mensagem e a publica com a sequência de tópicos que melhor se ajusta ao assunto da publicação. Para receber publicações, um assinante cria uma assinatura com uma sequência de tópicos com correspondência de padrões para selecionar tópicos de publicação. O gerenciador de filas entrega as publicações para os assinantes que têm assinaturas correspondentes ao tópico da publicação e que estão autorizados a receber as publicações. O artigo, “Sequências de tópicos” na página 71, descreve a sintaxe das sequências de tópicos que identificam o assunto de uma publicação. Os assinantes também criam sequências de tópicos para selecionar quais tópicos receber. As sequências de tópicos que os assinantes criam podem conter um dos dois esquemas curinga alternativos para correspondência de padrões com as sequências de tópicos nas publicações. A correspondência de padrão é descrita em “Esquemas Curinga” na página 72.

Na publicação/assinatura baseada em assunto, os publicadores ou administradores, são responsáveis por classificar assuntos nos tópicos. Normalmente os assuntos são organizados hierarquicamente, em árvores de tópicos, usando o caractere `'/'` para criar subtópicos na sequência de tópicos. Consulte “Árvores de Tópicos” na página 78 para obter exemplos de árvores de tópicos. Tópicos são nós na árvore de tópicos. Tópicos podem ser nós folha sem subtópicos ou nós intermediários com subtópicos.

Paralelamente com a organização dos assuntos em uma árvore de tópicos hierárquica, é possível associar tópicos a objetos do tópico administrativo. Você designa atributos a um tópico, como se o tópico é distribuído em um cluster, associando-o a um objeto do tópico administrativo. A associação é feita nomeando o tópico usando o atributo `TOPICSTR` do objeto do tópico administrativo. Se você não associar explicitamente um objeto do tópico administrativo a um tópico, o tópico herdará os atributos de seu ancestral mais próximo na árvore de tópicos que *foi* associado a um objeto do tópico administrativo. Se você não tiver definido nenhum tópico-pai, ele herdará de `SYSTEM.BASE.TOPIC`. Objetos de tópico administrativo são descritos em “Objetos de Tópico Administrativo” na página 79.

Nota: Mesmo se você herdar todos os atributos de um tópico de `SYSTEM.BASE.TOPIC`, defina um tópico raiz para seus tópicos que herde diretamente de `SYSTEM.BASE.TOPIC`. Por exemplo, no espaço de tópico dos estados americanos, no `USA/Alabama` `USA/Alaska`, e assim por diante, o `USA` é o tópico raiz. O principal propósito do tópico raiz é criar espaços de tópico discretos sem sobreposição para evitar que publicações correspondam às assinaturas erradas. Isso também significa que é possível alterar os atributos do tópico raiz para afetar o espaço de tópico inteiro. Por exemplo, você pode configurar o nome para o atributo **CLUSTER**.

Quando você consulta um tópico como publicador ou assinante, você tem a opção de fornecer uma sequência de tópicos, ou consultar um objeto de tópico. Ou pode-se fazer as duas coisas; nesse caso, a sequência de tópicos fornecida define um subtópico do objeto do tópico. O gerenciador de filas identifica o tópico anexando a sequência de tópicos ao prefixo da sequência de tópicos nomeado no objeto do tópico, inserindo um `'/'` adicional entre as duas sequências de tópicos, por exemplo, *sequência de tópicos/sequência de objetos*. “Combinando sequências de tópicos” na página 76 descreve isso com mais detalhes. A sequência de tópicos resultante é usada para identificar o tópico e associá-lo a um objeto do tópico administrativo. O objeto do tópico administrativo não é necessariamente o mesmo objeto do tópico que o objeto do tópico correspondente ao tópico principal.

Na publicação/assinatura baseada em conteúdo, você define quais mensagens quer receber fornecendo sequências de seleção que procuram o conteúdo de cada mensagem. O IBM MQ fornece um forma intermediária de publicação/assinatura baseada em conteúdo usando seletores de mensagem que varrem propriedades de mensagem em vez de o conteúdo completo da mensagem; consulte Seletores. O uso típico de seletores de mensagem é para se subscrever a um tópico e depois qualificar a seleção com uma propriedade numérica. O seletor permite que você especifique que está interessado nos valores de apenas um certo intervalo, o que não é possível fazer usando curingas baseados em tópicos ou caracteres. Se você precisar filtrar com base no conteúdo integral da mensagem, será necessário usar o IBM Integration Bus.

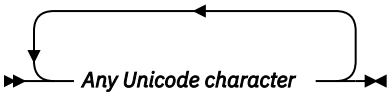
Sequências de tópicos

Informações de rótulo que você publicar como um tópico usando uma sequência de tópicos. Assine grupos de tópicos usando um caractere curinga ou sequência de tópicos do curinga baseado em tópico.

tópicos

Uma *sequência de tópicos* é uma sequência de caracteres que identifica o tópico de uma mensagem de publicação/assinatura. Você pode usar qualquer caractere que quiser quando você construir uma sequência de tópicos.

Cadeia do tópico



Três caracteres têm significado especial em publicar/assinar da IBM WebSphere MQ 7 Eles são permitidos em qualquer lugar em uma sequência de tópicos, mas use-os com cuidado. O uso dos caracteres especiais é explicado em “Esquema Curinga Baseado em Tópicos” na página 73.

Uma barra (/)

O separador de nível de tópico. Use o caractere ' / ' à estrutura do tópico em uma árvore de tópicos.

Evite níveis de tópico vazio, ' / / ', se você puder. Esses correspondem aos nós na hierarquia de tópicos sem sequência de tópicos. Uma liderança ou trilha ' / ' em uma sequência de tópicos corresponde a um nó vazio de liderança ou de trilha e deve ser evitada.

O sinal de hash (#)

Usado em combinação com ' / ' para construir um curinga de vários níveis em assinaturas. Tome cuidado usando '#' adjacente para ' / ' nas sequências de tópicos usadas para nomear tópicos publicados. “Exemplos de Sequências de Tópicos” na página 72 mostra um uso sensato de '#'.

As sequências de caracteres ' . . . /#/ . . . ', '#/ . . . ' e ' . . . /#' possuem um significado especial nas sequências de tópicos de assinatura. As sequências correspondem todos os tópicos em um ou mais níveis na hierarquia de tópicos. Portanto, se você criou um tópico com uma destas sequências, você não poderia assinar a ele, sem também assinar todos os tópicos em diversos níveis na hierarquia de tópicos.

O sinal de mais (+)

Usado em combinação com ' / ' para construir um curinga de único nível em assinaturas. Tome cuidado usando '+' adjacente para ' / ' nas sequências de tópicos usadas para nomear tópicos publicados.

As sequências de caracteres ' . . . /+/ . . . ', '+/ . . . ' e ' . . . /+' possuem um significado especial nas sequências de tópicos de assinatura. As sequências correspondem todos os tópicos em um nível na hierarquia de tópicos. Portanto, se você tiver criado um tópico com uma dessas sequências, não será possível assiná-la, sem também assinar todos os tópicos em um nível na hierarquia de tópicos.

Exemplos de Sequências de Tópicos

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

Referências relacionadas

t¢pico

Esquemas Curinga

Há dois esquemas curingas usados para assinar diversos tópicos. A opção de esquema é uma opção de assinatura.

MQSO_WILDCARD_TOPIC

Selecione os tópicos para assinar usando o esquema curinga baseado em tópico.

Esse será o padrão se nenhum esquema curinga for selecionado explicitamente.

MQSO_WILDCARD_CHAR
Selecione os tópicos para assinar usando o esquema curinga baseado em caracteres.

Configure um esquema especificando o parâmetro **wschema** no comando DEFINE SUB. Para obter mais informações, consulte DEFINE SUB.

Nota: assinaturas que foram criadas antes do IBM WebSphere MQ 7.0 sempre usam o esquema curinga baseado em caracteres.

Exemplos

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

Esquema Curinga Baseado em Tópicos
Curingas baseados em tópicos permitem que os assinantes assinem mais de um tópico por vez.

Curingas baseados em tópicos são um poderoso recurso do sistema de tópicos na publicação IBM MQ . O curinga de vários níveis e o curinga de nível único podem ser utilizados para assinaturas, mas não podem ser utilizados dentro de um tópico pelo publicador de uma mensagem.

O esquema curinga baseado em tópicos permite que você selecione as publicações agrupadas por nível de tópico. É possível escolher, para cada nível na hierarquia de tópicos, se a sequência na assinatura para esse nível de tópico deve corresponder exatamente à sequência na publicação ou não. Por exemplo, a assinatura `IBM/+/Results` seleciona todos os tópicos,

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

Existem dois tipos de curinga.

Curinga de vários níveis

- O curinga de vários níveis é usado em assinaturas. Quando usado em uma publicação, é tratado como um literal.
- O caractere curinga de vários níveis `'#'` é usado para corresponder a qualquer número de níveis dentro de um tópico. Por exemplo, usando o exemplo da árvore de tópicos, se você se assinar `'USA/Alaska/#'`, você receberá mensagens sobre os tópicos `'USA/Alaska'` e `'USA/Alaska/Juneau'`.
- O curinga de vários níveis pode representar zero ou mais níveis. Portanto, `'USA/#'` também pode corresponder ao singular `'USA'`, em que `'#'` representa níveis zero. O separador de nível de tópico é insignificante neste contexto, pois não há nenhum nível a ser separado.
- O curinga de vários níveis é efetivo apenas quando especificado sozinho ou próximo ao caractere separador de nível de tópico. Portanto, `'#'` e `'USA/#'` são tópicos válidos onde o caractere `'#'` é tratado como um curinga. No entanto, embora `'USA#'` também seja uma sequência de tópicos válida, o caractere `'#'` não é considerado como um curinga e não tem nenhum significado especial. Consulte a “Quando Curingas Baseados em Tópicos Não São Válidos” na página 75 para obter mais informações.

Curinga de nível único

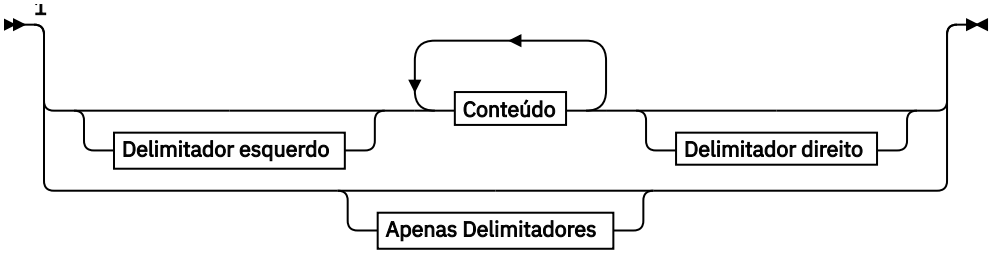
- O curinga único é usado em assinaturas. Quando usado em uma publicação, é tratado como um literal.
- O caractere curinga de nível único `'+'` corresponde a um, e apenas um nível de tópico. Por exemplo, `'USA/+'` corresponde `'USA/Alabama'`, mas não `'USA/Alabama/Auburn'`. Como o curinga de nível único corresponde apenas a um único nível, `'USA/+'` não corresponde a `'USA'`.

- O curinga de nível único pode ser usado em qualquer nível na árvore de tópicos e em conjunto com o curinga de vários níveis. O curinga de nível único deve ser especificado próximo ao separador de nível de tópico, exceto quando ele for especificado sozinho. Portanto, `'+'` e `'USA/+'` são tópicos válidos onde o caractere `'+'` é tratado como um curinga. No entanto, embora `'USA+'` também seja uma sequência de tópicos válida, o caractere `'+'` não é considerado como um curinga e não tem nenhum significado especial. Consulte a "Quando Curingas Baseados em Tópicos Não São Válidos" na página 75 para obter mais informações.

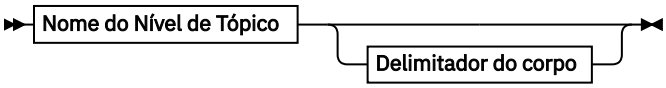
A sintaxe do esquema curinga baseado em tópico não tem caracteres de escape. Se `'#'` e `'+'` forem tratados como curingas ou não dependerá de seu contexto. Consulte"Quando Curingas Baseados em Tópicos Não São Válidos" na página 75 para obter mais informações.

Nota: O início e o fim de uma sequência de tópico são tratados de uma maneira especial. Usando `'$'` para denotar o final da cadeia, `'$#/...'` é um curinga multinível e `'$/#/..'`. será um nó vazio na raiz, seguido por um curinga de vários níveis.

Sequência Curinga Baseada em Tópicos

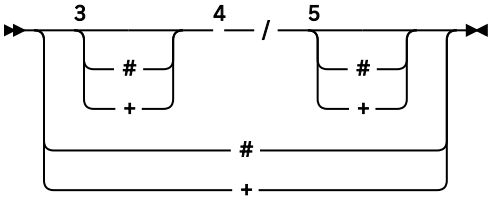


Conteúdo



Nome do Nível de Tópico

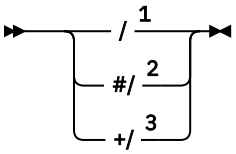
Apenas Delimitadores



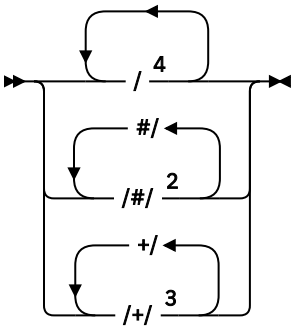
Notas:

[1] Uma sequência de tópicos de comprimento nula ou zero é inválida
[2] É aconselhável não usar nenhum `*, ?, %` em sequências de nomes de nível para compatibilidade entre esquemas curinga baseados em caractere e baseados em tópico.
[3] Estes casos são equivalentes ao *delimitador esquerdo* padrão.
[4] / sem curingas corresponde a um único tópico vazio.
[5] Estes casos são equivalentes ao *delimitador direito* padrão.
[6] Corresponda cada tópico.
[7] Corresponda cada tópico em que há apenas um nível.

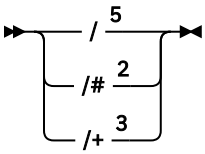
Delimitador Esquerdo



Delimitador de corpo



Delimitador direito



Notas:

[1] A sequência de tópicos começa com um tópico vazio

[2] Corresponde a zero ou mais níveis. Diversas sequências de correspondência de vários níveis têm o mesmo efeito como uma sequência de correspondência de vários níveis.

[3] Corresponde exatamente um nível.

[4] // é um tópico vazio – um objeto do tópico com nenhuma sequência de tópicos.

[5] A sequência de tópicos termina com um tópico vazio

Quando Curingas Baseados em Tópicos Não São Válidos

Os caracteres curinga '+' e '#' não possuem significado especial quando são misturados com outros caracteres (incluindo eles próprios) em um nível de tópico.

Isso significa que os tópicos que contêm '+' ou '#' juntos com outros caracteres em um nível de tópico podem ser publicados.

Por exemplo, considere os dois tópicos a seguir:

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

No primeiro exemplo, os caracteres '+' e '#' são tratados como curingas e, portanto, não são válidos em uma sequência de tópicos que deve ser publicada, mas são válidos em uma assinatura.

No segundo exemplo, os caracteres '+' e '#' não são tratados como curingas e, assim, a sequência de tópicos pode ser tanto publicada quanto assinada.

Exemplos

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

Esquema Curinga Baseado em Caracteres
O esquema curinga baseado em caracteres permite selecionar os tópicos com base no caractere tradicional correspondente.

É possível selecionar todos os tópicos em diversos níveis em uma hierarquia de tópico usando a sequência '*'. Usar '*' no esquema curinga baseado em caracteres é equivalente a usar a sequência curinga baseada em tópico '#'

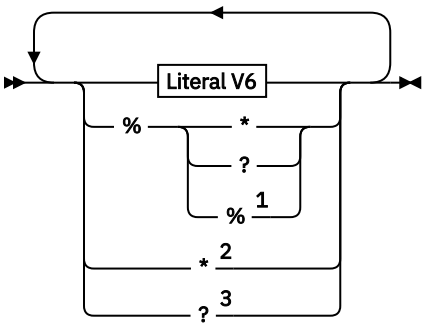
' *x*/*/*y* ' é equivalente a ' *x*/#/*y* ' no esquema baseado em tópico e seleciona todos os tópicos na hierarquia de tópicos entre os níveis ' *x* e *y* ', em que ' *x* ' e ' *y* ' são nomes de tópico que não estão no conjunto de níveis retornados pelo curinga

O '/+/' no esquema baseado em tópicos não tem um equivalente exato no esquema baseado em caracteres. O 'IBM/*/Results' também selecionaria 'IBM/Patents/Software/Results'. Apenas se o conjunto de nomes de tópico em cada nível da hierarquia for exclusivo, será possível construir sempre as consultas com dois esquemas que produzem correspondências idênticas.

Usado de uma maneira geral, '*' e '?' no esquema baseado em caracteres não possuem equivalentes no esquema baseado em tópicos. O esquema baseado em tópico não executa a correspondência parcial usando os curingas. A assinatura 'IBM/*ware/Results' de curinga baseado em caractere não tem nenhum equivalente baseado em tópico.

Nota: As correspondências que usam assinaturas de caractere curinga são mais lentas do que as correspondências que usam assinaturas baseadas em tópico.

Sequência Curinga Baseada em Caractere



Literal V6

Notas:

1. Significa escapar o caractere seguinte, de forma que ele seja tratado como uma literal. '%' deve ser seguido por '*', '?' ou '%'.. Consulte “Exemplos de Sequências de Tópicos” na página 72.
2. Significa corresponder a zero ou mais caracteres em uma assinatura.
3. Significa corresponder exatamente a um caractere em uma assinatura.

Exemplos

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

Combinando sequências de tópicos

Ao criar assinaturas, ou abrir tópicos para poder publicar mensagens neles, a sequência de tópicos pode ser formada pela combinação de duas sequências de subtópicos separadas, ou "subtópicos". Um subtópico é fornecido pelo aplicativo ou comando administrativo como uma sequência de tópicos, e o outro é a sequência de tópicos associada a um objeto do tópico. Você pode usar o subtópico como a própria sequência de tópicos ou combiná-los para formar um nome novo de tópico.

Por exemplo, quando você define uma assinatura usando o comando MQSC **DEFINE SUB**, o comando pode ter **TOPICSTR** (sequência de tópicos) ou **TOPICOBJ** (objeto de tópico) como um atributo, ou ambos juntos. Se apenas **TOPICOBJ** for fornecido, a sequência de tópicos associada a esse objeto de tópico será utilizada como a sequência de tópicos. Se apenas **TOPICSTR** for fornecido, ele será utilizado como a sequência de tópicos. Se ambos forem fornecidos, eles serão concatenados para formar uma sequência de tópicos única no formato **TOPICOBJ** / **TOPICSTR**, em que a sequência de tópicos configurada por **TOPICOBJ** será sempre configurada pela primeira vez e as duas partes da sequência são sempre separadas por um caractere "/".

Da mesma forma, em um programa MQI, o nome integral do tópico será criado por MQOPEN. Ele é composto de dois campos usados em chamadas MQI de publicação/assinatura, na ordem listada:

1. O atributo **TOPICSTR** do objeto de tópico, nomeado no campo **ObjectName**.
2. O parâmetro **ObjectString** define o subtópico fornecido pelo aplicativo.

A sequência de tópicos resultante é retornada no parâmetro **ResObjectString**.

Estes campos são considerados para estarem presentes somente se o primeiro caractere de cada campo não for um caractere em branco ou nulo e o comprimento do campo for maior que zero. Se apenas um dos campos estiver presente, ele será usado sem mudança como o nome do tópico. Se nenhum dos campos tiver um valor, a chamada falhará com o código de razão MQRC_UNKNOWN_OBJECT_NAME, ou MQRC_TOPIC_STRING_ERROR se o nome completo do tópico não for válido.

Se ambos os campos estiverem presentes, um caractere "/" será inserido entre os dois elementos do nome do tópico combinado resultante.

A tabela a seguir mostra exemplos de concatenação de sequência de tópicos:

Tabela 2. Exemplos de Concatenação da Sequência de Tópicos

TOPICSTR do objeto do tópico	**Sequência de tópicos fornecida pelo aplicativo ou comando DEFINE SUB**	**Nome Completo do Tópico**	**Comentário**
Futebol/Pontuações	''	Futebol/Pontuações	O TOPICSTR do objeto de tópico é utilizado sozinho.
''	Futebol/Pontuações	Futebol/Pontuações	O ObjectString/ TOPICSTR é utilizado sozinho.
Futebol	Pontuações	Futebol/Pontuações	Um caractere "/" é incluído no ponto de concatenação.
Futebol	/Pontuações	Futebol//Pontuações	Um 'nó vazio' é gerado entre as duas sequências. Isso é diferente de "Football/ Scores".
/Futebol	Pontuações	/Futebol/Pontuações	O tópico começa com um 'nó vazio'. Isso é diferente de "Football/ Scores".

O caractere "/" é considerado como um caractere especial, fornecendo estrutura para o nome completo do tópico em “Árvores de Tópicos” na página 78. O caractere "/" não deve ser usado por nenhuma outra razão, porque a estrutura da árvore de tópicos será afetada. O tópico "/Football" não é o mesmo que o tópico "Football".

Nota: Se você usar um objeto de tópico ao criar uma assinatura, o valor da sequência de tópicos do objeto do tópico será fixo na assinatura no momento da definição. Qualquer mudança subsequente para o objeto de tópico não afeta a sequência de tópicos para qual a assinatura é definida.

Caracteres curinga em sequência de tópicos

Os seguintes caracteres curinga são caracteres especiais:

- sinal de mais (+)
- sinal de número (#)
- asterisco (*)
- ponto de interrogação (?)

Caracteres curinga possuem significado especial quando utilizados apenas por uma assinatura. Esses caracteres não são considerados inválidos quando utilizados em outro lugar, no entanto, você deve estar seguro de que entende como eles são utilizados e que pode preferir não usar esses caracteres em suas sequências de tópicos ao publicar ou definir objetos do tópico.

Se você publicar em uma sequência de tópicos com # ou + combinados com outros caracteres (incluindo eles mesmos) dentro de um nível de tópico, a sequência de tópicos pode ser assinada com um esquema curinga.

Se você publicar em uma sequência de tópicos com # ou + como o único caractere entre dois caracteres /, a sequência de tópicos não pode ser assinada explicitamente por um aplicativo que usa o esquema de curingas MQSO_WILDCARD_TOPIC. Esta situação resulta no aplicativo obtendo mais publicações do que esperado.

Você não deve utilizar um caractere curinga na sequência de tópicos de um objeto do tópico definido. Se você fizer isso, o caractere será tratado como um caractere literal quando o objeto for utilizado por um publicador e como um caractere curinga quando utilizado por uma assinatura. Isso pode levar a confusões.

Fragmento do Código de Exemplo

Este fragmento de código extraído do exemplo de programa Exemplo 2: Publicador para um tópico de variável, combina um objeto do tópico com uma sequência de tópicos da variável:

```
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);
```

Árvores de Tópicos

Cada tópico que você definir é um elemento ou nó na árvore de tópicos. A árvore de tópicos pode estar vazia para iniciar ou conter tópicos que foram definidos anteriormente usando comandos MQSC ou PCF. Você pode definir um novo tópico usando os comandos create topic ou especificando o tópico pela primeira vez em uma publicação ou assinatura.

Embora você possa usar qualquer sequência de caracteres para definir uma sequência de tópicos do tópico, é recomendável escolher uma sequência de tópicos que cabe em uma estrutura de árvore hierárquica. Um cuidadoso design das sequências de tópicos e árvores de tópicos pode ajudá-lo com as seguintes operações:

- Assinar vários tópicos.
- Estabelecer políticas de segurança.

Embora você possa construir uma árvore de tópicos em uma estrutura linear, plana, é melhor construir uma árvore de tópicos em uma estrutura hierárquica com um ou mais tópicos de raiz. Para obter mais informações sobre o planejamento de segurança e tópicos, consulte Publicar/assinar segurança.

A Figura 18 na página 79 mostra um exemplo de uma árvore de tópicos com um tópico de raiz.

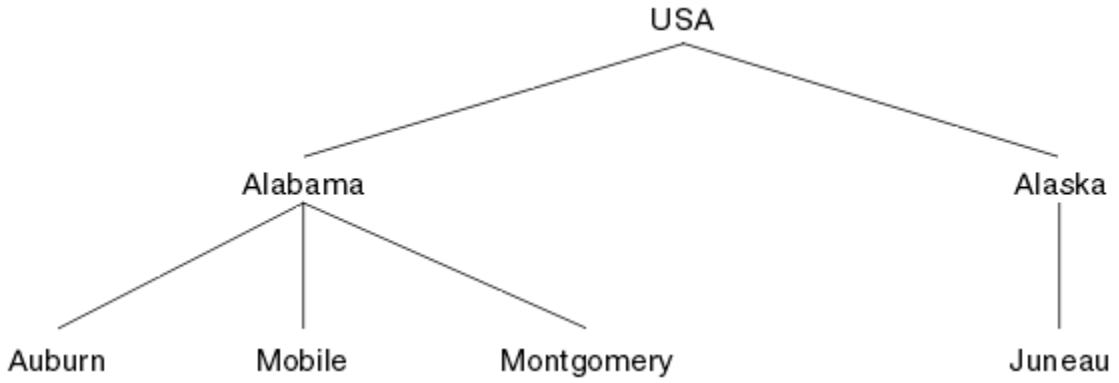


Figura 18. Exemplo de uma Árvore de Tópicos

Cada sequência de caracteres na figura representa um nó na árvore de tópicos. Uma sequência completa de tópicos é criada agregando nós de um ou mais níveis na árvore de tópicos. Os níveis são separados pelo caractere "/". O formato de uma sequência de tópicos totalmente especificada é: "root/level2/level3".

Os tópicos válidos na árvore de tópicos mostrada em Figura 18 na página 79 são:

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

Ao projetar sequências de tópicos e árvores de tópicos, lembre-se de que o gerenciador de filas não interpreta ou tenta derivar o significado, a própria sequência de tópicos. Ele simplesmente usa a sequência de tópicos para enviar mensagens selecionadas para assinantes desse tópico.

Os seguintes princípios se aplicam à construção e conteúdo de uma árvore de tópicos:

- Não há limite para o número de níveis em uma árvore de tópicos.
- Não há limite para o comprimento do nome de um nível em uma árvore de tópicos.
- Pode haver qualquer quantidade de nós de "raiz"; ou seja, pode haver qualquer quantidade de árvores de tópicos.

Tarefas relacionadas

Reduzindo o número de tópicos indesejados na árvore de tópicos

Objetos de Tópico Administrativo

Usando um objeto de tópico administrativo, é possível designar atributos específicos não padrão para os tópicos.

Figura 19 na página 80 mostra como um tópico de alto nível de `Sport` dividido em tópicos separados que abrangem diferentes esportes pode ser visualizado como uma árvore de tópicos:

Figura 19. Visualização de uma Árvore de Tópicos

Figura 20 na página 80 mostra como a árvore de tópicos pode ser dividida ainda mais, para separar os diferentes tipos de informações sobre cada esporte:

Figura 20. Árvore de Tópicos Estendida

Para criar a árvore de tópicos ilustrada, nenhum objeto de tópico administrativo precisa ser definido. Cada um dos nós nessa árvore é definido por uma sequência de tópicos criada em uma operação de publicação ou assinatura. Cada tópico na árvore herda seus atributos a partir do respectivo pai. Os atributos são herdados do objeto de tópico-pai, porque, por padrão, todos os atributos são configurados para ASPARENT. Neste exemplo, cada tópico tem os mesmos atributos que o tópico `Sport`. O tópico `Sport` não possui nenhum objeto de tópico administrativo e herda seus atributos de SYSTEM.BASE.TOPIC.

Observe que não é uma boa prática dar autoridade para usuários que não são mqm no nó de raiz da árvore de tópicos, que é SYSTEM.BASE.TOPIC, porque as autoridades são herdadas mas não podem ser restritas. Portanto, ao conceder autoridades neste nível, você estará concedendo autoridades para a árvore inteira. Você deve dar autoridade em um nível de tópico inferior na hierarquia.

Os objetos de tópico administrativo podem ser usados para definir atributos específicos para os nós específicos na árvore de tópicos. No exemplo a seguir, o objeto de tópico administrativo é definido para configurar a propriedade de assinaturas duráveis DURSUB do tópico de futebol para o valor `NO`:

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

A árvore de tópicos agora pode ser visualizada como:

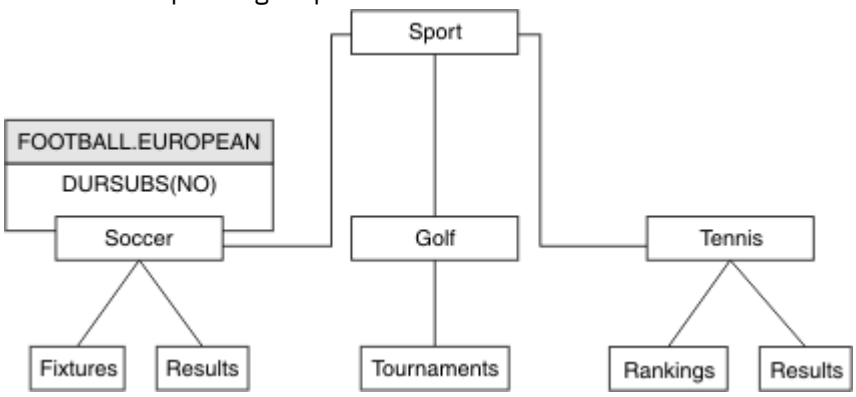


Figura 21. Visualização de um Objeto de Tópico Administrativo Associado ao Tópico Sport/Soccer

Quaisquer aplicativos subscritos para os tópicos sob Soccer na árvore ainda podem usar as sequência de tópicos que usaram antes que o objeto de tópico administrativo fosse incluído. No entanto, um aplicativo agora pode ser gravado para subscrever usando o nome do objeto FOOTBALL.EUROPEAN, em vez da sequência /Sport/Soccer. Por exemplo, para se subscrever em /Sport/Soccer/Results, um aplicativo pode especificar MQSD.ObjectName como FOOTBALL.EUROPEAN e MQSD.ObjectString como Results.

Com este recurso, é possível ocultar parte da árvore de tópicos dos desenvolvedores de aplicativos. Defina um objeto de tópico administrativo em um nó específico na árvore de tópicos e, em seguida, os desenvolvedores de aplicativo podem definir seus próprios tópicos como filhos do nó. Os desenvolvedores devem saber sobre o tópico pai, mas não sobre quaisquer outros nós na árvore pai.

Herdando Atributos

Se uma árvore de tópicos tiver vários objetos de tópico administrativo, cada objeto de tópico administrativo, por padrão, herdará seus atributos de seu tópico administrativo pai mais próximo. O exemplo anterior foi estendido no Figura 22 na página 81:

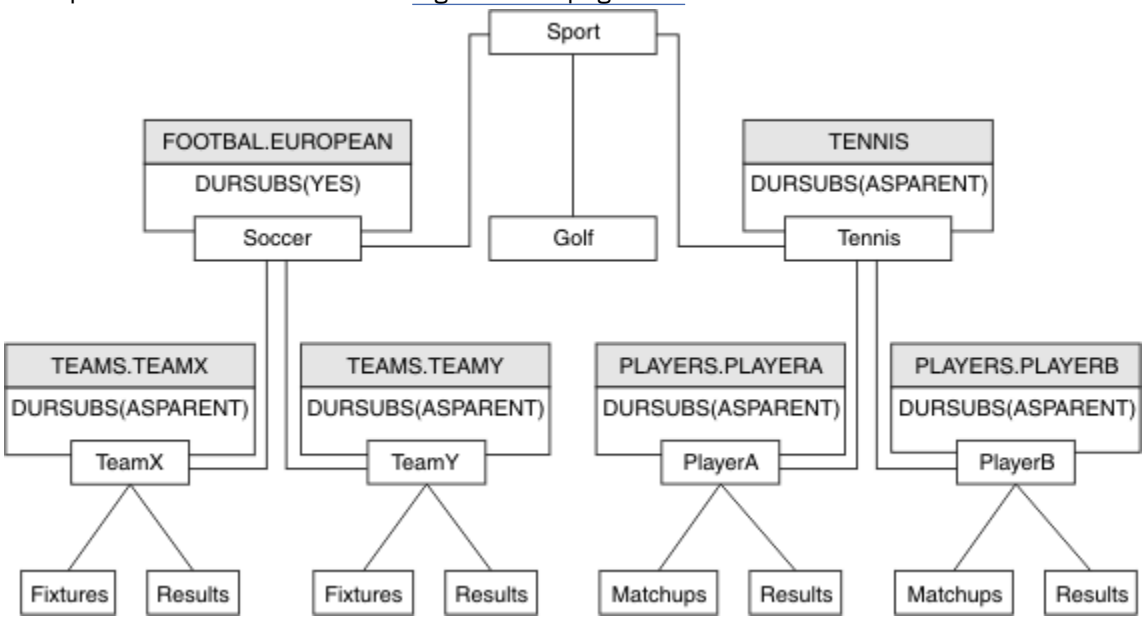


Figura 22. Árvore de Tópicos com Diversos Objetos de Tópico Administrativo

Por exemplo, use a herança para fornecer a todos os tópicos filhos de /Sport/Soccer a propriedade para que as assinaturas sejam não duráveis. Altere o atributo DURSUB de FOOTBALL.EUROPEAN para NO.

Este atributo pode ser configurado usando o seguinte comando:

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

Todos os objetos de tópico administrativo de tópicos filhos de Sport/Soccer possuem a propriedade DURSUB configurada para o valor padrão ASPARENT. Após alterar o valor da propriedade DURSUB de FOOTBALL.EUROPEAN para NO, os tópicos filhos de Sport/Soccer herdam o DURSUB valor da propriedade NO. Todos os tópicos filhos de Sport/Tennis herdam o valor de DURSUB do objeto SYSTEM.BASE.TOPIC .. SYSTEM.BASE.TOPIC tem o valor de YES.

Tentar tornar uma assinatura durável para o tópico Sport/Soccer/TeamX/Results falharia agora; no entanto, tentar tornar uma assinatura durável para Sport/Tennis/PlayerB/Results seria bem-sucedido.

Controlando o Uso de Curinga com a Propriedade WILDCARD

Use a propriedade MQSC **Topic** WILDCARD ou a propriedade PCF Topic `WildcardOperation` equivalente para controlar a entrega de publicações para aplicativos assinantes que usam nomes da sequência de tópicos curinga. A propriedade WILDCARD pode ter um dos dois valores possíveis:

CURINGA
O comportamento de assinaturas curingas com relação a este tópico.

PASSTHRU
As assinaturas feitas em um tópico curinga menos específico do que a sequência de tópicos neste objeto do tópico recebem as publicações feitas neste tópico e para sequências de tópicos mais específicas do que este tópico.

BLOCK
As assinaturas feitas em um tópico curinga menos específico do que a sequência de tópicos neste objeto do tópico não recebem as publicações feitas neste tópico ou nas sequências de tópicos mais específicas do que este tópico.

O valor deste atributo é usado quando as assinaturas são definidas. Se você alterar este atributo, o conjunto de tópicos coberto pelas assinaturas existentes não será afetado pela modificação. Este cenário também se aplica se a topologia for alterada quando os objetos do tópico forem criados ou excluídos; o conjunto de tópicos que corresponde às assinaturas criadas seguindo a modificação do atributo WILDCARD é criado usando a topologia modificada. Se você desejar forçar o conjunto de tópicos correspondente para ser reavaliado para assinaturas existentes, deve reiniciar o gerenciador de filas.

No exemplo, “Exemplo: Criar um Cluster de Publicação/Assinatura Sport” na página 85, é possível seguir as etapas para criar a estrutura da árvore de tópicos mostrada em Figura 23 na página 82.

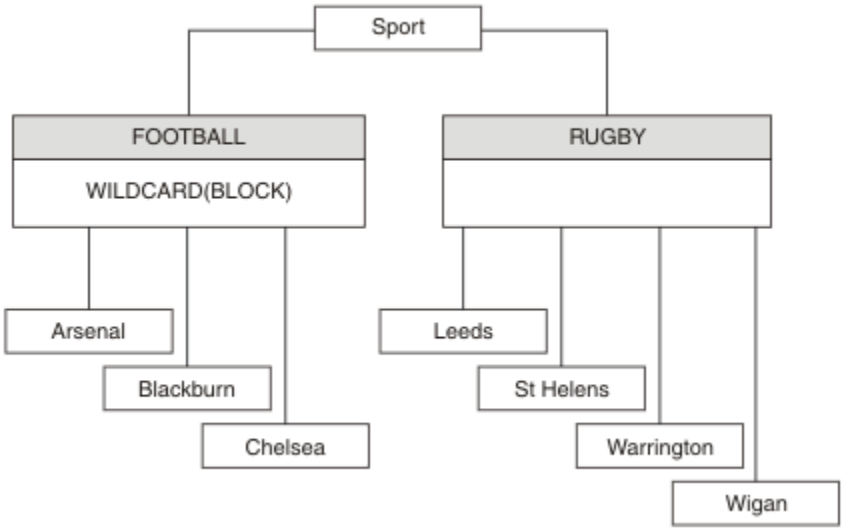


Figura 23. Uma árvore de tópicos que usa a propriedade WILDCARD, BLOCK

Um assinante que usa a sequência de tópicos curinga # recebe todas as publicações para o tópico `Sport` e subárvore `Sport/Rugby`. O assinante não recebe nenhuma publicação para a subárvore `Sport/Football`, porque o valor da propriedade `WILDCARD` do tópico `Sport/Football` é `BLOCK`.

PASSTHRU é a configuração padrão. É possível configurar o valor da propriedade PASSTHRU de `WILDCARD` para os nós na árvore `Sport`. Se os nós não tiverem o valor da propriedade `BLOCK` do `WILDCARD`, configurar PASSTHRU não vai alterar o comportamento observado pelos assinantes para os nós na árvore `Sports`.

No exemplo, crie assinaturas para ver como a configuração curinga afeta as publicações que são entregues; consulte Figura 27 na página 87. Execute o comando de publicação no Figura 30 na página 88 para criar algumas publicações.

```
pub QMA
```

Figura 24. Publicar para QMA

Os resultados são mostrados em Tabela 3 na página 83. Observe como a configuração do valor da propriedade BLOCK de WILDCARD, impede que as assinaturas com curingas recebam as publicações para os tópicos no escopo do curinga.

Tabela 3. Publicações Recebidas em QMA

Assinatura	Cadeia do tópico	Publicações Recebidas	Notas
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Todas as publicações para a subárvore Football bloqueadas por WILDCARD(BLOCK) em Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) em Sports/Football impede as assinaturas curingas em Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	WILDCARD padrão em Sports/Rugby não impede a assinatura curinga em Leeds.

Nota:

Imagine que uma assinatura tenha um curinga que corresponda a um objeto do tópico com o valor da propriedade BLOCK de WILDCARD. Se a assinatura também tiver uma sequência de tópicos à direita do curinga correspondente, a assinatura nunca receberá uma publicação. O conjunto de publicações que não estão bloqueadas são publicações para tópicos que são pais do curinga bloqueado. As publicações para os tópicos que são filhos do tópico com o valor da propriedade BLOCK são bloqueadas pelo curinga. Portanto, as sequências de tópicos de assinatura que incluem um tópico à direita do curinga nunca recebem qualquer publicação a ser correspondida.

Configurar o valor da propriedade WILDCARD para BLOCK não significa que você não possa se subscrever usando uma sequência de tópicos que inclua curingas. Essa assinatura é normal. A assinatura tem um tópico explícito que corresponde ao tópico com um objeto do tópico que possui um valor da propriedade BLOCK de WILDCARD. Ela usa os curingas para os tópicos que são pais ou filhos do tópico com o valor da propriedade BLOCK de WILDCARD. No exemplo em Figura 23 na página 82, uma assinatura, como Sports/Football/#, pode receber as publicações.

Curingas e Tópicos do Cluster

As definições do tópico do cluster são propagadas para cada gerenciador de filas em um cluster. Uma assinatura para um tópico de cluster em um gerenciador de filas em um cluster resulta no gerenciador de filas criando assinaturas de proxy. Uma assinatura de proxy é criada em cada outro gerenciador de filas no cluster. As assinaturas que usam sequências de tópicos que contêm curingas, combinados com tópicos de cluster, podem fornecer um comportamento difícil de ser previsto. O comportamento é explicado no seguinte exemplo.

Na configuração do cluster para o exemplo, “Exemplo: Criar um Cluster de Publicação/Assinatura Sport” na página 85, QMB tem o mesmo conjunto de assinaturas que QMA, embora QMB não tenha recebido nenhuma publicação após o publicador publicado para QMA, consulte Figura 24 na página 83. Embora os tópicos Sports/Football e Sports/Rugby sejam tópicos de cluster, as assinaturas definidas em fullsubs.tst não fazem referência a um tópico de cluster. Nenhuma assinatura de proxy é propagada de QMB para QMA. Sem as assinaturas de proxy, nenhuma publicação para QMA é encaminhada para QMB.

Algumas das assinaturas, como Sports/#/Leeds, podem parecer fazer referência a um tópico do cluster, neste caso, Sports/Rugby. A assinatura Sports/#/Leeds resolve, na realidade, para o objeto de tópico SYSTEM.BASE.TOPIC.

A regra para resolver o objeto do tópico referenciado por uma assinatura, como Sports/#/Leeds, é a seguinte. Truncar a sequência de tópicos para o primeiro curinga. Varra para a esquerda da sequência de tópicos procurando pelo primeiro tópico que tem um objeto de tópico administrativo associado. O objeto do tópico pode especificar um nome de cluster ou definir um objeto de tópico local. No exemplo, Sports/#/Leeds, a sequência de tópicos após o truncamento é Sports, que não possui nenhum objeto do tópico e, portanto, Sports/#/Leeds herda de SYSTEM.BASE.TOPIC, que é um objeto do tópico local.

Para ver como a assinatura nos tópicos em cluster pode alterar a maneira como a propagação de curinga funciona, execute o script em lote, upsubs.bat. O script limpa as filas de assinatura e inclui as assinaturas de tópico de cluster em fullsubs.tst. Execute puba.bat novamente para criar um lote de publicações; consulte Figura 24 na página 83.

Tabela 4 na página 84 mostra o resultado de incluir duas novas assinaturas no mesmo gerenciador de filas no qual as publicações foram publicadas. O resultado é o esperado, as novas assinaturas recebem uma publicação cada e os números de publicações recebidas pelas outras assinaturas ficam inalterados. Ocorrem os resultados inesperados no outro gerenciador de filas do cluster; consulte Tabela 5 na página 85.

Tabela 4. Publicações Recebidas em QMA

Assinatura	Cadeia do tópico	Publicações Recebidas	Notas
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Todas as publicações para a subárvore Football bloqueadas por WILDCARD(BLOCK) em Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) em Sports/Football impede as assinaturas curingas em Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	WILDCARD padrão em Sports/Rugby não impede a assinatura curinga em Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal recebe uma publicação porque a assinatura não tem um curinga.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds receberia uma publicação em qualquer evento.

Tabela 5 na página 85 mostra os resultados de incluir as duas novas assinaturas no QMB e publicar em QMA. A rechamada desse QMB não recebeu nenhuma publicação sem essas duas novas assinaturas. Conforme esperado, as duas novas assinaturas recebem as publicações, porque Sports/FootBall e Sports/Rugby são tópicos de cluster. O QMB encaminhou assinaturas de proxy para Sports/Football/Arsenal e Sports/Rugby/Leeds para QMA, que então enviou as publicações para QMB.

O resultado inesperado é que duas assinaturas Sports/# e Sports/#/Leeds que não receberam anteriormente nenhuma publicação, agora recebem as publicações. A razão é que as publicações Sports/Football/Arsenal e Sports/Rugby/Leeds encaminhadas para QMB para as outras assinaturas agora estão disponíveis para qualquer assinante conectado ao QMB. Consequentemente as assinaturas para os tópicos locais Sports/# E Sports/#/Leeds recebem a publicação Sports/Rugby/Leeds. Sports/#/Arsenal continua não recebendo uma publicação, porque Sports/Football tem seu valor de propriedade WILDCARD configurado como BLOCK.

Tabela 5. Publicações Recebidas em QMB			
Assinatura	**Cadeia do tópico**	**Publicações Recebidas**	**Notas**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*Todas as publicações para a subárvore Football bloqueadas por WILDCARD(BLOCK) em Sports/Football*
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) em Sports/Football impede as assinaturas curingas em Arsenal
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*WILDCARD padrão no Sports/Rugby não evita a assinatura curinga em Leeds.*
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal recebe uma publicação porque a assinatura não tem um curinga.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds receberia uma publicação em qualquer evento.

Na maioria dos aplicativos, é desejável para uma assinatura influenciar o comportamento de outra assinatura. Um uso importante da propriedade WILDCARD com valor BLOCK é fazer com que as assinaturas para a mesma sequência de tópicos que contém curingas se comportem uniformemente. Indica se a assinatura está no mesmo gerenciador de filas que o publicador ou um diferente, os resultados da assinatura são os mesmos.

Curingas e Fluxos

Para um novo aplicativo gravado para a API de publicação/assinatura, o efeito é que uma assinatura para * não recebe nenhuma publicação. Para receber todas as publicações de Sports você deve se subscrever a Sports/* ou Sports/#, de maneira semelhante para publicações Business.

O comportamento de um aplicativo de publicação / assinatura enfileirado existente não muda quando o broker de publicação / assinatura é migrado para uma versão posterior do IBM MQ. A propriedade **StreamName** nos comandos **Publish**, **Register Publisher**ou **Subscriber** é mapeada para o nome do tópico para o qual o fluxo foi migrado..

Curingas e Pontos de Assinatura

Para um novo aplicativo gravado para a API de publicação/assinatura, o efeito da migração é que uma assinatura para * não receba nenhuma publicação. Para receber todas as publicações de Sports você deve se subscrever a Sports/* ou Sports/#, de maneira semelhante para publicações Business.

O comportamento de um aplicativo de publicação / assinatura enfileirado existente não muda quando o broker de publicação / assinatura é migrado para uma versão posterior do IBM MQ. A propriedade **SubPoint** nos comandos **Publish**, **Register Publisher**ou **Subscriber** é mapeada para o nome do tópico para o qual a assinatura foi migrada..

Exemplo: Criar um Cluster de Publicação/Assinatura Sport

As etapas que seguem a criação de um cluster, CL1, com quatro gerenciadores de filas: dois repositórios completos, CL1A e CL1B, e dois repositórios parciais, QMA e QMB. Os repositórios completos são usados para reter apenas as definições de cluster. QMA designado ao host de tópico do cluster. As assinaturas duráveis são definidas em QMA e QMB.

Nota: O exemplo é codificado para Windows. Você deve recodificar Create qmgrs.bat e create pub.bat para configurar e testar o exemplo em outras plataformas.

1. Crie os arquivos de script.
   a. Create topics.tst
   b. Create wildsubs.tst
   c. Create fullsubs.tst
   d. Create qmgrs.bat
   e. create pub.bat
2. Execute Crie qmgrs.bat para criar a configuração.

```
qmgrs
```

Crie os tópicos em Figura 23 na página 82. O script na figura 5 cria os tópicos de cluster `Sports/Football` e `Sports/Rugby`

Nota: A opção REPLACE não substitui as propriedades `TOPICSTR` de um tópico. `TOPICSTR` é uma propriedade que é proveitosamente variada no exemplo para testar as diferentes árvores de tópicos. Para alterar os tópicos, exclua primeiro o tópico.

```
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. Excluir e Criar os Tópicos: topics.tst

Nota: Exclua os tópicos, porque o REPLACE não substitui as sequências de tópicos.

Crie assinaturas com curingas. Os curingas correspondentes dos tópicos com objetos do tópico no Figura 23 na página 82. Crie uma fila para cada assinatura. As filas são limpas e as assinaturas excluídas quando o script não é executado ou reexecutado.

Nota: A opção REPLACE não substitui as propriedades `TOPICOBJ` ou `TOPICSTR` de uma assinatura. `TOPICOBJ` ou `TOPICSTR` são propriedades proveitosamente variadas no exemplo para testar diferentes assinaturas. Para alterá-las, exclua primeiro a assinatura.

```
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. Criar Assinaturas Curinga: wildsubs.tst

Crie as assinaturas que fazem referência aos objetos do tópico de cluster.

Nota:

O delimitador, `/`, é inserido automaticamente entre a sequência de tópicos referenciada por `TOPICOBJ` e a sequência de tópicos definida por `TOPICSTR`.

A definição, `DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL)` cria a mesma assinatura. `TOPICOBJ` é usado como uma maneira rápida de fazer referência à sequência de tópicos que você já definiu. A assinatura, quando criada, não se refere mais ao objeto do tópico.

```
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. Excluir e Criar Assinaturas: fullsubs.tst

Crie um cluster com dois repositórios. Crie dois repositórios parciais para publicar e subscrever. Execute novamente o script para excluir tudo e inicie novamente. O script também cria a hierarquia de tópico e as assinaturas curingas iniciais.

Nota:

Em outras plataformas, grave um script semelhante ou digite todos os comandos. Usar um script facilita a exclusão de tudo e inicia novamente com uma configuração idêntica.

```
@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. Criar Gerenciadores de Filas: qmgrs.bat

Atualize a configuração incluindo as assinaturas nos tópicos do cluster.

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

Figura 29. Atualizar Assinaturas: upsubs.bat

Execute `pub.bat`, com um gerenciador de filas como um parâmetro, para publicar as mensagens que contêm a sequência de tópicos de publicação. `Pub.bat` usa o programa de amostra **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. Publicar: pub.bat

Fluxos e Tópicos

A publicação/assinatura enfileirada tem o conceito de um fluxo de publicação que não existe no modelo de publicação/assinatura integrada. Na publicação/assinatura enfileirada, os fluxos fornecem uma maneira de separar o fluxo de informações para os diferentes tópicos. Um fluxo é implementado como um tópico de nível superior que pode ser mapeado para um identificador de tópico diferente administrativamente.

O fluxo padrão `SYSTEM.BROKER.DEFAULT.STREAM` é configurado automaticamente para todos os corretores e gerenciadores de filas em uma rede e nenhuma configuração adicional é necessária para usar o fluxo padrão. Pense no fluxo padrão como um espaço de tópico padrão sem nome. Os tópicos publicados para o fluxo padrão estão imediatamente disponíveis para todos os gerenciadores de filas conectados, com a publicação / assinatura em fila ativada. Os fluxos nomeados são como espaços de tópico, nomeados, separados. O fluxo nomeado deve ser definido em cada broker no qual é usado.

Se os publicadores e assinantes estiverem em diferentes gerenciadores de fila, depois que os corretores forem conectados na mesma hierarquia do corretor, nenhuma configuração adicional será necessária para as publicações e assinaturas fluírem entre eles. A mesma interoperabilidade funciona em reverso, também.

Fluxos Nomeados

Um designer de solução trabalhando com o modelo de programação de publicação/assinatura enfileirada, pode decidir colocar todas as publicações de esportes em um fluxo nomeado denominado `Sport`. Para que o fluxo esteja disponível para um gerenciador de filas que é executado no IBM MQ com a publicação / assinatura em fila ativada, o fluxo deve ser incluído manualmente.

Aplicativos de publicação/assinatura enfileirada que assinam `Soccer/Results` no fluxo `Sport` trabalham sem mudança. Aplicativos de publicação / assinatura integrados que assinam o tópico `Sport` usando `MQSUB`e fornecendo a sequência de tópicos `Soccer/Results` também recebem as mesmas publicações.

A tarefa de incluir um fluxo está descrita no tópico Incluindo um Fluxo. Pode ser necessário incluir fluxos manualmente por duas razões.

1. Continue a desenvolver seus aplicativos de publicação/assinatura enfileirada que estão executando em gerenciadores de filas de versões mais recentes, ao invés de fazer a migração de aplicativos para a interface MQI de publicação/assinatura integrada.
2. O mapeamento padrão de fluxos para os tópicos conduz a uma "colisão" no espaço de tópico e as publicações em um fluxo possuem a mesma sequência de tópicos que as publicações de algum lugar.

Autoridade

Por padrão, na raiz da árvore de tópicos existe diversos objetos do tópico: `SYSTEM.BASE.TOPIC`, `SYSTEM.BROKER.DEFAULT.STREAM` e `SYSTEM.BROKER.DEFAULT.SUBPOINT`. Autoridades (por exemplo, para publicação ou assinatura) são determinadas pelas autoridades no `SYSTEM.BASE.TOPIC` ; Quaisquer autoridades no `SYSTEM.BROKER.DEFAULT.STREAM` ou `SYSTEM.BROKER.DEFAULT.SUBPOINT` são ignorados Se um dos `SYSTEM.BROKER.DEFAULT.STREAM` ou `SYSTEM.BROKER.DEFAULT.SUBPOINT` for excluído e recriado com uma sequência de tópicos não vazia, as autoridades definidas nesses objetos serão usadas da mesma maneira que um objeto de tópico normal..

Mapeando Entre Fluxos e Tópicos

Um fluxo de publicação / assinatura enfileirado é imitado no IBM MQ criando uma fila e fornecendo o mesmo nome que o fluxo. Às vezes, a fila é chamada de fila de fluxo, porque é como aparece para aplicativos de publicação/assinatura enfileirada. A fila é identificada para o mecanismo publicar/assinar, incluindo-a na lista de nomes especial denominada `SYSTEM.QPUBSUB.QUEUE.NAMELIST`. É possível incluir quantos fluxos precisar, incluindo as filas adicionais especiais na lista de nomes. Finalmente, você precisa incluir os tópicos, com os mesmos nomes que os fluxos e as mesmas sequências de tópicos que o nome do fluxo, de modo que você possa publicar e subscrever-se nos tópicos.

No entanto, em circunstâncias excepcionais, é possível fornecer aos tópicos correspondentes aos fluxos quaisquer sequências de tópicos que você escolher quando definir os tópicos. O propósito da sequência de tópicos é fornecer ao tópico um nome exclusivo no espaço de tópico. Geralmente, o nome do fluxo serve a esse propósito perfeitamente. Às vezes, um nome do fluxo e um nome de tópico existente se conflitam. Para resolver o problema, escolha outra sequência de tópicos para o tópico associado ao fluxo.. Escolha qualquer sequência de tópico, assegurando que ele seja exclusivo.

A sequência de tópicos definida na definição de tópicos recebe o prefixo da maneira normal para a sequência de tópicos fornecida pelos publicadores e assinantes usando as chamadas `MQOPEN` ou `MQSUB` MQI. Os aplicativos que se referem aos tópicos usando os objetos de tópico não são afetados pela opção da sequência de tópicos do prefixo - que é o motivo pelo qual é possível escolher qualquer sequência de tópicos que mantenha as publicações exclusivas no espaço de tópico.

O remapeamento de diferentes fluxos em diferentes tópicos depende dos prefixos usados para as sequências de tópicos serem exclusivos, para separar um conjunto de tópicos completamente do outro. Você deve definir uma convenção de nomenclatura de tópico universal que esteja rigorosamente junta para que o mapeamento funcione.

No IBM MQ, você usa o mecanismo de prefixação para remapear uma sequência de tópicos para outro local no espaço de tópico

Nota: Ao excluir um fluxo, exclua todas as assinaturas no fluxo primeiro. Essa ação é mais importante se alguma dessas assinaturas se originarem de outros corretores na hierarquia do corretor.

Tópicos e Pontos de Assinatura

Pontos de assinatura nomeados são emulados por tópicos e objetos de tópico.

Para incluir pontos de assinatura manualmente, consulte Incluindo um ponto de assinatura.

Pontos de assinatura no IBM MQ

O IBM MQ mapeia pontos de assinatura para diferentes espaços de tópico na árvore de tópicos do IBM MQ. Os tópicos nas mensagens de comando sem um ponto de assinatura são mapeados inalterados para a raiz da árvore de tópicos do IBM MQ e herdam as propriedades do `SYSTEM.BASE.TOPIC`.

As mensagens de comando com um ponto de assinatura estão em processamento usando a lista de objetos de tópico no `SYSTEM.QPUBSUB.SUBPOINT.NAMELIST`. O nome do ponto de assinatura na mensagem de comando é correspondida em uma sequência de tópicos para cada um dos objetos de tópico na lista. Se uma correspondência for localizada, o nome do ponto de assinatura será pré-anexado, como um nó de tópico, para a sequência de tópicos. O tópico herda suas propriedades a partir do objeto do tópico associado localizado no `SYSTEM.QPUBSUB.SUBPOINT.NAMELIST`.

O efeito de usar os pontos de assinatura é criar um espaço de tópico separado para cada ponto de assinatura. O espaço de tópico é enraizado em um tópico que tem o mesmo nome que o ponto de assinatura. Os tópicos em cada espaço de tópico herda suas propriedades a partir do objeto de tópico com o mesmo nome que o ponto de assinatura.

Quaisquer propriedades não definidas no objeto do tópico correspondente são herdadas normalmente do `SYSTEM.BASE.TOPIC`.

Os aplicativos de publicação/assinatura enfileirados, usando os cabeçalhos da mensagem `MQRFH2`, continuam trabalhando configurando a propriedade **`SubPoint`** nas mensagens de comando `Publish` ou `Register subscriber`. O ponto de assinatura é combinado com a sequência de tópicos na mensagem de comando e o tópico resultante processado como qualquer outro.

Aplicativos IBM MQ não são afetados por pontos de assinatura. Se um aplicativo usar um tópico que herda informações de um dos objetos do tópico correspondentes, esse aplicativo irá interoperar com um aplicativo enfileirado usando o ponto de assinatura correspondente.

exemplo

Um aplicativo de publicação/assinatura do WebSphere Message Broker existente (agora conhecido como IBM Integration Bus) que foi migrado para IBM MQ criou dois objetos do tópico, `GBP` e `USD`, com as sequências de tópicos correspondentes `'GBP'` e `'USD'`.

Os editores existentes para o tópico `NYSE/IBM/SPOT`, migrados para execução no IBM MQ e que usam o ponto de assinatura `USD`, criam publicações sobre o tópico `USD/NYSE/IBM/SPOT`. Da mesma forma, os assinantes existentes do `NYSE/IBM/SPOT`, que usam o ponto de assinatura do `USD`, criam assinaturas para o `USD/NYSE/IBM/SPOT`.

Assine o preço spot em dólar em um programa de publicação / assinatura IBM MQ chamando `MQSUB`. Crie uma assinatura usando o objeto tópico do `USD` e a sequência de tópicos do `'NYSE/IBM/SPOT'`, conforme ilustrado no 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. Configure o atributo CLUSTER dos objetos de tópico USD e GBP no host do tópico do cluster.
2. Exclua todas as cópias dos objetos do tópico USD e GBP em outros gerenciadores de filas no cluster.
3. Certifique-se de que o USD e o GBP estejam definidos no SYSTEM.QPUBSUB.SUBPOINT.NAMELIST em cada gerenciador de filas no cluster.

Exemplo da Configuração de um Único Gerenciador de Filas de Publicação/Assinatura

Figura 31 na página 91 ilustra uma configuração de publicação/assinatura do gerenciador de filas único e básico. O exemplo mostra a configuração para um serviço de notícias, onde as informações estão disponíveis a partir de publicadores sobre vários tópicos:

- Publicador 1 está publicando informações sobre resultados de esportes usando um tópico de Esporte
- Publicador 2 está publicando informações sobre os preços de ações usando um tópico de Estoque
- Publicador 3 está publicando informações sobre revisões de filme usando um tópico de Filmes, e sobre listagens de televisão usando um tópico de TV

Três assinantes registraram interesse em tópicos diferentes, portanto, o gerenciador de filas os envia as informações em que estão interessados:

- Assinante 1 recebe os resultados de esporte e os preços de ação
- Assinante 2 recebe as revisões de filmes
- Assinante 3 recebe os resultados de esporte

Nenhum dos assinantes registraram interesse nas listagens de televisão, portanto elas não são distribuídas.

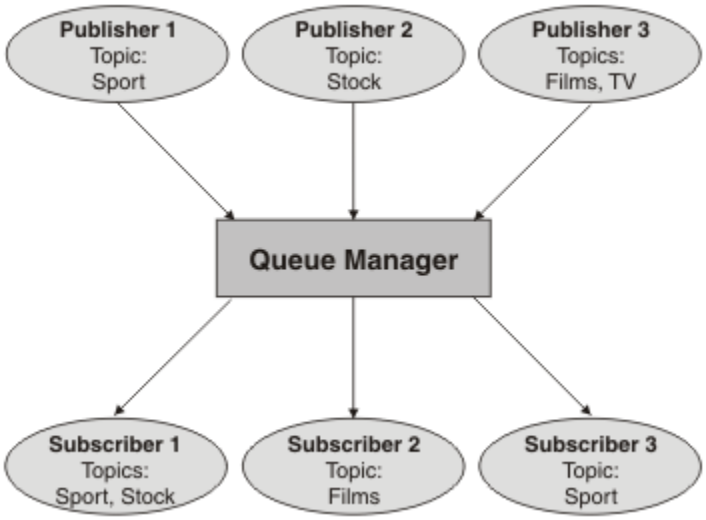


Figura 31. Exemplo de Gerenciador de Filas Único de Publicação/Assinatura

Redes publicar/assinar distribuídas

Cada gerenciador de filas corresponde mensagens publicadas em um tópico com as assinaturas criadas localmente que assinaram esse tópico. É possível configurar uma rede de gerenciadores de filas para que as mensagens publicadas por um aplicativo conectado a um gerenciador de filas sejam entregues a assinaturas correspondentes criadas em outros gerenciadores de filas na rede. Isso requer configuração adicional sobre canais simples entre gerenciadores de filas.

Uma configuração publicar/assinar distribuída é um conjunto de gerenciadores de filas conectados entre si. Todos os gerenciadores de filas podem estar no mesmo sistema físico ou podem ser distribuídos entre vários sistemas físicos. Ao conectar gerenciadores de filas, os assinantes podem assinar um gerenciador de filas e receber mensagens que foram inicialmente publicadas em outro gerenciador de filas. Para ilustrar isso, a figura a seguir inclui um segundo gerenciador de filas na configuração descrita em "Exemplo da Configuração de um Único Gerenciador de Filas de Publicação/Assinatura" na página 91.

- Gerenciador de filas 2 é usado pelo Publicador 4 para publicar as informações da previsão do tempo climático, usando um tópico de Tempo Climático e as informações sobre as condições de tráfego nas principais rodovias, usando um tópico de Tráfego.
- Assinante 4 também usa este gerenciador de filas e subscreve às informações sobre as condições de tráfego usando o tópico Tráfego.
- Assinante 3 também assina às informações sobre as condições climáticas, mesmo se usar um gerenciador de filas diferente do publicador. Isso é possível porque os gerenciadores de filas estão vinculados entre si.

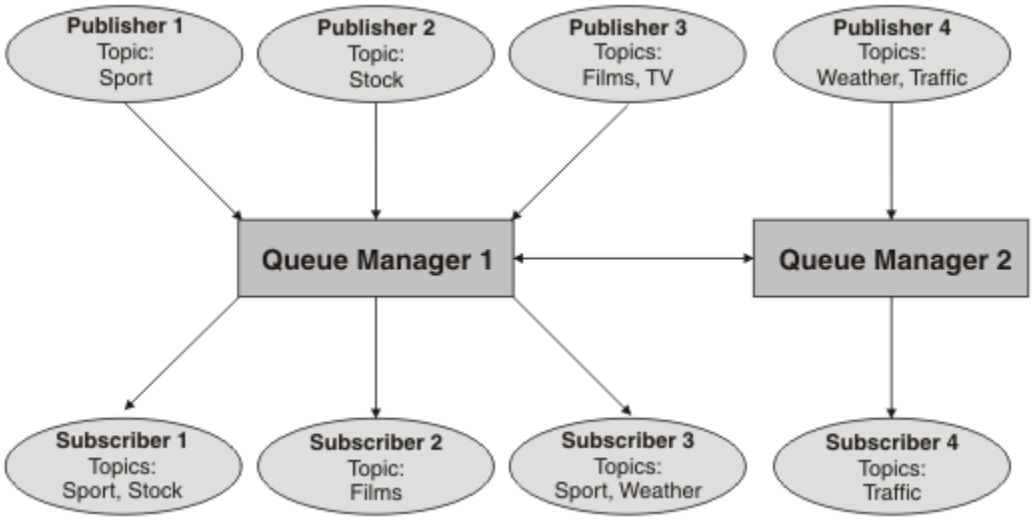


Figura 32. Exemplo de Publicação/Assinatura com Dois Gerenciadores de Filas

É possível conectar gerenciadores de filas manualmente em uma hierarquia de pai e filho ou criar um cluster de publicação/assinatura e deixar que o IBM MQ defina a maioria dos detalhes da conexão para você. Também é possível usar duas topologias em combinação, por exemplo, juntando vários clusters em uma hierarquia.

Visão geral de clusters publicar/assinar

Um cluster publicar/assinar é um cluster padrão com um ou mais objetos de tópico incluídos no cluster. Ao definir um objeto do tópico administrativo em qualquer gerenciador de filas em um cluster e tornar esse objeto do tópico em cluster especificando um nome de cluster, os publicadores e assinantes desse tópico podem conectar-se a qualquer um dos gerenciadores de filas no cluster e as mensagens publicadas são roteadas para os assinantes em canais de cluster entre gerenciadores de filas.

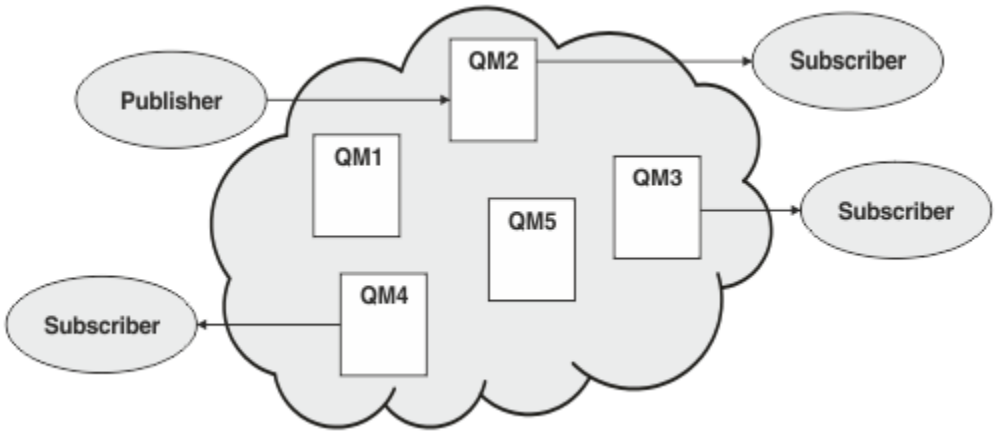


Figura 33. Cluster publicar/assinar

Há duas maneiras de configurar como as mensagens publicar/assinar são roteadas em um cluster:

- roteamento direto
- roteamento de host de tópico

Ao configurar um tópico em cluster roteado diretamente, as mensagens publicadas em um gerenciador de filas são enviadas diretamente desse gerenciador de filas para cada assinatura em qualquer outro gerenciador de filas no cluster. Isso pode fornecer o caminho mais direto para publicações, mas não resulta em todos os gerenciadores de filas em um cluster se tornando cientes de todos os outros gerenciadores de filas, cada um possivelmente tendo canais de cluster estabelecidos entre eles.

Ao usar o roteamento de host de tópico, as mensagens publicadas em um gerenciador de filas são enviadas de lá para um gerenciador de filas que hospeda uma definição do objeto do tópico administrado. O *gerenciador de filas do host de tópico* roteia a mensagem para cada assinatura em qualquer outro gerenciador de filas no cluster. Se os publicadores ou assinantes não forem localizados nos gerenciadores de filas de host de tópico, isso resulta em uma rota mais longa para publicações. No entanto, o benefício é que somente os gerenciadores de filas de host de tópico se tornam cientes de todos os outros gerenciadores de filas no cluster e, possivelmente, têm canais de cluster estabelecidos com eles.

Para obter mais informações, consulte “Publicar/assinar clusters” na página 95.

Visão Geral de hierarquias publicar/assinar

Uma hierarquia publicar/assinar é um conjunto de gerenciadores de filas conectados por canais em uma estrutura hierárquica. Cada gerenciador de filas identifica seu gerenciador de filas *pai*, conforme descrito em Conectando um gerenciador de filas a uma hierarquia publicar/assinar.

Os publicadores e assinantes de um tópico podem se conectar a qualquer gerenciador de filas na hierarquia, e as mensagens fluem entre eles usando a conectividade do gerenciador de filas hierárquico.

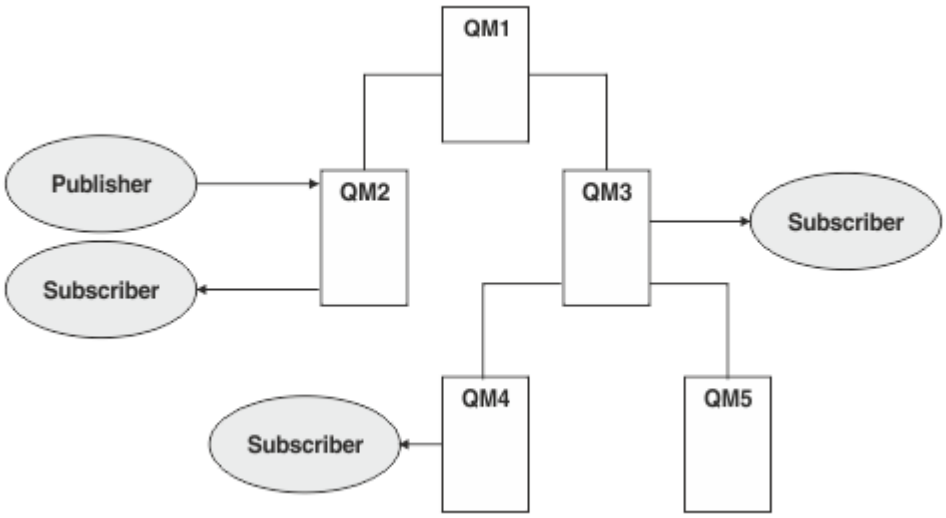


Figura 34. Hierarquia publicar/assinar

Na figura anterior, as publicações entregues aos assinantes em QM3 e QM4 foram roteadas de QM2 para QM1 e, em seguida, para QM3 e, por último, QM4.

As hierarquias fornecem controle direto sobre os relacionamentos entre cada gerenciador de filas na hierarquia. Isso permite um controle de baixa granularidade sobre o roteamento de mensagens de publicadores para assinantes e é útil principalmente ao rotear entre redes do gerenciador de filas conectividade restrita. É necessário considerar especialmente a disponibilidade e capacidade de cada gerenciador de filas por meio do qual uma mensagem é roteada em seu trajeto do publicador para assinantes.

Para obter mais informações, consulte “Hierarquias de Publicação/Assinatura” na página 97.

Distribuição de publicações entre gerenciadores de filas

Além das opções de roteamento, há duas abordagens para distribuir publicações em uma rede de gerenciadores de filas:

- Enviar publicações somente de um gerenciador de filas para os gerenciadores de filas que hospedam atualmente uma assinatura para essa publicação.
- Enviar cada publicação para todos os gerenciadores de filas e deixar que eles façam a correspondência dela com suas assinaturas.

A opção anterior resulta em mensagens sendo enviadas somente onde necessário, mas requer um nível de conhecimento de assinatura para ser compartilhado entre gerenciadores de filas. A última opção não requer o compartilhamento de conhecimento de assinatura, mas pode resultar no envio desnecessário de mensagens de publicação entre gerenciadores de filas.

Por padrão, o IBM MQ usa o método anterior, no qual as publicações são enviadas somente para gerenciadores de filas que possuam assinaturas. O conhecimento assinatura é propagado entre os gerenciadores de filas na forma de *assinaturas de proxy*. Isso depende da distribuição e do tempo de vida de assinaturas, e da frequência de publicações, com relação a qual é o mais eficiente para uso em uma topologia publicar/assinar distribuída. Consulte Desempenho de assinatura em redes publicar/assinar.

Conceitos relacionados

“Árvores de Tópicos” na página 78
Cada tópico que você definir é um elemento ou nó na árvore de tópicos. A árvore de tópicos pode estar vazia para iniciar ou conter tópicos que foram definidos anteriormente usando comandos MQSC ou PCF. Você pode definir um novo tópico usando os comandos create topic ou especificando o tópico pela primeira vez em uma publicação ou assinatura.

Cenários de Hierarquia de Publicação/Assinatura

Tarefas relacionadas

Projetando clusters publicar/assinar

Publicar/assinar clusters

Um cluster publicar/assinar é um cluster padrão de gerenciadores de filas interconectados, nos quais as publicações são automaticamente movidas de aplicativos de publicação para assinaturas que existem em qualquer um dos gerenciadores de filas no cluster. Há duas opções para rotear publicações em um cluster de publicação/assinatura: *roteamento direto* e *roteamento de host de tópico*. O roteamento escolhido depende do tamanho e de padrões de atividade esperados para seu cluster.

Um cluster que é usado para o sistema de mensagens de publicar/assinar não é diferente de um cluster do IBM MQ padrão. Como tal, os gerenciadores de filas no cluster de publicação/assinatura podem existir em computadores fisicamente separados e cada par de gerenciadores de filas é automaticamente conectado junto por canais de cluster quando necessário. Para obter informações adicionais, consulte Clusters.

Para configurar um cluster padrão de gerenciadores de filas para sistemas de mensagens de publicação/assinatura, você define um ou mais objetos de tópico administrado em um gerenciador de filas no cluster. Para tornar o tópico um tópico de cluster, você configura a propriedade **CLUSTER** com o nome do cluster. Ao fazer isso, qualquer tópico usado por um publicador ou assinante nesse ponto ou abaixo na árvore de tópicos é compartilhado entre todos os gerenciadores de filas no cluster, e as mensagens publicadas em uma ramificação de cluster da árvore de tópicos são automaticamente roteadas para assinaturas em outros gerenciadores de filas no cluster.

Somente uma cópia de cada mensagem é enviada entre o gerenciador de filas do publicador e cada um dos outros gerenciadores de filas, independentemente do número de assinantes para a mensagem no gerenciador de filas de destino. Na chegada a um gerenciador de filas com uma ou mais assinaturas, a mensagem é duplicada em todas as assinaturas.

Qualquer gerenciador de filas que se une ao cluster automaticamente se torna ciente dos tópicos em cluster e publicadores e assinantes nesse gerenciador de filas participam automaticamente do cluster.

Atividades que são em cluster de publicação/assinatura também podem ocorrer em um cluster de publicação/assinatura, trabalhando com as sequências de tópicos que não estão em um objeto de tópico em cluster.

Há duas opções para rotear publicações em um cluster de publicação/assinatura: *roteamento direto* e *roteamento de host de tópico*. Para escolher o roteamento de mensagens para usar dentro do cluster, você configura a propriedade **CLROUTE** no objeto de tópico administrado para um dos valores a seguir:

- **DIRECT**
- **TOPICHOST**

Por padrão, o roteamento de tópico é **DIRECT** Ao configurar um tópico de cluster roteado diretamente em um gerenciador de filas, todos os gerenciadores de filas no cluster ficam cientes de todos os outros gerenciadores de filas no cluster. Ao executar operações de publicação e assinatura, cada gerenciador de filas pode se conectar diretamente a qualquer outro gerenciador de filas no cluster.

A partir de IBM MQ 8.0, é possível configurar o roteamento de tópico como **TOPICHOST**. Quando você usa o roteamento de host de tópico, todos os gerenciadores de filas no cluster ficam cientes dos gerenciadores de filas do cluster que hospedam a definição de tópico roteado (ou seja, os gerenciadores de filas nos quais você definiu o objeto do tópico). Ao executar operações de publicação e assinatura, os gerenciadores de filas no cluster se conectam apenas a estes gerenciadores de filas do host de tópico, e não diretamente uns aos outros. Os gerenciadores de filas do host de tópico são responsáveis pelas publicações de roteamento a partir de gerenciadores de filas em que as publicações são publicadas para gerenciadores de filas com assinaturas correspondentes.

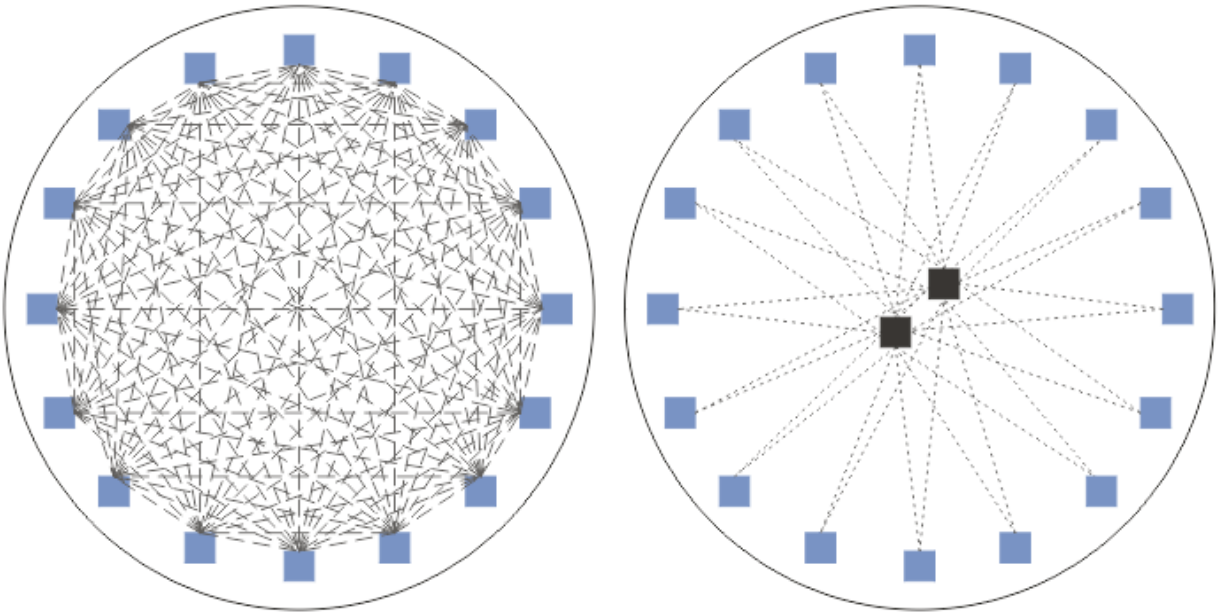


Figura 35. Roteamento Direto e Roteamento de Host de Tópico

Uma visão geral de roteamento direto

Quando um objeto do tópico administrado está configurado para roteamento direto, o objeto do tópico precisa somente ser definido em um dos gerenciadores de filas no cluster para que todos os gerenciadores de filas tomem conhecimento dele. A opção de gerenciador de filas no qual o tópico é definido não afeta o comportamento do sistema de mensagens publicar/assinar para o tópico.

Cada mensagem flui diretamente do gerenciador de filas do publicador para cada assinatura nos outros gerenciadores de filas no cluster, não passando por nenhum gerenciador de filas intermediário.

Por padrão, as mensagens são enviadas somente para outros gerenciadores de filas no cluster que hospedam uma ou mais assinaturas.

- Isso depende de cada gerenciador de filas informar diretamente todos os outros gerenciadores de filas no cluster de todos os tópicos que atualmente possuem uma ou mais assinaturas nele. Isso resulta em todos os gerenciadores de filas no cluster estando cientes de todos os tópicos que estão sendo assinados e de qualquer gerenciador de filas que hospeda uma assinatura estabelecendo um canal para cada um dos outros gerenciadores de filas. Isso independe de cada gerenciador de filas ter um publicador.
- O conhecimento de cada tópico individual assinado em todos os gerenciadores de filas pode ser removido, alterando para um modelo de envio de todas as publicações para todos os gerenciadores de filas no cluster, independentemente de terem assinaturas. Isso reduz o tráfego de conhecimento de assinatura, mas é provável que aumente o tráfego de publicações e o número de canais que cada gerenciador de filas estabelece. Consulte Desempenho de assinatura em redes publicar/assinar.

Os fluxos de mensagens publicar/assinar usando tópicos em cluster roteados diretamente podem abranger diversos clusters publicar/assinar, incluindo um gerenciador de filas de cada cluster em uma hierarquia publicar/assinar. Consulte Combinando os espaços de tópico de diversos clusters.

Para obter uma exploração mais detalhada de roteamento direto, consulte Roteamento direto de clusters publicar/assinar.

Uma visão geral do roteamento de host de tópico

Quando um objeto de tópico administrado é configurado para roteamento de host de tópico, as publicações de um gerenciador de filas no cluster são roteadas por meio de um gerenciador de filas

no qual o objeto de tópico está configurado (um " host de tópico "), e de lá para os gerenciadores de filas nos quais as assinaturas existem.

- Isso depende de cada gerenciador de filas informar todos os hosts de tópico de cada tópico que atualmente possui uma ou mais assinaturas para ele. Qualquer gerenciador de filas que hospeda uma assinatura estabelece um canal para cada host de tópico para o tópico ao qual a assinatura está relacionada.
- Os gerenciadores de filas sem host tópico não ficaram cientes de outros gerenciadores de filas sem host de tópico no cluster para os propósitos de publicar/assinar, e os canais não são estabelecidos entre eles para esse propósito.
- Se o aplicativo de publicação estiver conectado a um gerenciador de filas que hospeda o tópico, as mensagens publicadas serão roteadas diretamente para os gerenciadores de filas nos quais as assinaturas correspondentes foram criadas, sem requerer um 'hop' adicional. De forma semelhante, se as assinaturas correspondentes forem criadas no único gerenciador de filas que hospeda o tópico, as mensagens publicadas nesse tópico serão roteadas diretamente para esse gerenciador de filas, sem requerer um hop adicional.
- As assinaturas no mesmo gerenciador de filas que o publicador são atendidas sem primeiro rotear publicações para os hosts do objeto do tópico.

Assim como para filas em cluster, diversos gerenciadores de filas podem configurar o mesmo objeto do tópico administrativo. Isso fornece maior disponibilidade de roteamento de mensagem e escala horizontal por meio de balanceamento de carga de trabalho. Para objetos do tópico roteados por host de tópico, quando diversos gerenciadores de filas configuram o mesmo tópico nomeado para a mesma ramificação da árvore de tópicos, cada host de tópico se torna ciente dos tópicos assinados por cada gerenciador de filas que hospeda uma assinatura.

- Quando uma mensagem é publicada, ela é enviada para um dos gerenciadores de filas de host de tópico para encaminhar para os gerenciadores de filas que hospedam a assinatura. A opção do gerenciador de filas de host de tópico segue as mesmas regras de balanceamento de carga de trabalho padrão que para filas ponto a ponto em cluster.
- Se um ou mais gerenciadores de filas de host de tópico não puderem ser contatados por um gerenciador de filas de publicação, as mensagens serão roteadas para os gerenciadores de filas de host de tópico restantes disponíveis.

Cada publicação para um tópico em uma ramificação roteada da árvore de tópicos é encaminhada para um dos hosts de tópico, mesmo que não haja nenhuma assinatura para esse tópico em nenhum lugar no cluster. Por padrão, as mensagens são enviadas daqui somente para outros gerenciadores de filas no cluster que hospedam uma ou mais assinaturas.

- Isso depende de cada gerenciador de filas de host de tópico estar sendo informado de todas as sequências de tópicos assinadas em cada gerenciador de filas no cluster.
- O conhecimento de cada tópico individual assinado pode ser removido, alterando para um modelo de envio de todas as publicações roteadas para um host de tópico para todos os gerenciadores de filas no cluster, independentemente de terem assinaturas. Isso reduz o tráfego de conhecimento de assinatura, mas é provável que aumente o tráfego de publicações e, possivelmente, o número de canais estabelecidos com cada gerenciador de filas de host de tópico. Consulte Desempenho de assinatura em redes publicar/assinar.

Os fluxos de mensagens publicar/assinar usando tópicos em cluster roteados por host de tópico **não podem** abranger diversos clusters publicar/assinar usando uma hierarquia publicar/assinar.

Para obter uma exploração mais detalhada do roteamento de host de tópico, consulte Roteamento de host de tópico em clusters publicar/assinar.

Hierarquias de Publicação/Assinatura

Você constrói uma hierarquia publicar/assinar vinculando os gerenciadores de filas usando canais, em seguida, definindo um relacionamento pai-filho entre pares de gerenciadores de filas. Uma mensagem flui de um publicador para as assinaturas por meio das relações diretas em uma hierarquia. Observe que isso pode significar diversos "hops" para chegar lá.

Somente uma cópia da mensagem é enviada entre qualquer par de gerenciadores de filas, independentemente do número de assinantes para a mensagem no gerenciador de filas de destino. Na chegada a um gerenciador de filas com uma ou mais assinaturas, a mensagem é duplicada em todas as assinaturas.

Por padrão, as mensagens são enviadas somente a outros gerenciadores de filas na hierarquia que estão na rota para uma assinatura em outro gerenciador de filas:

- Isso depende de cada gerenciador de filas que informa cada relação direta de todos os tópicos que atualmente possuem uma ou mais assinaturas, neste gerenciador de filas ou em uma de suas outras relações. Isso resulta em todos os gerenciadores de filas na hierarquia ficando cientes de todos os tópicos que estão sendo assinados.
- Esse comportamento pode ser mudado para sempre enviar publicações para todos os gerenciadores de filas na hierarquia, independentemente de quaisquer assinaturas existentes. Isso remove a necessidade de propagar as informações de assinatura na hierarquia, mas pode aumentar o tráfego de publicações.

Ao criar um cluster, é preciso ter cuidado para não criar um loop que faz com que as mensagens entrem em um ciclo permanente na rede. Nenhum loop desse tipo pode ser criado em uma hierarquia.

Cada gerenciador de filas deve ter um nome do gerenciador de filas exclusivo.

Os fluxos de mensagens publicar/assinar podem abranger diversos clusters publicar/assinar. Para isso, inclua um gerenciador de filas de cada cluster em uma hierarquia publicar/assinar.

Para obter uma exploração mais detalhada, consulte Roteamento em hierarquias publicar/assinar.

Assinaturas de proxy em uma rede publicar/assinar

Uma assinatura de proxy é uma assinatura feita por um gerenciador de filas para tópicos publicados em outro gerenciador de filas. Uma assinatura de proxy flui entre gerenciadores de filas para cada sequência de tópico individual que é assinada por uma assinatura. Você não cria assinaturas de proxy explicitamente, o gerenciador de filas faz isso em seu nome.

É possível conectar gerenciadores de filas em um cluster publicar/assinar ou em uma hierarquia publicar/assinar. Fluxo de assinaturas de proxy entre os gerenciadores de filas conectados. As assinaturas de proxy fazem publicações em um tópico criadas por um publicador conectado a um gerenciador de filas serem recebidas por assinantes desse tópico conectados a outros gerenciadores de filas. Consulte o “Redes publicar/assinar distribuídas” na página 91.

Em topologias publicar/assinar com muitas milhares de assinaturas para sequências de tópicos individuais ou onde a existência dessas assinaturas pode ser rapidamente alterada, deve ser considerada a sobrecarga da propagação de assinaturas de proxy. Além da agregação automática descrita no restante deste tópico, é possível fazer mudanças manuais na configuração, que restringem ainda mais o fluxo de assinaturas de proxy e de publicações entre gerenciadores de filas conectados, e reduzem a latência de espera para que uma assinatura de proxy seja propagada para todos os gerenciadores de filas conectados. Consulte Desempenho de assinatura em redes publicar/assinar.

As assinaturas de proxy não contêm seletores usados por assinaturas locais e as sequências de tópicos de assinatura que contêm curingas podem ser simplificadas. Isso pode resultar em publicações correspondendo a assinaturas de proxy, em que as assinaturas reais não correspondem, resultando em um fluxo de publicações adicional entre gerenciadores de filas. O gerenciador de filas que hospeda as assinaturas filtra essas discrepâncias para que publicações adicionais não sejam retornadas às assinaturas.

Agregação de Assinatura de Proxy

As assinaturas de proxy são agregadas usando um sistema de eliminação duplicado. Para uma sequência de tópico resolvida específica, uma assinatura de proxy é enviada na primeira assinatura local ou assinatura de proxy recebida. Assinaturas subsequentes para a mesma sequência de tópicos faz uso desta assinatura de proxy existente.

A assinatura de proxy é cancelada após a última assinatura local ou após a assinatura de proxy recebida ser cancelada.

Agregação da Publicação

Quando houver mais de uma assinatura para a mesma sequência de tópicos em um gerenciador de filas, apenas uma única cópia de cada publicação correspondente a sequência de tópicos é enviada a partir de outros gerenciadores de filas na topologia de publicação/assinatura. Na chegada da mensagem, o gerenciador de filas local fornece uma cópia da mensagem para cada assinatura correspondente.

É possível para mais de uma assinatura de proxy para corresponder à sequência de tópicos de uma única publicação quando as assinaturas de proxy contiverem curingas. Se uma mensagem for publicada em um gerenciador de filas que corresponde a duas ou mais assinaturas de proxy criadas por um único gerenciador de filas conectado, apenas uma cópia da publicação será encaminhada para o gerenciador de filas remoto para satisfazer as assinaturas de proxy múltiplas.

Conceitos relacionados

Detecção de loop em uma rede de publicação/assinatura distribuída

Curingas em Assinaturas de Proxy

As assinaturas podem usar curingas nas sequências de tópicos para correspondências com sequências de tópicos em publicações.

Há dois esquemas curinga que uma assinatura pode usar: *baseado em tópico* e *baseado em caracteres*. Consulte o “Esquemas Curinga” na página 72.

No IBM MQ , todas as assinaturas de proxy para assinaturas curinga são convertidas para usar curingas baseados em tópico. Se um curinga baseado em caracteres for localizado, ele será substituído por um caractere `#`, de volta para a `/` mais próxima. Por exemplo, `/aaa/bbb/c*d` é convertido em `/aaa/bbb/#`. A conversão resulta em gerenciadores de filas remotos enviando um pouco mais de publicações a que foram explicitamente assinadas. As publicações adicionais são filtradas pelo gerenciador de filas local, quando ele entrega as publicações para seus assinantes locais.

Controlando o Uso de Curinga com a Propriedade WILDCARD

Use a propriedade MQSC **Topic** WILDCARD ou a propriedade PCF Topic `WildcardOperation` equivalente para controlar a entrega de publicações para aplicativos assinantes que usam nomes da sequência de tópicos curinga. A propriedade WILDCARD pode ter um dos dois valores possíveis:

CURINGA
O comportamento de assinaturas curingas com relação a este tópico.

PASSTHRU
As assinaturas feitas em um tópico curinga menos específico do que a sequência de tópicos neste objeto do tópico recebem as publicações feitas neste tópico e para sequências de tópicos mais específicas do que este tópico.

BLOCK
As assinaturas feitas em um tópico curinga menos específico do que a sequência de tópicos neste objeto do tópico não recebem as publicações feitas neste tópico ou nas sequências de tópicos mais específicas do que este tópico.

O valor deste atributo é usado quando as assinaturas são definidas. Se você alterar este atributo, o conjunto de tópicos coberto pelas assinaturas existentes não será afetado pela modificação. Este cenário também se aplica se a topologia for alterada quando os objetos do tópico forem criados ou excluídos; o conjunto de tópicos que corresponde às assinaturas criadas seguindo a modificação do atributo WILDCARD é criado usando a topologia modificada. Se você desejar forçar o conjunto de tópicos correspondente para ser reavaliado para assinaturas existentes, deve reiniciar o gerenciador de filas.

No exemplo, “Exemplo: Criar um Cluster de Publicação/Assinatura Sport” na página 85, é possível seguir as etapas para criar a estrutura da árvore de tópicos mostrada em Figura 23 na página 82.

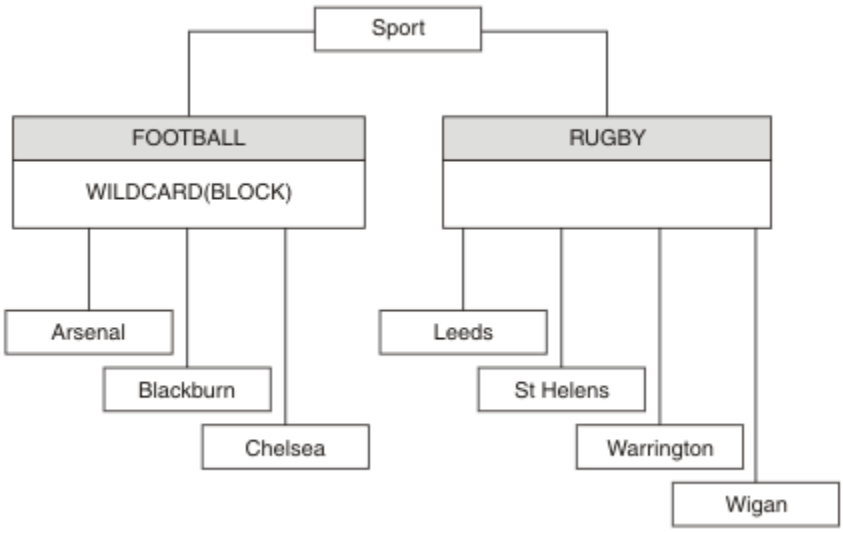


Figura 36. Uma árvore de tópicos que usa a propriedade WILDCARD, BLOCK

Um assinante que usa a sequência de tópicos curinga # recebe todas as publicações para o tópico `Sport` e subárvore `Sport/Rugby`. O assinante não recebe nenhuma publicação para a subárvore `Sport/Football`, porque o valor da propriedade `WILDCARD` do tópico `Sport/Football` é `BLOCK`.

PASSTHRU é a configuração padrão. É possível configurar o valor da propriedade PASSTHRU de `WILDCARD` para os nós na árvore `Sport`. Se os nós não tiverem o valor da propriedade `BLOCK` do `WILDCARD`, configurar PASSTHRU não vai alterar o comportamento observado pelos assinantes para os nós na árvore `Sports`.

No exemplo, crie assinaturas para ver como a configuração curinga afeta as publicações que são entregues; consulte Figura 27 na página 87. Execute o comando de publicação no Figura 30 na página 88 para criar algumas publicações.

```
pub QMA
```

Figura 37. Publicar para QMA

Os resultados são mostrados em Tabela 3 na página 83. Observe como a configuração do valor da propriedade `BLOCK` de `WILDCARD`, impede que as assinaturas com curingas recebam as publicações para os tópicos no escopo do curinga.

Tabela 6. Publicações Recebidas em QMA

Assinatura	Cadeia do tópico	Publicações Recebidas	Notas

SPORTS	`Sports/#`	`Sports`
		`Sports/Rugby`
		`Sports/Rugby/Leeds`
			Todas as publicações para a subárvore Football bloqueadas por `WILDCARD(BLOCK)` em `Sports/Football`

SARSENAL	`Sports/#/Arsenal`	-	`WILDCARD(BLOCK)` em `Sports/Football` impede as assinaturas curingas em `Arsenal`
SLEEDS	`Sports/#/Leeds`	`Sports/Rugby/Leeds`	`WILDCARD` padrão em Sports/Rugby não impede a assinatura curinga em `Leeds`.

Nota:

Imagine que uma assinatura tenha um curinga que corresponda a um objeto do tópico com o valor da propriedade BLOCK de WILDCARD. Se a assinatura também tiver uma sequência de tópicos à direita do curinga correspondente, a assinatura nunca receberá uma publicação. O conjunto de publicações que não estão bloqueadas são publicações para tópicos que são pais do curinga bloqueado. As publicações para os tópicos que são filhos do tópico com o valor da propriedade BLOCK são bloqueadas pelo curinga. Portanto, as sequências de tópicos de assinatura que incluem um tópico à direita do curinga nunca recebem qualquer publicação a ser correspondida.

Configurar o valor da propriedade WILDCARD para BLOCK não significa que você não possa se subscrever usando uma sequência de tópicos que inclua curingas. Essa assinatura é normal. A assinatura tem um tópico explícito que corresponde ao tópico com um objeto do tópico que possui um valor da propriedade BLOCK de WILDCARD. Ela usa os curingas para os tópicos que são pais ou filhos do tópico com o valor da propriedade BLOCK de WILDCARD. No exemplo em Figura 23 na página 82, uma assinatura, como Sports/Football/#, pode receber as publicações.

Curingas e Tópicos do Cluster

As definições do tópico do cluster são propagadas para cada gerenciador de filas em um cluster. Uma assinatura para um tópico de cluster em um gerenciador de filas em um cluster resulta no gerenciador de filas criando assinaturas de proxy. Uma assinatura de proxy é criada em cada outro gerenciador de filas no cluster. As assinaturas que usam sequências de tópicos que contêm curingas, combinados com tópicos de cluster, podem fornecer um comportamento difícil de ser previsto. O comportamento é explicado no seguinte exemplo.

Na configuração do cluster para o exemplo, “Exemplo: Criar um Cluster de Publicação/Assinatura Sport” na página 85, QMB tem o mesmo conjunto de assinaturas que QMA, embora QMB não tenha recebido nenhuma publicação após o publicador publicado para QMA, consulte Figura 24 na página 83. Embora os tópicos Sports/Football e Sports/Rugby sejam tópicos de cluster, as assinaturas definidas em fullsubs.tst não fazem referência a um tópico de cluster. Nenhuma assinatura de proxy é propagada de QMB para QMA. Sem as assinaturas de proxy, nenhuma publicação para QMA é encaminhada para QMB.

Algumas das assinaturas, como Sports/#/Leeds, podem parecer fazer referência a um tópico do cluster, neste caso, Sports/Rugby. A assinatura Sports/#/Leeds resolve, na realidade, para o objeto de tópico SYSTEM.BASE.TOPIC.

A regra para resolver o objeto do tópico referenciado por uma assinatura, como Sports/#/Leeds, é a seguinte. Truncar a sequência de tópicos para o primeiro curinga. Varra para a esquerda da sequência de tópicos procurando pelo primeiro tópico que tem um objeto de tópico administrativo associado. O objeto do tópico pode especificar um nome de cluster ou definir um objeto de tópico local. No exemplo, Sports/#/Leeds, a sequência de tópicos após o truncamento é Sports, que não possui nenhum objeto do tópico e, portanto, Sports/#/Leeds herda de SYSTEM.BASE.TOPIC, que é um objeto do tópico local.

Para ver como a assinatura nos tópicos em cluster pode alterar a maneira como a propagação de curinga funciona, execute o script em lote, upsubs.bat. O script limpa as filas de assinatura e inclui as assinaturas de tópico de cluster em fullsubs.tst. Execute puba.bat novamente para criar um lote de publicações; consulte Figura 24 na página 83.

Tabela 4 na página 84 mostra o resultado de incluir duas novas assinaturas no mesmo gerenciador de filas no qual as publicações foram publicadas. O resultado é o esperado, as novas assinaturas recebem uma publicação cada e os números de publicações recebidas pelas outras assinaturas ficam inalterados. Ocorrem os resultados inesperados no outro gerenciador de filas do cluster; consulte Tabela 5 na página 85.

Tabela 7. Publicações Recebidas em QMA			
Assinatura	**Cadeia do tópico**	**Publicações Recebidas**	**Notas**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Todas as publicações para a subárvore Football bloqueadas por WILDCARD(BLOCK) em Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) em Sports/Football impede as assinaturas curingas em Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	WILDCARD padrão em Sports/Rugby não impede a assinatura curinga em Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal recebe uma publicação porque a assinatura não tem um curinga.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds receberia uma publicação em qualquer evento.

Tabela 5 na página 85 mostra os resultados de incluir as duas novas assinaturas no QMB e publicar em QMA. A rechamada desse QMB não recebeu nenhuma publicação sem essas duas novas assinaturas. Conforme esperado, as duas novas assinaturas recebem as publicações, porque Sports/FootBall e Sports/Rugby são tópicos de cluster. O QMB encaminhou assinaturas de proxy para Sports/Football/Arsenal e Sports/Rugby/Leeds para QMA, que então enviou as publicações para QMB.

O resultado inesperado é que duas assinaturas Sports/# e Sports/#/Leeds que não receberam anteriormente nenhuma publicação, agora recebem as publicações. A razão é que as publicações Sports/Football/Arsenal e Sports/Rugby/Leeds encaminhadas para QMB para as outras assinaturas agora estão disponíveis para qualquer assinante conectado ao QMB. Consequentemente as assinaturas para os tópicos locais Sports/# E Sports/#/Leeds recebem a publicação Sports/Rugby/Leeds. Sports/#/Arsenal continua não recebendo uma publicação, porque Sports/Football tem seu valor de propriedade WILDCARD configurado como BLOCK.

Tabela 8. Publicações Recebidas em QMB			
Assinatura	**Cadeia do tópico**	**Publicações Recebidas**	**Notas**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*Todas as publicações para a subárvore Football bloqueadas por WILDCARD(BLOCK) em Sports/Football*
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) em Sports/Football impede as assinaturas curingas em Arsenal
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*WILDCARD padrão no Sports/Rugby não evita a assinatura curinga em Leeds.*
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal recebe uma publicação porque a assinatura não tem um curinga.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds receberia uma publicação em qualquer evento.

Na maioria dos aplicativos, é desejável para uma assinatura influenciar o comportamento de outra assinatura. Um uso importante da propriedade WILDCARD com valor BLOCK é fazer com que as assinaturas para a mesma sequência de tópicos que contém curingas se comportem uniformemente. Indica se a assinatura está no mesmo gerenciador de filas que o publicador ou um diferente, os resultados da assinatura são os mesmos.

Curingas e Fluxos

Para um novo aplicativo gravado para a API de publicação/assinatura, o efeito é que uma assinatura para * não recebe nenhuma publicação. Para receber todas as publicações de Sports você deve se subscrever a Sports/* ou Sports/#, de maneira semelhante para publicações Business.

O comportamento de um aplicativo de publicação / assinatura enfileirado existente não muda quando o broker de publicação / assinatura é migrado para uma versão posterior do IBM MQ. A propriedade **StreamName** nos comandos **Publish**, **Register Publisher**ou **Subscriber** é mapeada para o nome do tópico para o qual o fluxo foi migrado..

Curingas e Pontos de Assinatura

Para um novo aplicativo gravado para a API de publicação/assinatura, o efeito da migração é que uma assinatura para * não receba nenhuma publicação. Para receber todas as publicações de Sports você deve se subscrever a Sports/* ou Sports/#, de maneira semelhante para publicações Business.

O comportamento de um aplicativo de publicação / assinatura enfileirado existente não muda quando o broker de publicação / assinatura é migrado para uma versão posterior do IBM MQ. A propriedade **SubPoint** nos comandos **Publish**, **Register Publisher**ou **Subscriber** é mapeada para o nome do tópico para o qual a assinatura foi migrada..

Exemplo: Criar um Cluster de Publicação/Assinatura Sport

As etapas que seguem a criação de um cluster, CL1, com quatro gerenciadores de filas: dois repositórios completos, CL1A e CL1B, e dois repositórios parciais, QMA e QMB. Os repositórios completos são usados para reter apenas as definições de cluster. QMA designado ao host de tópico do cluster. As assinaturas duráveis são definidas em QMA e QMB.

Nota: O exemplo é codificado para Windows. Você deve recodificar Create qmgrs.bat e create pub.bat para configurar e testar o exemplo em outras plataformas.

1. Crie os arquivos de script.
   a. Create topics.tst
   b. Create wildsubs.tst
   c. Create fullsubs.tst
   d. Create qmgrs.bat
   e. create pub.bat
2. Execute Crie qmgrs.bat para criar a configuração.

```
qmgrs
```

Crie os tópicos em Figura 23 na página 82. O script na figura 5 cria os tópicos de cluster Sports/Football e Sports/Rugby

Nota: A opção REPLACE não substitui as propriedades TOPICSTR de um tópico. TOPICSTR é uma propriedade que é proveitosamente variada no exemplo para testar as diferentes árvores de tópicos. Para alterar os tópicos, exclua primeiro o tópico.

```
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. Excluir e Criar os Tópicos: topics.tst

Nota: Exclua os tópicos, porque o REPLACE não substitui as sequências de tópicos.

Crie assinaturas com curingas. Os curingas correspondentes dos tópicos com objetos do tópico no Figura 23 na página 82. Crie uma fila para cada assinatura. As filas são limpas e as assinaturas excluídas quando o script não é executado ou reexecutado.

Nota: A opção REPLACE não substitui as propriedades TOPICOBJ ou TOPICSTR de uma assinatura. TOPICOBJ ou TOPICSTR são propriedades proveitosamente variadas no exemplo para testar diferentes assinaturas. Para alterá-las, exclua primeiro a assinatura.

```
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. Criar Assinaturas Curinga: wildsubs.tst

Crie as assinaturas que fazem referência aos objetos do tópico de cluster.

Nota:

O delimitador, /, é inserido automaticamente entre a sequência de tópicos referenciada por TOPICOBJ e a sequência de tópicos definida por TOPICSTR.

A definição, DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) cria a mesma assinatura. TOPICOBJ é usado como uma maneira rápida de fazer referência à sequência de tópicos que você já definiu. A assinatura, quando criada, não se refere mais ao objeto do tópico.

```
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. Excluir e Criar Assinaturas: fullsubs.tst

Crie um cluster com dois repositórios. Crie dois repositórios parciais para publicar e subscrever. Execute novamente o script para excluir tudo e inicie novamente. O script também cria a hierarquia de tópico e as assinaturas curingas iniciais.

Nota:

Em outras plataformas, grave um script semelhante ou digite todos os comandos. Usar um script facilita a exclusão de tudo e inicia novamente com uma configuração idêntica.

```
@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. Criar Gerenciadores de Filas: qmgrs.bat

Atualize a configuração incluindo as assinaturas nos tópicos do cluster.

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

Figura 42. Atualizar Assinaturas: upsubs.bat

Execute `pub.bat`, com um gerenciador de filas como um parâmetro, para publicar as mensagens que contêm a sequência de tópicos de publicação. `Pub.bat` usa o programa de amostra **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. Publicar: pub.bat

Conceitos relacionados

Assinaturas curinga e publicações retidas

Escopo da publicação

Ao configurar um cluster ou hierarquia publicar/assinar, o escopo de uma publicação controla ainda mais se os gerenciadores de filas encaminharão uma publicação para gerenciadores de filas remotas. Use o atributo de tópico **PUBSCOPE** para administrar o escopo de publicações.

Se uma publicação não for encaminhada para gerenciadores de filas remotos, apenas assinantes locais recebem a publicação.

Ao usar um cluster publicar/assinar, o escopo de publicações é controlado principalmente pela definição de objetos do tópico em cluster em alguns pontos na árvore de tópicos. O escopo de publicação deve ser configurado para permitir o fluxo de publicações para outros gerenciadores de filas no cluster. É necessário restringir o escopo de publicação para um tópico em cluster somente quando for necessário o controle de baixa granularidade de tópicos específicos em alguns gerenciadores de filas.

Ao usar uma hierarquia publicar/assinar, o escopo de publicações é controlado principalmente por este atributo, em conjunto com o atributo de escopo de assinatura.

O atributo **PUBSCOPE** é usado para determinar o escopo de publicações feitas para um tópico específico. É possível configurar o atributo para um dos seguintes valores:

QMGR
: A publicação é entregue apenas para assinantes locais. Estas publicações são chamadas de *publicações locais*. As publicações locais não são encaminhadas para gerenciadores de filas remotos e, portanto, não são recebidas pelos assinantes conectados aos gerenciadores de filas remotas.

ALL
: A publicação é entregue para assinantes locais e assinantes conectados aos gerenciadores de filas remotas em um cluster ou hierarquia publicar/assinar. Estas publicações são chamadas de *publicações globais*.

ASPARENT
: Use a configuração **PUBSCOPE** do tópico pai na árvore de tópicos.

Os publicadores também podem especificar se uma publicação é local ou global usando a opção de colocar a mensagem MQPMO_SCOPE_QMGR. Se essa opção for usada, ela substitui qualquer comportamento que foi configurado usando o atributo do tópico **PUBSCOPE**.

Conceitos relacionados

“Objetos de Tópico Administrativo” na página 79
Usando um objeto de tópico administrativo, é possível designar atributos específicos não padrão para os tópicos.

Tarefas relacionadas

Configurando redes publicar/assinar distribuídas

Escopo da assinatura

O escopo de uma assinatura controla se uma assinatura em um gerenciador de filas recebe publicações que são publicadas em outro gerenciador de filas em um cluster ou uma hierarquia de publicação/assinatura ou apenas as publicações de publicadores locais.

Limitando o escopo de assinatura a um gerenciador de filas para as assinaturas de proxy de serem encaminhadas para outros gerenciadores de filas na topologia de publicação/assinatura. Isso reduz entre o tráfego de mensagens de publicação/assinatura do gerenciador de filas.

Ao usar um cluster publicar/assinar, o escopo de assinaturas é controlado principalmente pela definição de objetos do tópico em cluster em alguns pontos na árvore de tópicos. O escopo de assinatura deve ser configurado para permitir o fluxo de assinaturas de proxy para outros gerenciadores de filas no cluster. É necessário restringir o escopo de assinatura para um tópico em cluster somente quando for necessário o controle de baixa granularidade de tópicos específicos em alguns gerenciadores de filas.

Ao usar uma hierarquia publicar/assinar, o escopo de assinaturas é controlado principalmente por este atributo, em conjunto com o atributo de escopo de publicação.

O atributo de tópico **SUBSCOPE** é usado para determinar o escopo de assinaturas feitas em um tópico específico. É possível configurar o atributo para um dos seguintes valores:

QMGR
Uma assinatura receberá apenas publicações locais e assinaturas de proxy não são propagadas para os gerenciadores de filas remotas.

ALL
Uma assinatura de proxy é propagada para gerenciadores de filas remotas em um cluster ou hierarquia publicar/assinar e o assinante recebe publicações locais e remotas.

ASPARENT
Use a configuração **SUBSCOPE** do tópico pai na árvore de tópicos.

Quando o escopo de assinatura de um tópico é configurado como ALL, diretamente ou resolvido por meio de ASPARENT, as assinaturas individuais para esse tópico podem restringir seu escopo para QMGR especificando MQSO_SCOPE_QMGR ao criar a assinatura. Uma assinatura para um tópico que tem um escopo de QMGR não pode ampliar o escopo para ALL.

Conceitos relacionados

“Objetos de Tópico Administrativo” na página 79
Usando um objeto de tópico administrativo, é possível designar atributos específicos não padrão para os tópicos.

Tarefas relacionadas

Configurando redes publicar/assinar distribuídas

Espaços de Tópico

Um espaço de tópico é o conjunto de tópicos no qual é possível assinar e publicar. Um gerenciador de filas em uma topologia publicar/assinar distribuída possui um espaço de tópico que possivelmente inclui tópicos que foram assinados e publicados em gerenciadores de filas conectados nessa topologia.

Nota: Para obter uma visão geral de tópicos em um gerenciador de filas, como objetos do tópico administrativos, sequências de tópicos e árvores de tópicos, consulte “tópicos” na página 70. Referências adicionais a *tópicos* no artigo atual referem-se a *sequências de tópicos*, a menos que seja especificado de outra forma.

Os tópicos são inicialmente criados de uma das seguintes maneiras:

- de forma administrativa, ao definir um objeto do tópico ou assinatura durável.
- dinamicamente, quando um aplicativo cria uma publicação ou assinatura dinamicamente para um novo tópico.

Os tópicos são propagados para outros gerenciadores de filas tanto através de assinaturas de proxy e com a criação de objetos do tópico administrativo. As assinaturas de proxy resultam em publicações que estão sendo encaminhadas do gerenciador de filas para o qual um publicador está conectado, para os gerenciadores de filas de assinantes.

As assinaturas de proxy são propagadas entre todos os gerenciadores de filas que estão conectados juntos por relacionamentos pai-filho em uma hierarquia do gerenciador de filas. O resultado é, você pode assinar em um gerenciador de filas para um tópico definido em qualquer outro gerenciador de filas na

hierarquia. Contanto que haja um caminho conectado entre os gerenciadores de filas, não importa como os gerenciadores de filas estão conectados.

As assinaturas de proxy também são propagadas para assinaturas em tópicos de cluster em um cluster publicar/assinar. Um tópico de cluster é um tópico que está conectado a um objeto do tópico que tenha o atributo **CLUSTER** ou herda o atributo de seu pai. Os tópicos que não sejam tópicos de cluster são conhecidos como tópicos locais e não são replicados para o cluster. Nenhuma assinatura de proxy é propagada para o cluster de assinaturas para tópicos locais.

Para resumir, as assinaturas de proxy são criadas para os assinantes em duas circunstâncias.

1. Um gerenciador de filas é um membro de uma hierarquia e uma assinatura de proxy é redirecionada para o pai e filho do gerenciador de filas.
2. Um gerenciador de filas é um membro de um cluster e a sequência de tópicos de assinatura é resolvida para um tópico que está associado a um objeto do tópico de cluster. Quando o tópico é um tópico de cluster *roteado diretamente*, as assinaturas de proxy são encaminhadas para todos os membros do cluster. Quando o tópico é um tópico de cluster *roteado por host de tópico*, as assinaturas de proxy são encaminhadas somente para os gerenciadores de filas no cluster que definiram o objeto de tópico em cluster. Para obter mais informações, consulte “Publicar/assinar clusters” na página 95.

Se um gerenciador de filas for um membro de um cluster e uma hierarquia, as assinaturas de proxy são propagadas por ambos os mecanismos sem entregar publicações duplicadas para o assinante.

Os espaços de tópicos de três topologias de publicação/assinatura são descritos na lista a seguir:

- “Caso 1. Publicar/assinar clusters” na página 108.
- “Caso 2. Hierarquias de Publicação/Assinatura” na página 109.

Em tópicos separados, as seguintes tarefas de configuração descrevem como combinar espaços de tópico.

- Criando um único espaço de tópico em um cluster publicar/assinar.
- Combinando os espaços de tópico de diversos clusters.
- Combinando e isolando espaços de tópico em diversos clusters.
- Publicando e assinando espaços de tópico em diversos clusters.

Caso 1. Publicar/assinar clusters

No exemplo, suponha que o gerenciador de filas não está conectado a uma hierarquia de publicação/assinatura.

Se um gerenciador de filas for um membro de um cluster de publicação/assinatura, seu espaço de tópico será composto de tópicos locais e tópicos de cluster. Os tópicos locais são associados aos objetos do tópico sem o atributo **CLUSTER**. Se um gerenciador de filas tiver definições de objeto do tópico local, seu espaço de tópico será diferente de outro gerenciador de filas no cluster que também possui seus próprios objetos de tópico definidos localmente.

Em um cluster de publicação/assinatura, você não pode assinar um tópico definido em outro gerenciador de filas, a menos que o tópico que você assinar seja resolvido para um objeto de tópico de cluster.

Quando as definições com o mesmo nome de um objeto do tópico de cluster forem necessárias em diversos gerenciadores de filas, por exemplo, ao usar *roteamento de host de tópico*, é importante que todas as definições correspondam, onde necessário. Para obter informações adicionais, consulte Criando um único espaço de tópico em um cluster publicar/assinar.

Uma definição local de um objeto do tópico, se a definição for para um tópico de cluster ou um tópico local, tem precedência sobre o mesmo objeto do tópico definido em outro lugar no cluster. O tópico localmente definido é usado, mesmo se o objeto definido em outro lugar for mais recente.

É importante que um objeto do tópico de cluster esteja associado com a mesma sequência de tópicos em todo o cluster. Você não pode modificar a sequência de tópicos com o qual um objeto do tópico está associado. Para associar o mesmo objeto do tópico com uma sequência de tópicos diferente, você deve excluir o objeto do tópico e recria-lo com a sequência de tópico nova. Se o tópico for armazenado em

cluster, o efeito será excluir as cópias do objeto do tópico armazenadas em outros membros do cluster e, em seguida, para criar cópias do novo objeto do tópico em todos os lugares no cluster. As cópias do objeto do tópico se referem à mesma sequência de tópico.

É possível criar acidentalmente duas definições do mesmo objeto do tópico nomeado em gerenciadores de filas diferentes no cluster, com sequências de tópicos diferentes. Isto pode resultar em comportamento confuso, porque diversas definições do mesmo objeto do tópico com sequências de tópicos diferentes podem produzir diferentes resultados, dependendo de como e onde o tópico é referenciado. Consulte Diversas definições de tópico de cluster com o mesmo nome para obter informações adicionais sobre este ponto importante.

Caso 2. Hierarquias de Publicação/Assinatura

No exemplo, suponha que o gerenciador de filas não é membro de um cluster de publicação/assinatura.

No IBM MQ, se um gerenciador de filas for um membro de uma hierarquia de publicação / assinatura, seu espaço de tópico consistirá em todos os tópicos definidos localmente e em gerenciadores de filas conectados. O espaço de tópico de todos os gerenciadores de filas em uma hierarquia é o mesmo. Não há divisão de tópicos em tópicos locais e tópicos globais.

Configure uma das opções **PUBSCOPE** e **SUBSCOPE** para `QMGR`, para evitar que uma publicação em um tópico seja transmitida de um publicador para um assinante conectado aos gerenciadores de filas diferentes na hierarquia.

Suponha que você defina um objeto do tópico `Alabama` com a sequência de tópicos `USA/Alabama` no gerenciador de filas `QMA`. O resultado é o seguinte:

1. O espaço de tópico no `QMA` agora inclui o objeto do tópico `Alabama` e a sequência de tópicos `USA/Alabama`.
2. Um aplicativo ou administrador pode criar uma assinatura no `QMA` usando o nome do objeto do tópico `Alabama`.
3. Um aplicativo pode criar uma assinatura para qualquer tópico, incluindo `USA/Alabama`, em qualquer gerenciador de filas na hierarquia. Se o `QMA` não tiver sido definido localmente, o tópico `USA/Alabama` será resolvido para o objeto do tópico `SYSTEM.BASE.TOPIC`.

Conceitos relacionados

“Escopo da publicação” na página 106
Ao configurar um cluster ou hierarquia publicar/assinar, o escopo de uma publicação controla ainda mais se os gerenciadores de filas encaminharão uma publicação para gerenciadores de filas remotas. Use o atributo de tópico **PUBSCOPE** para administrar o escopo de publicações.

“Escopo da assinatura” na página 106
O escopo de uma assinatura controla se uma assinatura em um gerenciador de filas recebe publicações que são publicadas em outro gerenciador de filas em um cluster ou uma hierarquia de publicação/assinatura ou apenas as publicações de publicadores locais.

Tarefas relacionadas

Configurando redes publicar/assinar distribuídas

IBM MQ Multicast

O IBM MQ Multicast oferece baixa latência, alto fan-out, sistema de mensagens multicast confiável.

O multicast é uma forma eficiente de sistema de mensagens de publicação/assinatura porque pode ser escalada a um alto número de assinantes sem efeitos prejudiciais no desempenho. O IBM MQ ativa um sistema de mensagens Multicast confiável usando confirmações, confirmações negativas e números de sequência para obter um sistema de mensagens de baixa latência com alto fan-out.

A entrega justa do IBM MQ Multicast permite entrega quase simultânea, assegurando que nenhum destinatário tenha uma vantagem. Como o IBM MQ Multicast usa a rede para entregar as mensagens, um mecanismo de publicação/assinatura não é necessário para fan-out de dados. Depois que um tópico é mapeado para um endereço de grupo, não há necessidade para o gerenciador de filas porque

os publicadores e os assinantes podem operar em um modo ponto a ponto. Isso permite que o carregamento seja reduzido nos servidores do gerenciador de filas e o servidor do gerenciador de filas não é mais um ponto de falha em potencial.

Conceitos Iniciais e Multicast

OIBM MQ Multicast pode ser facilmente integrado a sistemas e aplicativos existentes usando o objeto Informações de comunicação (COMMINFO). Dois campos do objeto TOPIC permitem a configuração rápida de objetos TOPIC existentes para suportar ou ignorar tráfego multicast.

Objetos Necessários para Multicast

As informações a seguir são uma visão geral em síntese dos dois objetos necessários para o IBM MQ Multicast:

Objeto COMMINFO
O objeto COMMINFO contém os atributos associados à transmissão multicast. Para obter mais informações sobre os parâmetros de objeto COMMINFO, consulte DEFINE COMMINFO.

O único campo COMMINFO que deve ser configurado é o nome do objeto COMMINFO. Esse nome é então usado para identificar o objeto COMMINFO para um tópico. O campo **GRPADDR** do objeto COMMINFO deve ser verificado para assegurar que o valor seja um endereço de grupo multicast válido.

Objeto TOPIC
Um tópico é o assunto das informações que são publicadas em uma mensagem de publicação/assinatura e um tópico é definido criando um objeto TOPIC. Para obter mais informações sobre os parâmetros do objeto TOPIC, consulte DEFINE TOPIC.

Tópicos existentes podem ser usados com multicast alterando os valores dos seguintes parâmetros do objeto TOPIC: **COMMINFO** e **MCAST**.

- **COMMINFO** Este parâmetro especifica o nome do objeto de informações de comunicação multicast.
- **MCAST** Este parâmetro especifica se multicast é permitido nesta posição na árvore de tópicos. Por padrão, **MCAST** é configurado para ASPARENT, o que significa que o atributo multicast do tópico é herdado do pai. Configurar **MCAST** para ENABLED permite tráfego multicast neste nó.

Redes e Tópicos Multicast

As informações a seguir são uma visão geral do que ocorre a assinaturas com diferentes tipos de definição de assinatura e tópico. Todos estes exemplos supõem que o parâmetro **COMMINFO** do objeto TOPIC esteja configurado para o nome de um objeto COMMINFO válido:

Conjunto de tópicos para multicast ativado
Se o parâmetro **MCAST** da sequência de tópicos for configurado para ENABLED, assinaturas de clientes com capacidade multicast são permitidas e uma assinatura multicast é feita, a menos que:

- Seja uma assinatura durável de um cliente com capacidade multicast.
- Seja uma assinatura não gerenciada de um cliente com capacidade multicast.
- Seja uma assinatura de um cliente com capacidade não multicast.

Nesses casos, é feita uma assinatura não multicast e essas assinaturas são submetidas a downgrade para publicação/assinatura normal.

Conjunto de tópicos para multicast desativado
Se o parâmetro **MCAST** da sequência de tópicos for configurado para DISABLED, uma assinatura não multicast sempre é feita e essas assinaturas são submetidas a downgrade para publicação/assinatura normal.

Conjunto de tópicos somente para multicast
Se o parâmetro **MCAST** da sequência de tópicos for configurado para ONLY, assinaturas de clientes com capacidade multicast são permitidas e uma assinatura multicast é feita, a menos que:

- É uma assinatura durável: assinaturas duráveis são rejeitadas com código de razão 2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
- É uma assinatura não gerenciada: assinaturas não gerenciadas são rejeitadas com código de razão 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
- É uma assinatura de um cliente não compatível com multicast: essas assinaturas são rejeitadas com código de razão 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
- É uma assinatura de um aplicativo conectado localmente: essas assinaturas são rejeitadas com código de razão 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY

Windows Linux AIX Visão geral do MQ Telemetry

O MQ Telemetry inclui um serviço de telemetria (MQXR) que faz parte de um gerenciador de filas, clientes de telemetria que você mesmo pode gravar ou fazer download gratuitamente, e interfaces da linha de comandos e administrativas do explorer. Telemetria refere-se à coleta de dados e à administração de um amplo intervalo de dispositivos remotos. Com o MQ Telemetry, é possível integrar a coleta de dados e o controle de dispositivos com aplicativos da web.

O MQ Telemetry é um componente do IBM MQ. O upgrade para essas versões é essencialmente a instalação de uma versão mais recente do IBM MQ.

Os aplicativos de amostra continuam disponíveis livremente no Eclipse Paho e MQTT.org. Consulte Programas de amostra do IBM MQ Telemetry Transport.

Como MQ Telemetry é um componente de IBM MQ, MQ Telemetry pode ser instalado com o produto principal ou após o produto principal ter sido instalado. Para obter informações sobre migração, consulte Migrando o MQ Telemetry no Windows e Migrando o MQ Telemetry no Linux.

Estão incluídos no MQ Telemetry os seguintes componentes:

Canais de Telemetria
Use os canais de telemetria para gerenciar a conexão de clientes MQTT para IBM MQ. Os canais de telemetria usam novos objetos IBM MQ , como `SYSTEM.MQTT.TRANSMIT.QUEUE`, para interagir com o IBM MQ.

Serviço de telemetria (MQXR)
Os clientes do MQTT usam o serviço de telemetria `SYSTEM.MQXR.SERVICE` para se conectar a canais de telemetria.

Suporte do IBM MQ Explorer para o MQ Telemetry
O MQ Telemetry pode ser administrado usando o IBM MQ Explorer.

Documentação
A documentação do MQ Telemetry está incluída na documentação do produto padrão do IBM MQ A documentação do SDK para os clientes Java e C é fornecida na documentação do produto, e como Javadoc e HTML.

Conceitos de Telemetria

Você coleta informações do ambiente ao seu redor para decidir o que fazer. Como consumidor, você verifica o que há em uma loja antes de decidir o que comprar para comer. Você quer saber quanto tempo vai durar a viagem, se sair agora, antes de registrar uma conexão. Você verifica seus sintomas antes de decidir se vai agendar uma consulta médica. Você verifica quando um ônibus chegará antes de decidir se vai esperar por ele. As informações para essas decisões vêm diretamente de medidores e dispositivos, da palavra escrita no papel ou de uma tela ou de você. Em qualquer lugar, e em qualquer hora, você coleta informações, reúne todas elas, analisa uma por uma e toma uma ação.

Se as fontes de informações forem muito dispersas ou inacessíveis, será mais difícil e mais caro coletar informações exatas. Se você quiser fazer muitas mudanças ou se for difícil fazê-las, elas não serão feitas ou serão feitas quando forem menos efetivas.

E se os custos da coleção e do controle de informações de dispositivos totalmente dispersos forem consideravelmente reduzidos por meio da conexão de dispositivos com tecnologia digital à Internet?

Essas informações podem ser analisadas usando os recursos da Internet e da empresa. Você tem mais oportunidades de tomar decisões mais informadas e de agir sobre elas.

Tendências tecnológicas, e pressões econômicas e ambientais, estão fazendo estas mudanças acontecerem:

1. O custo da conexão e do controle de sensores e atuadores está reduzindo devido à normatização e à conexão com processadores digitais de baixo custo.
2. A Internet, e suas tecnologias, estão sendo cada vez mais usadas para a conexão com dispositivos. Em alguns países, telefones celulares excedem computadores pessoais em número de conexões com aplicativos de Internet. E certamente outros dispositivos seguem atrás.
3. A Internet, e suas tecnologias, facilitam a obtenção de dados para um aplicativo. O fácil acesso a dados está conduzindo o uso de analítica de dados para transformar dados de sensores em informações que são úteis em muitas outras soluções.
4. O uso inteligente de recursos é muitas vezes a forma mais rápida e barata de reduzir custos e emissões de carbono. As alternativas: localizar novos recursos ou desenvolver novas tecnologias para usar os recursos existentes, pode ser uma solução a longo prazo. A curto prazo, desenvolver novas tecnologias ou localizar novos recursos, muitas vezes é mais arriscado, mais demorado e mais caro do que melhorar soluções existentes.

exemplo

Um exemplo mostra como essas tendências criam novas oportunidades de interação com o ambiente de forma inteligente.

O International Convention for the Safety of Life at Sea (SOLAS) requer que um Automatic Identification System (AIS) seja implementado em vários navios. Ele é requerido em navios mercantes com mais de 300 toneladas e em navios de passageiros. Um AIS é principalmente um sistema de prevenção de colisões para navios costeiros. Ele é usado por autoridades marinhas para monitorar e controlar águas costeiras.

Entusiastas em todo o mundo estão implementando estações de rastreamento AIS de baixo custo e colocando informações sobre navios costeiros na Internet. Outros entusiastas estão compondo aplicativos que combinam informações do AIS com outras informações da Internet. Os resultados são colocados em sites e publicados usando Twitter e SMS.

Em um aplicativo, as informações das estações AIS próximas de Southampton são combinadas com informações geográficas e de propriedade dos navios. O aplicativo alimenta informações em tempo real sobre chegadas e partidas de balsas no Twitter. Pessoas que se deslocam para o trabalho regularmente usando as embarcações entre Southampton e a Ilha de Wight assinam o feed de notícias usando Twitter ou SMS. Se o feed mostrar que a balsa está atrasada, os comutadores poderão atrasar a sua partida e selecionar a mesma quando ela atracar após o seu horário de chegada planejado.

Para ver mais exemplos, consulte “Casos de Uso de Telemetria” na página 114.

Tarefas relacionadas

Instalando MQ Telemetry
Administrando MQ Telemetry
Migrando o MQ Telemetry no Windows
Migrando o MQ Telemetry no Linux
Desenvolvendo aplicativos para o MQ Telemetry
Resolução de problemas do MQ Telemetry ..

Referências relacionadas

Referência do MQ Telemetry

Introdução ao MQ Telemetry

Pessoas, empresas e governos desejam cada vez mais usar o MQ Telemetry para interagir de maneira mais inteligente com o ambiente no qual vivemos e trabalhamos. O MQ Telemetry conecta todos os tipos

de dispositivos à Internet e à empresa e reduz os custos da construção de aplicativos para dispositivos inteligentes.

O que é MQ Telemetry?

- Ele é um recurso do IBM MQ que estende o backbone de sistema de mensagens universal fornecido pelo IBM MQ para um amplo intervalo de sensores, acionadores e dispositivos de telemetria remotos. O MQ Telemetry estende o IBM MQ para que ele possa interconectar aplicativos corporativos inteligentes, serviços e tomadores de decisão com redes de dispositivos instrumentados.
- As partes principais do MQ Telemetry são:

  O serviço do MQ Telemetry (MQXR).
  Este serviço é executado no servidor IBM MQ e usa o protocolo IBM MQ Telemetry Transport (MQTT) para se comunicar com dispositivos de telemetria.

  Aplicativos MQTT que você grava.
  Estes aplicativos controlam as informações que são transportadas entre os dispositivos de telemetria e o gerenciador de filas do IBM MQ e as ações que são executadas em resposta a essas informações. Para ajudar a criar esses aplicativos, use as bibliotecas do cliente MQTT.

O Que Ele Pode Fazer por Mim?

- MQTT é um transporte de mensagens aberto que permite que as implementações do MQTT sejam criadas para uma ampla variedade de dispositivos.
- Os clientes MQTT podem ser executados em pequenos dispositivos de área de cobertura que possuem recursos limitados.
- O MQTT funciona de forma eficiente em redes nas quais a largura da banda é baixa, onde o custo de envio de dados é caro ou que pode ser frágil.
- A entrega da mensagem é garantida e separada do aplicativo.
- Os programadores de aplicativos não precisam ter conhecimento de programação de comunicações.
- As mensagens podem ser trocadas com outros aplicativos de sistema de mensagens. Estes podem ser outro aplicativo de telemetria ou um MQI, JMS ou aplicativo do sistema de mensagens corporativo.

Como eu Utilizo?

- Scripts de amostra são fornecidos que funcionam com um aplicativo cliente de amostra do IBM MQ Telemetry Transport v3 (`mqttv3app.jar`). Consulte Programas e amostras do IBM MQ Telemetry Transport.
- Use o IBM MQ Explorer e suas ferramentas associadas para administrar o recurso de telemetria do IBM MQ.
- Use as bibliotecas de cliente para ajudá-lo a criar os aplicativos MQTT que se conectam a um gerenciador de filas e que usam sistema de mensagens de publicação/assinatura.
- Distribua seu aplicativo e a biblioteca de cliente para o dispositivo no qual seu aplicativo será executado.

Como Funciona?

- MQTT é um protocolo de publicação e assinatura. Um aplicativo cliente MQTT pode publicar mensagens em um servidor MQTT ou assinar para mensagens que são enviadas por aplicativos que se conectam a um servidor MQTT.
- Os aplicativos clientes do MQTT usam bibliotecas de cliente que implementam o transporte de mensagens do MQTT.
- Um aplicativo cliente do MQTT funciona como um cliente MQ padrão, mas pode ser executado em uma variedade muito mais ampla de plataformas e redes.
- O serviço MQ Telemetry (MQXR) transforma um gerenciador de filas do IBM MQ em um servidor MQTT.

- Quando um gerenciador de filas do IBM MQ age como o servidor MQTT, outros aplicativos que se conectam ao gerenciador de filas podem assinar e receber as mensagens do cliente MQTT.
- O gerenciador de filas age como o roteador que distribui mensagens de aplicativos de publicação para aplicativos de assinatura.
- As mensagens podem ser distribuídas entre diferentes tipos de aplicativos clientes. Por exemplo, entre clientes de telemetria e clientes JMS.

Nota: O MQ Telemetry substitui os nós SCADA que foram retirados na versão 7 do WebSphere Message Broker (agora conhecido como IBM Integration Bus) e é executado no Windows, Linux e AIX.

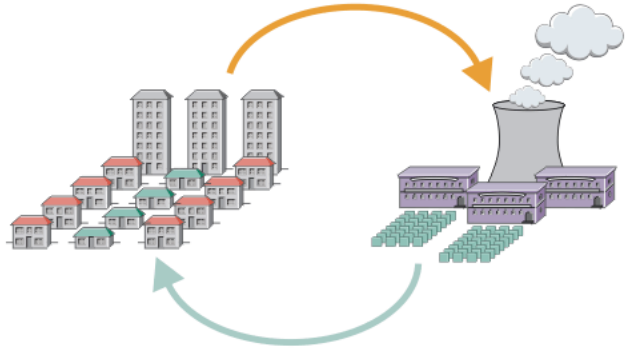
Windows Linux AIX

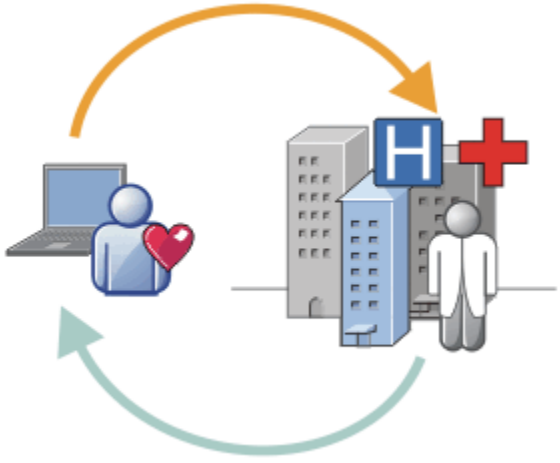
Casos de Uso de Telemetria

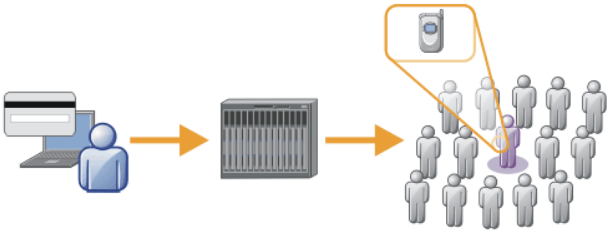
A Telemetria é uma tecnologia que permite sensoriamento, medição de dados e controle de dispositivos remotos de forma automatizada. A ênfase está na transmissão de dados a partir de dispositivos para um ponto de controle central. A telemetria também inclui o envio de informações de configuração e controle para dispositivos.

O MQ Telemetry conecta pequenos dispositivos usando o MQTT protocol e conecta os dispositivos com outros aplicativos usando IBM MQ. MQ Telemetry liga os dispositivos e a internet tornando mais fácil construir "soluções inteligentes". Soluções inteligentes desvendam a riqueza de informações disponíveis na Internet, e nos aplicativos corporativos, para aplicativos que monitoram e controlam dispositivos.

Os diagramas a seguir demonstram alguns usos típicos do MQ Telemetry:

Telemetria: Eletricidade Inteligente	
	• Mensagem do MQTT contendo dados de uso de energia enviados ao provedor de serviços. • O MQ Telemetry envia CONTROL COMMANDS com base na análise de dados de uso de energia. • Para obter mais informações, consulte o caso de uso a seguir: “Caso de Uso de Telemetria: Controle e Monitoramento de Energia Doméstica” na página 117

Telemetria: Serviços de Saúde Inteligentes	
• O MQ Telemetry envia dados de saúde para o seu médico e hospital. • Alertas ou feedback de mensagens do MQTT podem ser enviados com base na análise dos Dados de saúde. • Para obter mais informações, consulte o caso de uso a seguir: “Caso de Uso de Telemetria: Monitoramento de Pacientes no Lar” na página 115	

Telemetria: Um na Multidão	
	• Uma única transação de cartão é enviada ao servidor do banco. • O MQ Telemetry identifica a pessoa entre milhares, alertando o cliente de que seu cartão foi usado. • O MQ Telemetry pode usar a entrada de informações mais simples e localizar esse indivíduo.

Os casos de uso descritos nos subtópicos são obtidos de exemplos reais. Eles ilustram algumas maneiras de usar telemetria, e alguns dos problemas comuns que a tecnologia de telemetria deve resolver.

Windows Linux AIX Caso de Uso de Telemetria: Monitoramento de Pacientes no Lar

Na colaboração entre a IBM e um provedor de assistência médica em um sistema de saúde para pacientes cardíacos, um desfibrilador cardioversor implantável se comunica com um hospital. Dados sobre o paciente e o dispositivo implantado são transferidos com o uso de telemetria RF para o dispositivo MQTT na casa de um paciente.

Normalmente a transferência acontece toda noite para um transmissor localizado ao lado. O transmissor transfere os dados com segurança por meio de sistema de telefonia para o hospital, onde os dados são analisados.

O sistema reduz o número de visitas que um paciente deve fazer ao médico. Ele detecta quando o paciente ou o dispositivo precisa de atenção, e em caso de emergência, alerta o médico de plantão.

A colaboração entre a IBM e o provedor de assistência médica tem características que são comuns para um número de casos de uso de telemetria :

Invisibilidade

O dispositivo não requer intervenção do usuário além de fornecer energia, uma linha telefônica e estar próximo do dispositivo em uma parte do dia. Sua operação é confiável e ele é simples de usar.

Para remover a necessidade de o paciente configurar o dispositivo, o fornecedor do dispositivo faz sua pré-configuração. O paciente só precisa ligá-lo. A eliminação de configuração por parte do paciente simplifica a operação do dispositivo e reduz as chances de ele ser configurado de maneira incorreta.

O cliente MQTT é integrado como parte do dispositivo. O desenvolvedor do dispositivo integra a implementação do cliente MQTT no dispositivo e o desenvolvedor ou fornecedor configura o cliente MQTT como parte da pré-configuração.

O cliente MQTT é enviado como um arquivo JAR do Java SE, que o desenvolvedor inclui em seu aplicativo Java. Para ambientes não-Java, como este, o desenvolvedor do dispositivo pode implementar um cliente em uma linguagem diferente usando os formatos do MQTT e protocolos publicados. Alternativamente, o desenvolvedor pode usar um dos clientes C fornecidos como bibliotecas compartilhadas para as plataformas Windows, Linux e ARM.

Conectividade irregular

A comunicação entre o desfibrilador e o hospital tem características de rede irregulares. Duas redes diferentes são usadas para resolver problemas diferentes de coleção de dados do paciente e de envio de dados para o hospital. Entre a patente e o dispositivo MQTT, uma rede RF de baixa potência e de curto alcance é usada. O transmissor se conecta ao hospital usando uma conexão de VPN TCP/IP por uma linha telefônica de baixa largura de banda.

Muitas vezes não é prático encontrar uma maneira de conectar cada dispositivo diretamente a uma rede de protocolo da Internet. O uso de duas redes, conectadas por um hub, é uma solução comum.

O dispositivo MQTT é um hub simples, que armazena informações do paciente e encaminha para o hospital.

Segurança

O médico deve estar preparado para confiar na autenticidade dos dados do paciente, e o paciente quer que a privacidade de seus dados seja respeitada.

Em algumas situações, é suficiente criptografar a conexão, usando VPN ou TLS. Em outras situações, é desejável manter os dados seguros, mesmo após eles terem sido armazenados.

Às vezes o dispositivo de telemetria não está seguro. Ele pode estar em uma residência compartilhada, por exemplo. O usuário do dispositivo deve ser autenticado para assegurar que os dados sejam do paciente correto. O dispositivo em si pode ser autenticado para o servidor usando TLS e o servidor autenticado para o dispositivo.

O canal de telemetria entre dispositivo e o gerenciador de filas suporta JAAS para autenticação do usuário e TLS para criptografia de comunicação e autenticação de dispositivo. O acesso a uma publicação é controlado pelo gerenciador de autoridade de objeto no IBM MQ.

O identificador usado para autenticação do usuário pode ser mapeado para um identificador diferente, como a identidade de um paciente comum. Um identificador comum simplifica a configuração da autorização para tópicos de publicação no IBM MQ.

Conectividade

A conexão entre o dispositivo MQTT e o hospital usa discagem, e trabalha com largura de banda baixa, como 300 bauds.

Para operar efetivamente a 300 baud, o MQTT protocol inclui apenas alguns bytes extras em uma mensagem além dos cabeçalhos TCP/IP.

O MQTT protocol fornece o sistema de mensagens *fire-forget* de transmissão única, que mantém as latências baixas. Ele também poderá usar múltiplas transmissões para garantir *pelo menos uma vez* e *exatamente uma* entrega, se a entrega garantida for mais importante do que o tempo de resposta. Para garantia de entrega, as mensagens são armazenadas no dispositivo até terem sido entregues com sucesso. Se um dispositivo for conectado sem fio, a garantia de entrega será especialmente útil.

Escalabilidade

Os dispositivos de telemetria normalmente são implementados em grandes números, de dezenas de milhares a milhões.

A conexão de vários dispositivos a um sistema representa grandes demandas em uma solução. Existem demandas de negócios, como o custo dos dispositivos e seu software, e demandas de administração, como gerenciamento de licenças, dispositivos e usuários. Demandas técnicas incluem carregamentos na rede e nos servidores.

Abrir uma conexão exige mais recursos do servidor do que manter as conexões abertas. Mas em um caso de uso como esse, que usa linhas telefônicas, as despesas de conexão significam que as conexões são deixadas abertas não mais do que o necessário. As transferências de dados são feitas basicamente em lote. As conexões podem ser planejadas para a noite toda para evitar picos repentinos de conexões na hora de dormir.

No cliente, a escalabilidade dos clientes é beneficiada pelo mínimo de configuração de cliente requerida. O cliente de MQTT é integrado ao dispositivo. Não há requisitos de configuração ou etapa de aceitação da licença do cliente MQTT para integração com a implementação de dispositivos para pacientes.

No servidor, o MQ Telemetry tem uma meta inicial de 50.000 conexões abertas por gerenciador de filas.

As conexões são gerenciadas usando o IBM MQ Explorer. O IBM MQ Explorer filtra as conexões a serem exibidas para um número gerenciável. Com um esquema, adequadamente escolhido, para alocação de identificadores para clientes, você pode filtrar conexões com base na geografia ou em ordem alfabética por nome de paciente.

Windows Linux AIX Caso de Uso de Telemetria: Controle e Monitoramento de Energia Doméstica

Medidores inteligentes coletam mais detalhes sobre o consumo de energia do que os medidores tradicionais.

Os medidores inteligentes muitas vezes são acoplados a uma rede de telemetria local para monitorar e controlar dispositivos individuais em uma casa. Alguns também são conectados remotamente para monitoramento e controle a uma certa distância.

A conexão remota poderia ser configurada por um indivíduo, por um utilitário de energia ou por um ponto de controle central. O ponto de controle remoto pode ler o uso de energia e fornecer dados de uso. Ele pode fornecer dados para influenciar o uso, como informações contínuas sobre o tempo e preço. Ele pode limitar a carga para melhorar a eficiência de geração de energia geral.

Medidores inteligentes estão começando a ser amplamente implementados. O governo do Reino Unido, por exemplo, organizou um conselho para discutir sobre a implementação de medidores inteligentes em cada domicílio até 2020.

Os casos de uso de medidor do lar possuem um número de características comuns:

Invisibilidade

A menos que o usuário queira se envolver na economia de energia usando o medidor, o medidor não requer intervenção do usuário. Isso não deve reduzir a confiabilidade do fornecimento de energia para dispositivos individuais.

Um cliente MQTT pode ser integrado no software implementado com o medidor e não requer instalação nem configuração separadas.

Conectividade irregular

A comunicação entre dispositivos e medidores inteligentes demanda normas de conectividade diferentes das que existem entre o medidor e o ponto de conexão remota.

A conexão do medidor inteligente com dispositivos deve ser altamente disponível e compatível com as normas de rede para uma rede doméstica.

É provável que a rede remota use várias conexões físicas. Algumas delas, como um celular, têm um alto custo de transmissão e podem ser intermitentes. A especificacão MQTT v3 tem como objetivo as conexões remotas e as conexões entre adaptadores locais e o medidor inteligente.

As conexões entre tomadas de energia e dispositivos e o medidor usam uma rede doméstica, como Zigbee. MQTT para redes de sensores (MQTT-S) foi projetado para trabalhar com Zigbee e outros protocolos de rede de baixa largura de banda. MQ Telemetry não suporta MQTT-S diretamente. Ele requer um gateway para conectar o MQTT-S ao MQTT v3.

Assim como o monitoramento de pacientes em casa, as soluções para monitoramento e controle de energia doméstica requerem diversas redes, conectadas usando um medidor inteligente como um hub.

Segurança

Há inúmeros problemas de segurança associados aos medidores inteligentes. Esses problemas incluem não repúdio de transações, autorização de todas as ações de controle que são iniciadas e privacidade dos dados de consumo de energia.

Para assegurar a privacidade, os dados transferidos entre o medidor e o ponto de controle remoto pelo MQTT podem ser criptografados usando TLS. Para assegurar a autorização de ações de controle, a conexão do MQTT entre o medidor e o ponto de controle remoto pode ser mutuamente autenticada usando TLS.

Conectividade

A natureza física da rede remota pode variar consideravelmente. Ela pode usar uma conexão de banda larga existente, ou uma rede remota com alto custo de chamadas e disponibilidade

intermitente. O MQTT é um protocolo eficiente e confiável para conexões intermitentes e de alto custo. Consulte “Caso de Uso de Telemetria: Monitoramento de Pacientes no Lar” na página 115.

Escalabilidade

Eventualmente as empresas de energia, ou pontos de controle central, planejam a implementação de dezenas de milhares de medidores inteligentes. Inicialmente, os números de medidores por implementação estão entre dezenas e centenas de milhares. Esse número é comparável com o destino inicial do MQTT de 50.000 conexões de cliente abertas por gerenciador de filas.

Um aspecto crítico da arquitetura para monitoramento e controle de energia doméstica é o uso do medidor inteligente como um concentrador de rede. Cada adaptador de dispositivo é um sensor separado. Por eles são conectados a um hub local usando o MQTT, o hub pode concentrar os fluxos de dados em uma única sessão TCP/IP com o ponto de controle central e também armazenar mensagens por um curto período para superar indisponibilidades na sessão.

As conexões remotas devem ser deixadas abertas em casos de uso de energia doméstica por duas razões. Primeiro porque a abertura de conexões leva muito tempo em relação ao envio de solicitações. O tempo que se leva para abrir várias conexões para enviar solicitações de "limitação de carga" em um curto intervalo é muito longo. Segundo, para receber solicitações de limitação de carga da empresa de energia, primeiro a conexão deve ser aberta pelo cliente. Com o MQTT, as conexões são sempre iniciadas pelo cliente, e para receber solicitações de limitação de carga da empresa de energia, a conexão deve ser deixada aberta.

Se a taxa de abertura de conexões for crítica ou se o servidor iniciar solicitações com tempo crítico, normalmente a solução será manter várias conexões abertas.

Windows Linux AIX Casos de Uso de Telemetria: Radio Frequency Identification (RFID)

RFID é o uso de uma identificação RFID integrada para identificar e controlar um objeto por meio de modo wireless. Identificações RFID podem ser lidas a vários metros de distância e fora da linha de visão do leitor de RFID. Identificações passivas são ativadas por um leitor de RFID. Identificações ativas fazem transmissões sem ativação externa. As identificações ativas devem ter uma fonte de alimentação. As identificações passivas podem incluir uma fonte de alimentação para aumentar sua distância.

O RFID é usado em vários aplicativos e os tipos de casos de uso variam bastante. Os casos de uso do RFID e os casos de monitoramento de pacientes no lar e monitoramento e controle de energia do lar têm algumas semelhanças e diferenças.

Invisibilidade

Em muitos casos de uso, o leitor de RFID é implementado em grandes números e deve trabalhar sem intervenção do usuário. O leitor inclui um cliente integrado do MQTT para se comunicar com um ponto de controle central.

Por exemplo, em um armazém de distribuição, um leitor usa um sensor de movimentos para detectar uma paleta. Ele ativa identificações RFID de itens na paleta e envia dados e solicitações para aplicativos centrais. Os dados são usados para atualização do local do estoque. As solicitações controlam o que acontece com a paleta em seguida, como sua mudança para um compartimento específico. Sistemas de bagagens de aeroportos e linhas aéreas estão usando RFID dessa forma.

Em alguns casos de uso do RFID, o leitor possui um ambiente de computação padrão, como Java Platform, Micro Edition (Java ME). Nesses casos, o cliente do MQTT pode ser implementado em uma etapa de configuração distinta, após a fabricação.

Conectividade irregular

Os leitores de RFID podem ser separados do dispositivo de controle local que contém um cliente MQTT, ou cada leitor pode integrar um cliente MQTT. Normalmente, fatores geográficos ou de comunicações indicam a escolha da topologia.

Segurança

Privacidade e autenticidade são questões de segurança para a anexação de identificações RFID. As identificações RFID são reservadas e podem ser monitoradas, falsificadas e corrompidas secretamente.

A solução dos problemas de segurança de RFID aumenta a oportunidade de implementação de novas soluções de RFID. Embora haja exposição da segurança na identificação RFID, e no leitor local, o uso de processamento de informações central sugere abordagens para reagir a diferentes ameaças. Por exemplo, a violação de uma identificação pode ser detectada correlacionando dinamicamente os níveis de estoque com as entregas e os envios.

Conectividade

Normalmente os aplicativos RFID envolviam armazenamento e encaminhamento em lote de informações reunidas em consultas imediatas e leitores de RFID. No caso de uso de armazém de distribuição, o leitor de RFID fica conectado o tempo todo. Quando uma identificação é lida, ela é publicada junto com as informações sobre o leitor. O aplicativo warehousing publica a resposta de volta para o leitor.

No aplicativo warehousing, a rede é geralmente confiável e as solicitações imediatas podem usar mensagens *fire-forget* para desempenho de baixa latência. Os dados de armazenamento e encaminhamento em lote podem usar o sistema de mensagens *exatamente uma vez* para minimizar os custos de administração associados à perda de dados.

Escalabilidade

Se o aplicativo RFID requerer respostas imediatas, em um ou dois segundos, os leitores de RFID deverão ficar conectados.

Windows Linux AIX Casos de Uso de Telemetria: Sensoriamento de Ambiente

O sensoriamento de ambiente usa telemetria para coletar informações sobre qualidade e níveis de água dos rios, poluentes atmosféricos e outros dados ambientes.

Os sensores localizam-se frequentemente em locais remotos, sem acesso à comunicação com fio. A largura de banda wireless é cara e a confiabilidade pode ser baixa. Normalmente inúmeros sensores de ambiente em uma pequena área geográfica são conectados a um dispositivo de monitoramento local em um local seguro. As conexões locais podem ser com e sem fio.

Invisibilidade

É provável que dispositivos sensores sejam menos acessíveis, tenham potência mais baixa e sejam implementados em maiores números do que um dispositivo de monitoramento central. Às vezes os sensores não conseguem se comunicar e o dispositivo de monitoramento local inclui adaptadores para transformar e armazenar dados do sensor. O dispositivo de monitoramento provavelmente irá incorporar um computador de propósito geral que suporta o Java Platform, Standard Edition (Java SE) ou o Java Platform, Micro Edition (Java ME). A invisibilidade dificilmente será um grande requisito ao configurar o cliente do MQTT.

Conectividade irregular

A capacidade dos sensores e o custo e a largura de banda da conexão remota normalmente resultam em um hub de monitoramento local conectado a um servidor central.

Segurança

A menos que a solução esteja sendo usada em um caso de uso militar ou de defesa, a segurança não é o principal requisito.

Conectividade

Muitos usos não requerem monitoramento contínuo ou disponibilidade imediata dos dados. Dados de exceção, como um alerta de nível de enchente, não precisam ser encaminhados diretamente. Dados do sensor são agregados no monitor local para reduzir custos de conexão e comunicação e depois

transferidos com o uso de conexões planejadas. Dados de exceção são encaminhados assim que detectados no monitor.

Escalabilidade

Os sensores estão concentrados ao redor de hubs locais e seus dados estão agregados em pacotes que são transmitidos de acordo com um planejamento. Ambos esses fatores reduzem a carga no servidor central que seria imposta pelo uso direto de sensores conectados.

Windows Linux AIX Casos de Uso de Telemetria: Aplicativos Remotos

Aplicativos remotos são aqueles executados em dispositivos wireless. Os dispositivos são plataformas de aplicativo genéricas ou dispositivos customizados.

Plataformas gerais incluem dispositivos portáteis, como telefones, Personal Data Assistants e notebooks. Dispositivos customizados usam hardware de propósito especial padronizado para aplicativos específicos. Um dispositivo para registrar a entrega de pacotes "assinada" é um exemplo de dispositivo móvel customizado. Aplicativos em dispositivos móveis customizados muitas vezes são baseados em plataformas de software genéricas.

Invisibilidade

A implementação de aplicativos móveis customizados é gerenciada e pode incluir configuração do aplicativo cliente MQTT. Invisibilidade não deve ser o principal requisito durante a configuração do cliente de MQTT.

Conectividade irregular

Ao contrário da topologia de hub local dos casos de uso anteriores, os clientes móveis se conectam remotamente. A camada do aplicativo cliente se conecta diretamente a um aplicativo no hub central.

Segurança

Com pouca segurança física, o dispositivo móvel e o usuário remoto devem ser autenticados. O TLS é usado para confirmar a identidade do dispositivo e o JAAS para autenticar o usuário.

Conectividade

Se o aplicativo remoto depender de cobertura wireless, ele deverá ter capacidade para operar no modo off-line e para lidar efetivamente com uma conexão interrompida. Nesse ambiente, o objetivo é ficar conectado, mas o aplicativo deve ter capacidade de armazenar e encaminhar mensagens. Muitas vezes as mensagens são pedidos ou confirmações de entrega, e têm valor comercial importante. Elas precisam ser armazenadas e encaminhadas de forma confiável.

Escalabilidade

A escalabilidade não é um grande problema. O número de aplicativos clientes são não deve exceder milhares ou dezenas de milhares, em casos de uso de aplicativo remoto customizado.

Windows Linux AIX Conectando Dispositivos de Telemetria a um Gerenciador de Filas

Dispositivos de telemetria se conectam a um gerenciador de filas usando um cliente MQTT v3. O cliente MQTT v3 usa TCP/IP para se conectar a um listener TCP/IP chamado de serviço de telemetria (MQXR).

Quando você conectar um dispositivo de telemetria a um gerenciador de filas, o cliente do MQTT iniciará uma conexão TCP/IP usando o método `MqttClient.connect`. Como os clientes IBM MQ, um cliente MQTT deve ser conectado ao gerenciador de filas para enviar e receber mensagens. A conexão é estabelecida no servidor usando um listener TCP/IP, instalado com o MQ Telemetry, chamado de serviço de telemetria (MQXR). Cada gerenciador de filas executa no máximo um serviço de telemetria (MQXR).

O serviço de telemetria (MQXR) usa o endereço de soquete remoto configurado por cada cliente no método `MqttClient.connect` para alocar a conexão com um canal de telemetria. Um endereço de

soquete é a combinação do número da porta e do nome do host TCP/IP. Diversos clientes que usam o mesmo endereço de soquete remoto são conectados ao mesmo canal de telemetria pelo serviço de telemetria (MQXR).

Se houver diversos gerenciadores de filas em um servidor, divida os canais de telemetria entre os gerenciadores de filas. Aloque os endereços de soquete remotos entre os gerenciadores de filas. Defina cada canal de telemetria com um endereço de soquete remoto exclusivo. Dois canais de telemetria não devem usar o mesmo endereço de soquete.

Se o mesmo endereço de soquete remoto for configurado para canais de telemetria em diversos gerenciadores de filas, o primeiro canal de telemetria para conexão tem precedência. Os canais subsequentes que se conectam no mesmo endereço falham.

Se houver diversos adaptadores de rede no servidor, divida os endereços de soquete remotos entre canais de telemetria. A alocação de endereços de soquete é totalmente arbitrária, contanto que todo endereço de soquete específico seja configurado em apenas um canal de telemetria.

Configure o IBM MQ para conectar clientes MQTT usando os assistentes fornecidos no suplemento do MQ Telemetry para o IBM MQ Explorer. Alternativamente, siga as instruções em Configurando um Gerenciador de Filas para Telemetria em Linux e AIX e Configurando um Gerenciador de Filas para Telemetria em Windows para configurar a telemetria manualmente.

Referências relacionadas

Propriedades MQXR

Windows Linux AIX Protocolos de Conexão do Telemetry

O MQ Telemetry suporta TCP/IP IPv4 e IPv6 e TLS.

Windows Linux AIX Serviço de telemetria (MQXR)

O serviço de telemetria (MQXR) é um listener TCP/IP que é gerenciado como um serviço do IBM MQ. Crie o serviço usando um assistente do IBM MQ Explorer ou com um comando **runmqsc**.

O serviço MQ Telemetry (MQXR) é chamado SYSTEM.MQXR.SERVICE ..

O assistente de Configuração de amostra de telemetria, fornecido na função do MQ Telemetry para o IBM MQ Explorer, cria o serviço de telemetria e um canal de telemetria de amostra; veja Verificando a instalação do MQ Telemetry usando o IBM MQ Explorer .

Crie a configuração de amostra a partir da linha de comandos; consulte Verificando a instalação do MQ Telemetry usando a linha de comandos.

O serviço de telemetria (MQXR) começa a para automaticamente com o gerenciador de filas. Controle o serviço usando a pasta de serviços no IBM MQ Explorer. Para ver o serviço, deve-se clicar no ícone para parar IBM MQ Explorer a filtragem de objetos SYSTEM da exibição.

Para obter um exemplo de como criar o serviço manualmente, consulte

- Linux AIX Criando o SYSTEM.MQXR.SERVICE no Linux
- Windows Criando o SYSTEM.MQXR.SERVICE no Windows

Esses tópicos também especificam a chave padrão para requerer que passphrases para canais TLS do MQTT sejam criptografados. Para obter mais informações, consulte Criptografia de senha para canais TLS do MQTT.

Windows Linux AIX Canais de Telemetria

Crie canais de telemetria para criar conexões com propriedades diferentes, como Java Authentication and Authorization Service (JAAS) ou autenticação TLS, ou para gerenciar grupos de clientes.

Crie canais de telemetria usando o assistente **New Telemetry Channel** , fornecido na função MQ Telemetry para IBM MQ Explorer. Configure um canal, usando o assistente, para aceitar conexões de

clientes MQTT em uma porta TCP/IP específica. Desde a IBM WebSphere MQ 7.1, é possível configurar o MQ Telemetry usando o programa da linha de comandos, **runmqsc**.

Crie diversos canais de telemetria, em diferentes portas, para tornar grandes números de conexões do cliente fáceis de gerenciar, dividindo os clientes em grupos. Cada canal de telemetria tem um nome diferente.

É possível configurar canais de telemetria com diferentes atributos de segurança para criar diferentes tipos de conexão. Crie diversos canais para aceitar conexões do cliente em diferentes endereços TCP/IP. Use o TLS para criptografar mensagens e autenticar o canal de telemetria e o cliente. Consulte Configuração do TLS de clientes e canais de telemetria do MQTT. Especifique o ID do usuário para simplificar a autorização do acesso aos objetos do IBM MQ. Especifique uma configuração do JAAS para autenticar o usuário do MQTT com o JAAS. Consulte Identificação, autorização e autenticação do cliente MQTT.

Windows Linux AIX Protocolo IBM MQ Telemetry Transport

O protocolo IBM MQ Telemetry Transport (MQTT) v3 foi projetado para trocar mensagens entre pequenos dispositivos em baixa largura de banda, ou conexões caras, e para enviar mensagens no modo confiável. Ele usa TCP/IP.

O MQTT protocol está publicado; consulte Formato e protocolo do IBM MQ Telemetry Transport. A versão 3 do protocolo usa publicação/assinatura e suporta três qualidades de serviço: *fire-forget*, *pelo menos uma vez* e *exatamente uma vez*.

O tamanho pequeno dos cabeçalhos do protocolo e a carga útil da mensagem da matriz de bytes mantêm as mensagens pequenas. Os cabeçalhos abrangem um cabeçalho fixo de 2 bytes e até 12 bytes de cabeçalhos variáveis adicionais. O protocolo usa cabeçalhos variáveis de 12 bytes para se subscrever e se conectar e cabeçalhos variáveis de apenas 2 bytes para a maioria das publicações.

Com três qualidades de serviço, é possível fazer trocas entre baixa latência e confiabilidade; consulte Qualidades de serviço fornecidas por um cliente MQTT. *Fire-forget* não usa nenhum armazenamento de dispositivo persistente e apenas uma transmissão para enviar ou receber uma publicação. *Pelo menos uma vez* e *exatamente uma vez* requerem armazenamento persistente no dispositivo para manter o estado do protocolo e salvar uma mensagem até que seja reconhecida.

Windows Linux AIX Clientes MQTT

Um aplicativo do cliente MQTT é responsável por coletar informações do dispositivo de telemetria, conectando-se ao servidor e publicando as informações para o servidor. Ele também pode se subscrever a tópicos, receber publicações e controlar o dispositivo de telemetria.

Diferente dos aplicativos do cliente IBM MQ, os aplicativos do clienteMQTT não são aplicativos IBM MQ. Eles não especificam um gerenciador de filas para conexão. Eles não se limitam a usar interfaces de programação específicas do IBM MQ. Em vez disso, clientes do MQTT implementam o protocolo MQTT 3. É possível escrever sua própria biblioteca do cliente para interface com o MQTT protocol na linguagem de programação e na plataforma de sua escolha. Consulte IBM MQ Telemetry Transport formato e protocolo.

Para simplificar a gravação de aplicativos clientes do MQTT, use as bibliotecas do cliente C, Java, e JavaScript que encapsulam o MQTT protocol para diversas plataformas. Se você incorporar essas bibliotecas em seus apps MQTT, um cliente do MQTT totalmente funcional poderá ser tão curto quanto 15 linhas de código. As bibliotecas do cliente do MQTT estão disponíveis livremente por meio do Eclipse Paho e do MQTT.org. Consulte Programas de amostra do IBM MQ Telemetry Transport.

O aplicativo do cliente MQTT é sempre responsável por iniciar uma conexão com um canal de telemetria. Após a conexão, o aplicativo cliente do MQTT ou um aplicativo IBM MQ pode iniciar uma troca de mensagens.

Os aplicativos do cliente MQTT e os aplicativos do IBM MQ publicam e assinam o mesmo conjunto de tópicos. Um aplicativo IBM MQ também pode enviar uma mensagem diretamente para um aplicativo cliente MQTT sem que o aplicativo cliente crie primeiro uma assinatura. Consulte Configurar enfileiramento distribuído para enviar mensagens para clientes MQTT.

Aplicativos do cliente MQTT são conectados ao IBM MQ usando um canal de telemetria. O canal de telemetria age como uma ponte entre os diferentes tipos de mensagem usados pelo MQTT e IBM MQ. Ele cria publicações e assinaturas no gerenciador de filas em nome do cliente do aplicativo MQTT. O canal de telemetria envia publicações que correspondem às assinaturas de um aplicativo do cliente MQTT do gerenciador de filas para o aplicativo do cliente MQTT.

Windows Linux AIX Envie uma mensagem para um cliente MQTT

Os aplicativos IBM MQ podem enviar mensagens de clientes do MQTT v3 publicando em assinaturas criadas por clientes ou enviando mensagens diretamente. Os clientes MQTT os podem enviar mensagens para outros publicando em tópicos assinados por outros clientes.

Um cliente MQTT assina uma publicação, que ele recebe do IBM MQ

Execute a tarefa, “Publicando uma mensagem para o utilitário do cliente MQTT a partir do IBM MQ Explorer” na página 125 para enviar uma publicação do IBM MQ para um cliente de MQTT.

A maneira padrão de um cliente MQTT v3 receber mensagens é criando uma assinatura para um tópico ou conjunto de tópicos. No fragmento de código de exemplo, Figura 44 na página 124, o cliente MQTT assina usando a sequência de tópicos `"MQTT Examples"`. Um IBM MQ aplicativo C Figura 45 na página 124publica no tópico usando a sequência de tópicos `"MQTT Examples"`. No fragmento de código Figura 46 na página 124, o cliente MQTT recebe a publicação no método de retorno de chamada, `messageArrived`.

Para obter informações adicionais sobre como configurar o IBM MQ para enviar publicações em resposta a assinaturas de clientes do MQTT, consulte Publicando uma mensagem em resposta a uma assinatura do cliente do MQTT.

Um aplicativo IBM MQ envia uma mensagem diretamente para um cliente MQTT

Execute a tarefa, “Enviando uma mensagem para um cliente MQTT usando o IBM MQ Explorer” na página 129, para enviar uma mensagem diretamente do IBM MQ para um cliente MQTT.

Uma mensagem enviada dessa forma para um cliente MQTT é chamada de mensagem não solicitada. Os clientes MQTT v3 recebem mensagens não solicitadas como publicações com um nome de tópico configurado. O serviço de telemetria (MQXR) configura o nome do tópico para o nome da fila remota.

Para obter mais informações sobre como configurar o IBM MQ para enviar mensagens diretamente para clientes do MQTT, consulte Enviando uma mensagem para um cliente diretamente.

Um cliente MQTT publica uma mensagem

Um cliente MQTT v3 pode publicar uma mensagem que é recebida por outro cliente de MQTT v3, mas não pode enviar uma mensagem não solicitada. O fragmento de código Figura 47 na página 125 mostra como um cliente MQTT v3, gravado em Java, publica uma mensagem.

O padrão típico para enviar uma mensagem para um cliente específico do MQTT v3, é cada cliente criar uma assinatura para seu próprio `ClientIdentifier`. Execute a tarefa “Publicando uma mensagem para um cliente MQTT v3” na página 130 para publicar uma mensagem de um cliente MQTT para outro cliente MQTT usando `ClientIdentifier` como uma sequência de tópicos.

Fragmentos de Código de Exemplo

O fragmento de código em Figura 44 na página 124 mostra como um cliente MQTT gravado em Java cria uma assinatura. Ele também precisa de um método de retorno de chamada, `messageArrived`, para receber publicações para a assinatura.

```
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. Assinante do cliente MQTT v3

O fragmento de código em Figura 45 na página 124 mostra como um aplicativo IBM MQ gravado em C envia uma publicação. O fragmento de código é extraído da tarefa Criar um publicador para um tópico de variável

```
/* 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 do IBM MQ

Quando a publicação chega, o cliente MQTT chama o método `messageArrived` da classe do MQTT aplicativo cliente `MqttCallback`

```
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 `messageArrived`

Figura 47 na página 125 mostra um MQTT v3 publicando uma mensagem para a assinatura criada em Figura 44 na página 124.

```
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 do cliente MQTT v3

Windows Linux AIX Publicando uma mensagem para o utilitário do cliente MQTT a partir do IBM MQ Explorer

Siga as etapas nesta tarefa para publicar uma mensagem usando o IBM MQ Explorer e assine-a com o utilitário do cliente MQTT. Uma tarefa adicional mostra como configurar um alias de gerenciador de filas em vez de configurar a fila de transmissão padrão para `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Antes de começar

A tarefa assume que você esteja familiarizado com o IBM MQ e o IBM MQ Explorer e que o IBM MQ e o recurso MQ Telemetry estejam instalados.

O usuário criando os recursos do gerenciador de filas para esta tarefa deve ter autoridade suficiente para isso. Para fins de demonstração, o ID do usuário IBM MQ Explorer é assumido como um membro do grupo `mqm`.

Sobre esta tarefa

Na tarefa, você cria um tópico no IBM MQ e subscreve-se para o tópico usando o utilitário do cliente MQTT. Quando você publica para o tópico usando o IBM MQ Explorer, o cliente MQTT recebe a publicação.

Procedimento

Execute uma das seguintes tarefas:

- Você instalou o MQ Telemetry, mas ainda não o iniciou. Execute a tarefa: “Iniciando Tarefa sem Serviço de Telemetria (MQXR) Definido” na página 126.
- Você executou o IBM MQ Telemetry antes, mas quer usar um novo gerenciador de filas para fazer a demonstração. Execute a tarefa: “Iniciando Tarefa sem Serviço de Telemetria (MQXR) Definido” na página 126.
- Você quer executar a tarefa usando um gerenciador de filas existente que não tem recursos de telemetria definidos. Você não quer executar o assistente **Definir Configuração de Amostra**.
  a. Execute uma das seguintes tarefa para configurar a telemetria:
     – Configurando um gerenciador de filas para telemetria no Linux e no AIX
     – Configurando um gerenciador de filas para telemetria no Windows
  b. Execute a tarefa: “Iniciando Tarefa com Serviço de Telemetria (MQXR) em Execução” na página 127
- Se quiser executar a tarefa usando um gerenciador de filas existente que já tem recursos de telemetria definidos, execute a tarefa: “Iniciando Tarefa com Serviço de Telemetria (MQXR) em Execução” na página 127.

Como proceder a seguir

Execute "Enviando uma mensagem para um cliente MQTT usando o IBM MQ Explorer" na página 129 para enviar uma mensagem diretamente para o utilitário do cliente.

Iniciando Tarefa sem Serviço de Telemetria (MQXR) Definido

Crie um gerenciador de filas e execute **Definir Configuração de Amostra** para definir recursos de telemetria de amostra para o gerenciador de filas. Publique uma mensagem usando o IBM MQ Explorer e assine-a com o utilitário do cliente MQTT.

Sobre esta tarefa

Quando você configura recursos de telemetria de amostra usando **Definir Configuração de Amostra**, o assistente configura as permissões de ID do usuário guest. Considere atentamente se você quer que o ID do usuário guest seja autorizado dessa forma. `guest` on Windowse `nobody` on Linuxrecebem permissão para publicar e assinar a raiz da árvore de tópicos e para colocar mensagens em `SYSTEM.MQTT.TRANSMIT.QUEUE`.

O assistente também configura a fila de transmissão padrão como `SYSTEM.MQTT.TRANSMIT.QUEUE`, o que pode interferir nos aplicativos em execução em um gerenciador de filas existente. É possível, mas trabalhoso, configurar a telemetria e não usar a fila de transmissão padrão; execute a seguinte tarefa: "Usando um Alias de Gerenciador de Filas" na página 128. Nessa tarefa, você cria um gerenciador de filas para evitar a possibilidade de interferência em qualquer fila de transmissão padrão existente.

Procedimento

1. Usando o IBM MQ Explorer, crie e inicie um novo gerenciador de filas.
   a) Clique com o botão direito na `Queue Managers`Pasta do > **Novo** > **Gerenciador de filas...** Digite um nome de gerenciador de filas > **Concluir**.

      Crie um nome de gerenciador de filas; por exemplo, `MQTTQMGR`.
2. Crie e inicie o serviço de telemetria (MQXR) e crie um canal de telemetria de amostra.
   a) Abra a pasta `Queue Managers\`*`QmgrName`*`\Telemetry`.
   b) Clique em **Definir configuração de amostra...** > **Concluir**

      Deixe a caixa de seleção **Ativar utilitário do cliente MQTT** marcada.
3. Crie uma assinatura para `MQTT Example` usando o utilitário do cliente MQTT.
   a) Clique em **Conectar**.

      O **Histórico do Cliente** registra um evento do `Connected`
   b) Digite `MQTT Example` no campo **Assinatura\Tópico** > **Assinar**.

   O **Histórico do Cliente** registra um evento do `Subscribed`
4. Crie `MQTTExampleTopic` no IBM MQ.
   a) Clique com o botão direito na pasta `Queue Managers\`*`QmgrName`*`\Topics` no **MQ Explorer** > **Novo** > **Tópico**.
   b) Digite `MQTTExampleTopic` como o **Nome** > **Avançar**.
   c) Digite `MQTT Example` como a **Sequência de tópicos** > **Concluir**.
   d) Clique em **OK** para fechar a janela de reconhecimento.
5. Publicar `Hello World!` no tópico `MQTT Example` usando IBM MQ Explorer.
   a) Clique na pasta `Queue Managers\`*`QmgrName`*`\Topics` no IBM MQ Explorer.
   b) Clique com o botão direito em `MQTTExampleTopic` > **Publicação de teste...**
   c) Digite `Hello World!` no campo **Dados da mensagem** > **Publicar mensagem** > Alternar para a janela Utilitário do cliente do MQTT.

   O **Histórico do Cliente** registra um evento do `Received`

Iniciando Tarefa com Serviço de Telemetria (MQXR) em Execução

Crie um canal de telemetria e um tópico. Autorize o usuário a usar o tópico e a fila de transmissão de telemetria. Publique uma mensagem usando o IBM MQ Explorer e assine-a com o utilitário do cliente MQTT.

Antes de começar

Nesta versão da tarefa, um gerenciador de filas, *QmgrName*, está definido e em execução. Um serviço de telemetria (MQXR) está definido e em execução. O serviço de telemetria (MQXR) pode ter sido criado manualmente ou pela execução do assistente **Definir Configuração de Amostra**.

Sobre esta tarefa

Nesta tarefa você configura um gerenciador de filas existente para enviar uma publicação para o utilitário do cliente MQTT.

A etapa "1" na página 127 da tarefa configura a fila de transmissão padrão como `SYSTEM.MQTT.TRANSMIT.QUEUE`, o que pode interferir nos aplicativos em execução em um gerenciador de filas existente. É possível, mas trabalhoso, configurar a telemetria e não usar a fila de transmissão padrão; execute a seguinte tarefa: "Usando um Alias de Gerenciador de Filas" na página 128.

Procedimento

1. Configure `SYSTEM.MQTT.TRANSMIT.QUEUE` como a fila de transmissão padrão.
   a) Clique com o botão direito em `Queue Managers\`*`QmgrName`* `folder` > **Propriedades...**
   b) Clique em **Comunicação** no navegador.
   c) Clique em **Selecionar ...** > Selecionar ... `SYSTEM.MQTT.TRANSMIT.QUEUE` > **OK** > **OK**
2. Crie um canal de telemetria `MQTTExampleChannel` para conectar o utilitário do cliente MQTT ao IBM MQ e iniciar o utilitário do cliente MQTT.
   a) Clique com o botão direito na pasta `Queue Managers\`*`QmgrName`* `\Telemetry\Channels` no **MQ Explorer** > **Novo** > **Canal de telemetria...**.
   b) Digite `MQTTExampleChannel` no campo **Nome do canal** > **Avançar** > **Avançar**.
   c) Mude o **ID do usuário fixo** no painel de autorização do cliente para o ID do usuário que publicará e assinará `MQTTExample` > **Próximo**.
   d) Deixe **Ativar utilitário do cliente** marcado > **Concluir**.
3. Crie uma assinatura para `MQTT Example` usando o utilitário do cliente MQTT.
   a) Clique em **Conectar**.

      O **Histórico do Cliente** registra um evento do `Connected`
   b) Digite `MQTT Example` no campo **Assinatura\Tópico** > **Assinar**.

   O **Histórico do Cliente** registra um evento do `Subscribed`
4. Crie `MQTTExampleTopic` no IBM MQ.
   a) Clique com o botão direito na pasta `Queue Managers\`*`QmgrName`*`\Topics` no **MQ Explorer** > **Novo** > **Tópico**.
   b) Digite `MQTTExampleTopic` como o **Nome** > **Avançar**.
   c) Digite `MQTT Example` como a **Sequência de tópicos** > **Concluir**.
   d) Clique em **OK** para fechar a janela de reconhecimento.
5. Se você deseja que um usuário que não esteja no grupo `mqm` publique e assine o tópico `MQTTExample`, faça o seguinte:
   a) Autorize o usuário a publicar e assinar o tópico `MQTTExampleTopic`:

```
setmqaut -m qMgrName -t topic -n MQTTExampleTopic -p User ID -all +pub +sub
```

b) Autorize o usuário a colocar uma mensagem na SYSTEM.MQTT.TRANSMIT.QUEUE:

```
setmqaut -m qMgrName -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p User ID -all +put
```

6. Publicar Hello World! no tópico MQTT Example usando IBM MQ Explorer.
   a) Clique na pasta Queue Managers*QmgrName*\Topics no IBM MQ Explorer.
   b) Clique com o botão direito em MQTTExampleTopic > **Publicação de teste...**
   c) Digite Hello World! no campo **Dados da mensagem** > **Publicar mensagem** > Alternar para a janela Utilitário do cliente do MQTT.

   O **Histórico do Cliente** registra um evento do Received

Usando um Alias de Gerenciador de Filas

Publique uma mensagem no utilitário do cliente MQTT usando IBM MQ Explorer sem configurar a fila de transmissão padrão como SYSTEM.MQTT.TRANSMIT.QUEUE.

A tarefa é uma continuação da tarefa anterior e usa um alias de gerenciador de filas para evitar a configuração da fila de transmissão padrão como SYSTEM.MQTT.TRANSMIT.QUEUE.

Antes de começar

Conclua qualquer uma das tarefas, "Iniciando Tarefa sem Serviço de Telemetria (MQXR) Definido" na página 126 ou "Iniciando Tarefa com Serviço de Telemetria (MQXR) em Execução" na página 127.

Sobre esta tarefa

Quando um cliente MQTT cria uma assinatura, o IBM MQ envia sua resposta usando ClientIdentifier como o nome do gerenciador de filas remotas. Nessa tarefa, ele usa o ClientIdentifier, MyClient.

Se não houver fila de transmissão ou alias do gerenciador de filas chamado MyClient, a resposta será colocada na fila de transmissão padrão. Ao configurar a fila de transmissão padrão para SYSTEM.MQTT.TRANSMIT.QUEUE, o cliente MQTT obtém a resposta

É possível evitar a configuração da fila de transmissão padrão como SYSTEM.MQTT.TRANSMIT.QUEUE usando um alias de gerenciador de filas. Você deve configurar um alias de gerenciador de filas para cada ClientIdentifier. Normalmente há vários clientes para tornar prático o uso de alias de gerenciador de filas. Muitas vezes o ClientIdentifier é imprevisível, tornando impossível a configuração de telemetria dessa maneira.

Contudo, em algumas circunstâncias, você pode ter que configurar a fila de transmissão padrão para algo além de SYSTEM.MQTT.TRANSMIT.QUEUE. As etapas em Procedimento configuram um alias do gerenciador de filas em vez de configurar a fila de transmissão padrão para SYSTEM.MQTT.TRANSMIT.QUEUE

Procedimento

1. Remova SYSTEM.MQTT.TRANSMIT.QUEUE como a fila de transmissão padrão.
   a) Clique com o botão direito em Queue Managers*QmgrName* folder > **Propriedades...**
   b) Clique em **Comunicação** no navegador.
   c) Remova SYSTEM.MQTT.TRANSMIT.QUEUE do campo **Fila de transmissão padrão** > **OK**.
2. Verifique se não é mais possível criar uma assinatura com o utilitário do cliente MQTT:
   a) Clique em **Conectar**.

      O **Histórico do Cliente** registra um evento do Connected
   b) Digite MQTT Example no campo **Assinatura\Tópico** > **Assinar**.

O **Histórico do Cliente** registra um evento `Subscribe failed` e um evento `Connection lost`

3. Crie um alias de gerenciador de filas para `ClientIdentifier`, `MyClient`.
   a) Clique com o botão direito na pasta `Queue Managers\`*`QmgrName`*`\Queues` > **Novo** > **Definição de fila remota**.
   b) Dê à definição o nome `MyClient` > **Avançar**.
   c) Digite `MyClient` no campo **Gerenciador de Filas Remotas**.
   d) Digite `SYSTEM.MQTT.TRANSMIT.QUEUE` no campo **Fila de transmissão** > **Concluir**.
4. Conecte o utilitário do cliente MQTT novamente.
   a) Verifique se **Identificador de Cliente** está configurado como `MyClient`.
   b) **Conectar**

   O **Histórico do Cliente** registra um evento do `Connected`
5. Crie uma assinatura para `MQTT Example` usando o utilitário do cliente MQTT.
   a) Clique em **Conectar**.

      O **Histórico do Cliente** registra um evento do `Connected`
   b) Digite `MQTT Example` no campo **Assinatura\Tópico** > **Assinar**.

   O **Histórico do Cliente** registra um evento do `Subscribed`
6. Publicar `Hello World!` no tópico `MQTT Example` usando IBM MQ Explorer.
   a) Clique na pasta `Queue Managers\`*`QmgrName`*`\Topics` no IBM MQ Explorer.
   b) Clique com o botão direito em `MQTTExampleTopic` > **Publicação de teste...**
   c) Digite `Hello World!` no campo **Dados da mensagem** > **Publicar mensagem** > Alternar para a janela Utilitário do cliente do MQTT.

   O **Histórico do Cliente** registra um evento do `Received`

Windows Linux AIX Enviando uma mensagem para um cliente MQTT usando o IBM MQ Explorer

Envie uma mensagem para o utilitário do cliente MQTT colocando uma mensagem em uma fila do IBM MQ usando o IBM MQ Explorer. A tarefa mostra como configurar uma definição de fila remota para enviar uma mensagem diretamente para um cliente MQTT.

Antes de começar

Execute a tarefa “Publicando uma mensagem para o utilitário do cliente MQTT a partir do IBM MQ Explorer” na página 125. Deixe o utilitário do cliente de MQTT conectado.

Sobre esta tarefa

A tarefa demonstra o envio de uma mensagem para um cliente MQTT usando uma fila em vez de publicá-la para um tópico. Você não cria uma assinatura no cliente. A etapa “2” na página 130 da tarefa demonstra que a assinatura anterior foi excluída.

Procedimento

1. Descarte quaisquer assinaturas existentes desconectando e reconectando o utilitário do cliente MQTT.

   A assinatura é descartada porque, a menos que você altere os padrões, o utilitário do cliente MQTT se conecta a uma sessão limpa; consulte Sessões limpas.

   Para tornar mais fácil a execução da tarefa, digite seu próprio `ClientIdentifier`, em vez de usar o `ClientIdentifier` gerado criado pelo utilitário cliente de MQTT.

   a) Clique em **Desconectar** para desconectar o utilitário do cliente MQTT do canal de telemetria.

O **Histórico do Cliente** registra um evento `Disconnected`

b) Altere **Identificador de Cliente** para `MyClient`.

c) Clique em **Conectar**.

O **Histórico do Cliente** registra um evento `Connected`

2. Verifique se o utilitário do cliente do MQTT não recebe mais publicação para o `MQTTExampleTopic`.
   a) Clique na pasta `Queue Managers\`*`QmgrName`*`\Topics` no IBM MQ Explorer.
   b) Clique com o botão direito em `MQTTExampleTopic` > **Publicação de teste...**
   c) Digite `Hello World!` no campo **Dados da mensagem** > **Publicar mensagem** > Alternar para a janela Utilitário do cliente do MQTT.

   Nenhum evento é registrado no **Histórico do Cliente**.
3. Crie uma definição de fila remota para o cliente.

   Configure `ClientIdentifier`, `MyClient`, como o nome do gerenciador de filas remotas na definição de fila remota. Use qualquer nome como nome da fila remota. O nome da fila remota é passado para um cliente MQTT como o nome do tópico.
   a) Clique com o botão direito na pasta `Queue Managers\`*`QmgrName`*`\Queues` > **Novo** > **Definição de fila remota**.
   b) Dê à definição o nome `MyClientRemoteQueue` > **Avançar**.
   c) Digite `MQTTExampleQueue` no campo **Fila Remota**.
   d) Digite `MyClient` no campo **Gerenciador de Filas Remotas**.
   e) Digite `SYSTEM.MQTT.TRANSMIT.QUEUE` no campo **Fila de transmissão** > **Concluir**.
4. Coloque uma mensagem de teste no `MyClientRemoteQueue`.
   a) Clique com o botão direito em **MyClientRemoteQueue** > **Colocar mensagem de teste...**
   b) Digite `Hello queue!` no campo de dados Mensagem> **Colocar mensagem** > **Fechar**

   O **Histórico do Cliente** registra um evento do `Received`
5. Remova `SYSTEM.MQTT.TRANSMIT.QUEUE` como a fila de transmissão padrão.
   a) Clique com o botão direito em `Queue Managers\`*`QmgrName`* `folder` > **Propriedades...**
   b) Clique em **Comunicação** no navegador.
   c) Remova `SYSTEM.MQTT.TRANSMIT.QUEUE` do campo **Fila de transmissão padrão** > **OK**.
6. Reexecute a etapa “4” na página 130.

   `MyClientRemoteQueue` é uma definição de fila remota que nomeia explicitamente a fila de transmissão. Não é necessário definir uma fila de transmissão padrão para enviar uma mensagem para o `MyClient`.

Como proceder a seguir

Com a fila de transmissão padrão não mais configurada como `SYSTEM.MQTT.TRANSMIT.QUEUE`, o MQTT Client Utility não consegue criar uma nova assinatura, a menos que um alias do gerenciador de filas seja definido para `ClientIdentifier`, `MyClient`. Restaure a fila de transmissão padrão para `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Windows Linux AIX Publicando uma mensagem para um cliente MQTT v3

Publique uma mensagem de um cliente MQTT v3 para outro usando `ClientIdentifier` como o nome do tópico e o IBM MQ como o broker de publicação/assinatura.

Antes de começar

Execute a tarefa "Publicando uma mensagem para o utilitário do cliente MQTT a partir do IBM MQ Explorer" na página 125. Deixe o utilitário do cliente de MQTT conectado.

Sobre esta tarefa

A tarefa demonstra duas coisas:

1. Assinar um tópico em um cliente MQTT e receber uma publicação de outro cliente MQTT .
2. Configurar assinaturas "ponto a ponto" usando `ClientIdentifier` como a sequência de tópicos.

Procedimento

1. Descarte quaisquer assinaturas existentes desconectando e reconectando o utilitário do cliente MQTT.

   A assinatura é descartada porque, a menos que você altere os padrões, o utilitário do cliente MQTT se conecta a uma sessão limpa; consulte Sessões limpas.

   Para tornar mais fácil a execução da tarefa, digite seu próprio `ClientIdentifier`, em vez de usar o `ClientIdentifier` gerado criado pelo utilitário cliente de MQTT.

   a) Clique em **Desconectar** para desconectar o utilitário do cliente MQTT do canal de telemetria.

      O **Histórico do Cliente** registra um evento `Disconnected`

   b) Altere **Identificador de Cliente** para `MyClient`.

   c) Clique em **Conectar**.

   O **Histórico do Cliente** registra um evento `Connected`
2. Crie uma assinatura para o tópico, `MyClient`

   `MyClient` é `ClientIdentifier` desse cliente.

   a) Digite `MyClient` no campo **Assinatura\Tópico** > **Assinar**.

   O **Histórico do Cliente** registra um evento do `Subscribed`
3. Inicie outro utilitário do cliente de MQTT.

   a) Abra a pasta `Queue Managers\`*`QmgrName`*`\Telemetry\channels`.

   b) Clique com o botão direito no canal **Texto sem formatação** > **Executar utilitário do cliente MQTT...**

   c) Clique em **Conectar**.

   O **Histórico do Cliente** registra um evento `Connected`
4. Publique `Hello MyClient!` para o tópico `MyClient`

   a) Copie o tópico da assinatura, `MyClient`, a partir do utilitário do cliente do MQTT em execução com o `ClientIdentifier`, `MyClient`.

   b) Cole `MyClient` no campo **Publicação\Tópico** de cada uma das instâncias do utilitário do cliente do MQTT.

   c) Digite `Hello MyClient!` no campo **Publicação\mensagem**.

   d) Clique em **Publicar** em ambas as instâncias.

Resultados

O **Histórico do cliente** no utilitário do cliente do MQTT com o `ClientIdentifier`, `MyClient`, registra dois eventos em **Recebidos** e um evento em **Publicados**. A outra instância do utilitário do cliente MQTT registra um evento **Publicado**.

Se você vir apenas um evento **Recebido**, verifique as causas possíveis a seguir:

1. A fila de transmissão padrão para o gerenciador de filas está configurada como `SYSTEM.MQTT.TRANSMIT.QUEUE`?

2. Você criou aliases de gerenciador de filas ou de definições de filas remotas referenciando `MyClient` ao fazer os outros exercícios? Caso você tenha um problema de configuração, exclua quaisquer recursos que referenciam `MyClient`, como aliases de gerenciador de filas ou filas de transmissão. Desconecte os utilitários do cliente, pare e reinicie o serviço de telemetria (MQXR).

Windows Linux AIX Enviando uma mensagem para um aplicativo IBM MQ por meio de um cliente do MQTT

Um aplicativo IBM MQ pode receber uma mensagem de um cliente do MQTT v3 assinando um tópico. O cliente do MQTT se conecta ao IBM MQ usando um canal de telemetria e envia uma mensagem para o aplicativo IBM MQ publicando no mesmo tópico.

Execute a tarefa, "Publicando uma mensagem para IBM MQ a partir de um cliente MQTT" na página 132, para saber como enviar uma publicação de um cliente MQTT para uma assinatura definida no IBM MQ.

Se o tópico for armazenado em cluster ou distribuído usando uma hierarquia de publicação/assinatura, a assinatura poderá estar em um gerenciador de filas diferente para o gerenciador de filas ao qual o cliente MQTT está conectado.

Windows Linux AIX Publicando uma mensagem para IBM MQ a partir de um cliente MQTT

Crie uma assinatura para um tópico usando o IBM MQ Explorer e publique no tópico usando o utilitário do cliente MQTT.

Antes de começar

Execute a tarefa "Publicando uma mensagem para o utilitário do cliente MQTT a partir do IBM MQ Explorer" na página 125. Deixe o utilitário do cliente de MQTT conectado.

Sobre esta tarefa

A tarefa demonstra a publicação de uma mensagem com um cliente MQTT e o recebimento da publicação usando uma assinatura durável não gerenciada criada usando o IBM MQ Explorer.

Procedimento

1. Crie uma assinatura durável para a sequência de tópicos `MQTT Example`.

   Execute as etapas a seguir para criar a fila e a assinatura usando o IBM MQ Explorer.

   a) Clique com o botão direito na pasta `Queue Managers\`*`QmgrName`*`\Queues` em IBM MQ Explorer > **Novo** > **Fila local...**.

   b) Digite `MQTTExampleQueue` como o nome da fila > **Concluir**.

   c) Clique com o botão direito na pasta `Queue Managers\`*`QmgrName`*`\Subscriptions` em IBM MQ Explorer > **Novo** > **Assinatura...**.

   d) Digite `MQTTExampleSubscription` como o nome da fila > **Avançar**.

   e) Clique em **Selecionar...** > `MQTTExampleTopic` > **OK**.

   Você já criou o tópico, `MQTTExampleTopic` na etapa "4" na página 126 de "Publicando uma mensagem para o utilitário do cliente MQTT a partir do IBM MQ Explorer" na página 125.

   f) Digite `MQTTExampleQueue` como o nome do destino > **Concluir**.

2. Como uma etapa opcional, configure a fila para uso por um usuário diferente, sem autoridade do `mqm`.

   Se você estiver definindo a configuração para usuários com menos autoridade do que `mqm`, será necessário fornecer ao `put` e `get` autoridade para o `MQTTExampleQueue`. O acesso ao tópico e à fila de transmissão foi configurado em "Publicando uma mensagem para o utilitário do cliente MQTT a partir do IBM MQ Explorer" na página 125.

   a) Autorize um usuário a colocar e chegar à fila `MQTTExampleQueue`:

```
setmqaut -m qMgrName -t queue -n MQTTExampleQueue -p User ID -all +put +get
```

3. Publique `Hello IBM MQ!` no tópico `MQTT Example` usando o utilitário cliente MQTT .

   Se você não deixou o utilitário do cliente do MQTT conectado, clique com o botão direito no canal **PlainText** > **Executar o utilitário do cliente do MQTT...** > **Conectar**.

   a) Digite `MQTT Example` no campo **Publicação\Tópico**.

   b) Digite `Hello IBM MQ!` no campo **Publicação\Mensagem** > **Publicar**.
4. Abra a pasta `Queue Managers\`*`QmgrName`*`\Queues` e localize `MQTTExampleQueue`.

   O campo **Profundidade da fila atual** é 1
5. Clique com o botão direito em `MQTTExampleQueue` > **Procurar mensagens ...** e examinar a publicação.

Windows Linux AIX Aplicativos de publicação/assinatura do MQTT

Use publicação/assinatura baseada em tópico para escrever aplicativos MQTT .

Quando o cliente MQTT está conectado, publicações fluem em qualquer direção entre o cliente e o servidor. As publicações são enviadas do cliente quando informações são publicadas no cliente. As publicações são recebidas no cliente quando uma mensagem é publicada para um tópico que corresponde a uma assinatura criada pelo cliente.

O broker de publicação/assinatura do IBM MQ gerencia os tópicos e as assinaturas criados pelos clientes do MQTT. Os tópicos criados pelos clientes MQTT compartilham o mesmo espaço de tópico que os tópicos criados pelos aplicativos do IBM MQ.

Publicações que correspondem à sequência de tópicos em uma assinatura do cliente MQTT são colocadas em `SYSTEM.MQTT.TRANSMIT.QUEUE` com o nome do gerenciador de filas remotas configurado para o `ClientIdentifier` do cliente. O serviço de telemetria (MQXR) encaminha as publicações para o cliente que criou a assinatura. Ele usa `ClientIdentifier`, que foi configurado como o nome do gerenciador de filas remotas para identificar o cliente.

Normalmente, `SYSTEM.MQTT.TRANSMIT.QUEUE` deve ser definida como a fila de transmissão padrão. É possível, mas oneroso, configurar o MQTT para não usar a fila de transmissão padrão; consulte Configurar enfileiramento distribuído para enviar mensagens para clientes MQTT.

Um cliente MQTT pode criar uma sessão persistente; consulte “Sessões stateless e stateful do MQTT” na página 136. Assinaturas criadas em uma sessão persistente são duráveis. Publicações que chegam para um cliente com uma sessão persistente são armazenadas na `SYSTEM.MQTT.TRANSMIT.QUEUE` e encaminhadas para o cliente quando ele se reconecta.

Um cliente MQTT também pode publicar e assinar as publicações retidas; consulte Publicações retidas e clientes MQTT. Um assinante para um tópico de publicação retida recebe a publicação mais recente para o tópico. O assinante recebe a publicação retida quando cria uma assinatura ou quando se reconecta à sua sessão anterior.

Windows Linux AIX Aplicativos de Telemetria

Gravar os aplicativos de telemetria usando fluxos de mensagens do IBM MQ ou do IBM Integration Bus.

Use o JMS, MQI ou outras interfaces de programação do IBM MQ para programar aplicativos de telemetria no IBM MQ.

O serviço de telemetria (MQXR) converte entre mensagens do MQTT v3 e mensagens do IBM MQ. Ele cria assinaturas e publicações em nome de clientes MQTT e encaminha as publicações para clientes do MQTT. Uma publicação é a carga útil de uma mensagem do MQTT v3. A carga útil abrange cabeçalhos de mensagem e uma matriz de bytes em formato `jms-bytes`. O servidor de telemetria mapeia os cabeçalhos entre uma mensagem do MQTT v3 e uma mensagem do IBM MQ; consulte “Integração do MQ Telemetry com gerenciadores de filas” na página 134.

Utilize os nós Publication, MQInput e JMSInput para enviar e receber publicações entre os clientes IBM Integration Bus e MQTT.

Usando fluxos de mensagens, é possível integrar a telemetria com websites usando HTTP e com outros aplicativos usando IBM MQ e WebSphere Adaptadores

Windows Linux AIX Integração do MQ Telemetry com gerenciadores de filas

O cliente MQTT está integrado com o IBM MQ como um aplicativo de publicação/assinatura. Ele pode publicar ou se subscrever para tópicos no IBM MQ, criando novos tópicos ou usando tópicos existentes. Ele recebe publicações do IBM MQ como um resultado dos clientes MQTT, incluindo a si mesmo, ou outros aplicativos de publicação do IBM MQ nos tópicos de suas assinaturas. Regras são aplicadas para decidir os atributos de uma publicação.

Muitos dos atributos associados a tópicos, publicações, assinaturas e mensagens que são fornecidos pelo IBM MQ, não são suportados. “Cliente MQTT para broker de publicação/assinatura do IBM MQ” na página 134 e “IBM MQ para um cliente MQTT” na página 135 descrevem como atributos de publicações são configurados. As configurações dependem de se a publicação está indo para um broker de publicação/assinatura do IBM MQ, ou a partir dele.

No IBM MQ, os tópicos de publicação/assinatura são associados aos objetos do tópico administrativo. Os tópicos criados pelos clientes MQTT não são diferentes. Quando um cliente MQTT cria uma sequência de tópicos para uma publicação no IBM MQ, o broker de publicação/assinatura o associa a um objeto do tópico administrativo. O broker mapeia a sequência de tópicos na publicação para o pai do objeto do tópico administrativo mais próximo. O mapeamento é o mesmo para aplicativos IBM MQ. Se não houver um tópico criado por usuário, o tópico da publicação será mapeado para `SYSTEM.BASE.TOPIC`. Os atributos que são aplicados à publicação são derivados do objeto do tópico.

Quando um aplicativo IBM MQ ou um administrador cria uma assinatura, a assinatura é nomeada. Listar assinaturas usandoIBM MQ Explorer , ou usando**runmqsc** ou comandos PCF. Todas as assinaturas do cliente MQTT são nomeadas. Eles recebem um nome no formato:*`ClientIdentifier:Topic name`*

Cliente MQTT para broker de publicação/assinatura do IBM MQ

Um cliente MQTT enviou uma publicação para o IBM MQ. O serviço de telemetria (MQXR) converte a publicação em uma mensagem do IBM MQ. A mensagem do IBM MQ contém três partes:

1. MQMD
2. RFH2
3. Mensagem

As propriedades do MQMD são configuradas para seus valores padrão, exceto onde estiver anotado na Tabela 9 na página 134.

Tabela 9. Configurações para propriedades do MQMD

Campo MQMD	**tipo**	**Value**
Format	MQCHAR8	MQFMT_RF_HEADER_2

UserIdentifier	MQCHAR12	Configure para uma das opções a seguir:
		`MqttClient.ClientIdentifier`
		`MqttConnectOptions.UserName`
		Um ID do usuário configurado pelo administrador do IBM MQ para o canal de telemetria.

Priority	MQLONG	MQPRI_PRIORITY_AS_Q_DEF (Padrão para o IBM MQ, que é diferente para JMS cujo padrão é 4.)

Tabela 9. Configurações para propriedades do MQMD (continuação)		
Campo MQMD	**tipo**	**Value**
Persistence	MQLONG	QoS=0→MQPER_NOT_PERSISTENT QoS=1→MQPER_PERSISTENT QoS=2→MQPER_PERSISTENT

O cabeçalho RFH2 não contém uma pasta <msd> para definir o tipo da mensagem do JMS. O serviço de telemetria (MQXR) cria a mensagem do IBM MQ como uma mensagem padrão do JMS. O tipo de mensagem padrão do JMS é uma mensagem `jms-bytes`. Um aplicativo pode acessar informações do cabeçalho adicionais como propriedades de mensagem; consulte Propriedades de Mensagem.

Os valores de RFH2 são configurados conforme mostrado na Tabela 10 na página 135. A propriedade Format é configurada no cabeçalho fixo RFH2 e os outros valores são configurados nas pastas do RFH2.

Tabela 10. Configurações para propriedades RFH2		
Propriedade RFH2	**Tipo/Pasta**	**Cabeçalho**
Format	MQCHAR8	MQFMT_NONE
ClientIdentifier	mqtt/clientId	Cópia de MqttClient.ClientIdentifier com um comprimento de 1...23 bytes.
QoS	mqtt/qos	Cópia de QoS da mensagem recebida do MQTT.
ID de mensagem	mqtt/msgid	Cópia de ID de mensagem da mensagem recebida do MQTT, se QoS for 1 ou 2.
MQIsRetained	mqps/Ret	Configurado se a publicação original do MQTT foi enviada com a propriedade RETAIN configurada e a mensagem for recebida como uma publicação retida.
MQTopicString	mqps/Top	O tópico para o qual a mensagem do MQTT foi publicada.

A carga útil em uma publicação do MQTT é mapeada para o conteúdo de uma mensagem do IBM MQ:

Tabela 11. Como a carga útil em uma publicação do MQTT é mapeada para os conteúdos da mensagem do IBM MQ		
Conteúdo da Mensagem	**tipo**	**Conteúdos da mensagem do IBM MQ**
Buffer	MQBYTE *n*	Cópia de bytes da mensagem recebida do MQTT. O comprimento pode ser zero.

IBM MQ para um cliente MQTT

Um cliente foi inscrito para o tópico de uma publicação. Um aplicativo IBM MQ publicou no tópico, resultando em uma publicação sendo enviada para o assinante de MQTT pelo broker de publicação/assinatura do IBM MQ. Como alternativa, um aplicativo IBM MQ enviou uma mensagem não solicitada diretamente para um cliente MQTT. Tabela 12 na página 136 descreve como cabeçalhos de mensagem fixos são configurados na mensagem que é enviada para o cliente MQTT. Quaisquer outros dados no cabeçalho de mensagem do IBM MQ ou quaisquer outros cabeçalhos são descartados. Os dados da mensagem na mensagem do IBM MQ são enviados como carga útil da mensagem na mensagem do MQTT, sem nenhuma mudança. A mensagem do MQTT é enviada para o cliente MQTT pelo serviço de telemetria (MQXR).

Tabela 12. Como cabeçalhos de mensagem fixos são configurados em uma mensagem do IBM MQ enviada para o cliente MQTT		
campo MQTT	**tipo**	**Value**
DUP	Booleano	Configure se QoS = 1 ou 2 e se a mensagem foi enviada para esse cliente em uma transmissão anterior e não tiver sido reconhecida após um período de tempo.
QoS	int	A maneira como o valor de QoS em uma publicação de saída do broker de publicação/assinatura no IBM MQ é configurado depende da publicação recebida. Ela depende de se a publicação recebida foi enviada a partir de um cliente MQTT ou a partir de um aplicativo IBM MQ. **MQTT** Diminua o valor de QoS na publicação recebida e da QoS solicitada pelo assinante. **IBM MQ** Diminua o valor da QoS derivada da publicação recebida: MQPER_NOT_PERSISTENT→QoS=0 MQPER_PERSISTENT→QoS=2 e QoS solicitada pelo assinante. Se a mensagem for enviada para o cliente sem uma assinatura, a QoS será configurada como 2 por padrão. Um cliente pode alterar esse valor subscrevendo-se para DEFAULT.QoS com uma QoS diferente.
RETAIN	Booleano	Configure se a publicação recebida tiver a propriedade retida configurada.

Tabela 13 na página 136 descreve como cabeçalhos de mensagem variáveis são configurados na mensagem do MQTT que é enviada para o cliente MQTT.

Tabela 13. Como as propriedades do cabeçalho de variável do MQTT são configuradas em uma mensagem do MQTT enviada para o cliente do MQTT		
campo MQTT	**tipo**	**Value**
Topic name	Sequência	A sequência de tópicos com a qual a mensagem foi publicada.
Message ID	Sequência	Os últimos 2 bytes da propriedade MQMD.MsgId da publicação quando ela é colocada na SYSTEM.MQTT.TRANSMIT.QUEUE.
Payload	byte[]	Cópia direta dos bytes da publicação recebida para o broker de publicação/assinatura. O comprimento pode ser zero.

Windows Linux AIX Sessões stateless e stateful do MQTT

Os clientes MQTT podem criar uma sessão stateful com o gerenciador de filas. Quando um cliente MQTT stateful se desconecta, o gerenciador de filas mantém as assinaturas criadas pelo cliente e as mensagens em andamento. Quando o cliente se reconecta, ele resolve a mensagem em andamento. Ele envia quaisquer mensagens enfileiradas para entrega e recebe quaisquer mensagens publicadas para suas assinaturas enquanto ele estava desconectado.

Quando um cliente MQTT se conecta a um canal de telemetria ele inicia uma nova sessão ou retoma uma sessão antiga. Uma sessão nova não tem mensagens pendentes que não foram reconhecidas, nem

assinaturas e nem publicações aguardando entrega. Quando um cliente se conecta, ele especifica se vai começar com uma sessão limpa ou se vai retomar uma sessão existente; consulte Sessões limpas.

Se o cliente retomar uma sessão existente, ele continuará como se a conexão não tivesse sido interrompida. Publicações aguardando entrega são enviadas para o cliente e todas as transferências de mensagens não confirmadas são concluídas. Quando um cliente em uma sessão persistente se desconecta do serviço de telemetria (MQXR), todas as assinaturas criadas por ele permanecem. Publicações para as assinaturas são enviadas para o cliente quando ele se reconecta. Se ele se reconectar sem retomar a sessão antiga, as publicações serão descartadas pelo serviço de telemetria (MQXR).

Informações do estado da sessão são salvas pelo gerenciador de filas na fila `SYSTEM.MQTT.PERSISTENT.STATE`.

O administrador do IBM MQ pode se desconectar e limpar uma sessão.

Windows Linux AIX Quando um MQTT do cliente não está conectado

Quando um cliente não está conectado, o gerenciador de filas pode continuar recebendo publicações em seu nome. Elas são encaminhadas para o cliente quando ele se reconecta. Um cliente pode criar um "última informação e testamento", que o gerenciador de filas publica em nome do cliente caso o cliente se desconecte inesperadamente.

Se você quiser ser notificado quando um cliente se desconectar inesperadamente, é possível registrar um última publicação de testamento; consulte Última publicação de testamento. Ela será enviada pelo serviço de telemetria (MQXR) se detectar que a conexão com o cliente foi interrompida sem a solicitação do cliente.

Um cliente pode publicar uma publicação retida a qualquer momento; consulte Publicações retidas e clientes MQTT. Uma nova assinatura para um tópico pode solicitar que seja enviada para qualquer publicação retida associada ao tópico. Se você criar a última informação e testamento como uma publicação retida, será possível usá-la para monitorar o status de um cliente.

Por exemplo, o cliente publica uma publicação retida quando se conecta informando sua disponibilidade. Ao mesmo tempo, ele cria uma publicação última informação e testamento retida que anuncia sua indisponibilidade. Além disso, antes de fazer uma desconexão planejada, ele publica sua indisponibilidade como uma publicação retida. Para saber se o cliente está disponível, você teria que se subscrever para o tópico da publicação retida. Você sempre receberia uma das três publicações.

Se o cliente tiver que receber mensagens publicadas quando estiver desconectado, reconecte o cliente à sessão anterior; consulte “Sessões stateless e stateful do MQTT” na página 136. Suas assinaturas ficarão ativas até serem excluídas ou até o cliente criar uma sessão limpa.

Windows Linux AIX Loose coupling entre clientes MQTT e aplicativos IBM MQ

O fluxo de publicações entre clientes MQTT e aplicativos IBM MQ são fracamente acoplados. Publicações podem se originar de um cliente MQTT ou aplicativo IBM MQ, sem uma ordem configurada. Os publicadores e assinantes são fracamente acoplados. Eles interagem uns com os outros indiretamente por meio de publicações e assinaturas. Você também pode enviar mensagens diretamente para um cliente MQTT a partir de um aplicativo IBM MQ.

Os clientes MQTT e aplicativos IBM MQ são fracamente acoplados em dois sentidos:

1. Publicadores e assinantes são fracamente acoplados pela associação de uma publicação e de uma assinatura com um tópico. Publicadores e assinantes normalmente não têm conhecimento do endereço ou da identidade da outra origem de uma publicação ou assinatura.
2. Clientes MQTT publicam, se subscrevem, recebem publicações e processam confirmações de entrega em encadeamentos separados.

Um aplicativos cliente MQTT não espera até uma publicação ser entregue. O aplicativo passa uma mensagem para o cliente MQTT e depois o aplicativo continua em seu próprio encadeamento. Um token

de entrega é usado para sincronizar o aplicativo com a entrega de uma publicação; consulte Tokens de entrega.

Após passar uma mensagem para o cliente MQTT, o aplicativo tem a opção de esperar o token de entrega. Em vez de esperar, o cliente pode fornecer um método de retorno de chamada que é chamado quando a publicação é entregue para o IBM MQ. Ele também pode ignorar o token de entrega.

Dependendo da qualidade de serviço associada à mensagem, o token de entrega é retornado imediatamente para o método de retorno de chamada ou provavelmente após um tempo considerável. O token de entrega pode até ser retornado após o cliente se desconectar e se reconectar. Se a qualidade de serviço for *fire-forget*, o token de entrega será retornado imediatamente. Nos outros dois casos, o token de entrega será retornado apenas quando o cliente receber reconhecimento de que a publicação foi enviada para os assinantes.

Publicações enviadas para um cliente MQTT como resultado de uma assinatura do cliente são entregues para o método de retorno de chamada `messageArrived`. `messageArrived` é executado em um encadeamento diferente para o aplicativo principal.

Enviando mensagens diretamente para um cliente MQTT

É possível enviar uma mensagem para um determinado cliente MQTT de uma de duas maneiras.

1. Um aplicativo IBM MQ pode enviar uma mensagem diretamente para um cliente MQTT sem uma assinatura; consulte Enviando uma mensagem para um cliente diretamente.
2. Uma abordagem alternativa é usar a convenção de nomenclatura `ClientIdentifier`. Faça todos os assinantes do MQTT criarem assinaturas usando seus `ClientIdentifier` exclusivos como um tópico. Publique para o *ClientIdentifier*. A publicação é enviada para o cliente inscrito para o tópico *ClientIdentifier*. Usando essa técnica, é possível enviar uma publicação para um determinado assinante do MQTT.

Windows Linux AIX MQ Telemetrysegurança

A proteção de dispositivos de telemetria pode ser importante, já que os dispositivos têm a probabilidade de serem portáteis e usados em locais que não podem ser adequadamente controlados. É possível usar VPN para proteger a conexão do dispositivo MQTT para o serviço de telemetria (MQXR). O MQ Telemetry fornece dois outros mecanismos de segurança, TLS e JAAS.

O TLS é usado principalmente para criptografar comunicações entre o dispositivo e o canal de telemetria e para autenticar se o dispositivo está se conectando ao servidor correto; veja Autenticação de canal de telemetria usando TLS. Também é possível usar o TLS para verificar se o dispositivo do cliente tem permissão para se conectar ao servidor; consulte Autenticação do cliente MQTT usando TLS.

O JAAS é usado principalmente para verificar se o usuário do dispositivo tem permissão para usar um aplicativo do servidor; consulte Autenticação do cliente MQTT usando uma senha. O JAAS pode ser usado com LDAP para verificar uma senha usando um diretório de conexão única.

O TLS e o JAAS podem ser usados juntos para fornecer autenticação de dois fatores. É possível restringir as cifras usadas por TLS às cifras que atendem às normas do FIPS.

Com pelo menos dezenas de milhares de usuários, nem sempre é prático fornecer perfis de segurança individuais. E também nem sempre é prático usar os perfis para autorizar usuários individuais a acessarem objetos do IBM MQ. Em vez disso, agrupe usuários em classes para autorizar publicação e assinatura para tópicos e envio de publicações para clientes.

Configure cada canal de telemetria para mapear clientes para IDs de usuário cliente comum. Use um ID de usuário comum para cada cliente que se conecta em um canal específico; consulte Identidade e autorização do cliente MQTT.

A autorização de grupos de usuários não compromete a autenticação de cada indivíduo. Cada usuário individual pode ser autenticado, no cliente ou no servidor, com `Nome de Usuário` e Senha, e depois autorizado no servidor usando um ID de usuário comum.

Windows Linux AIX Globalização do MQ Telemetry

A carga útil da mensagem no protocolo MQTT v3 é codificada como matriz de bytes. Geralmente, o texto de manipulação de aplicativos cria a carga útil da mensagem em UTF-8. O canal de telemetria descreve a carga útil da mensagem como UTF-8, mas não faz nenhuma conversão de página de códigos. A sequência de tópicos de publicação deve ser UTF-8.

O aplicativo é responsável por converter dados alfabéticos em dados numéricos e de página de códigos corretos para a codificação de números correta.

O cliente MQTT Java tem um método `MqttMessage.toString` conveniente. O método trata a carga útil da mensagem como sendo codificada no conjunto de caracteres padrão da plataforma local, que geralmente é UTF-8. Ele converte a carga útil para uma sequência Java. Java tem um método de sequência, `getBytes`, que converte uma sequência em uma matriz de bytes codificada usando o conjunto de caracteres padrão da plataforma local. Dois programas MQTT Java trocando texto na carga útil da mensagem entre plataformas com o mesmo conjunto de caracteres padrão fazem isso de maneira fácil e eficiente em UTF-8.

Se o conjunto de caracteres padrão de uma das plataformas não for UTF-8, os aplicativos deverão estabelecer uma convenção para a troca de mensagens. Por exemplo, um publicador especifica conversão de uma sequência em UTF-8 usando o método `getBytes("UTF8")`. Para receber o texto de uma mensagem, o assinante assume que a mensagem esteja codificada no conjunto de caracteres UTF-8.

O serviço de telemetria (MQXR) descreve a codificação de todas as publicações recebidas de mensagens de clientes MQTT como sendo UTF-8. Ele configura o MQMD do `MQMD.CodedCharSetId` para UTF-8e `RFH2.CodedCharSetId` para `MQCCSI_INHERIT` ; Consulte “Integração do MQ Telemetry com gerenciadores de filas” na página 134. O formato da publicação é configurado como `MQFMT_NONE`, portanto, nenhuma conversão pode ser executada por canais ou por `MQGET`.

Windows Linux AIX Desempenho e escalabilidade do MQ Telemetry

Considere os seguintes fatores ao gerenciar grandes números de clientes e melhorar a escalabilidade do MQ Telemetry.

Planejamento de Capacidade

Para obter informações sobre relatórios de desempenho para o MQ Telemetry, consulte Documentos de desempenho do MQ.

Conexões

Os custos envolvidos em conexões incluem

- O custo da configuração de uma conexão em si em termos de tempo e uso de processador.
- Custos de rede.
- Memória usada para manter uma conexão aberta, mas sem ser usada.

Há uma carga extra incorrida quando clientes ficam conectados. Se uma conexão é mantida aberta, os fluxos do TCP/IP e mensagens do MQTT usam a rede para verificar se a conexão ainda está lá. Além disso, a memória é usada no servidor para cada conexão do cliente que é mantida aberta.

Se você estiver enviando mais de uma mensagem por minuto, mantenha sua conexão aberta para evitar o custo de iniciar uma nova conexão. Se você estiver enviando menos de uma mensagem a cada 10 - 15 minutos, considere descartar a conexão para evitar o custo de mantê-la aberta. Você talvez deseje manter uma conexão TLS aberta, mas inativa, por períodos mais longos porque é mais caro para configurar.

Além disso, considere o recurso do cliente. Se houver um recurso de armazenamento e encaminhamento no cliente, você pode armazenar mensagens em um lote e descartar a conexão entre o envio dos lotes. No

entanto, se o cliente estiver desconectado, ele não poderá receber uma mensagem do servidor. Portanto, o propósito de seu aplicativo tem influência na decisão.

Se seu sistema tiver um cliente enviando várias mensagens, por exemplo, transferências de arquivos, não espere uma resposta do servidor por mensagem. Em vez disso, envie todas as mensagens e verifique no final se todas elas foram recebidas. Alternativamente, use Qualidade de Serviço (QoS).

É possível variar o QoS por mensagem, entregando mensagens sem importância usando o QoS 0 e mensagens importantes usando um QoS de 2. O rendimento da mensagem pode ser aproximadamente duas vezes maior com um QoS de 0 do que com um QoS de 2.

Convenções de Nomenclatura

Se você estiver projetando seu aplicativo para muitos clientes, implemente uma convenção de nomenclatura efetiva. Para mapear cada cliente para o `ClientIdentifier` correto, faça com que o `ClientIdentifier` seja significativo. Uma boa convenção de nomenclatura facilita para o Administrador a descoberta de quais clientes estão em execução. Uma convenção de nomenclatura ajuda o administrador a filtrar uma longa lista de clientes no IBM MQ Explorer e ajuda na determinação de problemas; consulte Identificador de cliente.

Rendimento

O comprimento dos nomes de tópicos afeta o número de bytes que fluem na rede. Durante uma publicação ou assinatura, o número de bytes em uma mensagem pode ser importante. Portanto, limite o número de caracteres no nome de um tópico. Quando um cliente MQTT se subscreve para um tópico do IBM MQ dá a ele um nome no formato:

```
ClientIdentifier: TopicName
```

Para visualizar todas as assinaturas para um cliente de MQTT, é possível usar o comando IBM MQ MQSC **DISPLAY**:

```
DISPLAY SUB(' ClientID1:*')
```

Definindo recursos no IBM MQ para ser utilizado por clientes do MQTT

Um cliente do MQTT se conecta a um gerenciador de filas remotas do IBM MQ. Há dois métodos básicos para um aplicativo IBM MQ enviar mensagens para um cliente MQTT : configurar a fila de transmissão padrão para `SYSTEM.MQTT.TRANSMIT.QUEUE` ou usar aliases do gerenciador de filas. Defina a fila de transmissão padrão de um gerenciador de filas, caso haja um grande número de clientes do MQTT. O uso da configuração de fila de transmissão padrão simplifica o esforço de administração; consulte Configurar o enfileiramento distribuído para enviar mensagens para clientes MQTT.

Melhorando a Escalabilidade Evitando Assinaturas

Quando um cliente do MQTT V3 assina um tópico, uma assinatura é criada pelo serviço de telemetria (MQXR) no IBM MQ. A assinatura roteia publicações para o cliente no `SYSTEM.MQTT.TRANSMIT.QUEUE`. O nome do gerenciador de filas remotas no cabeçalho de transmissão de cada publicação é configurado para `ClientIdentifier` do cliente do MQTT que fez a assinatura. Se houver vários clientes, cada um fazendo suas próprias assinaturas, isso resultará em várias assinaturas de proxy que estão sendo mantidas por todo o cluster ou hierarquia de publicação/assinatura do IBM MQ. Para obter informações sobre não usar publicação/assinatura, mas usar uma solução baseada em ponto a ponto, consulte Enviando uma mensagem para um cliente diretamente.

Gerenciando Grandes Números de Clientes

Para suportar diversos clientes conectados simultaneamente, aumente a memória disponível para o serviço de telemetria (MQXR) configurando os parâmetros de JVM **-Xms** e **-Xmx**. Siga estas etapas:

1. Encontre o arquivo `java.properties` no diretório de configuração do serviço de telemetria; consulte Diretório de configuração de serviço de Telemetria (MQXR) no Windows ou Diretório de configuração do serviço de Telemetria no Linux.
2. Siga as direções no arquivo; um heap de 1 GB é suficiente para 50.000 clientes conectados simultaneamente.

```
# Heap sizing options - uncomment the following lines to set the heap to 1G
#-Xmx1024m
#-Xms1024m
```

3. Inclua outros argumentos da linha de comandos a serem transmitidos para a JVM executando o serviço de telemetria (MQXR) no arquivo `java.properties`; consulte Transmitindo parâmetros da JVM para o serviço de telemetria (MQXR).

Para aumentar o número de descritores de arquivos abertos no Linux, inclua as linhas a seguir no `/etc/security/limits.conf/` e efetue login novamente.

```
@mqm soft nofile 65000
@mqm hard nofile 65000
```

Cada soquete requer um descritor de arquivo. O serviço de telemetria requer alguns descritores de arquivo adicionais, portanto, esse número deve ser maior que o número de soquetes abertos necessário.

O gerenciador de filas usa manipulação de objetos para cada assinatura não durável. Para suportar diversas assinaturas não duráveis ativas, aumente o número máximo de manipulações ativas no gerenciador de filas; por exemplo:

```
echo ALTER QMGR MAXHANDS(999999999) | runmqsc qMgrName
```

Figura 48. Alterar número máximo de manipulações no Windows

```
echo "ALTER QMGR MAXHANDS(999999999)" | runmqsc qMgrName
```

Figura 49. Alterar número máximo de manipulações no Linux

Outras contraprestações

Ao planejar os requisitos do sistema, considere a quantidade de tempo levada para reiniciar o sistema. O tempo de inatividade planejado pode ter implicações quanto ao número de mensagens que estão enfileiradas aguardando para serem processadas. Configure o sistema para que as mensagens possam ser processadas com sucesso em um tempo aceitável. Revise o armazenamento em disco, a memória e a energia de processamento. Com alguns aplicativos clientes, talvez seja possível descartar mensagens quando o cliente se reconectar. Para descartar mensagens, configure `CleanSession` nos parâmetros de conexão do cliente; consulte Sessões limpas. Alternativamente, publique e assine usando o melhor esforço de Qualidade de serviço `0` em um cliente do MQTT; consulte Qualidade de serviço. Use mensagens `não persistentes` ao enviar mensagens a partir do IBM MQ. Mensagens com essas qualidades de serviço não são recuperadas quando o sistema ou a conexão são reiniciados.

Windows Linux AIX Dispositivos suportados pelo MQ Telemetry

Os clientes do MQTT podem ser executados em vários dispositivos, de sensores e atuadores, para identificar sistemas de veículos e dispositivos em espera.

Os clientes do MQTT são pequenos e executados em dispositivos restritos por pouca memória e baixa energia de processamento. O MQTT protocol é confiável e tem cabeçalhos pequenos, o que é adequado para redes restritas por baixa largura de banda, alto custo e disponibilidade intermitente.

O MQ Telemetry se comunica com dispositivos de telemetria por meio de aplicativos clientes do MQTT. Esses aplicativos usam os seguintes recursos, todos os quais implementam o protocolo MQTT v3:

- As seguintes bibliotecas de clientes:
  - O *MQTT client for Java*, que é usado para construir aplicativos nativos para (por exemplo) dispositivos Android, OS X, Linux ou Windows. Aplicativos que usam esta biblioteca de cliente podem ser executados em todas as variações de Java desde o menor CLDC (Connected Limited Device Configuration)/MIDP (Mobile Information Device Profile) até o CDC (Connected Device Configuration) /Foundation, J2SE (Java Platform, Standard Edition) e J2EE (Java Platform, Enterprise Edition). A biblioteca de classes customizada do IBM também é suportada. A plataforma Java ME geralmente é usada em dispositivos pequenos, como atuadores, sensores, telefones celulares e outros dispositivos integrados. A plataforma Java SE geralmente é instalada em dispositivos integrados, como computadores desktop e servidores.
  - O *MQTT client for C*, que é usado para construção de aplicativos nativos para dispositivos iOS, OS X, Linux ou Windows (por exemplo). Esta biblioteca cliente fornece uma implementação de referência C juntamente com o cliente nativo pré-construído para os sistemas Windows e Linux. A implementação de referência C permite que o MQTT seja portado para uma ampla faixa de dispositivos e plataformas. Alguns sistemas Windows no Intel, incluindo Windows 7, RedHat, Ubuntu, e alguns sistemas Linux em plataformas ARM, como Eurotech Viper, implementam versões do Linux que executam o cliente C, mas a IBM não fornece suporte de serviço para as plataformas. Você deverá reproduzir os problemas com o cliente em uma plataforma suportada caso pretenda chamar seu centro de suporte IBM .
  - O *MQTT client for Java*, que é usado para construir aplicativos da web baseados em navegador.

As bibliotecas do cliente do MQTT estão disponíveis livremente por meio do Eclipse Paho e do MQTT.org. Consulte Programas de amostra do IBM MQ Telemetry Transport.

Segurança no IBM MQ

No IBM MQ, há vários métodos de fornecimento de segurança: a interface de serviço de autorização; saídas do canal gravadas pelo usuário ou terceiros; segurança do canal usando Segurança da Camada de Transporte (TLS), registros de autenticação de canal e segurança de mensagem.

Interface de serviço de autorização

A autorização para usar chamadas MQI, comandos e acesso aos objetos para objetos é fornecida pelo **Object Authority Manager** (OAM), que está ativado, por padrão. O acesso às entidades IBM MQ é controlado por meio de grupos de usuários do IBM MQ e do OAM. Os administradores podem usar uma interface de linha de comandos para conceder ou revogar as autorizações, conforme necessário.

Para obter mais informações sobre a criação de componentes de serviço de autorização, consulte Configurando segurança em sistemas AIX, Linux, and Windows.

Saídas de Canal Gravadas pelo Usuário ou de Terceiros

Os canais podem usar saídas de canal gravadas pelo usuário ou de terceiros. Para obter informações adicionais, consulte Programas de saída do canal para canais de sistemas de mensagens.

Segurança de canal usando TLS

O protocolo Segurança da Camada de Transporte (TLS) fornece segurança de canal de padrão de mercado, com proteção contra espionagem do tráfego de rede, violação e personificação.

O TLS usa técnicas simétricas e de chave pública para fornecer confidencialidade de mensagem e integridade e autenticação mútua.

Para uma revisão abrangente de segurança no IBM MQ incluindo informações detalhadas sobre o TLS, veja Segurança. Para obter uma visão geral de TLS, incluindo ponteiros para os comandos descritos nesta seção, veja Protocolos de segurança criptográfica: TLS.

Registros de Autenticação de Canal

Use os registros de autenticação de canal para exercer controle preciso sobre o acesso concedido para conectar-se aos sistemas em um nível de canal. Para obter mais informações, consulte Registros de Autenticação de Canal.

Segurança de Mensagem

Use o Advanced Message Security, que é um componente instalado e licenciado separadamente do IBM MQ, para fornecer proteção criptográfica às mensagens enviadas e recebidas usando o IBM MQ. Consulte o Advanced Message Security.

Tarefas relacionadas

Assegurando

Planejando para seus requisitos de segurança

Suporte de TLS do cliente gerenciado pelo IBM MQ.NET

O cliente totalmente gerenciado pelo IBM MQ.NET fornece suporte de Segurança da Camada de Transporte (TLS) que é baseado no kit SSLStreams do Microsoft.NET. Isso é diferente dos outros clientes do IBM MQ , que são baseados em IBM Global Security Kit (GSKit)

É possível desenvolver aplicativos IBM MQ.NET para execução no modo gerenciado ou no modo não gerenciado.

- No modo gerenciado, os aplicativos .NET trabalham dentro do .NET CLR (Common Language Runtime) sem qualquer chamada de plataforma cruzada, tal como chamar o C MQI.
- No modo não gerenciado, o C MQI é chamado para as operações MQI subjacentes. Basicamente, a interface de modo não gerenciado compõe as classes de agrupador do .NET sobre C MQI.

O cliente IBM MQ.NET gerenciado usa as bibliotecas do Microsoft.NET Framework para implementar protocolos de soquete seguro TLS. A classe System.NET.Security.SSLStream do Microsoft é usada para implementar a segurança (TLS) no IBM MQ.NET.

O modo cliente do IBM MQ.NET não gerenciado já suporta o recurso TLS, que é baseado em C MQI (e GSKit). Ou seja, as operações de TLS são manipuladas pelo C MQI. Nesse caso, o GSKit implementa os protocolos de soquete seguro do TLS

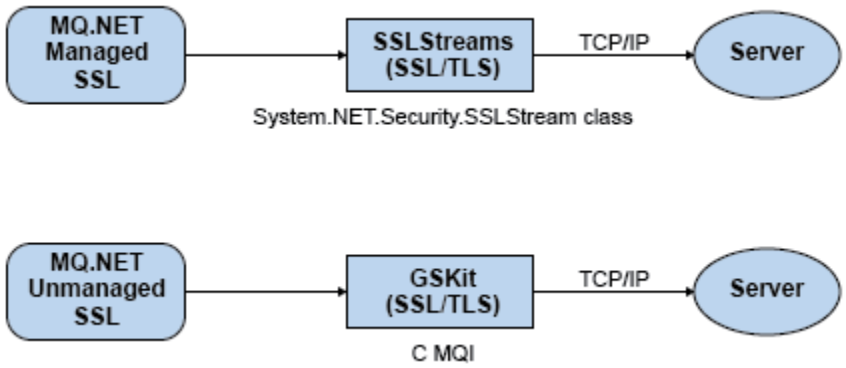


Figura 50. Comparação de TLS gerenciado e não gerenciado pelo IBM MQ.NET

A tabela a seguir resume as diferenças entre as implementações gerenciadas e não gerenciadas:

Tabela 14. Diferenças entre implementações gerenciadas e não gerenciadas			
Modo	**Protocolos**	**Implementação**	**Comments**
SSL gerenciado pelo IBM MQ.NET	TLS	Classe System.NET.Security.SSLStream A classe SSLStream opera como um fluxo sobre um soquete TCP conectado	TLS 1.0 TLS 1.2 (com Microsoft.NET Framework v4.5 apenas)
SSL não gerenciado pelo IBM MQ.NET	TLS	GSKit e C-MQI	Protocolos de soquete seguro TLS

Conceitos relacionados

Suporte de Secure Sockets Layer (SSL) e Segurança da Camada de Transporte (TLS) para .NET

IBM MQ MQI clients

Um IBM MQ MQI client é um componente do produto IBM MQ que pode ser instalado em um sistema no qual nenhum gerenciador de filas é executado.

Um IBM MQ MQI *cliente* é um componente que permite que um aplicativo em execução em um sistema emita chamadas MQI para um gerenciador de filas em execução em outro sistema A saída da chamada é enviada novamente ao cliente, que a transmite novamente ao aplicativo.

Usando um IBM MQ MQI client, um aplicativo em execução no mesmo sistema que o cliente pode se conectar a um gerenciador de filas que está em execução em outro sistema. O aplicativo pode emitir chamadas MQI para esse gerenciador de fila. Esse aplicativo é chamado de um aplicativo IBM MQ MQI client e o gerenciador de filas é chamado de *gerenciador de filas*.

Um IBM MQ *servidor* é um gerenciador de filas que fornece serviços de enfileiramento para um ou mais clientes Todos os objetos IBM MQ, por exemplo, filas, existem apenas na máquina do gerenciador de filas (a máquina servidor do IBM MQ) e não no cliente. Um servidor IBM MQ também pode oferecer suporte a aplicativos locais do IBM MQ.

A diferença entre um servidor IBM MQ e um gerenciador de filas comum é que um servidor possui um link de comunicações dedicadas com cada cliente. Para obter informações adicionais sobre como criar canais para clientes e servidores, consulte Configurando o enfileiramento distribuído.

Um aplicativo IBM MQ MQI client e um gerenciador de filas do servidor se comunicam um com o outro usando um *canal MQI*. Um canal MQI é iniciado quando o aplicativo cliente emite uma chamada **MQCONN** ou **MQCONNX** para conectar-se ao gerenciador de filas e é finalizado quando o aplicativo cliente emite uma chamada **MQDISC** para se desconectar do gerenciador de fila. Os parâmetros de entrada de uma chamada MQI fluem em uma direção em um canal MQI e os parâmetros de saída fluem na direção oposta.

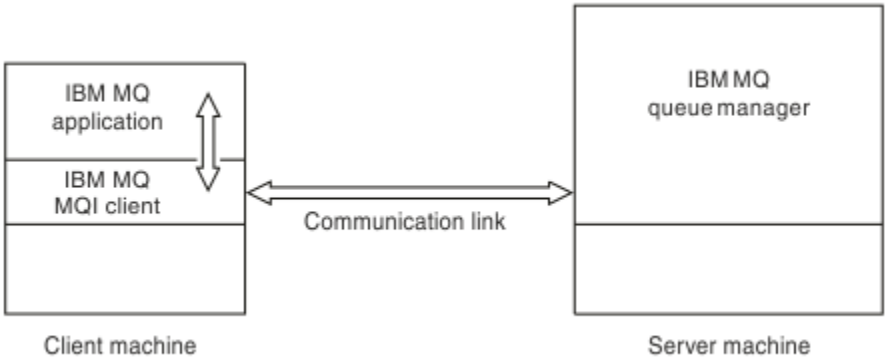


Figura 51. Link entre um Cliente e Servidor

As plataformas a seguir podem ser usadas. As combinações dependem de qual produto IBM MQ você está usando e estão descritas em "Suporte à plataforma para clientes IBM MQ" na página 146.

IBM MQ MQI client	**Servidor IBM MQ**
AIX and Linux Windows IBM i	AIX and Linux Windows IBM i z/OS

O MQI está disponível para aplicativos em execução na plataforma cliente; as filas e outros objetos IBM MQ são retidos em um gerenciador de fila que você instalou em um servidor.

Um aplicativo que você quer executar no ambiente do IBM MQ MQI client deve ser vinculado primeiro à biblioteca do cliente relevante. Quando o aplicativo emite uma chamada MQI, o IBM MQ MQI client direciona a solicitação para um gerenciador de filas, em que ela é processada e de onde uma resposta é enviada de volta para o IBM MQ MQI client.

O link entre o aplicativo e o IBM MQ MQI client é estabelecido dinamicamente no tempo de execução.

Também é possível desenvolver aplicativos clientes usando o IBM MQ classes for .NET, o IBM MQ classes for Java ou o IBM MQ classes for Java Message Service (JMS) e também usar os clientes do Java e do JMS nas plataformas a seguir:

- IBM i
- AIX
- Linux
- Windows

O uso de Java e JMS não está descrito aqui. Para obter detalhes completos sobre como instalar, configurar e usar o IBM MQ classes for Java e IBM MQ classes for JMS, consulte Usando o IBM MQ classes for Java e Usando o IBM MQ classes for JMS.

Aplicativos IBM MQ em um Ambiente Cliente-Servidor

Quando vinculado a um servidor, os aplicativos IBM MQ do cliente podem emitir a maioria das chamadas MQI da mesma maneira que os aplicativos locais. O aplicativo cliente emite uma chamada `MQCONN` para conectar-se a um gerenciador de filas especificado. Qualquer chamada MQI adicional que especificar o identificador de conexões retornado da solicitação de conexão é então processada por este gerenciador de filas.

Você deve vincular seus aplicativos para as bibliotecas de cliente apropriadas. Consulte Construindo aplicativos para IBM MQ MQI clients.

Conceitos relacionados

“Por que usar clientes IBM MQ?” na página 146
O uso de clientes do IBM MQ é uma maneira eficiente de implementar o sistema de mensagens e o enfileiramento do IBM MQ.

“O que É um Cliente Transacional Estendido?” na página 148
Um cliente transacional estendido do IBM MQ pode atualizar os recursos gerenciados por outro gerenciador de recursos, sob o controle de um gerenciador de transações externo.

“Como o Cliente se Conecta ao Servidor” na página 149
Um cliente se conecta a um servidor usando MQCONN ou MQCONNX e se comunica por meio de um canal.

“Gerenciamento de Transação e Suporte” na página 150
Uma introdução ao gerenciamento de transação e como o IBM MQ suporta as transações.

“Estendendo as instalações do gerenciador de filas” na página 152
É possível estender os recursos do gerenciador de filas usando saídas do usuário, saídas da API ou serviços instaláveis.

Informações relacionadas

Como configurar um IBM MQ MQI client

Por que usar clientes IBM MQ?

O uso de clientes do IBM MQ é uma maneira eficiente de implementar o sistema de mensagens e o enfileiramento do IBM MQ.

Você pode ter um aplicativo que usa o MQI em execução em uma máquina e o gerenciador de filas em execução em uma máquina diferente (física ou virtual). Os benefícios de fazer isto são:

- Não há a necessidade de uma implementação completa do IBM MQ na máquina cliente.
- Os requisitos de hardware no sistema do cliente foram reduzidos.
- Os requisitos de administração do sistema foram reduzidos.
- Um aplicativo IBM MQ em execução em um cliente pode se conectar a múltiplos gerenciadores de filas em sistemas diferentes.
- Canais alternativos usando diferentes protocolos de transmissão podem ser usados.

Suporte à plataforma para clientes IBM MQ

O IBM MQ em todas as plataformas do servidor suportadas aceita as conexões do cliente do IBM MQ MQI clients em várias plataformas.

O IBM MQ instalado como um *Produto base e servidor* em todas as plataformas do servidor suportadas pode aceitar conexões dos IBM MQ MQI clients nas plataformas a seguir:

- 

IBM i
- 

AIX
- 

Linux
- 

Windows

As conexões do cliente estão sujeitas a diferenças no coded character set identifier (CCSID) e no protocolo de comunicação.

Quais aplicativos executar em um IBM MQ MQI client?

O MQI completo é suportado no ambiente do cliente. Isso permite que quase qualquer aplicativo IBM MQ seja configurado para execução em um sistema IBM MQ MQI client vinculando o aplicativo no IBM MQ MQI client à biblioteca MQIC, e não à biblioteca MQI. As exceções são:

- MQGET com sinal
- Um aplicativo que precisa de coordenação do ponto de sincronização com outros gerenciadores de recursos deve usar um cliente transacional estendido

Se a leitura antecipada estiver ativada, para melhorar o desempenho do sistema de mensagens persistentes, nem todas as opções MQGET estarão disponíveis. A tabela mostra as opções que são permitidas e se elas podem ser alteradas entre chamadas MQGET.

Tabela 15. Opções MQGET Permitidas quando a Leitura Antecipada Está Ativada

Valores	**Permitido quando a leitura antecipada está ativada e pode ser alterada entre chamadas MQGET**	**Permitido quando a leitura antecipada está ativada e não pode ser alterada entre chamadas MQGET[1]**	**Opções MQGET que não são permitidas quando a leitura antecipada estiver ativada[2]**
Valores de MQGET MD	MsgId [3] CorrelId [3]	Codificação CodedCharSetId	
Opções 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 de MQGMO		MsgHandle	

1. Se estas opções forem alteradas entre chamadas MQGET, um código de razão MQRC_OPTIONS_CHANGED será retornado.
2. Se estas opções forem especificadas na primeira chamada MQGET, a leitura antecipada é desativada. Se essas opções forem especificadas em uma chamada MQGET subsequente, um código de razão MQRC_OPTIONS_ERROR será retornado.
3. Os aplicativos clientes precisam estar cientes que, se os valores de MsgId e CorrelId forem alterados entre chamadas MQGET, mensagens com os valores anteriores poderão já ter sido enviadas ao cliente e permanecerão no buffer de leitura antecipada do cliente até serem consumidos (ou limpos automaticamente).
4. A primeira chamada MQGET determina se mensagens devem ser navegadas ou obtidas a partir de uma fila quando a leitura antecipada está ativada. Se o aplicativo tentar usar uma combinação de navegação e obtenção, um código de razão MQRC_OPTIONS_CHANGED será retornado.
5. MQGMO_MSG_UNDER_CURSOR não é possível com a leitura antecipada. As mensagens podem ser navegadas ou obtidas quando a leitura antecipada é ativada, mas não uma combinação de ambos.

Um aplicativo em execução em um IBM MQ MQI client pode se conectar a mais de um gerenciador de filas simultaneamente ou usar um nome de gerenciador de filas com um asterisco (*) em uma chamada MQCONN ou MQCONNX (consulte os exemplos em Conectando aplicativos IBM MQ MQI client aos gerenciadores de filas).

O que É um Cliente Transacional Estendido?

Um cliente transacional estendido do IBM MQ pode atualizar os recursos gerenciados por outro gerenciador de recursos, sob o controle de um gerenciador de transações externo.

Se você não estiver familiarizado com os conceitos de gerenciamento de transações, consulte “Gerenciamento de Transação e Suporte” na página 150.

Observe que o cliente transacional XA agora é fornecido como parte do IBM MQ.

Um aplicativo cliente pode participar de uma unidade de trabalho gerenciada por um gerenciador de filas ao qual ele está conectado. Na unidade de trabalho, o aplicativo cliente pode colocar mensagens em, e obter mensagens de, filas que são pertencentes a esse gerenciador de filas. O aplicativo cliente pode então usar a chamada **MQCMIT** para confirmar a unidade de trabalho ou a chamada **MQBACK** para restaurar a unidade de trabalho. Entretanto, na mesma unidade de trabalho, o aplicativo cliente não pode atualizar os recursos de outro gerenciador de recursos, as tabelas de um banco de dados do Db2, por exemplo. O uso de um cliente transacional estendido do IBM MQ remove essa restrição.

Um cliente transacional estendido do IBM MQ é um IBM MQ MQI client com alguma função adicional. Usando esta função um aplicativo cliente, na mesma unidade de trabalho, pode executas as seguintes tarefas:

- Colocar mensagens em, e obter mensagens de, filas pertencentes ao gerenciador de filas ao qual ele está conectado
- Atualizar os recursos de um gerenciador de recursos diferente de um gerenciador de filas do IBM MQ

Esta unidade de trabalho deve ser gerenciada por um gerenciador de transações externo que está em execução no mesmo sistema que o aplicativo cliente. A unidade de trabalho não pode ser gerenciada pelo gerenciador de filas ao qual o aplicativo cliente está conectado. Isto significa que o gerenciador de filas pode agir somente como um gerenciador de recursos, não como um gerenciador de transações. Também significa que o aplicativo cliente pode confirmar ou efetuar backout da unidade de trabalho usando somente a interface de programação de aplicativos (API) fornecida pelo gerenciador de transações externo. O aplicativo cliente não pode, portanto, usar as chamadas MQI, **MQBEGIN**, **MQCMIT** e **MQBACK**.

O gerenciador de transações externo se comunica com o gerenciador de filas como um gerenciador de recursos usando o mesmo canal MQI que o usado pelo aplicativo cliente que está conectado ao gerenciador de filas. Entretanto, em uma situação de recuperação seguinte a uma falha, quando nenhum aplicativo está em execução, o gerenciador de transações pode usar um canal MQI dedicado para recuperar qualquer unidade de trabalho incompleta na qual o gerenciador de filas estava participando no momento da falha.

Nesta seção, um IBM MQ MQI client que não possui a função transacional estendida é referido como um cliente de base do IBM MQ. É possível considerar, portanto, um cliente transacional estendido do IBM MQ consistindo em um cliente de base do IBM MQ com a inclusão da função transacional estendida.

Nota: IBM i IBM MQ MQI client no IBM i não suporta a função transacional estendida do IBM MQ.

Suporte à Plataforma para Clientes Transacionais Estendidos

Multi

Clientes transacionais estendidos estão disponíveis para todas as Multiplataformas que suportam um cliente de base. Os clientes não estão disponíveis para o z/OS.

Um aplicativo cliente que está usando um cliente transacional estendido pode se conectar a um gerenciador de filas dos seguintes produtos IBM MQ 9.0ou mais recentes:

- AIX IBM MQ for AIX

- IBM i IBM MQ for IBM i
- Linux IBM MQ for Linux
- Windows IBM MQ for Windows

z/OS Embora não haja clientes transacionais estendidos que são executados no z/OS, um aplicativo cliente que está usando um cliente transacional estendido pode se conectar a um gerenciador de filas que é executado no z/OS.

Para cada plataforma, os requisitos de hardware e software para o cliente transacional estendido são iguais àqueles requisitos para o cliente base do IBM MQ. Uma linguagem de programação é suportada por um cliente transacional estendido se ele é suportado pelo cliente base do IBM MQ e pelo gerenciador de transações que está sendo usado.

Para obter informações sobre os gerenciadores de transações externas para todas as plataformas, consulte Requisitos do sistema para IBM MQ

Como o Cliente se Conecta ao Servidor

Um cliente se conecta a um servidor usando MQCONN ou MQCONNX e se comunica por meio de um canal.

Um aplicativo em execução no ambiente do cliente IBM MQ deve manter uma conexão ativa entre as máquinas do cliente e do servidor.

A conexão é feita por um aplicativo emitindo uma chamada MQCONN ou MQCONNX. Clientes e servidores se comunicam por meio de *canais MQI* ou, ao usar conversações de compartilhamento, cada conversação compartilha uma instância do canal MQI. Quando a chamada é bem-sucedida, a instância do canal MQI ou conversação permanece conectada até que o aplicativo emita uma chamada MQDISC. Este é o caso para cada gerenciador de filas ao qual um aplicativo precisa se conectar.

Cliente e Gerenciador de Filas na Mesma Máquina

Também é possível executar um aplicativo no ambiente do IBM MQ MQI client quando a sua máquina também tem um gerenciador de filas instalado.

Nesta situação, você tem a opção de vincular-se às bibliotecas do gerenciador de filas ou às bibliotecas do cliente, mas lembre-se que se você vincular-se às bibliotecas do cliente, ainda precisará definir as conexões de canal. Isto pode ser útil durante a fase de desenvolvimento de um aplicativo. É possível testar seu programa em sua própria máquina, sem dependência de outros, e ter a certeza de que ele ainda funcionará quando você movê-lo para um ambiente independente do IBM MQ MQI client.

Clientes em Plataformas Diferentes

Neste exemplo, a máquina servidor se comunica com três IBM MQ MQI clients em diferentes plataformas.

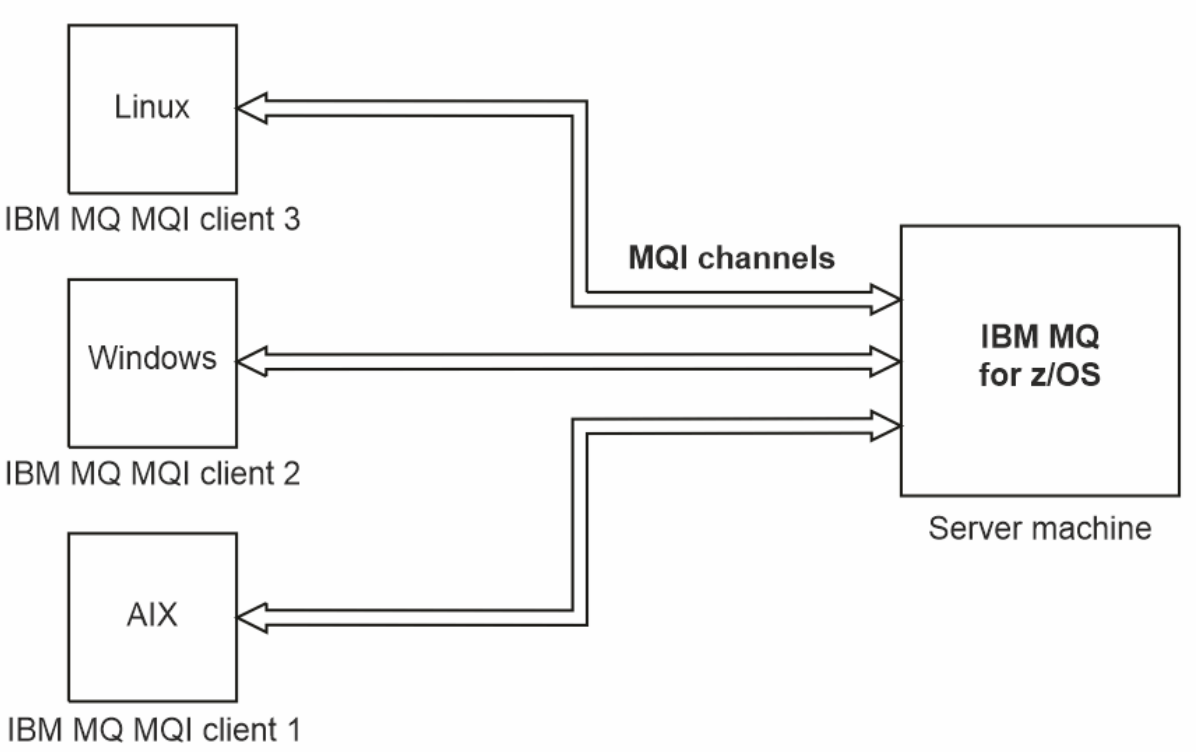


Figura 52. Servidor IBM MQ conectado aos clientes em differentes plataformas

Outros ambientes mais complexos são possíveis. Por exemplo, um cliente do IBM MQ pode se conectar a mais de um gerenciador de filas ou a qualquer número de gerenciadores de filas conectados como parte de um grupo de filas compartilhadas.

Usando Versões Diferentes do Software do Cliente e do Servidor

Se você estiver usando versões anteriores dos produtos IBM MQ, certifique-se de que a conversão de código do CCSID do seu cliente seja suportada pelo servidor.

Um cliente do IBM MQ pode se conectar a todas as versões suportadas do gerenciador de fila. Se você estiver conectando-se a um gerenciador de filas de versão anterior, não será possível usar recursos e estruturas de uma versão mais recente do produto em seu aplicativo IBM MQ no cliente.

Um gerenciador de filas do IBM MQ pode se comunicar com clientes em diferentes versões para si mesmo negociando até o nível de protocolo mais alto mutuamente suportado. Isso significa que os clientes mais antigos podem ser usados com níveis mais recentes do gerenciador de filas Recomenda-se que o cliente e o servidor estejam em versões do IBM MQ que estão atualmente no suporte para facilitar o diagnóstico de problemas e ativar o suporte por IBM

Para obter mais informações, consulte as linguagens de programação suportadas em Desenvolvendo aplicativos

Gerenciamento de Transação e Suporte

Uma introdução ao gerenciamento de transação e como o IBM MQ suporta as transações.

Um *gerenciador de recursos* é um subsistema do computador que possui e gerencia recursos que podem ser acessados e atualizados por aplicativos. Aqui estão exemplos de gerenciadores de recursos:

- Um gerenciador de filas do IBM MQ, com recursos que são suas filas
- Um Db2 do banco de dados, com recursos que são suas tabelas

Quando um aplicativo atualiza os recursos de um ou mais gerenciadores de recursos, pode haver um requisito de negócios para assegurar que determinadas atualizações sejam concluídas com êxito como um grupo ou que nenhuma delas seja concluída. A razão para este tipo de requisito é que os dados de negócios seriam deixados em um estado inconsistente se algumas destas atualizações fossem concluídas com êxito mas outras não.

As atualizações nos recursos que são gerenciados desta maneira são para ocorrer em uma *unidade de trabalho* ou em uma *transação*. Um programa de aplicativo pode agrupar um conjunto de atualizações em uma unidade de trabalho.

Durante uma unidade de trabalho, um aplicativo emite pedidos para gerenciadores de recursos atualizarem seus recursos. A unidade de trabalho é finalizada quando o aplicativo emite um pedido para confirmar todas as atualizações. Até as atualizações serem confirmadas, nenhuma delas se torna visível para outros aplicativos que estão acessando os mesmos recursos. Alternativamente, se o aplicativo decidir que ele não pode concluir a unidade de trabalho por qualquer razão, ele poderá emitir um pedido para restaurar todas as atualizações que ele solicitou até esse ponto. Neste caso, nenhuma das atualizações se tornará visível para outros aplicativos. Essas atualizações são relacionadas logicamente e devem ser todas bem-sucedidas para que a integridade de dados seja preservada. Se uma atualização for bem-sucedida enquanto outra falhar, a integridade de dados será perdida.

Quando uma unidade de trabalho for concluída com êxito, será considerada *confirmada*. Uma vez confirmadas, todas as atualizações feitas nessa unidade de trabalho serão permanentes e irreversíveis. No entanto, se a unidade de trabalho falhar, todas as atualizações serão *revertidas*. Este processo, no qual as unidades de trabalho são confirmadas ou revertidas com integridade, é conhecido como *coordenação do ponto de sincronização*.

O momento em que todas as atualizações em uma unidade de trabalho são confirmadas ou revertidas é chamado de *sync point*. Uma atualização em uma unidade de trabalho deve ocorrer *no controle do ponto de sincronização*. Se um aplicativo solicitar uma atualização que está *fora do controle do ponto de sincronização*, o gerenciador de recursos confirmará a atualização imediatamente, mesmo se houver uma unidade de trabalho em progresso e a atualização não puder ser revertida posteriormente.

O subsistema do computador que gerencia unidades de trabalho é chamado de *gerenciador de transações*, ou *ponto de coordenador*.

Uma unidade de trabalho *local* é aquela na qual os únicos recursos atualizados são aqueles do gerenciador de filas do IBM MQ. Aqui a coordenação do ponto de sincronização é fornecida pelo próprio gerenciador de filas usando um processo de confirmação de fase única.

Uma unidade global *global* é aquela na qual os recursos que pertencem a outros gerenciadores de recursos, como bancos de dados compatíveis com XA, também são atualizados. Aqui, um procedimento two-phase commit deve ser usado e a unidade de trabalho pode ser coordenada pelo próprio gerenciador de fila ou externamente por outro gerenciador de transações compatível com XA como IBM TXSeriesou BEA Tuxedo.

Um gerenciador de transações é responsável por assegurar que todas as atualizações nos recursos em uma unidade de trabalho sejam concluídas com êxito ou que nenhuma delas seja concluída. É para um gerenciador de transações que um aplicativo emite um pedido para confirmar ou restaurar uma unidade de trabalho. Exemplos de gerenciadores de transações são CICS e WebSphere Application Server, embora ambos tenham outra função também.

Alguns gerenciadores de recursos fornecem sua própria função de gerenciamento de transação. Por exemplo, um gerenciador de filas do IBM MQ pode gerenciar unidades de trabalho que envolvem atualizações em seus próprios recursos e atualizações para tabelas do Db2 . O gerenciador de filas não precisa de um gerenciador de transações separado para executar esta função, embora um possa ser usado se ele for um requisito do usuário. Se um gerenciador de transações separado for usado, ele será referido como um *gerenciador de transações externo*.

Para um gerenciador de transações externo gerenciar uma unidade de trabalho, deve haver uma interface padrão entre o gerenciador de transações e cada gerenciador de recursos que está participando da unidade de trabalho. Esta interface permite que o gerenciador de transações e um gerenciador de recursos se comuniquem entre si. Uma destas interfaces é a *Interface XA*, que é uma interface padrão

suportada por vários gerenciadores de transações e gerenciadores de recursos. A Interface XA é publicada pelo The Open Group em *Processamento de Transações Distribuídas: A Especificação XA*.

Quando mais de um gerenciador de recursos participa de uma unidade de trabalho, um gerenciador de transações deve usar um protocolo *two-phase commit* para assegurar que todas as atualizações na unidade de trabalho sejam concluídas com êxito ou que nenhuma delas seja concluída, mesmo se houver uma falha do sistema. Quando um aplicativo emitir um pedido para um gerenciador de transações confirmar uma unidade de trabalho, o gerenciador de transações fará o seguinte:

Fase 1 (Preparação para Confirmação)
O gerenciador de transações solicita a cada gerenciador de recursos que participa da unidade de trabalho para assegurar que todas as informações sobre as atualizações destinadas aos seus recursos estejam em um estado recuperável. Um gerenciador de recursos normalmente faz isto gravando as informações em um log e assegurando que as informações sejam gravadas no disco rígido. A Fase 1 é concluída quando o gerenciador de transações recebe notificação de cada gerenciador de recursos de que as informações sobre as atualizações destinadas aos seus recursos estão em um estado recuperável.

Fase 2 (Confirmação)
Quando a Fase 1 estiver concluída, o gerenciador de transações tomará a decisão irrevogável de confirmar a unidade de trabalho. Ele solicitará a cada gerenciador de recursos que participa da unidade de trabalho para confirmar as atualizações em seus recursos. Quando um gerenciador de recursos receber este pedido, ele deverá confirmar as atualizações. Ele não tem a opção de restaurar este estágio. A Fase 2 é concluída quando o gerenciador de transações recebe notificação de cada gerenciador de recursos de que ele confirmou as atualizações em seus recursos.

A Interface XA um protocolo two-phase commit.

Para obter mais informações, consulte Cenários de suporte transacional.

O IBM MQ também fornece suporte para o Microsoft Transaction Server (COM+). O uso do Microsoft Transaction Server (COM+) fornece informações sobre como configurar o IBM MQ para aproveitar o suporte do COM+.

Estendendo as instalações do gerenciador de filas

É possível estender os recursos do gerenciador de filas usando saídas do usuário, saídas da API ou serviços instaláveis.

Saídas do Usuário

As saídas do usuário fornecem um mecanismo para que você insira seu próprio código em uma função do gerenciador de filas. As saídas de usuário suportadas incluem:

Saídas do canal
Essas saídas alteram a maneira como os canais operam. As saídas de canal são descritas em Programas de Saída do Canal para Canais de Mensagem.

Saídas de conversão de dados
Essas saídas criam fragmentos do código de origem que podem ser colocados nos programas de aplicativos para converter os dados de um formato para outro. As saídas de conversão de dados estão descritas em Gravando saídas de conversão de dados.

A saída da carga de trabalho do cluster
A função executada por esta saída é definida pelo provedor da saída. As informações de definição de chamada são fornecidas em MQ_CLUSTER_WORKLOAD_EXIT - Descrição da Chamada.

Saídas de API

As saídas de API permitem gravar o código que altera o comportamento das chamadas de API do IBM MQ, como MQPUT e MQGET, e, em seguida, inserem esse código imediatamente antes ou depois dessas chamadas. A inserção é automática; o gerenciador de filas conduz o código de saída nos pontos

registrados. Para obter informações adicionais sobre as saídas de API, consulte Usando e Gravando as Saídas de API.

Serviços instaláveis

Os serviços instaláveis possuem interfaces formalizadas (uma API) com diversos pontos de entrada.

Uma implementação de um serviço instalável é denominada um *componente de serviço*. É possível usar os componentes fornecidos com o IBM MQ, ou você pode gravar seu próprio componente para executar as funções que precisa.

Atualmente, são fornecidos os seguintes serviços instaláveis:

Serviço de autorização
O serviço de autorização permite construir seu próprio recurso de segurança.

O componente de serviço padrão que implementa o serviço é o Object Authority Manager (OAM). Por padrão, o OAM fica ativo e você não tem que fazer nada para configurá-lo. É possível usar a interface do serviço de autorização para criar outros componentes para substituir ou aumentar o OAM. Para obter mais informações sobre o OAM, consulte Configurando segurança em sistemas AIX, Linux, and Windows.

Serviço de Nomes
O serviço de nomes permite que os aplicativos compartilhem as filas identificando as filas remotas como se fossem filas locais.

É possível gravar seu próprio componente de serviço de nomes. Talvez você queira fazer isso se pretender usar o serviço de nomes com IBM MQ, por exemplo. Para usar o serviço de nomes você deve ter um componente que seja gravado pelo usuário ou fornecido por um fornecedor de software diferente. Por padrão, o serviço de nomes está inativo.

Conceitos relacionados
Saídas de usuário, saídas de API e serviços instaláveis do IBM MQ

Interfaces de linguagem do IBM MQ Java

O IBM MQ fornece três interfaces de programação de aplicativos (APIs) para uso em Java aplicativos: IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMSe IBM MQ classes for Java.

O IBM suporta e é um participante ativo em padrões abertos.

- No IBM MQ 8.0, o produto implementa o padrão JMS 2.0 , que introduziu uma nova API simplificada juntamente com recursos como assinaturas compartilhadas.
- Em IBM MQ 9.3.0, Jakarta Messaging 3.0 também é suportado.
- Além disso, o WebSphere Liberty suporta JMS 2.0 e Jakarta Messaging 3.0 com IBM MQ.

No IBM MQ há três APIs para uso em aplicativos Java :

JM 3.0 **IBM MQ classes for Jakarta Messaging**
IBM MQ classes for Jakarta Messaging é um provedor Jakarta Messaging que implementa a interface Jakarta Messaging para IBM MQ como o sistema de mensagens. O Jakarta Connectors Architecture fornece uma maneira padrão de conectar aplicativos em execução em um ambiente do Jakarta EE a um Enterprise Information System (EIS), como IBM MQ ou Db2.

JMS 2.0 **IBM MQ classes for JMS**
IBM MQ classes for JMS é um provedor JMS que implementa a interface JMS para IBM MQ como o sistema de mensagens. O Java Platform, Enterprise Edition Connector Architecture (JCA) fornece uma maneira padrão de conectar aplicativos em execução em um ambiente Java EE a um Enterprise Information System (EIS) como IBM MQ ou Db2.

IBM MQ classes for Java

O IBM MQ classes for Java permite usar o IBM MQ em um ambiente Java. O IBM MQ classes for Java permite que um aplicativo Java se conecte ao IBM MQ como um cliente IBM MQ ou se conecte diretamente a um gerenciador de filas do IBM MQ.

Nota:

- O JMS 2.0 foi substituído por Jakarta Messaging O IBM MQ classes for JMS continua a suportar o padrão JMS 2.0 , mas aprimoramentos futuros no sistema de mensagens do Java surgirão apenas no Jakarta Messaging, portanto, no IBM MQ classes for Jakarta Messaging. IBM MQ classes for JMS são recomendados apenas para manter e estender aplicativos JMS 2.0 existentes. IBM MQ classes for Jakarta Messaging deve ser a tecnologia preferencial para novo desenvolvimento.
- Stabilized O IBM MQ classes for Java é funcionalmente estabilizado no nível enviado no IBM MQ 8.0. Aplicativos existentes que usam o IBM MQ classes for Java continuarão a ser totalmente suportados, mas essa API ficará estabilizada, portanto, novos recursos não serão incluídos e solicitações de aprimoramentos rejeitadas. "Totalmente suportado" significa que os defeitos serão corrigidos, além de quaisquer mudanças necessárias, por meio de mudanças nos requisitos do sistema IBM MQ.

JM 3.0 Em IBM MQ 9.3, o IBM MQ classes for Java, o IBM MQ classes for JMSe o IBM MQ classes for Jakarta Messaging são construídos com o Java 8. Os ambientes de tempo de execução do Java nesses ou acima desses níveis devem ser usados para executar aplicativos usando essas interfaces

Conceitos relacionados

Acessando o IBM MQ de Java -Opção de API

Por que devo usar as classes do IBM MQ para Jakarta Messaging?

Por que devo usar o IBM MQ classes for JMS?

Por que devo usar o IBM MQ classes for Java?

IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging são os provedores de sistemas de mensagens fornecidos com IBM MQ. Cada um desses provedores também fornece dois conjuntos para a API do sistema de mensagens. Os aplicativos Java Platform, Standard Edition (Java SE) e Java Platform, Enterprise Edition (Java EE) podem usar esses provedores de sistemas de mensagens.

JM 3.0 IBM MQ 9.3.0 introduziu o suporte para o Jakarta Messaging 3.0 JMS 2.0 ainda é totalmente suportado.

As especificações JMS e Jakarta Messaging definem um conjunto de interfaces que os aplicativos podem usar para executar operações do sistema de mensagens O produto suporta a versão do JMS 2.0 do padrão JMS Essa implementação oferece todos os recursos da API clássica, mas requer menos interfaces e é mais simples de usar. Para obter mais informações, consulte “JMS e Jakarta Messaging modelo” na página 158 e a especificação do JMS 2.0 em Java.net. JM 3.0 De IBM MQ 9.3.0, Jakarta Messaging também é suportado.

O pacote jakarta.jms (Jakarta Messaging 3.0) ou javax.jms (JMS 2.0) especifica os detalhes das interfaces de sistema de mensagens e um provedor de sistemas de mensagens implementa essas interfaces para um produto de sistema de mensagens específico. Por exemplo:

- O IBM MQ classes for JMS é um provedor do JMS que implementa as interfaces do JMS para o IBM MQ e também fornece os dois conjuntos de extensões a seguir para a API do JMS:
  - Extensões do IBM MQ JMS
  - Extensões do IBM JMS
- Um connection factory, fila ou objeto de tópico criado usando as interfaces javax.dimsou jakarta.jmsou um conjunto de extensões JMS pode ser endereçado usando qualquer uma dessas APIs, ou seja, ele pode ser convertido para qualquer uma das interfaces. Para manter a portabilidade do aplicativo no nível mais alto, use a API mais genérica adequada para seus requisitos.

Como JMS e Jakarta Messaging compartilham muito em comum, referências adicionais a JMS neste tópico podem ser consideradas referentes a ambos. Quaisquer diferenças são destacadas conforme necessário.

Extensões do IBM MQ JMS

IBM MQ classes for JMS também fornece extensões para a API do JMS. O IBM MQ classes for JMS contém extensões que são implementadas nos objetos MQConnectionFactory, MQQueue e MQTopic. Esses objetos possuem propriedades e métodos que são específicos para IBM MQ. Os objetos podem ser objetos administrados ou um aplicativo pode criar os objetos dinamicamente no tempo de execução. Essas extensões são denominadas extensões IBM MQ JMS . Observe que, nesta documentação, os objetos criados dinamicamente por um aplicativo no tempo de execução não são considerados objetos administrados.

Extensões do IBM JMS

Além das extensões IBM MQ JMS , IBM MQ classes for JMS fornece um conjunto mais genérico de extensões para a JMS API ou Java como a linguagem de programação usada. Essas extensões são denominadas extensões do IBM JMS e têm os objetivos amplos a seguir:

- Para fornecer um nível maior de consistência entre os provedores do IBM JMS
- Para facilitar a gravação de um aplicativo de ponte entre dois sistemas de mensagens do IBM .
- Para facilitar a porta de um aplicativo de um provedor do IBM JMS para outro.

O principal foco dessas extensões refere-se à criação e configuração de connection factories e destinos dinamicamente no tempo de execução, mas as extensões também fornecem a função que não está diretamente relacionada ao sistema de mensagens, como a função para determinação de problema.

Tarefas relacionadas

Usando classes do IBM MQ para o Sistema de Mensagens JMS/Jakarta

Configurando recursos do JMS e do Jakarta Messaging

JM 3.0 IBM MQ classes for Jakarta Messaging: uma visão geral

IBM MQ 9.3.0 apresenta suporte para Jakarta Messaging. Para Jakarta Messaging 3.0, o controle da especificação JMS foi movido de Oracle para o Java Community Process. No entanto, o Oracle retém o controle do nome "javax", que é usado em outras tecnologias Java . Portanto, embora Jakarta Messaging 3.0 seja funcionalmente equivalente a JMS 2.0, há algumas diferenças na nomenclatura.. O nome oficial para a versão 3.0 é Jakarta Messaging em vez de Java Message Servicee os nomes de pacotes e constantes são prefixados com `jakarta` em vez de `javax`.

Plano de fundo

Por muitos anos, a plataforma Java veio em dois formulários- Standard Edition e Enterprise Edition.

Java Platform, Standard Edition (às vezes abreviado como Java SE) é a linguagem principal e bibliotecas de classe, capaz de ser executado em um contexto independente. A maioria dos pacotes Java em Java SE têm nomes que começam com "java.".

Java Platform, Enterprise Edition (Java EE) estende isso, incluindo funcionalidade como Sistema de Mensagens, vários Beans, transacional e assim por diante. Algumas dessas tecnologias também podem ser usadas em um contexto do Java SE A maioria dos pacotes Java no Java EE historicamente têm nomes que começam com "javax."-há algum cruzamento, no entanto, alguns pacotes Java SE têm "javax." como o prefixo para o nome deles.

O Java Message Service (JMS) faz parte de Java Platform, Enterprise Edition Java EE 7 incorpora JMS 2.0.

Até o Java EE 7, as tecnologias estavam sob a administração do Oracle

As tecnologias Java EE mudaram recentemente da administração do Oracle para um processo da comunidade supervisionado pela Fundação Eclipse .

Como o "javax". O nome não pôde ser movido para o novo projeto, a nova nomenclatura foi adotada-todos os pacotes e nomes de propriedades agora são prefixados com "jakarta". e o Java Platform, Enterprise Edition será chamado de "Jakarta EE" no futuro. A numeração da versão continuou: a versão 8 foi uma versão provisória que pode ser amplamente ignorada, e Jakarta EE 9 é o ponto em que o "jakarta". O prefixo entra em vigor

A principal tecnologia Jakarta EE que se aplica no contexto IBM MQ é Jakarta Messaging 3.0 -o sucessor de Java Message Service (JMS) 2.0. Portanto, o Jakarta EE 9 incorpora Jakarta Messaging 3.0

O IBM MQ continua a suportar Java EE 7 e JMS 2.0, enquanto introduz suporte para Jakarta EE 9 e Jakarta Messaging 3.0.

O que é entregue: Java SE

Para o Java Platform, Standard Edition, além do IBM MQ classes for JMS (que suporta operações JMS 2.0 com IBM MQ) IBM MQ 9.3.0 e versões mais recentes, forneça IBM MQ classes for Jakarta Messaging. Essas classes fornecem um provedor Jakarta Messaging 3.0 que se integra ao IBM MQ, permitindo o uso de gerenciadores de filas do IBM MQ para facilitar operações do Jakarta Messaging .

Elas são fornecidas como um arquivo JAR padrão, `com.ibm.mq.jakarta.client.jar`, no subdiretório `java/lib` da instalação do IBM MQ

Para uso em contêineres OSGi, como Apache Felix ou Eclipse Equinox, IBM MQ também fornece um par de pacotes configuráveis OSGi:

- `com.ibm.mq.osgi.jms30.clientprereqs_`*`V.R.M.F`*`.jar`
- `com.ibm.mq.osgi.jms30.client_`*`V.R.M.F`*`.jar`

em que *V.R.M.F* representa a versão de IBM MQ, por exemplo 9.3.0.0. Esses pacotes configuráveis podem ser localizados no subdiretório `java/lib/OSGi` da instalação IBM MQ .

O que é entregue: Jakarta EE 9

Para suportar o sistema de mensagens baseado no IBM MQem um Jakarta EE 9 servidor de aplicativos compatível, o IBM MQ fornece um Jakarta EE 9Adaptador de Recursos compatível: `wmq.jakarta.jmsra.rar`. Isso pode ser localizado no subdiretório `java/lib/jca` da instalação IBM MQ .

O IBM MQ continua a fornecer um Java EE 7 Adaptador de Recursos compatível, `wmq.jmsra.rar`, no subdiretório `java/lib/jca` da instalação do IBM MQ

Como esses artefatos são entregues?.

Esses JARs e o arquivo RAR para o Adaptador de Recursos são empacotados com os artefatos pré-existentes na mídia de instalação usual do IBM MQ -a mídia de instalação específica da plataforma, como arquivos "`.rpm`" e a mídia redistribuível, como os arquivos JAR do cliente redistribuíveis autoextraídos.

O que mudou entre JMS 2.0 e Jakarta Messaging 3.0

Jakarta EE 9 e Jakarta Messaging 3.0 não introduzem nenhuma nova funcionalidade Tudo o que muda são nomes. Por exemplo, quando você usa "javax.jms.Connection" em JMS 2.0, usa "jakarta.jms.Connection" em Jakarta Messaging 3.0.

À medida que a Eclipse Foundation leva a plataforma Jakarta EE adiante, ela será construída sobre essa base e essa convenção de nomenclatura será usada para novas funcionalidades introduzidas no futuro

O que mudou entre IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging

Resumo

O IBM MQ classes for JMS, que fornece suporte para o JMS 2.0, permanece disponível e é recomendado principalmente para manter e estender aplicativos existentes. Eles são totalmente suportados.

O IBM MQ classes for Jakarta Messaging, que fornece suporte para o Jakarta Messaging 3.0, é recomendado para novo desenvolvimento.

No IBM MQ 9.3.0, essas duas ofertas eram funcionalmente equivalentes.. Apenas a nomenclatura é diferente No entanto, é mais provável que a nova funcionalidade do sistema de mensagens surja no IBM MQ classes for Jakarta Messaging do que no IBM MQ classes for JMS

As duas ofertas são interoperáveis. As mensagens produzidas por IBM MQ classes for JMS podem ser consumidas por IBM MQ classes for Jakarta Messaginge vice-versa Mas as duas ofertas não devem coexistir em um único aplicativo.

Mudanças de nomenclatura

Tabela 16. Mudanças nos nomes de pacotes..

IBM MQ classes for JMS nome do pacote	**IBM MQ classes for Jakarta Messaging nome do pacote**
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.*

Os pacotes relacionados a serviços comuns (rastreio, criação de log, suporte de idioma nacional, etc.) e as implementações de JMQI (local e remoto) são comuns ao IBM MQ classes for JMS e ao IBM MQ classes for Jakarta Messaging, portanto, nenhuma mudança é necessária nessas áreas.

Observe que os nomes de propriedades também foram alterados Por exemplo, a propriedade para ativar as extensões IBM MQ em IBM MQ classes for Jakarta Messaging é **com.ibm.mq.jakarta.jms.SupportMQExtensions**

Nomes de propriedades que são independentes de IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, como as várias propriedades **com.ibm.msg.client.commonservices.trace.*** , se aplicam igualmente a ambas as ofertas.

Utilitários Administrativos

Os utilitários **crtmqenv** e **setmqenv** agora aceitam uma opção para especificar se o caminho de classe deve ser configurado para IBM MQ classes for JMS (`-j 2.0`) ou IBM MQ classes for Jakarta Messaging (`-j 3.0`) e há IBM MQ classes for Jakarta Messaging variantes dos utilitários **runjms** , chamados **runjms30** e nomes semelhantes.

O utilitário **dspmqver** , quando solicitado a relatar sobre componentes Java , inclui IBM MQ classes for Jakarta Messaging em sua saída.

Para configurar objetos IBM MQ classes for Jakarta Messaging a serem recuperados por meio de JNDI, o novo utilitário **JMS30Admin** é equivalente ao utilitário **JMSAdmin** para IBM MQ classes for JMS.

Observe que os objetos subjacentes são de pacotes diferentes. As definições JNDI criadas por **JMSAdmin** não podem ser usadas por IBM MQ classes for Jakarta Messaging, nem aquelas criadas por **JMS30Admin** não podem ser usadas por IBM MQ classes for JMS

Nota: Não há suporte para os objetos IBM MQ classes for Jakarta Messaging fornecidos pelo IBM MQ Explorer; sua integração JNDI é apenas para IBM MQ classes for JMS

Conceitos relacionados

Por que devo usar as classes do IBM MQ para Jakarta Messaging?

JMS e Jakarta Messaging modelo

O modelo JMS e Jakarta Messaging define um conjunto de interfaces que os aplicativos Java podem usar para executar operações do sistema de mensagens.. IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging são provedores de sistemas de mensagens que definem como os objetos de sistema de mensagens do Java estão relacionados aos conceitos do IBM MQ As especificações JMS e Jakarta Messaging esperam que determinados objetos do mensagens sejam objetos administrados.

A partir do IBM MQ 8.0, o produto suporta a versão JMS 2.0 do padrão JMS, que introduziu uma API simplificada, enquanto também retém a API clássica, do JMS 1.1.

JM 3.0 IBM MQ 9.3.0 introduziu o suporte para o Jakarta Messaging 3.0 JMS 2.0 ainda é totalmente suportado. Como JMS e Jakarta Messaging compartilham muito em comum, referências adicionais a JMS neste tópico podem ser consideradas referentes a ambos. Quaisquer diferenças são destacadas conforme necessário.

API simplificada

O JMS 2.0 introduziu a API simplificada, enquanto também retém as interfaces específicas do domínio e independentes do domínio do JMS 1.1. A API simplificada reduz o número de objetos que são necessários para enviar e receber mensagens e consiste nas interfaces a seguir:

ConnectionFactory
: Um ConnectionFactory é um objeto administrado usado por um cliente JMS para criar um Connection. Essa interface também é usada na API clássica.

JMSContexto
: Esse objeto combina os objetos Connection e Session da API clássica. Os objetos JMSContext podem ser criados a partir de outros objetos JMSContext, com a conexão subjacente sendo duplicada.

JMSProdutor
: Um JMSProducer é criado por um JMSContext e é usado para enviar mensagens para uma fila ou tópico. O objeto JMSProducer cria objetos que são necessários para enviar a mensagem.

Consumidor do JMS
: Um JMSConsumer é criado por um JMSContext e é usado para receber mensagens de um tópico ou uma fila.

A API simplificada tem diversos efeitos:

- O objeto JMSContext sempre inicia automaticamente a conexão subjacente.
- JMSProducers e JMSConsumers agora podem trabalhar diretamente com corpos de mensagens, sem precisarem obter todo o objeto de mensagem, usando o método getBody da Mensagem.
- Propriedades de mensagens podem ser configuradas no objeto JMSProducer, usando encadeamento de método antes de enviar um 'body', um conteúdo de mensagem. O JMSProducer manipulará a criação de todos os objetos necessários para enviar a mensagem. Usando o JMS 2.0, as propriedades podem ser configuradas e uma mensagem pode ser enviada da seguinte forma:

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

O JMS 2.0 também introduziu assinaturas compartilhadas em que as mensagens podem ser compartilhadas entre diversos consumidores Todas as assinaturas do JMS 1.1 são tratadas como assinaturas não compartilhadas.

API clássica

A lista a seguir resume as principais interfaces do JMS da API clássica:

Destino
Um destino é para onde um aplicativo envia mensagens ou é uma origem da qual um aplicativo recebe mensagens, ou ambos.

ConnectionFactory
Um objeto ConnectionFactory contém um conjunto de propriedades de configuração para uma conexão. Um aplicativo usa um connection factory para criar uma conexão.

Conexão
Um objeto Connection contém a conexão ativa de um aplicativo com um servidor de sistema de mensagens. Um aplicativo usa uma conexão para criar sessões.

Sessão
Uma sessão é um único contexto encadeado para enviar e receber mensagens. Um aplicativo usa uma sessão para criar mensagens, produtores de mensagens e consumidores de mensagens. Uma sessão é transacionada ou não transacionada.

Mensagem
Um objeto Message contém uma mensagem que um aplicativo envia ou recebe.

MessageProducer
Um aplicativo usa um produtor de mensagem para enviar mensagens para um destino.

MessageConsumer
Um aplicativo usa um consumidor de mensagem para receber mensagens enviadas a um destino.

Figura 53 na página 159 mostra esses objetos e seus relacionamentos.

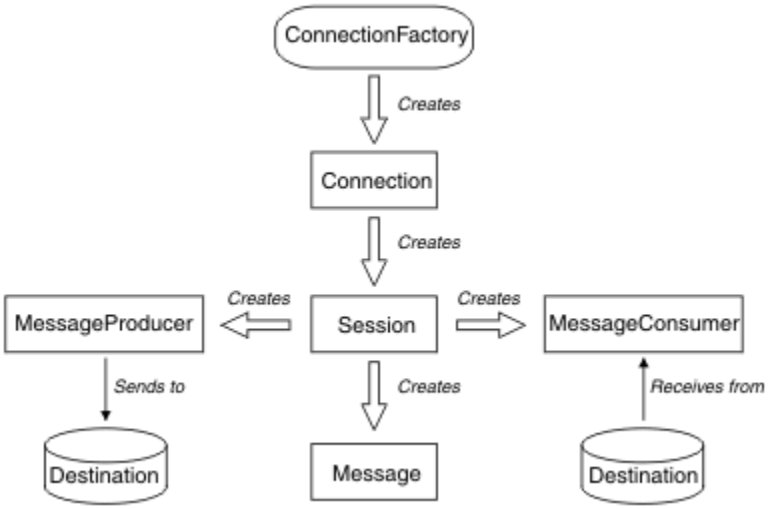


Figura 53. Os objetos do JMS e seus relacionamentos

O diagrama mostra as principais interfaces: ConnectionFactory, Connection, Session, MessageProducer, MessageConsumer, Message e Destination. Um aplicativo usa um connection factory para criar uma conexão e usa uma conexão para criar sessões. O aplicativo pode então usar uma sessão para criar mensagens, produtores de mensagens e consumidores de mensagens. O aplicativo usa um produtor de mensagem para enviar mensagens para um destino e usa um consumidor de mensagens para receber mensagens enviadas para um destino.

Um objeto Destination, ConnectionFactory ou Connection pode ser utilizado simultaneamente por diferentes encadeamentos de um aplicativo multiencadeado, mas um objeto Session, MessageProducer ou MessageConsumer não pode ser usado simultaneamente por diferentes encadeamentos. A maneira mais simples de garantir que um objeto Session, MessageProducer ou MessageConsumer não é usado simultaneamente é criar um objeto Session separado para cada encadeamento.

O JMS suporta dois estilos de sistema de mensagens:

- Sistema de mensagens ponto a ponto
- Sistema de Mensagens de Publicação/Assinatura

Esses estilos de sistema de mensagens também são referidos como *domínios do sistema de mensagens* e é possível combinar ambos os estilos do sistema de mensagens em um aplicativo. No domínio ponto a ponto, um destino é uma fila e, no domínio de publicar/assinar, um destino é um tópico.

Com versões do JMS anteriores ao JMS 1.1, a programação para o domínio ponto a ponto usa um conjunto de interfaces e métodos e a programação para o domínio de publicação/assinatura usa outro conjunto. Os dois conjuntos são semelhantes, mas separados. A partir do JMS 1.1, é possível usar um conjunto comum de interfaces e métodos que suportam ambos os domínios do sistema de mensagens. As interfaces comuns fornecem uma visualização independente de cada domínio de sistema de mensagens. Tabela 17 na página 160 lista as interfaces independentes de domínio do JMS e suas interfaces específicas de domínio correspondentes.

Tabela 17. O domínio do JMS independente e suas interfaces específicas de domínio correspondentes

Interfaces independentes de domínio	**Interfaces específicas de domínio para o domínio ponto a ponto**	**Interfaces específicas de domínio para o domínio de publicar/assinar**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Conexão	QueueConnection	TopicConnection
Destino	Fila	Tópico
Sessão	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 O IBM MQ classes for JMS 2.0 suporta ambas as interfaces específicas do domínio do JMS 1.1 anteriores e a API simplificada do JMS 2.0 IBM MQ classes for JMS 2.0 pode, portanto, ser usado para manter aplicativos existentes, incluindo o desenvolvimento de nova função em aplicativos existentes.

JM 3.0 O IBM MQ classes for Jakarta Messaging 3.0 suporta as versões do Jakarta Messaging das mesmas interfaces e é recomendado para o novo desenvolvimento de aplicativo

Em IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, JMS objetos são relacionados a conceitos IBM MQ das seguintes maneiras:

- Um objeto Connection tem propriedades derivadas das propriedades do connection factory que foi usado para criar a conexão. Essas propriedades controlam como um aplicativo se conecta a um gerenciador de filas. Os exemplos dessas propriedades são o nome do gerenciador de filas e, para um aplicativo que se conecta ao gerenciador de filas no modo cliente, o nome do host ou o endereço IP do sistema em que o gerenciador de filas está em execução.
- Um objeto Session contém uma manipulação de conexões do IBM MQ que, portanto, define o escopo transacional da sessão.
- Cada objeto MessageProducer e cada objeto MessageConsumer contém uma manipulação de objetos do IBM MQ.

Ao usar IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, todas as regras normais de IBM MQ se aplicam.. Observe, especificamente, que um aplicativo pode enviar uma mensagem a uma fila remota, mas pode receber uma mensagem somente de uma fila de propriedade do gerenciador de filas ao qual o aplicativo está conectado.

A especificação do JMS espera que os objetos ConnectionFactory e Destination sejam objetos administrados. Um administrador cria e mantém objetos administrados em um repositório central e um aplicativo JMS recupera esses objetos usando a Java Naming and Directory Interface (JNDI).

Em IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, a implementação da interface de Destino é uma superclasse abstrata de Fila e Tópico e, portanto, uma instância de Destino é um objeto Fila ou um objeto Tópico. As interfaces independentes de domínio tratam uma fila ou um tópico como um destino. O domínio do sistema de mensagens para um objeto MessageProducer ou MessageConsumer é determinado por se o destino é uma fila ou um tópico.

Em IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging , portanto, os objetos dos seguintes tipos podem ser objetos administrados:

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- Fila
- Tópico
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

IBM MQ classes for JMS/Jakarta Messaging arquitetura

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging têm uma arquitetura em camadas. A camada mais superior do código é uma camada comum que qualquer provedor de sistemas de mensagens do IBM Java pode utilizar.

JM 3.0 IBM MQ 9.3.0 introduziu o suporte para o Jakarta Messaging 3.0 JMS 2.0 ainda é totalmente suportado.

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging possuem uma arquitetura em camadas, conforme mostrado no diagrama Figura 54 na página 162. A camada superior do código é uma camada comum que pode ser usada por qualquer provedor IBM JMS ou Jakarta Messaging. Quando um aplicativo chama um método JMS ou Jakarta Messaging, qualquer processamento da chamada que não seja específico para um sistema de mensagens é executado por uma camada comum, que também fornece uma resposta consistente para a chamada Qualquer processamento da chamada que seja específico a um sistema de mensagens é delegado a uma camada inferior. No diagrama a seguir, o provedor de sistemas de mensagens do IBM MQ é mostrado na camada inferior, juntamente com dois provedores de sistema de mensagens adicionais (Provedor de sistemas de mensagens A e Provedor de sistemas de mensagens B).

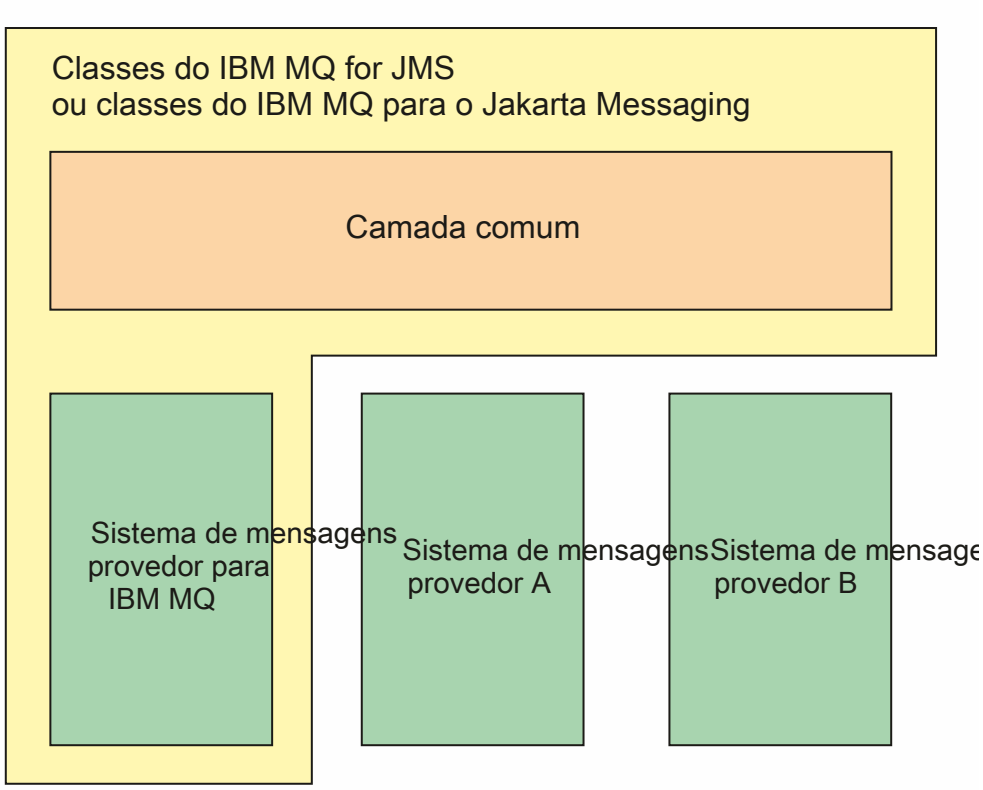


Figura 54. A arquitetura em camadas para os provedores IBM JMS e Jakarta Messaging

Uma arquitetura em camadas atinge os seguintes objetivos:

- Para melhorar a consistência do comportamento dos vários provedores IBM JMS e Jakarta Messaging
- Facilitar a gravação de um aplicativo de ligação entre dois sistemas de mensagens IBM
- Para facilitar a porta de um aplicativo de um provedor do IBM JMS ou Jakarta Messaging para outro

Tarefas relacionadas

Usando classes do IBM MQ para o Sistema de Mensagens JMS/Jakarta

Suporte para objetos administrados

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging suportam o uso de objetos administrados.

JM 3.0 De IBM MQ 9.3.0, Jakarta Messaging 3.0 é suportado para desenvolver novos aplicativos. IBM MQ 9.3.0 e posterior continuam a suportar o JMS 2.0 para aplicativos existentes. Não é suportado usar a API Jakarta Messaging 3.0 e a API JMS 2.0 no mesmo aplicativo. Para obter mais informações, consulte Usando IBM MQ classes para JMS/Jakarta Messaging.

O fluxo de lógica em um aplicativo JMS ou IBM MQ classes for Jakarta Messaging começa com ConnectionFactory e objetos de Destino. O aplicativo usa um objeto ConnectionFactory para criar um objeto Connection, que representa a conexão ativa do aplicativo com um servidor de sistema de mensagens. O aplicativo usa o objeto Connection para criar um objeto Session, que é um único contexto encadeado para produzir e consumir mensagens. O aplicativo pode então usar o objeto Session e um objeto Destination para criar um objeto MessageProducer, que o aplicativo usa para enviar mensagens para o destino especificado. O destino é uma fila ou um tópico no sistema de mensagens e é contido pelo objeto Destination. O aplicativo também pode usar o objeto Session e um objeto Destination para criar um objeto MessageConsumer, que o aplicativo usa para receber mensagens que foram enviadas para o destino especificado.

As especificações JMS e Jakarta Messaging esperam que ConnectionFactory e objetos de Destino sejam objetos administrados. Um administrador cria e mantém objetos administrados em um repositório central e um aplicativo JMS ou Jakarta Messaging recupera esses objetos usando Java Naming Directory Interface (JNDI). O repositório de objetos administrados pode variar de um arquivo simples para um diretório LDAP (Lightweight Directory Access Protocol)..

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging suportam o uso de objetos administrados. Um aplicativo pode usar todos os recursos do IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging que são expostos por meio do IBM MQ sem ter nenhuma informação específica do IBM MQcodificada permanentemente no próprio aplicativo Esta organização fornece ao aplicativo um grau de independência da configuração do IBM MQ subjacente.

Para alcançar essa independência, o aplicativo pode usar JNDI para recuperar connection factories e destinos armazenados como objetos administrados e usar apenas as interfaces definidas no pacote `javax.jms` (JMS 2.0) ou `jakarta.jms` (Jakarta Messaging 3.0) para executar operações do sistema de mensagens.

JMS 2.0 Para JMS 2.0, um administrador pode usar a IBM MQ JMS ferramenta de administração **JMSAdmin**ou IBM MQ Explorerpara criar e manter objetos administrados em um repositório central.

JM 3.0 Para Jakarta Messaging 3.0, não é possível administrar o JNDI usando IBM MQ Explorer A administração de JNDI é suportada pela variante Jakarta Messaging 3.0 de **JMSAdmin**, que é **JMS30Admin**

Um servidor de aplicativos geralmente fornece seu próprio repositório para objetos administrados e suas ferramentas para criar e manter os objetos. Um aplicativo Java EE JM 3.0 ou Jakarta EE pode, portanto, usar JNDI para recuperar objetos administrados a partir do repositório do servidor de aplicativos ou de um repositório central.

Tarefas relacionadas

Configurando recursos do JMS e do Jakarta Messaging

Tipos de comunicação suportados nas plataformas Java EE e Jakarta EE

Nas plataformas Java EE e Jakarta EE , IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging suportam dois tipos de comunicação entre um componente de um aplicativo e um gerenciador de filas do IBM MQ .

JM 3.0 IBM MQ 9.3.0 introduziu o suporte para o Jakarta Messaging 3.0 JMS 2.0 ainda é totalmente suportado. Como JMS e Jakarta Messaging compartilham muito em comum, referências adicionais a JMS neste tópico podem ser consideradas referentes a ambos. Quaisquer diferenças são destacadas conforme necessário.

Os dois tipos de comunicação a seguir entre um componente de um aplicativo e um gerenciador de filas do IBM MQ são suportados:

- Comunicação de saída
- Comunicação de entrada

Comunicação de saída

Usando a API JMS ou Jakarta Messaging diretamente, um componente de aplicativo cria uma conexão com um gerenciador de filas e, em seguida, envia e recebe mensagens..

Por exemplo, o componente de aplicativo pode ser um aplicativo cliente, um servlet, um JavaServer Page (JSP), um enterprise Java bean (EJB), ou um bean acionado por mensagens (MDB). Neste tipo de comunicação, o contêiner do servidor de aplicativos fornece somente funções de baixo nível para suportar operações do sistema de mensagens, como definição do conjunto de conexões e gerenciamento de encadeamentos.

Comunicação de entrada

No caso da comunicação de entrada, uma mensagem que chega a um destino é entregue a um MDB que, em seguida, processa a mensagem.

Os aplicativos Java EE JM 3.0 e Jakarta EE usam MDBs para processar mensagens assincronamente. Um MDB age como um listener de mensagem do JMS e é implementado por um método onMessage(), que define como uma mensagem é processada. Um MDB está implementado no contêiner EJB de um servidor de aplicativos. A maneira precisa na qual um MDB é configurado depende de qual servidor de aplicativos que está sendo usado, mas as informações de configuração devem especificar a qual gerenciador de filas conectar-se, como conectar-se ao gerenciador de filas, qual destino monitorar para mensagens e o comportamento transacional do MDB. Estas informações são então usadas pelo contêiner EJB. Quando uma mensagem que satisfaz os critérios de seleção do MDB chega ao destino especificado, o contêiner EJB usa IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging para recuperar a mensagem do gerenciador de filas e, em seguida, entrega a mensagem ao MDB chamando seu método onMessage().

Relacionamento com o IBM MQ classes for Java

IBM MQ classes for Java, IBM MQ classes for Jakarta Messaging e IBM MQ classes for JMS são peers que usam uma interface comum do Java para o MQI

A Figura 55 na página 164 mostra o relacionamento entre IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging e IBM MQ classes for Java

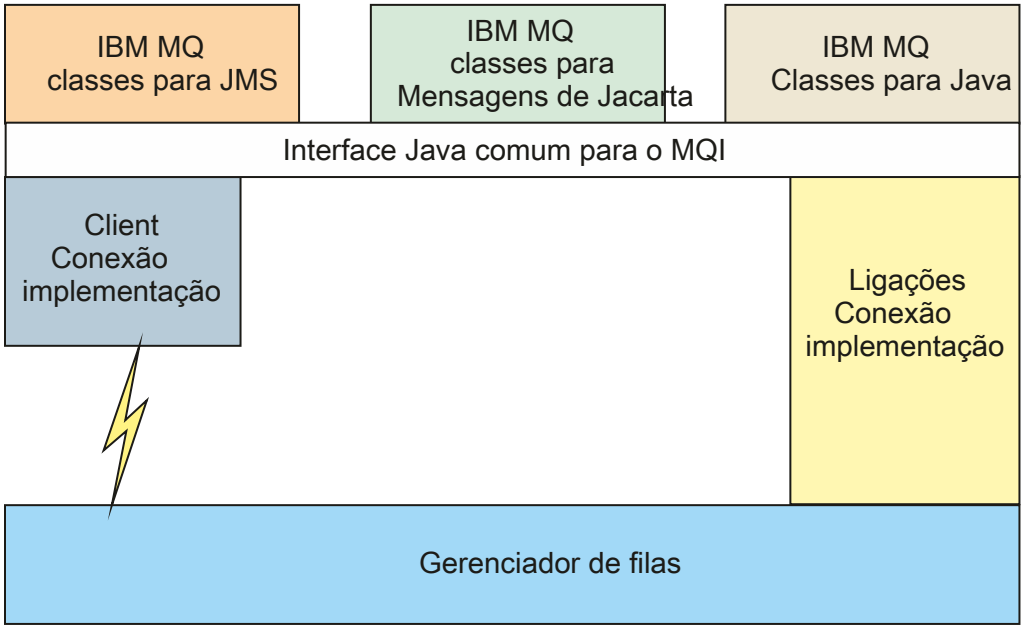


Figura 55. O relacionamento entre IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging e IBM MQ classes for Java

Em geral, os programas Java devem usar apenas uma interface para fazer a interface com IBM MQ - IBM MQ classes for Java, IBM MQ classes for Jakarta Messaging ou IBM MQ classes for JMS Interfaces de combinação não são suportadas, com uma exceção. Para manter a compatibilidade com liberações anteriores do IBM WebSphere MQ 7.0, as classes de saída do canal que são gravadas no Java ainda podem usar as interfaces do IBM MQ classes for Java, mesmo se as classes de saída do canal sejam chamadas por meio do IBM MQ classes for JMS. No entanto, usar as interfaces IBM MQ classes for Java significa que seus aplicativos ainda são dependentes de:

- JMS 2.0 O IBM MQ classes for Java arquivo JAR `com.ibm.mq.jar`. Se você não quiser `com.ibm.mq.jar` em seu caminho de classe, será possível usar o conjunto de interfaces no pacote `com.ibm.mq.exits`.

- JM 3.0 Uso do `com.ibm.mq.jakarta.client.jar`, ao interoperar com IBM MQ classes for Jakarta Messaging.

Conceitos relacionados

Por que devo usar as classes do IBM MQ para Jakarta Messaging?

Por que devo usar as classes IBM MQ para JMS?

Por que devo usar classes do IBM MQ para Java?

Provedor de sistema de mensagens do IBM MQ

O provedor de mensagens do IBM MQ possui três modos de operação: modo normal, o modo normal com restrições e modo de migração.

O provedor de sistemas de mensagens IBM MQ tem três modos de operação:

- Modo normal do provedor de mensagens IBM MQ
- Modo normal com restrições do provedor de mensagens IBM MQ
- Modo de migração do provedor de mensagens IBM MQ

O modo normal do provedor de mensagens IBM MQ usa todos os recursos de um gerenciador de filas IBM MQ para implementar JMS. Esse modo é otimizados para usar a API e a funcionalidade do JMS 2.0 JM 3.0 ou Jakarta Messaging 3.0

Se:

- O cliente especifica uma versão do provedor de 6 em um **ConnectionFactory**, o cliente se comporta de maneira compatível com o cliente fornecido com IBM WebSphere MQ 6.0. Apenas interfaces JMS 1.1 e JMS 2 são suportadas, mas algumas funcionalidades JMS 2, como assinaturas compartilhadas, atraso de entrega e envio assíncrono, estão desativadas. Não há compartilhamento de conexão..
- O cliente especifica uma versão do provedor de 7 em um **ConnectionFactory**, as interfaces JMS 1.1 e JMS 2 são totalmente suportadas.
- Nenhuma versão do provedor foi especificada, foi feita uma tentativa de conexão com o provedor versão 7. Se isso falhar, uma nova tentativa será feita com o provedor versão 6.

Se você desejar conectar-se ao IBM Integration Bus usando o IBM MQ Enterprise Transport, use o modo de migração. Se você usar o IBM MQ Real-Time Transport, o modo de migração será selecionado automaticamente, porque você selecionou explicitamente as propriedades no objeto de connection factory. A conexão com o IBM Integration Bus usando o IBM MQ Enterprise Transport segue as regras gerais para seleção de modo descritas em Configurando a propriedade JMS **PROVIDERVERSION**.

Tarefas relacionadas

Configurando recursos do 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 167
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 168
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 169
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 172
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 216
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 229
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 233
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 254
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 270
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 276
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 281
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 244
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 280
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 167 illustrates a queue manager, showing connections to different application environments, and the channel initiator.

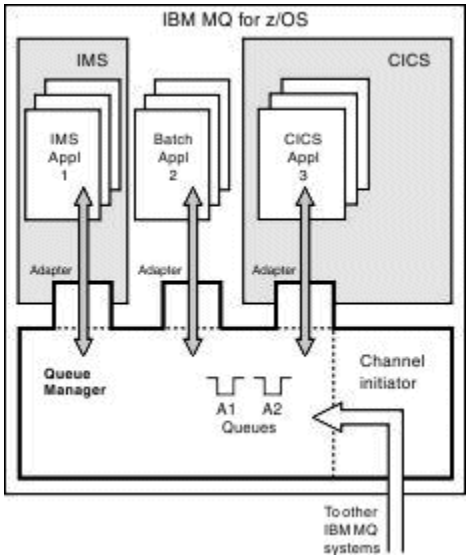


Figure 56. Overview of IBM MQ for z/OS

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 168 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.

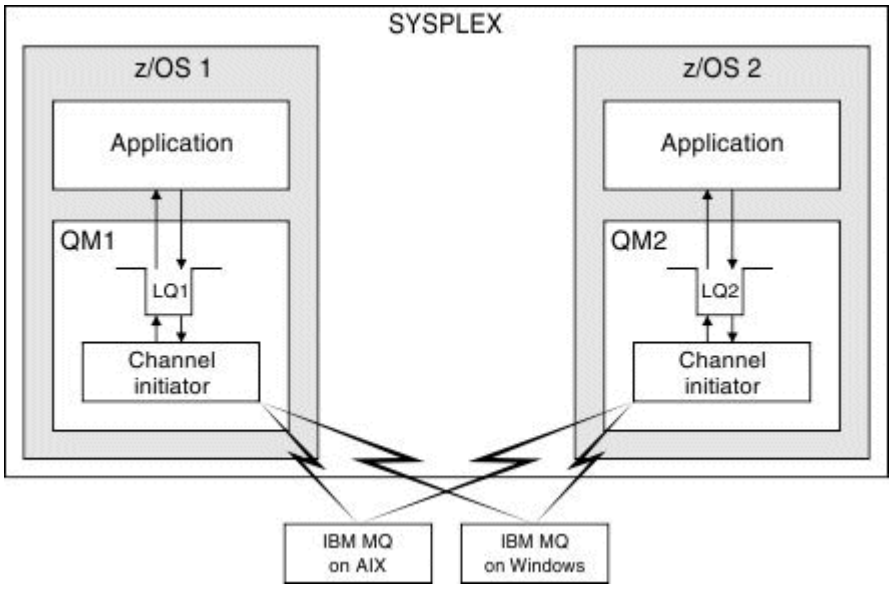


Figure 57. Communication between queue managers

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 233.

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 244.

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 254.

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 270.

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 276.

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 280.

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 283.

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 173 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.

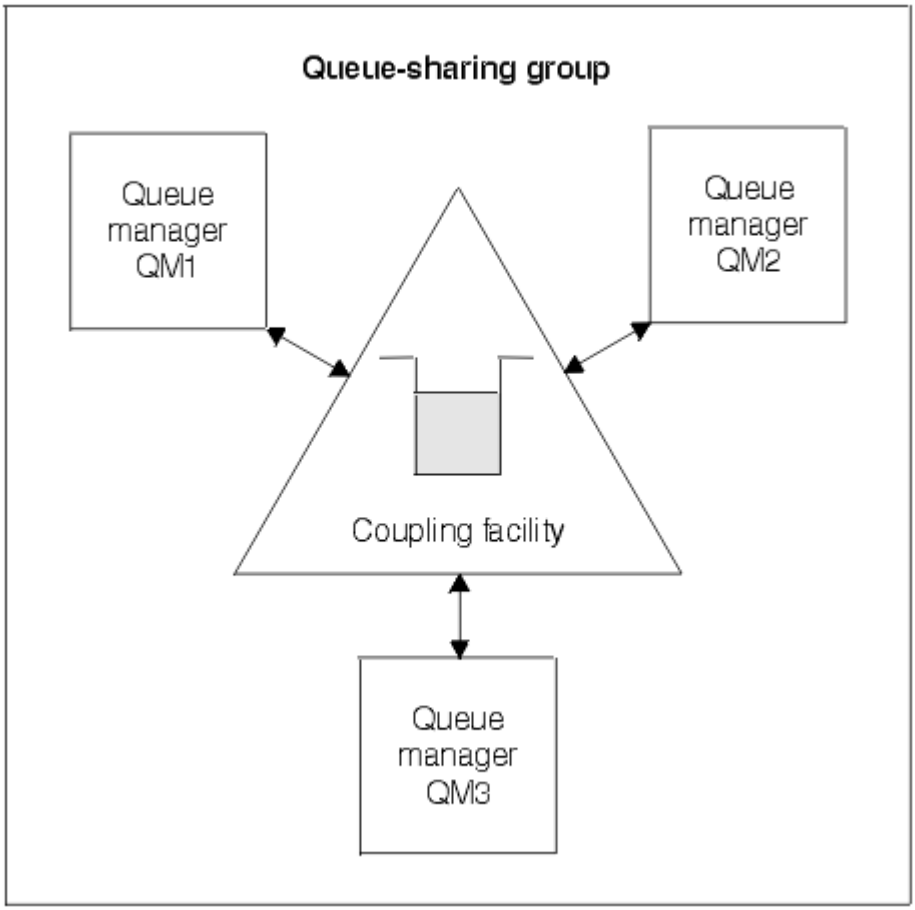


Figure 58. A queue sharing group

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 174 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.

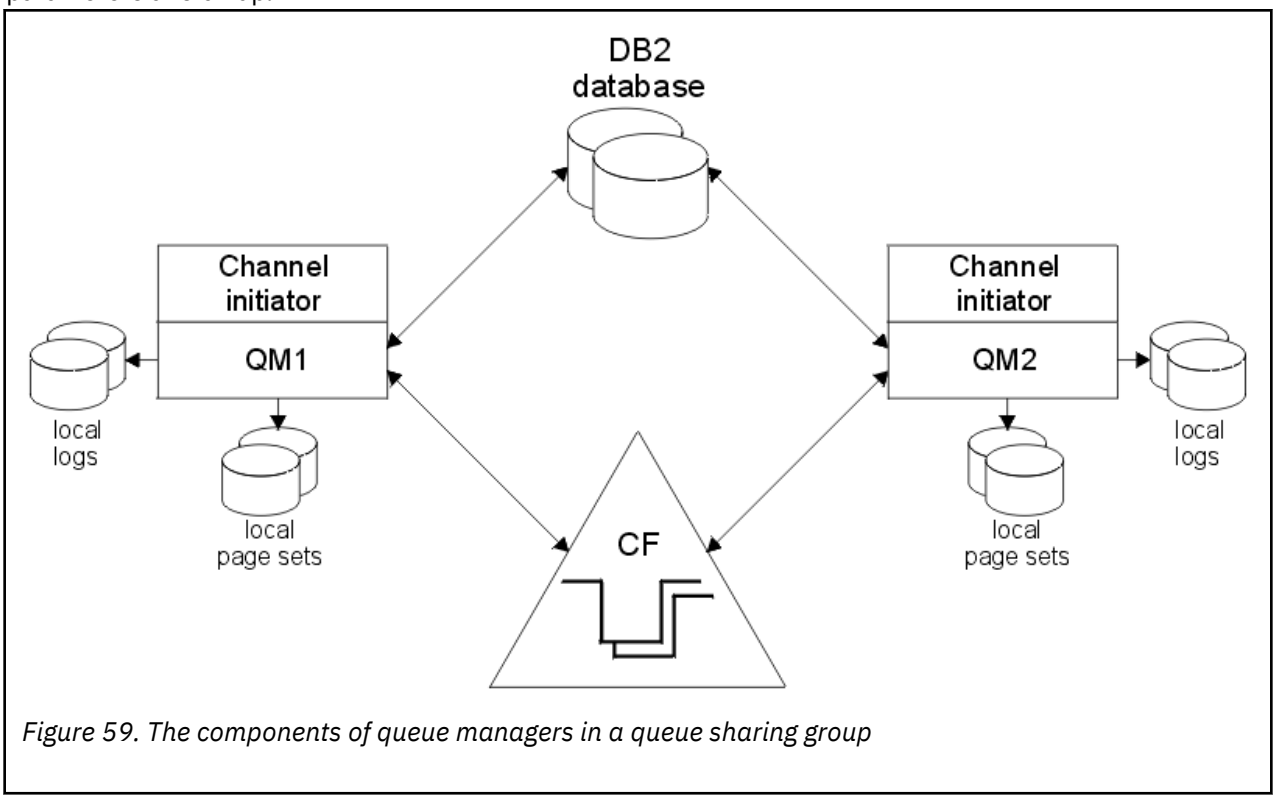


Figure 59. The components of queue managers in a queue sharing group

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 175
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 191

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 210
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 214
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 215
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 177 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 192 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 177
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 179
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 190

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 192.

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 210.

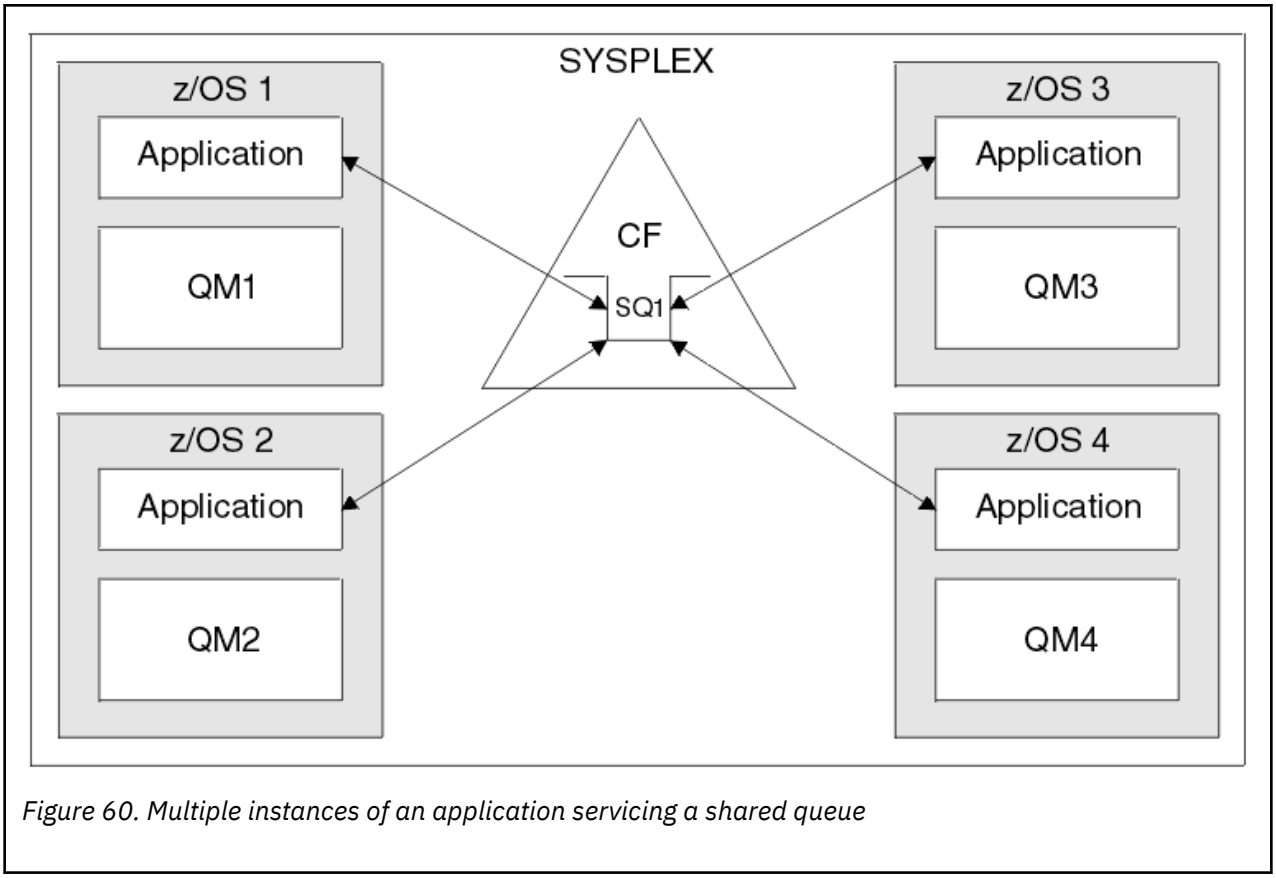


Figure 60. Multiple instances of an application servicing a shared queue

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 199
- “Improved performance - basic configuration” on page 205

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 194 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 200.

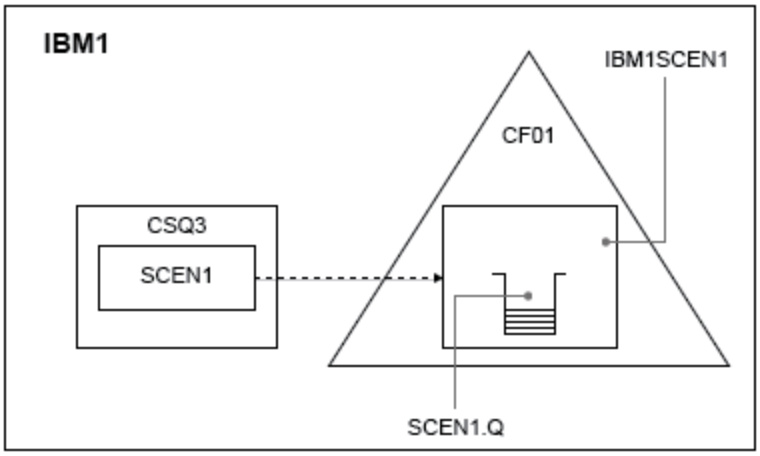


Figure 61. Basic configuration

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 192

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 199. 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 202.

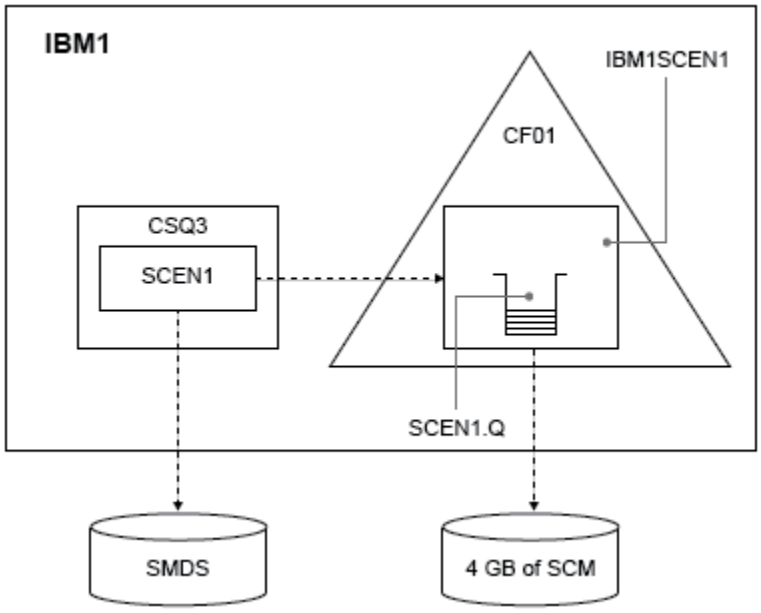


Figure 62. Configuration adding SMDS and SCM for emergency storage

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 204 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 206.

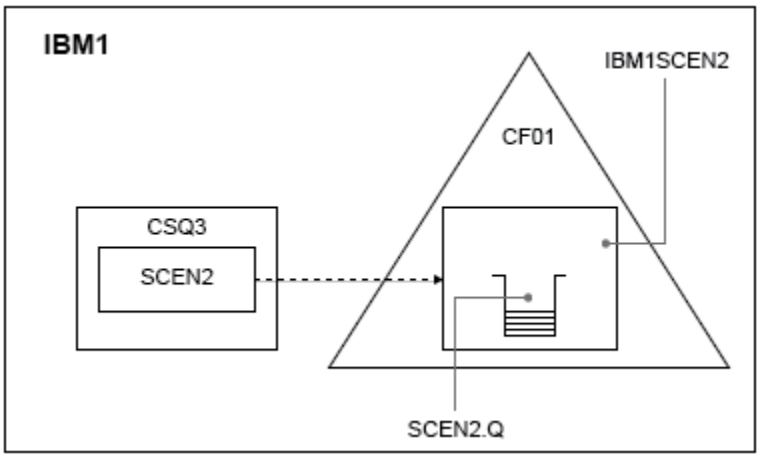


Figure 63. Basic configuration

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 208.

Related concepts

“Use of storage class memory with shared queues” on page 192

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 205. The scenario describes the addition of SCM to the initial structure.

This final configuration is illustrated in Figure 64 on page 209.

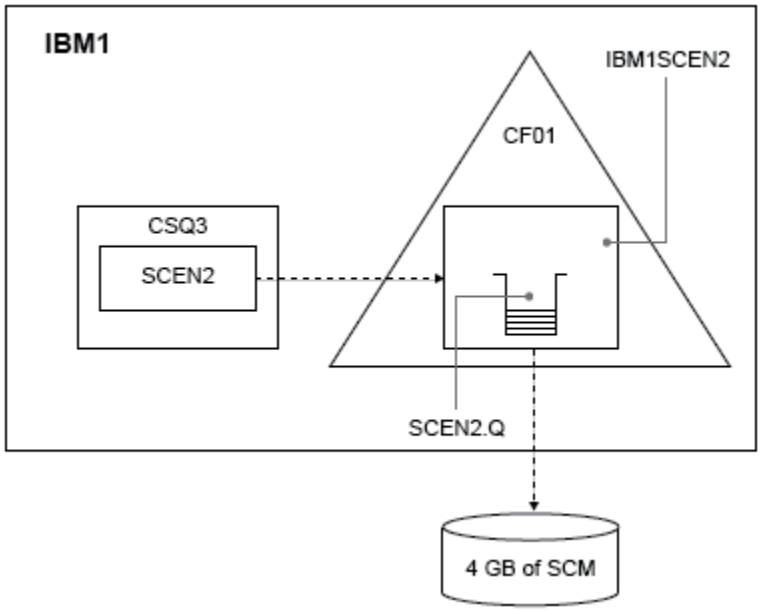


Figure 64. Configuration adding SCM for improved performance

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 205:
  - 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 205.

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 211 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.

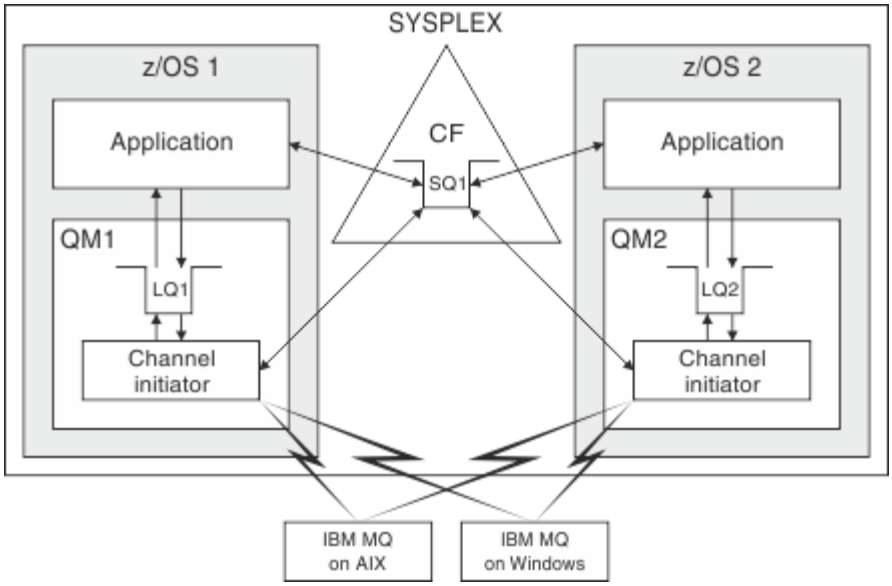


Figure 65. Distributed queuing and queue sharing groups

Related concepts

"Shared channels" on page 211
Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

"Intra-group queuing" on page 216
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 213
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 214 shows how members of a cluster can access a shared queue through any member of the queue sharing group.

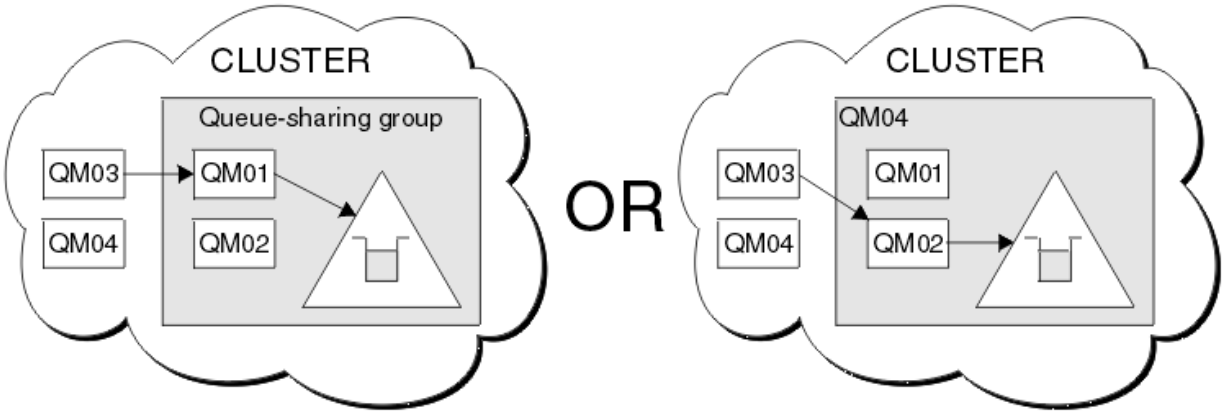


Figure 66. A queue sharing group as part of a cluster

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 254
Queue sharing group security	“Security concepts in IBM MQ for z/OS” on page 270
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 172
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 172.

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.

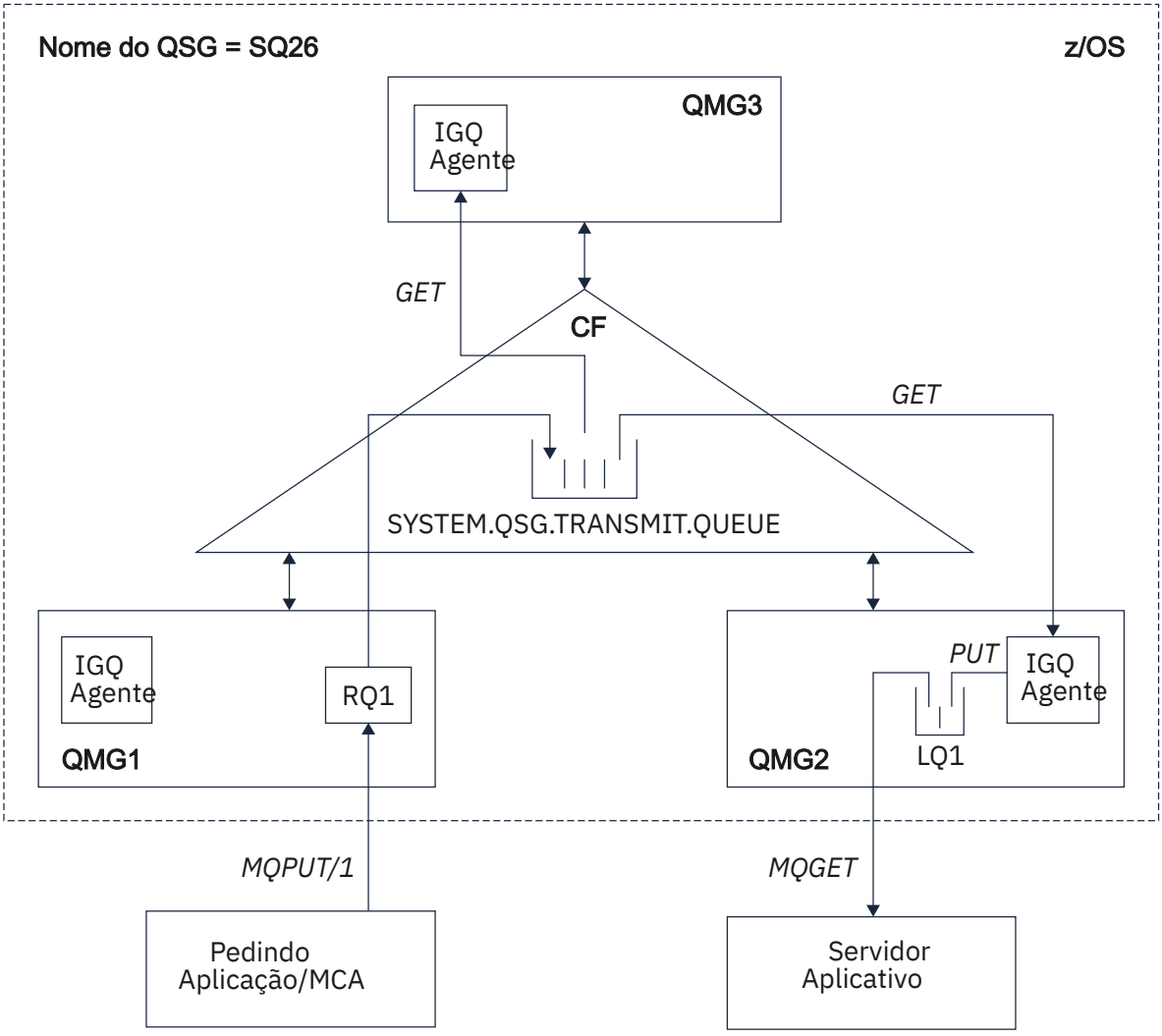


Figure 67. An example of intra-group queuing

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.

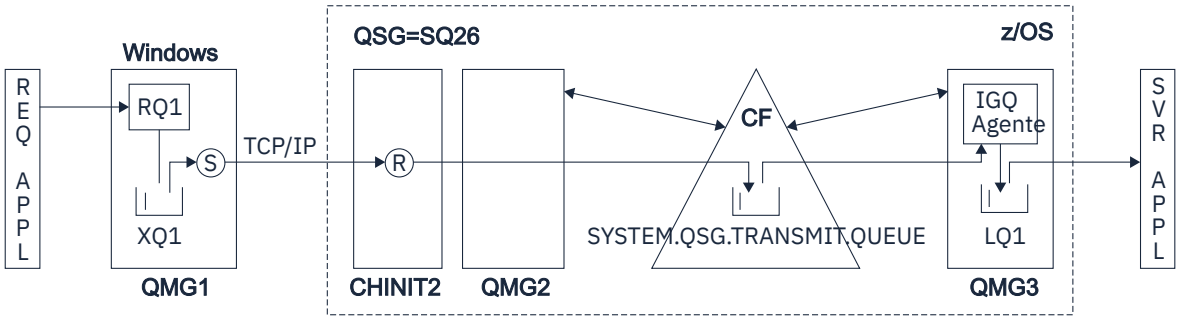


Figure 68. An example of migration support

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 228 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 220 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 228 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 220 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 216 describes the typical configuration.

Related concepts

“Distributed queuing with intra-group queuing (multiple delivery paths)” on page 221
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 223
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 225
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.

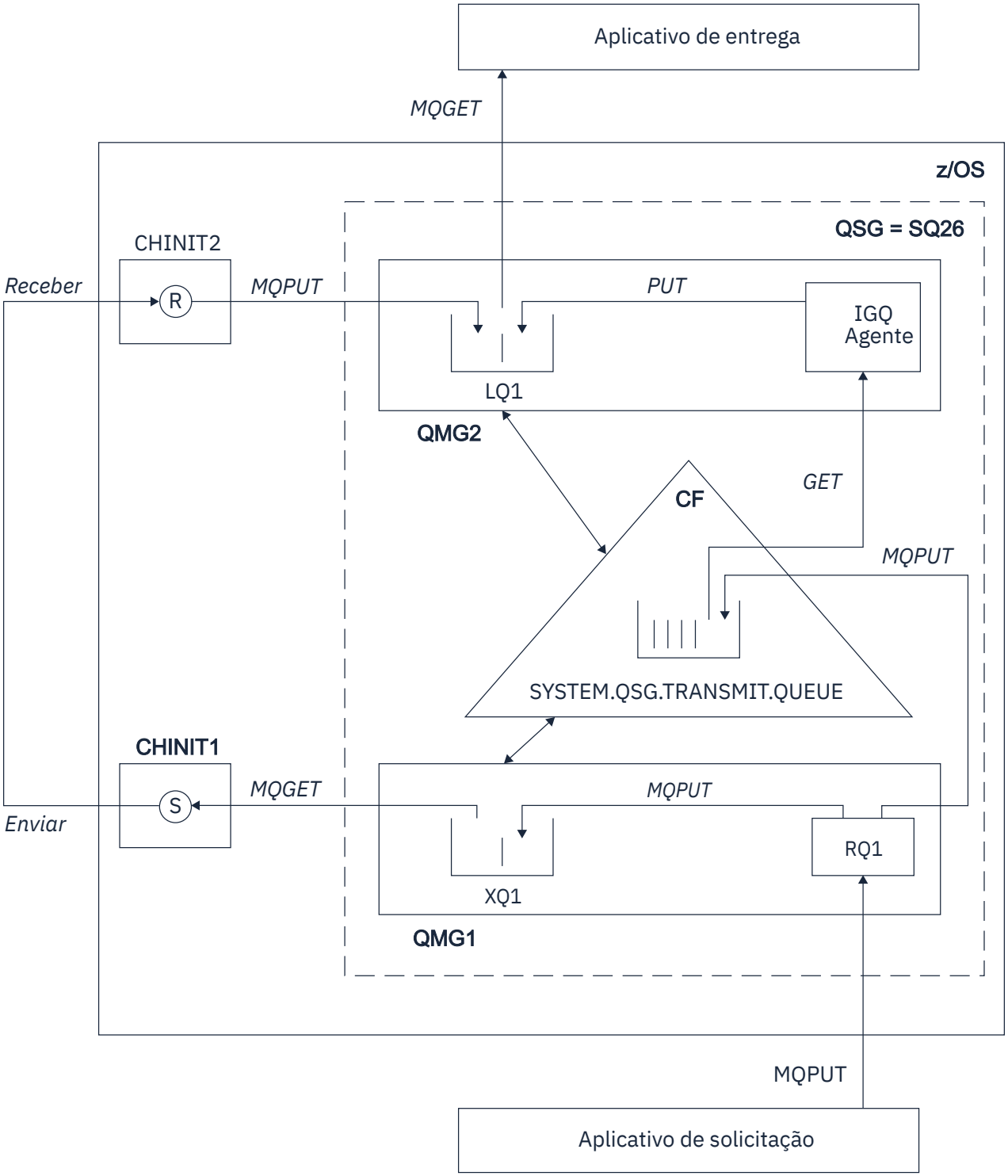


Figure 69. An example configuration

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).

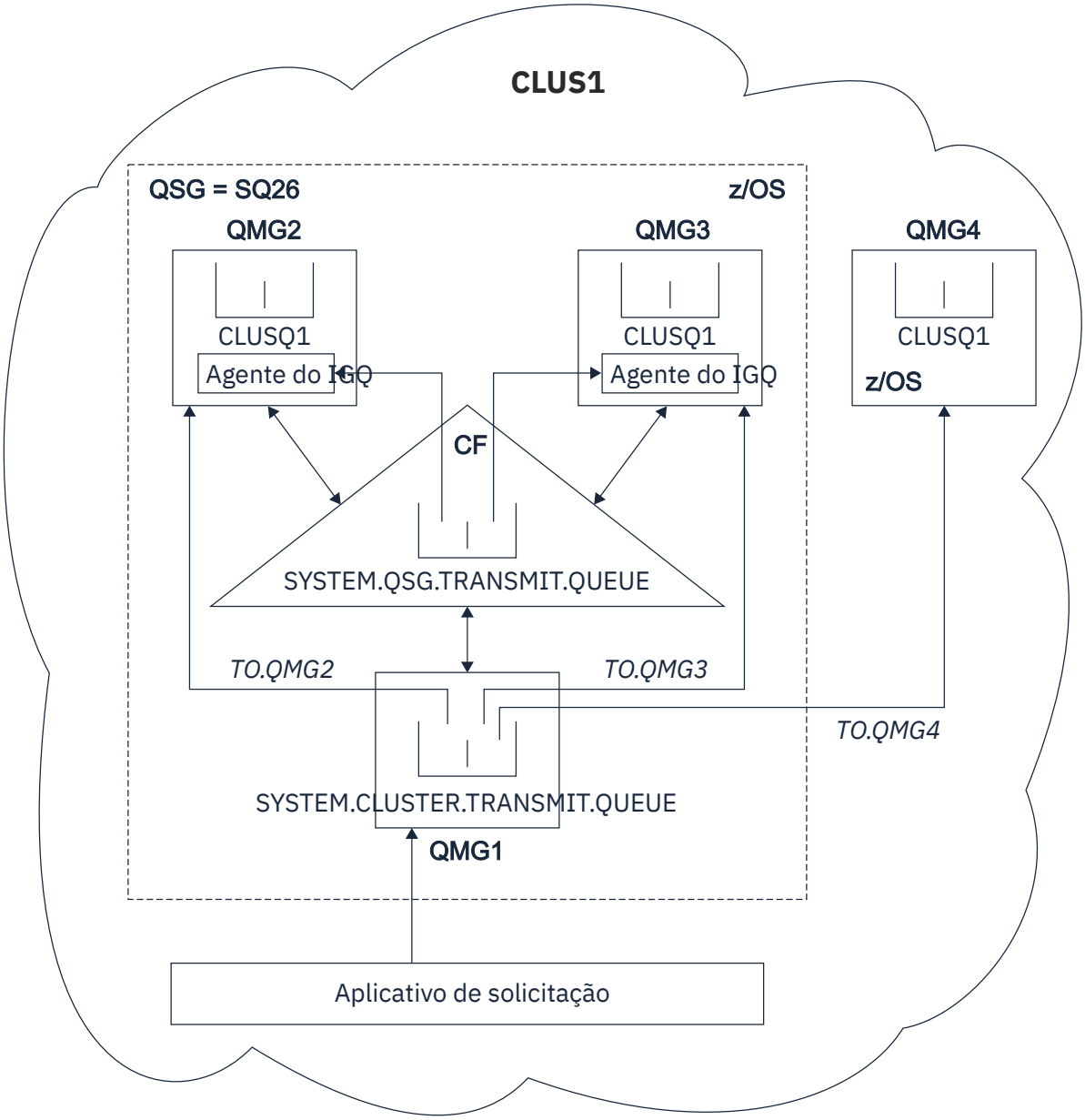


Figure 70. An example of clustering with intra-group queuing

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 221.

Clustering with intra-group queuing is described in “Clustering with intra-group queuing (multiple delivery paths)” on page 223.

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 226
This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

“Report messages - Intra Group Queuing” on page 227
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 228.
- 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 228.
- 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 229
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 230
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 232
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 233
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 172, 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 285).

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 231, 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.

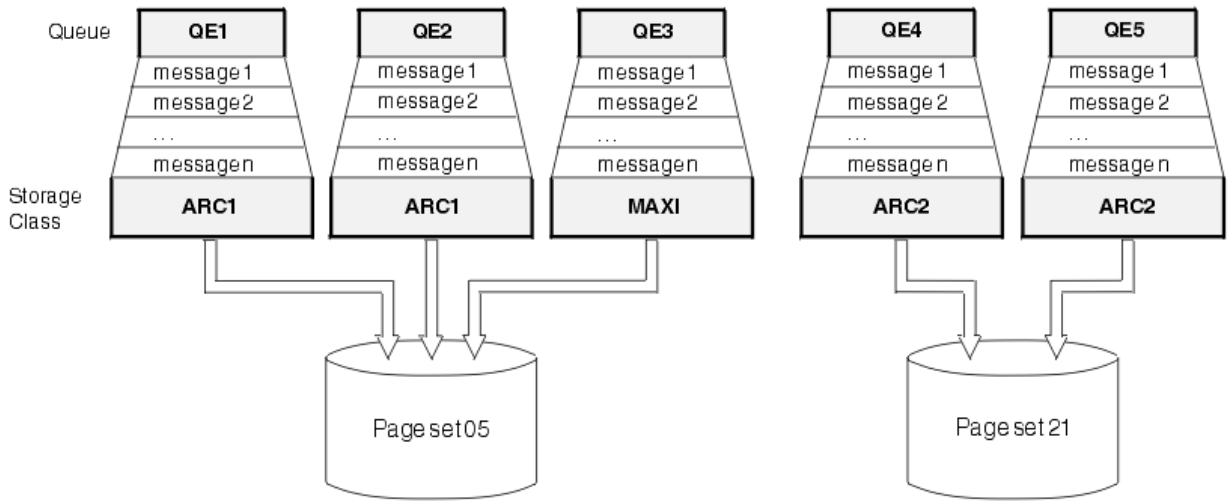


Figure 71. Mapping queues to page sets through storage classes

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 232, 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 232 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.

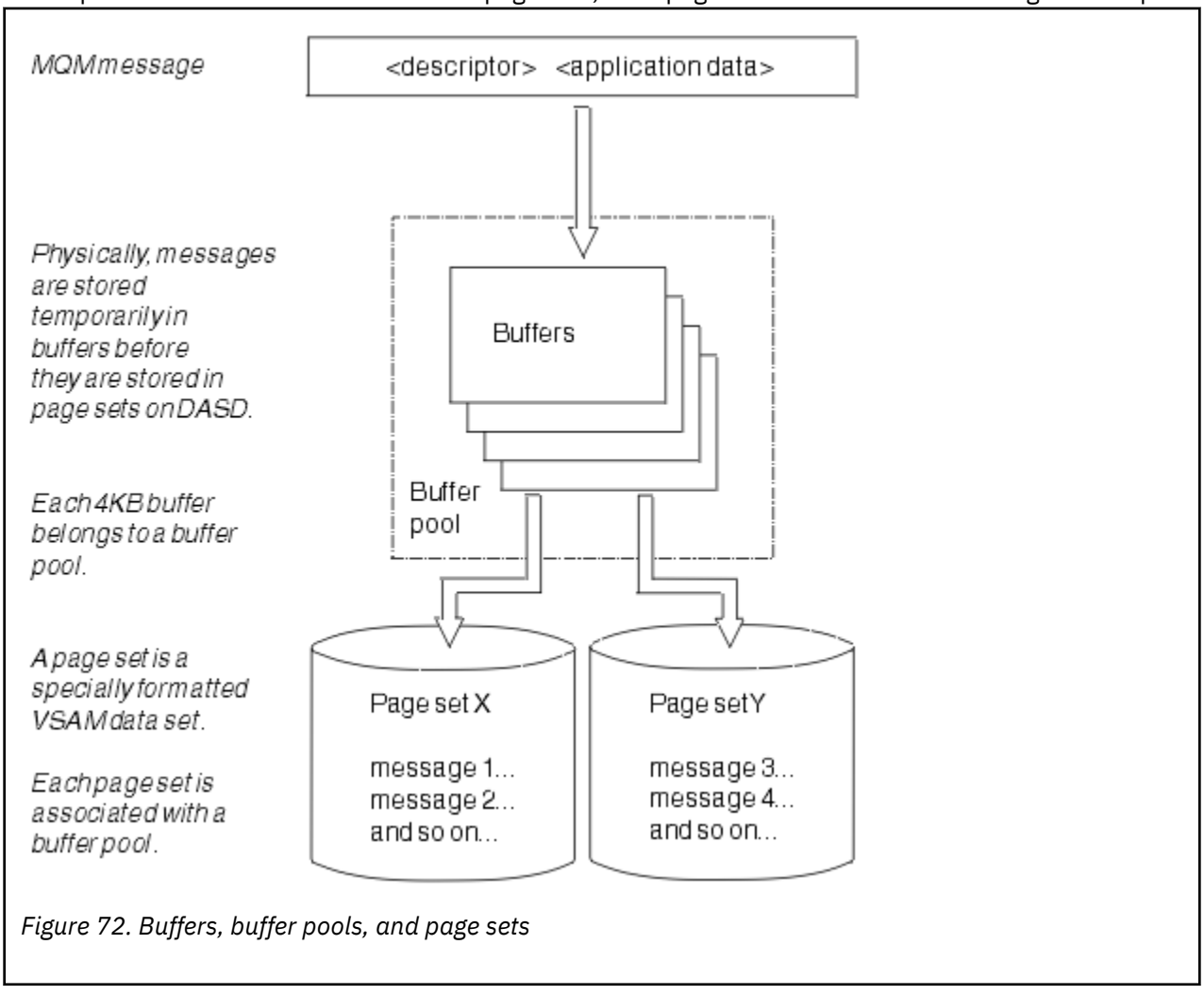


Figure 72. Buffers, buffer pools, and page sets

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 234
- “How the log is structured” on page 237
- “How the IBM MQ for z/OS logs are written” on page 238
- “Larger log Relative Byte Address” on page 241
- “The bootstrap data set” on page 242

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 239 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.

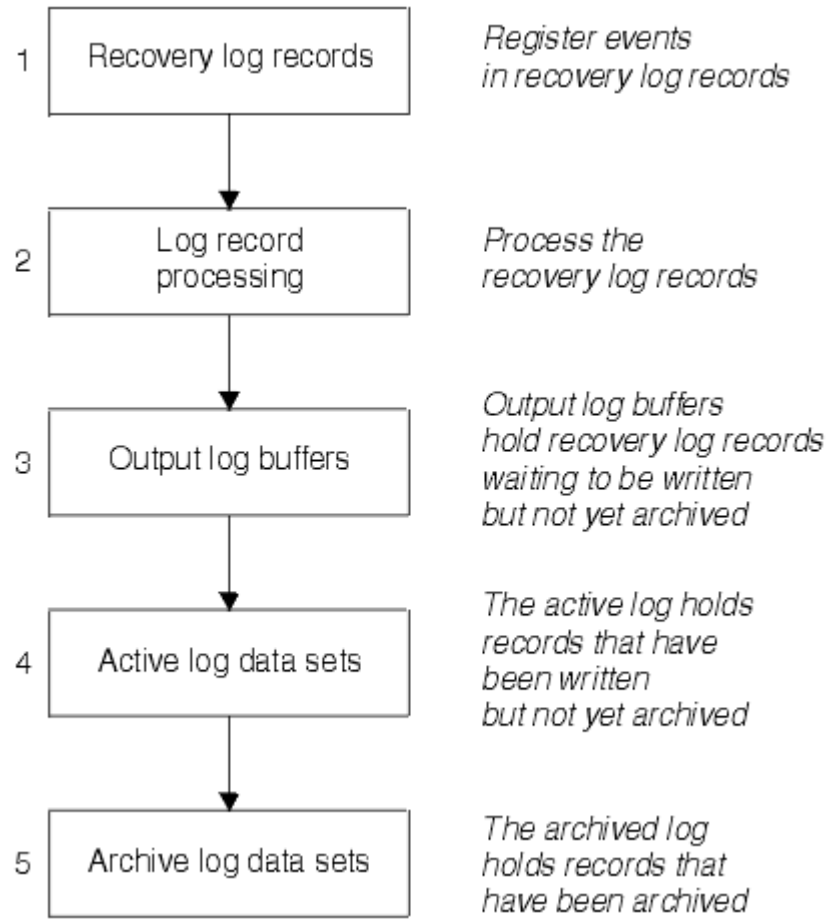


Figure 73. The logging process

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 240
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 240.

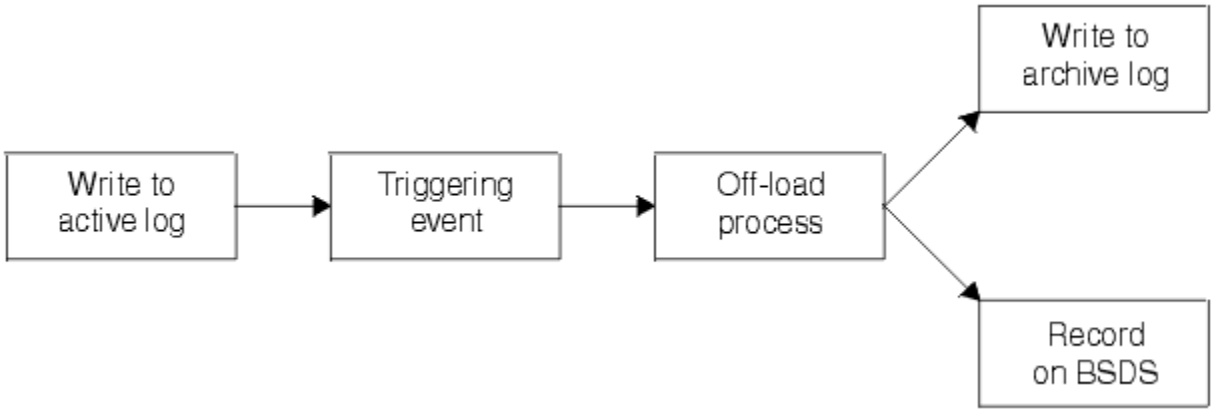


Figure 74. The offloading process

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 Administrando 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 Administrando 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 241 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 244
- “Tuning your queue manager on IBM MQ for z/OS” on page 249
- “Sample definitions supplied with IBM MQ for z/OS” on page 250

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 248

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 para z/OS Planejamento e ajuste de capacidade 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 255
- “How consistency is maintained in IBM MQ for z/OS” on page 256
- “What happens during termination in IBM MQ for z/OS” on page 258
- “What happens during restart and recovery in IBM MQ for z/OS” on page 259
- “How in-doubt units of recovery are resolved” on page 261
- “Shared queue recovery” on page 264

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.

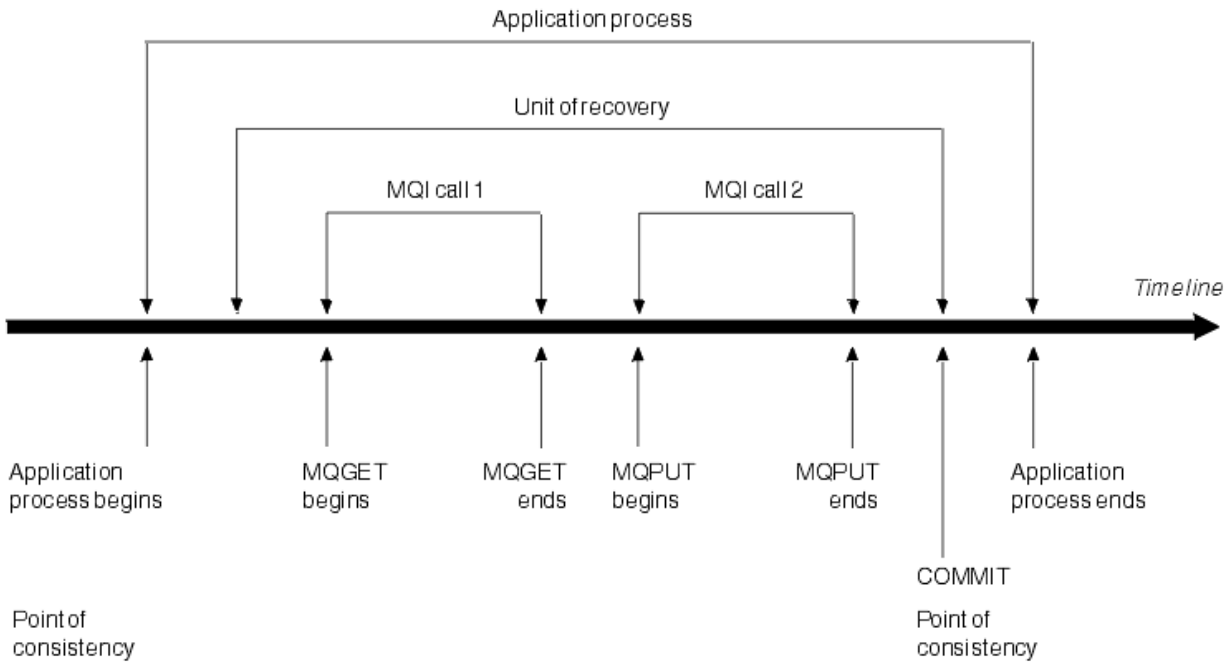


Figure 75. A unit of recovery within an application program

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 255 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 256.

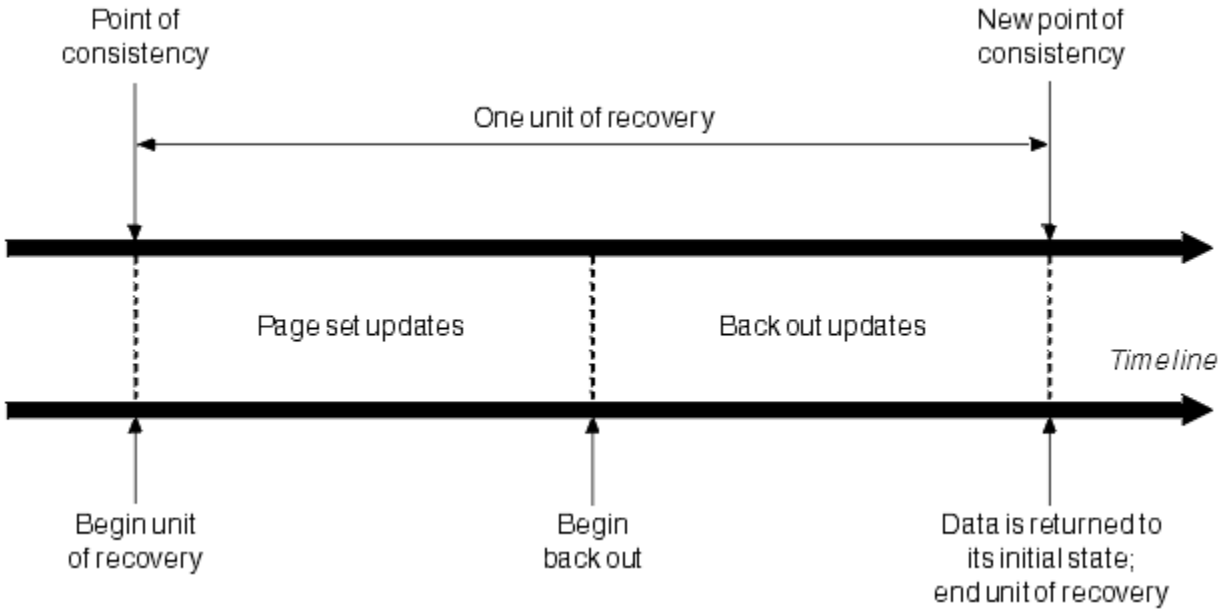


Figure 76. A unit of recovery showing back out

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 257 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.

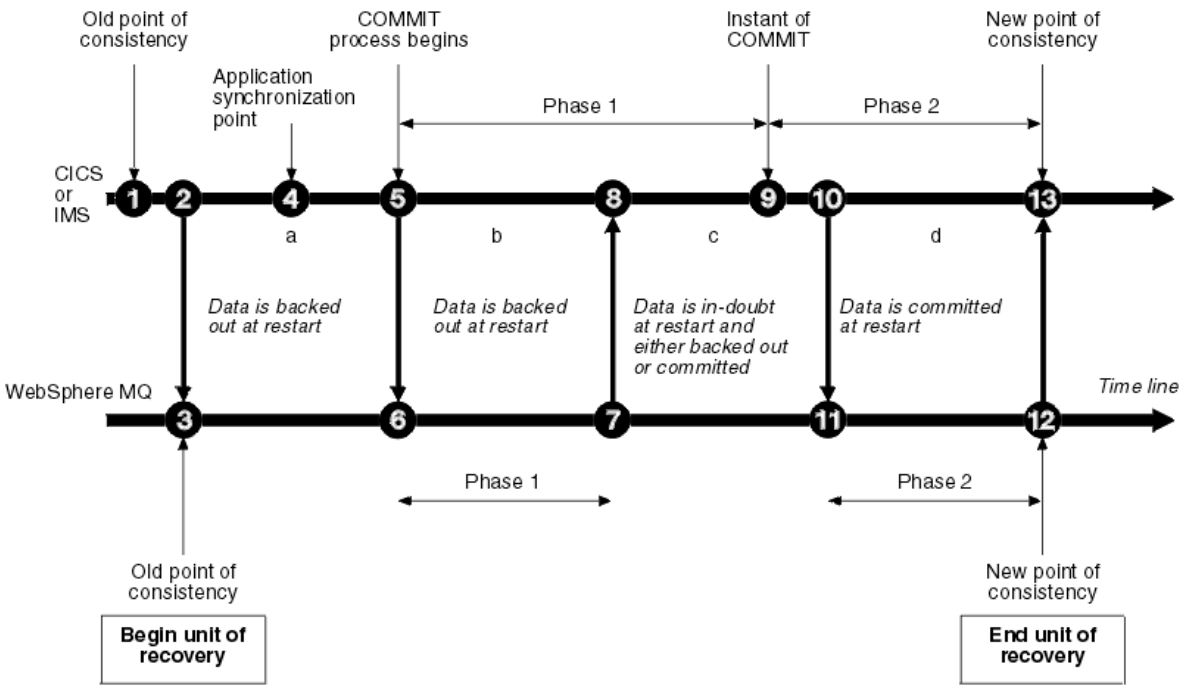


Figure 77. The two-phase commit process

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 257 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 259.

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 261).

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 mensagens, conclusão e códigos de razão do IBM MQ for z/OS 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 Administrando 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 Administrando 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 mensagens, conclusão e códigos de razão do IBM MQ for z/OS 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 Administrando 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 Referência de Comandos do Sistema z/OS MVS.

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 274.

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 275
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 191.

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 264.

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 278.

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 211.

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 172
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 280
- "IBM MQ trace" on page 280
- "Events" on page 280

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 281
A queue sharing group can configure and enable support for GROUP units of recovery.

“Application support” on page 282
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 281 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 282.

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 283
Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

“CICS applications” on page 283
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 Servidor de Comunicações and z/OS Habilidades Básicas: endereçamento virtual dinâmico 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 284
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 290
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 285
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 288

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 284
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 281

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 286.

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 287.

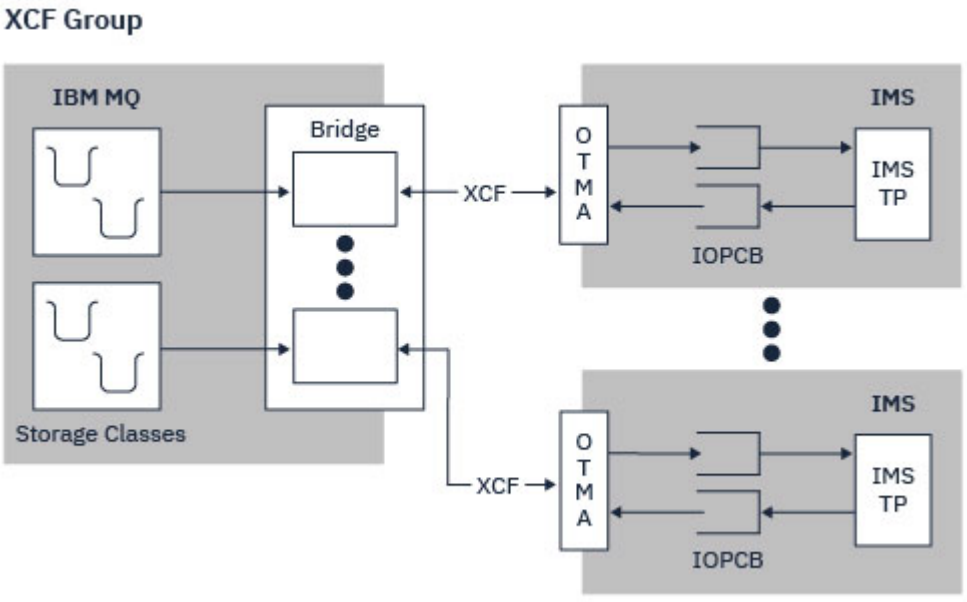


Figure 78. The IBM MQ - IMS bridge

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 285

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.

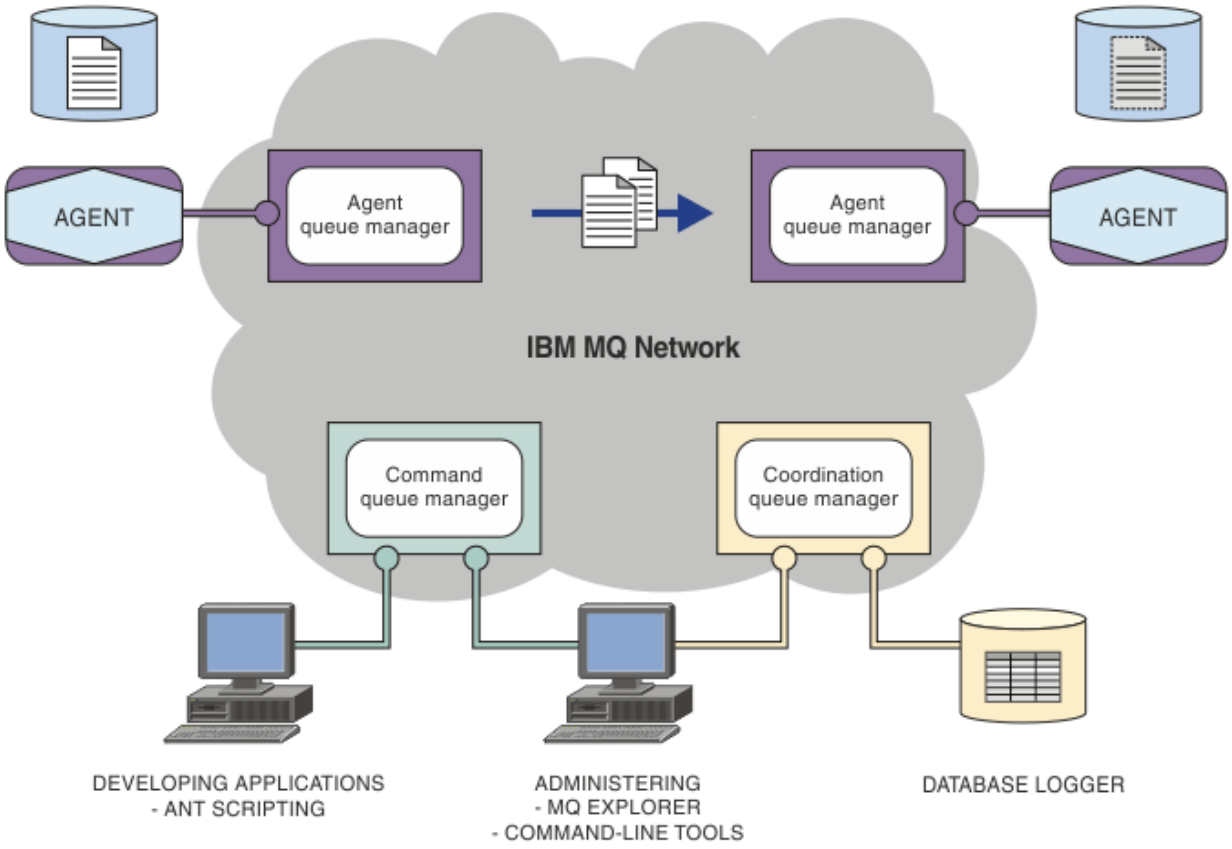
Using IBM MQ functions from JMS applications

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

O Managed File Transfer transfere arquivos entre sistemas de maneira gerenciada e auditável, independentemente do tamanho do arquivo ou do sistema operacional usado.

Você pode usar o Managed File Transfer para construir uma solução personalizada, escalável e automatizada que permite a você gerenciar, confiar e proteger transferências de arquivos. O Managed File Transfer elimina a redundância de custo, reduz os custos de manutenção e maximiza seus investimentos existentes de TI.



O diagrama mostra uma topologia simples do Managed File Transfer. Há dois agentes, cada um conecta-se a seu próprio gerenciador de filas do agente em uma rede do IBM MQ. Um arquivo é transferido do agente em um lado do diagrama, por meio da rede do IBM MQ, para o agente no outro lado do diagrama. Além disso, na rede do IBM MQ, estão o gerenciador de filas de coordenação e um gerenciador de filas de comando. Os aplicativos e as ferramentas se conectam a esses gerenciadores de filas para configurar, administrar, operar e registrar a atividade do Managed File Transfer na rede do IBM MQ.

O Managed File Transfer pode ser instalado como quatro opções diferentes, dependendo do seu sistema operacional e da configuração geral. Essas opções são Managed File Transfer Agent, Managed File Transfer Logger, Managed File Transfer Service ou Managed File Transfer Tools. Para obter mais informações, consulte Opções do produto Managed File Transfer.

Você pode usar o Managed File Transfer para executar as seguintes tarefas:

- Criar transferências de arquivos gerenciados
  - Windows Linux Criar novas transferências de arquivos do IBM MQ Explorer nas plataformas Linux ou Windows.
  - Criar novas transferências de arquivos da linha de comandos em todas as plataformas suportadas.
  - Integrar a função de transferência de arquivos à ferramenta do Apache Ant.
  - Gravar aplicativos que controlam o Managed File Transfer colocando mensagens nas filas de comando do agente.
  - Programar transferência de arquivos para ocorrer em momento posterior. Também é possível acionar as transferências de arquivos planejadas com base em um intervalo de eventos do sistema de arquivos, por exemplo, um novo arquivo que está sendo criado.
  - Monitorar continuamente um recurso, por exemplo, um diretórioe iniciar uma tarefa quando o conteúdo desse recurso atender alguma condição predefinida. Essa tarefa pode ser uma transferência de arquivos, um script do Ant ou uma tarefa de JCL.

  - Transferir arquivos para e de filas do IBM MQ.
  - Transferir arquivos para e a partir de servidores FTP, FTPS ou SFTP.
  - Transferir arquivos para e de nós Connect:Direct.
  - Transferir ambos, arquivos de texto e binários. Os arquivos texto são convertidos automaticamente entre páginas de código e convenções de final-de-linha dos sistemas de origem e destino.
  - As transferências podem ser protegidas, usando os padrões de mercado para conexões com base em Secure Socket Layer (SSL).
- Visualizar transferências em andamento e informação de registro sobre todas as transferências em sua rede.
  - Windows Linux Visualizar o status de transferências em andamento do IBM MQ Explorer em plataformas Linux ou Windows.
  - Windows Linux Verificar o status das transferências concluídas usando o IBM MQ Explorer nas plataformas Linux ou Windows.
  - Usar o recurso de criador de logs de banco de dados do Managed File Transfer para salvar mensagens de log para um banco de dados Db2 ou Oracle.

O Managed File Transfer é construído em IBM MQ, o que oferece entrega segura e única de mensagens entre aplicativos. Você pode tirar vantagem de vários recursos do IBM MQ. Por exemplo, você pode usar a compactação de canal para compactar os dados que envia entre agentes via canais do IBM MQ e usar os canais SSL para proteger os dados que você envia entre agentes. Os arquivos são transferidos de modo confiável e podem tolerar a falha da infra-estrutura por meio da qual a transferência será executada. Se tiver uma indisponibilidade da rede, a transferência de arquivos reinicia de onde ela parou, quando a conectividade é restabelecida.

Por consolidar a transferência de arquivos com sua rede IBM MQ existente, você pode evitar o gasto de recursos necessários para manter as duas infraestruturas separadas. Se você ainda não for um cliente do IBM MQ, ao criar uma rede do IBM MQ para suportar o Managed File Transfer, você construirá o backbone de uma implementação futura do SOA. Se você já for um cliente do IBM MQ, o Managed File Transfer poderá aproveitar sua infraestrutura existente do IBM MQ, incluindo IBM MQ Internet Pass-Thru e IBM Integration Bus.

É possível aproveitar as soluções de alta disponibilidade do IBM MQ para melhorar a resiliência de sua configuração do Managed File Transfer Se seus agentes usarem gerenciadores de filas de dados replicados (RDQMs), eles deverão ser configurados para usar o recurso de endereço IP flutuante. Isso significa que os agentes usam o mesmo endereço IP para se comunicar com qualquer uma das três instâncias do RDQM atualmente em execução e se reconectam automaticamente no failover (consulte Alta disponibilidade do RDQM e Criando e excluindo um endereço IP flutuante). Se você usar a solução do gerenciador de filas de várias instâncias, os aplicativos usarão um endereço IP diferente para se comunicar com cada instância, que é manipulada pela reconexão do cliente no failover (consulte Gerenciadores de filas de várias instâncias e Canal e reconexão do cliente).

O Managed File Transfer integra-se com vários outros produtos IBM:

IBM Integration Bus

Arquivos de processo que foram transferidos pelo Managed File Transfer como parte de um fluxo do IBM Integration Bus. Para obter mais informações, consulte Trabalhando com MFT por meio do IBM Integration Bus.

IBM Sterling Connect:Direct

Transferir arquivos para e de uma rede Connect:Direct existente utilizando a ponte Managed File Transfer Connect:Direct. Para obter mais informações, consulte A ponte Connect:Direct.

IBM Tivoli Composite Application Manager

O IBM Tivoli Composite Application Manager fornece um agente que você pode usar para monitorar informações que são publicadas para o gerenciador de filas de coordenação.

Conceitos relacionados

Opções do produto Managed File Transfer

“Visão Geral da Topologia do MFT” na página 293
Uma visão geral de como os agentes do Managed File Transfer são conectados ao gerenciador de filas de coordenação em uma rede do IBM MQ.

“Como o MFT trabalha com o IBM MQ?” na página 293
O Managed File Transfer interage de várias maneiras com o IBM MQ.

Como o MFT trabalha com o IBM MQ?

O Managed File Transfer interage de várias maneiras com o IBM MQ.

- O Managed File Transfer transfere arquivos entre os processos do agente, dividindo cada arquivo em uma ou mais mensagens e transmitindo as mensagens por meio da rede do IBM MQ.
- Os processos do agente movem os dados do arquivo usando mensagens não persistentes para minimizar o impacto sobre os logs do IBM MQ. Por se comunicarem entre si, os processos do agente regulam o fluxo de mensagens quem contêm dados de arquivo. Isso evita que mensagens contendo dados do arquivo se acumulem em filas de transmissão do IBM MQ e assegura que, se algumas das mensagens não persistentes não forem entregues, os dados do arquivo sejam enviados novamente.
- Os agentes do Managed File Transfer usam várias filas do IBM MQ. Para obter mais informações, consulte as MFTfilas do sistema e o tópico do sistema.
- Embora algumas destas filas sejam estritamente para uso interno, um agente pode aceitar trabalho na forma de mensagens de comando especialmente formatadas, enviadas para uma fila específica da qual o agente pode ler. Os comandos da linha de comandos e o plug-in do IBM MQ Explorer enviam mensagens do IBM MQ para o agente para instruí-lo a executar a ação desejada. É possível gravar aplicativos IBM MQ que interagem com o agente dessa maneira. Para obter mais informações, consulte Controlando o MFT ao colocar mensagens na fila de comandos do agente.
- Os agentes do Managed File Transfer enviam informações sobre seu estado e o progresso e resultado de transferências para um gerenciador de filas do MQ que foi designado como o gerenciador de filas de coordenação. Essas informações são publicadas pelo gerenciador de filas de coordenação e podem ser assinadas pelos aplicativos que desejam monitorar o progresso da transferência ou manter registros das transferências ocorridas. Os comandos da linha de comandos e o plug-in do IBM MQ Explorer podem usar as informações publicadas. É possível gravar aplicativos do IBM MQ que usam essas informações. Para obter mais informações sobre o tópico para o qual as informações são publicadas, consulte SYSTEM.FTE tópicodo FTE
- Os componentes chave do Managed File Transfer tiram vantagem do recurso de gerenciadores de filas do IBM MQ para armazenar e encaminhar mensagens. Isso significa que, em face de uma interrupção, as partes não afetadas da infra-estrutura podem continuar a transferir arquivos. Isso estende o gerenciador de filas de coordenação, em que uma combinação de armazenamento e encaminhamento e assinaturas duráveis permitem que o gerenciador de filas de coordenação seja capaz de ficar indisponível sem perder as principais informações sobre as transferências de arquivos realizadas.

Visão Geral da Topologia do MFT

Uma visão geral de como os agentes do Managed File Transfer são conectados ao gerenciador de filas de coordenação em uma rede do IBM MQ.

Os agentes do Managed File Transfer enviam e recebem os arquivos que são transferidos. Cada agente possui seu próprio conjunto de filas em seu gerenciador de filas associado e o agente conecta-se a esse gerenciador no modo de ligações ou cliente. Um agente também pode utilizar o gerenciador de filas de coordenação como seu gerenciador de filas.

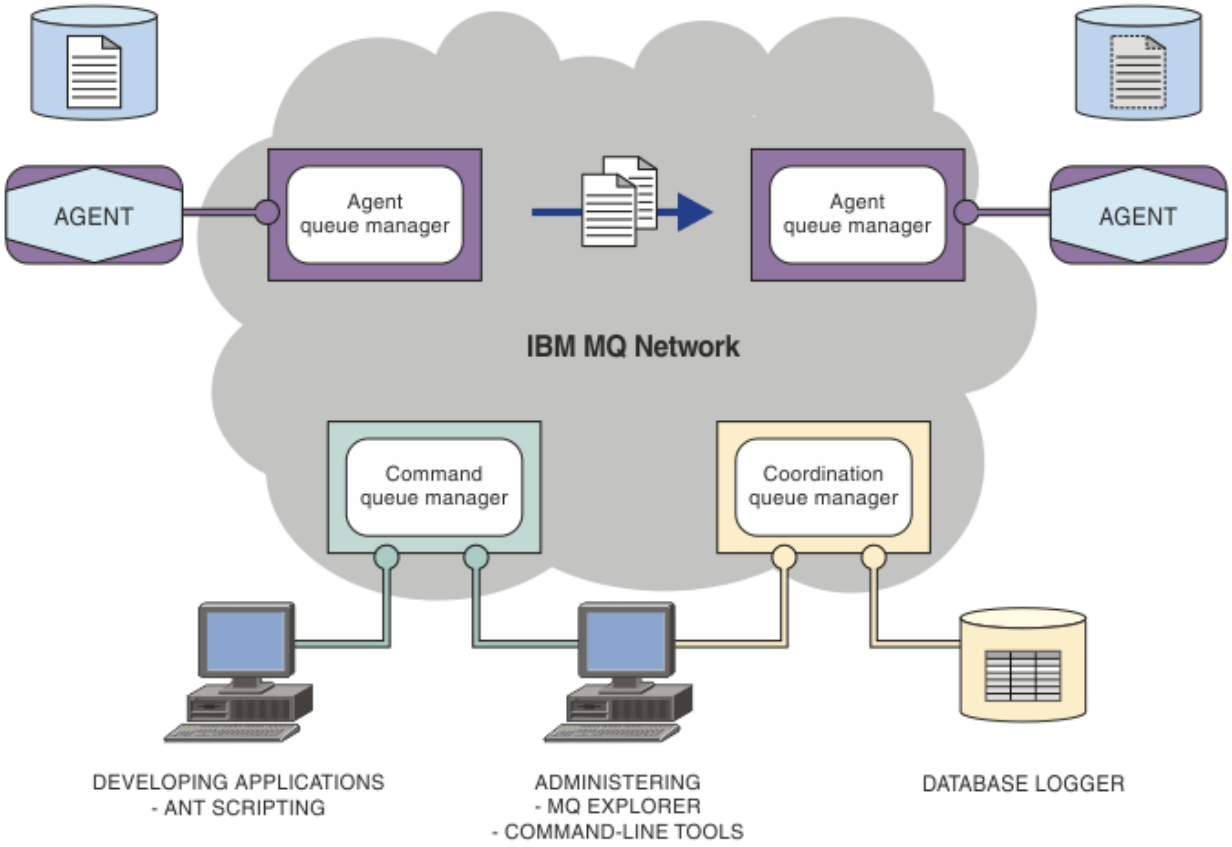
O gerenciador de filas de coordenação divulga informações de auditoria e transferência de arquivos. O gerenciador de filas de coordenação representa um único ponto para a coleta de informações do agente, de status de transferência e de auditoria de transferência. O gerenciador de filas de coordenação não precisa estar disponível para que as transferências ocorram. Se o gerenciador de filas de coordenação ficar temporariamente indisponível, as transferências continuarão normalmente. Mensagens de auditoria

e de status são armazenadas no gerenciador de filas do agente até que o gerenciador de filas de coordenação fique disponível e pode, então, ser processadas normalmente.

Os agentes registram-se com o gerenciador de filas de coordenação e publicam seus detalhes para esse gerenciador de filas. Estas informações do agente são usadas pelo plug-in do Managed File Transfer para ativar o início das transferências do IBM MQ Explorer. As informações do agente coletadas no gerenciador de filas de coordenação também são usadas pelos comandos para exibir as informações do agente e o status do agente.

As informações de auditoria de transferência e de status de transferência são publicadas no gerenciador de filas de coordenação. As informações do status de transferência e da auditoria de transferência são usadas pelo plug-in do Managed File Transfer para monitorar o progresso das transferências do IBM MQ Explorer. As informações de auditoria de transferência armazenadas no gerenciador de filas de coordenação podem ficar retidas para fornecer capacidade de auditoria.

O gerenciador de filas de comando é usado para se conectar à rede do IBM MQ e é o gerenciador de filas ao qual você se conecta ao emitir comandos do Managed File Transfer.



Conceitos relacionados

“Managed File Transfer” na página 290
O Managed File Transfer transfere arquivos entre sistemas de maneira gerenciada e auditável, independentemente do tamanho do arquivo ou do sistema operacional usado.

“Como o MFT trabalha com o IBM MQ?” na página 293
O Managed File Transfer interage de várias maneiras com o IBM MQ.

Cenário do Managed File Transfer

Visão Geral do MFT REST API

A REST API suporta determinados comandos do Managed File Transfer, incluindo a listagem de transferências e os detalhes sobre os agentes de transferência de arquivos.

A REST API inclui opções para listar todas as transferências atuais do Managed File Transfer e para consultar os status de agentes Managed File Transfer. Para obter informações adicionais, consulte Introdução ao REST API MFT.

IBM MQ Internet Pass-Thru

O IBM MQ Internet Pass-Thru (MQIPT) é um componente opcional do IBM MQ que pode ser usado para implementar soluções do sistema de mensagens entre sites remotos através da Internet.

Para obter os arquivos de instalação do MQIPT para o IBM MQ 9.4.x, acesse o IBM Fix Central for IBM MQ.

É possível usar o MQIPT para conectar qualquer versão suportada do IBM MQ. Não é necessário instalar nenhum outro componente do IBM MQ na mesma versão que o MQIPT

Se você comprou a autorização do IBM MQ, será possível instalar quantas cópias forem necessárias do MQIPT. As instalações do MQIPT não são contadas com relação à sua autorização adquirida do IBM MQ. Para obter mais informações sobre o licenciamento do IBM MQ , consulte IBM MQ informações sobre licença.

Nota: Esta documentação se relaciona com MQIPT em IBM MQ 9.4 Para a documentação do pacote de suporte do MQIPT (versão 2.1) no IBM Documentation, consulte MQIPT (SupportPac MS81) na documentação do IBM MQ 9.0.

Nota: Se você estiver usando MQIPT 2.1 ou anterior, será recomendado fazer upgrade para o MQIPT for IBM MQ 9.4, pois a data de término do suporte para o pacote de suporte MQIPT foi 30th de setembro de 2020.

O IBM MQ Internet Pass-Thru é executado como um serviço independente que pode receber e encaminhar fluxos de mensagens do IBM MQ, entre dois gerenciadores de filas do IBM MQ ou entre um cliente do IBM MQ e um gerenciador de filas do IBM MQ.

MQIPT ativa essa conexão quando o cliente e o servidor não estão na mesma rede física.

Uma ou mais instâncias do MQIPT podem ser colocadas no caminho de comunicação entre dois gerenciadores de filas IBM MQ , ou entre um cliente IBM MQ e um gerenciador de filas IBM MQ . As instâncias do MQIPT permitem que os dois sistemas IBM MQ troquem mensagens sem a necessidade de uma conexão TCP/IP direta entre eles. Essa arquitetura é útil se a configuração de firewall proibir uma conexão TCP/IP direta entre os dois sistemas...

MQIPT atende em uma ou mais portas TCP/IP para conexões recebidas. Essas conexões podem transportar mensagens normais do IBM MQ , mensagens IBM MQ tuneladas dentro do HTTP ou mensagens criptografadas usando Segurança da camada de transporte (TLS) ou Secure Sockets Layer (SSL). MQIPT pode manipular várias conexões simultâneas.

O canal IBM MQ que faz a solicitação de conexão TCP/IP inicial é chamado de *responsável pela chamada*, o canal ao qual ele está tentando se conectar de *respondente* e o gerenciador de filas com quem ele está finalmente tentando entrar em contato de *gerenciador de filas de destino*.

MQIPT retém dados na memória conforme ele a encaminha de sua origem para seu destino. Nenhum dado é salvo no disco (exceto para a memória que o sistema operacional pagará no disco). A única vez que o MQIPT acessa o disco explicitamente é ler seu arquivo de configuração e gravar registros de log de conexão e de rastreio.

O intervalo completo de tipos de canais do IBM MQ pode se conectar por meio de uma ou mais instâncias do MQIPT.. A presença de MQIPT em um caminho de comunicação não tem efeito nas características funcionais dos componentes IBM MQ conectados. No entanto, o desempenho da transferência de mensagens pode ser afetado

MQIPT pode ser usado com IBM MQ conforme descrito em “Possíveis configurações do MQIPT” na página 299.

Para instalar o MQIPT, consulte Instalando o MQIPT.

Tarefas relacionadas

Configurando o IBM MQ Internet Pass-Thru

Administrando e configurando IBM MQ Internet Pass-Thru

Referências relacionadas

IBM MQ Internet Pass-Thru Referência de configuração

Usos do MQIPT

Há vários usos potenciais para o IBM MQ Internet Pass-Thru (MQIPT).

O MQIPT pode ser usado como um concentrador de canal

Ao usar MQIPT desta maneira, os canais para ou a partir de vários hosts separados podem aparecer para um firewall como se eles fossem todos para ou a partir do host MQIPT . Isso torna mais fácil definir e gerenciar regras de filtragem de firewall.

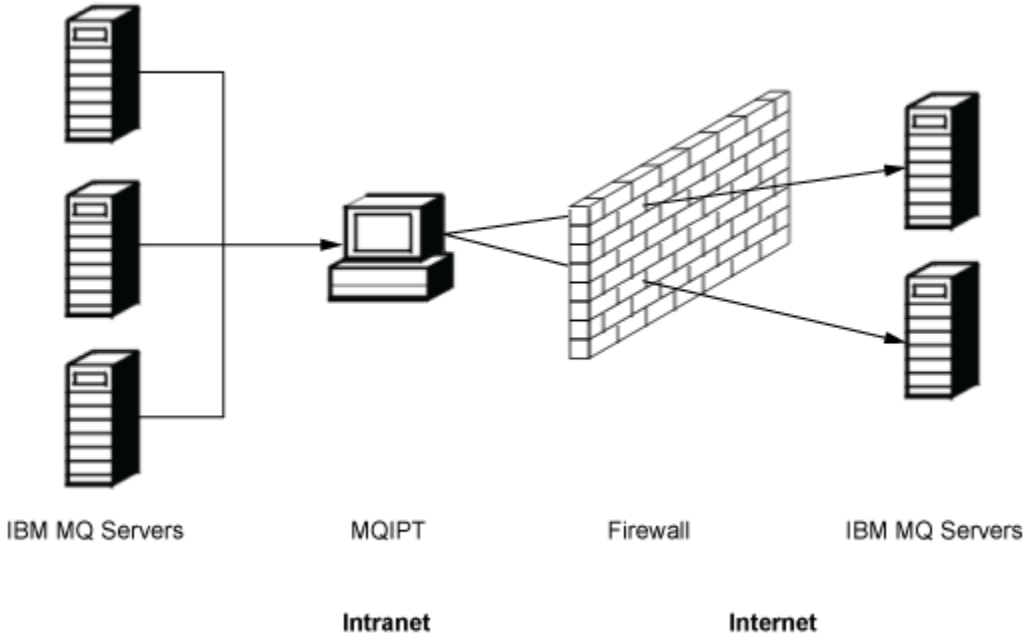


Figura 79. Exemplo de MQIPT como um concentrador de canal

MQIPT pode ser colocado em uma DMZ para fornecer um único ponto de acesso

Se MQIPT for colocado dentro de um firewall DMZ (uma configuração de firewall para proteção de redes locais), em um computador com um endereço de protocolo de Internet (IP) conhecido e confiável, MQIPT pode ser usado para atender as conexões de canal IBM MQ recebidas que ele pode, então, encaminhar para a intranet confiável; o firewall interno deve permitir que esse computador confiável faça conexões de entrada. Nessa configuração, o MQIPT evita que solicitações externas para acesso recebam os endereços IP verdadeiros dos computadores na intranet confiável. Desta maneira, MQIPT fornece um único ponto de acesso. Se necessário, o MQIPT pode ser configurado para aceitar conexões TLS e encaminhar dados para o destino usando uma conexão TLS separada, finalizando, portanto, a sessão TLS na DMZ

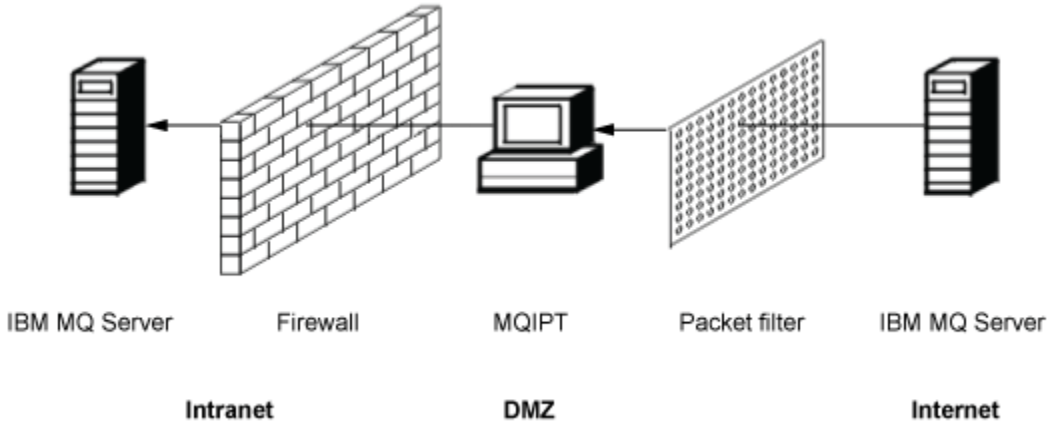


Figura 80. Exemplo de MQIPT em um Firewall DMZ

MQIPT pode se comunicar por meio de tunelamento de HTTP

Se duas instâncias de MQIPT forem implementadas na linha, elas poderão se comunicar usando HTTP. O recurso de tunelamento HTTP permite que pedidos sejam transmitidos através de firewalls, pelo uso de proxies de HTTP existentes. O primeiro MQIPT insere o protocolo IBM MQ em HTTP e o segundo extrai o protocolo do IBM MQ a partir de seu wrapper HTTP e o encaminha para o gerenciador de filas de destino.

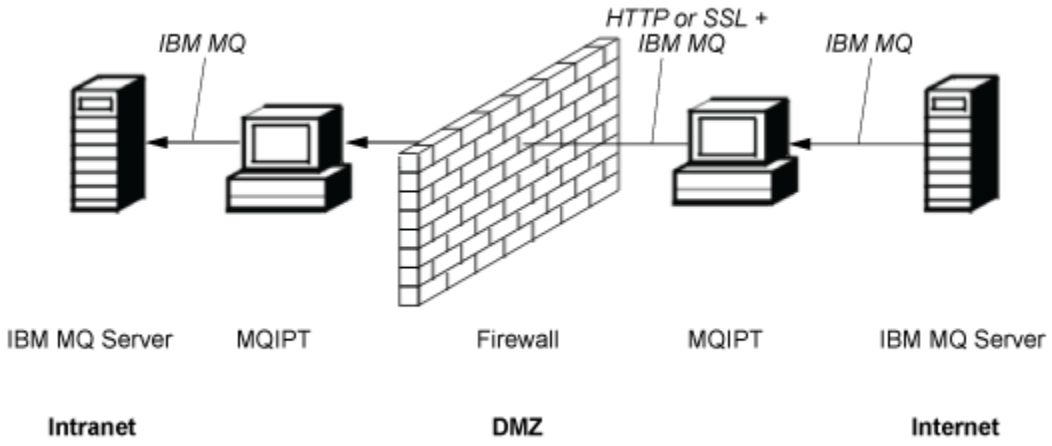


Figura 81. Exemplo de tunelamento de MQIPT e HTTP

O MQIPT pode criptografar mensagens

Se MQIPT for configurado como no exemplo anterior, as solicitações podem ser criptografadas antes da transmissão através de firewalls. O primeiro MQIPT criptografa os dados e o segundo decriptografa-o usando SSL/TLS antes de enviá-lo para o gerenciador de filas de destino.

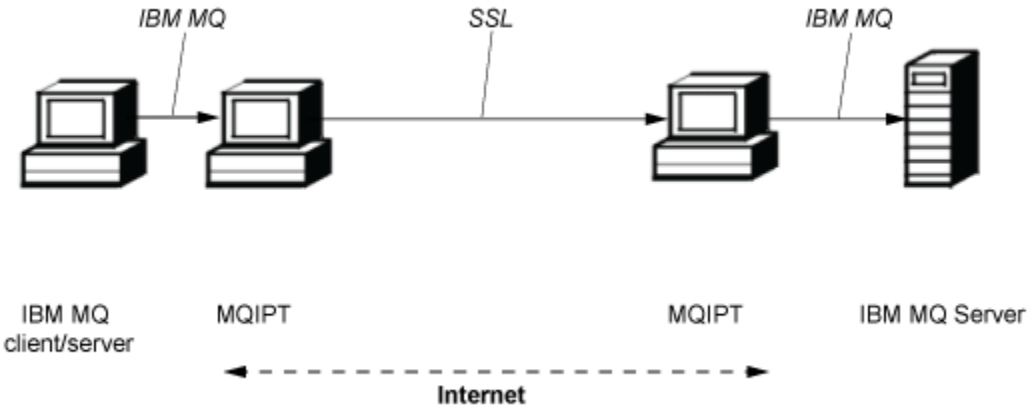


Figura 82. Exemplo de MQIPT e SSL/TLS

Como o MQIPT funciona

Em sua configuração mais simples, MQIPT age como um encaminhador de protocolo IBM MQ . Ele atende em uma porta TCP/IP e aceita solicitações de conexão de canais IBM MQ .

Se uma solicitação bem-formada for recebida, MQIPT estabelecerá uma conexão TCP/IP adicional entre si mesmo e o gerenciador de filas IBM MQ de destino. Em seguida, ele transmite todos os pacotes de protocolo que recebe de sua conexão de entrada para o gerenciador de filas de destino e retorna pacotes de protocolo do gerenciador de filas de destino de volta para a conexão de entrada original.

Nenhuma mudança para o protocolo IBM MQ (cliente / servidor ou gerenciador de filas para o gerenciador de filas) está envolvida porque nenhuma das finalização está diretamente ciente da presença do intermediário. Novas versões do cliente IBM MQ ou código do servidor não são necessárias.

Para usar MQIPT, o canal do responsável pela chamada deve ser configurado para usar o nome do host MQIPT e a porta, não o nome do host e a porta do gerenciador de filas de destino. Isso é definido com a propriedade **CONNAME** do canal IBM MQ. MQIPT lê os dados recebidos e simplesmente transmite-o para o gerenciador de filas de destino. Outros campos de configuração, como o ID do usuário e a senha em um canal de cliente / servidor, são transmitidos de forma semelhante ao gerenciador de filas de destino.

Vários gerenciadores de filas

MQIPT pode ser usado para permitir o acesso a mais de um gerenciador de filas de destino. Para que isso funcione, deve haver um mecanismo para informar ao MQIPT a qual gerenciador de filas se conectar, portanto, o MQIPT usa o número da porta TCP/IP recebida para determinar a qual gerenciador de filas se conectar.

Portanto, é possível configurar MQIPT para atender em várias portas TCP/IP. Cada porta de atendimento é mapeada para um gerenciador de filas de destino por meio de uma MQIPT*rota*. É possível definir até 100 dessas rotas, que associam uma porta TCP/IP de atendimento com o nome do host e a porta do gerenciador de filas de destino. Isso significa que o nome do host (endereço IP) do gerenciador de filas de destino nunca é visível para o canal de origem. Cada rota pode manipular várias conexões entre sua porta de atendimento e de destino, cada conexão agindo independentemente.

Arquivo de configuração do MQIPT

O MQIPT usa um arquivo de configuração chamado `mqipt.conf`. Este arquivo contém definições de todas as rotas e suas propriedades associadas. Veja Administrando e configurando o IBM MQ Internet Pass-Thru para obter mais informações sobre o `mqipt.conf`.

Quando MQIPT for ativado, ele iniciará cada rota que está listada no arquivo de configuração. As mensagens são gravadas no console do sistema mostrando o status de cada rota. Quando a mensagem MQCPI078 é mostrada para uma rota, essa rota está pronta para aceitar solicitações de conexão.

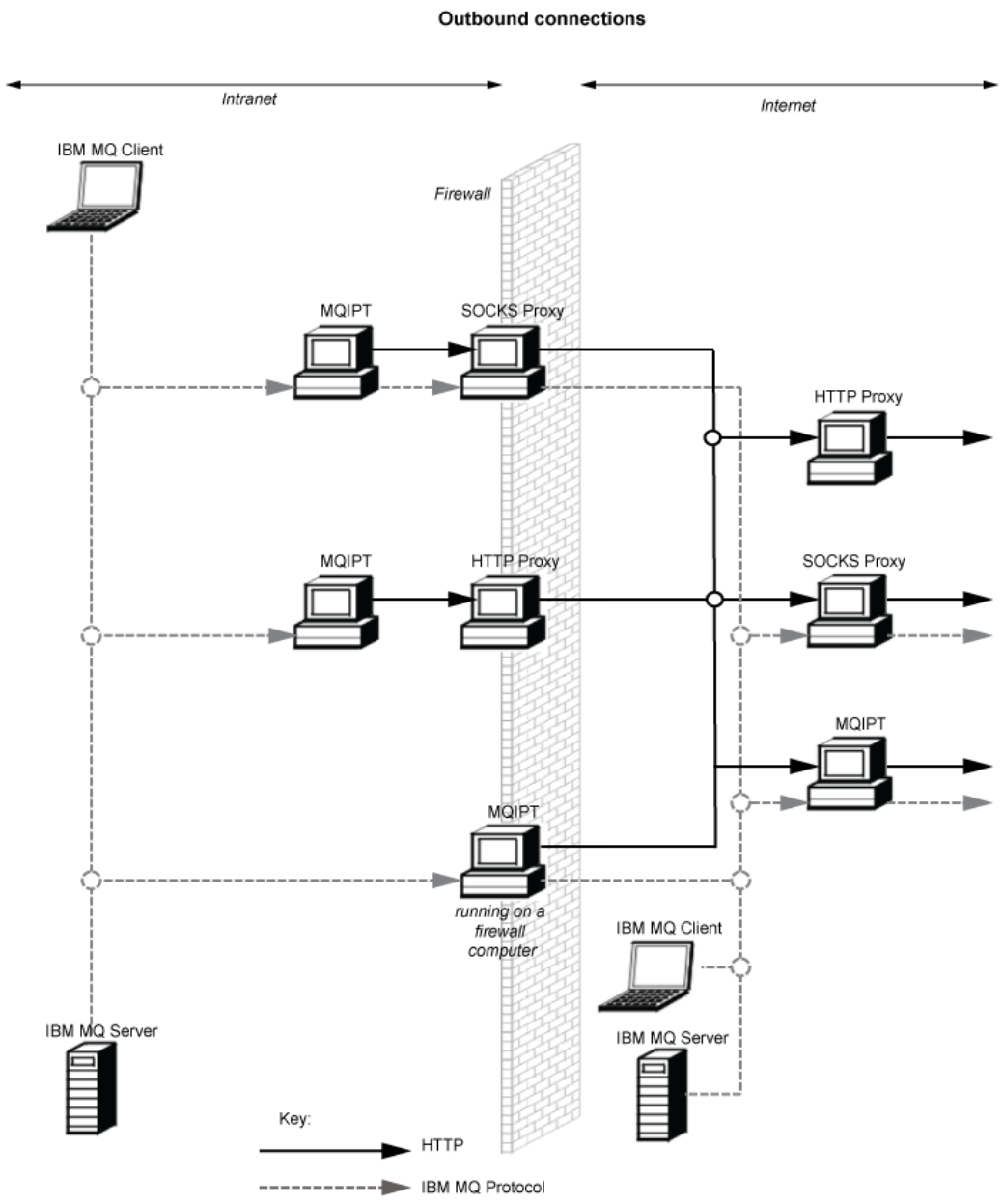
Possíveis configurações do MQIPT

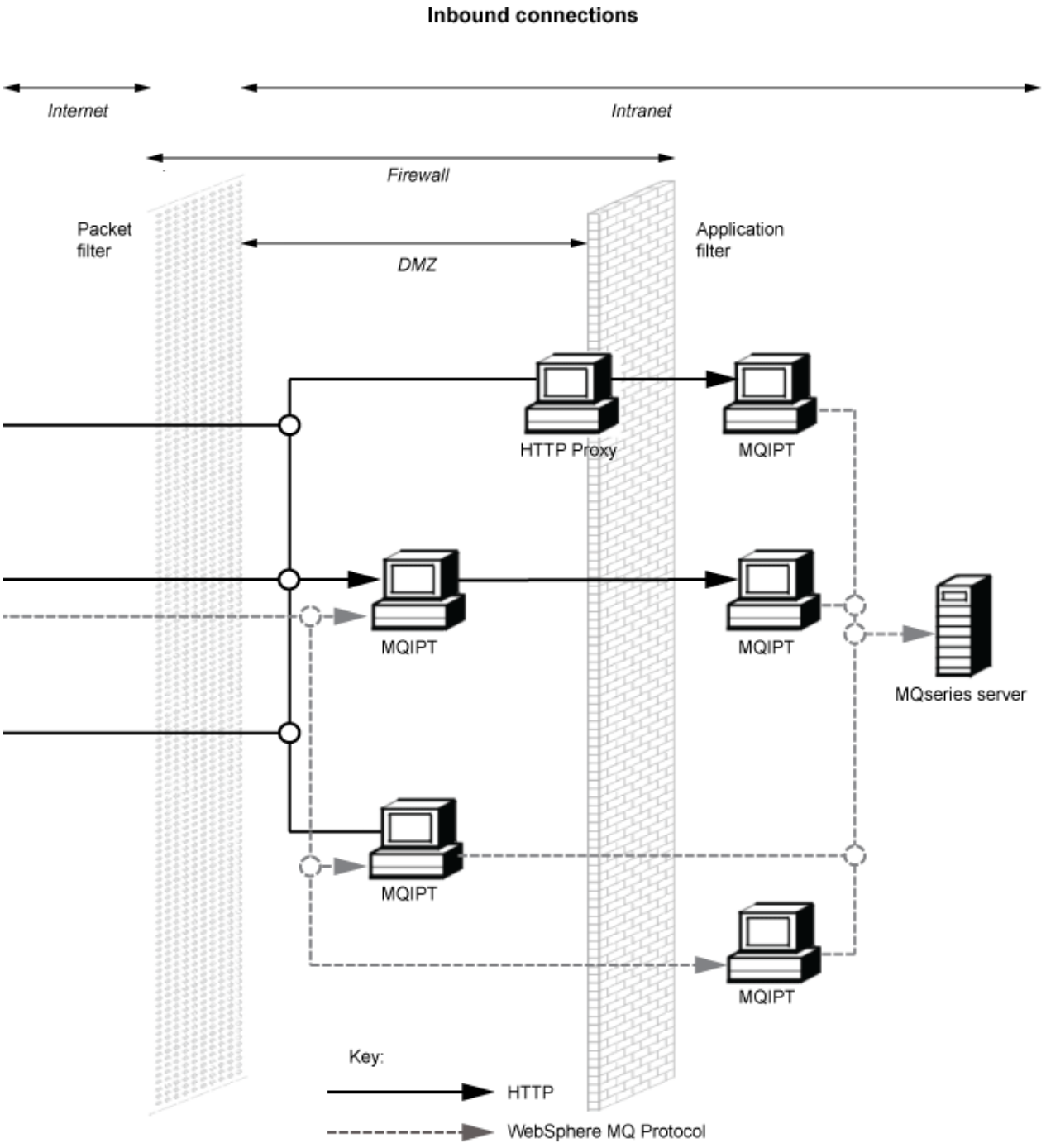
MQIPT pode ser usado em conjunto com IBM MQ e IBM Integration Bus.

A figura de várias partes a seguir mostra muitas das configurações possíveis para o MQIPT em uma topologia do IBM MQ. Ele ilustra maneiras diferentes nas quais o MQIPT pode enviar mensagens. Ele mostra clientes e servidores em uma intranet, dentro de um firewall, e na Internet fora do firewall, transmitindo mensagens para MQIPT, proxy HTTP ou proxy SOCKS, o que os encaminha.

As mensagens são recebidas por um proxy MQIPT ou um proxy HTTP em um DMZ antes de passar a mensagem pelo firewall de entrada para um servidor.

Observe que o proxy HTTP, o proxy SOCKS e os computadores MQIPT no lado da intranet do firewall representam a possibilidade de vários computadores encadeados juntos na Internet. Por exemplo, um computador MQIPT pode se comunicar por meio de um ou mais computadores de proxy SOCKS ou HTTP, ou computadores adicionais MQIPT , antes de atingir seu 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



Configurações compatíveis

Cenários de conexão compatíveis em que um cliente IBM MQ ou gerenciador de filas se comunica com MQIPT. A mesma rota ou uma segunda rota do MQIPT é usada para se comunicar com um gerenciador de filas de destino.

Configurações compatíveis com uma única rota MQIPT

É possível usar uma única rota MQIPT para se comunicar com IBM MQ.

As colunas no Tabela 26 na página 302 contêm as informações a seguir:

1. O protocolo usado entre IBM MQ e a rota MQIPT . A conexão pode ser criada por um cliente IBM MQ ou gerenciador de filas, e pode usar IBM MQ Formats and Protocols (FAP) ou um protocolo SSL/TLS.
2. O modo no qual a rota MQIPT opera. O formato da comunicação em toda a Internet entre MQIPT e IBM MQé determinado pela configuração da rota MQIPT . Observe que, em que a tabela menciona SSL, também é possível usar TLS.
3. O protocolo usado entre a rota MQIPT e o gerenciador de filas de destino.

Tabela 26. Configurações válidas com uma única instância do MQIPT

1. IBM MQ Protocolo de origem	**2. Modo da rota MQIPT**	**3. IBM MQ Protocolo de destino**
FAP	FAP-proxy (padrão)	FAP
	FAP-servidor e cliente SSL	SSL/TLS
SSL/TLS	SSL-proxy	SSL/TLS
	Servidor SSL e cliente FAP	FAP
	SSL-servidor e cliente SSL	SSL/TLS

Configurações compatíveis com mais de uma rota MQIPT

É possível optar por usar mais de uma rota, em uma ou mais instâncias de MQIPT, para se comunicar com IBM MQ.

As colunas no Tabela 27 na página 303 contêm as informações a seguir:

1. O protocolo usado entre IBM MQ e a primeira rota MQIPT . A conexão pode ser criada por um cliente IBM MQ ou gerenciador de filas, e pode usar IBM MQ Formats and Protocols (FAP) ou um protocolo SSL/TLS.
2. O modo no qual a primeira rota MQIPT opera. O formato da comunicação em toda a Internet entre MQIPT e IBM MQé determinado pela configuração da rota MQIPT . Observe que, em que a tabela menciona SSL, também é possível usar TLS.
3. O modo no qual a segunda rota MQIPT opera.
4. O protocolo usado entre a segunda rota MQIPT e o gerenciador de filas de destino.

Tabela 27. Configurações válidas com múltiplas instâncias do MQIPT			
1. IBM MQ Protocolo de origem	**2. Modo da primeira rota MQIPT**	**3. Modo da segunda rota MQIPT**	**4. IBM MQ Protocolo de destino**
FAP (padrão)	FAP-proxy (padrão)	FAP-proxy (padrão)	FAP
	FAP-servidor e cliente SSL	SSL-proxy	SSL/TLS
		Servidor SSL e cliente FAP	FAP
		SSL-servidor e cliente SSL	SSL/TLS
	HTTP-cliente	HTTP-servidor e cliente SSL	SSL/TLS
	HTTPS-cliente	HTTPS-servidor e cliente SSL	SSL/TLS
	HTTP-cliente	HTTP-servidor	FAP
	HTTPS-cliente	HTTPS-servidor	FAP
SSL/TLS	SSL-proxy	SSL-proxy	SSL/TLS
		Servidor SSL e cliente FAP	FAP
		SSL-servidor e cliente SSL	SSL/TLS
	HTTP-cliente	HTTP-servidor	FAP
	HTTPS-cliente	HTTPS-servidor	SSL/TLS
	HTTP-cliente	HTTP-servidor e cliente SSL	FAP
	HTTPS-cliente	HTTPS-servidor e cliente SSL	SSL/TLS

Configurações de Canal Suportadas

Todos os tipos de canais IBM MQ são suportados, mas a configuração está restrita a conexões TCP/IP. Para um cliente IBM MQ ou gerenciador de filas, MQIPT aparece como se fosse o gerenciador de filas de destino. Em que a configuração do canal requer um host de destino e número de porta, o nome do host MQIPT e o número da porta do listener são especificados.

Canais cliente / servidor

MQIPT atende as solicitações de conexão do cliente recebidas e, em seguida, as encaminha usando os pacotes de protocolo HTTP Tunneling, SSL/TLS ou como padrão IBM MQ . Se MQIPT estiver usando o túnel HTTP ou SSL/TLS, ele os encaminha em uma conexão com um segundo MQIPT. Se não estiver usando o tunelamento HTTP, ele os encaminhará em uma conexão com o que ele vir como o gerenciador de filas de destino (embora isso possa, por sua vez, ser um MQIPT adicional). Quando o gerenciador de filas de destino tiver aceito a conexão do cliente, os pacotes serão retransmitidos entre o cliente e o servidor.

Emissor de cluster / canais receptores

Se MQIPT receber uma solicitação de entrada de um canal do emissor de clusters, ele assumirá que o gerenciador de filas tenha sido ativado para SOCKS e o endereço de destino verdadeiro será obtido durante o processo de handshaking SOCKS. Ele encaminha a solicitação para o próximo MQIPT ou gerenciador de filas de destino exatamente da mesma maneira que para canais de conexão do cliente. Isso também inclui canais de emissores de cluster definidos automaticamente.

Emissor/receptor
Se MQIPT receber uma solicitação recebida de um canal emissor, ele a encaminhará para o próximo gerenciador de filas de destino MQIPT ou de destino exatamente da mesma maneira que para canais de conexão do cliente. O gerenciador de filas de destino valida a solicitação recebida e inicia o canal receptor, caso seja apropriado. Todas as comunicações entre o canal emissor e receptor (incluindo os fluxos de segurança) são retransmitidas.

Solicitante-servidor
Essa combinação é manipulada da mesma maneira que as configurações anteriores. A validação da solicitação de conexão é executada pelo canal do servidor no gerenciador de filas de destino.

Solicitante-remetente
A configuração "retorno de chamada" poderia ser útil se os dois gerenciadores de filas não estiverem autorizados a estabelecer conexões diretas entre si, mas ambos podem se conectar ao MQIPT e aceitar conexões a partir dele.

Servidor / solicitante e servidor / receptor
Elas são manipuladas pelo MQIPT da mesma maneira que manipula a configuração `Sender/Receiver`.

Condições de término e falha do canal

Quando o MQIPT detecta o fechamento (normal ou anormal) de um canal IBM MQ , ele propaga o fechamento do canal. Se você fechar uma rota usando MQIPT, todos os canais que passarem por essa rota serão fechados.

MQIPT fornece um recurso de tempo limite inativo opcional. Se MQIPT detectar que um canal ficou inativo por um período de tempo excedendo o tempo limite, ele executará um encerramento imediato nas duas conexões em questão.

Os sistemas IBM MQ em qualquer extremidade do canal observam essas condições de encerramento anormais, como falhas de rede ou como finalização do canal pelo parceiro. O canal é, então, capaz de reiniciar e recuperar (se a falha ocorrer durante um período de protocolo em dúvida) como se MQIPT não estivesse sendo usado.

Segurança de Mensagens

O gerenciamento de filas distribuído do IBM MQ garante que as mensagens sejam entregues adequadamente. Esse ainda é o caso quando o MQIPT está presente entre as duas extremidades do canal. O MQIPT não armazena nenhum dado da mensagem ou participa do procedimento de ponto de sincronização que assegura a entrega de mensagem correta.

Ao usar mensagens IBM MQ rápidas, não persistentes, se a rota MQIPT falhar ou for reiniciado quando uma mensagem IBM MQ estiver em trânsito, a mensagem poderá ser perdida. Antes de reiniciar a rota, assegure-se de que todos os canais IBM MQ usando a rota MQIPT estejam inativos.

Para obter mais informações sobre a segurança de mensagens no IBM MQ, consulte Segurança de mensagens.

Gerenciadores de filas de várias instâncias e alta disponibilidade

MQIPT pode ser usado com gerenciadores de filas de várias instâncias em ambientes de alta disponibilidade.

MQIPT não possui um estado persistente e, portanto, não há nenhum benefício na falha por meio de MQIPT para outro sistema. Em vez disso, tenha várias instâncias do MQIPT com arquivos de configuração do `mqipt.conf` idênticos em execução em sistemas diferentes. Monitore cada instância de MQIPT para disponibilidade e reinicie-a (no mesmo sistema), se necessário. Isso fornece um conjunto de instâncias MQIPT idênticas que podem ser usadas para rotear conexões. Em seguida, deve-se assegurar que IBM MQ possa rotear conexões para MQIPT e que MQIPT possa encaminhar essas conexões para o gerenciador de filas de destino.

Os canais IBM MQ de saída podem ser direcionados para uma instância disponível do MQIPT de várias maneiras, por exemplo:

- Use um balanceador de carga ou roteador de alta disponibilidade, como o IBM Network Dispatcher a partir do produto WebSphere Edge Components.
- Especifique vários nomes de conexão na definição de canal IBM MQ usando uma lista separada por vírgula. IBM MQ tenta, então, se conectar a cada endereço MQIPT , por vez, até localizar uma instância MQIPT disponível.

Você também deve direcionar conexões de MQIPT para o gerenciador de filas de destino. Se a configuração de alta disponibilidade assegurar que o endereço IP efetua failover com o gerenciador de filas de destino, nenhuma configuração especial do MQIPT será necessária: especifique o endereço IP de destino na propriedade de rota **Destination** e permita que a operação de failover mova o endereço IP com o gerenciador de filas.

No entanto, se o endereço IP do gerenciador de filas for alterado após um failover, você deverá organizar para MQIPT para encaminhar a conexão para o destino correto. Isso pode ser feito de uma de várias maneiras:

- Grave uma saída de roteamento que verifique qual endereço IP e número da porta estão acessíveis e, em seguida, substitua o destino de rota para cada conexão. Algumas saídas de roteamento de amostra são fornecidas com MQIPT; elas podem ser adaptadas para esse propósito.
- Use um balanceador de carga de alta disponibilidade para redirecionar a conexão.
- Defina diversas rotas MQIPT , uma para cada endereço IP e porta em que o gerenciador de filas pode estar em execução. Em seguida, direcione as conexões IBM MQ para as várias rotas MQIPT , por exemplo, listando todos os endereços IP de rota e números de porta em uma lista separada por vírgulas no nome da conexão do canal de saída.

Também é importante ajustar todos os componentes de ponta a ponta no caminho da rede:

1. As tentativas de conexão para sistemas indisponíveis devem falhar imediatamente para que as tentativas de reconexão possam se mover para o primeiro destino disponível.

   Para rotas SSL do MQIPT, ajuste a rota **SSLClientConnectTimeout** para assegurar falha na conexão de prompt para destinos indisponíveis. Consulte a documentação do IBM MQ para obter detalhes dos parâmetros de ajuste IBM MQ . Além disso, consulte a documentação do sistema operacional para obter detalhes sobre o ajuste do TCP/IP para o sistema operacional. Em todos os casos, as tentativas de conexão com falha devem retornar rapidamente uma falha de rede (por exemplo, um pacote de reconfiguração TCP) ou deve passar o tempo limite sem atraso indevido.
2. As conexões ativas para um sistema com falha devem ser interrompidas imediatamente para que novas conexões possam ser estabelecidas.

   Também é necessário considerar o impacto de um failover em um momento em que as conexões estão ativamente usando MQIPT. É provável que as conexões de rede sejam interrompidas durante um failover. Para aplicativos clientes, é possível usar o recurso de reconexão automática do cliente IBM MQ para restabelecer conexões interrompidas. Para canais de mensagens, é possível especificar um intervalo de nova tentativa curto para que o canal reconecte-se imediatamente. Consulte a documentação IBM MQ para obter mais informações sobre reconexão automática do cliente e a configuração de nova tentativa do canal de mensagem.

O IBM MQ Console e REST API

É possível usar o IBM MQ Console e REST API para administrar IBM MQe executar operações do sistema de mensagens usando HTTP.

- É possível usar o IBM MQ Console para executar tarefas básicas de administração de um navegador da web. Para obter mais informações, consulte Administração Usando o IBM MQ Console .
- É possível usar a administrative REST API para administrar objetos do IBM MQ, como gerenciadores de filas e filas e agentes do Managed File Transfer e transferências. Para obter mais informações, consulte Administração Usando o REST API.

- É possível usar o messaging REST API para executar o sistema de mensagens simples ponto a ponto e de publicação Para obter mais informações, consulte Sistema de mensagens que usa o REST API

Opções de instalação

O IBM MQ Console e o REST API são executados em um servidor WebSphere Liberty , chamado mqweb No IBM MQ 9.3.5, é possível instalar o servidor mqweb como um componente opcional em uma instalação do IBM MQ ou como uma instalação independente do IBM MQ Web Server .

Linux V 9.4.0 **Instalação independente do IBM MQ Web Server**

No IBM MQ 9.4.0, o servidor mqweb pode ser executado em uma instalação independente do IBM MQ Web Server. Uma instalação independente do IBM MQ Web Server permite instalar e executar o servidor mqweb em sistemas que são separados para suas instalações do IBM MQ . A instalação de um IBM MQ Web Server independente fornece maior flexibilidade sobre quais sistemas e o número de sistemas nos quais você escolhe executar seus servidores mqweb. Se necessário, várias instâncias do servidor mqweb podem ser executadas em máquinas diferentes para fornecer a escalabilidade e a disponibilidade necessárias.

Se você tiver comprado a autorização IBM MQ , será possível instalar quantas cópias forem necessárias do IBM MQ Web Serverindependente. As instalações do IBM MQ Web Server não são contadas com relação à sua autorização adquirida do IBM MQ. Para obter mais informações sobre o licenciamento do IBM MQ, consulte Informações sobre licença do IBM MQ.

As restrições a seguir se aplicam em uma instalação independente do IBM MQ Web Server :

- O IBM MQ Console pode ser usado para administrar apenas os gerenciadores de filas remotas
- O messaging REST API pode ser usado apenas com gerenciadores de fila remotos..
- O administrative REST API não está disponível

O IBM MQ Web Server independente é suportado apenas em plataformas Linux ..

Para obter mais informações sobre como instalar o IBM MQ Web Serverindependente, consulte Instalando o IBM MQ Web Serverindependente.

Componente opcional de uma instalação do IBM MQ

É possível optar por instalar o componente IBM MQ Console e REST API como parte de uma instalação do IBM MQ .

Todos os recursos IBM MQ Console e REST API estão disponíveis quando o servidor mqweb é executado em uma instalação do IBM MQ .

- O IBM MQ Console pode ser usado para administrar gerenciadores de fila locais e remotos..
- O messaging REST API pode ser usado com os gerenciadores de filas locais e remotos
- O administrative REST API pode ser usado para administrar gerenciadores de fila locais e remotos..

Para usar o componente IBM MQ Console e REST API , instale o componente a seguir como parte de sua instalação do IBM MQ :

- AIX No AIX, instale o conjunto de arquivos `mqm.web.rte`.
- IBM i No IBM i, instale o componente WEB.
- Linux No Linux, instale o componente `MQSeriesWeb`.
- Windows No Windows, instale o recurso `Web Administration`.
- z/OS No z/OS, instale o recurso `IBM MQ for z/OS UNIX System Services Web Components`.

Avisos

Estas informações foram desenvolvidas para produtos e serviços oferecidos nos Estados Unidos.

É possível que a IBM não ofereça os produtos, serviços ou recursos discutidos nesta publicação em outros países. Consulte seu representante local do IBM para obter informações sobre produtos e serviços disponíveis atualmente em sua área. Qualquer referência a produtos, programas ou serviços IBM não significa que apenas produtos, programas ou serviços IBM possam ser utilizados. Qualquer outro produto, programa ou serviço, funcionalmente equivalente, poderá ser utilizado em substituição daqueles, desde que não infrinja nenhum direito de propriedade intelectual da IBM. Entretanto, a avaliação e verificação da operação de qualquer produto, programa ou serviço não IBM são de responsabilidade do Cliente.

A IBM pode ter patentes ou aplicativos de patentes pendentes relativas aos assuntos tratados nesta publicação. O fornecimento desta publicação não garante ao Cliente nenhum sobre tais patentes. é possível enviar pedidos de licença, por escrito, para:

Gerência de Relações Comerciais e Industriais da IBM Brasil
Av. Pasteur, 138-146
Botafogo
Rio de Janeiro, RJ
U.S.A.

Para pedidos de licença relacionados a informações de DBCS (Conjunto de Caracteres de Byte Duplo), entre em contato com o Departamento de Propriedade Intelectual da IBM em seu país ou envie pedidos de licença, por escrito, para:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

O parágrafo a seguir não se aplica a nenhum país em que tais disposições não estejam de acordo com a legislação local: A INTERNATIONAL BUSINESS MACHINES CORPORATION FORNECE ESTA PUBLICAÇÃO "NO ESTADO EM QUE SE ENCONTRA", SEM GARANTIA DE NENHUM TIPO, SEJA EXPRESSA OU IMPLÍCITA, INCLUINDO, MAS A ELAS NÃO SE LIMITANDO, AS GARANTIAS IMPLÍCITAS DE NÃO INFRAÇÃO, COMERCIALIZAÇÃO OU ADEQUAÇÃO A UM DETERMINADO PROPÓSITO. Alguns países não permitem a exclusão de garantias expressas ou implícitas em certas transações; portanto, essa disposição pode não se aplicar ao Cliente.

Essas informações podem conter imprecisões técnicas ou erros tipográficos. São feitas alterações periódicas nas informações aqui contidas; tais alterações serão incorporadas em futuras edições desta publicação. IBM pode aperfeiçoar e/ou alterar no produto(s) e/ou programa(s) descritos nesta publicação a qualquer momento sem aviso prévio.

Todas as referências nessas informações a websites não IBM são fornecidas somente por conveniência e de forma alguma são um endosso a esses websites. Os materiais contidos nesses websites não fazem parte dos materiais desse produto IBM e a utilização desses websites é de inteira responsabilidade do Cliente.

A IBM pode utilizar ou distribuir as informações fornecidas da forma que julgar apropriada sem incorrer em qualquer obrigação para com o Cliente.

Os licenciados deste programa que desejarem obter informações sobre este assunto com o propósito de permitir: (i) a troca de informações entre programas criados independentemente e outros programas (incluindo este) e (ii) o uso mútuo das informações trocadas, deverão entrar em contato com:

Av. Pasteur, 138-146
Av. Pasteur, 138-146

Botafogo
Rio de Janeiro, RJ
U.S.A.

Tais informações podem estar disponíveis, sujeitas a termos e condições apropriadas, incluindo em alguns casos o pagamento de uma taxa.

O programa licenciado descrito nesta publicação e todo o material licenciado disponível para ele são fornecidos pela IBM sob os termos do IBM Customer Agreement, IBM Contrato de Licença do Programa Internacional ou qualquer contrato equivalente entre as partes.

Todos os dados de desempenho aqui contidos foram determinados em um ambiente controlado. Portanto, os resultados obtidos em outros ambientes operacionais podem variar significativamente. Algumas medidas podem ter sido tomadas em sistemas em nível de desenvolvimento e não há garantia de que estas medidas serão iguais em sistemas geralmente disponíveis. Além disto, algumas medidas podem ter sido estimadas através de extrapolação. Os resultados reais podem variar. usuários deste documento devem verificar os dados aplicáveis para seu ambiente específico.

As informações relativas a produtos não IBM foram obtidas junto aos fornecedores dos respectivos produtos, de seus anúncios publicados ou de outras fontes disponíveis publicamente. A IBM não testou estes produtos e não pode confirmar a precisão de seu desempenho, compatibilidade nem qualquer outra reivindicação relacionada a produtos não IBM. Dúvidas sobre os recursos de produtos não IBM devem ser encaminhadas diretamente a seus fornecedores.

Todas as declarações relacionadas aos objetivos e intenções futuras da IBM estão sujeitas a alterações ou cancelamento sem aviso prévio e representam somente metas e objetivos.

Essas informações contêm exemplos de dados e relatórios utilizados em operações diárias de negócios. Para ilustrá-los da forma mais completa possível, os exemplos incluem nomes de indivíduos, empresas, marcas e produtos. Todos estes nomes são fictícios e qualquer semelhança com os nomes e endereços utilizados por uma empresa real é mera coincidência.

LICENÇA DE COPYRIGHT:

Estas informações contêm programas de aplicativos de amostra na linguagem fonte, ilustrando as técnicas de programação em diversas plataformas operacionais. O Cliente pode copiar, modificar e distribuir estes programas de amostra sem a necessidade de pagar à IBM, com objetivos de desenvolvimento, uso, marketing ou distribuição de programas aplicativos em conformidade com a interface de programação de aplicativo para a plataforma operacional para a qual os programas de amostra são criados. Esses exemplos não foram testados completamente em todas as condições. Portanto, a IBM não pode garantir ou implicar a confiabilidade, manutenção ou função destes programas.

Se estiver visualizando estas informações em formato eletrônico, as fotografias e ilustrações coloridas poderão não aparecer.

Informações sobre a Interface de Programação

As informações da interface de programação, se fornecidas, destinam-se a ajudá-lo a criar software aplicativo para uso com este programa.

Este manual contém informações sobre as interfaces de programação desejadas que permitem que o cliente grave programas para obter os serviços do IBM MQ

No entanto, estas informações também podem conter informações sobre diagnósticos, modificações e ajustes. As informações sobre diagnósticos, modificações e ajustes são fornecidas para ajudá-lo a depurar seu software aplicativo.

Importante: Não use essas informações de diagnóstico, modificação e ajuste como uma interface de programação, pois elas estão sujeitas a mudanças

Marcas comerciais

IBM, o logotipo IBM , ibm.com, são marcas registradas da IBM Corporation, registradas em várias jurisdições no mundo todo Uma lista atual de marcas registradas da IBM está disponível na Web em "Informações de copyright e marca registrada"www.ibm.com/legal/copytrade.shtml. Outros nomes de produtos e serviços podem ser marcas comerciais da IBM ou de outras empresas.

Microsoft e Windows são marcas registradas da Microsoft Corporation nos Estados Unidos e/ou em outros países.

UNIX é uma marca registrada do The Open Group nos Estados Unidos e em outros países.

Linux é uma marca registrada de Linus Torvalds nos Estados Unidos e/ou em outros países.

Este produto inclui software desenvolvido pelo Projeto Eclipse (https://www.eclipse.org/).

Java e todas as marcas registradas e logotipos baseados em Java são marcas ou marcas registradas da Oracle e/ou de suas afiliadas.

Part Number:

(1P) P/N: