9.4

IBM MQ Panoramica tecnica

Nota

Prima di utilizzare queste informazioni e il prodotto che supportano, leggere le informazioni in “Informazioni particolari” a pagina 307.

Questa edizione si applica alla versione 9 release 4 di IBM® MQ e a tutte le successive release e modifiche se non diversamente indicato nelle nuove edizioni.

Indice

IBM MQ - Sommario tecnico

Utilizzare IBM MQ per collegare le applicazioni e gestire la distribuzione delle informazioni nell'organizzazione.

IBM MQ consente ai programmi di comunicare tra loro attraverso una rete di componenti diversi (processori, sistemi operativi, sottosistemi e protocolli di comunicazione) utilizzando un'API (application programming interface) coerente. Le applicazioni progettate e scritte utilizzando questa interfaccia sono note come applicazioni di accodamento messaggi.

Utilizzare i seguenti argomenti secondari per informazioni sull'accodamento dei messaggi e su altre funzioni fornite da IBM MQ.

Concetti correlati

Introduzione a IBM MQ

Dove trovare i requisiti del prodotto e le informazioni di supporto

Attività correlate

Pianificazione di un'architettura IBM MQ

Riferimenti correlati

“Caratteristiche principali e vantaggi dell'accodamento dei messaggi” a pagina 7

Queste informazioni evidenziano alcune funzioni e vantaggi dell'accodamento dei messaggi. Descrive funzioni quali la sicurezza e l'integrità dei dati dell'accodamento dei messaggi.

Introduzione all'accodamento dei messaggi

I prodotti IBM MQ consentono ai programmi di comunicare tra loro attraverso una rete di componenti diversi (processori, sistemi operativi, sottosistemi e protocolli di comunicazione) utilizzando un'API (application programming interface) coerente.

Le applicazioni progettate e scritte utilizzando questa interfaccia sono note come applicazioni di *accodamento messaggi* , poiché utilizzano lo stile di *messaggistica* e *accodamento* :

- La messaggistica significa che i programmi comunicano inviando reciprocamente i dati nei messaggi piuttosto che chiamandosi direttamente.
- L'accodamento significa che i messaggi vengono inseriti nelle code nella memoria, consentendo ai programmi di essere eseguiti indipendentemente l'uno dall'altro, a velocità e tempi diversi, in ubicazioni diverse e senza avere una connessione logica tra loro.

L'accodamento dei messaggi è stato utilizzato nell'elaborazione dei dati per molti anni. È più comunemente usato oggi nella posta elettronica. Senza accodamento, l'invio di un messaggio elettronico su lunghe distanze richiede che ogni nodo dell'instradamento sia disponibile per l'inoltro dei messaggi e che i destinatari siano collegati e consapevoli del fatto che si sta tentando di inviare loro un messaggio. In un sistema di accodamento, i messaggi vengono memorizzati su nodi intermedi finché il sistema è pronto ad inoltrarli. Alla loro destinazione finale sono memorizzati in una casella di posta elettronica fino a quando il destinatario è pronto a leggerli.

Anche così, molte transazioni aziendali complesse vengono elaborate oggi senza accodare. In una rete di grandi dimensioni, il sistema potrebbe mantenere molte migliaia di connessioni in uno stato di pronto utilizzo. Se una parte del sistema subisce un problema, molte parti del sistema diventano inutilizzabili.

È possibile pensare all'accodamento di messaggi come posta elettronica per programmi. In un ambiente di accodamento messaggi, ogni programma che fa parte di una suite di applicazioni esegue una funzione ben definita e autonoma in risposta a una specifica richiesta. Per comunicare con un altro programma, un programma deve inserire un messaggio su una coda predefinita. L'altro programma richiama il messaggio dalla coda ed elabora le richieste e le informazioni contenute nel messaggio. Quindi l'accodamento dei messaggi è uno stile di comunicazione tra programmi.

L'accodamento è il meccanismo con cui i messaggi vengono conservati fino a quando un'applicazione non è pronta per elaborarli. L'accodamento consente di:

- Comunicare tra i programmi (che potrebbero essere in esecuzione in ambienti differenti) senza dover scrivere il codice di comunicazione.
- Selezionare l'ordine in cui un programma elabora i messaggi.
- Bilanciare i caricamenti su un sistema facendo in modo che più di un programma servano una coda quando il numero di messaggi supera una soglia.
- Aumentare la disponibilità delle applicazioni disponendo un sistema alternativo per servire le code se il sistema primario non è disponibile.

Cos' è una coda messaggi?

Una coda messaggi, nota semplicemente come coda, è una destinazione denominata a cui possono essere inviati i messaggi. I messaggi si accumulano sulle code fino a quando non vengono richiamate dai programmi che gestiscono tali code.

Le code si trovano in, e sono gestite da, un gestore code (consultare “Terminologia di accodamento messaggi” a pagina 9). La natura fisica di una coda dipende dal sistema operativo su cui è in esecuzione il gestore code. Una coda può essere un'area di buffer volatile nella memoria di un computer o un dataset su una periferica di archiviazione permanente (come un disco). La gestione fisica delle code è responsabilità del gestore code e non è resa evidente ai programmi di applicazione partecipanti.

I programmi accedono alle code solo tramite i servizi esterni del gestore code. Possono aprire una coda, inserire messaggi, ricevere messaggi da essa e chiudere la coda. Possono anche impostare e analizzare gli attributi delle code.

Diversi stili di accodamento dei messaggi

Point-to-point

Un messaggio viene inserito nella coda e un'applicazione riceve quel messaggio.

Nella messaggistica point - to - point, un'applicazione di invio deve conoscere le informazioni sull'applicazione di ricezione prima di poter inviare un messaggio a tale applicazione. Ad esempio, l'applicazione mittente potrebbe dover conoscere il nome della coda a cui inviare le informazioni e potrebbe anche specificare un nome gestore code.

Pubblicazione/Sottoscrizione

Una copia di ogni messaggio pubblicato da un'applicazione di pubblicazione viene consegnata a ogni applicazione interessata. Potrebbero essere presenti molte, una o nessuna applicazione interessata. Nella pubblicazione / sottoscrizione un'applicazione interessata è nota come sottoscrittore e i messaggi vengono accodati su una coda identificata da una sottoscrizione.

La messaggistica di pubblicazione / sottoscrizione consente di disaccoppiare il fornitore di informazioni dai consumatori di tali informazioni. L'applicazione mittente e l'applicazione ricevente non hanno bisogno di sapere molto l'una sull'altra per le informazioni da inviare e ricevere. Per ulteriori informazioni, consultare “Pubblicazione/sottoscrizione della messaggistica” a pagina 63.

Vantaggi dell'accodamento dei messaggi allo sviluppatore e allo sviluppatore dell'applicazione

IBM MQ consente ai programmi di applicazione di utilizzare l' *accodamento dei messaggi* per partecipare all'elaborazione basata sui messaggi. I programmi applicativi possono comunicare tra diverse piattaforme utilizzando i prodotti software di accodamento dei messaggi appropriati. Ad esempio, le applicazioni z/OS possono comunicare tramite IBM MQ for z/OS. Le applicazioni sono protette dalla meccanica delle comunicazioni sottostanti. Alcuni degli altri vantaggi dell'accodamento dei messaggi sono:

- È possibile progettare applicazioni utilizzando piccoli programmi che è possibile condividere tra molte applicazioni.

- È possibile creare rapidamente nuove applicazioni riutilizzando questi blocchi di creazione.
- Le applicazioni scritte per utilizzare le tecniche di accodamento dei messaggi non sono influenzate dalle modifiche apportate al funzionamento dei gestori code.
- Non è necessario utilizzare alcun protocollo di comunicazione. Il gestore code gestisce tutti gli aspetti della comunicazione.
- Non è necessario che i programmi che ricevono messaggi siano in esecuzione nel momento in cui vengono loro inviati. I messaggi vengono conservati nelle code.

I progettisti possono ridurre il costo delle applicazioni perché lo sviluppo è più rapido, sono necessari meno sviluppatori e le richieste di capacità di programmazione sono inferiori rispetto a quelle per le applicazioni che non utilizzano l'accodamento dei messaggi.

IBM MQ implementa un'API (application programming interface) comune nota come *interfaccia coda messaggi* (o MQI) ovunque vengano eseguite le applicazioni. In questo modo, è più semplice trasferire i programmi applicativi da una piattaforma all'altra.

Per i dettagli su MQI, vedere Panoramica di Message Queue Interface.

Caratteristiche principali e vantaggi dell'accodamento dei messaggi

Queste informazioni evidenziano alcune funzioni e vantaggi dell'accodamento dei messaggi. Descrive funzioni quali la sicurezza e l'integrità dei dati dell'accodamento dei messaggi.

Le funzioni principali delle applicazioni che utilizzano le tecniche di accodamento dei messaggi sono:

Nota: Quando si considerano client e server IBM MQ , non è necessario modificare un'applicazione server per supportare ulteriori IBM MQ MQI clients su nuove piattaforme. Allo stesso modo, IBM MQ MQI client può, senza modifiche, funzionare con ulteriori tipi di server.

Nessuna connessione diretta tra programmi

L'accodamento dei messaggi è una tecnica di comunicazione indiretta da programma a programma. Può essere utilizzato all'interno di qualsiasi applicazione in cui i programmi comunicano tra loro. La comunicazione viene eseguita da un programma che immette i messaggi su una coda (di proprietà di un gestore code) e da un altro programma che riceve i messaggi dalla coda.

I programmi possono ricevere messaggi che sono stati inseriti in una coda da altri programmi. Gli altri programmi possono essere connessi allo stesso gestore code del programma di ricezione o a un altro gestore code. Questo altro gestore code potrebbe trovarsi su un altro sistema, su un altro sistema di computer o anche all'interno di un'azienda o di un'azienda diversa.

Non esistono connessioni fisiche tra i programmi che comunicano utilizzando le code messaggi. Un programma invia messaggi a una coda di proprietà di un gestore code e un altro programma richiama i messaggi dalla coda (consultare Figura 1 a pagina 8).

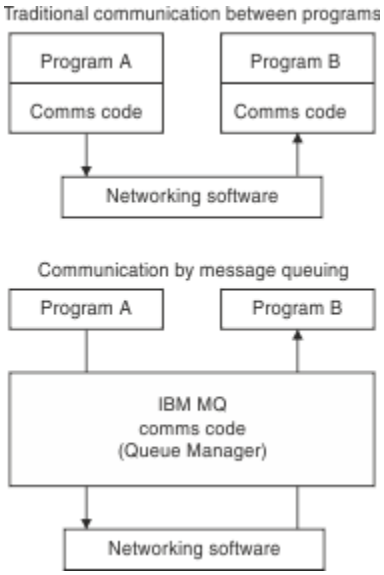


Figura 1. Accodamento messaggi rispetto alla comunicazione tradizionale

Come per la posta elettronica, i singoli messaggi che fanno parte di una transazione viaggiano attraverso una rete su un store - and - forwardbase. Se un collegamento tra i nodi ha esito negativo, il messaggio viene conservato fino a quando il collegamento non viene ripristinato oppure l'operatore o il programma reindirizza il messaggio.

Il meccanismo mediante il quale un messaggio viene spostato dalla coda alla coda è nascosto dai programmi. Pertanto i programmi sono più semplici.

Comunicazione indipendente dal tempo

I programmi che richiedono ad altri di lavorare non devono attendere la risposta ad una richiesta. Possono eseguire altre operazioni ed elaborare la risposta quando arriva o in un secondo momento. Quando si scrive un'applicazione di messaggistica, non è necessario sapere (o preoccuparsi) quando un programma invia un messaggio o quando la destinazione è in grado di ricevere il messaggio. Il messaggio non viene perso; viene conservato dal gestore code finché la destinazione non è pronta per elaborarlo. Il messaggio rimane nella coda fino a quando non viene rimosso dal programma. Ciò significa che i programmi applicativi di invio e ricezione vengono disaccoppiati; il mittente può continuare l'elaborazione senza attendere che il destinatario confermi la ricezione del messaggio. L'applicazione di destinazione non deve essere in esecuzione quando il messaggio viene inviato. Può richiamare il messaggio dopo che è stato avviato.

Piccoli programmi

L'accodamento dei messaggi consente di utilizzare i vantaggi dell'utilizzo di programmi piccoli e autonomi. Invece di un singolo, grande programma che esegue tutte le parti di un lavoro in modo sequenziale, è possibile distribuire il lavoro su diversi programmi più piccoli e indipendenti. Il programma richiedente invia messaggi a ciascuno dei programmi separati, chiedendo loro di eseguire la loro funzione; quando ogni programma è completo, i risultati vengono inviati indietro come uno o più messaggi.

Elaborazione basata sui messaggi

Quando i messaggi arrivano su una coda, possono avviare automaticamente un'applicazione utilizzando il *trigger*. Se necessario, le applicazioni possono essere arrestate quando il messaggio (o i messaggi) sono stati elaborati.

Elaborazione basata sugli eventi

I programmi possono essere controllati in base allo stato delle code. Ad esempio, è possibile organizzare l'avvio di un programma non appena un messaggio arriva su una coda oppure specificare che il programma non viene avviato fino a quando, ad esempio, non vi sono 10 messaggi al di sopra di una determinata priorità sulla coda o 10 messaggi di qualsiasi priorità sulla coda.

Priorità messaggio

Un programma può assegnare una priorità a un messaggio quando inserisce il messaggio su una coda. Ciò determina la posizione nella coda in cui viene aggiunto il nuovo messaggio.

I programmi possono richiamare i messaggi da una coda nell'ordine in cui i messaggi si trovano nella coda oppure ottenendo un messaggio specifico. (Un programma potrebbe voler ottenere un messaggio specifico se sta cercando la risposta a una richiesta che ha inviato in precedenza.)

Sicurezza

Vengono fornite funzioni di protezione, inclusa l'autenticazione delle applicazioni quando utilizzano un gestore code, i controlli di autorizzazione quando utilizzano risorse come una coda sul gestore code e la codifica dei dati dei messaggi quando viaggiano sulla rete e quando risiedono sulle code. Per ulteriori informazioni sulla sicurezza, consultare Panoramica della sicurezza.

Integrità dei dati

L'integrità dei dati è fornita dalle unità di lavoro. La sincronizzazione dell'inizio e della fine delle unità di lavoro è completamente supportata come opzione su ogni MQGET o MQPUT, consentendo il commit o il rollback dei risultati dell'unità di lavoro. Il supporto del punto di sincronizzazione opera internamente o esternamente a IBM MQ in base al formato di coordinamento del punto di sincronizzazione selezionato per l'applicazione.

Supporto di ripristino

Per consentire il ripristino, vengono registrati tutti gli aggiornamenti IBM MQ persistenti. Se il ripristino è necessario, vengono ripristinati tutti i messaggi persistenti, viene eseguito il rollback di tutte le transazioni in corso e qualsiasi commit e backout del punto di sincronizzazione viene gestito nel modo normale del gestore del punto di sincronizzazione in controllo. Per ulteriori informazioni sui messaggi persistenti, consultare Persistenza messaggio.

Terminologia di accodamento messaggi

Queste informazioni forniscono informazioni su alcuni termini utilizzati nell'accodamento dei messaggi.

Tra di essi ricordiamo:

- Canali
- Cluster
- IBM MQ MQI client
- z/OS Accodamento all'interno del gruppo
- Messaggio
- Agent del canale messaggi
- descrittore messaggi
- Point-to-point
- Pubblicazione/sottoscrizione
- Coda
- Gestore code

- z/OS Gruppo di condivisione code
- z/OS coda condivisa
- abbonamento
- Argomento

Canali

I canali vengono utilizzati per spostare i messaggi da un gestore code a un altro e proteggono le applicazioni dai protocolli di comunicazione sottostanti. I gestori code potrebbero esistere sullo stesso sistema, su sistemi differenti sulla stessa piattaforma o su piattaforme differenti. I messaggi inviati possono avere origine da molti luoghi:

- Programmi di applicazione scritti dall'utente che trasferiscono i dati da un nodo all'altro.
- Applicazioni di gestione scritte dall'utente che utilizzano i comandi PCF o MQAI.
- Il IBM MQ Explorer.
- Gestori code che inviano messaggi di evento di strumentazione ad un altro gestore code.
- Gestori code che inviano comandi di gestione remoti ad un altro gestore code. Ad esempio, utilizzando i comandi MQSC o administrative REST API.

Per ulteriori informazioni sui canali, consultare “Definizioni canale” a pagina 32.

Cluster

Un *cluster* è una rete di gestori code associati logicamente in qualche modo.

In una rete IBM MQ che utilizza l'accodamento distribuito senza cluster, ogni gestore code è indipendente. Se un gestore code deve inviare messaggi a un altro, deve aver definito una coda di trasmissione e un canale per il gestore code remoto.

Esistono due diversi motivi per utilizzare i cluster: per ridurre la gestione del sistema e migliorare la disponibilità e il bilanciamento del carico di lavoro.

Non appena si stabilisce anche il cluster più piccolo, si beneficia di una gestione semplificata del sistema. I gestori code che fanno parte di un cluster hanno bisogno di un numero minore di definizioni e quindi il rischio di commettere un errore nelle definizioni è ridotto.

Per ulteriori informazioni sul cluster, consultare Cluster.

IBM MQ MQI client

I IBM MQ client *MQI* sono componenti installabili in modo indipendente di IBM MQ. Un client MQI consente di eseguire applicazioni IBM MQ con un protocollo di comunicazione, di interagire con uno o più server MQI (Message Queue Interface) su altre piattaforme e di connettersi ai relativi gestori code.

Per i dettagli completi su come installare e utilizzare i componenti IBM MQ MQI client , consultare i seguenti argomenti:

- AIX Installazione di un client di IBM MQ su AIX
- Linux Installazione di un client di IBM MQ su Linux®
- Windows Installazione di un client di IBM MQ su Windows
- IBM i Installazione di un client di IBM MQ su IBM i

e Configurazione delle connessioni tra il server e client.

Accodamento all'interno del gruppo

z/OS

I gestori code in un gruppo di condivisione code possono comunicare utilizzando canali normali oppure è possibile utilizzare una tecnica denominata *IGQ (intra - group queuing*), che consente di eseguire un trasferimento messaggi rapido senza definire i canali. Ciò si applica solo a IBM MQ for z/OS.

Per ulteriori informazioni sull'accodamento all'interno del gruppo, consultare “Intra-group queuing” a pagina 215.

Messaggio

Nell'accodamento dei messaggi, un messaggio è una raccolta di dati inviati da un programma e destinati a un altro programma. Vedere IBM MQ messaggi.

Per informazioni sui tipi di messaggio, vedere Tipi di messaggio.

Agent del canale messaggi

Un MCA (message channel agent) è un'estremità di un canale. Una coppia di agent del canale dei messaggi, uno che invia e uno che riceve, crea un canale e sposta i messaggi da un gestore code all'altro.

Per informazioni su come vengono utilizzati gli agent del canale dei messaggi, consultare Introduzione alla gestione delle code distribuite.

Descrittore messaggio

Un messaggio IBM MQ è costituito da informazioni di controllo e dati dell'applicazione.

Le informazioni di controllo sono definite in una struttura del descrittore del messaggio (MQMD) e contengono elementi quali:

- Il tipo di messaggio
- Un identificatore per il messaggio
- La priorità per la consegna del messaggio

La struttura e il contenuto dei dati dell'applicazione sono determinati dai programmi partecipanti, non da IBM MQ.

Per ulteriori informazioni, vedere MQMD.

Messaggistica point-to-point

Nella messaggistica point-to-point, ogni messaggio passa da un'applicazione di produzione a un'applicazione di consumo. I messaggi vengono trasferiti tramite l'applicazione di produzione inserendo i messaggi in una coda e l'applicazione di consumo li richiama da tale coda.

Pubblicazione/sottoscrizione della messaggistica

Nella messaggistica di pubblicazione / sottoscrizione, una copia di ciascun messaggio pubblicato da un'applicazione di pubblicazione viene consegnata a ciascuna applicazione interessata. Potrebbero essere presenti molte, una o nessuna applicazione interessata. Nella pubblicazione / sottoscrizione un'applicazione interessata è nota come sottoscrittore e i messaggi vengono accodati su una coda identificata da una sottoscrizione.

Per ulteriori informazioni, consultare “Pubblicazione/sottoscrizione della messaggistica” a pagina 63.

Coda

Una destinazione denominata a cui possono essere inviati i messaggi. I messaggi si accumulano sulle code fino a quando non vengono richiamate dai programmi che gestiscono tali code.

Per ulteriori informazioni, consultare “Code” a pagina 20.

Gestore code

Un *gestore code* è un programma di sistema che fornisce servizi di accodamento alle applicazioni.

Fornisce un'interfaccia di programmazione dell'applicazione in modo che i programmi possano inserire messaggi e ricevere messaggi dalle code. Un gestore code fornisce funzioni supplementari in modo che gli amministratori possano creare nuove code, modificare le proprietà delle code esistenti e controllare il funzionamento del gestore code.

Perché i servizi di accodamento messaggi IBM MQ siano disponibili su un sistema, è necessario che sia in esecuzione un gestore code. È possibile avere più di un gestore code in esecuzione su un singolo sistema (ad esempio, per separare un sistema di test da un sistema *attivo*). Per un'applicazione, ogni gestore code è identificato da un *handle di connessione* (*Hconn*).

Molte applicazioni differenti possono utilizzare i servizi del gestore code allo stesso tempo e queste applicazioni possono essere completamente non correlate. Affinché un programma utilizzi i servizi di un gestore code, è necessario stabilire una connessione a tale gestore code.

Per consentire alle applicazioni di inviare messaggi alle applicazioni connesse ad altri gestori code, è necessario che i gestori code siano in grado di comunicare tra loro. IBM MQ implementa un protocollo *store - and - forward* per assicurare la consegna sicura dei messaggi tra tali applicazioni.

Per ulteriori informazioni, consultare “Gestori code” a pagina 29.

Gruppo di condivisione code

z/OS

I gestori code che possono accedere alla stessa serie di code condivise formano un gruppo denominato *gruppo di condivisione code* (QSG). Comunicano tra loro con una CF (coupling facility) che memorizza le code condivise. Ciò si applica solo a IBM MQ for z/OS.

Per ulteriori informazioni, consultare “Shared queues and queue sharing groups” a pagina 171.

Coda condivisa

z/OS

Una *coda condivisa* è un tipo di coda locale con messaggi a cui possono accedere uno o più gestori code che si trovano in un sysplex. Non è la stessa coda condivisa da più di un'applicazione, utilizzando lo stesso gestore code. Ciò si applica solo a IBM MQ for z/OS.

Sottoscrizione

Un'applicazione di pubblicazione / sottoscrizione può registrare un interesse nei messaggi su argomenti specifici. Quando un'applicazione esegue questa operazione, è nota come sottoscrittore e il termine sottoscrizione definisce in che modo i messaggi corrispondenti vengono accodati per l'elaborazione.

Una sottoscrizione contiene informazioni sull'identità del sottoscrittore e sull'identità della coda di destinazione in cui devono essere collocate le pubblicazioni. Contiene inoltre informazioni sul modo in cui una pubblicazione deve essere inserita nella coda di destinazione.

Per ulteriori informazioni, consultare “Sottoscrittori e sottoscrizioni” a pagina 66.

Argomento

Un argomento è una stringa di caratteri che descrive l'oggetto dell'informazione pubblicata in un messaggio di pubblicazione/sottoscrizione.

Gli argomenti sono fondamentali per una consegna riuscita dei messaggi in un sistema di pubblicazione/sottoscrizione. Invece di includere un indirizzo di destinazione specifico in ciascun messaggio, un

publisher assegna un argomento a ciascun messaggio. Il gestore code fa corrispondere l'argomento a un elenco di sottoscrittori che hanno sottoscritto tale argomento e consegna il messaggio a ciascuno di questi sottoscrittori.

Per ulteriori informazioni, consultare “Argomenti” a pagina 69.

Messaggi e code

I messaggi e le code sono i componenti di base di un sistema di accodamento messaggi.

Cos' è un messaggio?

Un *messaggio* è una stringa di byte significativa per le applicazioni che lo utilizzano. I messaggi vengono utilizzati per trasferire le informazioni da un programma applicativo ad un altro (o tra diverse parti della stessa applicazione). Le applicazioni possono essere in esecuzione sulla stessa piattaforma o su piattaforme differenti.

Un messaggio IBM MQ è composto da:

- *I dati dell'applicazione.*Il contenuto e la struttura dei dati dell'applicazione sono definiti dai programmi applicativi che li utilizzano.
- *Un descrittore di messaggi.*Il descrittore del messaggio identifica il messaggio e contiene ulteriori informazioni di controllo, ad esempio il tipo di messaggio e la priorità assegnata al messaggio dall'applicazione mittente.

  Il formato del descrittore del messaggio è definito da IBM MQ. Per una descrizione completa del descrittore del messaggio, vedere MQMD - Descrittore del messaggio.
- *Proprietà del messaggio.* Metadati relativi al messaggio. Il contenuto delle proprietà del messaggio viene definito dai programmi applicativi che le utilizzano. Per ulteriori informazioni, consultare Proprietà del messaggio.

Lunghezze dei messaggi

La lunghezza massima predefinita del messaggio è 4 MB, anche se è possibile aumentarla a una lunghezza massima di 100 MB (dove 1 MB equivale a 1 048 576 byte). In pratica, la lunghezza del messaggio potrebbe essere limitata da:

- La lunghezza massima del messaggio definita per la coda di ricezione
- La lunghezza massima del messaggio definita per il gestore code
- La lunghezza massima del messaggio definita dalla coda
- La lunghezza massima del messaggio definita dall'applicazione mittente o ricevente
- La quantità di memoria disponibile per il messaggio

Potrebbero essere necessari diversi messaggi per inviare tutte le informazioni richieste da un'applicazione.

In che modo le applicazioni inviano e ricevono messaggi?

I programmi applicativi inviano e ricevono messaggi utilizzando **chiamate MQI**.

Ad esempio, per inserire un messaggio in una coda, un'applicazione:

1. Apre la coda richiesta emettendo una chiamata MQI `MQOPEN`
2. Emette una chiamata MQI `MQPUT` per inserire il messaggio nella coda

Un'altra applicazione può richiamare il messaggio dalla stessa coda emettendo una chiamata MQI `MQGET`

Per ulteriori informazioni sulle chiamate MQI, vedi Chiamate MQI.

Cos' è una coda?

Una *coda* è una struttura di dati utilizzata per memorizzare i messaggi.

Ogni coda è di un *gestore code*. Il gestore code è responsabile della gestione delle code di sua proprietà e della memorizzazione di tutti i messaggi ricevuti nelle code appropriate. I messaggi potrebbero essere inseriti nella coda dai programmi di applicazione o da un gestore code come parte della normale operazione.

Code predefinite e code dinamiche

Le code possono essere caratterizzate dal modo in cui vengono create:

- Le **code predefinite** vengono create da un amministratore utilizzando i comandi MQSC o PCF appropriati.Le code predefinite sono permanenti; esistono indipendentemente dalle applicazioni che le utilizzano e sopravvivono ai riavvii di IBM MQ .
- Le **code dinamiche** vengono create quando un'applicazione emette una richiesta MQOPEN specificando il nome di una *coda modello*. La coda creata è basata su una *definizione di coda modello*, denominata coda modello.È possibile creare una coda modello utilizzando il comando MQSC DEFINE QMODEL. Gli attributi di una coda modello (ad esempio, il numero massimo di messaggi che possono essere memorizzati su di essa) vengono ereditati da qualsiasi coda dinamica creata da essa.

  Le code modello hanno un attributo che specifica se la coda dinamica deve essere permanente o temporanea. Le code permanenti sopravvivono al riavvio dell'applicazione e del gestore code; le code temporanee vengono perse al riavvio.

Richiamo dei messaggi dalle code

Le applicazioni opportunamente autorizzate possono richiamare i messaggi da una coda in base ai seguenti algoritmi di recupero:

- FIFO (First - in - first - out).
- Priorità del messaggio, come definito nel descrittore del messaggio. I messaggi con la stessa priorità vengono richiamati su base FIFO.
- Una richiesta di programma per un messaggio specifico.

La richiesta MQGET dall'applicazione determina il metodo utilizzato.

Oggetti IBM MQ

I gestori code definiscono le proprietà degli oggetti IBM MQ . I valori di queste proprietà influenzano il modo in cui IBM MQ elabora questi oggetti. Gli oggetti vengono creati e gestiti utilizzando le interfacce e i comandi IBM MQ . Dalle proprie applicazioni, si utilizza MQI (Message Queue Interface) per controllare gli oggetti. Gli oggetti sono identificati da un MQOD (IBM MQ *object descriptor*) quando vengono indirizzati da un programma.

Gestione oggetti

La gestione degli oggetti include le seguenti attività:

- Avvio e arresto dei gestori code.
- Creazione di oggetti, in particolare code, per applicazioni.
- Visualizzazione o modifica degli attributi degli oggetti.
- Eliminazione oggetti.
- Utilizzo dei canali per creare percorsi di comunicazione per i gestori code su altri sistemi (remoti).
- Creazione di *cluster* di gestori code per semplificare il processo di gestione generale e bilanciare il carico di lavoro.

Ad eccezione delle code dinamiche, gli oggetti devono essere definiti nel gestore code prima di poterli utilizzare.

Quando si utilizza un comando IBM MQ per eseguire un'operazione di gestione degli oggetti, il gestore code verifica di disporre del livello di autorizzazione richiesto per eseguire l'operazione. Allo stesso modo, quando un'applicazione utilizza la chiamata MQOPEN per aprire un oggetto, il gestore code controlla che l'applicazione disponga del livello di autorizzazione richiesto prima di consentire l'accesso a tale oggetto. Le verifiche vengono effettuate sul nome dell'oggetto da aprire.

È possibile definire e gestire gli oggetti utilizzando i seguenti metodi:

- I comandi PCF descritti in Programmable command formats reference e Automating administration tasks
- I comandi MQSC descritti in Comandi MQSC
- z/OS Le operazioni e i pannelli di controllo IBM MQ for z/OS , descritti in Amministrazione IBM MQ for z/OS
- Linux Windows IBM MQ Explorer (solo Windows e Linux per sistemi Intel). Per ulteriori informazioni, consultare Introduzione a MQ Explorer.

È anche possibile gestire gli oggetti utilizzando i seguenti metodi:

- Comandi di controllo, immessi da una tastiera. Consultare Amministrazione IBM MQ for Multiplatforms utilizzando i comandi di controllo.
- MQAI (IBM MQ Administration Interface) richiama un programma. Vedere IBM MQ Administration Interface (MQAI).

ALW Per le sequenze di comandi IBM MQ su AIX, Linux, and Windows, è possibile utilizzare la funzionalità MQSC per eseguire una serie di comandi contenuti in un file. Per ulteriori informazioni, consultare Amministrazione IBM MQ utilizzando i comandi MQSC.

IBM i Per le sequenze di comandi IBM MQ per IBM i che si utilizzano regolarmente è possibile scrivere programmi CL. Per ulteriori informazioni, consultare Gestione IBM MQ for IBM i utilizzando i comandi CL.

z/OS Per sequenze di comandi IBM MQ for z/OS che si utilizzano regolarmente, è possibile scrivere programmi di gestione che creano messaggi contenenti comandi e che inseriscono tali messaggi nella coda di input dei comandi del sistema. Il gestore code elabora i messaggi su questa coda nello stesso modo in cui elabora i comandi immessi dalla riga comandi o dalle operazioni e dai pannelli di controllo. Questa tecnica è descritta in Scrittura di programmi per la gestione di IBM MQe illustrata nell'applicazione di esempio di Mail Manager fornita con IBM MQ for z/OS. Per una descrizione di questo esempio, vedere Programmi di esempio per IBM MQ for z/OS .

Attributi oggetto

Le proprietà di un oggetto sono definite dai suoi attributi. Alcuni possono essere specificati, altri possono essere solo visualizzati.

Ad esempio, la lunghezza massima del messaggio che una coda può contenere è definita dal relativo attributo **MaxMsgLength** ; è possibile specificare questo attributo quando viene creata una coda. L'attributo **DefinitionType** specifica come è stata creata la coda; è possibile visualizzare solo questo attributo.

In IBM MQ, esistono due modi per fare riferimento ad un attributo:

- Utilizzando il nome PCF, ad esempio, **MaxMsgLength**.
- Utilizzando il nome del comando MQSC, ad esempio MAXMSGL.

Gruppi di condivisione code

z/OS

I gestori code che possono accedere alla stessa serie di code condivise formano un gruppo denominato *gruppo di condivisione code* (QSG) e comunicano tra loro utilizzando una CF (coupling facility) che memorizza le code condivise. Si noti che un QSG non è un oggetto.

Una coda condivisa è un tipo di coda locale con messaggi a cui possono accedere uno o più gestori code che si trovano in un gruppo di condivisione code. Non è la stessa coda condivisa da più di un'applicazione, utilizzando lo stesso gestore code.

I gruppi di condivisione code hanno un nome composto da un massimo di quattro caratteri. Il nome deve essere univoco nella rete e diverso da qualsiasi altro nome di gestore code.

Importante: Le code condivise e i gruppi di condivisione code sono supportati solo su IBM MQ for z/OS.

Per ulteriori informazioni, consultare “Shared queues and queue sharing groups” a pagina 171.

Oggetti predefiniti del sistema

Gli *oggetti predefiniti di sistema* sono una serie di definizioni di oggetti che vengono create automaticamente ogni volta che viene creato un gestore code. È possibile copiare e modificare una qualsiasi di queste definizioni di oggetto da utilizzare nelle applicazioni durante l'installazione.

I nomi oggetto predefiniti hanno la radice SYSTEM; ad esempio, la coda locale predefinita è SYSTEM.DEFAULT.LOCAL.QUEUEe il canale ricevente predefinito è SYSTEM.DEF.RECEIVER. Non è possibile ridenominare questi oggetti; sono richiesti gli oggetti predefiniti di questi nomi.

Quando si definisce un oggetto, tutti gli attributi non specificati esplicitamente vengono copiati dall'oggetto predefinito appropriato. Ad esempio, se si definisce una coda locale, gli attributi non specificati vengono presi dalla coda predefinita SYSTEM.DEFAULT.LOCAL.QUEUE.

Per ulteriori informazioni, consultare Oggetti di sistema e predefiniti .

Tipi di oggetto

Molte delle attività di gestione riguardano la manipolazione di vari tipi di IBM MQ *oggetti*.

Per informazioni sulla denominazione di oggetti IBM MQ , consultare “Denominazione degli oggetti IBM MQ” a pagina 37.

Per informazioni sugli oggetti predefiniti creati su un gestore code, consultare “Oggetti predefiniti del sistema” a pagina 16.

Per informazioni sui diversi tipi di oggetti IBM MQ , consultare quanto segue:

Oggetti delle informazioni di autenticazione

Un oggetto delle informazioni di autenticazione fornisce le definizioni richieste per eseguire il controllo della revoca del certificato.

L'oggetto delle informazioni di autenticazione del gestore code fa parte del supporto IBM MQ per TLS (Transport Layer Security). Fornisce le definizioni necessarie per controllare i certificati revocati. Le autorità di certificazione revocano i certificati che non sono più attendibili.

È possibile utilizzare il comando MQSC **DEFINE AUTHINFO** per definire un oggetto delle informazioni di autenticazione. Per ulteriori informazioni sugli attributi degli oggetti delle informazioni di autenticazione, consultare **DEFINE AUTHINFO**.

È possibile utilizzare i comandi di controllo IBM MQ seguenti con un oggetto delle informazioni di autenticazione:

- **setmqaut** (concessione o revoca dell'autorizzazione)
- **dspmqaut** (visualizza autorizzazione oggetto)

- **dmpmqaut** (autorizzazioni dump)
- **rcrmqobj** (ricrea oggetto)
- **rcdmqimg** (registrazione immagine supporto)
- **dspmqfls** (visualizza nomi file)

Per una panoramica di TLS e sull'utilizzo degli oggetti delle informazioni di autenticazione, consultare Transport Layer Security (TLS) concepts e TLS security protocols in IBM MQ.

Canali

I canali sono oggetti che forniscono un percorso di comunicazione da un gestore code all'altro.

Per ulteriori informazioni, consultare “Canali” a pagina 30.

Oggetti informazioni di comunicazione

Il multicast IBM MQ offre un'affidabile messaggistica multicast a bassa latenza ed elevato fanout. È necessario un oggetto Informazioni di comunicazione (COMMINFO) per utilizzare la trasmissione multicast.

Per ulteriori informazioni, consultare “IBM MQ Multicast” a pagina 109.

Un oggetto COMMINFO è un oggetto IBM MQ che contiene gli attributi associati alla trasmissione multicast. Per ulteriori informazioni su questi attributi, consultare DEFINE COMMINFO. Per ulteriori informazioni sulla creazione di un oggetto COMMINFO, consultare Introduzione al multicast.

Listener

I *listener* sono processi che accettano le richieste di rete da altri gestori code o applicazioni client e avviano i canali associati.

I *processi listener* possono essere avviati utilizzando il comando di controllo **runmqlsr** .

I *listener* sono IBM MQ oggetti che consentono di gestire l'avvio e l'arresto dei processi listener dall'ambito di un gestore code. Definendo gli attributi di un oggetto listener si effettua quanto segue:

- Configurare il processo listener.
- Specificare se il processo listener viene avviato e arrestato automaticamente quando il gestore code viene avviato e arrestato.

Importante: z/OS Gli oggetti listener non sono supportati su IBM MQ for z/OS. Per ulteriori informazioni su come IBM MQ for z/OS implementa l'ascolto, utilizzando l'iniziatore di canali, consultare “The channel initiator on z/OS” a pagina 166.

Elenchi nomi

Un *elenco nomi* è un oggetto IBM MQ che contiene un elenco di nomi cluster, nomi coda o nomi oggetto delle informazioni di autenticazione. In un cluster, può essere utilizzato per identificare un elenco di cluster per cui il gestore code detiene i repository.

Un elenco nomi è un oggetto IBM MQ che contiene un elenco di altri oggetti IBM MQ . Di solito, gli elenchi nomi sono utilizzati dalle applicazioni, quali i monitor del trigger, dove sono utilizzati per identificare un gruppo delle code. Il vantaggio di utilizzare un elenco nomi è che viene gestito indipendentemente dalle applicazioni; può essere aggiornato senza arrestare alcuna delle applicazioni che lo utilizzano. Inoltre, se si verifica un malfunzionamento di un'applicazione, l'elenco nomi non viene influenzato e le altre applicazioni possono continuare a utilizzarlo.

Gli elenchi di nomi vengono utilizzati anche con i cluster del gestore code per gestire un elenco di cluster a cui fanno riferimento più oggetti IBM MQ .

È possibile definire e modificare gli elenchi nomi utilizzando i comandi MQSC DEFINE NAMELIST e ALTER NAMELIST .

Nota: z/OS In alternativa, su z/OS, è possibile utilizzare le operazioni IBM MQ for z/OS e i pannelli di controllo

I programmi possono utilizzare MQI per scoprire quali code sono incluse in questi elenchi nomi. L'organizzazione degli elenchi nomi è responsabilità del progettista dell'applicazione e dell'amministratore di sistema.

Per un elenco degli attributi dell'elenco nomi disponibili da utilizzare, consultare Attributi per gli elenchi nomi,

Definizioni dei processi

Gli oggetti di definizione del processo consentono alle applicazioni di essere avviate senza la necessità di un intervento dell'operatore definendo gli attributi dell'applicazione per l'utilizzo da parte del gestore code.

L'oggetto di definizione del processo definisce un'applicazione che inizia in risposta a un evento trigger su un gestore code IBM MQ . Gli attributi di definizione del processo includono l'ID applicazione, il tipo di applicazione e i dati specifici dell'applicazione. Per ulteriori informazioni, consultare *Code di iniziazione* in “Code utilizzate per scopi specifici da parte di IBM MQ” a pagina 27.

Per consentire l'avvio di un'applicazione senza la necessità di intervento dell'operatore, come descritto in Avvio di applicazioni IBM MQ utilizzando i trigger, gli attributi dell'applicazione devono essere noti al gestore code. Questi attributi sono definiti in un *oggetto definizione processo*.

L'attributo **ProcessName** è fisso quando l'oggetto viene creato. Tuttavia, è possibile modificare altri attributi utilizzando i comandi IBM MQ .

Nota: z/OS In alternativa, su z/OS, è possibile utilizzare le operazioni IBM MQ for z/OS e i pannelli di controllo.

È possibile analizzare i valori di tutti gli attributi utilizzando MQINQ - Interroga attributi oggetto.

Per un elenco degli attributi di definizione del processo disponibili per l'uso, consultare Attributi per le definizioni del processo.

Code

Una IBM MQ *coda* è un oggetto denominato su cui le applicazioni possono inserire messaggi e da cui le applicazioni possono ricevere messaggi.

Per ulteriori informazioni, consultare “Code” a pagina 20.

Gestori code

I gestori code IBM MQ forniscono i servizi di accodamento alle applicazioni e gestiscono le code che vi appartengono.

Per ulteriori informazioni, consultare “Gestori code” a pagina 29.

Servizi

Gli oggetti *Servizio* sono un metodo per definire i programmi da eseguire quando un gestore code viene avviato o arrestato.

I programmi possono essere uno dei seguenti tipi:

Server
Un *server* è un oggetto di servizio che ha il parametro SERVTYPE specificato come SERVER. Un oggetto servizio server è la definizione di un programma che verrà eseguito quando viene avviato un

gestore code specificato. È possibile eseguire contemporaneamente una sola istanza di un processo server. Durante l'esecuzione, lo stato di un processo server può essere monitorato utilizzando il comando MQSC, DISPLAY SVSTATUS. In genere gli oggetti di servizio del server sono definizioni di programmi come gestori di messaggi non recapitabili o controlli trigger, tuttavia i programmi che possono essere eseguiti non sono limitati a quelli forniti con IBM MQ. Inoltre, è possibile definire un oggetto servizio server per includere un comando che verrà eseguito quando il gestore code specificato viene arrestato per terminare il programma.

Comandi
Un *comando* è un oggetto servizio che ha il parametro SERVTYPE specificato come COMMAND. Un oggetto servizio comandi è la definizione di un programma che verrà eseguito quando viene avviato o arrestato un gestore code specificato. Più istanze di un processo di comando possono essere eseguite contemporaneamente. Gli oggetti servizio comando differiscono dagli oggetti servizio server in quanto, una volta eseguito il programma, il gestore code non monitorerà il programma. In genere, gli oggetti servizio comandi sono definizioni di programmi di breve durata che eseguono un'attività specifica, ad esempio l'avvio di una o più altre attività.

Importante: z/OS Gli oggetti di servizio non sono supportati su IBM MQ for z/OS.

Per ulteriori informazioni, vedi Utilizzo dei servizi.

Classi di memoria

z/OS

Una classe di memoria associa una o più code ad una serie di pagine.

Ciò significa che i messaggi per tale coda vengono memorizzati (in base al buffer) su tale serie di pagine.

Le classi di memoria sono supportate solo su IBM MQ for z/OS.

Per ulteriori informazioni sulle classi di archiviazione, consultare Pianificazione dell'ambiente IBM MQ su z/OS.

Oggetti argomento

Un *oggetto argomento* è un oggetto IBM MQ che permette di assegnare specifici attributi non predefiniti agli argomenti.

Un *argomento* è definito da un'applicazione che pubblica o sottoscrive una particolare *stringa argomento*. Una stringa di argomenti può specificare una gerarchia di argomenti separandoli con una barra (/). Ciò può essere visualizzato da un *albero degli argomenti*. Ad esempio, se un'applicazione esegue la pubblicazione nelle stringhe argomento `/Sport/American Football` e `/Sport/Soccer`, viene creata una struttura ad albero degli argomenti che ha un nodo parent `Sport` con due child, `American Football`e `Soccer`.

Gli argomenti ereditano i relativi attributi dal primo nodo di gestione parent trovato nella relativa struttura ad albero degli argomenti. Se non esistono nodi di argomenti di gestione in una particolare struttura ad albero degli argomenti, tutti gli argomenti ereditano i relativi attributi dall'oggetto argomento di base, SYSTEM.BASE.TOPIC.

È possibile creare un oggetto argomento in qualsiasi nodo in una struttura ad albero degli argomenti specificando la stringa argomento del nodo nell'attributo TOPICSTR dell'oggetto argomento. È anche possibile definire altri attributi per il nodo argomento di amministrazione. Per ulteriori informazioni su questi attributi, consultare I comandi MQSCo Gestione automatica mediante comandi PCF. Ciascun oggetto argomento, per impostazione predefinita, eredita gli attributi dal nodo argomento di gestione principale più vicino.

Gli oggetti argomento possono essere utilizzati anche per nascondere l'intera struttura ad albero degli argomenti agli sviluppatori di applicazioni. Se un oggetto argomento denominato `FOOTBALL.US` viene creato per l'argomento `/Sport/American Football`, un'applicazione può pubblicare o sottoscrivere l'oggetto denominato `FOOTBALL.US` invece della stringa `/Sport/American Football` con lo stesso risultato.

Se si immette un carattere #, +, / o * all'interno di una stringa di argomenti su un oggetto argomento, il carattere viene considerato come un carattere normale all'interno della stringa e viene considerato parte della stringa di argomenti associata a un oggetto argomento.

Per ulteriori informazioni sugli oggetti argomento, consultare “Pubblicazione/sottoscrizione della messaggistica” a pagina 63.

Concetti correlati

“Introduzione all'accodamento dei messaggi” a pagina 5
I prodotti IBM MQ consentono ai programmi di comunicare tra loro attraverso una rete di componenti diversi (processori, sistemi operativi, sottosistemi e protocolli di comunicazione) utilizzando un'API (application programming interface) coerente.

Riferimenti correlati

Comandi MQSC

Code

Introduzione alle code e agli attributi della coda IBM MQ .

I messaggi vengono memorizzati su una coda, in modo che se l'applicazione di inserimento è in attesa di una risposta al suo messaggio, è libera di eseguire altre operazioni mentre è in attesa di tale risposta. Le applicazioni accedono ad una coda utilizzando MQI (Message Queue Interface), descritto in Panoramica di Message Queue Interface.

Prima che un messaggio possa essere inserito in una coda, la coda deve essere già stata creata. Una coda è di proprietà di un gestore code e tale gestore code può possedere molte code. Tuttavia, ogni coda deve avere un nome univoco all'interno del gestore code.

Una coda viene gestita tramite un gestore code. Nella maggior parte dei casi, ogni coda è gestita fisicamente dal relativo gestore code, ma ciò non è evidente per un programma applicativo. IBM MQ for z/OS le code condivise possono essere gestite da qualsiasi gestore code nel gruppo di condivisione code.

Per creare una coda è possibile utilizzare i comandi IBM MQ (MQSC), i comandi PCF o le interfacce specifiche della piattaforma. Ad esempio, i pannelli di controllo e le operazioni IBM MQ for z/OS sono specifici della piattaforma.

È possibile creare code locali per lavori temporanei *dinamicamente* dall'applicazione. Ad esempio, è possibile creare code *reply - to* (che non sono necessarie una volta terminata un'applicazione). Per ulteriori informazioni, consultare “Code dinamiche e modello” a pagina 25.

Prima di utilizzare una coda, è necessario aprire la coda, specificando le operazioni da eseguire. Ad esempio, è possibile aprire una coda per:

- Solo messaggi di esplorazione (non richiamandoli)
- Richiamo dei messaggi (e condivisione dell'accesso con altri programmi o con accesso esclusivo)
- Inserimento di messaggi nella coda
- Richiesta di informazioni sugli attributi della coda
- Impostazione degli attributi della coda

Per un elenco completo delle opzioni che è possibile specificare quando si apre una coda, consultare MQOPEN-Open object.

Attributi delle code

Alcuni degli attributi di una coda vengono specificati quando la coda viene definita e non possono essere modificati successivamente (ad esempio, il tipo di coda). Gli altri attributi delle code possono essere raggruppati in quelli che possono essere modificati:

- Dal gestore code durante l'elaborazione della coda (ad esempio, la grandezza corrente di una coda)
- Solo per comandi (ad esempio, la descrizione testuale della coda)

- Dalle applicazioni, utilizzando la chiamata MQSET (ad esempio, se le operazioni di inserimento sono consentite sulla coda)

È possibile trovare i valori di tutti gli attributi utilizzando la chiamata MQINQ.

Gli attributi comuni a più di un tipo di coda sono:

QName
Il nome della coda.

QTYPE
Il tipo della coda.

QDesc
Descrizione testuale della coda.

InhibitGet
Indica se ai programmi è consentito richiamare i messaggi dalla coda. Tuttavia, non è possibile ottenere messaggi dalle code remote.

InhibitPut
Indica se ai programmi è consentito inserire messaggi nella coda.

DefPriority
Priorità predefinita per i messaggi inseriti nella coda.

DefPersistence
Persistenza predefinita per i messaggi inseriti nella coda

Ambito
Controlla se una voce per questa coda esiste anche in un servizio nomi.

z/OS L'attributo **Scope** non è supportato su z/OS .

Per una descrizione completa di questi attributi, vedere Attributi per le code.

Modi di definire le code

È possibile definire le code in IBM MQ utilizzando il comando MQSC DEFINE o il comando PCF Crea coda . I comandi specificano il tipo di coda e i suoi attributi. Ad esempio, un oggetto coda locale dispone di attributi che specificano cosa accade quando le applicazioni fanno riferimento a tale coda nelle chiamate MQI. Esempi di attributi sono:

- Se le applicazioni possono richiamare i messaggi dalla coda (GET abilitato)
- Indica se le applicazioni possono inserire messaggi nella coda (PUT abilitato)
- Se l'accesso alla coda è esclusivo per un'applicazione o condiviso tra le applicazioni
- Il numero massimo di messaggi che possono essere memorizzati nella coda contemporaneamente (profondità massima della coda)
- La lunghezza massima dei messaggi che possono essere inseriti nella coda

Esistono anche diverse interfacce specifiche della piattaforma che è possibile utilizzare per definire le code.

Concetti correlati

“Code cluster” a pagina 59
Una coda cluster è una coda ospitata da un gestore code cluster e resa disponibile ad altri gestori code del cluster.

“Code di messaggi non recapitabili” a pagina 50
La coda di messaggi non recapitabili (o coda di messaggi non recapitati) è la coda a cui vengono inviati i messaggi se non possono essere instradati alla destinazione corretta. Ogni gestore code generalmente ha una coda di messaggi non recapitabili.

Automatizzazione della gestione mediante comandi PCF
IBM MQ Console: gestione delle code

Attività correlate

Amministrazione di IBM MQ utilizzando i comandi MQSC

Creazione e configurazione di gestori code e oggetti con MQ Explorer

IBM i Gestione di IBM MQ for IBM i utilizzando i comandi CL

z/OS Origini da cui è possibile emettere comandi MQSC e PCF su IBM MQ for z/OS

Riferimenti correlati

“Comparison between shared queues and cluster queues” a pagina 59

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

Informazioni correlate

“What is a shared queue?” a pagina 171

Code locali

Le code di trasmissione, iniziazione, lettera non recapitabile, comando, valore predefinito, canale e eventi sono tipi di coda locale.

Una coda è nota a un programma come *locale* se è di proprietà del gestore code al quale è connesso il programma. È possibile ricevere messaggi dalle code locali e inserire messaggi nelle code locali.

L'oggetto di definizione della coda contiene le informazioni di definizione della coda, nonché messaggi fisici inseriti nella coda.

Ogni gestore code può avere alcune code locali che utilizza per scopi speciali:

Code di trasmissione

Quando un'applicazione invia un messaggio a una coda remota, il gestore code locale memorizza il messaggio in una speciale coda locale, denominata *coda di trasmissione*. Le applicazioni possono inserire i messaggi direttamente su una coda di trasmissione o indirettamente tramite una definizione di coda remota.

Quando un gestore code invia messaggi a un gestore code remoto, identifica la coda di trasmissione utilizzando la seguente sequenza:

1. La coda di trasmissione denominata nell'attributo XMITQ della definizione locale di una coda remota.
2. Una coda di trasmissione con lo stesso nome del gestore code remoto. Questo valore è il valore predefinito su XMITQ della definizione locale di una coda remota.
3. La coda di trasmissione denominata nell'attributo DEFXMITQ del gestore code locale.

Un *agent del canale dei messaggi* è un programma del canale associato alla coda di trasmissione e consegna il messaggio alla destinazione successiva. La destinazione successiva è il Gestore code a cui è collegato il canale dei messaggi. Non è necessariamente lo stesso gestore code della destinazione finale del messaggio. Quando il messaggio viene consegnato alla destinazione successiva, viene eliminato dalla coda di trasmissione. Il messaggio potrebbe dover passare attraverso molti gestori code durante il viaggio verso la destinazione finale. È necessario definire una coda di trasmissione in ogni gestore code lungo l'instradamento, ogni messaggio in attesa di essere trasmesso alla destinazione successiva. Una coda di trasmissione normale contiene i messaggi per la destinazione successiva, anche se i messaggi potrebbero avere destinazioni diverse. Una coda di trasmissione cluster contiene i messaggi per più destinazioni. Il `correlID` di ogni messaggio identifica il canale su cui viene inserito il messaggio per trasferirlo alla sua successiva destinazione.

È possibile definire diverse code di trasmissione in un gestore code. È possibile definire diverse code di trasmissione per la stessa destinazione, ognuna delle quali viene utilizzata per una classe di servizio differente. Ad esempio, è possibile che si desideri creare code di trasmissione diverse per messaggi di piccole dimensioni e messaggi di grandi dimensioni che vanno alla stessa destinazione. È quindi possibile trasferire i messaggi utilizzando canali di messaggi differenti, in modo che i messaggi di grandi dimensioni non trattengano i messaggi più piccoli. Tutti i messaggi per le

code cluster o gli argomenti cluster vengono posizionati nella singola coda di trasmissione cluster SYSTEM.CLUSTER.TRANSMIT.QUEUE, per impostazione predefinita. Come opzione, è possibile modificare il valore predefinito e separare il traffico di messaggi che va a gestori code di cluster differenti in code di trasmissione cluster differenti. Se si imposta l'attributo DEFCLXQ del gestore code su CHANNEL, ogni canale mittente del cluster crea una coda di trasmissione del cluster separata. In alternativa, è possibile definire manualmente le code di trasmissione del cluster per i canali mittente del cluster da utilizzare.

Le code di trasmissione possono attivare un MCA (message channel agent) per inviare i messaggi in avanti; consultare Avvio delle applicazioni IBM MQ utilizzando i trigger.

z/OS Su IBM MQ for z/OS, se si sta utilizzando l'accodamento all'interno del gruppo, la coda di trasmissione viene servita da un *agent di accodamento all'interno del gruppo*. Una coda di trasmissione condivisa viene utilizzata quando si utilizza l'accodamento all'interno del gruppo su IBM MQ for z/OS.

Code di iniziazione

Una *coda di iniziazione* è una coda locale in cui il gestore code inserisce un messaggio trigger quando si verifica un evento trigger su una coda dell'applicazione.

Un evento trigger è un evento che ha lo scopo di far sì che un programma avvii l'elaborazione di una coda. Ad esempio, un evento potrebbe essere più di 10 messaggi in arrivo. Per ulteriori informazioni su come funziona il trigger, consulta Avvio delle applicazioni IBM MQ utilizzando i trigger.

Coda di messaggi non recapitati (messaggio non consegnato)

Una *coda di messaggi non recapitabili (non recapitati)* è una coda locale in cui il gestore code inserisce i messaggi che non può consegnare.

Quando il gestore code inserisce un messaggio nella coda di messaggi non recapitabili, aggiunge un'intestazione al messaggio. Le informazioni di intestazione includono il motivo per cui il gestore code ha inserito il messaggio nella coda di messaggi non instradabili. Contiene anche la destinazione del messaggio originale, la data e l'ora in cui il gestore code ha inserito il messaggio nella coda di messaggi non recapitabili.

Le applicazioni possono anche utilizzare la coda per i messaggi che non possono consegnare. Per ulteriori informazioni, consultare Utilizzo della coda di messaggi non recapitabili.

Coda comandi di sistema

La *coda dei comandi di sistema* è una coda a cui le applicazioni opportunamente autorizzate possono inviare comandi IBM MQ . Queste code ricevono i comandi PCF, MQSC e CL, come supportato sulla piattaforma, in modo che il gestore code possa eseguirli.

z/OS Su IBM MQ for z/OS la coda è denominata SYSTEM.COMMAND.INPUT ; su altre piattaforme è denominato SYSTEM.ADMIN.COMMAND.QUEUE. I comandi accettati variano per piattaforma. Consultare Riferimento ai formati dei comandi programmabili per i dettagli.

Code predefinite di sistema

Le *code predefinite del sistema* contengono le definizioni iniziali delle code per il proprio sistema. Quando si crea una definizione di coda, il gestore code copia la definizione dalla coda predefinita di sistema appropriata. La creazione di una definizione di coda è diversa dalla creazione di una coda dinamica. La definizione della coda dinamica si basa sulla coda modello scelta come modello per la coda dinamica.

Code eventi

Le *code eventi* contengono messaggi di eventi. Questi messaggi vengono notificati dal gestore code o da un canale.

Code remote

Per un programma, una coda è *remota* se è di proprietà di un gestore code diverso rispetto a quello a cui è connesso il programma.

Se è stato stabilito un collegamento di comunicazione, un programma può inviare un messaggio a una coda remota. Un programma non può mai ricevere un messaggio da una coda remota.

L'oggetto di definizione della coda, creato quando si definisce una coda remota, contiene solo le informazioni necessarie al gestore code locale per individuare la coda a cui si desidera inviare il messaggio. Questo oggetto è noto come *definizione locale di una coda remota*. Tutti gli attributi della coda remota sono conservati dal gestore code che ne è proprietario, poiché è una coda locale per tale gestore code.

Quando si apre una coda remota, per identificare la coda è necessario specificare uno dei seguenti:

- Il nome della definizione locale che definisce la coda remota. Dal punto di vista di una applicazione, ciò equivale all'apertura di una coda locale. Non è necessario che un'applicazione sappia se una coda è locale o remota.

  Per creare una definizione locale di una coda remota su tutte le piattaforme tranne IBM i, utilizzare il comando DEFINE QREMOTE .

  IBM i Su IBM i, utilizzare il comando CRTMQMQ .
- Il nome del gestore code remoto e il nome della coda così come è noto al gestore code remoto.

Le definizioni locali delle code remote hanno tre attributi in aggiunta agli attributi comuni descritti in “Attributi delle code” a pagina 20. Questi tre attributi sono:

RemoteQName
Il nome con cui il gestore code proprietario della coda lo riconosce.

RemoteQmgrName
Il nome del gestore code proprietario.

XmitQName
Il nome della coda di trasmissione locale utilizzata durante l'inoltro dei messaggi ad altri gestori code.

Per ulteriori informazioni su questi attributi, consultare Attributi per le code.

Se si utilizza la chiamata MQINQ rispetto alla definizione locale di una coda remota, il gestore code restituisce solo gli attributi della definizione locale, ovvero il nome della coda remota, il nome del gestore code remoto e il nome della coda di trasmissione, non gli attributi della coda locale corrispondente nel sistema remoto.

Vedere anche Code di trasmissione.

Code alias

Una *coda alias* è un oggetto IBM MQ che è possibile utilizzare per accedere a un'altra coda o a un altro argomento. Ciò significa che più di un programma può gestire la stessa coda, accedendo ad essa utilizzando nomi diversi.

La coda risultante dalla risoluzione di un nome alias, noto come coda di base, può essere uno dei seguenti tipi di code, come supportato dalla piattaforma:

- Una coda locale
- La definizione locale di una coda remota.
- z/OS Una coda condivisa, che è un tipo di coda locale disponibile solo su IBM MQ for z/OS.
- Una coda predefinita
- Una coda dinamica

Un nome alias può anche essere risolto in un argomento. Se un'applicazione attualmente inserisce messaggi in una coda, è possibile effettuare la pubblicazione in un argomento rendendo il nome della coda un alias per l'argomento. Non è necessaria alcuna modifica al codice dell'applicazione.

Nota: Un alias non può essere risolto direttamente in un altro alias sullo stesso gestore code.

Un esempio di utilizzo delle code alias è per un responsabile di sistema per fornire autorizzazioni di accesso differenti al nome coda di base (ovvero, la coda in cui l'alias si risolve) e al nome coda alias. Ciò significa che un programma o un utente può essere autorizzato ad utilizzare la coda alias, ma non la coda di base.

In alternativa, l'autorizzazione può essere impostata per impedire le operazioni di inserimento per il nome alias, ma per consentire la coda di base.

In alcune applicazioni, l'utilizzo di code alias significa che gli amministratori di sistema possono modificare facilmente la definizione di un oggetto coda alias senza dover modificare l'applicazione.

IBM MQ effettua i controlli di autorizzazione rispetto al nome alias quando i programmi tentano di utilizzare tale nome. Non controlla che il programma sia autorizzato ad accedere al nome su cui si risolve l'alias. Un programma può quindi essere autorizzato ad accedere ad un nome coda alias, ma non al nome coda risolto.

Oltre agli attributi generali della coda descritti in “Code” a pagina 20, le code alias hanno un attributo **BaseQName** . Questo è il nome della coda di base in cui viene risolto il nome alias. Per una descrizione più completa di questo attributo, consultare BaseQName (MQCHAR48).

Gli attributi *InhibitGet* e **InhibitPut** (consultare “Code” a pagina 20) delle code alias appartengono al nome alias. Ad esempio, se il nome della coda alias ALIAS1 si risolve nel nome della coda di base BASE, le inibizioni su ALIAS1 influiscono solo su ALIAS1 e BASE non è inibito. Tuttavia, le inibizioni su BASE influenzano anche ALIAS1.

Gli attributi *DefPriority* e **DefPersistence** appartengono anche al nome alias. Quindi, ad esempio, è possibile assegnare diverse priorità predefinite a diversi alias della stessa coda base. Inoltre, è possibile modificare queste priorità senza dover modificare le applicazioni che utilizzano gli alias.

Code dinamiche e modello

Queste informazioni forniscono informazioni dettagliate sulle code dinamiche, sulle proprietà delle code dinamiche temporanee e permanenti, sugli utilizzi delle code dinamiche, su alcune considerazioni relative all'uso delle code dinamiche e sulle code modello.

Quando un programma applicativo emette una chiamata MQOPEN per l'apertura di una coda modello, il gestore code crea dinamicamente un'istanza di una coda locale con gli stessi attributi della coda modello. In base al valore del campo *DefinitionType* della coda modello, il gestore code crea una coda dinamica temporanea o permanente (consultare Creazione di code dinamiche).

Proprietà delle code dinamiche temporanee

Le *code dinamiche temporanee* hanno le seguenti proprietà:

- z/OS Non possono essere code condivise, accessibili dai gestori code in un gruppo di condivisione code.

  Notare che i gruppi di condivisione code sono disponibili solo su IBM MQ for z/OS.
- Contengono solo messaggi non persistenti.
- Sono irrecuperabili.
- Vengono eliminati quando il gestore code viene avviato.
- Vengono eliminati quando l'applicazione che ha emesso la chiamata MQOPEN che ha creato la coda chiude la coda o termina.
  - Se sulla coda sono presenti messaggi di cui è stato eseguito il commit, vengono eliminati.
  - Se sono presenti chiamate MQGET, MQPUT o MQPUT1 non sottoposte a commit in attesa rispetto alla coda in questo momento, la coda viene contrassegnata come eliminata logicamente e viene eliminata fisicamente (dopo che è stato eseguito il commit) come parte dell'elaborazione di chiusura o quando l'applicazione termina.

- Se la coda è in uso in questo momento (dalla creazione o da un'altra applicazione), la coda viene contrassegnata come logicamente eliminata e viene eliminata fisicamente solo quando viene chiusa dall'ultima applicazione che utilizza la coda.
- I tentativi di accedere a una coda eliminata logicamente (non per chiuderla) hanno esito negativo con codice motivo MQRC_Q_DELETED.
- MQCO_NONE, MQCO_DELETE e MQCO_DELETE_PURGE vengono tutti trattati come MQCO_NONE quando vengono specificati su una chiamata MQCLOSE per la chiamata MQOPEN corrispondente che ha creato la coda.

Proprietà delle code dinamiche permanenti

Le *code dinamiche permanenti* hanno le seguenti proprietà:

- Contengono messaggi persistenti o non persistenti.
- Sono recuperabili in caso di errori di sistema.
- Vengono eliminati quando un'applicazione (non necessariamente quella che ha emesso la chiamata MQOPEN che ha creato la coda) chiude correttamente la coda utilizzando l'opzione MQCO_DELETE o MQCO_DELETE_PURGE.
  - Una richiesta di chiusura con l'opzione MQCO_DELETE ha esito negativo se sulla coda sono ancora presenti messaggi (di cui è stato eseguito il commit o di cui non è stato eseguito il commit). Una richiesta di chiusura con l'opzione MQCO_DELETE_PURGE ha esito positivo anche se ci sono messaggi di cui è stato eseguito il commit sulla coda (i messaggi vengono eliminati come parte della chiusura), ma ha esito negativo se sono presenti chiamate MQGET, MQPUT o MQPUT1 non sottoposte a commit in sospeso sulla coda.
  - Se la richiesta di eliminazione ha esito positivo, ma la coda è in uso (dalla creazione o da un'altra applicazione), la coda viene contrassegnata come eliminata logicamente e viene eliminata fisicamente solo quando viene chiusa dall'ultima applicazione che utilizza la coda.
- Non vengono eliminati se chiusi da un'applicazione che non è autorizzata a eliminare la coda, a meno che l'applicazione di chiusura non abbia emesso la chiamata MQOPEN che ha creato la coda. I controlli di autorizzazione vengono eseguiti sull'identificativo utente (o sull'identificativo utente alternativo se è stato specificato MQOO_ALTERNATE_USER_AUTHORITY) utilizzato per convalidare la chiamata MQOPEN corrispondente.
- Possono essere eliminati allo stesso modo di una coda normale.

Utilizzi delle code dinamiche

È possibile utilizzare code dinamiche per:

- Le applicazioni che non richiedono code da conservare dopo la chiusura dell'applicazione.
- Applicazioni che richiedono che le risposte ai messaggi vengano elaborate da un'altra applicazione. Tali applicazioni possono creare in modo dinamico una coda di risposta aprendo una coda modello. Ad esempio, un'applicazione client può:
  1. Creare una coda dinamica.
  2. Fornire il suo nome nel campo **ReplyToQ** della struttura descrittore del messaggio di richiesta.
  3. Inserire la richiesta su una coda elaborata da un server.

Il server può quindi posizionare il messaggio di risposta sulla coda di risposta. Infine, il client potrebbe elaborare la risposta e chiudere la coda di risposta con l'opzione di eliminazione.

Considerazioni sull'utilizzo di code dinamiche

Considerare i seguenti punti quando si utilizzano le code dinamiche:

- In un modello client - server, ogni client deve creare e utilizzare la propria coda di risposta dinamica. Se una coda di risposta dinamica è condivisa tra più di un client, l'eliminazione della coda di risposta potrebbe essere ritardata a causa della presenza di un'attività di cui non è stato eseguito il commit in sospeso sulla coda o perché la coda è utilizzata da un altro client. Inoltre, la coda potrebbe essere

contrassegnata come eliminata logicamente e inaccessibile per le successive richieste API (diverse da MQCLOSE).

- Se l'ambiente dell'applicazione richiede che le code dinamiche siano condivise tra le applicazioni, assicurarsi che la coda sia chiusa (con l'opzione di eliminazione) solo quando è stato eseguito il commit di tutta l'attività rispetto alla coda. Deve essere l'ultimo utente. Ciò garantisce che l'eliminazione della coda non venga ritardata e riduce al minimo il periodo in cui la coda è inaccessibile perché è stata contrassegnata come logicamente eliminata.

Code modello

Una *coda modello* è un template di una definizione di coda che si utilizza quando si crea una coda dinamica.

È possibile creare dinamicamente una coda locale da un programma IBM MQ , denominando la coda modello che si desidera utilizzare come modello per gli attributi della coda. A quel punto è possibile modificare alcuni attributi della nuova coda. Tuttavia, non è possibile modificare **DefinitionType**. Se, ad esempio, si richiede una coda permanente, selezionare una coda modello con il tipo di definizione impostato su permanente. Alcune applicazioni di conversazione possono utilizzare code dinamiche per conservare le risposte alle loro query, poiché probabilmente non hanno bisogno di gestire queste code dopo aver elaborato le risposte.

Specificare il nome di una coda modello nel MQOD (*object descriptor*) della chiamata MQOPEN. Utilizzando gli attributi della coda modello, il gestore code crea dinamicamente una coda locale per l'utente.

È possibile specificare un nome (completo) per la coda dinamica o la radice di un nome (ad esempio, ABC) e consentire al gestore code di aggiungere una parte univoca oppure è possibile consentire al gestore code di assegnare un nome univoco completo. Se il gestore code assegna il nome, lo inserisce nella struttura MQOD.

Non è possibile emettere una chiamata MQPUT1 direttamente a una coda modello, ma è possibile emettere una MQPUT1 per la coda dinamica creata aprendo una coda modello.

MQSET e MQINQ non possono essere emessi rispetto a una coda modello. L'apertura di una coda modello con MQOO_INQUIRE o MQOO_SET comporta l'esecuzione di chiamate MQINQ e MQSET successive rispetto alla coda creata dinamicamente.

Gli attributi di una coda modello sono un sottoinsieme di quelli di una coda locale. Per una descrizione più completa, consultare Attributi per le code.

Code utilizzate per scopi specifici da parte di IBM MQ

IBM MQ utilizza alcune code locali per scopi specifici correlati al suo funzionamento.

È necessario definire queste code prima che IBM MQ possa utilizzarle.

Code di iniziazione

Le code di iniziazione sono code utilizzate nel trigger. Un gestore code inserisce un messaggio trigger su una coda di iniziazione quando si verifica un evento trigger. Un evento trigger è una combinazione logica di condizioni rilevate da un gestore code. Ad esempio, un evento trigger potrebbe essere generato quando il numero di messaggi su una coda raggiunge una profondità predefinita. Questo evento fa sì che il gestore code inserisca un messaggio trigger su una coda di avvio specificata. Questo messaggio trigger viene richiamato da un *controllo trigger*, un'applicazione speciale che controlla una coda di iniziazione. Il controllo trigger, quindi, avvia il programma applicativo specificato nel messaggio trigger.

Se un gestore code deve utilizzare l'attivazione, è necessario definire almeno una coda di avvio per tale gestore code. Consultare Gestione degli oggetti per l'attivazione di, runmqtrme Avvio delle applicazioni IBM MQ mediante i trigger

Code di trasmissione

Le code di trasmissione sono code che memorizzano temporaneamente i messaggi destinati a un gestore code remoto. È necessario definire almeno una coda di trasmissione per ogni gestore code

remoto a cui il gestore code locale deve inviare direttamente i messaggi. Queste code vengono utilizzate anche nell'amministrazione remota; consultare Gestione remota da un gestore code locale. Per informazioni sull'utilizzo delle code di trasmissione nell'accodamento distribuito, consultare IBM MQ tecniche di accodamento distribuito.

Ogni gestore code può avere una coda di trasmissione predefinita. Se un gestore code che non fa parte di un cluster inserisce un messaggio in una coda remota, l'azione predefinita consiste nell'utilizzare la coda di trasmissione predefinita. Se è presente una coda di trasmissione con lo stesso nome del gestore code di destinazione, il messaggio viene inserito in tale coda di trasmissione. Se esiste una definizione di alias del gestore code, in cui il parametro **RQMNAME** corrisponde al gestore code di destinazione e viene specificato il parametro **XMITQ** , il messaggio viene posizionato nella coda di trasmissione denominata da **XMITQ**. Se non esiste alcun parametro **XMITQ** , il messaggio viene inserito nella coda locale indicata nel messaggio.

Code di trasmissione cluster
Ogni gestore code all'interno di un cluster ha una coda di trasmissione cluster denominata SYSTEM.CLUSTER.TRANSMIT.QUEUEe una coda di trasmissione cluster modello, SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. Le definizioni di queste code vengono create per impostazione predefinita quando si definisce un gestore code. Se l'attributo del gestore code, **DEFCLXQ**, è impostato su CHANNEL, viene creata automaticamente una coda di trasmissione del cluster dinamica permanente per ogni canale mittente del cluster creato. Le code sono denominate SYSTEM.CLUSTER.TRANSMIT. *ChannelName*. È anche possibile definire manualmente le code di trasmissione del cluster.

Un gestore code che fa parte del cluster invia messaggi su una di tali code ad altri gestori code che si trovano nello stesso cluster.

Durante la risoluzione dei nomi, una coda di trasmissione del cluster ha la precedenza sulla coda di trasmissione predefinita e una specifica coda di trasmissione del cluster ha la precedenza su SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Code di messaggi non recapitabili
Una coda di messaggi non recapitabili (undelivered-message) è una coda che memorizza i messaggi che non possono essere instradati alle destinazioni corrette. Un messaggio non può essere instradato quando, ad esempio, la coda di destinazione è piena. La coda di messaggi non instradabili fornita è denominata SYSTEM.DEAD.LETTER.QUEUE.

Per l'accodamento distribuito, definire una coda di messaggi non instradabili su ogni gestore code coinvolto.

Code comandi
La coda comandi, SYSTEM.ADMIN.COMMAND.QUEUE, è una coda locale a cui le applicazioni adeguatamente autorizzate possono inviare comandi MQSC per l'elaborazione. Questi comandi vengono quindi richiamati da un componente IBM MQ denominato server dei comandi. Il server dei comandi convalida i comandi, passa quelli validi per l'elaborazione da parte del gestore code e restituisce tutte le risposte alla coda di risposta appropriata.

Una coda comandi viene creata automaticamente per ogni gestore code quando viene creato tale gestore code.

Code di risposta
Quando un'applicazione invia un messaggio di richiesta, l'applicazione che riceve il messaggio può restituire un messaggio di risposta all'applicazione mittente. Questo messaggio viene inserito in una coda, denominata coda di risposta, che normalmente è una coda locale all'applicazione di invio. Il nome della coda di risposta è specificato dall'applicazione mittente come parte del descrittore del messaggio.

Code eventi
Gli eventi di strumentazione possono essere utilizzati per monitorare i gestori code indipendentemente dalle applicazioni MQI.

Quando si verifica un evento di strumentazione, il gestore code inserisce un messaggio evento su una coda eventi. Questo messaggio può quindi essere letto da un'applicazione di controllo, che potrebbe informare un amministratore o avviare alcune azioni correttive se l'evento indica un problema.

Nota: Gli eventi trigger sono differenti dagli eventi di strumentazione. Gli eventi trigger non sono causati dalle stesse condizioni e non generano messaggi di evento.

Per ulteriori informazioni sugli eventi di strumentazione, vedi Eventi di strumentazione.

Gestori code

Introduzione ai *gestori code* e ai servizi di accodamento che forniscono alle applicazioni.

Un programma deve avere una connessione a un gestore code prima di poter utilizzare i servizi di tale gestore code. Un programma può effettuare questa connessione in modo esplicito (utilizzando la chiamata MQCONN o MQCONNX) oppure la connessione potrebbe essere effettuata in modo implicito (ciò dipende dalla piattaforma e dall'ambiente in cui il programma è in esecuzione).

Un gestore code IBM MQ assicura le seguenti azioni:

- Gli attributi dell'oggetto vengono modificati in base ai comandi ricevuti.
- Gli eventi speciali come gli eventi trigger o gli eventi di strumentazione vengono generati quando vengono soddisfatte le condizioni appropriate.
- I messaggi vengono inseriti sulla coda corretta, come richiesto dall'applicazione che effettua la chiamata MQPUT . L'applicazione viene informata se ciò non è possibile e viene fornito un codice di errore appropriato.

Ciascuna coda appartiene a un singolo gestore code e viene detta *coda locale* per tale gestore code. Il gestore code a cui è connessa un'applicazione viene detto *gestore code locale* per tale applicazione. Per l'applicazione, le code che appartengono al gestore code locale sono code locali.

Una *coda remota* è una coda che appartiene a un altro gestore code. Un *gestore code remoto* è un gestore code diverso dal gestore code locale. Un gestore code remoto può esistere su una macchina remota nella rete o sulla stessa macchina del gestore code locale. IBM MQ supporta più gestori code sulla stessa macchina.

Un oggetto gestore code può essere utilizzato in alcune chiamate MQI. Ad esempio, è possibile richiedere informazioni sugli attributi dell'oggetto gestore code utilizzando la chiamata MQI MQINQ.

Attributi dei gestori code

A ciascun gestore code è associata una serie di attributi (o proprietà) che ne definiscono le caratteristiche.Alcuni degli attributi di un gestore code vengono corretti quando viene creato; è possibile modificarne altri utilizzando i comandi IBM MQ . È possibile richiedere informazioni sui valori di tutti gli attributi, eccetto quelli utilizzati per la crittografia TLS (Transport Layer Security), utilizzando la chiamata MQINQ.

Gli attributi fissi includono:

- Il nome del gestore code
- La piattaforma su cui viene eseguito il gestore code (ad esempio, Windows)
- Il livello di comandi di controllo del sistema supportato dal gestore code
- La priorità massima che è possibile assegnare ai messaggi elaborati dal gestore code
- Il nome della coda a cui i programmi possono inviare comandi IBM MQ
- La lunghezza massima dei messaggi che il gestore code può elaborare z/OS (fissa solo in IBM MQ for z/OS)
- Se il gestore code supporta il punto di sincronizzazione quando i programmi inserono e ricevono i messaggi

Gli attributi *modificabili* includono:

- Una descrizione testuale del gestore code
- L'identificativo della serie di caratteri che il gestore code utilizza per le stringhe di caratteri quando elabora le chiamate MQI
- L'intervallo di tempo utilizzato dal gestore code per limitare il numero di messaggi trigger
- z/OS L'intervallo di tempo utilizzato dal gestore code per determinare la frequenza con cui le code devono essere sottoposte a scansione per i messaggi scaduti (solo IBM MQ for z/OS)
- Il nome della coda di messaggi non recapitabili (messaggi non recapitati) del gestore code
- Il nome della coda di trasmissione predefinita del gestore code
- Il numero massimo di handle aperti per una connessione
- Abilitazione e disabilitazione di varie categorie di report di eventi
- Il numero massimo di messaggi di cui non è stato eseguito il commit all'interno di un'unità di lavoro

Gestori code e gestione del carico di lavoro

È possibile impostare un cluster di gestori code con più di una definizione per la stessa coda (ad esempio, i gestori code nel cluster potrebbero essere cloni l'uno dell'altro). I messaggi per una particolare coda possono essere gestiti da qualsiasi gestore code che ospita un'istanza della coda. Un algoritmo di gestione del carico di lavoro decide quale gestore code gestisce il messaggio e quindi distribuisce il carico di lavoro tra i gestori code; per ulteriori informazioni, fare riferimento a Algoritmo di gestione del carico di lavoro del cluster .

Canali

Un *canale* è un collegamento di comunicazione logico, utilizzato dai gestori code distribuiti, tra un server IBM MQ MQI client e un server IBM MQ o tra due server IBM MQ .

I canali vengono utilizzati per spostare i messaggi da un gestore code a un altro e proteggono le applicazioni dai protocolli di comunicazione sottostanti. I gestori code potrebbero esistere sullo stesso sistema, su sistemi differenti sulla stessa piattaforma o su piattaforme differenti. I messaggi inviati possono avere origine da molti luoghi:

- Programmi di applicazione scritti dall'utente che trasferiscono i dati da un nodo all'altro.
- Applicazioni di gestione scritte dall'utente che utilizzano i comandi PCF o MQAI.
- Il IBM MQ Explorer.
- Gestori code che inviano messaggi di evento di strumentazione ad un altro gestore code.
- Gestori code che inviano comandi di gestione remoti ad un altro gestore code. Ad esempio, utilizzando i comandi MQSC o administrative REST API.

Un canale ha due definizioni: una ad ogni estremità della connessione. Affinché i gestori code comunichino tra loro, è necessario definire un oggetto canale sul gestore code che deve inviare i messaggi e un altro, complementare, sul gestore code che deve riceverli. Lo stesso *nome canale* deve essere utilizzato ad ogni estremità della connessione e il tipo di canale utilizzato deve essere compatibile.

Ci sono tre categorie di canali in IBM MQ, con diversi tipi di canali all'interno di queste categorie:

- I canali di messaggi, che sono unidirezionali, e trasferiscono i messaggi da un gestore code all'altro.
- I canali MQI, che sono bidirezionali e trasferiscono le chiamate MQI da un IBM MQ MQI client a un gestore code e le risposte da un gestore code a un client IBM MQ .
- I canali AMQP, che sono bidirezionali e collegano il client AMQP a un gestore code su una macchina server. IBM MQ utilizza canali AMQP per trasferire chiamate e risposte AMQP tra applicazioni AMQP e gestori code

Canali dei messaggi

Lo scopo di un canale di messaggi è di trasferire i messaggi da un gestore code all'altro. I canali di messaggi non sono richiesti dall'ambiente del server client.

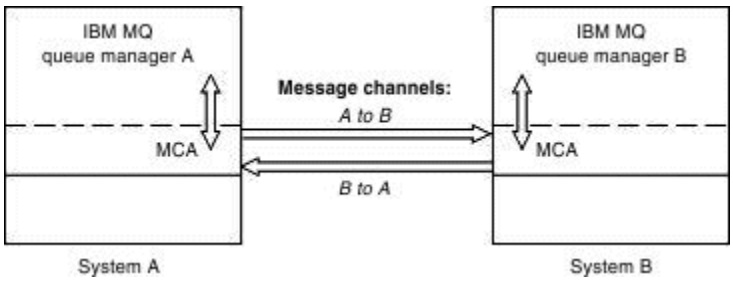


Figura 2. Canali di messaggi tra due gestori code

Un canale di messaggi è un collegamento unidirezionale. Se si desidera che un gestore code remoto risponda ai messaggi inviati da un gestore code locale, è necessario impostare un secondo canale per inviare le risposte al gestore code locale.

Un canale dei messaggi connette due gestori code utilizzando gli MCA (*message channel agent*). Esiste un agente del canale dei messaggi ad ogni estremità di un canale. È possibile consentire a un MCA di trasferire i messaggi utilizzando più thread. Questo processo è noto come *pipelining*. Pipelining consente all'MCA di trasferire i messaggi in modo più efficiente, migliorando le prestazioni del canale. Per ulteriori informazioni sulla pipeline, vedi Attributi dei canali.

Per ulteriori informazioni sui canali, consultare Chiamate di uscita del canale e strutture datie “Componenti di accodamento distribuiti” a pagina 47.

Canali MQI

Un canale MQI (Message Queue Interface) connette un IBM MQ MQI client a un gestore code su una macchina server e viene stabilito quando si emette una chiamata `MQCONN` o `MQCONNX` da un'applicazione IBM MQ MQI client .

Si tratta di un collegamento bidirezionale e viene utilizzato solo per il trasferimento di chiamate e risposte MQI, incluse le chiamate `MQPUT` che contengono dati di messaggio e le chiamate `MQGET` che risultano nella restituzione dei dati di messaggio. Esistono diversi modi per creare e utilizzare le definizioni di canale (vedere Definizione dei canali MQI).

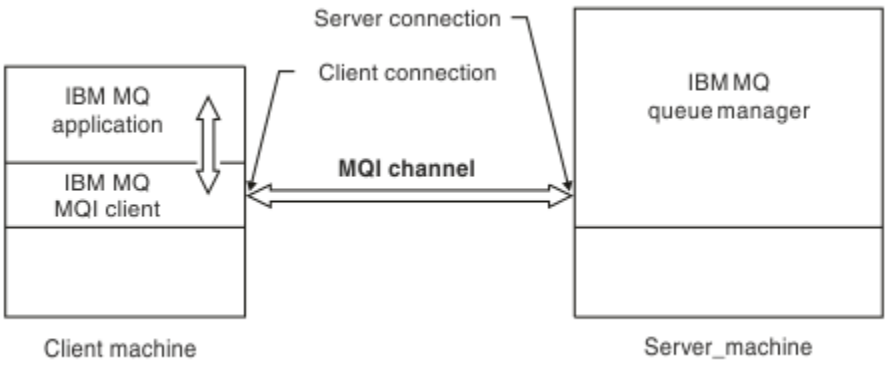


Figura 3. Connessione client e connessione server su un canale MQI

z/OS Un canale MQI può essere utilizzato per collegare un client a un singolo gestore code o a un gestore code che fa parte di un gruppo di condivisione code (consultare Connessione di un client a un gruppo di condivisione code).

Ci sono due tipi di canale per definizioni di canale MQI. Definiscono il canale MQI bidirezionale.

Canale connessione client
Questo tipo è per IBM MQ MQI client.

Canale di connessione server
Questo tipo è per il server che esegue il gestore code, con cui l'applicazione IBM MQ , in esecuzione in un ambiente IBM MQ MQI client , deve comunicare.

Canali AMQP

Esiste un solo tipo di canale AMQP.

Il canale è utilizzato per connettere un'applicazione di messaggistica AMQP a un gestore code, abilitando l'applicazione allo scambio di messaggi con le applicazioni IBM MQ. Un canale AMQP consente di sviluppare un'applicazione utilizzando MQ Light e di distribuirla quindi come un'applicazione enterprise, avvalendosi delle funzioni di livello enterprise fornite da IBM MQ.

Canali di connessione client

I *canali di connessione client* sono oggetti che forniscono un percorso di comunicazione da un IBM MQ MQI client a un gestore code.

I canali di connessione client vengono utilizzati nell'accodamento distribuito per spostare i messaggi tra un gestore code e un client. Proteggono le applicazioni dai protocolli di comunicazione sottostanti. Il client potrebbe esistere sulla stessa piattaforma o su una piattaforma diversa dal gestore code.

Definizioni canale

Consultare “Definizioni canale” a pagina 32 per descrizioni di ciascun tipo di canale.

Concetti correlati

“Accodamento distribuito e cluster” a pagina 43
L'accodamento distribuito significa inviare messaggi da un gestore code ad un altro. Il gestore code di ricezione può trovarsi sulla stessa macchina o su un'altra; nelle vicinanze o dall'altra parte del mondo. Può essere in esecuzione sulla stessa piattaforma del gestore code locale o su una qualsiasi delle piattaforme supportate da IBM MQ. È possibile definire manualmente tutte le connessioni in un ambiente di accodamento distribuito oppure è possibile creare un cluster e consentire a IBM MQ di definire gran parte dei dettagli di connessione.

Panoramica su Interfaccia coda messaggi

Attività correlate

Gestione di oggetti IBM MQ remoti
Arresto dei canali MQI
Configurazione delle connessioni tra il server e il client

Riferimenti correlati

Chiamate di uscita canale e strutture dati
“Comunicazioni” a pagina 36
IBM MQ MQI clients utilizzare i canali MQI per comunicare con il server.

Definizioni canale

Tabelle che descrivono i diversi tipi di canali di messaggi e canali MQI utilizzati da IBM MQ .

Quando si parla di canali di messaggi, la parola canale spesso viene utilizzata come sinonimo per la definizione di canale. Di solito il contesto aiuta a capire se si parla di una canale completo, che ha due estremità, o di una definizione di canale, che ha una estremità sola.

Canali dei messaggi

Le definizioni dei canali di messaggi possono essere di uno dei seguenti tipi:

Tipo di definizione di canali dei messaggi	Descrizione
Mittente	Un canale mittente è un canale dei messaggi che il gestore code utilizza per inviare i messaggi ad altri gestori code. Per inviare i messaggi mediante un canale mittente, è necessario creare, sull'altro gestore code, un canale ricevente con lo stesso nome del canale mittente. È anche possibile utilizzare i canali del mittente con i canali del richiedente, se si implementa un meccanismo di "callback".
Server	Un canale server è un canale di messaggi che il gestore code utilizza per inviare i messaggi ad altri gestori code. Per inviare i messaggi mediante un canale server, è anche necessario creare, sull'altro gestore code, un canale ricevente con lo stesso nome del canale server. I canali server possono essere utilizzati anche con i canali richiedenti. In questo caso, la definizione di canale richiedente richiede l'avvio della definizione di canale server all'altra estremità del canale. Il server invia i messaggi al richiedente. Quindi, il server inizia la comunicazione, attiva fino a che il nome della connessione del canale partner è noto.
Ricevente	Un canale ricevente è un canale di messaggi utilizzato dal gestore code per ricevere i messaggi dagli altri gestori code. Per ricevere i messaggi mediante un canale ricevente, è anche necessario creare, sull'altro gestore code, un canale mittente o server con lo stesso nome di questo canale ricevente.
Richiedente	Un canale Richiedente è un canale di messaggi utilizzato dal gestore code per ricevere messaggi da altri gestori code. Un canale RIchiedente può richiedere che sia avviato il canale partner definito all'estremità remota. Se il canale partner è un canale Server, il canale Server accetta la richiesta di avvio e inizia a inviare messaggi, dalla coda di trasmissione identificata nella definizione di canale Server, al canale Richiedente. Se il canale partner è un canale Mittente, il canale Mittente accetta la richiesta di avvio ma chiude quindi la connessione con il Richiedente. Il canale Mittente viene quindi avviato, negozia una sessione con il canale Richiedente partner e inizia a inviare messaggi dalla coda di trasmissione identificata nella definizione del canale Mittente. Quest'ultimo caso fornisce fondamentalmente un meccanismo di richiamo, dal momento che il canale Richiedente richiede al canale Mittente di richiamare.

Tipo di definizione di canali dei messaggi	Descrizione
Mittente del cluster	Una definizione di canale mittente del cluster (CLUSSDR) definisce l'estremità di invio su cui il gestore code di un cluster può inviare le informazioni sul cluster a uno dei repository completi. Il canale mittente del cluster viene utilizzato per notificare al repository le modifiche apportate allo stato del gestore code, ad esempio nel caso dell'aggiunta o della rimozione di una coda. Essa viene utilizzata anche per trasmettere i messaggi. I gestori code con repository completo hanno canali mittente cluster che fanno riferimento l'uno all'altro. Tali canali vengono utilizzati per comunicare le modifiche allo stato del cluster. Non è particolarmente importante conoscere a quale repository completo una definizione del canale CLUSSDR del gestore code fa riferimento. Una volta stabilito il contatto iniziale, vengono definiti automaticamente ulteriori oggetti del gestore code del cluster in modo che il gestore code possa inviare le informazioni sul cluster a ogni repository completo e i messaggi a ogni gestore code.
Ricevente del cluster	Una definizione di canale ricevente del cluster (CLUSRCVR) definisce l'estremità di ricezione su cui il gestore code di un cluster può ricevere le informazioni sul cluster da altri gestori code. Un canale ricevente del cluster trasporta inoltre le informazioni sul destinate al repository. Definendo il canale ricevente del cluster, il gestore code indica agli altri gestori code che è disponibile per la ricezione di messaggi. È necessario disporre di almeno un canale ricevente del cluster per ogni gestore code del cluster.

Per ogni canale, è necessario definire entrambe le estremità in modo da avere una definizione di canale per ognuna di esse. Le due estremità del canale devono essere di tipo compatibile.

È possibile avere le seguenti combinazioni di definizioni di canali:

- Mittente-Ricevente
- Server-Ricevente
- Richiedente-Server
- Richiedente-Mittente (callback)
- Mittente del cluster-Ricevente del cluster

Agente canale dei messaggi (MCA, message channel agent)

Ogni definizione di canale che viene creata appartiene a un determinato gestore code. Un gestore code può avere diversi canali, dello stesso tipo o di tipo differente. A ogni estremità del canale è presente un programma, un MCA (message channel agent), ovvero un agent dei messaggi dei canali. A un'estremità del canale, l'MCA chiamante prende i messaggi dalla coda di trasmissione e li inoltra sul canale. All'altra estremità del canale, l'MCA rispondente riceve i messaggi e li consegna al gestore code remoto.

Un MCA chiamante può essere associato a un canale mittente, server o richiedente. Un MCA rispondente può essere associato a qualsiasi tipo di canale.

IBM MQ supporta le seguenti combinazioni di tipi di canale alle due estremità di una connessione:

Chiamante		Direzione del flusso di messaggi	Rispondente	
Tipo di canale	**È necessario il listener?**		**È necessario il listener?**	**Tipo di canale**
Mittente	No	Dal chiamante al rispondente	Sì	Ricevente
Server	No	Dal chiamante al rispondente	Sì	Ricevente
Server	No	Dal chiamante al rispondente	Sì	Richiedente
Richiedente	No	Dal rispondente al chiamante	Sì	Server
Richiedente	Sì	Dal rispondente al chiamante	Sì	Mittente

Canali MQI

I canali MQI possono essere di uno dei seguenti tipi:

Tipo canale MQI	Descrizione
Connessione server	Un canale di connessione server è un canale MQI bidirezionale utilizzato per connettere un client IBM MQ a un server IBM MQ. Il canale di connessione server è l'estremità server del canale.
Connessione client	Un canale di connessione client è un canale MQI bidirezionale utilizzato per connettere un client IBM MQ a un server IBM MQ. IBM MQ Explorer utilizza anche le connessioni client per connettersi ai gestori code remoti. Il canale di connessione client è l'estremità client del canale. Quando si crea un canale di connessione client, viene creato un file sul computer su cui è presente il gestore code. È necessario quindi copiare il file di connessione client al computer client IBM MQ.

Multi ***Supporto thread multipli - pipelining***

Facoltativamente, è possibile consentire a un MCA (message channel agent) di trasferire i messaggi utilizzando più thread. Questo processo, denominato *pipelining*, consente all'MCA di trasferire i messaggi in modo più efficiente, con un minor numero di stati di attesa, migliorando le prestazioni del canale. Ogni MCA è limitato a un massimo di due thread.

Si controlla la pipeline con il parametro *PipeLineLength* nel file qm.ini . Questo parametro viene aggiunto alla sezione Canali.

Nota: Il pipelining è efficace solo per i canali TCP/IP.

Quando si utilizza la pipeline, i gestori code su entrambe le estremità del canale devono essere configurati in modo che *PipeLineLength* sia maggiore di 1.

Considerazioni sull'uscita del canale

La pipeline può causare il malfunzionamento di alcuni programmi di uscita, perché:

- Le uscite potrebbero non essere richiamate in modo seriale.

- Le uscite possono essere richiamate alternativamente da thread differenti.

Verificare la progettazione dei programmi di uscita prima di utilizzare la pipeline:

- Le uscite devono essere rientranti in tutte le fasi della loro esecuzione.
- Quando si utilizzano chiamate MQI, ricordare che non è possibile utilizzare la stessa gestione MQI quando l'exit viene richiamata da thread differenti.

Considerare un'uscita del messaggio che apre una coda e ne utilizza l'handle per le chiamate MQPUT su tutte le chiamate successive dell'uscita. L'operazione non riesce in modalità pipeline perché l'uscita viene richiamata da thread differenti. Per evitare questo errore, mantenere un handle di coda per ciascun thread e controllare l'ID thread ogni volta che viene richiamata l'uscita.

Comunicazioni

IBM MQ MQI clients utilizzare i canali MQI per comunicare con il server.

Una definizione di canale deve essere creata sia all'estremità IBM MQ MQI client che al server della connessione. Come creare le definizioni di canale è spiegato in Definizione dei canali MQI.

I protocolli di trasmissione possibili sono mostrati nella seguente tabella:

Tabella 1. Protocolli di trasmissione per canali MQI

Piattaforma client	LU 6.2	TCP/IP	NetBIOS	SPX
IBM i IBM i		Sì		
Linux AIX SistemiAIX and Linux	Sì[1]	Sì		
Windows Windows	Sì	Sì	Sì	Sì

Nota:

1. Linux LU6.2 non è supportato sulle piattaforme seguenti:
   - Linux (piattaforma POWER)
   - Linux (piattaformax86-64)
   - Linux (piattaformazSeries s390x)

Protocolli di trasmissione - combinazione di IBM MQ MQI client e piattaforme server mostra le possibili combinazioni di IBM MQ MQI client e piattaforme server, utilizzando questi protocolli di trasmissione.

Un'applicazione IBM MQ su IBM MQ MQI client può utilizzare tutte le chiamate MQI nello stesso modo di quando il gestore code è locale. **MQCONN** o **MQCONNX** associa l'applicazione IBM MQ al gestore code selezionato, creando un *handle di connessione*. Le altre chiamate che utilizzano tale handle di connessione vengono quindi elaborate dal gestore code connesso. La comunicazione IBM MQ MQI client richiede una connessione attiva tra il client e il server, a differenza della comunicazione tra i gestori code, che è indipendente dalla connessione e dal tempo.

Il protocollo di trasmissione viene specificato utilizzando la definizione di canale e non influisce sull'applicazione. Ad esempio, un'applicazione Windows può connettersi a un gestore code su TCP/IP e a un altro gestore code su NetBIOS.

Considerazioni sulle prestazioni

Il protocollo di trasmissione utilizzato potrebbe influire sulle prestazioni del sistema client e server IBM MQ . In alcune situazioni in cui la trasmissione è lenta, è possibile utilizzare la compressione del canale IBM MQ .

Denominazione degli oggetti IBM MQ

La convenzione di denominazione adottata per gli oggetti IBM MQ dipende dall'oggetto. Anche il nome delle macchine e gli ID utente utilizzati con IBM MQ sono soggetti ad alcune limitazioni di denominazione.

Ciascuna istanza di un gestore code è nota con il relativo nome. Questo nome deve essere univoco nella rete di gestori code interconnessi, in modo che un gestore code possa identificare in modo non ambiguo il gestore code di destinazione a cui viene inviato un determinato messaggio.

Per gli altri tipi di oggetti, ogni oggetto ha un nome associato e può essere indicato con tale nome. Questi nomi devono essere univoci all'interno di un gestore code e di un tipo di oggetto. Ad esempio, è possibile avere una coda e un processo con lo stesso nome, ma non è possibile avere due code con lo stesso nome.

In IBM MQ, i nomi possono avere un massimo di 48 caratteri, ad eccezione di *canali* che hanno un massimo di 20 caratteri. Per ulteriori informazioni sulla denominazione degli oggetti IBM MQ , consultare “Regole per la denominazione degli oggetti IBM MQ” a pagina 37.

Anche il nome delle macchine e gli ID utente utilizzati con IBM MQ sono soggetti ad alcune limitazioni di denominazione:

- Assicurarsi che il nome della macchina non contenga spazi. IBM MQ non supporta nomi di macchine che contengono spazi. Se si installa IBM MQ su tale macchina, non è possibile creare alcun gestore code.
- Per le autorizzazioni IBM MQ , i nomi degli ID utente e dei gruppi non devono superare i 20 caratteri (gli spazi non sono consentiti).
- Windows Un server di IBM MQ for Windows non supporta la connessione di un IBM MQ MQI client se il client è in esecuzione con un ID utente che contiene il carattere @, ad esempio, abc@d.

Concetti correlati

“IBM MQNOMI DI FILE” a pagina 40
Ogni oggetto gestore code, coda, definizione del processo, elenco nomi, canale, canale di connessione client, listener, servizio e informazioni di autenticazione di IBM MQ è rappresentato da un file. Poiché i nomi oggetto non sono necessariamente nomi file validi, il gestore code converte il nome oggetto in un nome file valido, se necessario.

Riferimenti correlati

“Regole per la denominazione degli oggetti IBM MQ” a pagina 37
I nomi oggetto IBM MQ hanno lunghezze massime e sono sensibili al maiuscolo / minuscolo. Non tutti i caratteri sono supportati per ogni tipo di oggetto e molti oggetti hanno regole relative all'unicità dei nomi.

Regole per la denominazione degli oggetti IBM MQ

I nomi oggetto IBM MQ hanno lunghezze massime e sono sensibili al maiuscolo / minuscolo. Non tutti i caratteri sono supportati per ogni tipo di oggetto e molti oggetti hanno regole relative all'unicità dei nomi.

Esistono molti tipi diversi di oggetti IBM MQ e gli oggetti di ogni tipo possono avere lo stesso nome perché esistono in spazi dei nomi oggetto separati: ad esempio, una coda locale e un canale mittente possono avere entrambi lo stesso nome. Tuttavia, un oggetto non può avere lo stesso nome di un altro oggetto nello stesso spazio dei nomi. Ad esempio, una coda locale non può avere lo stesso nome di una coda modello, mentre un canale mittente non può avere lo stesso nome di un canale ricevente.

I seguenti oggetti IBM MQ esistono in spazi dei nomi oggetto separati:

- Informazioni di autenticazione
- Canale
- Canale client
- Listener
- Elenco nomi
- Processo
- Coda

- Servizio
- Classe di memoria
- Sottoscrizione
- Argomento

Lunghezza carattere dei nomi oggetto

In generale, i nomi oggetto IBM MQ possono avere una lunghezza massima di 48 caratteri. Questa regola si applica ai seguenti oggetti:

- Informazioni di autenticazione
- Cluster
- Listener
- Elenco nomi
- Definizione di processo
- Coda
- Gestore code
- Servizio
- Sottoscrizione
- Argomento

Esistono delle limitazioni:

1. z/OS Sui sistemi z/OS , i gestori code devono avere una lunghezza massima di 4 caratteri e devono contenere solo caratteri maiuscoli e numerici.
2. La lunghezza massima dei nomi oggetto canale e dei nomi canale di connessione client è 20 caratteri. Consultare Definizione dei canali per ulteriori informazioni sui canali.
3. Le stringhe argomento possono essere un massimo di 10240 byte. Tutti i nomi oggetto IBM MQ sono sensibili al maiuscolo / minuscolo.
4. I nomi delle sottoscrizioni possono avere una lunghezza massima di 10240 byte e possono contenere spazi.
5. La lunghezza massima dei nomi delle classi di memoria è di 8 caratteri.
6. La lunghezza massima dei nomi della struttura CF è 12 caratteri.

Caratteri nei nomi oggetto

I caratteri validi per i nomi oggetto IBM MQ sono:

Caratteri	Limitazioni
Maiuscolo A - Z	• Nessuna

Caratteri	Limitazioni
Minuscolo a - z	• Negli script MQSC, i nomi con caratteri minuscoli devono essere racchiusi tra virgolette singole. Ciò impedisce che i caratteri minuscoli vengano ripiegati in maiuscolo. • I sistemi che utilizzano EBCDIC Katakana non possono utilizzare caratteri a - z minuscoli nei nomi oggetto. • z/OS Potrebbero esserci delle limitazioni quando si utilizzano caratteri minuscoli sui sistemi z/OS , ad esempio, i nomi dei gestori code non possono contenere caratteri minuscoli. • IBM i Su sistemi IBM i quando si utilizzano comandi CL, i nomi con caratteri minuscoli devono essere racchiusi tra virgolette singole. Ciò impedisce che i caratteri minuscoli vengano ripiegati in maiuscolo.
Numeri 0-9	• Nessuna
Punto (.)	• Nessuna
Trattino basso (_)	• Multi Nessuna • z/OS Evitare di utilizzare nomi con caratteri di sottolineatura iniziali o finali perché non possono essere gestiti dalle operazioni e dai pannelli di controllo IBM MQ for z/OS .
Barra (/)	• Windows Sui sistemi Windows , il primo carattere di un nome gestore code non può essere una barra. • IBM i Su sistemi IBM i quando si utilizzano comandi CL, i nomi contenenti una barra devono essere racchiusi tra virgolette singole. • z/OS Nessuna
Segno percentuale (%)	• ALW Nessuna • z/OS Se si utilizza RACF come gestore della sicurezza esterno per IBM MQ for z/OS, non utilizzare% nei nomi oggetto perché i nomi non sono inclusi nelle verifiche di sicurezza quando si utilizzano i profili generici RACF . • IBM i Su sistemi IBM i quando si utilizzano comandi CL, i nomi che contengono un segno di percentuale devono essere racchiusi tra virgolette singole.

Ci sono anche alcune regole generali riguardanti i caratteri sui nomi degli oggetti:

1. Non sono consentiti spazi vuoti iniziali o centrali.
2. I caratteri della lingua nazionale non sono consentiti.
3. Qualsiasi nome inferiore alla lunghezza del campo completo può essere riempito a destra con spazi. Tutti i nomi brevi restituiti dal gestore code vengono sempre riempiti a destra con spazi.

Nomi coda

Il nome di una coda è composto da due parti:

- Il nome di un gestore code
- Il nome locale della coda così come è noto a tale gestore code

Ogni parte del nome della coda è lunga 48 caratteri.

Per fare riferimento a una coda locale, è possibile omettere il nome del gestore code (sostituendolo con caratteri vuoti o utilizzando un carattere null iniziale). Tuttavia, tutti i nomi coda restituiti a un programma da IBM MQ contengono il nome del gestore code.

z/OS Una coda condivisa, accessibile a qualsiasi gestore code nel relativo gruppo di condivisione code, non può avere lo stesso nome di una coda locale non condivisa nello stesso gruppo di condivisione code. Questa limitazione evita la possibilità che un'applicazione apra erroneamente una coda condivisa quando intendeva aprire una coda locale o viceversa. Le code condivise e i gruppi di condivisione code sono disponibili solo su IBM MQ for z/OS.

Per fare riferimento ad una coda remota, un programma deve includere il nome del gestore code nel nome completo della coda oppure deve esistere una definizione locale della coda remota.

Quando un'applicazione utilizza un nome coda, tale nome può essere il nome di una coda locale (o un alias a uno) o il nome di una definizione locale di una coda remota, ma l'applicazione non ha bisogno di sapere quale, a meno che non abbia bisogno di richiamare un messaggio dalla coda (quando la coda deve essere locale). Quando l'applicazione apre l'oggetto coda, la chiamata MQOPEN esegue una funzione di risoluzione dei nomi per stabilire su quale coda eseguire le operazioni successive. Il significato di ciò è che l'applicazione non ha alcuna dipendenza integrata su code particolari definite in posizioni particolari in una rete di gestori code. Pertanto, se un amministratore di sistema riposiziona le code nella rete e ne modifica le definizioni, non è necessario modificare le applicazioni che utilizzano tali code.

Nomi oggetto riservati

I nomi oggetto che iniziano con SYSTEM. sono riservati agli oggetti definiti dal gestore code. È possibile utilizzare i comandi **Alter**, **Define**e **Replace** per modificare queste definizioni di oggetti in modo da adattarle alla propria installazione. I nomi definiti per IBM MQ sono elencati in modo completo in Nomi coda.

z/OS Su IBM MQ for z/OS, il nome della struttura dell'applicazione Coupling Facility CSQSYSAPPL è riservato.

Concetti correlati

Nome installazione su AIX, Linux, and Windows

IBM MQNOMI DI FILE

Ogni oggetto gestore code, coda, definizione del processo, elenco nomi, canale, canale di connessione client, listener, servizio e informazioni di autenticazione di IBM MQ è rappresentato da un file. Poiché i nomi oggetto non sono necessariamente nomi file validi, il gestore code converte il nome oggetto in un nome file valido, se necessario.

Il percorso predefinito per una directory del gestore code è il seguente:

- Un prefisso, definito nelle informazioni di configurazione IBM MQ :

– Linux AIX Su AIX and Linux, il prefisso predefinito è `/var/mqm`. Questo è configurato nella stanza `DefaultPrefix` del file di configurazione mqs.ini .

– Windows Su sistemi Windows a 32 bit, il prefisso predefinito è `C:\Program Files (x86)\IBM\WebSphere MQ`. Su sistemi Windows a 64 bit, il prefisso predefinito è `C:\Program Files\IBM\MQ`. Per le installazioni a 32 bit e a 64 bit, le directory di dati sono installate in `C:\ProgramData\IBM\MQ`. Questo è configurato nella stanza `DefaultPrefix` del file di configurazione mqs.ini .

Laddove disponibile, il prefisso può essere modificato utilizzando la pagina delle proprietà IBM MQ in Esplora risorse di IBM MQ , altrimenti modificare manualmente il file di configurazione `mqs.ini` .

- Il nome del gestore code viene trasformato in un nome directory valido. Ad esempio, il gestore code:

```
queue.manager
```

sarà rappresentato come:

```
queue!manager
```

Questo processo viene definito *trasformazione del nome*.

In IBM MQ, è possibile assegnare a un gestore code un nome contenente fino a 48 caratteri.

Ad esempio, è possibile denominare un gestore code:

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

Tuttavia, ogni gestore code è rappresentato da un file e vi sono limitazioni sulla lunghezza massima di un nome file e sui caratteri che possono essere utilizzati nel nome. Di conseguenza, i nomi dei file che rappresentano gli oggetti vengono trasformati automaticamente per soddisfare i requisiti del file system.

Le regole che gestiscono la trasformazione di un nome gestore code sono le seguenti:

1. Trasforma singoli caratteri:
   - Da. a!
   - Da / a &
2. Se il nome non è ancora valido:
   a. Tronca a otto caratteri
   b. Accoda un suffisso numerico a tre caratteri

Ad esempio, supponendo il prefisso predefinito e un gestore code con il nome `queue.manager`:

- Windows Su Windows con NTFS o FAT32, il nome gestore code diventa:

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

- Windows Su Windows con FAT, il nome del gestore code diventa:

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

- Linux AIX Su AIX and Linux, il nome gestore code diventa:

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

L'algoritmo di trasformazione distingue anche tra nomi che differiscono solo in maiuscolo / minuscolo su file system che non sono sensibili al maiuscolo / minuscolo.

Trasformazione nome oggetto

I nomi oggetto non sono necessariamente nomi file system validi. Potrebbe essere necessario trasformare i nomi oggetto. Il metodo utilizzato è diverso da quello utilizzato per i nomi dei gestori code poiché, sebbene vi siano solo pochi nomi di gestori code su ciascuna macchina, è possibile che vi sia un numero elevato di altri oggetti per ciascun gestore code. Code, definizioni di processi, elenchi nomi, canali, canali di connessione client, listener, servizi e oggetti delle informazioni di autenticazione sono rappresentati nel file system.

Quando il processo di trasformazione genera un nuovo nome, non esiste una semplice relazione con il nome oggetto originale. È possibile utilizzare il comando **dspmqfls** per eseguire la conversione tra nomi di oggetti reali e trasformati.

Riferimenti correlati

dspmqfls (nomi file di visualizzazione)

Informazioni correlate

Stanza AllQueueManagers del file mqs.ini

IBM i Nomi oggetto su IBM i

Un gestore code ha una libreria del gestore code associata che ha un nome univoco. Potrebbe essere necessario trasformare i nomi dei gestori code e dei nomi oggetto per soddisfare i requisiti dell'IFS (Integrated File System) IBM i Integrated File System .

Quando viene creato un gestore code, IBM MQ associa una libreria del gestore code. A questa libreria del gestore code viene assegnato un nome univoco, non più lungo di 10 caratteri, basato in gran parte sul nome del gestore code definito dall'utente. Sia il gestore code che la libreria del gestore code vengono collocati in una directory che si basa anche sul nome del gestore code con il prefisso `/QIBM/UserData/mqm`. Di seguito è riportato un esempio di gestore code, libreria del gestore code e directory:

Nome del gestore code	ARANCIO
Nome libreria gestore code	QMORANGE
Directory	`/QIBM/UserData/mqm/ORANGE`

Tutti i nomi dei gestori code e i nomi delle librerie dei gestori code vengono scritti nelle stanze nel file `/QIBM/UserData/mqm/mqs.ini`.

IBM MQ File e directory IFS

L'IFS (Integrated File System) IBM i Integrated File System viene utilizzato ampiamente da IBM MQ per memorizzare i dati. Per ulteriori informazioni su IFS, consultare *Integrated File System Introduzione*.

Ogni oggetto IBM MQ , ad esempio un canale o un gestore code, è rappresentato da un file. Poiché i nomi oggetto non sono necessariamente nomi file validi, il gestore code converte il nome oggetto in un nome file valido, se necessario.

Il percorso di una directory del gestore code è formato da:

- Un prefisso, definito nel file di configurazione del gestore code, `qm.ini`. Il prefisso predefinito è `/QIBM/UserData/mqm`.
- Una costante letterale, `qmgrs`.
- Un nome gestore code codificato, che è il nome del gestore code trasformato in un nome directory valido. Ad esempio, il gestore code `queue/manager` è rappresentato da `queue&manager`.

Questo processo viene definito trasformazione del nome.

Trasformazione nome gestore code IFS

In IBM MQ, è possibile assegnare a un gestore code un nome contenente fino a 48 caratteri.

Ad esempio, è possibile denominare un gestore code `QUEUE/MANAGER/ACCOUNTING/SERVICES`. Nello stesso modo in cui viene creata una libreria per ciascun gestore code, ogni gestore code è rappresentato anche da un file. A causa dei punti di codice delle varianti in EBCDIC, esistono delle limitazioni ai caratteri che possono essere utilizzati nel nome. Di conseguenza, i nomi dei file IFS che rappresentano gli oggetti vengono trasformati automaticamente per soddisfare i requisiti del file system.

Utilizzando l'esempio di un gestore code con il nome `queue/manager`, trasformando il carattere `/` in `&e` assumendo il prefisso predefinito, il nome del gestore code in IBM MQ for IBM i diventa `/QIBM/UserData/mqm/qmgrs/queue&manager`.

Trasformazione nome oggetto

I nomi oggetto non sono necessariamente nomi file system validi, quindi i nomi oggetto potrebbero dover essere trasformati. Il metodo utilizzato è diverso da quello per i nomi dei gestori code perché, sebbene vi siano solo pochi nomi di gestori code per ciascuna macchina, è possibile che vi sia un numero elevato di altri oggetti per ciascun gestore code. Solo le definizioni dei processi, le code e gli elenchi nomi sono rappresentati nel file system; i canali non sono influenzati da queste considerazioni.

Quando il processo di trasformazione genera un nuovo nome, non esiste una semplice relazione con il nome oggetto originale. È possibile utilizzare il comando DSPMQMOBJN per visualizzare i nomi trasformati per gli oggetti IBM MQ .

Accodamento distribuito e cluster

L'accodamento distribuito significa inviare messaggi da un gestore code ad un altro. Il gestore code di ricezione può trovarsi sulla stessa macchina o su un'altra; nelle vicinanze o dall'altra parte del mondo. Può essere in esecuzione sulla stessa piattaforma del gestore code locale o su una qualsiasi delle piattaforme supportate da IBM MQ. È possibile definire manualmente tutte le connessioni in un ambiente di accodamento distribuito oppure è possibile creare un cluster e consentire a IBM MQ di definire gran parte dei dettagli di connessione.

accodamento distribuito

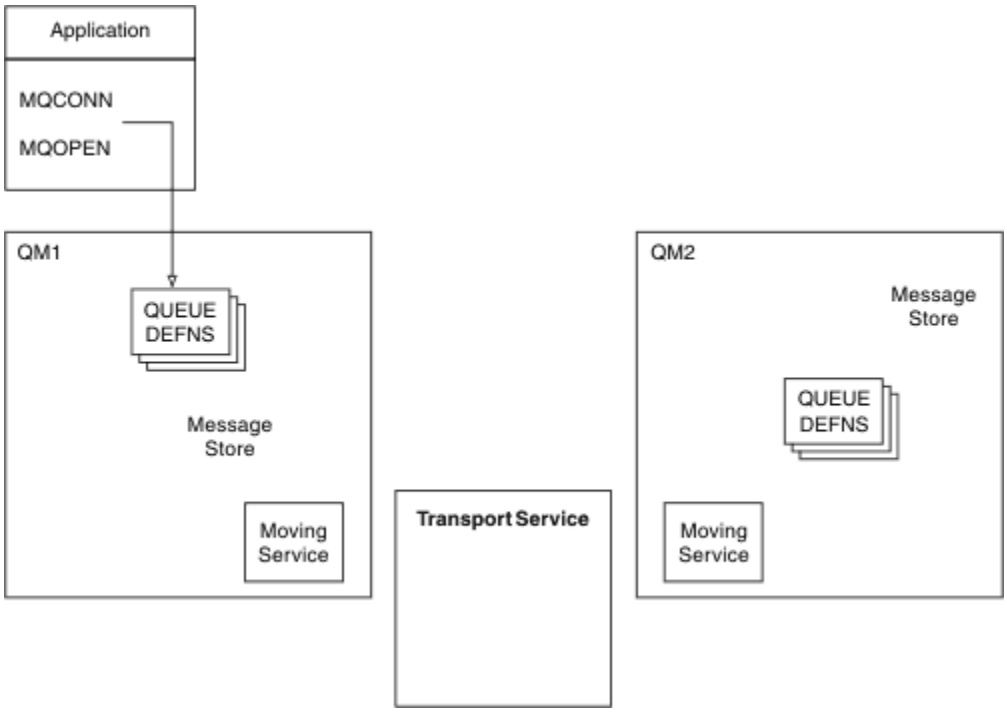


Figura 4. Panoramica dei componenti dell'accodamento distribuito

Nella figura precedente:

- Un'applicazione utilizza la chiamata MQCONN per connettersi ad un gestore code. L'applicazione utilizza quindi la chiamata MQOPEN per aprire una coda in modo che possa inserire i messaggi nella coda.
- Ogni gestore code ha una definizione per ciascuna delle sue code. Può avere definizioni di *code locali* (ossia, ospitate da questo gestore code) e definizioni di *code remote* (ossia, ospitate da altri gestori code).
- Se i messaggi sono destinati a una coda remota, il gestore code locale li conserva su una coda di trasmissione , che li conserva in un archivio messaggi, fino a quando non possono essere inoltrati al gestore code remoto.
- Ogni gestore code contiene software di comunicazioni, noto come *servizio di spostamento*, che il gestore code utilizza per comunicare con altri gestori code.
- Il *servizio di trasporto* è indipendente dal gestore code e può essere uno dei seguenti (a seconda della piattaforma):
  - SNA APPC (Systems Network Architecture Advanced Program - to Program Communication)
  - Transmission Control Protocol/Internet Protocol (TCP/IP)
  - Sistema di input / output di base della rete (NetBIOS)
  - SPX (Sequenced Packet Exchange)

Componenti necessari per inviare un messaggio

Se un messaggio deve essere inviato a un gestore code remoto, il gestore code locale necessita di definizioni per una *coda di trasmissione* e un *canale*. Un canale è un collegamento di comunicazione unidirezionale tra due gestori code. Può trasportare messaggi destinati a qualsiasi numero di code sul gestore code remoto.

Ogni estremità di un canale ha una definizione separata, definendolo, ad esempio, come l'estremità di invio o l'estremità di ricezione. Un canale semplice è costituito da una definizione di canale *mittente* nel gestore code locale e una definizione di canale *destinatario* nel gestore code remoto. Queste due definizioni devono avere lo stesso nome e insieme costituiscono un canale.

Il software che gestisce l'invio e la ricezione di messaggi è denominato *MCA (Message Channel Agent)* . Esiste un MCA (*message channel agent*) ad ogni estremità di un canale.

Ogni gestore code deve avere una *coda di messaggi non recapitabili* (nota anche come *coda di messaggi non recapitati*). I messaggi vengono inseriti in questa coda se non possono essere consegnati alla loro destinazione.

La seguente figura mostra la relazione tra gestori code, code di trasmissione, canali e MCA:

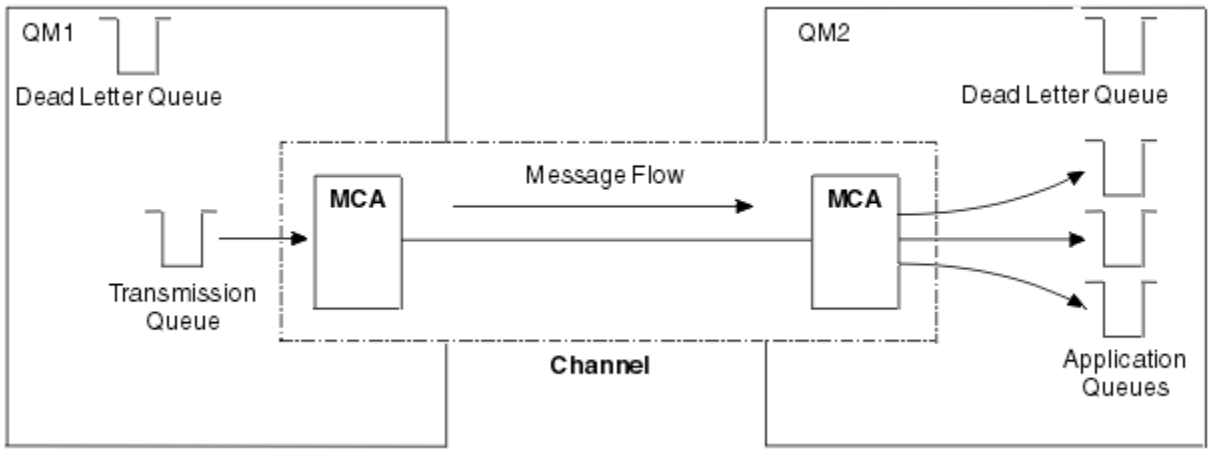


Figura 5. invio di messaggi

Componenti necessari per restituire un messaggio

Se l'applicazione richiede che i messaggi vengano restituiti dal gestore code remoto, è necessario definire un altro canale, da eseguire nella direzione opposta tra i gestori code, come mostrato nella seguente figura:

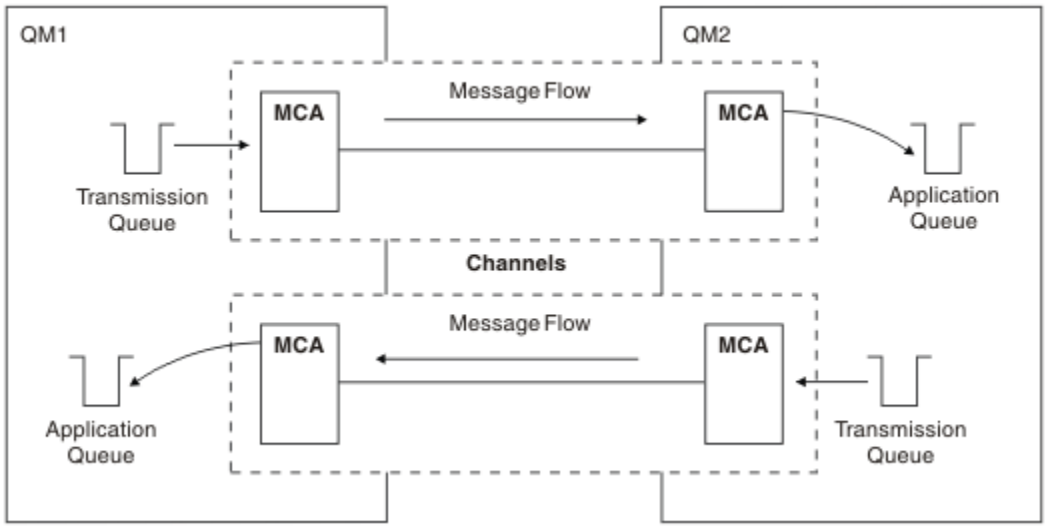


Figura 6. Invio di messaggi in entrambe le direzioni

Cluster

Anziché definire manualmente tutte le connessioni in un ambiente di accodamento distribuito, è possibile raggruppare una serie di gestori code in un cluster. Quando si esegue questa operazione, i gestori code possono rendere le code che ospitano disponibili ad altri gestori code nel cluster senza la necessità di definizioni di canale esplicite, definizioni di coda remota o code di trasmissione per ciascuna destinazione. Ogni gestore code in un cluster ha una singola coda di trasmissione che trasmette i messaggi a qualsiasi altro gestore code nel cluster. Per ogni gestore code, è necessario solo definire un canale ricevente del

cluster e un canale mittente del cluster; eventuali canali aggiuntivi vengono gestiti automaticamente dal cluster.

Un IBM MQ client può connettersi a un gestore code che fa parte di un cluster, proprio come può connettersi a qualsiasi altro gestore code. Come con l'accodamento distribuito configurato manualmente, utilizzare la chiamata MQPUT per inserire un messaggio in una coda su qualsiasi gestore code. Utilizzare la chiamata MQGET per richiamare i messaggi da una coda locale.

I gestori code su piattaforme che supportano cluster non devono necessariamente far parte di un cluster. È possibile continuare a configurare manualmente l'accodamento distribuito e / o utilizzare i cluster.

Vantaggi dell'utilizzo dei cluster

Il clustering fornisce due vantaggi principali:

- I cluster semplificano la gestione delle reti IBM MQ , che di solito richiedono la configurazione di molte definizioni di oggetti per canali, code di trasmissione e code remote. Questa situazione è particolarmente vera nelle reti di grandi dimensioni, potenzialmente in fase di modifica, in cui molti gestori code devono essere interconnessi. Questa architettura è particolarmente difficile da configurare e mantenere attivamente.
- I cluster possono essere utilizzati per distribuire il carico di lavoro del traffico di messaggi tra code e gestori code nel cluster. Tale distribuzione consente la distribuzione del carico di lavoro dei messaggi di una singola coda tra istanze equivalenti di quella coda ubicata su più gestori code. Questa distribuzione del carico di lavoro può essere utilizzata per ottenere una maggiore resilienza agli errori di sistema e per migliorare le prestazioni di scalabilità dei flussi di messaggi particolarmente attivi in un sistema. In un ambiente di questo tipo, ciascuna delle istanze delle code distribuite utilizza le applicazioni che elaborano i messaggi. Per ulteriori informazioni, consultare Utilizzo dei cluster per la gestione del carico di lavoro.

Modalità di instradamento dei messaggi in un cluster

È possibile considerare un cluster come una rete di gestori code gestiti da un amministratore di sistema coscienzioso. Ogni volta che si definisce una coda cluster, l'amministratore di sistema crea automaticamente le definizioni di coda remota corrispondenti, in base alle esigenze, sugli altri gestori code.

Non è necessario creare definizioni di code di trasmissione perché IBM MQ fornisce una coda di trasmissione su ciascun gestore code del cluster. Questa singola coda di trasmissione può essere utilizzata per trasportare i messaggi a qualsiasi altro gestore code nel cluster. Non si è limitati all'utilizzo di una singola coda di trasmissione. Un gestore code può utilizzare più code di trasmissione per separare i messaggi diretti a ciascun gestore code in un cluster. Generalmente, un gestore code utilizza una singola coda di trasmissione cluster. È possibile modificare l'attributo del gestore code DEFCLXQ, in modo che un gestore code utilizzi una diversa coda di trasmissione cluster per ciascun gestore code in un cluster. È anche possibile definire manualmente le code di trasmissione del cluster.

Tutti i gestori code che si uniscono a un cluster accettano di lavorare in questo modo. Inviano informazioni su se stessi e sulle code che ospitano e ricevono informazioni sugli altri membri del cluster.

Per garantire che non vengano perse informazioni quando un gestore code diventa non disponibile, specificare due gestori code nel cluster che fungano da *repository completi*. Questi gestori code memorizzano una serie completa di informazioni su tutti i gestori code e le code nel cluster. Tutti gli altri gestori code nel cluster memorizzano solo le informazioni su tali gestori code e code con cui scambiano messaggi. Questi gestori code sono noti come *repository parziali*. Per ulteriori informazioni, consultare “Repository cluster” a pagina 57.

Per far parte di un cluster, un gestore code deve avere due canali: un canale mittente del cluster e un canale ricevente del cluster:

- Un canale mittente cluster è un canale di comunicazione come un canale mittente. È necessario creare manualmente un canale mittente del cluster su un gestore code per connetterlo a un repository completo già membro del cluster.

- Un canale ricevente cluster è un canale di comunicazione come un canale ricevente. È necessario creare manualmente un canale ricevente del cluster. Il canale funge da meccanismo per il gestore code per ricevere le comunicazioni cluster.

Tutti gli altri canali necessari per la comunicazione tra questo gestore code e gli altri membri del cluster vengono quindi creati automaticamente.

La seguente figura mostra i componenti di un cluster denominato CLUSTER:

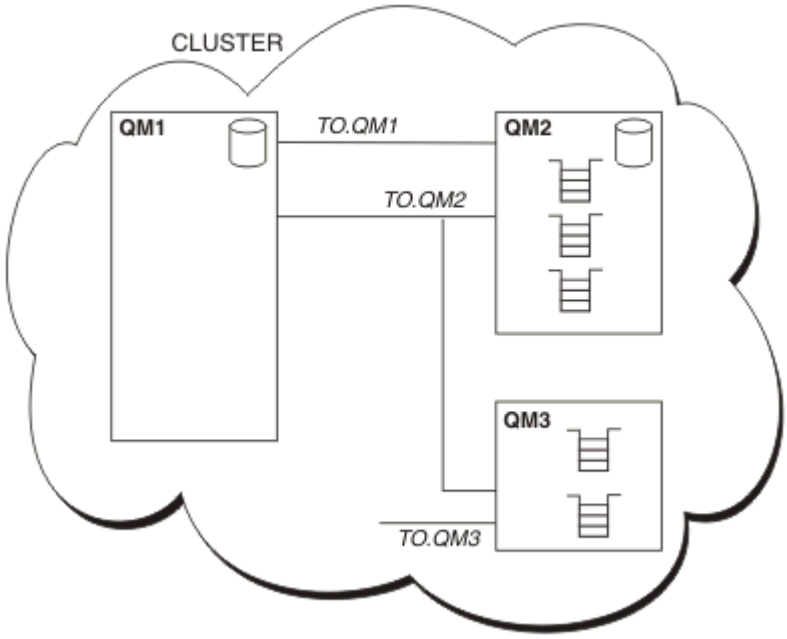


Figura 7. Un cluster di gestori code

- CLUSTER contiene tre gestori code, QM1, QM2e QM3.
- QM1 e QM2 ospitano repository completi di informazioni sui gestori code e sulle code nel cluster.
- QM2 e QM3 ospitano alcune code cluster, ossia code accessibili a qualsiasi altro gestore code nel cluster.
- Ogni gestore code dispone di un canale ricevente del cluster denominato TO.qmgr su cui è possibile ricevere messaggi.
- Ogni gestore code ha anche un canale mittente del cluster su cui può inviare informazioni a uno dei gestori code del repository.
- QM1 e QM3 inviano al repository in QM2 e QM2 invia al repository in QM1.

Componenti di accodamento distribuiti

I componenti dell'accodamento distribuito sono i canali di messaggi, gli agenti dei canali di messaggi, le code di trasmissione, i listener e gli iniziatori dei canali e i programmi di uscita dei canali. La definizione di ciascuna estremità di un canale di messaggi può essere di diversi tipi.

I canali dei messaggi sono i canali che trasportano i messaggi da un gestore code all'altro. Non confondere i canali dei messaggi con quelli MQI. Esistono due tipi di canale MQI, SVRCONN (server - connection) e CLNTCONN (client - connection). Per ulteriori informazioni, vedi Canali.

La definizione di ciascuna estremità di un canale di messaggi può essere uno dei seguenti tipi:

- Mittente (SDR)
- Ricevitore (RCVR)
- Server (SVR)

- Richiedente (RQSTR)
- Mittente cluster (CLUSSDR)
- Ricevitore cluster (CLUSRCVR)

Un canale di messaggi viene definito utilizzando uno di questi tipi definiti ad un'estremità e un tipo compatibile all'altra estremità. Le combinazioni possibili sono:

- Mittente-Ricevente
- Richiedente-Server
- Richiedente-Mittente (callback)
- Server-Ricevente
- Mittente cluster - ricevente cluster

Istruzioni dettagliate per la creazione di un canale mittente - destinatario sono incluse in Definizione dei canali. Per esempi dei parametri necessari per impostare i canali mittente - destinatario, consultare Informazioni di configurazione di esempio applicabili alla propria piattaforma. Per i parametri necessari per definire un canale di qualsiasi tipo, consultare DEFINE CHANNEL.

Canali mittente-ricevente

Un mittente in un sistema avvia il canale in modo che possa inviare messaggi all'altro sistema. Il mittente richiede l'avvio del destinatario all'altra estremità del canale. Il mittente invia i messaggi dalla coda di trasmissione al ricevitore. Il destinatario inserisce i messaggi nella coda di destinazione. Figura 8 a pagina 48 illustra questo.

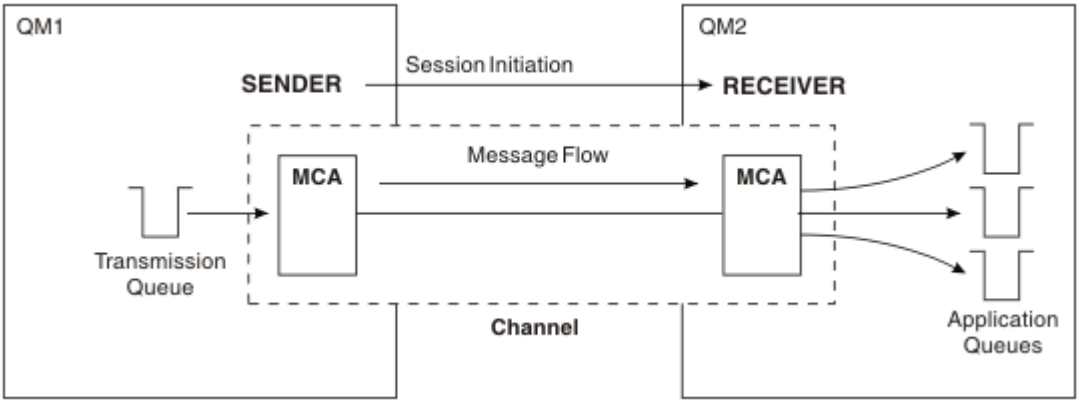


Figura 8. Un canale mittente - destinatario

Canali richiedente-server

Un richiedente in un sistema avvia il canale in modo che possa ricevere messaggi dall'altro sistema. Il richiedente richiede l'avvio del server all'altra estremità del canale. Il server invia i messaggi al richiedente dalla coda di trasmissione definita nella definizione del canale.

Un canale server può anche avviare la comunicazione e inviare messaggi a un richiedente. Ciò si applica solo ai server *completi* , ossia i canali server che hanno il nome connessione del partner specificato nella definizione del canale. Un server completo può essere avviato da un richiedente o può avviare una comunicazione con un richiedente.

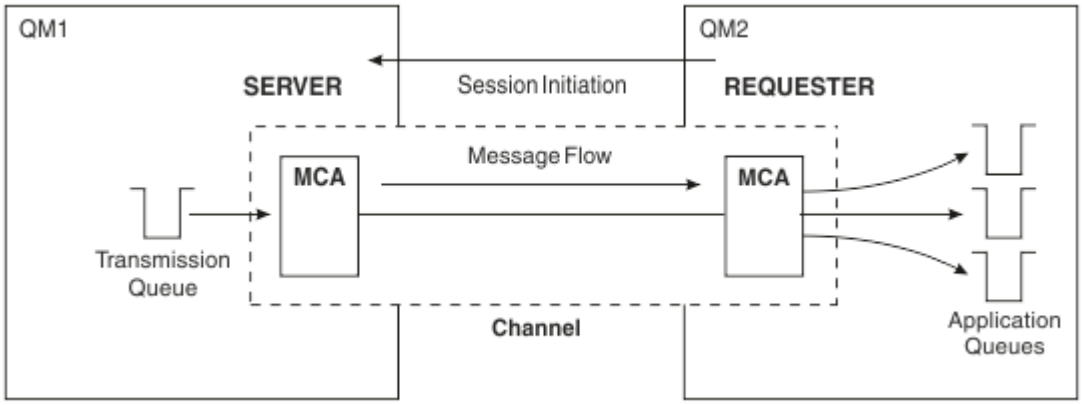


Figura 9. Un canale richiedente - server

Canali richiedente-mittente

Il richiedente avvia il canale e il mittente termina la chiamata. Il mittente riavvia quindi la comunicazione in base alle informazioni nella relativa definizione di canale (nota come *callback*). Invia i messaggi dalla coda di trasmissione al richiedente.

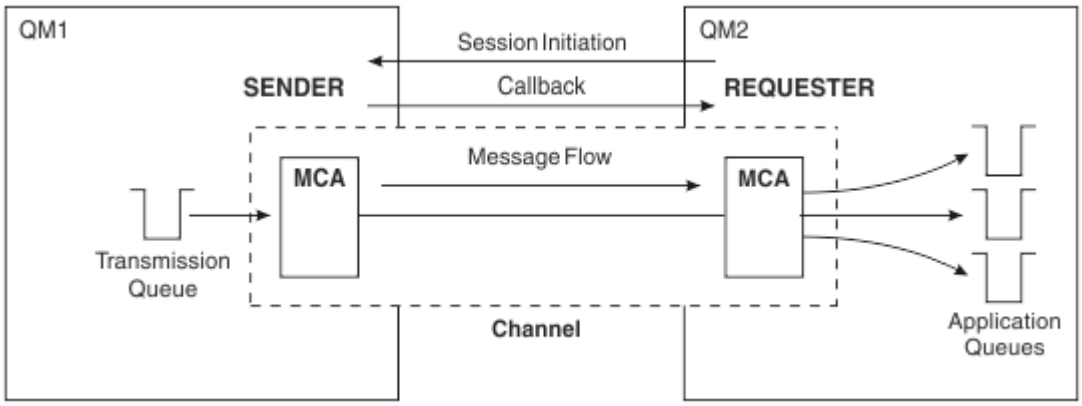


Figura 10. Un canale richiedente - mittente

Canali riceventi del server

È come il mittente - destinatario, ma si applica solo ai server *completi* , ossia i canali server che hanno il nome connessione del partner specificato nella definizione del canale. L'avvio del canale deve essere avviato all'estremità del server del collegamento. L'illustrazione è come l'illustrazione in Figura 8 a pagina 48.

Canali mittenti del cluster

In un cluster, ogni gestore code dispone di un canale mittente del cluster su cui è possibile inviare le informazioni del cluster a uno dei gestori code del repository completo. I gestori code possono anche inviare messaggi ad altri gestori code sui canali mittenti del cluster.

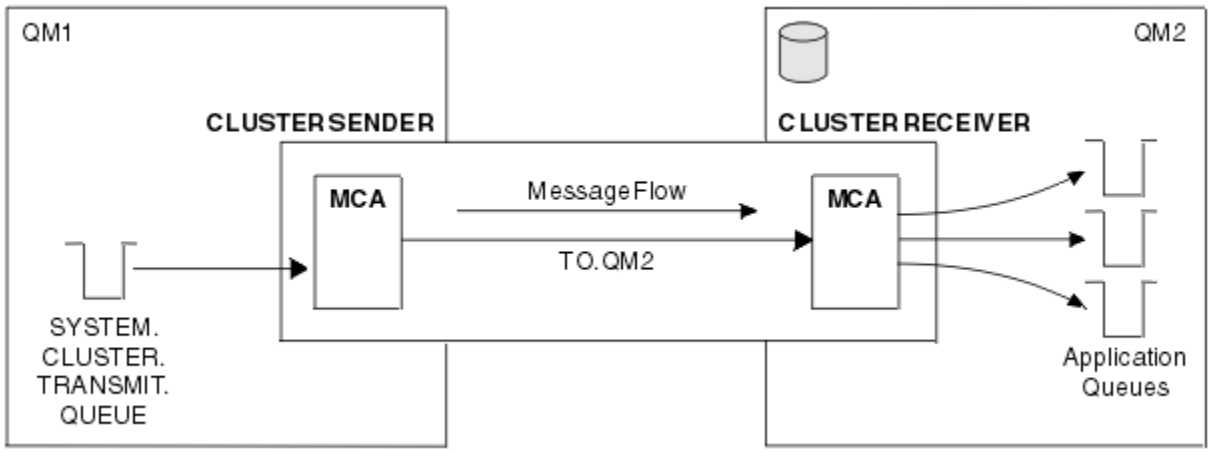


Figura 11. Un canale mittente del cluster

Canali riceventi del cluster

In un cluster, ogni gestore code ha un canale ricevente del cluster su cui può ricevere messaggi e informazioni sul cluster. L'illustrazione è come l'illustrazione in Figura 11 a pagina 50.

Code di messaggi non recapitabili

La coda di messaggi non recapitabili (o coda di messaggi non recapitati) è la coda a cui vengono inviati i messaggi se non possono essere instradati alla destinazione corretta. Ogni gestore code generalmente ha una coda di messaggi non recapitabili.

Una *coda di messaggi non recapitabili* (DLQ), a volte indicata come *coda di messaggi non recapitati*, è una coda di attesa per i messaggi che non possono essere consegnati alle relative code di destinazione, ad esempio perché la coda non esiste o perché è piena. Le code di messaggi non recapitabili vengono utilizzate anche all'estremità di invio di un canale, per errori di conversione dati. Ogni gestore code in una rete generalmente ha una coda locale da utilizzare come coda di messaggi non recapitabili in modo che i messaggi che non possono essere consegnati alla destinazione corretta possano essere memorizzati per un successivo richiamo.

I messaggi possono essere inseriti nella DLQ da gestori code, MCA (message channel agent) e applicazioni. Tutti i messaggi sulla DLQ devono avere come prefisso una struttura *dead-letter header* , MQDLH. Il campo *Motivo* della struttura MQDLH contiene un codice motivo che identifica il motivo per cui il messaggio si trova sulla DLQ.

Generalmente, è necessario definire una coda di messaggi non instradabili per ogni gestore code. In caso contrario, e l'MCA non è in grado di inserire un messaggio, questo viene lasciato sulla coda di trasmissione e il canale viene arrestato. Inoltre, se i messaggi sono rapidi, non persistenti (consultare Messaggi rapidi, non persistenti) non può essere consegnato e non esiste alcuna coda di messaggi non instradabili sul sistema di destinazione, questi messaggi vengono eliminati.

Tuttavia, l'utilizzo di code di messaggi non recapitabili può influenzare la sequenza in cui i messaggi vengono consegnati, pertanto è possibile scegliere di non utilizzarle.

Attività correlate

Gestione delle code di messaggi non recapitabili

Risoluzione dei problemi dei messaggi non recapitati

Riferimenti correlati

runmqdlq (esecuzione gestore code di messaggi non instradabili)

Definizioni di coda remota

Le definizioni di coda remota sono definizioni per le code di proprietà di un altro gestore code.

Mentre le applicazioni possono richiamare i messaggi solo dalle code locali, possono inserire i messaggi sulle code locali o remote. Pertanto, oltre a una definizione per ognuna delle proprie code locali, un gestore code può avere *definizioni di code remote*. Il vantaggio delle definizioni della coda remota è che consentono a un'applicazione di inserire un messaggio in una coda remota senza dover specificare il nome della coda remota o del gestore code remoto o il nome della coda di trasmissione. Le definizioni della coda remota forniscono l'indipendenza dell'ubicazione.

Esistono altri usi per le definizioni di coda remota, descritti in seguito.

Come accedere al gestore code remoto

È possibile che non si disponga sempre di un canale tra ciascun gestore code di origine e di destinazione. Ci sono una serie di altri modi di collegamento tra i due, tra cui multi - hopping, canali di condivisione, utilizzando canali diversi e il clustering.

Multi - hopping

Se non vi è alcun collegamento di comunicazione diretto tra il gestore code di origine e il gestore code di destinazione, è possibile passare attraverso uno o più *gestori code intermedi* nel percorso verso il gestore code di destinazione. Questo è noto come *multi - hop*.

È necessario definire i canali tra tutti i gestori code e le code di trasmissione sui gestori code intermedi. Viene mostrato in Figura 12 a pagina 51.

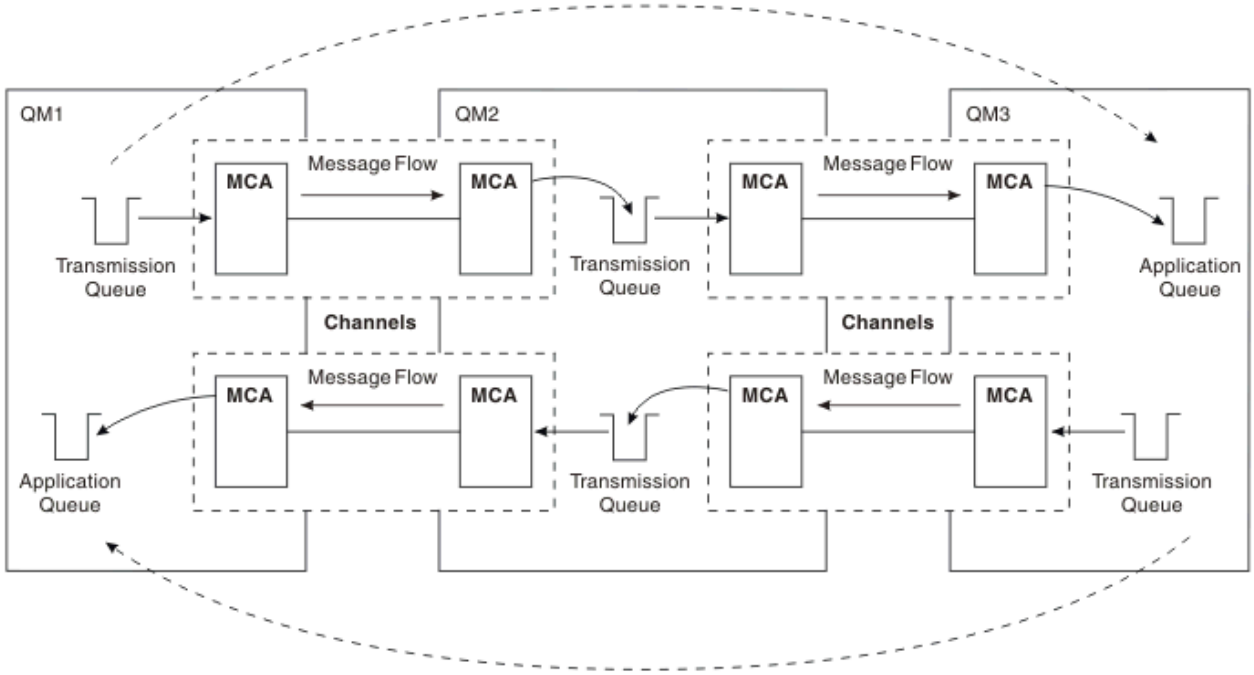


Figura 12. Passaggio attraverso gestori code intermedi

Condivisione dei canali

In qualità di progettista dell'applicazione, è possibile forzare le applicazioni a specificare il nome del gestore code remoto insieme al nome della coda oppure creare una *definizione della coda remota* per ogni coda remota. Questa definizione contiene il nome del gestore code remoto, il nome della coda e il nome della coda di trasmissione. In entrambi i casi, tutti i messaggi di tutte le applicazioni che indirizzano le code nella stessa ubicazione remota vengono inviati attraverso la stessa coda di trasmissione. Viene mostrato in Figura 13 a pagina 52.

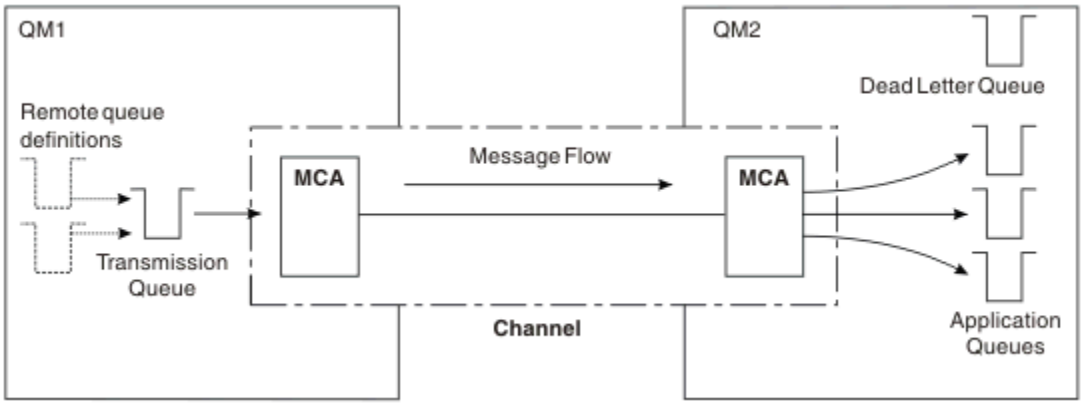


Figura 13. Condivisione di una coda di trasmissione

Figura 13 a pagina 52 illustra che i messaggi da più applicazioni a più code remote possono utilizzare lo stesso canale.

Utilizzo di canali diversi

Se si dispone di messaggi di tipo diverso da inviare tra due gestori code, è possibile definire più di un canale tra i due. Ci sono momenti in cui hai bisogno di canali alternativi, magari per motivi di sicurezza, o per scambiare la velocità di consegna contro la maggior parte del traffico di messaggi.

Per impostare un secondo canale, è necessario definire un altro canale e un'altra coda di trasmissione e creare una definizione della coda remota specificando l'ubicazione e il nome della coda di trasmissione. Le applicazioni possono quindi utilizzare entrambi i canali, ma i messaggi vengono ancora consegnati alle stesse code di destinazione. Viene mostrato in Figura 14 a pagina 52.

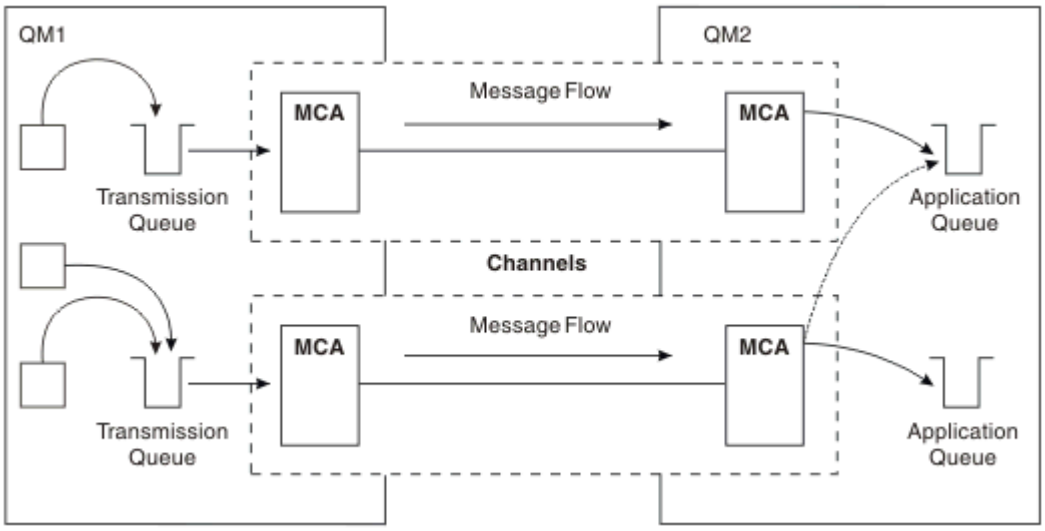


Figura 14. Utilizzo di più canali

Quando si utilizzano le definizioni della coda remota per specificare una coda di trasmissione, le applicazioni **non** devono specificare l'ubicazione (ossia, il gestore code di destinazione). In tal caso, il gestore code non utilizza le definizioni della coda remota. Le definizioni della coda remota forniscono l'indipendenza dell'ubicazione. Le applicazioni possono inserire messaggi in una coda *logica* senza sapere dove si trova la coda ed è possibile modificare la coda *fisica* senza dover modificare le proprie applicazioni.

Utilizzo del clustering

Ogni gestore code all'interno di un cluster definisce un canale ricevente del cluster. Quando un altro gestore code desidera inviare un messaggio a tale gestore code, definisce automaticamente il canale mittente del cluster corrispondente. Ad esempio, se c'è più di un'istanza di una coda in un cluster, il canale mittente del cluster potrebbe essere definito per qualsiasi gestore code che ospita la coda. IBM MQ utilizza un algoritmo di gestione del carico di lavoro che utilizza una routine round robin per selezionare un gestore code disponibile a cui instradare un messaggio. Per ulteriori informazioni, consultare Cluster.

Informazioni sull'indirizzamento

Quando un'applicazione inserisce messaggi destinati a un gestore code remoto, il gestore code locale aggiunge loro un'intestazione di trasmissione prima di inserirli nella coda di trasmissione. Questa intestazione contiene il nome della coda di destinazione e del gestore code, ovvero le *informazioni di indirizzamento*.

In un singolo ambiente del gestore code, l'indirizzo di una coda di destinazione viene stabilito quando un'applicazione apre una coda per l'inserimento dei messaggi. Poiché la coda di destinazione si trova sullo stesso gestore code, non è necessaria alcuna informazione di indirizzamento.

In un ambiente di accodamento distribuito, il gestore code deve conoscere non solo il nome della coda di destinazione, ma anche l'ubicazione di tale coda (ossia, il nome del gestore code) e l'instradamento a tale ubicazione remota (ossia, la coda di trasmissione). Queste informazioni di indirizzamento sono contenute nell'intestazione di trasmissione. Il canale di ricezione rimuove l'intestazione di trasmissione e utilizza le informazioni in essa contenute per individuare la coda di destinazione.

È possibile evitare che le applicazioni debbano specificare il nome del gestore code di destinazione se si utilizza una definizione di coda remota. Questa definizione specifica il nome della coda remota, il nome del gestore code remoto a cui sono destinati i messaggi e il nome della coda di trasmissione utilizzata per trasportare i messaggi.

Cosa sono gli alias?

Gli alias vengono utilizzati per fornire una qualità del servizio per i messaggi. L'alias del gestore code consente all'amministratore di sistema di modificare il nome di un gestore code di destinazione senza dover modificare le applicazioni. Inoltre, consente all'amministratore di sistema di modificare l'instradamento a un gestore code di destinazione o di impostare un instradamento che implica il passaggio attraverso un numero di altri gestori code (multi - hopping). L'alias della coda di risposta fornisce una qualità del servizio per le risposte.

Gli alias del gestore code e gli alias della coda di risposta vengono creati utilizzando una definizione di coda remota con un RNAME vuoto. Queste definizioni non definiscono code reali; vengono utilizzate dal gestore code per risolvere i nomi delle code fisiche, i nomi dei gestori code e le code di trasmissione.

Le definizioni alias sono caratterizzate da un RNAME vuoto.

Risoluzione nome coda

La risoluzione del nome della coda si verifica in ogni gestore code ogni volta che una coda viene aperta. Il suo scopo è identificare la coda di destinazione, il gestore code di destinazione (che potrebbe essere locale) e l'instradamento a tale gestore code (che potrebbe essere null). Il nome risolto è composto da tre parti: il nome del gestore code, il nome della coda e, se il gestore code è remoto, la coda di trasmissione.

Quando esiste una definizione di coda remota, non si fa riferimento a nessuna definizione di alias. Il nome coda fornito dall'applicazione viene risolto nel nome della coda di destinazione, del gestore code remoto e della coda di trasmissione specificati nella definizione della coda remota. Per informazioni più dettagliate sulla risoluzione dei nomi delle code, consultare Risoluzione dei nomi delle code.

Se non è presente alcuna definizione di coda remota e viene specificato un nome gestore code o viene risolto dal servizio dei nomi, il gestore code controlla se esiste una definizione alias del gestore code che corrisponde al nome del gestore code fornito. In questo caso, le informazioni vengono utilizzate per

risolvere il nome del gestore code nel nome del gestore code di destinazione. La definizione alias del gestore code può essere utilizzata anche per determinare la coda di trasmissione al gestore code di destinazione.

Se il nome della coda risolta non è una coda locale, sia il nome del gestore code che il nome della coda sono inclusi nell'intestazione di trasmissione di ogni messaggio inserito dall'applicazione alla coda di trasmissione.

La coda di trasmissione utilizzata in genere ha lo stesso nome del gestore code risolto, a meno che non venga modificata da una definizione di coda remota o da una definizione di alias del gestore code. Se non è stata definita una coda di trasmissione di questo tipo ma è stata definita una coda di trasmissione predefinita, viene utilizzata.

z/OS I nomi dei gestori code in esecuzione su z/OS sono limitati a quattro caratteri.

Definizioni alias gestore code

Le definizioni degli alias del gestore code si applicano quando un'applicazione che apre una coda per inserire un messaggio, specifica il nome della coda **e** il nome del gestore code.

Le definizioni alias del gestore code hanno tre utilizzi:

- Quando si inviano messaggi, riassociare il nome del gestore code
- Durante l'invio di messaggi, la modifica o la specifica della coda di trasmissione
- Quando si ricevono i messaggi, determinando se il gestore code locale è la destinazione prevista per tali messaggi

Messaggi in uscita - rimappatura del nome gestore code

Le definizioni di alias del gestore code possono essere utilizzate per riassociare il nome gestore code specificato in una chiamata MQOPEN. Ad esempio, una chiamata MQOPEN specifica un nome coda THISQ e un nome gestore code YOURQM. Nel gestore code locale, è presente una definizione alias del gestore code simile al seguente esempio:

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

Mostra che il gestore code reale da utilizzare, quando un'applicazione inserisce i messaggi nel gestore code YOURQM, è REALQM. Se il gestore code locale è REALQM, inserisce i messaggi nella coda THISQ, che è una coda locale. Se il gestore code locale non è denominato REALQM, instrada il messaggio ad una coda di trasmissione denominata REALQM. Il gestore code modifica l'intestazione di trasmissione in REALQM invece di YOURQM.

Messaggi in uscita - modifica o specifica della coda di trasmissione

La Figura 15 a pagina 55 mostra uno scenario in cui i messaggi arrivano al gestore code QM1 con intestazioni di trasmissione che mostrano i nomi delle code sul gestore code QM3. In questo scenario, QM3 è raggiungibile passando attraverso QM2.

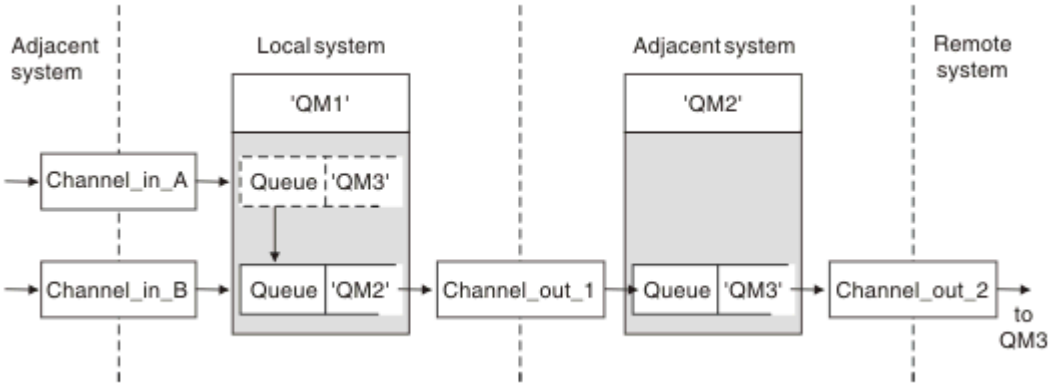


Figura 15. Alias gestore code

Tutti i messaggi per QM3 vengono acquisiti in QM1 con un alias del gestore code. L'alias del gestore code è denominato QM3 e contiene la definizione QM3 tramite la coda di trasmissione QM2. La definizione è simile al seguente esempio:

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

Il gestore code inserisce i messaggi nella coda di trasmissione QM2 ma non modifica l'intestazione della coda di trasmissione perché il nome del gestore code di destinazione, QM3, non cambia.

Anche tutti i messaggi che arrivano a QM1 e che mostrano un'intestazione di trasmissione contenente un nome coda in QM2 vengono inseriti sulla coda trasmissione QM2 . In questo modo, i messaggi con destinazioni differenti vengono raccolti su una coda di trasmissione comune a un sistema adiacente appropriato, per la trasmissione in avanti alle relative destinazioni.

Messaggi in ingresso - determinazione della destinazione

Un MCA di ricezione apre la coda a cui si fa riferimento nell'intestazione di trasmissione. Se esiste una definizione di alias del gestore code con lo stesso nome del gestore code a cui si fa riferimento, il nome del gestore code ricevuto nell'intestazione di trasmissione viene sostituito con RQMNAME da tale definizione.

Questo processo ha due usi:

- Indirizzamento di messaggi a un altro gestore code
- Modifica del nome del gestore code in modo che sia uguale al gestore code locale

Definizioni alias coda di risposta

Una definizione dell'alias della coda di risposta specifica nomi alternativi per le informazioni di risposta nel descrizione del messaggio. Il vantaggio è che è possibile modificare il nome di una coda o di un gestore code senza dover modificare le applicazioni.

Risoluzione nome coda

Quando un'applicazione risponde a un messaggio, utilizza i dati nel *descrittore del messaggio* del messaggio ricevuto per individuare il nome della coda a cui rispondere. L'applicazione di invio indica dove vengono inviate le risposte e allega queste informazioni ai relativi messaggi. Questo concetto deve essere coordinato come parte della progettazione dell'applicazione.

La risoluzione del nome della coda avviene all'estremità di invio dell'applicazione, prima che il messaggio venga inserito in una coda. La risoluzione del nome della coda si verifica quindi prima dell'interazione con l'applicazione remota a cui viene inviato il messaggio. Questa è l'unica situazione in cui la risoluzione dei nomi si verifica in un momento in cui una coda non viene aperta.

Risoluzione del nome coda utilizzando un alias del gestore code

Normalmente un'applicazione specifica una coda di risposta e lascia vuoto il nome del gestore code di risposta. Il gestore code completa il proprio nome al momento dell'inserimento. Questo metodo funziona bene tranne quando si desidera utilizzare un canale alternativo per le repliche, ad esempio, un canale che utilizzi la coda di trasmissione `QM1_relief` invece del canale di ritorno predefinito che utilizza la coda di trasmissione `QM1`. In questa situazione, i nomi dei gestore code specificati nelle intestazioni delle code di trasmissione non corrispondono ai nomi dei gestori code "reali", ma vengono rispecificati utilizzando le definizioni di alias dei gestori code. Per restituire le risposte lungo instradamenti alternativi, è necessario associare anche i dati della coda reply - to, utilizzando le definizioni degli alias della coda reply - to.

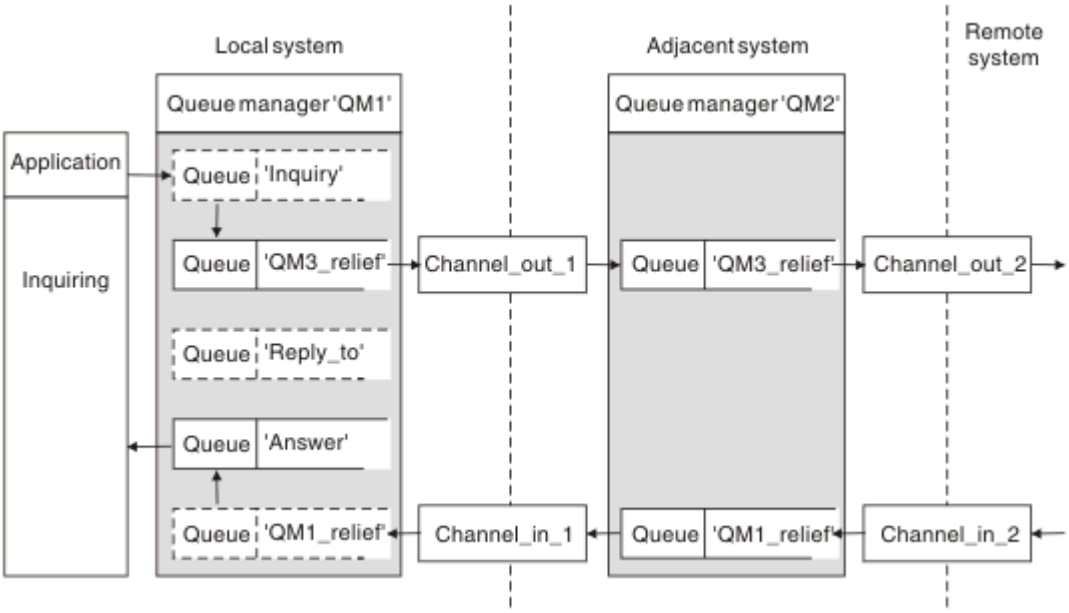


Figura 16. Alias della coda di risposta utilizzato per modificare l'ubicazione della risposta

Nell'esempio in Figura 16 a pagina 56:

1. L'applicazione inserisce un messaggio utilizzando la chiamata MQPUT e specificando le seguenti informazioni nel descrittore del messaggio:

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

   ReplyToIl gestore code deve essere vuoto per poter utilizzare l'alias della coda di risposta.
2. Si crea una definizione alias della coda di risposta denominata `Reply_to`, che contiene il nome `Answere` il nome gestore code `QM1_relief`.

   ```
   DEFINE QREMOTE ('Reply_to') RNAME ('Answer')
   RQMNAME ('QM1_relief')
   ```
3. I messaggi vengono inviati con un descrittore di messaggi che mostra `ReplyToQ='Answer'` e `ReplyToQMgr='QM1_relief'`.
4. La specificazione dell'applicazione deve includere le informazioni che le risposte devono trovare nella coda `Answer` piuttosto che `Reply_to`.

Per prepararsi per le risposte è necessario creare il canale di ritorno parallelo, definendo:

- Su QM2, la coda di trasmissione denominata QM1_relief

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

- In QM1, l'alias del gestore code QM1_relief

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

  Questo alias del gestore code termina la catena di canali di ritorno paralleli e cattura i messaggi per QM1.

Se si pensa di voler eseguire questa operazione in futuro, assicurarsi che le applicazioni utilizzino il nome alias dall'inizio. Per ora si tratta di un alias di coda normale per la coda di risposta, ma successivamente può essere modificato in un alias del gestore code.

Nome delle repliche alla coda

È necessario prestare attenzione alla denominazione delle code di risposta. Il motivo per cui un'applicazione inserisce un nome coda di risposta nel messaggio è che può specificare la coda a cui vengono inviate le proprie risposte. Quando si crea una definizione dell'alias della coda di risposta con questo nome, non è possibile avere la coda di risposta effettiva (ovvero, una definizione della coda locale) con lo stesso nome. Pertanto, la definizione dell'alias della coda reply - to deve contenere un nuovo nome coda e il nome del gestore code e la specifica dell'applicazione deve includere le informazioni che le sue risposte vengono trovate in questa altra coda.

Le applicazioni ora devono richiamare i messaggi da una coda diversa da quella indicata come coda di risposta quando inseriscono il messaggio originale.

Componenti cluster

I cluster sono composti da gestore code, repository di cluster, canali cluster e code cluster.

Consultare i seguenti argomenti secondari per informazioni su ciascuno dei componenti cluster:

Concetti correlati

Confronto tra cluster e accodamento distribuito

Attività correlate

Configurazione di un cluster di gestore code

Configurazione di un nuovo cluster

Repository cluster

Un repository è una raccolta di informazioni sui gestori code che sono membri di un cluster.

Le informazioni sul repository includono i nomi dei gestori code, le loro ubicazioni, i loro canali, quali code ospitano e altre informazioni. Le informazioni vengono memorizzate sotto forma di messaggi su una coda denominata SYSTEM.CLUSTER.REPOSITORY.QUEUE. Questa coda è uno degli oggetti predefiniti. Multi Su Multipiattaforme, viene definito quando viene creato un gestore code IBM MQ . z/OS Su IBM MQ for z/OS, viene definito come parte della personalizzazione del gestore code.

Repository completo e repository parziale

In genere, due gestori code in un cluster contengono un repository completo. Tutti i rimanenti gestori code contengono un repository parziale.

Un gestore code che ospita una serie completa di informazioni su ogni gestore code nel cluster ha un repository completo. Altri gestori code nel cluster dispongono di repository parziali contenenti un sottoinsieme di informazioni nei repository completi.

Un repository parziale contiene informazioni solo sui gestori code con cui il gestore code deve scambiare messaggi. I gestori code richiedono aggiornamenti alle informazioni di cui hanno bisogno, in modo che, in caso di modifiche, il gestore code del repository completo invii loro le nuove informazioni. Per gran parte del tempo, un repository parziale contiene tutte le informazioni che un gestore code deve eseguire all'interno del cluster. Quando un gestore code richiede ulteriori informazioni, interroga il repository completo e aggiorna quindi il repository parziale. I gestori code utilizzano la coda SYSTEM.CLUSTER.COMMAND.QUEUE per richiedere e ricevere aggiornamenti ai repository.

Quando si migra i gestori code che sono membri di un cluster, migrare i repository completi prima dei repository parziali. Ciò è dovuto al fatto che un repository meno recente non può memorizzare attributi più recenti introdotti in una versione più recente. Li tollera, ma non li conserva.

Gestore code del cluster

Un gestore code cluster è un gestore code membro di un cluster.

Un gestore code può essere un membro di più di un cluster. Ogni gestore code cluster deve avere un nome univoco in tutti i cluster di cui è membro.

Un gestore code del cluster può ospitare code, che vengono pubblicizzate agli altri gestori code del cluster. Tuttavia, non è necessario che lo faccia. Può invece alimentare i messaggi nelle code ospitate altrove nel cluster e ricevere solo le risposte che sono indirizzate esplicitamente ad essa.

z/OS In IBM MQ for z/OS, un gestore code del cluster può essere membro di un gruppo di condivisione code. In questo caso, condivide le definizioni di coda con altri gestori code nello stesso gruppo di condivisione code.

I gestori code del cluster sono autonomi. Hanno il controllo completo delle code e dei canali che definiscono. Le loro definizioni non possono essere modificate da altri gestori code (diversi dai gestori code nello stesso gruppo di condivisione code). I gestori code del repository non controllano le definizioni in altri gestori code nel cluster. Essi contengono una serie completa di tutte le definizioni, da utilizzare quando richiesto. Un cluster è una federazione di gestori code.

Dopo aver creato o modificato una definizione su un gestore code del cluster, le informazioni vengono inviate al gestore code del repository completo. Gli altri repository nel cluster vengono aggiornati successivamente.

Gestore code con repository completo

Un gestore code del repository completo è un gestore code del cluster che contiene una rappresentazione completa delle risorse del cluster. Per garantire la disponibilità, impostare due o più gestori code del repository completo in ciascun cluster. I gestori code del repository completo ricevono le informazioni inviate dagli altri gestori code nel cluster e aggiornano i relativi repository. Inviano messaggi l'uno all'altro per essere certi di essere entrambi aggiornati con nuove informazioni sul cluster.

Gestori code e repository

Ogni cluster ha almeno un gestore code (preferibilmente due) che contiene repository completi di informazioni sui gestori code, le code e i canali in un cluster. Questi repository contengono anche richieste dagli altri gestori code nel cluster per aggiornamenti alle informazioni.

Gli altri gestori code contengono un repository parziale, contenente informazioni sul sottoinsieme di code e gestori code con cui devono comunicare. I gestori code creano i propri repository parziali effettuando richieste quando devono accedere per la prima volta a un'altra coda o a un altro gestore code. Richiedono la notifica di eventuali nuove informazioni relative a tale coda o gestore code.

Ciascun gestore code memorizza le proprie informazioni sul repository in messaggi su una coda denominata SYSTEM.CLUSTER.REPOSITORY.QUEUE. I gestori code scambiano le informazioni del repository nei messaggi su una coda denominata SYSTEM.CLUSTER.COMMAND.QUEUE.

Ogni gestore code che unisce un cluster definisce un canale mittente del cluster, CLUSSDR, in uno dei repository. Apprende immediatamente quali altri gestori code nel cluster contengono repository completi.

Da quel momento in poi, il gestore code può richiedere informazioni da uno qualsiasi dei repository. Quando il gestore code invia le informazioni al repository scelto, invia anche le informazioni a un altro repository (se presente).

Un repository completo viene aggiornato quando il gestore code che lo ospita riceve nuove informazioni da uno dei gestori code ad esso collegati. Le nuove informazioni vengono inviate anche a un altro repository, per ridurre il rischio che vengano ritardate se un gestore code del repository è fuori servizio. Poiché tutte le informazioni vengono inviate due volte, i repository devono eliminare i duplicati. Ogni elemento di informazioni contiene un numero di sequenza, che i repository utilizzano per identificare i duplicati. Tutti i repository vengono mantenuti in linea tra loro scambiando messaggi.

Code cluster

Una coda cluster è una coda ospitata da un gestore code cluster e resa disponibile ad altri gestori code del cluster.

Una definizione di coda cluster viene pubblicizzata in altri gestori code nel cluster. Gli altri gestori code nel cluster possono inserire i messaggi in una coda cluster senza la necessità di una definizione di coda remota corrispondente. Una coda cluster può essere pubblicizzata in più di un cluster utilizzando un elenco dei nomi di cluster.

Quando una coda viene pubblicizzata, qualsiasi gestore code del cluster può inserire dei messaggi al suo interno. Per inserire un messaggio, il gestore code deve scoprire, dai repository completi, la posizione in cui è ospitata la coda. Aggiunge quindi alcune informazioni di instradamento al messaggio e inserisce tale messaggio su una coda di trasmissione del cluster.

Una coda del cluster può essere una coda condivisa dai membri di un gruppo di condivisione di code in IBM MQ for z/OS.

Attività correlate

Definizione di code cluster

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.

Canali cluster

Su ogni repository completo, si definisce manualmente un canale ricevente del cluster e una serie di canali mittenti del cluster per connettersi a ogni altro repository completo nel cluster. Quando si aggiunge un repository parziale, definire manualmente un canale ricevente del cluster e un singolo canale mittente del cluster che si connette a uno dei repository completi. Ulteriori canali mittente del cluster vengono definiti automaticamente dal cluster quando necessario. I canali mittenti del cluster definiti automaticamente prendono i loro attributi dalla corrispondente definizione di canale ricevente del cluster sul gestore code di ricezione.

Canale ricevente del cluster: CLUSRCVR

Una definizione di canale CLUSRCVR definisce la fine di un canale su cui un gestore code cluster può ricevere messaggi da altri gestori code nel cluster.

È necessario definire almeno un canale CLUSRCVR per ogni gestore code cluster. Definendo il canale CLUSRCVR , il gestore code mostra agli altri gestori code del cluster che è disponibile per ricevere messaggi.

Una definizione di canale CLUSRCVR consente inoltre ad altri gestori code di definire automaticamente le corrispondenti definizioni di canale mittente del cluster. Consultare la sezione “Canali mittenti del cluster definiti automaticamente” a pagina 61 di questo articolo.

Canale mittente del cluster: CLUSSDR

Definire manualmente il canale CLUSSDR da ogni gestore code del repository completo a ogni altro gestore code del repository completo nel cluster. Tutti gli aggiornamenti scambiati dai repository completi fluiscono esclusivamente su questi canali. Definendo manualmente questi canali, si controlla esplicitamente la rete di repository completi.

Quando si aggiunge un gestore code del repository parziale a un cluster, si definisce manualmente un singolo canale CLUSSDR per connettersi ad uno dei repository completi. Fa poca differenza quale repository completo si sceglie, perché una volta stabilito il contatto iniziale, ulteriori oggetti del gestore code del cluster per il gestore code, inclusi i canali CLUSSDR , vengono definiti automaticamente come necessario. Ciò consente al gestore code di inviare informazioni sul cluster a qualsiasi repository completo e di inviare messaggi a qualsiasi gestore code nel cluster.

Come spiegato nella sezione di questo articolo, i canali mittenti definiti automaticamente si basano sulla configurazione del canale ricevente del cluster. Pertanto, tutte le proprietà del canale impostate sui canali cluster devono essere impostate in modo identico per i canali CLUSSDR e i canali riceventi del cluster oppure solo per i canali riceventi del cluster.

È necessario definire manualmente solo i canali CLUSSDR per i motivi precedentemente descritti. Vale a dire, per collegare inizialmente un repository parziale a un repository completo o per collegare due repository completi insieme. La configurazione manuale di un canale CLUSSDR che si connette a un repository parziale o a un gestore code non incluso nel cluster, causa l'emissione di messaggi di errore come AMQ9427 e AMQ9428 . Anche se a volte ciò potrebbe essere inevitabile come situazione temporanea, ad esempio quando si modifica l'ubicazione di un repository completo, la definizione manuale deve essere eliminata il più presto possibile.

Canali mittenti del cluster definiti automaticamente

Di solito, quando si aggiunge un gestore code del repository parziale a un cluster, si definiscono manualmente solo due canali cluster sul gestore code:

- Un mittente del cluster (CLUSSDR) da un canale ad un gestore code del repository completo per il cluster.
- Un ricevitore cluster (CLUSRCVR) canale.

Il canale CLUSSDR definito consente al gestore code di stabilire un contatto iniziale con il cluster. Una volta stabilito il contatto iniziale, ulteriori canali CLUSSDR vengono definiti automaticamente dal cluster quando necessario.

Un canale CLUSSDR definito automaticamente prende i propri attributi dalla corrispondente definizione di canale CLUSRCVR sul gestore code di ricezione. Anche se è presente un canale CLUSSDR definito manualmente, vengono utilizzati gli attributi del canale CLUSSDR definito automaticamente. Si supponga, ad esempio, di definire un canale CLUSRCVR senza specificare un numero di porta nel parametro **CONNAME** e definire manualmente un canale CLUSSDR che specifichi un numero di porta. Quando il canale CLUSSDR definito automaticamente sostituisce quello definito manualmente, il numero di porta (preso dal canale CLUSRCVR) diventa vuoto. Viene utilizzato il numero di porta predefinito e il canale ha esito negativo.

Laddove vi sono differenze di configurazione tra un canale CLUSSDR definito manualmente e la corrispondente definizione di canale CLUSRCVR , alcune differenze diventano effettive immediatamente (ad esempio, i parametri di bilanciamento del carico di lavoro) e alcune diventano effettive solo al riavvio del canale (ad esempio, la configurazione TLS).

Per evitare confusione, osservare per quanto possibile le seguenti linee guida:

- Definire solo manualmente i canali CLUSSDR per puntare a repository completi.
- Se si dispone di canali CLUSSDR definiti manualmente, configurarli in modo che corrispondano in modo identico alla definizione di canale CLUSRCVR corrispondente sul gestore code di ricezione.

Consultare anche Gestione dei canali definiti automaticamente.

Concetti correlati

Utilizzo dei canali definiti automaticamente

Utilizzo delle code di trasmissione del cluster e dei canali mittente del cluster

Attività correlate

Configurazione di un nuovo cluster

Aggiunta di un gestore code a un cluster

Argomenti cluster

sono argomenti di amministrazione con l'attributo **cluster** definito. Le informazioni sugli argomenti cluster vengono inoltrate a tutti i membri di un cluster e combinate con argomenti locali per creare parti di uno spazio argomento che si estende su più gestori code. Ciò fa sì che i messaggi pubblicati su un argomento in un gestore code vengano consegnati alle sottoscrizioni di altri gestori code nel cluster.

Quando si definisce un argomento cluster su un gestore code, la definizione dell'argomento cluster viene inviata ai gestori code del repository completo. I repository completi propagano quindi la definizione dell'argomento cluster a tutti i gestori code all'interno del cluster, rendendo disponibile lo stesso argomento cluster ai publisher e sottoscrittori di qualsiasi gestore code del cluster. Il gestore code su cui si crea un argomento cluster è noto come host dell'argomento cluster. L'argomento cluster può essere utilizzato da qualsiasi gestore code nel cluster, ma le modifiche a un cluster devono essere effettuate sul gestore code dove è definito tale argomento (l'host) nel punto in cui la modifica viene propagata a tutti i membri del cluster attraverso i repository completi.

Per informazioni sulla configurazione degli argomenti del cluster per l'utilizzo dell' *instradamento diretto* o dell' *instradamento host dell'argomento*e sull'eredità dell'argomento in cluster e le sottoscrizioni con caratteri jolly, consultare Definizione degli argomenti del cluster.

Per informazioni sui comandi da utilizzare per visualizzare gli argomenti cluster, consultare le informazioni correlate.

Concetti correlati

Utilizzo degli argomenti di gestione

Utilizzo delle sottoscrizioni

Riferimenti correlati

VISUALIZZAZIONEARGOMENTO

STATOVISUALIZZAZIONEP

VISUALIZZAZIONESUB

Oggetti cluster predefiniti

Multi Su Multiplatforms, gli oggetti cluster predefiniti sono inclusi nella serie di oggetti predefiniti creati automaticamente quando si definisce un gestore code. z/OS Su z/OS, le definizioni di oggetti cluster predefinite sono disponibili negli esempi di personalizzazione.

Nota: È possibile modificare le definizioni di canale predefinite come qualsiasi altra definizione di canale, eseguendo i comandi MQSC o PCF. Non modificare le definizioni di coda predefinite, tranne per SYSTEM.CLUSTER.HISTORY.QUEUE.

SYSTEM.CLUSTER.COMMAND.QUEUE
Ogni gestore code in un cluster ha una coda locale denominata SYSTEM.CLUSTER.COMMAND.QUEUE che viene utilizzata per trasferire i messaggi al repository completo. Il messaggio contiene qualsiasi informazione nuova o modificata sul gestore code o qualsiasi richiesta di informazioni su altri gestori code. SYSTEM.CLUSTER.COMMAND.QUEUE è normalmente vuoto.

SYSTEM.CLUSTER.HISTORY.QUEUE
Ogni gestore code in un cluster ha una coda locale denominata SYSTEM.CLUSTER.HISTORY.QUEUE. SYSTEM.CLUSTER.HISTORY.QUEUE viene utilizzato per memorizzare la cronologia delle informazioni sullo stato del cluster per scopi di servizio.

Nelle impostazioni dell'oggetto predefinito, SYSTEM.CLUSTER.HISTORY.QUEUE è impostata su PUT (ENABLED). Per eliminare la raccolta cronologica, modificare l'impostazione in PUT (DISABLED).

SYSTEM.CLUSTER.REPOSITORY.QUEUE
Ogni gestore code in un cluster ha una coda locale denominata SYSTEM.CLUSTER.REPOSITORY.QUEUE. Questa coda viene utilizzata per memorizzare tutte le informazioni complete del repository. Questa coda non è normalmente vuota.

SYSTEM.CLUSTER.TRANSMIT.QUEUE
Ogni Gestore code ha una definizione per una coda locale denominata SYSTEM.CLUSTER.TRANSMIT.QUEUE. SYSTEM.CLUSTER.TRANSMIT.QUEUE è la coda di trasmissione predefinita per tutti i messaggi a tutte le code e i gestori code all'interno dei cluster. È possibile modificare la coda di trasmissione predefinita per ogni canale mittente del cluster in SYSTEM.CLUSTER.TRANSMIT. *ChannelName*, modificando l'attributo gestore code DEFCLXQ . Non è possibile eliminare SYSTEM.CLUSTER.TRANSMIT.QUEUE. Viene utilizzato anche

per definire i controlli di autorizzazione se la coda di trasmissione predefinita utilizzata è SYSTEM.CLUSTER.TRANSMIT.QUEUE o SYSTEM.CLUSTER.TRANSMIT. *ChannelName*.

SYSTEM.DEF.CLUSRCVR
Ogni cluster ha una definizione di canale CLUSRCVR predefinita denominata SYSTEM.DEF.CLUSRCVR. SYSTEM.DEF.CLUSRCVR viene utilizzato per fornire i valori predefiniti per gli attributi che non vengono specificati quando si crea un canale ricevente del cluster su un gestore code nel cluster.

SYSTEM.DEF.CLUSSDR
Ogni cluster ha una definizione di canale CLUSSDR predefinita denominata SYSTEM.DEF.CLUSSDR. SYSTEM.DEF.CLUSSDR viene utilizzato per fornire valori predefiniti per gli attributi che non vengono specificati quando si crea un canale mittente del cluster su un gestore code nel cluster.

Concetti correlati
Utilizzo degli oggetti cluster predefiniti

Pubblicazione/sottoscrizione della messaggistica

La messaggistica di pubblicazione / sottoscrizione consente di disaccoppiare il fornitore di informazioni dai consumatori di tali informazioni. L'applicazione mittente e l'applicazione ricevente non hanno bisogno di sapere nulla l'uno sull'altro per le informazioni da inviare e ricevere.

Prima che un'applicazione IBM MQ point-to-point possa inviare un messaggio ad un'altra applicazione, è necessario che conosca qualcosa su tale applicazione. Ad esempio, deve conoscere il nome della coda a cui inviare le informazioni e potrebbe anche specificare un nome gestore code.

La pubblicazione / sottoscrizione IBM MQ elimina la necessità che la tua applicazione conosca qualcosa sull'applicazione di destinazione. Tutto ciò che l'applicazione di invio deve fare è questo:

- *Inserire* un messaggio IBM MQ che contiene le informazioni richieste dall'applicazione.
- Assegnare il messaggio a un argomento che denota l'oggetto delle informazioni.
- Lasciare che IBM MQ gestisca la distribuzione di tali informazioni.

Allo stesso modo, l'applicazione di destinazione non deve conoscere l'origine delle informazioni che riceve.

La seguente figura mostra il più semplice sistema di pubblicazione / sottoscrizione. Sono presenti un publisher, un gestore code e un sottoscrittore. Una sottoscrizione viene effettuata dal sottoscrittore su un gestore code, una pubblicazione viene inviata dal publisher al gestore code e la pubblicazione viene quindi inoltrata dal gestore code al sottoscrittore.

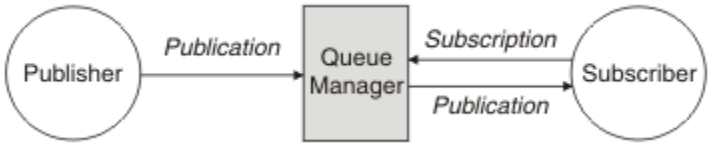


Figura 17. Configurazione di pubblicazione / sottoscrizione semplice

Un tipico sistema di pubblicazione / sottoscrizione ha più di un publisher e più di un sottoscrittore su diversi argomenti e spesso ha più di un gestore code. Un'applicazione può essere sia un publisher che un sottoscrittore.

Un'altra differenza significativa tra la messaggistica di pubblicazione / sottoscrizione e quella point - to - point è che un messaggio inviato a una coda point - to - point viene elaborato solo da una singola applicazione. Un messaggio pubblicato in un argomento di pubblicazione / sottoscrizione, in cui più di un sottoscrittore ha registrato un interesse, viene elaborato da ogni sottoscrittore interessato.

Componenti di pubblicazione / sottoscrizione

La pubblicazione / sottoscrizione è il meccanismo mediante il quale i sottoscrittori possono ricevere informazioni, sotto forma di messaggi, dai publisher. Le interazioni tra i publisher e i sottoscrittori sono controllate dai gestori code, utilizzando le funzioni IBM MQ standard.

Un tipico sistema di pubblicazione / sottoscrizione ha più di un publisher e più di un sottoscrittore su diversi argomenti e spesso ha più di un gestore code. Un'applicazione può essere sia un publisher che un sottoscrittore.

Il fornitore di informazioni è denominato *publisher*. Gli editori forniscono informazioni su un argomento, senza bisogno di sapere nulla sulle applicazioni che sono interessate a tali informazioni. I publisher generano queste informazioni sotto forma di messaggi, denominati *pubblicazioni* che desiderano pubblicare e definire l'argomento di questi messaggi.

Il destinatario di un'informazione è chiamato *sottoscrittore*. I sottoscrittori creano *sottoscrizioni* che descrivono l'argomento a cui il sottoscrittore è interessato. Pertanto, la sottoscrizione determina quali pubblicazioni vengono inoltrate al sottoscrittore. I sottoscrittori possono effettuare più sottoscrizioni e possono ricevere informazioni da diversi publisher.

Le informazioni pubblicate vengono inviate in un messaggio IBM MQ e l'oggetto delle informazioni viene identificato dal relativo argomento . Il publisher specifica l'argomento quando pubblica le informazioni e il sottoscrittore specifica gli argomenti su cui desidera ricevere le pubblicazioni. Al sottoscrittore vengono inviate informazioni solo sugli argomenti sottoscritti.

È l'esistenza di argomenti che consentono ai fornitori e ai consumatori di informazioni di essere disaccoppiati nella messaggistica di pubblicazione / sottoscrizione, eliminando la necessità di includere una specifica destinazione in ogni messaggio come richiesto nella messaggistica point - to - point.

Le interazioni tra i publisher e i sottoscrittori sono tutte controllate da un gestore code. Il gestore code riceve i messaggi dai publisher e le sottoscrizioni dai sottoscrittori (a una gamma di argomenti). Il lavoro del gestore code consiste nell'instradare i messaggi pubblicati ai sottoscrittori che hanno registrato un interesse nell'argomento dei messaggi.

Le funzioni IBM MQ standard vengono utilizzate per distribuire i messaggi, in modo che le proprie applicazioni possano utilizzare tutte le funzioni disponibili per le applicazioni IBM MQ esistenti. Ciò significa che è possibile utilizzare i messaggi persistenti per ottenere una consegna garantita una sola volta e che i messaggi possono far parte di un'unità di lavoro transazionale per garantire che i messaggi vengano consegnati al sottoscrittore solo se ne è stato eseguito il commit da parte del publisher.

Editori e pubblicazioni

In IBM MQ la pubblicazione / sottoscrizione di un publisher è un'applicazione che rende disponibili le informazioni su un argomento specificato per un gestore code sotto forma di un messaggio IBM MQ standard denominato pubblicazione. Un publisher può pubblicare informazioni su più di un argomento.

I publisher utilizzano il verbo MQPUT per inserire un messaggio in un argomento precedentemente aperto, questo messaggio è una pubblicazione. Il gestore code locale instrada quindi la pubblicazione a tutti i sottoscrittori che hanno sottoscrizioni all'argomento della pubblicazione. Un messaggio pubblicato può essere utilizzato da più di un sottoscrittore.

Oltre a distribuire le pubblicazioni a tutti i sottoscrittori locali che dispongono di sottoscrizioni appropriate, un gestore code può anche distribuire la pubblicazione a qualsiasi altro gestore code ad esso connesso, direttamente o tramite una rete di gestori code che hanno sottoscrittori all'argomento.

In una rete di pubblicazione / sottoscrizione IBM MQ , un'applicazione di pubblicazione può essere anche un sottoscrittore.

Pubblicazioni nel punto di sincronizzazione

Gli autori (publisher) possono emettere chiamate MQPUT o MQPUT1 nel punto di sincronizzazione per includere tutti i messaggi consegnati ai sottoscrittori in un'unità di lavoro. Se l'opzione MQPMO_RETAIN o le opzioni di consegna dell'argomento NPMSGDLV o PMSGDLV con i valori ALL o ALLDUR sono specificate,

il gestore code utilizza le chiamate interne MQPUT o MQPUT1 nel punto di sincronizzazione, nell'ambito della chiamata MQPUT o MQPUT1 del publisher.

Informazioni di stato ed evento

Le pubblicazioni possono essere categorizzate come pubblicazioni di stato, come il prezzo corrente di un'azione, o pubblicazioni di eventi, come una compravendita di tale azione.

Pubblicazioni di stato

Le *pubblicazioni di stato* contengono informazioni sullo stato corrente di qualcosa, come il prezzo delle azioni o il punteggio corrente in una partita di calcio. Quando succede qualcosa (ad esempio, si verifica una modifica nel listino di borsa o nel risultato della partita di calcio), le informazioni relative allo stato precedente non vengono più richieste, poiché vengono sostituite dalle nuove informazioni.

Un sottoscrittore desidera ricevere la versione corrente delle informazioni di stato all'avvio e ricevere nuove informazioni ogni volta che lo stato cambia.

Se una pubblicazione contiene informazioni sullo stato, viene spesso pubblicata come una pubblicazione conservata. Un nuovo sottoscrittore in genere desidera che le informazioni sullo stato corrente vengano immediatamente; il sottoscrittore non desidera attendere un evento che causa la ripubblicazione delle informazioni. I sottoscrittori riceveranno automaticamente la pubblicazione conservata di un argomento quando lo sottoscriveranno, a meno che il sottoscrittore non utilizzi le opzioni MQSO_PUBLICATIONS_ON_REQUEST o MQSO_NEW_PUBLICATIONS_ONLY.

Pubblicazioni di eventi

Le *pubblicazioni di eventi* contengono informazioni sui singoli eventi che si verificano, ad esempio una compravendita in qualche borsa o il calcolo del punteggio di un determinato gol. Ciascun evento è indipendente dagli altri.

Un sottoscrittore desidera ricevere informazioni sugli eventi man mano che si verificano.

Pubblicazioni conservate

Per impostazione predefinita, dopo che una pubblicazione viene inviata a tutti i sottoscrittori interessati, viene eliminata. Tuttavia, un editore può specificare che venga conservata una copia di una pubblicazione in modo che possa essere inviata ai futuri sottoscrittori che registrano un interesse nell'argomento.

L'eliminazione delle pubblicazioni dopo che sono state inviate a tutti i sottoscrittori interessati è adatta per le informazioni sugli eventi, ma non è sempre adatta per le informazioni sullo stato. Conservando un messaggio, i nuovi sottoscrittori non devono attendere che le informazioni vengano pubblicate di nuovo prima di ricevere le informazioni di stato iniziali. Ad esempio, un sottoscrittore con una sottoscrizione a un prezzo azionario riceverà immediatamente il prezzo corrente, senza attendere che il prezzo del titolo cambi (e quindi venga ripubblicato).

Il gestore code può conservare solo una pubblicazione per ciascun argomento, quindi la pubblicazione conservata esistente di un argomento viene eliminata quando una nuova pubblicazione conservata arriva al gestore code. Tuttavia, l'eliminazione della pubblicazione esistente potrebbe non verificarsi in modo sincrono con l'arrivo della nuova pubblicazione conservata. Pertanto, ove possibile, non più di un editore che invia pubblicazioni conservate su qualsiasi argomento.

I sottoscrittori possono specificare che non desiderano ricevere le pubblicazioni conservate utilizzando l'opzione di sottoscrizione MQSO_NEW_PUBLICATIONS_ONLY. I sottoscrittori esistenti possono richiedere l'invio di copie duplicate delle pubblicazioni conservate.

In alcuni casi, è possibile che non si desideri conservare le pubblicazioni, anche per le informazioni sullo stato:

- Se tutte le sottoscrizioni ad un argomento vengono effettuate prima di qualsiasi pubblicazione su tale argomento e non si prevedono o non si consentono nuove sottoscrizioni, non è necessario conservare

le pubblicazioni perché vengono consegnate alla serie completa di sottoscrittori la prima volta che vengono pubblicate.

- Se le pubblicazioni si verificano frequentemente, ad esempio ogni secondo, un nuovo sottoscrittore (o un sottoscrittore in fase di ripristino da un errore) riceve lo stato corrente quasi immediatamente dopo la sottoscrizione iniziale, quindi non è necessario conservare tali pubblicazioni.
- Se le pubblicazioni sono di grandi dimensioni, potrebbe essere necessaria una notevole quantità di spazio di archiviazione per memorizzare la pubblicazione conservata per ogni argomento. In un ambiente con più gestori code, le pubblicazioni conservate sono memorizzate da tutti i gestori code della rete che hanno una sottoscrizione corrispondente.

Quando si decide se utilizzare le pubblicazioni conservate, considerare il modo in cui le applicazioni di sottoscrizione si ripristinano da un errore. Se l'autore (publisher) non utilizza pubblicazioni conservate, l'applicazione del sottoscrittore (subscriber) potrebbe aver necessità di memorizzare il relativo stato corrente in locale.

Per assicurarsi che una pubblicazione venga conservata, utilizzare l'opzione put - message MQPMO_RETAIN. Se viene utilizzata questa opzione e la pubblicazione non può essere conservata, il messaggio non viene pubblicato e la chiamata ha esito negativo con MQRC_PUT_NOT_USED.

Se un messaggio è una pubblicazione conservata, ciò viene indicato dalla proprietà del messaggio MQIsRetained . La persistenza di un messaggio è quella di quando è stato originariamente pubblicato.

Concetti correlati

Considerazioni di progettazione per le pubblicazioni conservate nei cluster di pubblicazione / sottoscrizione

Pubblicazioni nel punto di sincronizzazione

Nella pubblicazione / sottoscrizione IBM MQ , il punto di sincronizzazione può essere utilizzato dai publisher o internamente dal gestore code.

I publisher utilizzano il punto di sincronizzazione quando emettono chiamate MQPUT/MQPUT1 con opzione MQPMO_SYNCPOINT. Tutti i messaggi inviati ai sottoscrittori vengono conteggiati per il numero massimo di messaggi di cui non è stato eseguito il commit in un'unità di lavoro work.The MAXUMSGS specifica questo limite. Se il limite viene raggiunto, il publisher riceve il codice di errore 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_RAGGIUNTO .

Quando un publisher emette chiamate MQPUT/MQPUT1 utilizzando MQPMO_NO_SYNCPOINT con l'opzione MQPMO_RETAIN o le opzioni di consegna argomenti NPMSGDLV/PMSGDLV con valori ALL o ALLDUR, il gestore code utilizza punti di sincronizzazione interni per garantire che i messaggi vengano consegnati come richiesto. Il publisher può ricevere il codice motivo 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_RAGGIUNTI se il limite viene raggiunto nell'ambito della chiamata MQPUT/MQPUT1 del publisher.

Sottoscrittori e sottoscrizioni

Nella pubblicazione / sottoscrizione IBM MQ , un sottoscrittore è un'applicazione che richiede informazioni su un argomento specifico da un gestore code in una rete di pubblicazione / sottoscrizione. Un sottoscrittore può ricevere messaggi, sullo stesso argomento o su argomenti diversi, da più di un publisher.

Le sottoscrizioni possono essere create manualmente utilizzando un comando MQSC o dalle applicazioni. Queste sottoscrizioni vengono emesse per il gestore code locale e contengono informazioni sulle pubblicazioni che il sottoscrittore desidera ricevere:

- L'argomento a cui il sottoscrittore è interessato; questo può risolversi in più argomenti se vengono utilizzati i caratteri jolly.
- Una stringa di selezione facoltativa da applicare ai messaggi pubblicati.
- Un handle per una coda (nota come *coda del sottoscrittore*), su cui devono essere collocate le pubblicazioni selezionate e il CorrelIdfacoltativo.

Il gestore code locale memorizza le informazioni sulla sottoscrizione e, quando riceve una pubblicazione, esegue la scansione delle informazioni per stabilire se esiste una sottoscrizione che corrisponde all'argomento della pubblicazione e alla stringa di selezione. Per ogni sottoscrizione corrispondente, il gestore code indirizza la pubblicazione alla coda del sottoscrittore del sottoscrittore. Le informazioni che un gestore code memorizza sulle sottoscrizioni possono essere visualizzate utilizzando i comandi DIS SUB e DIS SBSTATUS.

Una sottoscrizione viene eliminata solo quando si verifica uno dei seguenti eventi:

- Il sottoscrittore (subscriber) annulla la sottoscrizione utilizzando la chiamata MQCLOSE (se la sottoscrizione è stata effettuata in modo non duraturo).
- La sottoscrizione scade.
- La sottoscrizione viene cancellata dal responsabile di sistema utilizzando il comando DELETE SUB.
- L'applicazione del sottoscrittore viene terminata (se la sottoscrizione è stata effettuata in modo non duraturo).
- Il gestore code viene arrestato o riavviato (se la sottoscrizione è stata effettuata in maniera non durevole).

Quando si ricevono i messaggi, utilizzare le opzioni appropriate sulla chiamata MQGET. Se l'applicazione elabora solo i messaggi per una sottoscrizione, è necessario utilizzare almeno `get-by-correlid`, come dimostrato nel programma di esempio C `amqssbxa.c` e all'indirizzo sottoscrittore MQ non gestito. Il **`CorrelId`** da utilizzare viene restituito da MQSUB in MQSD.Campo **`SubCorrelId`** .

Concetti correlati

Sottoscrizioni clonate e condivise

Riferimenti correlati

Esempi di come definire la proprietà sharedSubscription

Code gestite e pubblicazione / sottoscrizione

Quando si crea una sottoscrizione è possibile scegliere di utilizzare l'accodamento gestito. Se si utilizza l'accodamento gestito, una coda di sottoscrizione viene creata automaticamente quando si crea una sottoscrizione. Le code gestite vengono ridefinite automaticamente in base alla durata della sottoscrizione. L'utilizzo delle code gestite significa che non è necessario preoccuparsi della creazione di code per ricevere pubblicazioni e le pubblicazioni non utilizzate vengono rimosse automaticamente dalle code del sottoscrittore se viene chiusa una connessione di sottoscrizione non durevole.

Se un'applicazione non ha bisogno di utilizzare una particolare coda come coda del sottoscrittore, la destinazione per le pubblicazioni che riceve, può utilizzare le *sottoscrizioni gestite* utilizzando l'opzione di sottoscrizione MQSO_MANAGED. Se si crea una sottoscrizione gestita, il gestore code restituisce un handle di oggetto al sottoscrittore per una coda sottoscrittore creata dal gestore code in cui verranno ricevute le pubblicazioni. Ciò è dovuto al fatto che una *sottoscrizione gestita* è quella in cui IBM MQ gestisce la sottoscrizione. L'handle dell'oggetto della coda verrà restituito consentendo di sfogliare, richiamare o analizzare la coda (non è possibile inserire o impostare attributi di una coda gestita a meno che non sia stato esplicitamente concesso l'accesso alle code dinamiche temporanee).

La durata della sottoscrizione determina se la coda gestita rimane quando la connessione dell'applicazione di sottoscrizione al gestore code è interrotta.

Le sottoscrizioni gestite sono particolarmente utili quando vengono utilizzate con sottoscrizioni non durevoli perché quando la connessione dell'applicazione viene terminata, i messaggi non consumati rimarrebbero sulla coda del sottoscrittore occupando spazio indefinito nel gestore code. Se si utilizza una sottoscrizione gestita, la coda gestita sarà una coda dinamica temporanea e come tale verrà eliminata insieme ai messaggi non utilizzati quando la connessione viene interrotta per uno dei seguenti motivi:

- MQCLOSE con MQCO_REMOVE_SUB viene utilizzato e l'Hobj gestito viene chiuso.
- una connessione viene persa per un'applicazione che utilizza una sottoscrizione non durevole (MQSO_NON_DURABLE).
- una sottoscrizione viene rimossa perché è scaduta e l'Hobj gestito è chiuso.

Le sottoscrizioni gestite possono essere utilizzate anche con sottoscrizioni durevoli, ma è possibile che si desideri lasciare messaggi non utilizzati nella coda del sottoscrittore in modo che possano essere richiamati quando la connessione viene riaperta. Per questo motivo, le code gestite per le sottoscrizioni durevoli assumono la forma di una coda dinamica permanente e rimarranno quando la connessione dell'applicazione di sottoscrizione al gestore code viene interrotta.

È possibile impostare una scadenza sulla sottoscrizione se si desidera utilizzare una coda gestita dinamica permanente in modo che, anche se la coda esisterà ancora dopo che la connessione è stata interrotta, non continuerà a esistere indefinitamente.

Se si elimina la coda gestita, si riceverà un messaggio di errore.

Le code gestite che vengono create sono denominate con numeri alla fine (timestamp) in modo che ciascuno sia univoco.

Durata sottoscrizione

Le sottoscrizioni possono essere configurate come durevoli o non durevoli. La durata della sottoscrizione determina cosa succede alle sottoscrizioni quando le applicazioni di sottoscrizione si disconnettono da un gestore code.

Sottoscrizioni durevoli

Le sottoscrizioni durevoli continuano a esistere quando una connessione dell'applicazione di sottoscrizione al gestore code viene chiusa. Se una sottoscrizione è durevole, quando l'applicazione di sottoscrizione si disconnette, la sottoscrizione rimane attiva e può essere utilizzata dall'applicazione di sottoscrizione quando si riconnette richiedendo nuovamente la sottoscrizione utilizzando il **SubName** restituito quando è stata creata la sottoscrizione.

Quando si esegue la sottoscrizione in modo duraturo, un nome sottoscrizione (**SubName**) è obbligatorio. I nomi delle sottoscrizioni devono essere univoci all'interno di un gestore code in modo che possano essere utilizzati per identificare una sottoscrizione. Questo metodo di identificazione è necessario quando si specifica una sottoscrizione che si desidera riprendere, se la connessione alla sottoscrizione è stata deliberatamente chiusa (utilizzando l'opzione MQCO_KEEP_SUB) o è stata disconnessa dal gestore code. È possibile riprendere una sottoscrizione esistente utilizzando la chiamata MQSUB con opzione MQSO_RESUME. I nomi delle sottoscrizioni vengono visualizzati anche se si utilizza il comando DISPLAY SBSTATUS con SUBTYPE ALL o ADMIN.

Quando un'applicazione non richiede più una sottoscrizione duratura, è possibile rimuoverla utilizzando la chiamata alla funzione MQCLOSE con l'opzione MQCO_REMOVE_SUB oppure è possibile eliminarla manualmente utilizzando il comando MQSC DELETE SUB.

È possibile utilizzare l'attributo dell'argomento **DURSUB** per specificare se è possibile effettuare o meno sottoscrizioni durevoli a un argomento.

Al ritorno da una chiamata MQSUB che utilizza l'opzione MQSO_RESUME, la scadenza della sottoscrizione viene impostata sulla scadenza originale della sottoscrizione e non sulla scadenza rimanente.

Un gestore code continua a inviare pubblicazioni per soddisfare una sottoscrizione durevole anche se l'applicazione del sottoscrittore non è connessa. Ciò porta ad un accumulo di messaggi nella coda del sottoscrittore. Il modo più semplice per evitare questo problema è quello di utilizzare un abbonamento non durevole dove appropriato. Tuttavia, quando è necessario utilizzare sottoscrizioni durevoli, è possibile evitare una creazione di messaggi se il sottoscrittore (subscriber) effettua la sottoscrizione utilizzando l'opzione Pubblicazioni conservate . Un sottoscrittore può quindi controllare quando riceve le pubblicazioni utilizzando la chiamata MQSUBRQ.

Sottoscrizioni non durevoli

Le sottoscrizioni non durevoli esistono solo finché la connessione dell'applicazione di sottoscrizione al gestore code rimane aperta. La sottoscrizione viene rimossa quando l'applicazione di sottoscrizione si disconnette dal gestore code deliberatamente o a causa di un'interruzione. Quando la connessione viene chiusa, le informazioni relative alla sottoscrizione vengono rimosse dal gestore code e non vengono più

visualizzate se si visualizzano le sottoscrizioni utilizzando il comando DISPLAY SBSTATUS. Nessun altro messaggio viene inserito nella coda del sottoscrittore.

Ciò che accade alle pubblicazioni non utilizzate sulla coda del sottoscrittore per le sottoscrizioni non durevoli è determinato come segue.

- Se un'applicazione di sottoscrizione utilizza una destinazione gestita, tutte le pubblicazioni che non sono state utilizzate vengono rimosse automaticamente.
- Se l'applicazione di sottoscrizione fornisce un handle alla propria coda del sottoscrittore quando effettua la sottoscrizione, i messaggi non utilizzati non verranno rimossi automaticamente. È responsabilità dell'applicazione cancellare la coda, se appropriato. Se la coda è condivisa da più di un sottoscrittore o da altre applicazioni point - to - point, potrebbe non essere appropriato cancellare completamente la coda.

Sebbene non sia richiesto per sottoscrizioni non durevoli, il gestore code utilizza un nome di sottoscrizione, se fornito. I nomi delle sottoscrizioni devono essere univoci nel gestore code in modo che possano essere utilizzati per identificare una sottoscrizione.

Concetti correlati

Sottoscrizioni clonate e condivise

Attività correlate

Utilizzo delle sottoscrizioni condivise JMS 2.0

Riferimenti correlati

Esempi di come definire la proprietà sharedSubscription

Stringhe di selezione

Una *stringa di selezione* è un'espressione che viene applicata a una pubblicazione per determinare se corrisponde a una sottoscrizione. Le stringhe di selezione possono includere caratteri jolly.

Quando si esegue la sottoscrizione, oltre a specificare un argomento, è possibile specificare una stringa di selezione per selezionare le pubblicazioni in base alle relative proprietà del messaggio.

La stringa di selezione viene valutata rispetto al messaggio inserito dal publisher prima che venga modificato per la consegna a ogni sottoscrittore. Prestare attenzione quando si utilizzano i campi nella stringa di selezione che potrebbero essere modificati come parte dell'operazione di pubblicazione. Ad esempio, i campi MQMD `UserIdentifier`, `MsgIde` `CorrelId`.

Le stringhe di selezione non devono fare riferimento ai campi delle proprietà del messaggio aggiunti dal gestore code come parte dell'operazione di pubblicazione (vedere Proprietà del messaggio di pubblicazione / sottoscrizione), ad eccezione della proprietà del messaggio `MQTopicString`, che contiene la stringa dell'argomento per la pubblicazione.

Concetti correlati

Regole e limitazioni della stringa di selezione

Argomenti

Un argomento è l'oggetto delle informazioni pubblicate in un messaggio di pubblicazione / sottoscrizione.

I messaggi nei sistemi point-to-point vengono inviati a un indirizzo di destinazione specifico. I messaggi nei sistemi di pubblicazione / sottoscrizione basati sull'oggetto vengono inviati ai sottoscrittori in base all'oggetto che descrive il contenuto del messaggio. Nei sistemi basati sul contenuto, i messaggi vengono inviati agli utenti in base al contenuto del messaggio stesso.

Il sistema di pubblicazione / sottoscrizione IBM MQ è un sistema di pubblicazione / sottoscrizione basato sull'oggetto. Un publisher crea un messaggio e lo pubblica con una stringa di argomenti che meglio si adatta all'oggetto della pubblicazione. Per ricevere pubblicazioni, un sottoscrittore crea una sottoscrizione con una stringa di argomenti corrispondente al modello per selezionare gli argomenti di pubblicazione. Il gestore code consegna le pubblicazioni ai sottoscrittori che hanno sottoscrizioni che corrispondono all'argomento della pubblicazione e sono autorizzati a ricevere le pubblicazioni. L'articolo, “Stringhe argomento” a pagina 70, descrive la sintassi delle stringhe argomento che identificano l'argomento

di una pubblicazione. I sottoscrittori creano anche stringhe di argomenti per selezionare quali argomenti ricevere. Le stringhe di argomenti create dai sottoscrittori possono contenere uno dei due schemi di caratteri jolly alternativi per la corrispondenza del modello rispetto alle stringhe di argomenti nelle pubblicazioni. La corrispondenza del modello è descritta in “Schemi dei caratteri jolly” a pagina 71.

Nella pubblicazione / sottoscrizione basata sull'oggetto, i publisher o gli amministratori sono responsabili della classificazione dei soggetti in argomenti. In genere, i soggetti sono organizzati gerarchicamente, in strutture ad alberi di argomenti, utilizzando il carattere ' / ' per creare sottoargomenti nella stringa di argomento. Consultare “Strutture ad albero degli argomenti” a pagina 77 per esempi di strutture ad albero degli argomenti. Gli argomenti sono nodi nella struttura ad albero degli argomenti. Gli argomenti possono essere nodi foglia senza ulteriori argomenti secondari o nodi intermedi con argomenti secondari.

In parallelo con l'organizzazione dei soggetti in una struttura gerarchica di argomenti, è possibile associare gli argomenti agli oggetti argomento di gestione. Gli attributi vengono assegnati a un argomento, ad esempio se l'argomento viene distribuito in un cluster, associandolo a un oggetto argomento di gestione. L'associazione viene effettuata denominando l'argomento utilizzando l'attributo `TOPICSTR` dell'oggetto argomento di gestione. Se non si associa esplicitamente un oggetto argomento di gestione a un argomento, l'argomento eredita gli attributi del suo predecessore più vicino nella struttura ad albero degli argomenti *associata* a un oggetto argomento di gestione. Se non è stato definito alcun argomento principale, viene ereditato da `SYSTEM.BASE.TOPIC`. Gli oggetti argomento di gestione sono descritti in “Oggetti argomento di gestione” a pagina 78.

Nota: Anche se si ereditano tutti gli attributi di un argomento da `SYSTEM.BASE.TOPIC`, definire un argomento root per i propri argomenti che eredita direttamente da `SYSTEM.BASE.TOPIC`. Ad esempio, nello spazio argomento degli stati degli Stati Uniti, `USA/Alabama` `USA/Alaskae` così via, `USA` è l'argomento root. Lo scopo principale dell'argomento root è creare spazi argomento discreti e non sovrapposti per evitare che le pubblicazioni corrispondano alle sottoscrizioni errate. Ciò significa anche che è possibile modificare gli attributi dell'argomento root per influire sull'intero spazio argomento. Ad esempio, è possibile impostare il nome per l'attributo **CLUSTER** .

Quando si fa riferimento a un argomento come publisher o sottoscrittore, è possibile scegliere di fornire una stringa di argomenti o fare riferimento a un oggetto argomento. Oppure è possibile eseguire entrambe le operazioni, nel qual caso la stringa di argomenti fornita definisce un sottoargomento dell'oggetto argomento. Il gestore code identifica l'argomento accodando la stringa dell'argomento al prefisso della stringa dell'argomento indicato nell'oggetto argomento, inserendo un ulteriore ' / ' tra le due stringhe dell'argomento, ad esempio *stringa dell'argomento/stringa dell'oggetto*. “Combinazione di stringhe di argomenti” a pagina 75 lo descrive ulteriormente. La stringa di argomenti risultante viene utilizzata per identificare l'argomento e associarlo a un oggetto argomento di gestione. L'oggetto argomento di gestione non è necessariamente lo stesso oggetto argomento dell'oggetto argomento corrispondente all'argomento principale.

Nella pubblicazione / sottoscrizione basata sul contenuto, si definiscono i messaggi che si desidera ricevere fornendo stringhe di selezione che ricercano il contenuto di ogni messaggio. IBM MQ fornisce una forma intermedia di pubblicazione / sottoscrizione basata su contenuto utilizzando i selettori di messaggi che eseguono la scansione delle proprietà del messaggio piuttosto che del contenuto completo del messaggio, consultare Selettori. L'utilizzo archetipico dei selettori di messaggi consiste nel sottoscrivere un argomento e quindi qualificare la selezione con una proprietà numerica. Il selettore consente di specificare che si è interessati ai valori solo in un determinato intervallo; operazione che non è possibile eseguire utilizzando caratteri o caratteri jolly basati sull'argomento. Se è necessario filtrare in base al contenuto completo del messaggio, è necessario utilizzare IBM Integration Bus.

Stringhe argomento

Etichettare le informazioni pubblicate come argomento utilizzando una stringa di argomenti. Eseguire la sottoscrizione a gruppi di argomenti utilizzando stringhe di argomenti jolly basate su caratteri o argomenti.

Argomenti

Una *stringa argomento* è una stringa di caratteri che identifica l'argomento di un messaggio di pubblicazione / sottoscrizione. È possibile utilizzare qualsiasi carattere che si desidera quando si crea una stringa di argomenti.

Stringa argomento

Tre caratteri hanno un significato particolare nella pubblicazione / sottoscrizione IBM WebSphere MQ 7 . Sono consentiti in qualsiasi punto di una stringa di argomenti, ma sono utilizzati con cautela. L'uso dei caratteri speciali è spiegato in “Schema dei caratteri jolly basati su argomenti” a pagina 72.

Una barra (/)

Il separatore di livello argomento. Utilizzare il carattere `'/'` per strutturare l'argomento in una albero degli argomenti.

Evitare i livelli di argomento vuoti, `'//'`, se possibile. Questi corrispondono ai nodi nella gerarchia degli argomenti senza alcuna stringa di argomenti. Un `'/'` iniziale o finale in una stringa di argomento corrisponde a un nodo vuoto iniziale o finale e deve essere evitato.

Il segno hash (#)

Utilizzato in combinazione con `'/'` per creare un carattere jolly a più livelli nelle sottoscrizioni. Prestare attenzione all'utilizzo di `'#'` accanto a `'/'` nelle stringhe di argomento utilizzate per denominare gli argomenti pubblicati. “Esempi di stringhe di argomento” a pagina 71 mostra un uso ragionevole di `'#'`.

Le stringhe `'.../#/...'`, `'#/...'` e `'.../#'` hanno un significato speciale nelle stringhe degli argomenti di sottoscrizione. Le stringhe corrispondono a tutti gli argomenti a uno o più livelli nella gerarchia degli argomenti. Pertanto, se è stato creato un argomento con una di queste sequenze, non è possibile sottoscriverlo, senza sottoscrivere anche tutti gli argomenti a più livelli nella gerarchia degli argomenti.

Il segno più (+)

Utilizzato in combinazione con `'/'` per creare un carattere jolly a livello singolo nelle sottoscrizioni. Prestare attenzione all'utilizzo di `'+'` accanto a `'/'` nelle stringhe di argomento utilizzate per denominare gli argomenti pubblicati.

Le stringhe `'.../+/...'`, `'+/...'` e `'.../+'` hanno un significato speciale nelle stringhe degli argomenti di sottoscrizione. Le stringhe corrispondono a tutti gli argomenti ad un livello nella gerarchia degli argomenti. Pertanto, se è stato creato un argomento con una di queste sequenze, non è possibile sottoscriverlo, senza sottoscrivere anche tutti gli argomenti ad un unico livello nella gerarchia degli argomenti.

Esempi di stringhe di argomento

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

Riferimenti correlati

TOPIC

Schemi dei caratteri jolly

Esistono due schemi di caratteri jolly utilizzati per sottoscrivere più argomenti. La scelta dello schema è un'opzione di sottoscrizione.

MQSO_WILDCARD_TOPIC

Selezionare gli argomenti da sottoscrivere utilizzando lo schema di caratteri jolly basato sugli argomenti.

Questo è il valore predefinito se non viene selezionato esplicitamente alcuno schema di caratteri jolly.

MQSO_WILDCARD_CHAR
Selezionare gli argomenti da sottoscrivere utilizzando lo schema di caratteri jolly basato sui caratteri.

Impostare uno schema specificando il parametro **wschema** nel comando DEFINE SUB. Per ulteriori informazioni, vedere DEFINE SUB.

Nota: Le sottoscrizioni create prima di IBM WebSphere MQ 7.0 utilizzano sempre lo schema di caratteri jolly basato sui caratteri.

Esempi

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

Schema dei caratteri jolly basati su argomenti
I caratteri jolly basati sull'argomento consentono ai sottoscrittori di sottoscrivere più di un argomento alla volta.

I caratteri jolly basati su argomenti sono una potente funzione del sistema di argomenti nella pubblicazione / sottoscrizione IBM MQ . I caratteri jolly multilivello e di livello singolo possono essere usati per le sottoscrizioni, ma non possono essere usati all'interno di un argomento dal publisher del messaggio.

Lo schema di caratteri jolly basati su argomenti consente di selezionare le pubblicazioni raggruppate per livello di argomento. È possibile scegliere, per ogni livello nella gerarchia degli argomenti, se la stringa nella sottoscrizione per quel livello di argomento deve corrispondere esattamente alla stringa nella pubblicazione o meno. Ad esempio, la sottoscrizione `IBM/+/Results` seleziona tutti gli argomenti,

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

Esistono due tipi di caratteri jolly.

wildcard a più livelli

- Il carattere jolly multilivello viene utilizzato nelle sottoscrizioni. Quando viene utilizzato in una pubblicazione, viene considerato come un valore letterale.
- Il carattere jolly multilivello `'#'` viene utilizzato per corrispondere a qualsiasi numero di livelli all'interno di un argomento. Ad esempio, utilizzando la struttura ad albero degli argomenti di esempio, se si esegue la sottoscrizione a `'USA/Alaska/#'`, si ricevono messaggi sugli argomenti `'USA/Alaska'` e `'USA/Alaska/Juneau'`.
- Il carattere jolly multilivello può rappresentare zero o più livelli. Pertanto, `'USA/#'` può anche corrispondere al singolare `'USA'`, dove `'#'` rappresenta zero livelli. Il separatore di livello argomento non ha senso in questo contesto, perché non ci sono livelli da separare.
- Il carattere jolly multilivello è valido solo quando viene specificato da solo o accanto al carattere separatore del livello di argomento. Pertanto, `'#'` e `'USA/#'` sono argomenti validi in cui il carattere `'#'` viene considerato come un carattere jolly. Tuttavia, sebbene `'USA#'` sia anche una stringa di argomenti valida, il carattere `'#'` non viene considerato come un carattere jolly e non ha alcun significato speciale. Per ulteriori informazioni, consultare “Quando i caratteri jolly basati sugli argomenti non sono jolly” a pagina 74 .

Carattere jolly di livello singolo

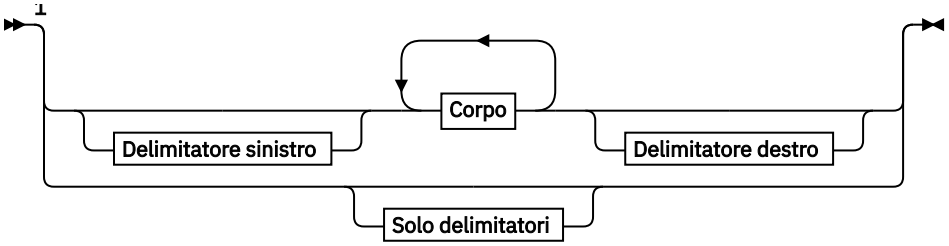
- Il singolo carattere jolly viene utilizzato nelle sottoscrizioni. Quando viene utilizzato in una pubblicazione, viene considerato come un valore letterale.
- Il carattere jolly a livello singolo `'+'` corrisponde ad un solo livello di argomento. Ad esempio, `'USA/+'` corrisponde a `'USA/Alabama'`, ma non `'USA/Alabama/Auburn'`. Poiché il carattere jolly a livello singolo corrisponde solo ad un singolo livello, `'USA/+'` non corrisponde a `'USA'`.

- Il carattere jolly a livello singolo può essere utilizzato a qualsiasi livello nella struttura ad albero degli argomenti e insieme al carattere jolly multilivello. Il carattere jolly a livello singolo deve essere specificato accanto al separatore di livello argomento, tranne quando è specificato da solo. Pertanto, `'+'` e `'USA/+'` sono argomenti validi in cui il carattere `'+'` viene considerato come un carattere jolly. Tuttavia, sebbene `'USA+'` sia anche una stringa di argomenti valida, il carattere `'+'` non viene considerato come un carattere jolly e non ha alcun significato speciale. Per ulteriori informazioni, consultare “Quando i caratteri jolly basati sugli argomenti non sono jolly” a pagina 74 .

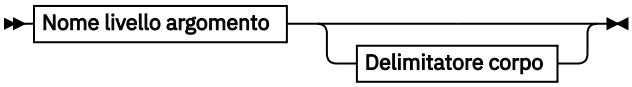
La sintassi per lo schema del carattere jolly basato sull'argomento non ha caratteri escape. Il fatto che `'#'` e `'+'` vengano trattati o meno come caratteri jolly dipende dal loro contesto. Per ulteriori informazioni, consultare “Quando i caratteri jolly basati sugli argomenti non sono jolly” a pagina 74 .

Nota: L'inizio e la fine di una stringa di argomenti vengono trattati in modo speciale. Utilizzando `'$'` per indicare la fine della stringa, `'$#/...'` è un carattere jolly multilivello e `'$/#/..'`. è un nodo vuoto nella root, seguito da un carattere jolly multilivello.

Stringa di caratteri jolly basata sull'argomento

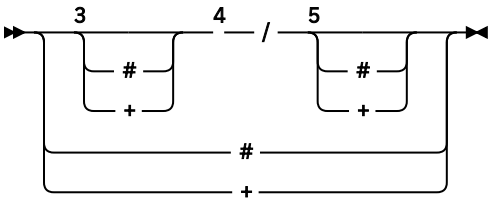


Corpo



Nome livello argomento

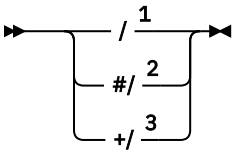
Solo delimitatori



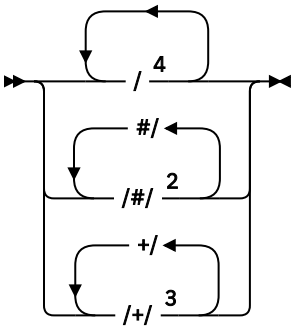
Note:

[1] Una stringa di argomento di lunghezza null o zero non è valida
[2] Si consiglia di non utilizzare `*`, `?`, `%` nelle stringhe dei nomi di livello per la compatibilità tra schemi di caratteri jolly basati su caratteri e su argomenti.
[3] Questi casi sono equivalenti al pattern *delimitatore di sinistra* .
[4] / senza caratteri jolly corrisponde a un singolo argomento vuoto.
[5] Questi casi sono equivalenti al pattern *delimitatore di destra* .
[6] Associa ogni argomento.
[7] Corrisponde a ogni argomento in cui è presente un solo livello.

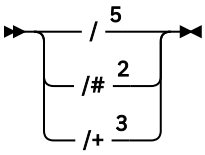
Delimitatore di sinistra



Delimitatore corpo



Delimitatore di destra



Note:

[1] La stringa dell'argomento inizia con un argomento vuoto
[2] Corrisponde a zero o più livelli. Più stringhe di corrispondenza multilivello hanno lo stesso effetto di una stringa di corrispondenza multilivello.
[3] Corrisponde esattamente a un livello.
[4] // è un argomento vuoto - un oggetto argomento senza stringa di argomenti.
[5] La stringa di argomenti termina con un argomento vuoto

Quando i caratteri jolly basati sugli argomenti non sono jolly

I caratteri jolly '+' e '#' non hanno alcun significato speciale quando sono combinati con altri caratteri (inclusi se stessi) in un livello di argomento.

Ciò significa che gli argomenti che contengono '+' o '#' insieme ad altri caratteri in un livello di argomento possono essere pubblicati.

Per esempio, considerare i due argomenti seguenti:

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

Nel primo esempio, i caratteri '+' e '#' vengono trattati come caratteri jolly e quindi non sono validi in una stringa di argomenti che deve essere pubblicata ma che sono validi in una sottoscrizione.

Nel secondo esempio, i caratteri '+' e '#' non vengono considerati come caratteri jolly e quindi la stringa di argomenti può essere sia pubblicata che sottoscritta.

Esempi

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

Schema di caratteri jolly basato su caratteri
Lo schema di caratteri jolly basato sui caratteri consente di selezionare argomenti basati sulla corrispondenza dei caratteri tradizionali.

È possibile selezionare tutti gli argomenti a più livelli in una gerarchia di argomenti utilizzando la stringa `'*'`. L'utilizzo di `'*'` nello schema di caratteri jolly basato sui caratteri equivale all'utilizzo della stringa di caratteri jolly basata sull'argomento `'#'`

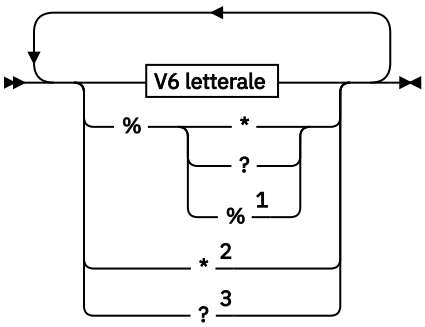
`' x/*/y '` è equivalente a `' x/#/y '` nello schema basato sull'argomento e seleziona tutti gli argomenti nella gerarchia degli argomenti tra i livelli `' x` e `y '`, dove `' x '` e `' y '` sono nomi di argomenti che non si trovano nella serie di livelli restituiti dal carattere jolly.

`'/+/'` nello schema basato sull'argomenti non dispone di un equivalente esatto nello schema basato sui caratteri. `'IBM/*/Results'` seleziona anche `'IBM/Patents/Software/Results'`. Solo se la serie di nomi argomento a ciascun livello della gerarchia è univoca, è sempre possibile costruire query con i due schemi che producono corrispondenze identiche.

Utilizzati in modo generale, `'*'` e `'?'` nello schema basato su caratteri non hanno equivalenti nello schema basato sugli argomenti. Lo schema basato sull'argomento non esegue una corrispondenza parziale utilizzando caratteri jolly. La sottoscrizione del carattere jolly basata sui caratteri `'IBM/*ware/Results'` non ha un equivalente basato sull'argomento.

Nota: Le corrispondenze che utilizzano sottoscrizioni con caratteri jolly sono più lente delle corrispondenze che utilizzano sottoscrizioni basate su argomenti.

Stringa di caratteri jolly basata su caratteri



V6 letterale

Note:

[1] Indica l'escape del seguente carattere, in modo che venga considerato come un letterale. `'%'` deve essere seguito da `'*'`, `'?'` o `'%'`. Consultare “Esempi di stringhe di argomento” a pagina 71.
[2] Indica la corrispondenza di zero o più caratteri in una sottoscrizione.
[3] Indica che corrisponde esattamente a un carattere in una sottoscrizione.

Esempi

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

Combinazione di stringhe di argomenti

Quando si creano sottoscrizioni o si aprono argomenti in modo da poter pubblicare messaggi, la stringa di argomenti può essere formata combinando due stringhe di argomenti secondari separate o "argomenti secondari". Un argomento secondario viene fornito dall'applicazione o dal comando di gestione come stringa di argomenti e l'altro è la stringa di argomenti associata a un oggetto argomento. È possibile

utilizzare un argomento secondario come stringa di argomenti da solo o combinarli per formare un nuovo nome di argomento.

Ad esempio, quando si definisce una sottoscrizione utilizzando il comando MQSC **DEFINE SUB**, il comando può utilizzare **TOPICSTR** (stringa argomento) o **TOPICOBJ** (oggetto argomento) come attributo o entrambi insieme. Se viene fornito solo **TOPICOBJ** , la stringa argomento associata a tale oggetto argomento viene utilizzata come stringa argomento. Se viene fornito solo **TOPICSTR** , viene utilizzato come stringa di argomenti. Se vengono forniti entrambi, vengono concatenati per formare una singola stringa di argomenti nel modulo **TOPICOBJ** / **TOPICSTR**, dove la stringa di argomenti configurata **TOPICOBJ** è sempre la prima e le due parti della stringa sono sempre separate da un carattere "/" .

Allo stesso modo, in un programma MQI il nome completo dell'argomento viene creato da MQOPEN. Si compone di due campi utilizzati nelle chiamate MQI di pubblicazione / sottoscrizione, nell'ordine elencato:

1. L'attributo **TOPICSTR** dell'oggetto argomento, denominato nel campo **ObjectName** .
2. Il parametro **ObjectString** che definisce l'argomento secondario fornito dall'applicazione.

La stringa di argomenti risultante viene restituita nel parametro **ResObjectString** .

Questi campi vengono considerati come presenti se il primo carattere di ogni campo non è uno spazio vuoto o un carattere null e la lunghezza del campo è maggiore di zero. Se è presente solo uno dei campi, viene utilizzato non modificato come nome dell'argomento. Se nessuno dei due campi ha un valore, la chiamata ha esito negativo con codice di errore MQRC_UNKNOWN_OBJECT_NAMEo MQRC_TOPIC_STRING_ERROR se il nome completo dell'argomento non è valido.

Se sono presenti entrambi i campi, viene inserito un carattere "/" tra i due elementi del nome argomento combinato risultante.

La seguente tabella mostra esempi di concatenazione di stringhe di argomenti:

Tabella 2. Esempi di concatenazione di stringhe di argomenti

TOPICSTR dell'oggetto argomento	**Stringa argomento fornita dall'applicazione o dal comando DEFINE SUB**	**Nome argomento completo**	**Commento**
Calcio / Punteggi	' '	Calcio / Punteggi	Il TOPICSTR dell'oggetto argomento viene utilizzato da solo.
' '	Calcio / Punteggi	Calcio / Punteggi	ObjectString/TOPICSTR viene utilizzato da solo.
football	Punteggi	Calcio / Punteggi	Un carattere "/" viene aggiunto al punto di concatenazione.
football	/Punteggi	Calcio / /Punteggi	Viene prodotto un 'nodo vuoto ' tra le due stringhe. È diverso da "Calcio / Punteggi".
/Calcio	Punteggi	/Calcio / Punteggi	L'argomento inizia con un 'nodo vuoto '. È diverso da "Calcio / Punteggi".

Il carattere "/" viene considerato come un carattere speciale, fornendo una struttura al nome completo dell'argomento in“Strutture ad albero degli argomenti” a pagina 77. Il carattere "/" non deve essere utilizzato per altri motivi, poiché la struttura della struttura ad albero degli argomenti è interessata. L'argomento "/Football" non è uguale all'argomento "Football".

Nota: Se si utilizza un oggetto argomento durante la creazione di una sottoscrizione, il valore della stringa di argomenti dell'oggetto argomento viene fissato nella sottoscrizione al momento della definizione. Qualsiasi modifica successiva all'oggetto argomento non influisce sulla stringa argomento per cui è definita la sottoscrizione.

Caratteri jolly nelle stringhe di argomenti

I seguenti caratteri jolly sono caratteri speciali:

- segno più (+)
- segno numerico (#)
- asterisco (*)
- punto interrogativo (?)

I caratteri jolly hanno un significato speciale solo se utilizzati da una sottoscrizione. Questi caratteri non sono considerati non validi quando vengono utilizzati altrove, tuttavia è necessario accertarsi di comprendere come vengono utilizzati e si potrebbe preferire di non utilizzare questi caratteri nelle stringhe argomento durante la pubblicazione o la definizione di oggetti argomento.

Se si esegue la pubblicazione su una stringa di argomenti con # o + combinati con altri caratteri (inclusi se stessi) all'interno di un livello di argomenti, la stringa di argomenti può essere sottoscritta con uno schema di caratteri jolly.

Se si pubblica su una stringa di argomenti con # o + come unico carattere compreso tra due / caratteri, la stringa di argomenti non può essere sottoscritta esplicitamente da un'applicazione utilizzando lo schema di caratteri jolly MQSO_WILDCARD_TOPIC. Questa situazione fa sì che l'applicazione ottena un numero di pubblicazioni superiore a quello previsto.

Non utilizzare un carattere jolly nella stringa dell'argomento di un oggetto argomento definito. In questo caso, il carattere viene considerato come un carattere letterale quando l'oggetto viene utilizzato da un publisher e come un carattere jolly quando viene utilizzato da una sottoscrizione. Ciò può portare a confusione.

Frammento di codice di esempio

Questo frammento di codice, estratto dal programma di esempio Esempio 2: Publisher su un argomento variabile, combina un oggetto argomento con una stringa argomento variabile:

```
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);
```

Strutture ad albero degli argomenti

Ciascun argomento che viene definito è un elemento, o nodo, della struttura ad albero degli argomenti. La struttura ad albero degli argomenti può essere vuota per iniziare o contenere argomenti che sono stati precedentemente definiti utilizzando i comandi MQSC o PCF. È possibile definire un nuovo argomento utilizzando i comandi di creazione degli argomenti o specificando l'argomento per la prima volta in una pubblicazione o in una sottoscrizione.

Sebbene sia possibile utilizzare qualsiasi stringa di caratteri per definire una stringa di argomenti, è consigliabile scegliere una stringa di argomenti che si adatti a una struttura ad albero gerarchica. Una progettazione accurata delle punte e delle strutture ad albero degli argomenti può essere utile per le seguenti operazioni:

- La sottoscrizione a più argomenti.

- Stabilire criteri di sicurezza.

Sebbene sia possibile costruire una struttura ad albero degli argomenti come una struttura lineare, è meglio crearne una in una struttura gerarchica con uno o più argomenti root. Per ulteriori informazioni sulla pianificazione della protezione e sugli argomenti, consultare Sicurezza di pubblicazione / sottoscrizione.

Figura 18 a pagina 78 mostra un esempio di una struttura ad albero di argomenti con un argomento root.

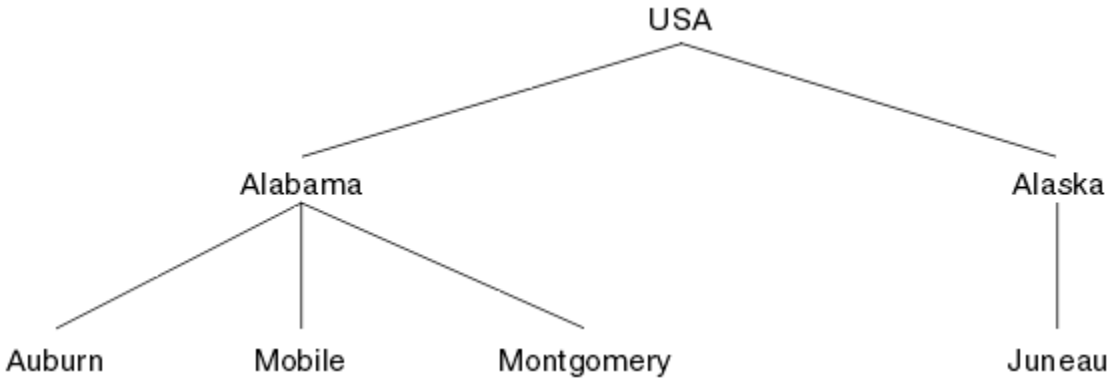


Figura 18. Esempio di struttura ad albero degli argomenti

Ciascuna stringa di caratteri nella figura rappresenta un nodo nella struttura ad albero degli argomenti. Una stringa di argomenti completa viene creata aggregando i nodi da uno o più livelli nella struttura di argomenti. I livelli sono separati dal carattere "/". Il formato di una stringa di argomenti completamente specificata è:root/level2/level3".

Gli argomenti validi nella struttura ad albero degli argomenti mostrata in Figura 18 a pagina 78 sono:

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

Quando si progettano le stringhe di argomenti e le strutture ad albero degli argomenti, tenere presente che il gestore code non interpreta o tenta di derivare il significato dalla stringa di argomenti stessa. Utilizza semplicemente la stringa di argomenti per inviare messaggi selezionati ai sottoscrittori di tale argomento.

Gli argomenti riportati di seguito sono validi per la costruzione e il contenuto di una struttura ad albero dell'argomento:

- Non esistono limiti relativi al numero di livelli di una struttura ad albero dell'argomento.
- Non esistono limiti relativi alla lunghezza del nome di un livello in una struttura ad albero dell'argomento.
- Può essere presente un numero qualsiasi di nodi "root"; ovvero, può essere presente un numero qualsiasi di strutture ad albero degli argomenti.

Attività correlate

Riduzione del numero di argomenti indesiderati nella struttura ad albero degli argomenti

Oggetti argomento di gestione

Utilizzando un oggetto argomento di amministrazione, è possibile assegnare specifici attributi non predefiniti agli argomenti.

Figura 19 a pagina 79 mostra in che modo un argomento di alto livello di `Sport` suddiviso in argomenti separati che coprono diversi sport può essere visualizzato come una struttura di argomenti:

Figura 19. Visualizzazione di una struttura ad albero degli argomenti

Figura 20 a pagina 79 mostra come la struttura ad albero degli argomenti può essere ulteriormente divisa, per separare diversi tipi di informazioni su ogni sport:

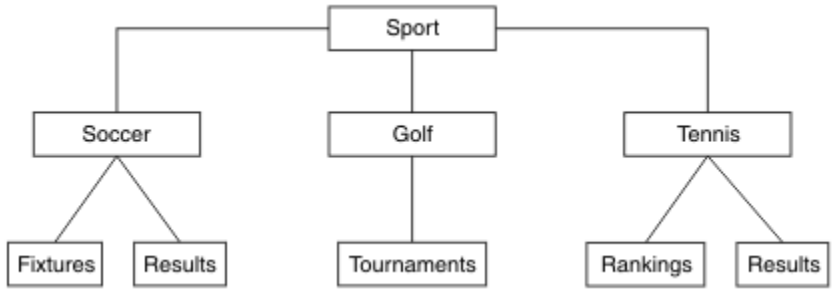


Figura 20. Struttura ad albero degli argomenti estesa

Per creare la struttura di argomenti illustrata, non è necessario definire alcun oggetto di argomento di gestione. Ciascuno dei nodi in questa struttura ad albero è definito da una stringa di argomenti creata in un'operazione di pubblicazione o sottoscrizione. Ogni argomento nella struttura ad albero eredita i propri attributi dal relativo parent. Gli attributi vengono ereditati dall'oggetto argomento principale, perché per impostazione predefinita tutti gli attributi sono impostati su ASPARENT. In questo esempio, ogni argomento ha gli stessi attributi dell'argomento `Sport` . L'argomento `Sport` non ha alcun oggetto argomento di gestione ed eredita i relativi attributi da SYSTEM.BASE.TOPIC.

Notare che non è consigliabile fornire autorizzazioni per gli utenti non mqm sul nodo root della struttura ad albero degli argomenti, che è SYSTEM.BASE.TOPIC, poiché le autorizzazioni sono ereditate ma non possono essere limitate. Pertanto, fornendo le autorizzazioni a questo livello, si forniscono le autorizzazioni all'intero albero. È necessario concedere l'autorità ad un livello di argomento inferiore nella gerarchia.

Gli oggetti argomento di gestione possono essere utilizzati per definire attributi specifici per particolari nodi nella struttura ad albero degli argomenti. Nel seguente esempio, l'oggetto argomento di gestione viene definito per impostare la proprietà delle sottoscrizioni durevoli DURSUB dell'argomento soccer sul valore NO:

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

La struttura ad albero degli argomenti può ora essere visualizzata come:

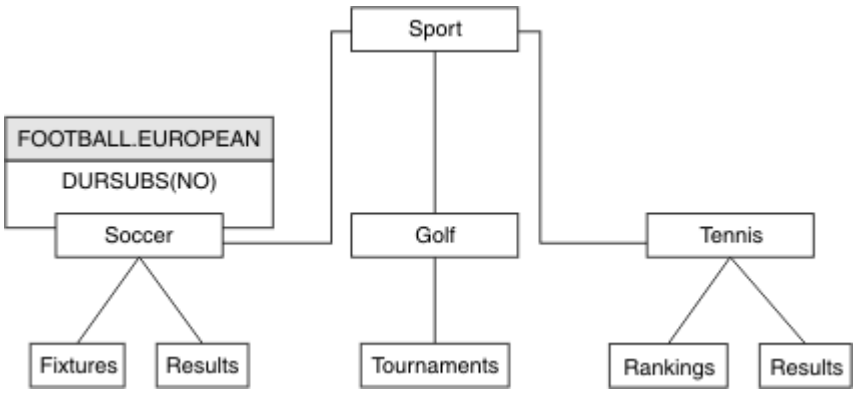


Figura 21. Visualizzazione di un oggetto di argomento amministrativo associato all'argomento Sport / Soccer

Le applicazioni che sottoscrivono gli argomenti sotto Soccer nella struttura ad albero possono ancora utilizzare le stringhe degli argomenti utilizzate prima dell'aggiunta dell'oggetto argomento di gestione. Tuttavia, è ora possibile scrivere un'applicazione per la sottoscrizione utilizzando il nome oggetto `FOOTBALL.EUROPEAN`, invece della stringa `/Sport/Soccer`. Ad esempio, per sottoscrivere `/Sport/Soccer/Results`, un'applicazione può specificare `MQSD.ObjectName` come `FOOTBALL.EUROPEAN` e `MQSD.ObjectString` come `Results`.

Con questa funzione, è possibile nascondere parte della struttura ad albero degli argomenti agli sviluppatori dell'applicazione. Definire un oggetto argomento di gestione in un nodo particolare nella struttura ad albero degli argomenti, quindi gli sviluppatori dell'applicazione possono definire i propri argomenti come elementi secondari del nodo. Gli sviluppatori devono conoscere l'argomento principale, ma non tutti gli altri nodi nella struttura ad albero principale.

Eredità di attributi

Se una struttura ad albero degli argomenti dispone di molti oggetti argomento di gestione, ciascun oggetto argomento di gestione, per impostazione predefinita, eredita i propri attributi dall'argomento di gestione principale più vicino. L'esempio precedente è stato esteso in Figura 22 a pagina 80:

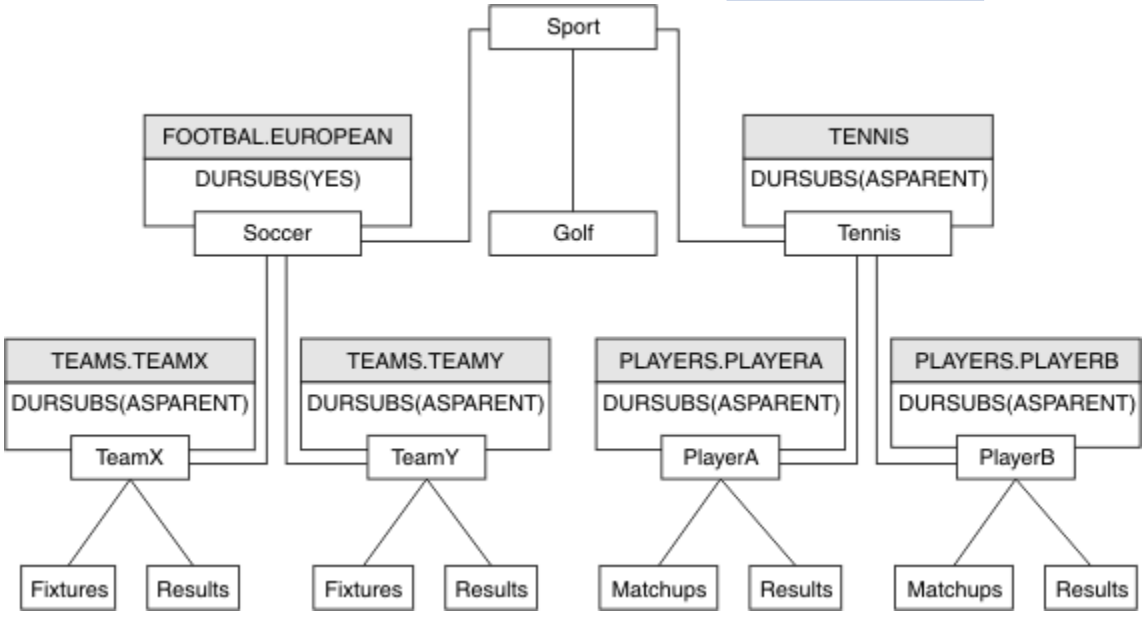


Figura 22. Struttura ad albero degli argomenti con diversi oggetti argomento di gestione

Ad esempio, utilizzare l'eredità per fornire a tutti gli argomenti secondari di `/Sport/Soccer` la proprietà che le sottoscrizioni non sono durevoli. Modificare l'attributo `DURSUB` di `FOOTBALL.EUROPEAN` in `NO`.

Questo attributo può essere impostato utilizzando il comando seguente:

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

Tutti gli oggetti argomento di gestione degli argomenti secondari di `Sport/Soccer` hanno la proprietà DURSUB impostata sul valore predefinito ASPARENT. Dopo aver modificato il valore della proprietà DURSUB di `FOOTBALL.EUROPEAN` in `NO`, gli argomenti secondari di `Sport/Soccer` ereditano il DURSUB valore della proprietà `NO`. Tutti gli argomenti child di `Sport/Tennis` ereditano il valore DURSUB dall'oggetto SYSTEM.BASE.TOPIC . SYSTEM.BASE.TOPIC ha il valore YES.

Il tentativo di effettuare una sottoscrizione durevole all'argomento `Sport/Soccer/TeamX/Results` non è riuscito; tuttavia, il tentativo di effettuare una sottoscrizione durevole a `Sport/Tennis/PlayerB/Results` avrà esito positivo.

Controllo dell'utilizzo dei caratteri jolly con la proprietà WILDCARD

Utilizzare la proprietà MQSC **Topic** WILDCARD o la proprietà Argomento `WildcardOperation` PCF equivalente per controllare la consegna di pubblicazioni alle applicazioni del sottoscrittore che utilizzano nomi stringa di argomenti con caratteri jolly. La proprietà WILDCARD può avere uno dei due valori possibili:

WILDCARD
Il funzionamento delle sottoscrizioni con caratteri jolly rispetto a questo argomento.

PASSTHRU
Le sottoscrizioni effettuate a un argomento con carattere jolly meno specifico della stringa argomento in questo oggetto argomento riceveranno le pubblicazioni relative a questo argomento e a stringhe argomento più specifiche di tale argomento.

BLOCK
Le sottoscrizioni effettuate a un argomento con carattere jolly meno specifico della stringa argomento in questo oggetto argomento non riceveranno le pubblicazioni relative a questo argomento o a stringhe argomento più specifiche di tale argomento.

Il valore di questo attributo è utilizzato quando vengono definite sottoscrizioni. Se si modifica questo attributo, la serie di argomenti trattati dalle sottoscrizioni esistenti non viene interessata dalla modifica. Questo scenario si applica anche se la topologia cambia quando si creano o eliminano oggetti argomento; la serie di argomenti che corrispondono alle sottoscrizioni create in seguito alla modifica dell'attributo WILDCARD viene creata utilizzando la topologia modificata. Se si desidera forzare una rivalutazione della serie corrispondente di argomenti per le sottoscrizioni esistenti, è necessario riavviare il gestore code.

Nell'esempio, “Esempio: creazione del cluster di pubblicazione / sottoscrizione Sport” a pagina 85, è possibile seguire la procedura per creare la struttura ad albero dell'argomento mostrata in Figura 23 a pagina 82.

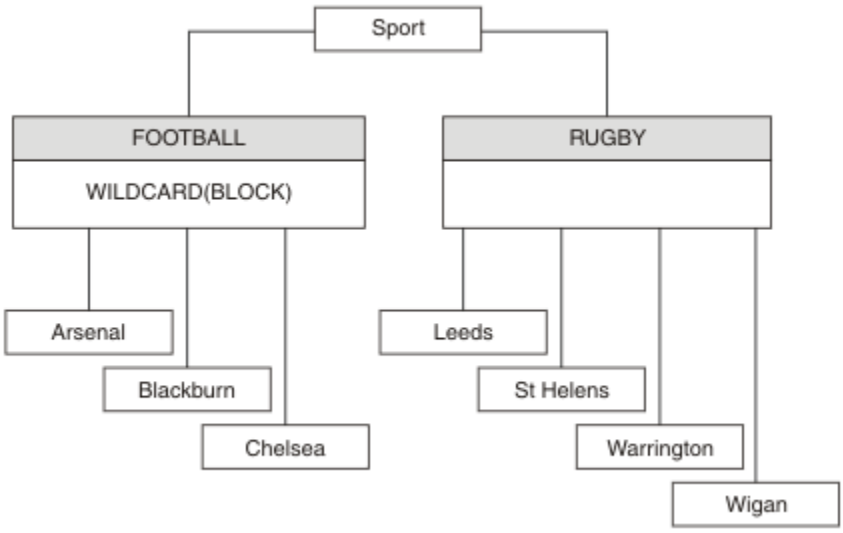


Figura 23. Una struttura di argomenti che utilizza la proprietà WILDCARD , BLOCK

Un sottoscrittore che utilizza la stringa di argomenti con caratteri jolly # riceve tutte le pubblicazioni nell'argomento `Sport` e nella struttura ad albero secondaria `Sport/Rugby` . Il sottoscrittore non riceve alcuna pubblicazione nella struttura secondaria `Sport/Football` , poiché il valore della proprietà `WILDCARD` dell'argomento `Sport/Football` è `BLOCK`.

`PASSTHRU` è l'impostazione predefinita. È possibile impostare il valore della proprietà `WILDCARD` `PASSTHRU` sui nodi nella struttura ad albero `Sport` . Se i nodi non hanno il valore della proprietà `WILDCARD` `BLOCK`, l'impostazione `PASSTHRU` non modifica il funzionamento osservato dai sottoscrittori ai nodi nella albero `Sports` .

Nell'esempio, creare sottoscrizioni per vedere in che modo l'impostazione del carattere jolly influisce sulle pubblicazioni fornite; consultare Figura 27 a pagina 87. Eseguire il comando publish in Figura 30 a pagina 88 per creare alcune pubblicazioni.

```
pub QMA
```

Figura 24. Pubblica in QMA

I risultati sono riportati in Tabella 3 a pagina 82. Si noti come l'impostazione del valore della proprietà `WILDCARD` `BLOCK`impedisca alle sottoscrizioni con caratteri jolly di ricevere pubblicazioni per argomenti nell'ambito del carattere jolly.

Tabella 3. Pubblicazioni ricevute su QMA

Sottoscrizione	**Stringa argomento**	**Pubblicazioni ricevute**	**Note**

SPORTS	`Sports/#`	`Sports`
		`Sports/Rugby`
		`Sports/Rugby/Leeds`
			Tutte le pubblicazioni della sottostruttura Football bloccate da `WILDCARD(BLOCK)` su `Sports/Football`

SARSENAL	`Sports/#/Arsenal`	-	`WILDCARD(BLOCK)` on `Sports/Football` impedisce la sottoscrizione di caratteri jolly su `Arsenal`

Tabella 3. Pubblicazioni ricevute su QMA (Continua)			
Sottoscrizione	**Stringa argomento**	**Pubblicazioni ricevute**	**Note**
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Il valore predefinito WILDCARD su Sports / Rugby non impedisce la sottoscrizione di caratteri jolly su Leeds.

Nota:

Si supponga che una sottoscrizione abbia un carattere jolly che corrisponde a un oggetto argomento con il valore della proprietà WILDCARD BLOCK. Se la sottoscrizione ha anche una stringa di argomenti a destra del carattere jolly corrispondente, la sottoscrizione non riceverà mai una pubblicazione. La serie di pubblicazioni che non sono bloccate sono pubblicazioni per argomenti che sono parent del carattere jolly bloccato. Le pubblicazioni per gli argomenti child dell'argomento con il valore della proprietà BLOCK sono bloccate dal carattere jolly. Pertanto, le stringhe di argomenti di sottoscrizione che includono un argomento alla destra del carattere jolly non ricevono mai alcuna pubblicazione corrispondente.

L'impostazione del valore della proprietà WILDCARD su BLOCK non significa che non è possibile effettuare la sottoscrizione utilizzando una stringa di argomenti che include caratteri jolly. Tale sottoscrizione è normale. La sottoscrizione dispone di un argomento esplicito che corrisponde all'argomento con un oggetto argomento con un valore della proprietà WILDCARD BLOCK. Utilizza i caratteri jolly per gli argomenti che sono parent o child dell'argomento con il valore della proprietà WILDCARD BLOCK. Nell'esempio in Figura 23 a pagina 82, una sottoscrizione come Sports/Football/# può ricevere pubblicazioni.

Caratteri jolly e argomenti cluster

Le definizioni degli argomenti del cluster vengono propagati a ogni gestore code in un cluster. Una sottoscrizione a un argomento cluster in un gestore code in un cluster determina la creazione di sottoscrizioni proxy da parte del gestore code. Una sottoscrizione proxy viene creata su ogni altro gestore code nel cluster. Le sottoscrizioni che utilizzano stringhe di argomenti contenenti caratteri jolly, combinate con argomenti cluster, possono fornire un comportamento difficile da prevedere. Il comportamento viene spiegato nel seguente esempio.

Nel cluster configurato per l'esempio, “Esempio: creazione del cluster di pubblicazione / sottoscrizione Sport” a pagina 85, QMB ha la stessa serie di sottoscrizioni di QMA, ma QMB non ha ricevuto alcuna pubblicazione dopo che il publisher è stato pubblicato in QMA, consultare Figura 24 a pagina 82. Anche se gli argomenti Sports/Football e Sports/Rugby sono argomenti del cluster, le sottoscrizioni definite in fullsubs.tst non fanno riferimento a un argomento del cluster. Nessuna sottoscrizione proxy viene propagata da QMB a QMA. Senza sottoscrizioni proxy, nessuna pubblicazione per QMA viene inoltrata a QMB.

Alcune sottoscrizioni, come Sports/#/Leeds, potrebbero far riferimento a un argomento cluster, in questo caso Sports/Rugby. La sottoscrizione Sports/#/Leeds viene effettivamente risolta nell'oggetto argomento SYSTEM.BASE.TOPIC.

La regola per risolvere l'oggetto argomento a cui fa riferimento una sottoscrizione come, Sports/#/Leeds è la seguente. Troncare la stringa argomento al primo carattere jolly. Eseguire la scansione a sinistra nella stringa dell'argomento cercando il primo argomento a cui è associato un oggetto argomento di gestione. L'oggetto argomento potrebbe specificare un nome cluster o definire un oggetto argomento locale. Nell'esempio, Sports/#/Leeds, la stringa di argomenti dopo il troncamento è Sports, che non ha alcun oggetto argomento e quindi Sports/#/Leeds eredita da SYSTEM.BASE.TOPIC, che è un oggetto argomento locale.

Per vedere come la sottoscrizione agli argomenti del cluster può modificare il funzionamento della propagazione dei caratteri jolly, eseguire lo script batch upsubs.bat. Lo script elimina le code di sottoscrizione e aggiunge le sottoscrizioni dell'argomento del cluster in fullsubs.tst. Eseguire nuovamente puba.bat per creare un batch di pubblicazioni; consultare Figura 24 a pagina 82.

Tabella 4 a pagina 84 mostra il risultato dell'aggiunta di due nuove sottoscrizioni allo stesso gestore code su cui sono state pubblicate le pubblicazioni. Il risultato è quello previsto, le nuove sottoscrizioni ricevono una pubblicazione ciascuna e il numero di pubblicazioni ricevute dalle altre sottoscrizioni è invariato. I risultati imprevisti si verificano sull'altro gestore code cluster; consultare Tabella 5 a pagina 84.

Tabella 4. Pubblicazioni ricevute su QMA

Sottoscrizione	**Stringa argomento**	**Pubblicazioni ricevute**	**Note**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Tutte le pubblicazioni della sottostruttura Football bloccate da WILDCARD(BLOCK) su Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football impedisce la sottoscrizione di caratteri jolly su Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Il valore predefinito WILDCARD su Sports / Rugby non impedisce la sottoscrizione di caratteri jolly su Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal riceve una pubblicazione perché la sottoscrizione non ha un carattere jolly.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds riceverà una pubblicazione in ogni caso.

Tabella 5 a pagina 84 mostra i risultati dell'aggiunta delle due nuove sottoscrizioni su QMB e della pubblicazione su QMA. Si ricordi che QMB non ha ricevuto alcuna pubblicazione senza queste due nuove sottoscrizioni. Come previsto, le due nuove sottoscrizioni ricevono le pubblicazioni, poiché Sports/FootBall e Sports/Rugby sono entrambi argomenti cluster. QMB ha inoltrato le sottoscrizioni proxy per Sports/Football/Arsenal e Sports/Rugby/Leeds a QMA, che ha quindi inviato le pubblicazioni a QMB.

Il risultato imprevisto è che le due sottoscrizioni Sports/# e Sports/#/Leeds che in precedenza non avevano ricevuto alcuna pubblicazione, ora ricevono le pubblicazioni. Il motivo è che le pubblicazioni Sports/Football/Arsenal e Sports/Rugby/Leeds inoltrate a QMB per le altre sottoscrizioni sono ora disponibili per tutti i sottoscrittori collegati a QMB. Di conseguenza, le sottoscrizioni agli argomenti locali Sports/# e Sports/#/Leeds ricevono la pubblicazione Sports/Rugby/Leeds . Sports/#/Arsenal continua a non ricevere una pubblicazione, poiché Sport / Calcio ha il valore della proprietà WILDCARD impostato su BLOCK.

Tabella 5. Pubblicazioni ricevute su QMB

Sottoscrizione	**Stringa argomento**	**Pubblicazioni ricevute**	**Note**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*Tutte le pubblicazioni nell'albero secondario Calcio bloccate da WILDCARD(BLOCK) su Sports/Football*

Tabella 5. Pubblicazioni ricevute su QMB (Continua)			
Sottoscrizione	**Stringa argomento**	**Pubblicazioni ricevute**	**Note**
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football impedisce la sottoscrizione di caratteri jolly su Arsenal
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*Predefinito WILDCARD su Sports / Rugby non impedisce la sottoscrizione di caratteri jolly su Leeds.*
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal riceve una pubblicazione perché la sottoscrizione non ha un carattere jolly.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds riceverà una pubblicazione in ogni caso.

Nella maggior parte delle applicazioni, non è desiderabile che una sottoscrizione influenzi il comportamento di un'altra sottoscrizione. Un utilizzo importante della proprietà WILDCARD con valore BLOCK consiste nel rendere le sottoscrizioni alla stessa stringa di argomenti contenenti caratteri jolly uniformi. Se la sottoscrizione si trova sullo stesso gestore code del publisher o su un gestore code diverso, i risultati della sottoscrizione sono gli stessi.

Caratteri jolly e stream

Per una nuova applicazione scritta nell'API di pubblicazione / sottoscrizione, l'effetto è che una sottoscrizione a * non riceve alcuna pubblicazione. Per ricevere tutte le pubblicazioni Sport, è necessario sottoscrivere Sports/*o Sports/#e in modo simile per le pubblicazioni Business .

Il comportamento di un'applicazione di pubblicazione / sottoscrizione accodata esistente non cambia quando il Broker di pubblicazione / sottoscrizione viene migrato a una versione successiva di IBM MQ. La proprietà **StreamName** nei comandi **Publish**, **Register Publisher**o **Subscriber** è associata al nome dell'argomento in cui è stato migrato il flusso.

Caratteri jolly e punti di sottoscrizione

Per una nuova applicazione scritta nell'API di pubblicazione / sottoscrizione, l'effetto della migrazione è che una sottoscrizione a * non riceve alcuna pubblicazione. Per ricevere tutte le pubblicazioni Sport, è necessario sottoscrivere Sports/*o Sports/#e in modo simile per le pubblicazioni Business .

Il comportamento di un'applicazione di pubblicazione / sottoscrizione accodata esistente non cambia quando il Broker di pubblicazione / sottoscrizione viene migrato a una versione successiva di IBM MQ. La proprietà **SubPoint** nei comandi **Publish**, **Register Publisher**o **Subscriber** è associata al nome dell'argomento a cui è stata migrata la sottoscrizione.

Esempio: creazione del cluster di pubblicazione / sottoscrizione Sport

I passi che seguono creano un cluster, CL1, con quattro gestori code: due repository completi, CL1A e CL1B, e due repository parziali, QMA e QMB. I repository completi vengono utilizzati per contenere solo le definizioni cluster. QMA è designato come host argomento del cluster. Le sottoscrizioni durevoli sono definite sia su QMA che su QMB.

Nota: L'esempio è codificato per Windows. È necessario ricodificare Crea qmgrs.bat e creare pub.bat per configurare e verificare l'esempio su altre piattaforme.

1. Creare i file script.

a. Crea topics.tst
b. Crea wildsubs.tst
c. Crea fullsubs.tst
d. Crea qmgrs.bat
e. create pub.bat

2. Eseguire Create qmgrs.bat per creare la configurazione.

```
qmgrs
```

Creare gli argomenti in Figura 23 a pagina 82. Lo script nella figura 5 crea gli argomenti del cluster `Sports/Football` e `Sports/Rugby`.

Nota: L'opzione REPLACE non sostituisce le proprietà TOPICSTR di un argomento. TOPICSTR è una proprietà che viene utilmente variata nell'esempio per testare diverse strutture ad albero degli argomenti. Per modificare gli argomenti, eliminare prima l'argomento.

```
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. Eliminare e creare argomenti: topics.tst

Nota: Eliminare gli argomenti, poiché REPLACE non sostituisce le stringhe di argomenti.

Creare sottoscrizioni con caratteri jolly. I caratteri jolly corrispondenti agli argomenti con gli oggetti argomento in Figura 23 a pagina 82. Creare una coda per ogni sottoscrizione. Le code vengono cancellate e le sottoscrizioni vengono eliminate quando lo script viene eseguito o rieseguito.

Nota: L'opzione REPLACE non sostituisce le proprietà TOPICOBJ o TOPICSTR di una sottoscrizione. TOPICOBJ o TOPICSTR sono le proprietà che sono utilmente variate nell'esempio per testare sottoscrizioni differenti. Per modificarli, eliminare prima la sottoscrizione.

```
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. Creare sottoscrizioni jolly: wildsubs.tst

Creare sottoscrizioni che facciano riferimento agli oggetti argomento del cluster.

Nota:

Il delimitatore, /, viene automaticamente inserito tra la stringa argomento a cui fa riferimento TOPICOBJe la stringa argomento definita da TOPICSTR.

La definizione DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) crea la stessa sottoscrizione. TOPICOBJ viene utilizzato come un modo rapido per fare riferimento alla stringa di argomenti già definita. La sottoscrizione, una volta creata, non fa più riferimento all'oggetto argomento.

```
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. Eliminare e creare sottoscrizioni: fullsubs.tst

Creare un cluster con due repository. Creare due repository parziali per la pubblicazione e la sottoscrizione. Eseguire nuovamente lo script per eliminare tutto e ricominciare. Lo script crea anche la gerarchia degli argomenti e le sottoscrizioni dei caratteri jolly iniziali.

Nota:

Su altre piattaforme, scrivere uno script simile oppure immettere tutti i comandi. L'uso di uno script rende veloce l'eliminazione di tutto e ricomincia con una configurazione identica.

```
@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. Creare gestori code: qmgrs.bat

Aggiorna la configurazione aggiungendo le sottoscrizioni agli argomenti del 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. Aggiorna sottoscrizioni: upsubs.bat

Eseguire pub.bat, con un gestore code come parametro, per pubblicare i messaggi contenenti la stringa di argomenti di pubblicazione. Pub.bat utilizza il programma di esempio **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. Pubblica: pub.bat

Stream e argomenti

La pubblicazione / sottoscrizione accodata ha il concetto di un flusso di pubblicazione che non esiste nel modello di pubblicazione / sottoscrizione integrato. Nella pubblicazione / sottoscrizione accodata, i flussi forniscono un metodo per separare il flusso di informazioni per argomenti differenti. Un flusso viene implementato come un argomento di livello superiore che può essere associato a un diverso identificativo dell'argomento dal punto di vista amministrativo.

Il flusso predefinito SYSTEM.BROKER.DEFAULT.STREAM viene impostato automaticamente per tutti i broker e i gestori code su una rete e non è richiesta alcuna configurazione aggiuntiva per utilizzare il flusso predefinito. Considerare il flusso predefinito come uno spazio argomento predefinito senza nome. Gli argomenti pubblicati nel flusso predefinito sono disponibili immediatamente per tutti i gestori code connessi, con la pubblicazione / sottoscrizione in coda abilitata. I flussi denominati sono spazi argomento separati e denominati. Il flusso denominato deve essere definito su ciascun broker in cui viene utilizzato.

Se i publisher e i sottoscrittori si trovano su gestori code differenti, dopo che i broker sono connessi nella stessa gerarchia broker, non è richiesta alcuna ulteriore configurazione per le pubblicazioni e le sottoscrizioni per il flusso tra di loro. La stessa interoperabilità funziona anche al contrario.

Flussi denominati

Un solution designer, che utilizza il modello di programmazione di pubblicazione / sottoscrizione accodata, potrebbe decidere di inserire tutte le pubblicazioni sportive in un flusso denominato Sport. Affinché il flusso sia disponibile per un gestore code in esecuzione su IBM MQ con la pubblicazione / sottoscrizione in coda abilitata, il flusso deve essere aggiunto manualmente.

Le applicazioni di pubblicazione / sottoscrizione accodate che eseguono la sottoscrizione a Soccer/Results sul flusso Sport funzionano senza modifiche. Le applicazioni di pubblicazione / sottoscrizione integrate che sottoscrivono l'argomento Sport utilizzando MQSUBe fornendo la stringa di argomenti Soccer/Results ricevono anche le stesse pubblicazioni.

L'attività di aggiunta di un flusso è descritta nell'argomento Aggiunta di un flusso. Potrebbe essere necessario aggiungere gli stream manualmente per due motivi.

1. Si continua a sviluppare le applicazioni di pubblicazione / sottoscrizione accodate in esecuzione su gestori code di versione successiva, piuttosto che migrare le applicazioni all'interfaccia MQI di pubblicazione / sottoscrizione integrata.
2. L'associazione predefinita dei flussi agli argomenti porta a una "collisione" nello spazio argomento e le pubblicazioni su un flusso hanno la stessa stringa argomento delle pubblicazioni provenienti da altrove.

Autorizzazioni

Per impostazione predefinita, nella root della struttura ad albero degli argomenti sono presenti più oggetti argomento: SYSTEM.BASE.TOPIC, SYSTEM.BROKER.DEFAULT.STREAMe SYSTEM.BROKER.DEFAULT.SUBPOINT. Le autorità (ad esempio, per la pubblicazione o la sottoscrizione) sono determinate dalle autorità sul SYSTEM.BASE.TOPIC ; qualsiasi autorizzazione su SYSTEM.BROKER.DEFAULT.STREAM o SYSTEM.BROKER.DEFAULT.SUBPOINT viene ignorata. Se SYSTEM.BROKER.DEFAULT.STREAM o SYSTEM.BROKER.DEFAULT.SUBPOINT vengono eliminati e ricreati con una stringa di argomento non vuota, le autorizzazioni definite su tali oggetti vengono utilizzate allo stesso modo di un normale oggetto argomento.

Associazione tra stream e argomenti

Un flusso di pubblicazione / sottoscrizione accodato viene imitato in IBM MQ creando una coda e assegnandogli lo stesso nome del flusso. A volte la coda viene chiamata coda di flusso, perché è così che appare per le applicazioni di pubblicazione / sottoscrizione accodate. La coda viene identificata nel motore di pubblicazione / sottoscrizione aggiungendolo all'elenco nomi speciale denominato SYSTEM.QPUBSUB.QUEUE.NAMELIST. È possibile aggiungere tutti i flussi necessari, aggiungendo ulteriori code speciali all'elenco nomi. Infine, è necessario aggiungere argomenti, con gli stessi nomi dei flussi e le stesse stringhe di argomenti del nome del flusso, in modo da poter pubblicare e sottoscrivere gli argomenti.

Tuttavia, in circostanze eccezionali, è possibile fornire agli argomenti corrispondenti agli stream qualsiasi stringa di argomenti scelta quando si definiscono gli argomenti. Lo scopo della stringa di argomenti è fornire all'argomento un nome univoco nello spazio argomenti. In genere il nome del flusso serve perfettamente a questo scopo. A volte, un nome di flusso e un nome di argomento esistente sono in conflitto. Per risolvere il problema, scegliere un'altra stringa di argomenti per l'argomento associato al flusso. Scegliere una stringa di argomenti, assicurandosi che sia univoca.

La stringa dell'argomento definita nella definizione dell'argomento viene preceduta nel modo normale dalla stringa dell'argomento fornita dai publisher e dai sottoscrittori utilizzando le chiamate MQI MQOPEN o MQSUB . Le applicazioni che fanno riferimento ad argomenti che utilizzano oggetti argomento non sono influenzate dalla scelta della stringa argomento prefisso, motivo per cui è possibile scegliere qualsiasi stringa argomento che mantenga le pubblicazioni univoche nello spazio argomento.

La rimappatura di flussi differenti su argomenti differenti si basa sui prefissi utilizzati per le stringhe di argomenti che sono univoci, per separare completamente una serie di argomenti da un'altra. È necessario definire una convenzione di denominazione dell'argomento universale che sia rigidamente rispettata per il funzionamento dell'associazione.

In IBM MQ, si utilizza il meccanismo di prefissaggio per riassociare una stringa di argomenti ad un'altra posizione nello spazio argomenti.

Nota: Quando si elimina uno stream, eliminare prima tutte le sottoscrizioni nello stream. Questa azione è più importante se una delle sottoscrizioni proviene da altri broker nella gerarchia del broker.

Argomenti e punti di sottoscrizione

I punti di sottoscrizione denominati sono emulati da argomenti e oggetti argomento.

Per aggiungere i punti di sottoscrizione manualmente, consultare Aggiunta di un punto di sottoscrizione.

Punti di sottoscrizione in IBM MQ

IBM MQ associa i punti di sottoscrizione a diversi spazi argomento all'interno della struttura ad albero degli argomenti IBM MQ . Gli argomenti nei messaggi di comandi senza un punto di sottoscrizione vengono associati non modificati alla root della struttura ad albero degli argomenti IBM MQ ed ereditano le proprietà da SYSTEM.BASE.TOPIC.

I messaggi di comandi con un punto di sottoscrizione vengono elaborati utilizzando l'elenco di oggetti argomento in SYSTEM.QPUBSUB.SUBPOINT.NAMELIST. Il nome del punto di sottoscrizione nel messaggio di comando viene confrontato con la stringa di argomento per ciascuno degli oggetti

argomento nell'elenco. Se viene trovata una corrispondenza, il nome del punto di sottoscrizione viene anteposto, come un nodo di argomento, alla stringa di argomento. L'argomento eredita le proprietà dall'oggetto argomento associato trovato in SYSTEM.QPUBSUB.SUBPOINT.NAMELIST.

L'effetto dell'utilizzo dei punti di sottoscrizione è quello di creare uno spazio argomento separato per ciascun punto di sottoscrizione. Lo spazio argomento è instradato in un argomento che ha lo stesso nome del punto di sottoscrizione. Gli argomenti in ogni spazio argomento ereditano le proprietà dall'oggetto argomento con lo stesso nome del punto di sottoscrizione.

Tutte le proprietà non impostate nell'oggetto argomento corrispondente vengono ereditate, nel modo normale, da SYSTEM.BASE.TOPIC.

Le applicazioni di pubblicazione / sottoscrizione accodate esistenti, utilizzando le intestazioni del messaggio MQRFH2 , continuano a funzionare impostando la proprietà **SubPoint** nei messaggi di comando Publish o Register subscriber . Il punto di sottoscrizione viene combinato con la stringa di argomenti nel messaggio di comando e l'argomento risultante viene elaborato come qualsiasi altro.

Le applicazioni IBM MQ non sono interessate dai punti di sottoscrizione. Se un'applicazione utilizza un argomento che eredita le informazioni da uno degli oggetti argomento corrispondenti, tale applicazione interagisce con un'applicazione in coda utilizzando il punto di sottoscrizione corrispondente.

Esempio

Un WebSphere Message Broker esistente (ora noto come IBM Integration Bus) l'applicazione di pubblicazione / sottoscrizione migrata in IBM MQ ha creato due oggetti argomento, GBP e USD, con le corrispondenti stringhe argomento 'GBP' e 'USD'.

I publisher esistenti nell'argomento NYSE/IBM/SPOT, migrati per essere eseguiti su IBM MQ, che utilizzano il punto di sottoscrizione USD creano pubblicazioni sull'argomento USD/NYSE/IBM/SPOT. Analogamente, i sottoscrittori esistenti a NYSE/IBM/SPOT, utilizzando il punto di sottoscrizione USD creano sottoscrizioni a USD/NYSE/IBM/SPOT.

sottoscrivere il prezzo spot in dollari in un programma di pubblicazione / sottoscrizione IBM MQ chiamando MQSUB. Creare una sottoscrizione utilizzando l' USD oggetto argomento e la stringa argomento 'NYSE/IBM/SPOT', come illustrato nel frammento di codice ' 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. Impostare l'attributo CLUSTERDEGLI oggetti argomento USD e GBP sull'host argomento cluster.
2. Eliminare tutte le copie degli oggetti argomento USD e GBP su altri gestori code nel cluster.
3. Assicurarsi che USD e GBP siano definiti in SYSTEM.QPUBSUB.SUBPOINT.NAMELIST su ogni gestore code nel cluster.

Esempio di configurazione di pubblicazione / sottoscrizione di un singolo gestore code

Figura 31 a pagina 91 illustra una singola configurazione di pubblicazione / sottoscrizione del gestore code di base. L'esempio mostra la configurazione per un servizio di notizie, dove le informazioni sono disponibili dai publisher su diversi argomenti:

- L'editore 1 pubblica informazioni sui risultati sportivi utilizzando un argomento di Sport
- Publisher 2 sta pubblicando informazioni sui prezzi delle azioni utilizzando un argomento di Stock
- L'editore 3 sta pubblicando informazioni sulle recensioni dei film utilizzando un argomento di Films e sugli elenchi televisivi utilizzando un argomento di TV

Tre sottoscrittori hanno registrato un interesse per argomenti differenti, quindi il gestore code invia loro le informazioni a cui sono interessati:

- L'abbonato 1 riceve i risultati sportivi e le quotazioni azionarie
- L'abbonato 2 riceve le recensioni del film
- L'abbonato 3 riceve i risultati sportivi

Nessuno degli abbonati ha registrato un interesse per gli elenchi televisivi, quindi questi non sono distribuiti.

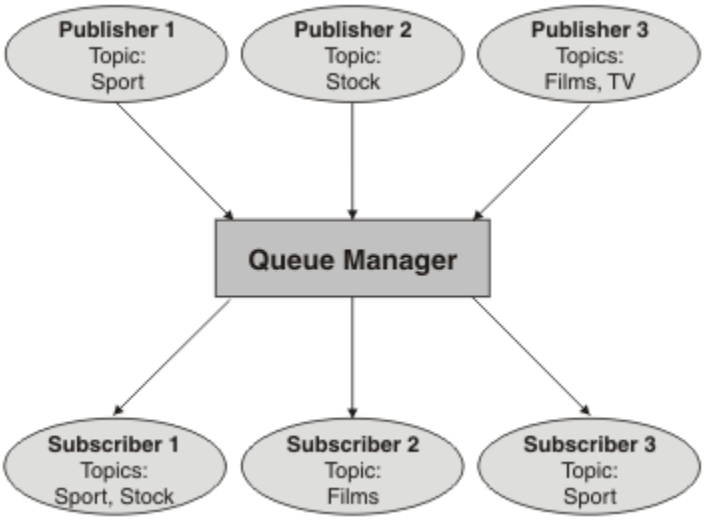


Figura 31. Esempio di pubblicazione / sottoscrizione di un singolo gestore code

Reti di pubblicazione / sottoscrizione distribuite

Ogni gestore code corrisponde ai messaggi pubblicati in un argomento con le sottoscrizioni create localmente che hanno sottoscritto tale argomento. È possibile configurare una rete di gestori code in modo che i messaggi pubblicati da un'applicazione connessa ad un gestore code vengano consegnati alle sottoscrizioni corrispondenti create su altri gestori code nella rete. Ciò richiede una configurazione aggiuntiva su canali semplici tra gestori code.

Una configurazione di pubblicazione / sottoscrizione distribuita è una serie di gestori code connessi tra loro. I gestori code possono trovarsi tutti sullo stesso sistema fisico oppure possono essere distribuiti su diversi sistemi fisici. Quando si collegano i gestori code, i sottoscrittori possono sottoscrivere un gestore code e ricevere i messaggi inizialmente pubblicati su un altro gestore code. Per illustrare questo aspetto, la seguente figura aggiunge un secondo gestore code alla configurazione descritta in “Esempio di configurazione di pubblicazione / sottoscrizione di un singolo gestore code” a pagina 90.

- Il gestore code 2 viene utilizzato dall'editore 4 per pubblicare le informazioni sulle previsioni meteo, utilizzando un argomento di Meteo, e le informazioni sulle condizioni del traffico sulle strade principali, utilizzando un argomento di Traffico.
- Anche il sottoscrittore (subscriber) 4 utilizza questo gestore code e sottoscrive le informazioni sulle condizioni del traffico utilizzando l'argomento Traffico.
- Il sottoscrittore 3 sottoscrive anche le informazioni sulle condizioni meteo, anche se utilizza un gestore code diverso dal publisher. Ciò è possibile perché i gestori code sono collegati tra loro.

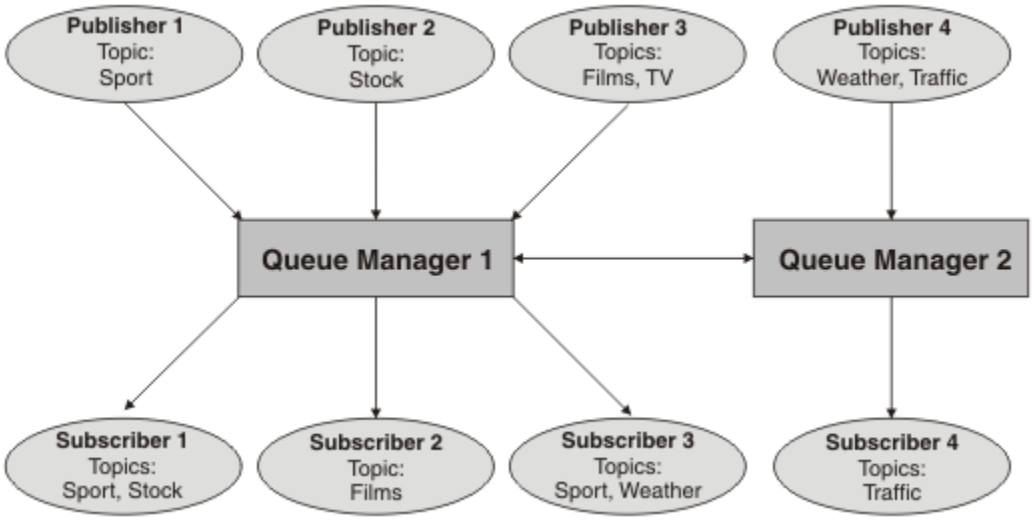


Figura 32. Esempio di pubblicazione / sottoscrizione con due gestori code

È possibile connettere manualmente i gestori code in una gerarchia parent e child oppure è possibile creare un cluster di pubblicazione / sottoscrizione e lasciare che IBM MQ definisca la maggior parte dei dettagli di connessione. È inoltre possibile utilizzare entrambe le topologie in combinazione, ad esempio unendo più cluster in una gerarchia.

Panoramica dei cluster di pubblicazione / sottoscrizione

Un cluster di pubblicazione / sottoscrizione è un cluster standard con uno o più oggetti argomento aggiunti al cluster. Quando si definisce un oggetto argomento di gestione su qualsiasi gestore code in un cluster e si crea un cluster di tale oggetto argomento specificando un nome cluster, i publisher e i sottoscrittori dell'argomento possono connettersi a qualsiasi gestore code nel cluster e i messaggi pubblicati vengono instradati ai sottoscrittori sui canali cluster tra i gestori code.

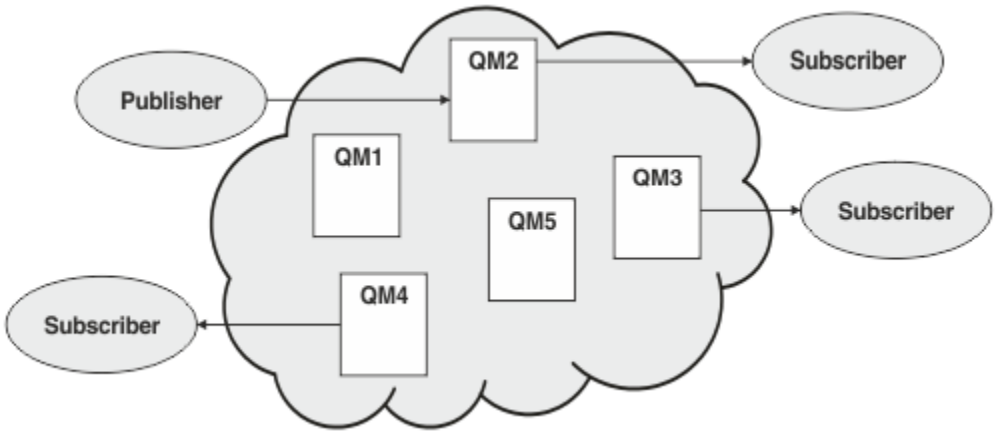


Figura 33. cluster di pubblicazione/sottoscrizione

Esistono due modi per configurare il modo in cui i messaggi di pubblicazione / sottoscrizione vengono instradati in un cluster:

- instradamento diretto
- instradamento host argomento

Quando si configura un argomento con cluster instradato direttamente, i messaggi pubblicati su un gestore code vengono inviati direttamente da tale gestore code a ogni sottoscrizione su qualsiasi altro gestore code nel cluster. Ciò può fornire il percorso più diretto per le pubblicazioni, ma fa sì che

tutti i gestori code in un cluster siano a conoscenza di tutti gli altri gestori code, ognuno dei quali potenzialmente dispone di canali cluster stabiliti tra loro.

Quando si utilizza l'instradamento host argomento, i messaggi pubblicati su un gestore code vengono inviati da lì a un gestore code su cui è presente una definizione dell'oggetto argomento gestito. Tale *gestore code dell'host argomento* instrada il messaggio a ogni sottoscrizione presente su qualsiasi altro gestore code nel cluster. Se i publisher o i sottoscrittori non si trovano sui gestori code dell'host argomento, ciò comporta un instradamento più lungo per le pubblicazioni. Tuttavia, il vantaggio consiste nel fatto che solo i gestori code dell'host argomento vengono a conoscenza di tutti gli altri gestori code nel cluster e potenzialmente dispongono di canali cluster stabiliti con essi.

Per ulteriori informazioni, consultare “Cluster di pubblicazione / sottoscrizione” a pagina 94.

Panoramica delle gerarchie di pubblicazione / sottoscrizione

Una gerarchia di pubblicazione / sottoscrizione è una serie di gestori code connessi da canali in una struttura gerarchica. Ogni gestore code identifica il proprio gestore code *principale* , come descritto in Connessione di un gestore code a una gerarchia di pubblicazione / sottoscrizione.

I publisher e i sottoscrittori di un argomento possono connettersi a qualsiasi gestore code nella gerarchia e il flusso di messaggi tra di essi utilizzando la connettività gerarchica del gestore code.

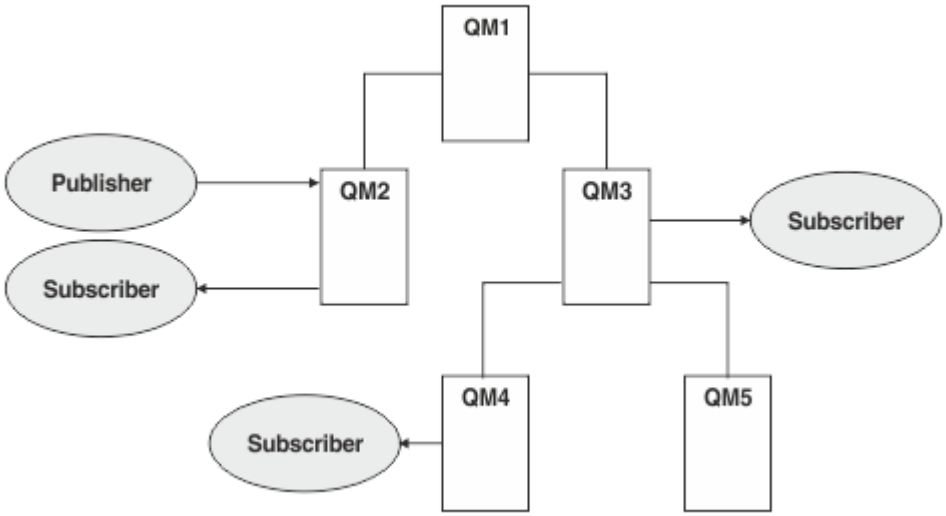


Figura 34. Gerarchia di pubblicazione / sottoscrizione

Nella figura precedente, le pubblicazioni fornite agli utenti su QM3 e QM4 sono state instradate da QM2 a QM1 e quindi su QM3 e infine su QM4.

Le gerarchie forniscono un controllo diretto sulle relazioni tra ogni gestore code nella gerarchia. Ciò consente un controllo dettagliato sull'instradamento dei messaggi dai publisher ai sottoscrittori ed è particolarmente utile quando si esegue l'instradamento tra reti di gestori code con connettività limitata. È necessario considerare attentamente la disponibilità e la funzionalità di ogni gestore code attraverso il quale un messaggio viene instradato dal publisher ai sottoscrittori.

Per ulteriori informazioni, consultare “Gerarchie di pubblicazione / sottoscrizione” a pagina 97.

Distribuzione della pubblicazione tra gestori code

Oltre alle scelte di instradamento, esistono due approcci per distribuire le pubblicazioni su una rete di gestori code:

- Inviare solo le pubblicazioni da un gestore code ai gestori code che attualmente ospitano una sottoscrizione per tale pubblicazione.
- Inviare ogni pubblicazione a tutti i gestori code e lasciare che corrispondano alle relative sottoscrizioni.

Il primo risultato è che i messaggi di pubblicazione vengono inviati solo dove necessario, ma richiede un livello di conoscenza della sottoscrizione da condividere tra i gestori code. Quest' ultima non richiede la condivisione della conoscenza della sottoscrizione, ma può comportare l'invio di messaggi di pubblicazione non necessari tra i gestori code.

Per impostazione predefinita, IBM MQ utilizza il metodo precedente, in cui le pubblicazioni sono inviate solo ai gestori code che dispongono di sottoscrizioni. La conoscenza della sottoscrizione viene propagata tra i gestori code sotto forma di *sottoscrizioni proxy*. Esso dipende dalla distribuzione e dalla durata delle sottoscrizioni e dalla frequenza delle pubblicazioni, per cui è il più efficiente da utilizzare in una topologia di pubblicazione / sottoscrizione distribuita. Vedere Prestazioni della sottoscrizione nelle reti di pubblicazione / sottoscrizione.

Concetti correlati

“Strutture ad albero degli argomenti” a pagina 77
Ciascun argomento che viene definito è un elemento, o nodo, della struttura ad albero degli argomenti. La struttura ad albero degli argomenti può essere vuota per iniziare o contenere argomenti che sono stati precedentemente definiti utilizzando i comandi MQSC o PCF. È possibile definire un nuovo argomento utilizzando i comandi di creazione degli argomenti o specificando l'argomento per la prima volta in una pubblicazione o in una sottoscrizione.

Scenari di gerarchia di pubblicazione / sottoscrizione

Attività correlate

Progettazione di cluster di pubblicazione / sottoscrizione

Cluster di pubblicazione / sottoscrizione

Un cluster di pubblicazione / sottoscrizione è un cluster standard di gestori code interconnessi, su cui le pubblicazioni vengono spostate automaticamente dalle applicazioni di pubblicazione alle sottoscrizioni che esistono su uno qualsiasi dei gestori code nel cluster. Esistono due opzioni per l'instradamento delle pubblicazioni attraverso un cluster di pubblicazione/sottoscrizione: *instradamento diretto* e *instradamento all'host argomento*. L'instradamento scelto dipende dalla dimensione e dai modelli di attività previsti per il cluster.

Un cluster utilizzato per la messaggistica di pubblicazione / sottoscrizione non è diverso da un cluster IBM MQ standard. Pertanto, i gestori code all'interno del cluster di pubblicazione / sottoscrizione possono esistere su computer fisicamente separati e ogni coppia di gestori code viene connessa automaticamente dai canali cluster quando necessario. Per ulteriori informazioni, consultare Cluster.

Per configurare un cluster standard di gestori code per la messaggistica di pubblicazione/sottoscrizione, definire uno o più oggetti argomento amministrati su un gestore code nel cluster. Per rendere l'argomento un argomento cluster, configurare la proprietà **CLUSTER** con il nome del cluster. Quando si esegue questa operazione, qualsiasi argomento utilizzato da un publisher o sottoscrittore in quel punto o al di sotto della struttura ad albero degli argomenti viene condiviso tra tutti i gestori code nel cluster e i messaggi pubblicati in un ramo cluster della struttura ad albero degli argomenti vengono instradati automaticamente alle sottoscrizioni su altri gestori code nel cluster.

Viene inviata solo una copia di ciascun messaggio tra il gestore code del publisher e ognuno degli altri gestori code, indipendentemente dal numero di sottoscrittori per il messaggio sul gestore code di destinazione. All'arrivo in un gestore code con una o più sottoscrizioni, il messaggio viene duplicato in tutte le sottoscrizioni.

Qualsiasi gestore code che si unisce al cluster viene automaticamente a conoscenza degli argomenti del cluster e i publisher e i sottoscrittori su tale gestore code partecipano automaticamente al cluster.

L'attività di pubblicazione / sottoscrizione non in cluster può essere eseguita anche in un cluster di pubblicazione / sottoscrizione, utilizzando le stringhe di argomenti che non rientrano in un oggetto argomento in cluster.

Esistono due opzioni per l'instradamento delle pubblicazioni attraverso un cluster di pubblicazione/sottoscrizione: *instradamento diretto* e *instradamento all'host argomento*. Per scegliere il routing dei messaggi da utilizzare all'interno del cluster, impostare la proprietà **CLROUTE** sull'oggetto argomento amministrato su uno dei seguenti valori:

- **DIRECT**
- **TOPICHOST**

Per impostazione predefinita, l'instradamento argomento è **DIRECT**. Quando si configura un argomento di cluster con instradamento diretto su un gestore code, tutti i gestori code presenti nel cluster sono a conoscenza di tutti gli altri gestori code del cluster. Quando si effettuano operazioni di pubblicazione e sottoscrizione, ogni gestore code può collegarsi direttamente ad ogni altro gestore code nel cluster.

Da IBM MQ 8.0, è invece possibile configurare l'instradamento argomento come **TOPICHOST**. Quando si utilizza l'instradamento all'host argomento, tutti i gestori code presenti nel cluster sono a conoscenza dei gestori code del cluster che ospitano le definizioni dell'argomento instradato (ossia, i gestori code in cui è stato definito l'oggetto dell'argomento). Quando si effettuano operazioni di pubblicazione e sottoscrizione, i gestori code del cluster si connettono soltanto a questi gestori code dell'host argomento e non direttamente l'uno all'altro. I gestori code dell'host argomento sono responsabili dell'instradamento delle pubblicazioni dai gestori code su cui vengono pubblicate le pubblicazioni ai gestori code con le sottoscrizioni corrispondenti.

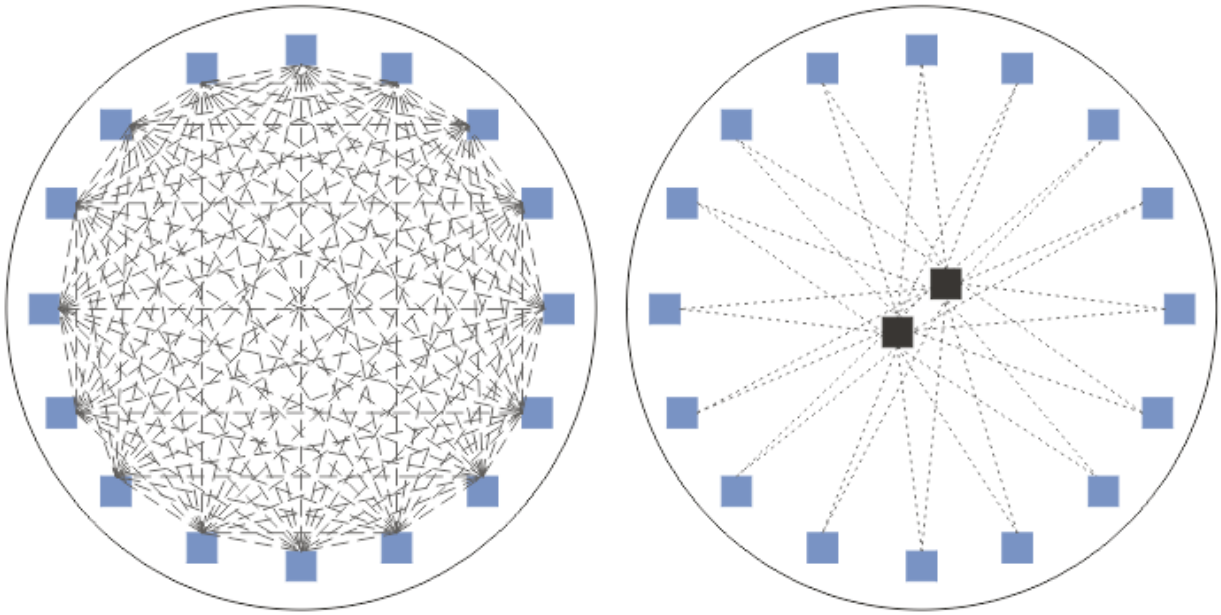


Figura 35. Instradamento diretto e instradamento host argomento

Una panoramica dell'instradamento diretto

Quando un oggetto argomento gestito è configurato per l'instradamento diretto, l'oggetto argomento deve essere definito solo su uno dei gestori code nel cluster per consentire a tutti i gestori code di conoscerlo. La scelta del gestore code su cui è definito il topic non influisce sul funzionamento della messaggistica di pubblicazione / sottoscrizione per il topic.

Ogni messaggio fluisce direttamente dal gestore code del publisher a ciascuna sottoscrizione sugli altri gestori code nel cluster, non passando attraverso alcun gestore code intermedio.

Per impostazione predefinita, i messaggi vengono inviati solo ad altri gestori code nel cluster che ospitano una o più sottoscrizioni.

- Ciò si basa su ciascun gestore code che informa direttamente tutti gli altri gestori code nel cluster di tutti gli argomenti che attualmente hanno una o più sottoscrizioni. Ciò comporta che tutti i gestori code nel cluster siano a conoscenza di tutti gli argomenti sottoscritti e di tutti i gestori code che ospitano una sottoscrizione che stabilisce un canale per ogni altro gestore code. Ciò è indipendente dal fatto che ciascun gestore code abbia o meno un publisher.
- La conoscenza di ogni singolo argomento sottoscritto su tutti i gestori code può essere rimossa passando a un modello di invio di tutte le pubblicazioni a tutti i gestori code nel cluster,

indipendentemente dal fatto che dispongano o meno di sottoscrizioni. Ciò riduce il traffico di conoscenza delle sottoscrizioni, ma è probabile che aumenti il traffico di pubblicazione e il numero di canali stabilito da ogni gestore code. Vedere Prestazioni della sottoscrizione nelle reti di pubblicazione / sottoscrizione.

I flussi di messaggi di pubblicazione / sottoscrizione che utilizzano argomenti con cluster instradati direttamente possono estendersi a più cluster di pubblicazione / sottoscrizione aggiungendo un gestore code da ciascun cluster in una gerarchia di pubblicazione / sottoscrizione. Consultare Combinazione di spazi argomento di più cluster.

Per un'esplorazione più dettagliata dell'instradamento diretto, consultare Instradamento diretto nei cluster di pubblicazione / sottoscrizione.

Una panoramica dell'instradamento dell'host argomento

Quando un oggetto argomento amministrato è configurato per l'instradamento dell'host argomento, le pubblicazioni da un gestore code nel cluster vengono instradate attraverso un gestore code in cui è configurato l'oggetto argomento (un "host argomento") e da lì in poi verso i gestori code in cui esistono le sottoscrizioni.

- Ciò si basa su ciascun gestore code che informa tutti gli host argomento di ogni argomento che attualmente dispone di una o più sottoscrizioni. Qualsiasi gestore code che ospita una sottoscrizione stabilisce un canale per ogni host argomento per l'argomento a cui è correlata la sottoscrizione.
- I gestori code che ospitano non argomenti non sono a conoscenza di altri gestori code che ospitano non argomenti nel cluster per scopi di pubblicazione / sottoscrizione e i canali non sono stabiliti tra loro per tale scopo.
- Se l'applicazione di pubblicazione è connessa a un gestore code che ospita l'argomento, i messaggi pubblicati vengono instradati direttamente ai gestori code in cui sono state create le sottoscrizioni corrispondenti, senza richiedere un ulteriore 'hop'. Allo stesso modo, se le sottoscrizioni corrispondenti vengono create sull'unico gestore code che ospita l'argomento, i messaggi pubblicati in tale argomento vengono instradati direttamente a tale gestore code, senza richiedere un ulteriore hop.
- Le sottoscrizioni sullo stesso gestore code del publisher vengono soddisfatte senza prima instradare le pubblicazioni agli host dell'oggetto argomento.

Come per le code cluster, più gestori code possono configurare lo stesso oggetto argomento di gestione. Ciò fornisce una maggiore disponibilità di instradamento dei messaggi e scalabilità orizzontale attraverso il bilanciamento del carico di lavoro. Per gli oggetti argomento instradati dell'host argomento, quando più gestori code configurano lo stesso argomento denominato per lo stesso ramo della struttura ad albero degli argomenti, ogni host argomento viene reso consapevole degli argomenti sottoscritti da ogni gestore code che ospita una sottoscrizione.

- Quando un messaggio viene pubblicato, viene inviato a un gestore code dell'host argomento per essere inoltrato alla sottoscrizione che ospita i gestori code. La scelta del gestore code dell'host argomento segue le stesse regole di bilanciamento del carico di lavoro predefinite delle code point-to-point con cluster.
- Se uno o più gestori code dell'host argomento non possono essere contattati da un gestore code di pubblicazione, i messaggi vengono instradati ai restanti gestori code dell'host argomento disponibili.

Ogni pubblicazione a un argomento in un ramo instradato della struttura ad albero degli argomenti viene inoltrata a uno degli host degli argomenti, anche se non vi sono sottoscrizioni a tale argomento in alcun punto del cluster. Per impostazione predefinita, i messaggi vengono inviati da qui solo ad altri gestori code nel cluster che ospitano una o più sottoscrizioni.

- Ciò si basa sul fatto che ciascun gestore code dell'host argomento venga informato di tutte le stringhe argomento sottoscritte su ciascun gestore code nel cluster.
- La conoscenza di ogni singolo argomento sottoscritto può essere rimossa passando a un modello di invio di tutte le pubblicazioni instradate a un host argomento su tutti i gestori code nel cluster, indipendentemente dal fatto che dispongano o meno di sottoscrizioni. Ciò riduce il traffico di conoscenza della sottoscrizione, ma è probabile che aumenti il traffico di pubblicazione e

potenzialmente il numero di canali stabiliti con ciascun gestore code che ospita l'argomento. Vedere Prestazioni della sottoscrizione nelle reti di pubblicazione / sottoscrizione.

I flussi di messaggi di pubblicazione / sottoscrizione utilizzando gli argomenti del cluster instradati dell'host argomento **non possono** estendersi a più cluster di pubblicazione / sottoscrizione tramite l'uso di una gerarchia di pubblicazione / sottoscrizione.

Per un'esplorazione più dettagliata dell'instradamento dell'host argomento, consultare Instradamento dell'host argomento nei cluster di pubblicazione / sottoscrizione.

Gerarchie di pubblicazione / sottoscrizione

Si crea una gerarchia di pubblicazione / sottoscrizione collegando i gestori code utilizzando i canali, quindi definendo una relazione child - parent tra coppie di gestori code. Un messaggio fluisce da un publisher alle sottoscrizioni attraverso le relazioni dirette in una gerarchia. Nota che questo potrebbe significare più "hop" per arrivarci.

Solo una copia del messaggio viene inviata tra una coppia di gestori code, indipendentemente dal numero di sottoscrittori del messaggio sul gestore code di destinazione. All'arrivo in un gestore code con una o più sottoscrizioni, il messaggio viene duplicato in tutte le sottoscrizioni.

Per impostazione predefinita, i messaggi vengono inviati solo ad altri gestori code nella gerarchia che si trovano sull'instradamento a una sottoscrizione su un altro gestore code:

- Ciò si basa sul fatto che ogni gestore code informi ogni relazione diretta di tutti gli argomenti che attualmente hanno una o più sottoscrizioni, su questo gestore code o su una delle sue altre relazioni. Ciò fa sì che tutti i gestori code nella gerarchia siano a conoscenza di tutti gli argomenti sottoscritti.
- Questo comportamento può essere modificato per inviare sempre le pubblicazioni a tutti i gestori code della gerarchia, indipendentemente dalle sottoscrizioni esistenti. Ciò elimina la necessità di propagare le informazioni di sottoscrizione nella gerarchia, ma può aumentare il traffico di pubblicazione.

Quando si crea un cluster, è necessario prestare attenzione a non creare un loop che causi il ciclo dei messaggi all'interno della rete. Non è possibile creare tali loop in una gerarchia.

Ogni gestore code deve avere un nome gestore code univoco.

I flussi di messaggi di pubblicazione / sottoscrizione possono estendersi a più cluster di pubblicazione / sottoscrizione. A tale scopo, aggiungere un gestore code da ciascun cluster in una gerarchia di pubblicazione / sottoscrizione.

Per un'esplorazione più dettagliata, consultare Instradamento nelle gerarchie di pubblicazione / sottoscrizione.

Sottoscrizioni proxy in una rete di pubblicazione / sottoscrizione

Una sottoscrizione proxy è una sottoscrizione effettuata da un gestore code per gli argomenti pubblicati su un altro gestore code. Una sottoscrizione proxy transita tra i gestori code per ogni singola stringa argomento sottoscritta da una sottoscrizione. Non si creano le sottoscrizioni proxy esplicitamente; il gestore code lo fa per conto dell'utente.

È possibile connettere i gestori code in un cluster di pubblicazione / sottoscrizione o in una gerarchia di pubblicazione / sottoscrizione. Le sottoscrizioni proxy passano tra i gestori code connessi. Le sottoscrizioni proxy fanno in modo che le pubblicazioni di un argomento creato da un publisher connesso a un gestore code vengano ricevute dai sottoscrittori di tale argomento connessi ad altri gestori code. Consultare “Reti di pubblicazione / sottoscrizione distribuite” a pagina 91.

Nelle topologie di pubblicazione / sottoscrizione con molte migliaia di sottoscrizioni a singole stringhe di argomenti o dove l'esistenza di tali sottoscrizioni potrebbe essere in rapida modifica, è necessario considerare il sovraccarico della propagazione della sottoscrizione proxy. Oltre all'aggregazione automatica descritta nel resto di questo argomento, è possibile apportare modifiche di configurazione manuali che limitano ulteriormente il flusso delle sottoscrizioni proxy e delle pubblicazioni tra i gestori code connessi e che riducono la latenza dell'attesa della propagazione di una sottoscrizione proxy a tutti i gestori code connessi. Vedere Prestazioni della sottoscrizione nelle reti di pubblicazione / sottoscrizione.

Le sottoscrizioni proxy non contengono selettori utilizzati dalle sottoscrizioni locali e le stringhe di argomenti di sottoscrizione che contengono caratteri jolly potrebbero essere semplificate. Ciò può risultare in pubblicazioni che corrispondono alle sottoscrizioni proxy laddove non lo fanno le sottoscrizioni effettive, con conseguente ulteriore flusso di pubblicazione tra i gestori code. Il gestore code che ospita le sottoscrizioni filtra tali discrepanze in modo che le pubblicazioni aggiuntive non vengano restituite alle sottoscrizioni.

Aggregazione sottoscrizione proxy

Le sottoscrizioni proxy vengono aggregate utilizzando un sistema di eliminazione duplicato. Per una particolare stringa di argomenti risolta, una sottoscrizione proxy viene inviata alla prima sottoscrizione locale o alla sottoscrizione proxy ricevuta. Le sottoscrizioni successive alla stessa stringa di argomenti utilizzano questa sottoscrizione proxy esistente.

La sottoscrizione proxy viene annullata dopo l'annullamento dell'ultima sottoscrizione locale o della sottoscrizione proxy ricevuta.

Aggregazione pubblicazione

Quando è presente più di una sottoscrizione alla stessa stringa di argomenti su un gestore code, solo una singola copia di ogni pubblicazione corrispondente a tale stringa di argomenti viene inviata da altri gestori code nella topologia di pubblicazione / sottoscrizione. All'arrivo del messaggio, il gestore code locale consegna una copia del messaggio a ciascuna sottoscrizione corrispondente.

È possibile che più di una sottoscrizione proxy corrisponda alla stringa argomento di una singola pubblicazione quando le sottoscrizioni proxy contengono caratteri jolly. Se un messaggio viene pubblicato su un gestore code che corrisponde a due o più sottoscrizioni proxy create da un singolo gestore code connesso, solo una copia della pubblicazione viene inoltrata al gestore code remoto per soddisfare le sottoscrizioni proxy multiple.

Concetti correlati

Rilevamento loop in una rete di pubblicazione / sottoscrizione distribuita

Caratteri jolly nelle sottoscrizioni proxy

Le sottoscrizioni possono utilizzare caratteri jolly nelle stringhe di argomenti per la corrispondenza con più stringhe di argomenti nelle pubblicazioni.

Esistono due schemi di caratteri jolly che una sottoscrizione può utilizzare: *basato su argomenti* e *basato su caratteri*. Consultare“Schemi dei caratteri jolly” a pagina 71.

In IBM MQ tutte le sottoscrizioni proxy per le sottoscrizioni jolly vengono convertite per utilizzare i caratteri jolly basati sull'argomento. Se viene trovato un carattere jolly basato sui caratteri, viene sostituito con un carattere `#` , di nuovo al carattere `/`più vicino. Ad esempio, `/aaa/bbb/c*d` viene convertito in `/aaa/bbb/#`. La conversione risulta in gestori code remoti che inviano un numero di pubblicazioni leggermente superiore a quello esplicitamente sottoscritto. Le pubblicazioni aggiuntive vengono filtrate dal gestore code locale, quando consegna le pubblicazioni ai relativi sottoscrittori locali.

Controllo dell'utilizzo dei caratteri jolly con la proprietà WILDCARD

Utilizzare la proprietà MQSC **Topic** WILDCARD o la proprietà Argomento `WildcardOperation` PCF equivalente per controllare la consegna di pubblicazioni alle applicazioni del sottoscrittore che utilizzano nomi stringa di argomenti con caratteri jolly. La proprietà WILDCARD può avere uno dei due valori possibili:

WILDCARD
Il funzionamento delle sottoscrizioni con caratteri jolly rispetto a questo argomento.

PASSTHRU
Le sottoscrizioni effettuate a un argomento con carattere jolly meno specifico della stringa argomento in questo oggetto argomento riceveranno le pubblicazioni relative a questo argomento e a stringhe argomento più specifiche di tale argomento.

BLOCK
Le sottoscrizioni effettuate a un argomento con carattere jolly meno specifico della stringa argomento in questo oggetto argomento non riceveranno le pubblicazioni relative a questo argomento o a stringhe argomento più specifiche di tale argomento.

Il valore di questo attributo è utilizzato quando vengono definite sottoscrizioni. Se si modifica questo attributo, la serie di argomenti trattati dalle sottoscrizioni esistenti non viene interessata dalla modifica. Questo scenario si applica anche se la topologia cambia quando si creano o eliminano oggetti argomento; la serie di argomenti che corrispondono alle sottoscrizioni create in seguito alla modifica dell'attributo WILDCARD viene creata utilizzando la topologia modificata. Se si desidera forzare una rivalutazione della serie corrispondente di argomenti per le sottoscrizioni esistenti, è necessario riavviare il gestore code.

Nell'esempio, “Esempio: creazione del cluster di pubblicazione / sottoscrizione Sport” a pagina 85, è possibile seguire la procedura per creare la struttura ad albero dell'argomento mostrata in Figura 23 a pagina 82.

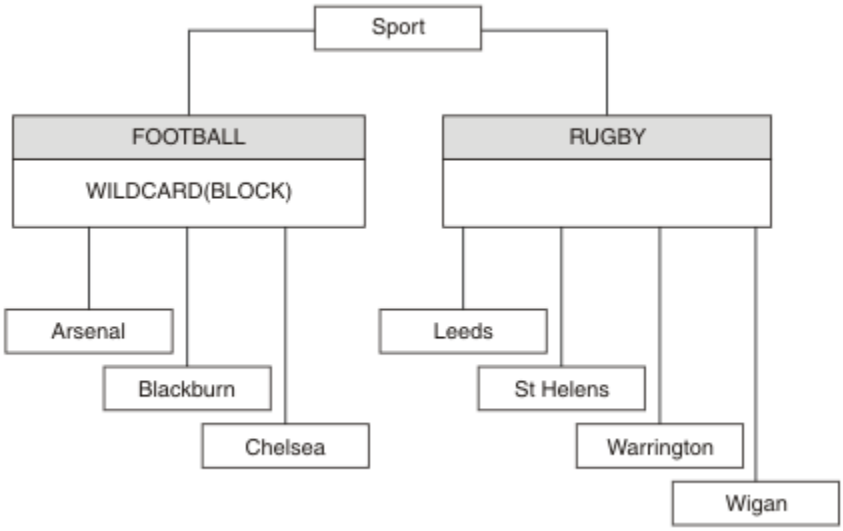


Figura 36. Una struttura di argomenti che utilizza la proprietà WILDCARD , BLOCK

Un sottoscrittore che utilizza la stringa di argomenti con caratteri jolly # riceve tutte le pubblicazioni nell'argomento `Sport` e nella struttura ad albero secondaria `Sport/Rugby` . Il sottoscrittore non riceve alcuna pubblicazione nella struttura secondaria `Sport/Football` , poiché il valore della proprietà `WILDCARD` dell'argomento `Sport/Football` è `BLOCK`.

`PASSTHRU` è l'impostazione predefinita. È possibile impostare il valore della proprietà `WILDCARD` `PASSTHRU` sui nodi nella struttura ad albero `Sport` . Se i nodi non hanno il valore della proprietà `WILDCARD` `BLOCK`, l'impostazione `PASSTHRU` non modifica il funzionamento osservato dai sottoscrittori ai nodi nella albero `Sports` .

Nell'esempio, creare sottoscrizioni per vedere in che modo l'impostazione del carattere jolly influisce sulle pubblicazioni fornite; consultare Figura 27 a pagina 87. Eseguire il comando publish in Figura 30 a pagina 88 per creare alcune pubblicazioni.

```
pub QMA
```

Figura 37. Pubblica in QMA

I risultati sono riportati in Tabella 3 a pagina 82. Si noti come l'impostazione del valore della proprietà `WILDCARD` `BLOCK`impedisca alle sottoscrizioni con caratteri jolly di ricevere pubblicazioni per argomenti nell'ambito del carattere jolly.

Tabella 6. Pubblicazioni ricevute su QMA			
Sottoscrizione	**Stringa argomento**	**Pubblicazioni ricevute**	**Note**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Tutte le pubblicazioni della sottostruttura Football bloccate da WILDCARD(BLOCK) su Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football impedisce la sottoscrizione di caratteri jolly su Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Il valore predefinito WILDCARD su Sports / Rugby non impedisce la sottoscrizione di caratteri jolly su Leeds.

Nota:

Si supponga che una sottoscrizione abbia un carattere jolly che corrisponde a un oggetto argomento con il valore della proprietà WILDCARD BLOCK. Se la sottoscrizione ha anche una stringa di argomenti a destra del carattere jolly corrispondente, la sottoscrizione non riceverà mai una pubblicazione. La serie di pubblicazioni che non sono bloccate sono pubblicazioni per argomenti che sono parent del carattere jolly bloccato. Le pubblicazioni per gli argomenti child dell'argomento con il valore della proprietà BLOCK sono bloccate dal carattere jolly. Pertanto, le stringhe di argomenti di sottoscrizione che includono un argomento alla destra del carattere jolly non ricevono mai alcuna pubblicazione corrispondente.

L'impostazione del valore della proprietà WILDCARD su BLOCK non significa che non è possibile effettuare la sottoscrizione utilizzando una stringa di argomenti che include caratteri jolly. Tale sottoscrizione è normale. La sottoscrizione dispone di un argomento esplicito che corrisponde all'argomento con un oggetto argomento con un valore della proprietà WILDCARD BLOCK. Utilizza i caratteri jolly per gli argomenti che sono parent o child dell'argomento con il valore della proprietà WILDCARD BLOCK. Nell'esempio in Figura 23 a pagina 82, una sottoscrizione come Sports/Football/# può ricevere pubblicazioni.

Caratteri jolly e argomenti cluster

Le definizioni degli argomenti del cluster vengono propagati a ogni gestore code in un cluster. Una sottoscrizione a un argomento cluster in un gestore code in un cluster determina la creazione di sottoscrizioni proxy da parte del gestore code. Una sottoscrizione proxy viene creata su ogni altro gestore code nel cluster. Le sottoscrizioni che utilizzano stringhe di argomenti contenenti caratteri jolly, combinate con argomenti cluster, possono fornire un comportamento difficile da prevedere. Il comportamento viene spiegato nel seguente esempio.

Nel cluster configurato per l'esempio, “Esempio: creazione del cluster di pubblicazione / sottoscrizione Sport” a pagina 85, QMB ha la stessa serie di sottoscrizioni di QMA, ma QMB non ha ricevuto alcuna pubblicazione dopo che il publisher è stato pubblicato in QMA, consultare Figura 24 a pagina 82. Anche se gli argomenti Sports/Football e Sports/Rugby sono argomenti del cluster, le sottoscrizioni definite in fullsubs.tst non fanno riferimento a un argomento del cluster. Nessuna sottoscrizione proxy viene propagata da QMB a QMA. Senza sottoscrizioni proxy, nessuna pubblicazione per QMA viene inoltrata a QMB.

Alcune sottoscrizioni, come Sports/#/Leeds, potrebbero far riferimento a un argomento cluster, in questo caso Sports/Rugby. La sottoscrizione Sports/#/Leeds viene effettivamente risolta nell'oggetto argomento SYSTEM.BASE.TOPIC.

La regola per risolvere l'oggetto argomento a cui fa riferimento una sottoscrizione come, Sports/#/Leeds è la seguente. Troncare la stringa argomento al primo carattere jolly. Eseguire la scansione a

sinistra nella stringa dell'argomento cercando il primo argomento a cui è associato un oggetto argomento di gestione. L'oggetto argomento potrebbe specificare un nome cluster o definire un oggetto argomento locale. Nell'esempio, `Sports/#/Leeds`, la stringa di argomenti dopo il troncamento è `Sports`, che non ha alcun oggetto argomento e quindi `Sports/#/Leeds` eredita da `SYSTEM.BASE.TOPIC`, che è un oggetto argomento locale.

Per vedere come la sottoscrizione agli argomenti del cluster può modificare il funzionamento della propagazione dei caratteri jolly, eseguire lo script batch upsubs.bat. Lo script elimina le code di sottoscrizione e aggiunge le sottoscrizioni dell'argomento del cluster in fullsubs.tst. Eseguire nuovamente puba.bat per creare un batch di pubblicazioni; consultare Figura 24 a pagina 82.

Tabella 4 a pagina 84 mostra il risultato dell'aggiunta di due nuove sottoscrizioni allo stesso gestore code su cui sono state pubblicate le pubblicazioni. Il risultato è quello previsto, le nuove sottoscrizioni ricevono una pubblicazione ciascuna e il numero di pubblicazioni ricevute dalle altre sottoscrizioni è invariato. I risultati imprevisti si verificano sull'altro gestore code cluster; consultare Tabella 5 a pagina 84.

Tabella 7. Pubblicazioni ricevute su QMA

Sottoscrizione	**Stringa argomento**	**Pubblicazioni ricevute**	**Note**

SPORTS	`Sports/#`	`Sports`
		`Sports/Rugby`
		`Sports/Rugby/Leeds`
			Tutte le pubblicazioni della sottostruttura Football bloccate da `WILDCARD(BLOCK)` su `Sports/Football`

SARSENAL	`Sports/#/Arsenal`	-	`WILDCARD(BLOCK)` on `Sports/Football` impedisce la sottoscrizione di caratteri jolly su `Arsenal`
SLEEDS	`Sports/#/Leeds`	`Sports/Rugby/Leeds`	Il valore predefinito `WILDCARD` su Sports / Rugby non impedisce la sottoscrizione di caratteri jolly su `Leeds`.
FARSENAL	`Sports/Football/Arsenal`	`Sports/Football/Arsenal`	`Arsenal` riceve una pubblicazione perché la sottoscrizione non ha un carattere jolly.
FLEEDS	`Sports/Rugby/Leeds`	`Sports/Rugby/Leeds`	`Leeds` riceverà una pubblicazione in ogni caso.

Tabella 5 a pagina 84 mostra i risultati dell'aggiunta delle due nuove sottoscrizioni su QMB e della pubblicazione su QMA. Si ricordi che QMB non ha ricevuto alcuna pubblicazione senza queste due nuove sottoscrizioni. Come previsto, le due nuove sottoscrizioni ricevono le pubblicazioni, poiché `Sports/FootBall` e `Sports/Rugby` sono entrambi argomenti cluster. QMB ha inoltrato le sottoscrizioni proxy per `Sports/Football/Arsenal` e `Sports/Rugby/Leeds` a QMA, che ha quindi inviato le pubblicazioni a QMB.

Il risultato imprevisto è che le due sottoscrizioni `Sports/#` e `Sports/#/Leeds` che in precedenza non avevano ricevuto alcuna pubblicazione, ora ricevono le pubblicazioni. Il motivo è che le pubblicazioni `Sports/Football/Arsenal` e `Sports/Rugby/Leeds` inoltrate a QMB per le altre sottoscrizioni sono ora disponibili per tutti i sottoscrittori collegati a QMB. Di conseguenza, le sottoscrizioni agli argomenti locali `Sports/#` e `Sports/#/Leeds` ricevono la pubblicazione `Sports/Rugby/Leeds` . `Sports/#/Arsenal` continua a non ricevere una pubblicazione, poiché Sport / Calcio ha il valore della proprietà `WILDCARD` impostato su `BLOCK`.

Tabella 8. Pubblicazioni ricevute su QMB			
Sottoscrizione	**Stringa argomento**	**Pubblicazioni ricevute**	**Note**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*Tutte le pubblicazioni nell'albero secondario Calcio bloccate da WILDCARD(BLOCK) su Sports/Football*
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football impedisce la sottoscrizione di caratteri jolly su Arsenal
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*Predefinito WILDCARD su Sports / Rugby non impedisce la sottoscrizione di caratteri jolly su Leeds.*
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal riceve una pubblicazione perché la sottoscrizione non ha un carattere jolly.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds riceverà una pubblicazione in ogni caso.

Nella maggior parte delle applicazioni, non è desiderabile che una sottoscrizione influenzi il comportamento di un'altra sottoscrizione. Un utilizzo importante della proprietà WILDCARD con valore BLOCK consiste nel rendere le sottoscrizioni alla stessa stringa di argomenti contenenti caratteri jolly uniformi. Se la sottoscrizione si trova sullo stesso gestore code del publisher o su un gestore code diverso, i risultati della sottoscrizione sono gli stessi.

Caratteri jolly e stream

Per una nuova applicazione scritta nell'API di pubblicazione / sottoscrizione, l'effetto è che una sottoscrizione a * non riceve alcuna pubblicazione. Per ricevere tutte le pubblicazioni Sport, è necessario sottoscrivere Sports/*o Sports/#e in modo simile per le pubblicazioni Business .

Il comportamento di un'applicazione di pubblicazione / sottoscrizione accodata esistente non cambia quando il Broker di pubblicazione / sottoscrizione viene migrato a una versione successiva di IBM MQ. La proprietà **StreamName** nei comandi **Publish**, **Register Publisher**o **Subscriber** è associata al nome dell'argomento in cui è stato migrato il flusso.

Caratteri jolly e punti di sottoscrizione

Per una nuova applicazione scritta nell'API di pubblicazione / sottoscrizione, l'effetto della migrazione è che una sottoscrizione a * non riceve alcuna pubblicazione. Per ricevere tutte le pubblicazioni Sport, è necessario sottoscrivere Sports/*o Sports/#e in modo simile per le pubblicazioni Business .

Il comportamento di un'applicazione di pubblicazione / sottoscrizione accodata esistente non cambia quando il Broker di pubblicazione / sottoscrizione viene migrato a una versione successiva di IBM MQ. La proprietà **SubPoint** nei comandi **Publish**, **Register Publisher**o **Subscriber** è associata al nome dell'argomento a cui è stata migrata la sottoscrizione.

Esempio: creazione del cluster di pubblicazione / sottoscrizione Sport

I passi che seguono creano un cluster, CL1, con quattro gestori code: due repository completi, CL1A e CL1B, e due repository parziali, QMA e QMB. I repository completi vengono utilizzati per contenere solo le definizioni cluster. QMA è designato come host argomento del cluster. Le sottoscrizioni durevoli sono definite sia su QMA che su QMB.

Nota: L'esempio è codificato per Windows. È necessario ricodificare Crea qmgrs.bat e creare pub.bat per configurare e verificare l'esempio su altre piattaforme.

1. Creare i file script.
   a. Crea topics.tst
   b. Crea wildsubs.tst
   c. Crea fullsubs.tst
   d. Crea qmgrs.bat
   e. create pub.bat
2. Eseguire Create qmgrs.bat per creare la configurazione.

```
qmgrs
```

Creare gli argomenti in Figura 23 a pagina 82. Lo script nella figura 5 crea gli argomenti del cluster `Sports/Football` e `Sports/Rugby`.

Nota: L'opzione REPLACE non sostituisce le proprietà TOPICSTR di un argomento. TOPICSTR è una proprietà che viene utilmente variata nell'esempio per testare diverse strutture ad albero degli argomenti. Per modificare gli argomenti, eliminare prima l'argomento.

```
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. Eliminare e creare argomenti: topics.tst

Nota: Eliminare gli argomenti, poiché REPLACE non sostituisce le stringhe di argomenti.

Creare sottoscrizioni con caratteri jolly. I caratteri jolly corrispondenti agli argomenti con gli oggetti argomento in Figura 23 a pagina 82. Creare una coda per ogni sottoscrizione. Le code vengono cancellate e le sottoscrizioni vengono eliminate quando lo script viene eseguito o rieseguito.

Nota: L'opzione REPLACE non sostituisce le proprietà TOPICOBJ o TOPICSTR di una sottoscrizione. TOPICOBJ o TOPICSTR sono le proprietà che sono utilmente variate nell'esempio per testare sottoscrizioni differenti. Per modificarli, eliminare prima la sottoscrizione.

```
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. Creare sottoscrizioni jolly: wildsubs.tst

Creare sottoscrizioni che facciano riferimento agli oggetti argomento del cluster.

Nota:

Il delimitatore, /, viene automaticamente inserito tra la stringa argomento a cui fa riferimento TOPICOBJe la stringa argomento definita da TOPICSTR.

La definizione DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) crea la stessa sottoscrizione. TOPICOBJ viene utilizzato come un modo rapido per fare riferimento alla stringa di argomenti già definita. La sottoscrizione, una volta creata, non fa più riferimento all'oggetto argomento.

```
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. Eliminare e creare sottoscrizioni: fullsubs.tst

Creare un cluster con due repository. Creare due repository parziali per la pubblicazione e la sottoscrizione. Eseguire nuovamente lo script per eliminare tutto e ricominciare. Lo script crea anche la gerarchia degli argomenti e le sottoscrizioni dei caratteri jolly iniziali.

Nota:

Su altre piattaforme, scrivere uno script simile oppure immettere tutti i comandi. L'uso di uno script rende veloce l'eliminazione di tutto e ricomincia con una configurazione identica.

```
@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. Creare gestori code: qmgrs.bat

Aggiorna la configurazione aggiungendo le sottoscrizioni agli argomenti del 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. Aggiorna sottoscrizioni: upsubs.bat

Eseguire pub.bat, con un gestore code come parametro, per pubblicare i messaggi contenenti la stringa di argomenti di pubblicazione. Pub.bat utilizza il programma di esempio **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. Pubblica: pub.bat

Concetti correlati

Sottoscrizioni jolly e pubblicazioni conservate

Ambito della pubblicazione

Quando si configura un cluster di pubblicazione / sottoscrizione o una gerarchia, l'ambito di una pubblicazione controlla ulteriormente se i gestori code inoltrano una pubblicazione ai gestori code remoti. Utilizzare l'attributo dell'argomento **PUBSCOPE** per gestire l'ambito delle pubblicazioni.

Se una pubblicazione non viene inoltrata ai gestori code remoti, solo i sottoscrittori locali ricevono la pubblicazione.

Quando si utilizza un cluster di pubblicazione / sottoscrizione, l'ambito delle pubblicazioni è principalmente controllato dalla definizione di oggetti argomento in cluster in determinati punti della

struttura ad albero dell'argomento. L'ambito di pubblicazione deve essere impostato per consentire il flusso di pubblicazioni ad altri gestori code nel cluster. È necessario limitare l'ambito di pubblicazione per un argomento con cluster solo quando è necessario un controllo più preciso di argomenti specifici su determinati gestori code.

Quando si utilizza una gerarchia di pubblicazione / sottoscrizione, l'ambito delle pubblicazioni è controllato principalmente da questo attributo in combinazione con l'attributo ambito sottoscrizione .

L'attributo **PUBSCOPE** viene utilizzato per stabilire l'ambito delle pubblicazioni effettuate per un argomento specifico. È possibile impostare l'attributo su uno dei seguenti valori:

QMGR
: La pubblicazione viene consegnata solo ai sottoscrittori locali. Queste pubblicazioni sono denominate *pubblicazioni locali*. Le pubblicazioni locali non vengono inoltrate ai gestori code remoti e quindi non vengono ricevute dai sottoscrittori connessi ai gestori code remoti.

TUTTO
: La pubblicazione viene consegnata ai sottoscrittori locali e ai sottoscrittori connessi ai gestori code remoti in una gerarchia o in un cluster di pubblicazione / sottoscrizione. Queste pubblicazioni sono denominate *pubblicazioni globali*.

Come parent
: Utilizzare l'impostazione **PUBSCOPE** dell'argomento principale nella struttura ad albero degli argomenti.

I publisher possono anche specificare se una pubblicazione è locale o globale utilizzando l'opzione di inserimento del messaggio MQPMO_SCOPE_QMGR . Se viene utilizzata questa opzione, sovrascrive qualsiasi comportamento impostato utilizzando l'attributo argomento **PUBSCOPE** .

Concetti correlati

“Oggetti argomento di gestione” a pagina 78
Utilizzando un oggetto argomento di amministrazione, è possibile assegnare specifici attributi non predefiniti agli argomenti.

Attività correlate

Configurazione delle reti di pubblicazione / sottoscrizione distribuite

Ambito della sottoscrizione

L'ambito di una sottoscrizione controlla se una sottoscrizione su un gestore code riceve le pubblicazioni pubblicate su un altro gestore code in un cluster o gerarchia di pubblicazione / sottoscrizione oppure solo le pubblicazioni dei publisher locali.

La limitazione dell'ambito di sottoscrizione a un gestore code impedisce l'inoltro delle sottoscrizioni proxy ad altri gestori code nella topologia di pubblicazione / sottoscrizione. Ciò riduce il traffico di messaggistica di pubblicazione / sottoscrizione tra gestori code.

Quando si utilizza un cluster di pubblicazione / sottoscrizione, l'ambito delle sottoscrizioni è controllato principalmente dalla definizione di oggetti argomento raggruppati in determinati punti della struttura ad albero degli argomenti. L'ambito della sottoscrizione deve essere impostato per consentire il flusso delle sottoscrizioni proxy ad altri gestori code nel cluster. Si consiglia di limitare l'ambito della sottoscrizione per un argomento cluster solo quando è necessario un controllo più preciso di argomenti specifici su determinati gestori code.

Quando si utilizza una gerarchia di pubblicazione / sottoscrizione, l'ambito delle sottoscrizioni è controllato principalmente da questo attributo in combinazione con l'attributo ambito di pubblicazione .

L'attributo dell'argomento **SUBSCOPE** viene utilizzato per determinare l'ambito delle sottoscrizioni effettuate a un argomento specifico. È possibile impostare l'attributo su uno dei seguenti valori:

QMGR
: Una sottoscrizione riceve solo pubblicazioni locali e le sottoscrizioni proxy non sono propagate ai gestori code remoti.

TUTTO
Una sottoscrizione proxy viene propagata ai gestori code remoti in un cluster o gerarchia di pubblicazione / sottoscrizione e il sottoscrittore riceve le pubblicazioni locali e remote.

Come parent
Utilizzare l'impostazione **SUBSCOPE** dell'argomento principale nella struttura ad albero degli argomenti.

Quando l'ambito della sottoscrizione per un argomento è impostato su TUTTO, direttamente o risolto tramite ASPARENT, le singole sottoscrizioni a tale argomento possono limitare il loro ambito a QMGR specificando MQSO_SCOPE_QMGR quando si crea la sottoscrizione. Una sottoscrizione a un argomento che ha un ambito QMGR non può ampliare l'ambito a ALL.

Concetti correlati

"Oggetti argomento di gestione" a pagina 78
Utilizzando un oggetto argomento di amministrazione, è possibile assegnare specifici attributi non predefiniti agli argomenti.

Attività correlate

Configurazione delle reti di pubblicazione / sottoscrizione distribuite

Spazi argomento

Uno spazio argomenti è una serie di argomenti su cui è possibile effettuare la sottoscrizione e la pubblicazione. Un gestore code in una topologia di pubblicazione / sottoscrizione distribuita ha uno spazio argomenti che potenzialmente include argomenti sottoscritti e pubblicati su gestori code connessi in tale topologia.

Nota: Per una panoramica degli argomenti all'interno di un gestore code, come gli oggetti argomento di gestione, le stringhe argomento e le strutture ad albero argomento, consultare"Argomenti" a pagina 69. Ulteriori riferimenti agli *argomenti* nell'articolo corrente fanno riferimento a *stringhe di argomenti* , se non diversamente specificato.

Gli argomenti vengono creati inizialmente in uno dei seguenti modi:

- amministrativamente, quando si definisce un oggetto argomento o una sottoscrizione durevole.
- dinamicamente, quando un'applicazione crea una pubblicazione o una sottoscrizione dinamicamente a un nuovo argomento.

Gli argomenti vengono propagati ad altri gestori code tramite sottoscrizioni proxy e creando oggetti argomento del cluster di gestione. Le sottoscrizioni proxy determinano l'inoltro delle pubblicazioni dal gestore code a cui è connesso un publisher ai gestori code dei sottoscrittori.

Le sottoscrizioni proxy vengono propagate tra tutti i gestori code connessi tra loro da relazioni padre - figlio in una gerarchia di gestori code. Il risultato è che è possibile sottoscrivere un gestore code a un argomento definito su qualsiasi altro gestore code nella gerarchia. Finché esiste un percorso connesso tra i gestori code, non importa come sono connessi i gestori code.

Le sottoscrizioni proxy vengono propagate anche per le sottoscrizioni agli argomenti cluster in un cluster di pubblicazione / sottoscrizione. Un argomento cluster è un argomento collegato a un oggetto argomento che ha l'attributo **CLUSTER** o eredita l'attributo dal relativo parent. Gli argomenti che non sono argomenti cluster sono noti come argomenti locali e non vengono replicati nel cluster. Nessuna sottoscrizione proxy viene propagata al cluster dalle sottoscrizioni agli argomenti locali.

Per riepilogare, le sottoscrizioni proxy vengono create per i sottoscrittori in due circostanze.

1. Un gestore code è un membro di una gerarchia e una sottoscrizione proxy viene inoltrata all'elemento principale e agli elementi secondari del gestore code.
2. Un gestore code è un membro di un cluster e la stringa dell'argomento della sottoscrizione si risolve in un argomento associato a un oggetto argomento del cluster. Quando l'argomento è un argomento cluster *diretto* , le sottoscrizioni proxy vengono inoltrate a tutti i membri del cluster. Quando l'argomento è un argomento cluster *host argomento instradato* , le sottoscrizioni proxy vengono

inoltrate solo ai gestori code nel cluster che hanno definito l'oggetto argomento cluster. Per ulteriori informazioni, consultare“Cluster di pubblicazione / sottoscrizione” a pagina 94.

Se un gestore code è un membro di un cluster e di una gerarchia, le sottoscrizioni proxy vengono propagate da entrambi i meccanismi senza consegnare le pubblicazioni duplicate al sottoscrittore.

Gli spazi argomenti di tre topologie di pubblicazione / sottoscrizione sono descritti nel seguente elenco:

- “Caso 1. Cluster di pubblicazione / sottoscrizione” a pagina 108.
- “Caso 2. Gerarchie di pubblicazione / sottoscrizione” a pagina 108.

In argomenti separati, le attività di configurazione riportate di seguito descrivono come combinare gli spazi argomento.

- Creazione di un singolo spazio argomento in un cluster di pubblicazione / sottoscrizione.
- Combinazione degli spazi argomento di più cluster.
- Combinazione e isolamento di spazi argomento in più cluster.
- Pubblicazione e sottoscrizione di spazi argomento in più cluster.

Caso 1. Cluster di pubblicazione / sottoscrizione

Nell'esempio, si supponga che il gestore code non sia connesso a una gerarchia di pubblicazione / sottoscrizione.

Se un gestore code è un membro di un cluster di pubblicazione / sottoscrizione, il relativo spazio argomenti è costituito da argomenti locali e argomenti cluster. Gli argomenti locali sono associati a oggetti argomento senza attributo **CLUSTER** . Se un gestore code dispone di definizioni di oggetti argomento locali, il relativo spazio argomento è diverso da un altro gestore code nel cluster che dispone anche di propri oggetti argomento definiti localmente.

In un cluster di pubblicazione / sottoscrizione, non è possibile sottoscrivere un argomento definito su un altro gestore code, a meno che l'argomento a cui si effettua la sottoscrizione non si risolva in un oggetto argomento cluster.

Quando le stesse definizioni di un oggetto argomento del cluster sono richieste su più gestori code, ad esempio quando si utilizza l' *instradamento host argomento*, è importante che tutte le definizioni corrispondano laddove necessario. Per ulteriori informazioni, vedi Creazione di un singolo spazio argomento in un cluster di pubblicazione / sottoscrizione.

Una definizione locale di un oggetto argomento, se la definizione è per un argomento cluster o per un argomento locale, ha la precedenza sullo stesso oggetto argomento definito altrove nel cluster. Viene utilizzato l'argomento definito localmente, anche se l'oggetto definito altrove è più recente.

È importante che un oggetto argomento cluster sia associato alla stessa stringa argomento ovunque nel cluster. Non è possibile modificare la stringa di argomenti a cui è associato un oggetto argomento. Per associare lo stesso oggetto argomento ad una stringa di argomenti differente, è necessario eliminare l'oggetto argomento e ricrearlo con la nuova stringa di argomenti. Se l'argomento è raggruppato in cluster, l'effetto è quello di eliminare le copie dell'oggetto argomento memorizzate sugli altri membri del cluster e quindi di creare copie del nuovo oggetto argomento ovunque nel cluster. Le copie dell'oggetto argomento si riferiscono tutte alla stessa stringa argomento.

È possibile creare accidentalmente due definizioni dello stesso oggetto argomento denominato su gestori code differenti nel cluster, con stringhe di argomenti differenti. Ciò può causare un comportamento confuso, poiché più definizioni dello stesso oggetto argomento con stringhe di argomento differenti possono produrre risultati differenti a seconda di come e dove si fa riferimento all'argomento. Per ulteriori informazioni su questo punto importante, consultare Definizioni di più argomenti cluster con lo stesso nome .

Caso 2. Gerarchie di pubblicazione / sottoscrizione

Nell'esempio, si presupponga che il gestore code non sia un membro di un cluster di pubblicazione / sottoscrizione.

In IBM MQ, se un gestore code è un membro di una gerarchia di pubblicazione / sottoscrizione, il relativo spazio argomenti è costituito da tutti gli argomenti definiti localmente e su gestori code connessi. Lo spazio argomento di tutti i gestori code in una gerarchia è lo stesso. Non vi è alcuna divisione degli argomenti in argomenti locali e globali.

Impostare una delle opzioni **PUBSCOPE** e **SUBSCOPE** su `QMGR`, per impedire che una pubblicazione su un argomento fluisce da un publisher a un sottoscrittore connesso a diversi gestori code nella gerarchia.

Si supponga di definire un oggetto argomento `Alabama` con la stringa argomento `USA/Alabama` sul gestore code `QMA`. Il risultato è il seguente:

1. Lo spazio argomenti in `QMA` ora include l'oggetto argomento `Alabama` e la stringa argomento `USA/Alabama`.
2. Un'applicazione o un amministratore possono creare una sottoscrizione in `QMA` utilizzando il nome oggetto argomento `Alabama`.
3. Un'applicazione può creare una sottoscrizione a qualsiasi argomento, incluso `USA/Alabama`, in qualsiasi gestore code nella gerarchia. Se `QMA` non è stato definito localmente, l'argomento `USA/Alabama` si risolve nell'oggetto argomento `SYSTEM.BASE.TOPIC`.

Concetti correlati

“Ambito della pubblicazione” a pagina 105
Quando si configura un cluster di pubblicazione / sottoscrizione o una gerarchia, l'ambito di una pubblicazione controlla ulteriormente se i gestori code inoltrano una pubblicazione ai gestori code remoti. Utilizzare l'attributo dell'argomento **PUBSCOPE** per gestire l'ambito delle pubblicazioni.

“Ambito della sottoscrizione” a pagina 106
L'ambito di una sottoscrizione controlla se una sottoscrizione su un gestore code riceve le pubblicazioni pubblicate su un altro gestore code in un cluster o gerarchia di pubblicazione / sottoscrizione oppure solo le pubblicazioni dei publisher locali.

Attività correlate

Configurazione delle reti di pubblicazione / sottoscrizione distribuite

IBM MQ Multicast

Il multicast IBM MQ offre un'affidabile messaggistica multicast a bassa latenza e ad elevato fanout.

Multicast è una forma efficiente di messaggistica di pubblicazione / sottoscrizione in quanto può essere scalata a un numero elevato di sottoscrittori senza effetti negativi sulle prestazioni. IBM MQ abilita la messaggistica Multicast affidabile utilizzando riconoscimenti, riconoscimenti negativi e numeri di sequenza per ottenere una messaggistica con latenza ridotta ed elevato fanout.

La consegna equa del multicast IBM MQ abilita una consegna quasi simultanea, garantendo così che nessun destinatario ottenga un vantaggio. Poiché il multicast IBM MQ si serve della rete per consegnare i messaggi, non è necessario un motore di pubblicazione/sottoscrizione per eseguire il fanout di dati. Dopo che un argomento è associato a un indirizzo di gruppo, non è necessario un gestore code perché i publisher e i sottoscrittori possono operare in modalità peer - to - peer. Ciò consente di ridurre il carico sui server del gestore code, pertanto tali server smettono di rappresentare un potenziale punto di errore.

Concetti multicast iniziali

IBM MQ Multicast può essere facilmente integrato in sistemi e applicazioni esistenti utilizzando l'oggetto COMMINFO (Communication Information). Due campi oggetto TOPIC consentono la configurazione rapida degli oggetti TOPIC esistenti per supportare o ignorare il traffico multicast.

Oggetti necessari per multicast

Le seguenti informazioni sono una breve panoramica dei due oggetti necessari per IBM MQ Multicast:

oggetto COMMINFO
L'oggetto COMMINFO contiene gli attributi associati alla trasmissione multicast. Per ulteriori informazioni sui parametri oggetto COMMINFO, consultare DEFINE COMMINFO.

L'unico campo COMMINFO che DEVE essere impostato è il nome dell'oggetto COMMINFO. Questo nome viene quindi utilizzato per identificare l'oggetto COMMINFO in un argomento. È necessario controllare il campo **GRPADDR** dell'oggetto COMMINFO per assicurarsi che il valore sia un indirizzo gruppo multicast valido.

Oggetto sezione
Un argomento è l'oggetto delle informazioni pubblicate in un messaggio di pubblicazione / sottoscrizione e un argomento viene definito creando un oggetto TOPIC. Per ulteriori informazioni sui parametri dell'oggetto TOPIC, consultare DEFINE TOPIC.

Gli argomenti esistenti possono essere utilizzati con il multicast modificando i valori dei seguenti parametri oggetto TOPIC: **COMMINFO** e **MCAST**.

- **COMMINFO** Questo parametro specifica il nome dell'oggetto delle informazioni di comunicazione multicast.
- **MCAST** Questo parametro specifica se il multicast è consentito in questa posizione nella struttura ad albero degli argomenti. Per impostazione predefinita, **MCAST** è impostato su ASPARENT , il che significa che l'attributo multicast dell'argomento viene ereditato dal parent. L'impostazione di **MCAST** su ENABLED consente il traffico multicast su questo nodo.

Argomenti e reti multicast

Le seguenti informazioni sono una panoramica di ciò che accade alle sottoscrizioni con diversi tipi di sottoscrizione e definizione argomento. Tutti questi esempi presuppongono che il parametro **COMMINFO** dell'oggetto TOPIC sia impostato sul nome di un oggetto COMMINFO valido:

Argomento impostato su multicast abilitato
Se il parametro della stringa di argomenti **MCAST** è impostato su ENABLED, le sottoscrizioni dai client con capacità multicast sono consentite e viene effettuata una sottoscrizione multicast a meno che:

- Si tratta di una sottoscrizione durevole da un client multicast.
- Si tratta di una sottoscrizione non gestita da un client multicast.
- Si tratta di una sottoscrizione da un client non multicast.

In questi casi viene effettuata una sottoscrizione non multicast e le sottoscrizioni vengono declassate alla normale pubblicazione / sottoscrizione.

Argomento impostato su multicast disabilitato
Se il parametro della stringa di argomenti **MCAST** è impostato su DISABLED, viene sempre effettuata una sottoscrizione non multicast e le sottoscrizioni vengono declassate alla normale pubblicazione / sottoscrizione.

Argomento impostato solo su multicast
Se il parametro **MCAST** della stringa dell'argomento è impostato su SOLO, le sottoscrizioni dai client con capacità multicast sono consentite e viene effettuata una sottoscrizione multicast a meno che:

- Si tratta di una sottoscrizione duratura: le sottoscrizioni durevoli vengono rifiutate con codice di errore 2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
- È una sottoscrizione non gestita: le sottoscrizioni non gestite vengono rifiutate con codice motivo 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
- Si tratta di una sottoscrizione da un client che non supporta il multicast: queste sottoscrizioni vengono rifiutate con il codice motivo 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
- È una sottoscrizione da un'applicazione collegata localmente: queste sottoscrizioni vengono rifiutate con codice motivo 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY

Linux Windows AIX MQ TelemetryPanoramica

MQ Telemetry comprende un servizio di telemetria (MQXR) che fa parte di un gestore code, i client di telemetria che è possibile scrivere o scaricare gratuitamente e le interfacce di gestione della riga comandi ed explorer. La telemetria si riferisce alla raccolta di dati e alla gestione di una vasta gamma di dispositivi remoti. Con MQ Telemetry è possibile integrare la raccolta di dati e il controllo dei dispositivi con le applicazioni web.

MQ Telemetry è un componente di IBM MQ. L'aggiornamento per queste versioni consiste essenzialmente nell'installazione di una versione successiva di IBM MQ.

Le applicazioni di esempio continuano ad essere liberamente disponibili da Eclipse Paho e MQTT.org. Vedere IBM MQ Telemetry Transport programmi di esempio.

Poiché MQ Telemetry è un componente di IBM MQ, MQ Telemetry può essere installato con il prodotto principale o dopo che il prodotto principale è stato installato. Per informazioni sulla migrazione, consultare Migrazione MQ Telemetry su Windows e Migrazione MQ Telemetry su Linux.

Inclusi in MQ Telemetry sono i seguenti componenti:

Canali di telemetria
Utilizzare i canali di telemetria per gestire la connessione dei client MQTT a IBM MQ . I canali di telemetria utilizzano nuovi oggetti IBM MQ , come `SYSTEM.MQTT.TRANSMIT.QUEUE`, per interagire con IBM MQ.

Servizio di telemetria (MQXR)
I client MQTT utilizzano il servizio di telemetria `SYSTEM.MQXR.SERVICE` per connettersi ai canali di telemetria.

IBM MQ Explorer supporto per MQ Telemetry
MQ Telemetry può essere amministrato utilizzando IBM MQ Explorer.

Documentazione
La documentazione MQ Telemetry è inclusa nella documentazione del prodotto IBM MQ standard. La documentazione SDK per client Java e C viene fornita nella documentazione del prodotto e come Javadoc e HTML.

Concetti di telemetria

Si raccolgono informazioni dall'ambiente circostante per decidere cosa fare. Come consumatore, si controlla ciò che si ha in negozio, prima di decidere quale cibo acquistare. Vuoi sapere quanto tempo ci vorrà per un viaggio se te ne vai ora, prima di prenotare una connessione. Controlli i sintomi, prima di decidere se visitare il medico. Si controlla quando un autobus sta per arrivare, prima di decidere se attendere. Le informazioni per tali decisioni provengono direttamente dai contatori e dai dispositivi, dalla parola scritta su carta o da uno schermo e da te. Ovunque tu sia, e quando hai bisogno di farlo, raccogli informazioni, le metti insieme, le analizza e agisci su di esse.

Se le fonti di informazione sono ampiamente disperse o inaccessibili, diventa difficile e costoso raccogliere le informazioni più accurate. Se ci sono molte modifiche che si desidera apportare, o è difficile apportare le modifiche, le modifiche non vengono apportate o vengono apportate quando sono meno efficaci.

E se i costi di raccolta delle informazioni e di controllo dei dispositivi ampiamente dispersi fossero notevolmente ridotti collegando i dispositivi con la tecnologia digitale a Internet? Le informazioni possono essere analizzate utilizzando le risorse di Internet e dell'azienda. Hai più opportunità di prendere decisioni informate e agire su di esse.

Le tendenze tecnologiche e le pressioni ambientali ed economiche stanno portando a questi cambiamenti:

1. Il costo della connessione e del controllo di sensori e attuatori è in diminuzione, grazie alla standardizzazione e alla connessione a processori digitali a basso costo.

2. Internet, e le tecnologie di internet, sono sempre più utilizzati per collegare i dispositivi. In alcuni paesi, i telefoni cellulari superano i personal computer nel numero di connessioni alle applicazioni Internet. Altri dispositivi stanno sicuramente seguendo.
3. Internet, e le tecnologie di internet, rendono molto più facile per un'applicazione ottenere dati. Un facile accesso ai dati sta guidando l'utilizzo dell'analisi dei dati per trasformare i dati dai sensori in informazioni utili in molte altre soluzioni.
4. L'uso intelligente delle risorse è spesso un modo più rapido e meno costoso di ridurre le emissioni di carbonio e i costi. Le alternative: trovare nuove risorse o sviluppare nuove tecnologie per utilizzare le risorse esistenti potrebbero essere la soluzione a lungo termine. A breve termine lo sviluppo di nuove tecnologie o la ricerca di nuove risorse è spesso più rischioso, più lento e più costoso del miglioramento delle soluzioni esistenti.

Esempio

Un esempio mostra come queste tendenze creano nuove opportunità per interagire in modo intelligente con l'ambiente.

La Convenzione internazionale per la salvaguardia della vita umana in mare (SOLAS) prevede l'impiego di un sistema di identificazione automatica (AIS) su molte navi. È richiesto su navi mercantili di oltre 300 tonnellate e navi passeggeri. L'AIS è principalmente un sistema di prevenzione delle collisioni per la navigazione costiera. È utilizzato dalle autorità marittime per monitorare e controllare le acque costiere.

Gli appassionati di tutto il mondo stanno distribuendo stazioni di tracciamento AIS a basso costo e inserendo informazioni sulla navigazione costiera su Internet. Altri appassionati stanno scrivendo applicazioni che combinano informazioni da AIS con altre informazioni da Internet. I risultati vengono inseriti su siti web e pubblicati utilizzando Twitter e SMS.

In un'applicazione, le informazioni provenienti dalle stazioni AIS vicino a Southampton sono combinate con la proprietà della nave e le informazioni geografiche. L'applicazione fornisce informazioni in tempo reale sugli arrivi e le partenze dei traghetti su Twitter. I pendolari regolari che utilizzano i traghetti tra Southampton e l'isola di Wight si abbonano al feed di notizie utilizzando Twitter o SMS. Se il feed mostra che il loro traghetto è in ritardo, i pendolari possono ritardare la loro partenza e prendere il traghetto quando attracca più tardi del suo orario di arrivo previsto.

Per ulteriori esempi, consultare “Casi di utilizzo della telemetria” a pagina 114.

Attività correlate

InstallazioneMQ Telemetry
AmministrazioneMQ Telemetry
Migrazione di MQ Telemetry su Windows
Migrazione di MQ Telemetry su Linux
Sviluppo di applicazioni per MQ Telemetry
Risoluzione dei problemi di MQ Telemetry

Riferimenti correlati

Riferimento di MQ Telemetry

Linux Windows AIX

Introduzione a MQ Telemetry

Le persone, le aziende e i governi vogliono sempre più utilizzare MQ Telemetry per interagire in modo più intelligente con l'ambiente in cui viviamo e lavoriamo. MQ Telemetry collega tutti i tipi di dispositivi a internet e all'azienda e riduce i costi di creazione di applicazioni per dispositivi intelligenti.

Che cos'è MQ Telemetry?

- È una funzione di IBM MQ che estende il backbone di messaggistica universale fornito da IBM MQ a una vasta gamma di sensori remoti, attuatori e dispositivi di telemetria. MQ Telemetry estende IBM MQ in modo da poter interconnettere le applicazioni aziendali intelligenti, i servizi e i responsabili delle decisioni con le reti di dispositivi strumentati.

- Le parti principali di MQ Telemetry sono:

  Il servizio MQ Telemetry (MQXR).
  Questo servizio viene eseguito nel server IBM MQ e utilizza il protocollo IBM MQ Telemetry Transport (MQTT) per comunicare con le periferiche di telemetria.

  Le applicazioni MQTT che si scrivono.
  Queste applicazioni controllano le informazioni trasmesse tra le periferiche di telemetria e il gestore code IBM MQ e tutte le azioni intraprese in risposta a tali informazioni. Per creare queste applicazioni, utilizzare le librerie client MQTT .

Cosa può fare per me?

- MQTT è un trasporto di messaggistica aperto che consente di creare implementazioni MQTT per un'ampia varietà di periferiche.
- I client MQTT possono essere eseguiti su dispositivi a ingombro ridotto con risorse limitate.
- MQTT funziona in modo efficiente su reti in cui la larghezza di banda è bassa, in cui i costi di invio dei dati sono costosi o che potrebbero essere fragili.
- La consegna dei messaggi è assicurata e disaccoppiata dall'applicazione.
- I programmatori di applicazioni non hanno bisogno di avere conoscenze di programmazione delle comunicazioni.
- I messaggi possono essere scambiati con altre applicazioni di messaggistica. Possono essere un'altra applicazione di telemetria o un'applicazione MQI, JMS o di messaggistica aziendale.

Come lo uso?

- Vengono forniti script di esempio che funzionano con un'applicazione client IBM MQ Telemetry Transport v3 di esempio (`mqttv3app.jar`). Vedere IBM MQ Telemetry Transport programmi di esempio.
- Utilizzare IBM MQ Explorer e gli strumenti associati per gestire la funzione di telemetria di IBM MQ.
- Utilizzare le librerie client per creare applicazioni MQTT che si connettono a un gestore code e che utilizzano la messaggistica di pubblicazione / sottoscrizione.
- Distribuire l'applicazione e la libreria client al dispositivo in cui deve essere eseguita l'applicazione.

Come funziona?

- MQTT è un protocollo di pubblicazione - sottoscrizione. Un'applicazione client MQTT può pubblicare messaggi su un server MQTT o sottoscrivere messaggi inviati dalle applicazioni che si collegano a un server MQTT .
- Le applicazioni client MQTT utilizzano librerie client che implementano il trasporto del messaggio MQTT .
- Un'applicazione client MQTT di base funziona come un client MQ standard ma può essere eseguita su una varietà molto più ampia di piattaforme e reti.
- Il servizio MQ Telemetry (MQXR) trasforma un gestore code IBM MQ in un server MQTT .
- Quando un gestore code IBM MQ agisce come server MQTT , altre applicazioni che si connettono al gestore code possono sottoscrivere e ricevere i messaggi dal client MQTT .
- Il gestore code agisce come router distribuendo i messaggi dalle applicazioni di pubblicazione alle applicazioni di sottoscrizione.
- I messaggi possono essere distribuiti tra diversi tipi di applicazioni client. Ad esempio, tra i client Telemetry e i client JMS .

Nota: MQ Telemetry sostituisce i nodi SCADA ritirati nella versione 7 di WebSphere Message Broker (ora noti come IBM Integration Bus) e viene eseguito su Windows, Linuxe AIX.

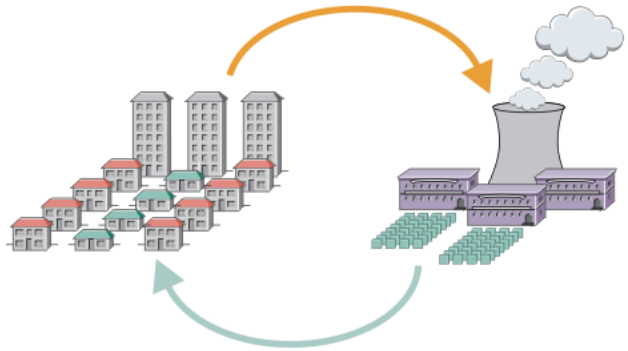
Linux Windows AIX Casi di utilizzo della telemetria

Telemetria è il rilevamento automatico, la misura dei dati e il controllo dei dispositivi remoti. L'accento è posto sulla trasmissione di dati da dispositivi a un punto di controllo centrale. La telemetria comprende anche l'invio di informazioni di controllo e di configurazione ai dispositivi.

MQ Telemetry connette piccoli dispositivi utilizzando MQTT protocole collega i dispositivi ad altre applicazioni utilizzando IBM MQ. MQ Telemetry colma un divario tra i dispositivi e Internet rendendo più semplice la creazione di "soluzioni intelligenti". Le soluzioni intelligenti sbloccano la ricchezza di informazioni disponibili su Internet e nelle applicazioni aziendali, per le applicazioni che monitorano e controllano i dispositivi.

I seguenti diagrammi illustrano alcuni utilizzi tipici di MQ Telemetry:

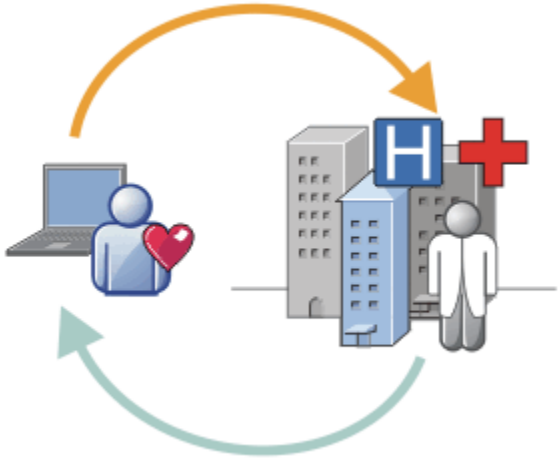
Telemetria: elettricità intelligente



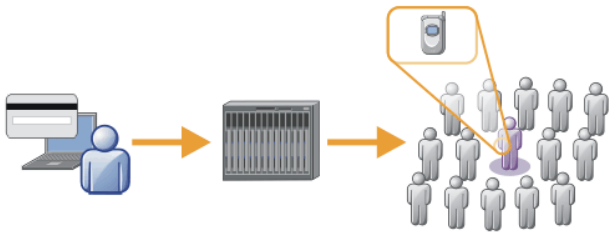
- Messaggio MQTT contenente i dati di utilizzo dell'energia inviati al service provider.
- MQ Telemetry invia i COMANDI DI CONTROLLO in base all'analisi dei dati di utilizzo dell'energia.
- Per ulteriori informazioni, consultare il seguente caso d'uso: “Caso d'uso della telemetria: monitoraggio e controllo dell'energia domestica” a pagina 116

Telemetria: servizi sanitari intelligenti

- MQ Telemetry invia i dati sanitari all'ospedale e al medico.
- Gli avvisi o il feedback dei messaggi MQTT possono essere inviati in base all'analisi dei dati di integrità.
- Per ulteriori informazioni, consultare il seguente caso d'uso: “Caso d'uso della telemetria: monitoraggio dei pazienti a domicilio” a pagina 115



Telemetria: Uno in mezzo alla folla



- Una semplice transazione con carta viene inviata al server della banca.
- MQ Telemetry identifica la persona tra le migliaia, avvisando il cliente che la sua carta è stata utilizzata.
- MQ Telemetry può utilizzare l'input più semplice di informazioni e individuare tale individuo.

I casi di utilizzo descritti nei sottoargomenti sono tratti da esempi reali. Essi illustrano alcuni modi di utilizzare la telemetria e alcuni dei problemi comuni che la tecnologia di telemetria deve risolvere.

Linux Windows AIX Caso d'uso della telemetria: monitoraggio dei pazienti a domicilio

In collaborazione tra IBM e un fornitore di assistenza sanitaria su un sistema di cura del paziente cardiaco, un defibrillatore cardioverter impiantato comunica con un ospedale. I dati sul paziente e sul dispositivo impiantato vengono trasferiti utilizzando la telemetria RF al dispositivo MQTT nella casa di un paziente.

Tipicamente il trasferimento avviene di notte a un trasmettitore situato sul lato del letto. Il trasmettitore trasferisce i dati in modo sicuro attraverso il sistema telefonico all'ospedale, dove i dati vengono analizzati.

Il sistema riduce il numero di visite che un paziente deve effettuare al medico. Rileva quando il paziente o il dispositivo ha bisogno di attenzione e, in caso di emergenza, avvisa il medico di guardia.

La collaborazione tra IBM e il fornitore di assistenza sanitaria ha caratteristiche comuni a diversi casi di utilizzo della telemetria:

Invisibilità

L'unità non richiede alcun intervento dell'utente se non la fornitura di alimentazione, una linea telefonica e la vicinanza all'unità per una parte della giornata. Il suo funzionamento è affidabile e semplice da usare.

Per rimuovere la necessità per il paziente di configurare il dispositivo, il fornitore del dispositivo preconfigura il dispositivo. Il paziente deve solo collegarlo. L'eliminazione della configurazione da parte del paziente semplifica il funzionamento del dispositivo e riduce la possibilità che il dispositivo venga configurato in modo errato.

Il client MQTT è integrato come parte della periferica. Lo sviluppatore del dispositivo integra l'implementazione del client MQTT nel dispositivo e lo sviluppatore o il fornitore configura il client MQTT come parte della pre - configurazione.

Il client MQTT viene fornito come un file JAR Java SE , che lo sviluppo include nella propria applicazione Java . Per ambienti nonJava , come questo, lo sviluppatore del dispositivo può implementare un client in un linguaggio diverso utilizzando il protocollo e formati MQTT pubblicati. In alternativa, lo sviluppatore può utilizzare uno dei client C forniti come librerie condivise per le piattaforme Windows, Linux e ARM.

Connettività irregolare

La comunicazione tra il defibrillatore e l'ospedale ha caratteristiche di rete irregolari. Due diverse reti vengono utilizzate per risolvere i diversi problemi di raccolta dei dati dal paziente e di invio dei dati all'ospedale. Tra il brevetto e il dispositivo MQTT , viene utilizzata una rete RF a bassa potenza a corto raggio. Il trasmettitore si connette all'ospedale utilizzando una connessione TCP/IP VPN su una linea telefonica a bassa larghezza di banda.

Spesso non è pratico trovare un modo per collegare ogni dispositivo direttamente a una rete Internet Protocol . Utilizzare due reti, collegate da un hub, è una soluzione comune. Il dispositivo MQTT è un hub semplice, che memorizza le informazioni dal paziente e le inoltra all'ospedale.

Sicurezza

Il medico deve essere in grado di fidarsi dell'autenticità dei dati del paziente e il paziente vuole che venga rispettata la privacy dei propri dati.

In alcune situazioni è sufficiente crittografare la connessione, utilizzando VPN o TLS. In altre situazioni, è auspicabile mantenere i dati al sicuro anche dopo che sono stati archiviati.

A volte il dispositivo di telemetria non è sicuro. Potrebbe essere in un'abitazione condivisa, ad esempio. L'utente del dispositivo deve essere autenticato per assicurarsi che i dati provengano dal

paziente corretto. Il dispositivo stesso può essere autenticato sul server utilizzando TLS e il server autenticato sul dispositivo.

Il canale di telemetria tra il dispositivo e il gestore code supporta JAAS per l'autenticazione utente e TLS per la crittografia delle comunicazioni e l'autenticazione del dispositivo. L'accesso a una pubblicazione è controllato dal gestore autorizzazione oggetto in IBM MQ.

L'identificativo utilizzato per autenticare l'utente può essere associato a un altro identificativo, ad esempio un'identità comune del paziente. Un identificativo comune semplifica la configurazione dell'autorizzazione per gli argomenti di pubblicazione in IBM MQ.

Connettività

La connessione tra la periferica MQTT e l'ospedale utilizza la connessione remota e funziona con una larghezza di banda inferiore a 300 baud.

Per operare efficacemente a 300 baud, MQTT protocol aggiunge solo pochi byte aggiuntivi a un messaggio oltre alle intestazioni TCP/IP.

MQTT protocol fornisce la messaggistica *fire and forget* a trasmissione singola, che mantiene basse le latenze. Può anche utilizzare più trasmissioni per garantire *almeno una volta* e *esattamente una volta* la consegna se la consegna garantita è più importante del tempo di risposta. Per garantire la consegna, i messaggi vengono memorizzati sul dispositivo fino a quando non vengono consegnati correttamente. Se un dispositivo è collegato in modalità wireless, la consegna garantita è particolarmente utile.

Scalabilità

I dispositivi di telemetria sono in genere distribuiti in grandi quantità, da decine di migliaia a milioni.

Il collegamento di molti dispositivi a un sistema richiede una soluzione di grandi dimensioni. Ci sono richieste di business come il costo dei dispositivi e il loro software, e le richieste di amministrazione di gestire licenze, dispositivi e utenti. I requisiti tecnici includono il carico sulla rete e sui server.

L'apertura delle connessioni utilizza più risorse del server rispetto alla gestione delle connessioni aperte. Ma in un caso d'uso come questo che utilizza le linee telefoniche, la spesa delle connessioni significa che le connessioni vengono lasciate aperte non più del necessario. I trasferimenti di dati sono in gran parte di natura batch. I collegamenti possono essere programmati per tutta la notte per evitare un picco improvviso di collegamenti al momento di coricarsi.

Sul client, la scalabilità dei client è agevolata dalla configurazione client minima richiesta. Il client MQTT è integrato nella periferica. Non c'è alcun requisito per una fase di configurazione o di accettazione della licenza client MQTT da incorporare nella distribuzione dei dispositivi ai pazienti.

Sul server, MQ Telemetry ha una destinazione iniziale di 50.000 connessioni aperte per gestore code.

Le connessioni sono gestite utilizzando IBM MQ Explorer. IBM MQ Explorer filtra le connessioni da visualizzare in un numero gestibile. Con uno schema scelto in modo appropriato di assegnare identificativi ai client, è possibile filtrare le connessioni in base all'area geografica o in ordine alfabetico in base al nome del paziente.

Linux Windows AIX Caso d'uso della telemetria: monitoraggio e controllo dell'energia domestica

I contatori intelligenti raccolgono maggiori dettagli sul consumo di energia rispetto ai contatori tradizionali.

I contatori intelligenti sono spesso accoppiati con una rete di telemetria locale per monitorare e controllare i singoli elettrodomestici in una casa. Alcuni sono anche collegati da remoto per il monitoraggio e il controllo a distanza.

La connessione remota può essere impostata da un individuo, da un programma di utilità o da un punto di controllo centrale. Il punto di controllo remoto può leggere l'utilizzo di energia e fornire i dati di utilizzo. Può fornire dati per influenzare l'utilizzo, come prezzi continui e informazioni meteorologiche. Può limitare il carico per migliorare l'efficienza generale della generazione di energia.

I contatori intelligenti stanno iniziando a essere ampiamente utilizzati. Il governo del Regno Unito, ad esempio, è in consultazione per l'introduzione di contatori intelligenti in ogni casa del Regno Unito entro il 2020.

I casi di utilizzo della misurazione a domicilio hanno una serie di caratteristiche comuni:

Invisibilità

A meno che l'utente non desideri essere coinvolto nel risparmio energetico utilizzando il contatore, il contatore non deve richiedere l'intervento dell'utente. Non deve ridurre l'affidabilità dell'approvvigionamento energetico dei singoli apparecchi.

Un client MQTT può essere incorporato nel software installato con il misuratore e non richiede un'installazione o una configurazione separate.

Connettività irregolare

La comunicazione tra le apparecchiature e il contatore intelligente richiede standard di connettività diversi rispetto a quelli tra il contatore e il punto di connessione remoto.

La connessione dal contatore intelligente alle apparecchiature deve essere altamente disponibile e conforme agli standard di rete per una rete domestica.

È probabile che la rete remota utilizzi varie connessioni fisiche. Alcuni di loro, come il cellulare, hanno un costo di trasmissione elevato e possono essere intermittenti. La specifica MQTT v3 è rivolta alle connessioni remote e alle connessioni tra gli adattatori locali e lo smart meter.

La connessione tra le prese di corrente e le applicazioni, e il contatore, utilizza una rete di aree domestiche, come Zigbee. MQTT per reti di sensori (MQTT-S), è progettato per funzionare con Zigbee e altri protocolli di rete a bassa larghezza di banda. MQ Telemetry non supporta MQTT-S direttamente. Richiede un gateway per collegare MQTT-S a MQTT v3.

Come il monitoraggio del paziente a casa, le soluzioni per il monitoraggio e il controllo dell'energia a casa richiedono reti multiple, connesse utilizzando lo smart meter come hub.

Sicurezza

Esistono diversi problemi di sicurezza associati ai contatori intelligenti. Questi problemi includono il non rifiuto delle transazioni, l'autorizzazione di eventuali azioni di controllo avviate e la riservatezza dei dati sul consumo di energia.

Per garantire la privacy, i dati trasferiti tra il contatore e il punto di controllo remoto da MQTT possono essere codificati utilizzando TLS. Per garantire l'autorizzazione delle azioni di controllo, la connessione MQTT tra il misuratore e il punto di controllo remoto può essere autenticata reciprocamente utilizzando TLS.

Connettività

La natura fisica della rete remota può variare notevolmente. Potrebbe utilizzare una connessione a banda larga esistente o utilizzare una rete mobile con costi di chiamata elevati e disponibilità intermittente. Per connessioni ad alto costo, intermittenti, MQTT è un protocollo efficiente e affidabile; consultare “Caso d'uso della telemetria: monitoraggio dei pazienti a domicilio” a pagina 115.

Scalabilità

Alla fine le aziende elettriche, o punti di controllo centrali, pianificano di implementare decine di milioni di contatori intelligenti. Inizialmente, il numero di metri per distribuzione è tra le decine e le centinaia di migliaia. Questo numero è paragonabile alla destinazione MQTT iniziale di 50.000 connessioni client aperte per gestore code.

Un aspetto critico dell'architettura per il monitoraggio e il controllo dell'energia domestica è quello di utilizzare il contatore intelligente come concentratore di rete. Ogni adattatore del dispositivo è un sensore separato. Collegandoli a un hub locale utilizzando MQTT, l'hub può concentrare i flussi di dati su una singola sessione TCP/IP con il punto di controllo centrale e memorizzare i messaggi per un breve periodo per superare le interruzioni di sessione.

Le connessioni remote devono essere lasciate aperte nei casi di utilizzo dell'energia domestica per due motivi. Innanzitutto, perché l'apertura delle connessioni richiede molto tempo rispetto all'invio delle richieste. Il tempo per aprire molte connessioni per inviare richieste di "limitazione del carico" in un breve intervallo è troppo lungo. In secondo luogo, per ricevere le richieste di limitazione del carico dalla società elettrica, la connessione deve essere prima aperta dal client. Con MQTT, le connessioni vengono sempre avviate dal client e, per ricevere le richieste di limitazione del carico dalla società elettrica, la connessione deve essere lasciata aperta.

Se la frequenza di apertura delle connessioni è critica, o se il server avvia richieste di tipo time - critical, la soluzione è in genere quella di mantenere molte connessioni aperte.

Linux Windows AIX Casi di utilizzo della telemetria: RFID (Radio Frequency Identification)

RFID è l'uso di un tag RFID incorporato per identificare e tenere traccia di un oggetto in modalità wireless. I tag RFID possono essere letti fino a una gamma di diversi metri e fuori dalla linea di vista del lettore RFID. I tag passivi sono attivati da un lettore RFID. I tag attivi vengono trasmessi senza attivazione esterna. I tag attivi devono avere una fonte di energia. I tag passivi possono includere una fonte di alimentazione per aumentare il loro intervallo.

L'RFID è utilizzato in molte applicazioni e i tipi di casi d'uso variano enormemente. I casi d'uso RFID e il monitoraggio dei pazienti a domicilio e il monitoraggio dell'energia a domicilio e i casi d'uso di controllo, hanno alcune somiglianze e differenze.

Invisibilità

In molti casi di utilizzo, il lettore RFID è distribuito in grandi quantità e deve funzionare senza l'intervento dell'utente. Il lettore include un client MQTT integrato per comunicare con un punto di controllo centrale.

Ad esempio, in un magazzino di distribuzione, un lettore utilizza un sensore di movimento per rilevare un pallet. Attiva i tag RFID degli elementi sul pallet e invia dati e richieste alle applicazioni centrali. I dati vengono utilizzati per aggiornare l'ubicazione dello stock. Le richieste controllano ciò che accade al pallet successivo, ad esempio spostarlo in una particolare baia. Le compagnie aeree e i sistemi di bagagli aeroportuali utilizzano la tecnologia RFID in questo modo.

In alcuni casi d'uso RFID, il lettore dispone di un ambiente di elaborazione standard, come Java Platform, Micro Edition (Java ME). In questi casi, il client MQTT potrebbe essere distribuito in un passo di configurazione distinto, dopo la produzione.

Connettività irregolare

I lettori RFID potrebbero essere separati dalla periferica di controllo locale che contiene un client MQTT oppure ogni lettore potrebbe incorporare un client MQTT . In genere, i fattori geografici o di comunicazione indicano la scelta della topologia.

Sicurezza

Privacy e autenticità sono problemi di sicurezza nell'allegato dei tag RFID. I tag RFID sono discreti e possono essere monitorati, falsificati o manomessi.

La soluzione dei problemi di sicurezza RFID aumenta l'opportunità di implementare nuove soluzioni RFID. Sebbene l'esposizione alla sicurezza sia nel tag RFID e nel lettore locale, l'utilizzo dell'elaborazione centrale delle informazioni suggerisce approcci per contrastare diverse minacce. Ad esempio, la manomissione di tag potrebbe essere rilevata correlando dinamicamente i livelli di scorte rispetto alle consegne e alle spedizioni.

Connettività

Le applicazioni RFID in genere coinvolgono sia l'archiviazione in batch che l'inoltro delle informazioni raccolte dai lettori RFID e dalle query immediate. Nel caso d'uso del magazzino di distribuzione, il lettore RFID è sempre collegato. Quando un tag viene letto, viene pubblicato insieme alle informazioni sul lettore. L'applicazione di archiviazione pubblica la risposta al lettore.

Nell'applicazione di archiviazione la rete è in genere affidabile e le richieste immediate potrebbero utilizzare i messaggi *fire and forget* (attiva e dimentica) per prestazioni a bassa latenza. I dati di archiviazione e inoltro in batch potrebbero utilizzare la messaggistica *esattamente una volta* per ridurre al minimo i costi di gestione associati alla perdita di dati.

Scalabilità

Se l'applicazione RFID richiede risposte immediate, nell'ordine di uno o due secondi, i lettori RFID devono rimanere connessi.

 Casi di utilizzo della telemetria: rilevamento dell'ambiente

Il rilevamento ambientale utilizza la telemetria per raccogliere informazioni sui livelli e sulla qualità delle acque fluviali, sugli inquinanti atmosferici e su altri dati ambientali.

I sensori si trovano spesso in luoghi remoti, senza accesso alla comunicazione cablata. La larghezza di banda wireless è costosa e l'affidabilità può essere bassa. In genere, una serie di sensori di ambiente in una piccola area geografica sono collegati a un dispositivo di monitoraggio locale in una posizione sicura. Le connessioni locali potrebbero essere cablate o wireless.

Invisibilità

È probabile che i dispositivi del sensore siano meno accessibili, meno alimentati e distribuiti in numero maggiore rispetto al dispositivo di monitoraggio centrale. I sensori sono a volte "stupidi" e il dispositivo di monitoraggio locale include adattatori per trasformare e memorizzare i dati del sensore. È probabile che il dispositivo di monitoraggio incorpori un computer di uso generale che supporta Java Platform, Standard Edition (Java SE) o Java Platform, Micro Edition (Java ME). È improbabile che l'invisibilità sia un requisito importante quando si configura il client MQTT .

Connettività irregolare

Le funzionalità dei sensori e il costo e la larghezza di banda della connessione remota, di solito si traduce in un hub di monitoraggio locale connesso a un server centrale.

Sicurezza

A meno che la soluzione non venga utilizzata in un caso di uso militare o difensivo, la sicurezza non è un requisito importante.

Connettività

Molti utilizzi non richiedono un monitoraggio continuo o una disponibilità immediata dei dati. I dati di eccezione, come un avviso di livello di inondazione, devono essere inoltrati immediatamente. I dati del sensore vengono aggregati sul monitor locale per ridurre i costi di connessione e comunicazione e quindi trasferiti utilizzando le connessioni pianificate. I dati di eccezione vengono inoltrati non appena vengono rilevati sul monitor.

Scalabilità

I sensori sono concentrati intorno agli hub locali e i dati del sensore vengono aggregati in pacchetti che vengono trasmessi in base a una pianificazione. Entrambi questi fattori riducono il carico sul server centrale che verrebbe imposto utilizzando sensori collegati direttamente.

 Casi di utilizzo della telemetria: applicazioni mobili

Le applicazioni mobili sono applicazioni che vengono eseguite su dispositivi wireless. I dispositivi sono piattaforme di applicazioni generiche o dispositivi personalizzati.

Le piattaforme generali includono dispositivi portatili come telefoni e assistenti di dati personali e dispositivi portatili come computer notebook. I dispositivi personalizzati utilizzano hardware per scopi speciali su misura per applicazioni specifiche. Un dispositivo per registrare la consegna dei pacchi "signed-for" è un esempio di dispositivo mobile personalizzato. Le applicazioni su dispositivi mobili personalizzati sono spesso costruite su una piattaforma software generica.

Invisibilità

La distribuzione delle applicazioni mobili personalizzate viene gestita e può includere la configurazione dell'applicazione client MQTT . È improbabile che l'invisibilità sia un requisito importante quando si configura il client MQTT .

Connettività irregolare

A differenza della topologia dell'hub locale dei precedenti casi di utilizzo, i client mobili si collegano in remoto. Il livello dell'applicazione client si connette direttamente a un'applicazione nell'hub centrale.

Sicurezza

Con poca sicurezza fisica, il dispositivo mobile e l'utente mobile devono essere autenticati. TLS viene utilizzato per confermare l'identità della periferica e JAAS per autenticare l'utente.

Connettività

Se l'applicazione mobile dipende dalla copertura wireless, deve essere in grado di operare offline e di gestire in maniera efficiente una connessione interrotta. In questo ambiente, l'obiettivo è rimanere connessi, ma l'applicazione deve essere in grado di memorizzare e inoltrare i messaggi. Spesso i messaggi sono ordini, o conferme di consegna, e hanno un valore di business importante. Devono essere archiviati e inoltrati in modo affidabile.

Scalabilità

La scalabilità non è un problema importante. È probabile che il numero di client applicativi non superi le migliaia, o le decine di migliaia, nei casi di utilizzo di applicazioni mobili personalizzate.

Linux Windows AIX Connessione di dispositivi di telemetria a un gestore code

I dispositivi di telemetria si collegano a un gestore code utilizzando un client MQTT v3 . Il client MQTT v3 utilizza TCP/IP per connettersi a un listener TCP/IP denominato servizio di telemetria (MQXR).

Quando si connette un dispositivo di telemetria a un gestore code, il client MQTT avvia una connessione TCP/IP utilizzando il metodo `MqttClient.connect` . Come i client IBM MQ , un client MQTT deve essere connesso al gestore code per inviare e ricevere messaggi. La connessione viene effettuata sul server utilizzando un listener TCP/IP, installato con MQ Telemetry, denominato servizio di telemetria (MQXR). Ogni gestore code esegue un massimo di un servizio di telemetria (MQXR).

Il servizio di telemetria (MQXR) utilizza l'indirizzo socket remoto impostato da ciascun client nel metodo `MqttClient.connect` per assegnare la connessione a un canale di telemetria. Un indirizzo socket è la combinazione di nome host TCP/IP e numero di porta. Più client che utilizzano lo stesso indirizzo socket remoto sono connessi allo stesso canale di telemetria dal servizio di telemetria (MQXR).

Se ci sono più gestori code su un server, suddividere i canali di telemetria tra i gestori code. Allocare gli indirizzi socket remoti tra i gestori code. Definire ogni canale di telemetria con un indirizzo socket remoto univoco. Due canali di telemetria non devono utilizzare lo stesso indirizzo socket.

Se lo stesso indirizzo del socket remoto è configurato per i canali di telemetria su più gestori code, vince il primo canale di telemetria a connettersi. I canali successivi che si collegano sullo stesso indirizzo non riescono.

Se sul server sono presenti più adattatori di rete, suddividere gli indirizzi socket remoti tra i canali di telemetria. L'assegnazione degli indirizzi socket è del tutto arbitraria, purché qualsiasi indirizzo socket specifico sia configurato su un solo canale di telemetria.

Configurare IBM MQ per connettere i client MQTT utilizzando le procedure guidate fornite nel supplemento MQ Telemetry per IBM MQ Explorer. In alternativa, seguire le istruzioni riportate in Configurazione di un gestore code per la telemetria su Linux e AIX e Configurazione di un gestore code per la telemetria su Windows per configurare la telemetria manualmente.

Riferimenti correlati

Proprietà MQXR

Linux Windows AIX Protocolli di connessione di telemetria

MQ Telemetry supporta TCP/IP IPv4 e IPv6e TLS.

Linux Windows AIX Servizio di telemetria (MQXR)

Il servizio di telemetria (MQXR) è un listener TCP/IP, gestito come servizio IBM MQ . Creare il servizio utilizzando una procedura guidata IBM MQ Explorer o con un comando **runmqsc** .

Il servizio MQ Telemetry (MQXR) è denominato SYSTEM.MQXR.SERVICE .

La procedura guidata di configurazione dell'esempio di telemetria, fornita in MQ Telemetry funzione per IBM MQ Explorer, crea il servizio di telemetria e un canale di telemetria di esempio; consultare Verifica dell'installazione di MQ Telemetry utilizzando IBM MQ Explorer .

Creare la configurazione di esempio dalla linea di comando; consultare Verifica dell'installazione di MQ Telemetry utilizzando la riga di comando.

Il servizio di telemetria (MQXR) viene avviato e arrestato automaticamente con il gestore code. Controllare il servizio utilizzando la cartella dei servizi in IBM MQ Explorer. Per vedere il servizio, devi fare clic sull'icona per arrestare IBM MQ Explorer il filtraggio degli oggetti SYSTEM dalla visualizzazione.

Per un esempio di come creare il servizio manualmente, consultare

- Linux AIX Creazione di SYSTEM.MQXR.SERVICE su Linux.
- Windows Creazione di SYSTEM.MQXR.SERVICE su Windows.

Questi argomenti specificano anche la chiave predefinita per richiedere le passphrase per i canali MQTT TLS da codificare. Per ulteriori informazioni, vedi Encryption of passphrase for MQTT TLS channels.

Linux Windows AIX Canali di telemetria

Creare canali di telemetria per creare connessioni con proprietà differenti, come JAAS (Java Authentication and Authorization Service) o autenticazione TLS oppure per gestire gruppi di client.

Creare canali di telemetria utilizzando la procedura guidata **New Telemetry Channel** , fornita nella funzione MQ Telemetry per IBM MQ Explorer. Configurare un canale, utilizzando il wizard, per accettare connessioni dai client MQTT su una particolare porta TCP/IP. Da IBM WebSphere MQ 7.1, è possibile configurare MQ Telemetry utilizzando il programma della riga comandi, **runmqsc**.

Creare più canali di telemetria, su porte diverse, per rendere più semplice la gestione di un gran numero di connessioni client, suddividendo i client in gruppi. Ogni canale di telemetria ha un nome differente.

È possibile configurare canali di telemetria con attributi di sicurezza differenti per creare diversi tipi di connessione. Creare più canali per accettare connessioni client su indirizzi TCP/IP differenti. Utilizzare TLS per crittografare i messaggi e autenticare il client e il canale di telemetria; consultare Configurazione TLS dei client MQTT e dei canali di telemetria. Specificare l'ID utente per semplificare l'autorizzazione dell'accesso agli oggetti IBM MQ . Specificare una configurazione JAAS per autenticare l'utente MQTT con JAAS; consultare MQTT client identification, authorization, and authentication.

Linux Windows AIX IBM MQ Telemetry Transport protocollo

Il protocollo IBM MQ Telemetry Transport (MQTT) v3 è progettato per lo scambio di messaggi tra periferiche di piccole dimensioni su una larghezza di banda bassa o su connessioni costose e per inviare messaggi in modo affidabile. Utilizza TCP/IP.

MQTT protocol è pubblicato; consultare IBM MQ Telemetry Transport formato e protocollo. La Versione 3 del protocollo utilizza la pubblicazione / sottoscrizione e supporta tre QOS (quality of service): *attiva e dimentica*, *almeno una volta*e *esattamente una volta*.

La dimensione ridotta delle intestazioni del protocollo e il payload del messaggio dell'array di byte, mantengono i messaggi piccoli. Le intestazioni comprendono un'intestazione fissa a 2 byte e fino a

12 byte di intestazioni variabili aggiuntive. Il protocollo utilizza intestazioni variabili a 12 byte per la sottoscrizione e la connessione e solo intestazioni variabili a 2 byte per la maggior parte delle pubblicazioni.

Con tre QoS (quality of service), puoi scambiare tra bassa latenza e affidabilità; vedi Qualità del servizio fornito da un client MQTT. *Fire and forget* non utilizza alcuna memoria del dispositivo persistente e solo una trasmissione per inviare o ricevere una pubblicazione. *Almeno una volta*e *esattamente una volta* richiedono l'archiviazione persistente sul dispositivo per mantenere lo stato del protocollo e salvare un messaggio fino a quando non viene riconosciuto.

Linux Windows AIX MQTT client

Un'app del client MQTT è responsabile della raccolta delle informazioni dal dispositivo di telemetria, della connessione al server e della pubblicazione delle informazioni sul server. Può anche sottoscrivere argomenti, ricevere pubblicazioni e controllare il dispositivo di telemetria.

A differenza delle applicazioni client IBM MQ , le MQTT applicazioni client non sono applicazioni IBM MQ . Non specificano un gestore code a cui connettersi. Non sono limitati all'utilizzo di interfacce di programmazione IBM MQ . Invece, i client MQTT implementano il protocollo MQTT 3 . È possibile scrivere la libreria client per interfacciarsi con MQTT protocol nel linguaggio di programmazione e sulla piattaforma di propria scelta. Vedere IBM MQ Telemetry Transport formato e protocollo.

Per semplificare la scrittura di applicazioni client MQTT , utilizzare le librerie client C, Javae JavaScript che incapsulano MQTT protocol per un numero di piattaforme. Se incorpori queste librerie nelle tue applicazioni MQTT , un client MQTT completamente funzionale può essere breve fino a 15 righe di codice. Le librerie client MQTT sono disponibili gratuitamente da Eclipse Paho e MQTT.org. Vedere IBM MQ Telemetry Transport programmi di esempio.

L'applicazione del client MQTT è sempre responsabile dell'avvio di un collegamento con un canale di telemetria. Dopo la connessione, l'app client MQTT o un'applicazione IBM MQ possono avviare uno scambio di messaggi.

Le applicazioni client MQTT e le applicazioni IBM MQ pubblicano e sottoscrivono la stessa serie di argomenti. Un'applicazione IBM MQ può anche inviare un messaggio direttamente a un'applicazione client MQTT senza che l'applicazione client crei prima una sottoscrizione. Consultare Configurazione dell'accodamento distribuito per inviare messaggi ai client MQTT.

Le applicazioni client MQTT vengono connesse a IBM MQ utilizzando un canale di telemetria. Il canale di telemetria funge da ponte tra i diversi tipi di messaggi utilizzati da MQTT e IBM MQ. Crea pubblicazioni e sottoscrizioni nel gestore code per conto dell'applicazione client MQTT . Il canale di telemetria invia pubblicazioni che corrispondono alle sottoscrizioni di un'app client MQTT dal gestore code all'applicazione client MQTT .

Linux Windows AIX Invio di un messaggio a un client MQTT

Le applicazioni IBM MQ possono inviare i messaggi dei client MQTT v3 mediante la pubblicazione su sottoscrizioni create dai client o inviando direttamente i messaggi. I client MQTT possono inviare messaggi tra loro pubblicando gli argomenti sottoscritti da altri client.

Un client MQTT sottoscrive una pubblicazione, che riceve da IBM MQ

Eseguire l'attività “Pubblicazione di un messaggio nel programma di utilità del client MQTT da IBM MQ Explorer” a pagina 124 per inviare una pubblicazione da IBM MQ a un client MQTT .

Il modo standard per un client MQTT v3 per ricevere i messaggi consiste nel creare una sottoscrizione a un argomento o a una serie di argomenti. Nel frammento di codice di esempio, Figura 44 a pagina 123, il client MQTT effettua la sottoscrizione utilizzando la stringa argomento `"MQTT Examples"`. Un'applicazione IBM MQ C, Figura 45 a pagina 124, pubblica l'argomento utilizzando la stringa di argomenti `"MQTT Examples"`. Nel frammento di codice Figura 46 a pagina 124, il client MQTT riceve la pubblicazione nel metodo callback, `messageArrived`.

Per ulteriori informazioni su come configurare IBM MQ per inviare pubblicazioni in risposta alle sottoscrizioni dai client MQTT , consultare Pubblicazione di un messaggio in risposta a una sottoscrizione client MQTT.

Un'applicazione IBM MQ invia un messaggio direttamente ad un client MQTT

Eseguire l'attività, “Invio di un messaggio a un client MQTT utilizzando IBM MQ Explorer” a pagina 129 per inviare un messaggio direttamente da IBM MQ a un client MQTT .

Un messaggio inviato in tal modo a un client MQTT viene definito messaggio non richiesto. I client MQTT v3 ricevono messaggi non richiesti come pubblicazioni con un nome argomento impostato. Il servizio di telemetria (MQXR) imposta il nome dell'argomento sul nome della coda remota.

Per ulteriori informazioni su come configurare IBM MQ per inviare messaggi direttamente ai client MQTT , consultare Invio diretto di un messaggio a un cliente.

Un client MQTT pubblica un messaggio

Un MQTT v3 client può pubblicare un messaggio ricevuto da un altro client MQTT v3 , ma non può inviare un messaggio non richiesto. Il frammento di codici Figura 47 a pagina 124 mostra come un client MQTT v3 , scritto in Java, pubblica un messaggio.

Il modello tipico per l'invio di un messaggio a un client MQTT v3 , è che ciascun client crei una sottoscrizione al proprio `ClientIdentifier`. Eseguire l'attività “Pubblicazione di un messaggio in un client MQTT v3 specifico” a pagina 130 per pubblicare un messaggio da un client MQTT a un altro MQTT utilizzando `ClientIdentifier` come stringa di argomento.

Frammenti di codice di esempio

Il frammento di codice in Figura 44 a pagina 123 mostra come un client MQTT scritto in Java crea una sottoscrizione. È inoltre necessario un metodo di callback, `messageArrived` per ricevere le pubblicazioni per la sottoscrizione.

```
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. sottoscrittore client MQTT v3

Il frammento di codice in Figura 45 a pagina 124 mostra come un'applicazione IBM MQ scritta in C invia una pubblicazione. Il frammento di codice viene estratto dall'attività Crea un publisher in un argomento variabile

```
/* 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. IBM MQ publisher

Quando la pubblicazione arriva, il client MQTT richiama il metodo `messageArrived` della classe MQTT client delle applicazioni `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. `messageArrived` metodo

Figura 47 a pagina 124 mostra una pubblicazione MQTT v3 di un messaggio nella sottoscrizione creata in Figura 44 a pagina 123.

```
      String          address = "localhost";
      String         clientId = String.format("%-23.23s",
                                System.getProperty("user.name") + "_" +
                                (UUID.randomUUID().toString())).trim()).replace('-', '_');
      MqttClient       client = new MqttClient(address, clientId);
      String      topicString = "MQTT Examples";
      MqttTopic         topic = client.getTopic(Example.topicString);
      String     publication  = "Hello world";
      MqttMessage     message = new MqttMessage(publication.getBytes());
      MqttDeliveryToken token = topic.publish(message);
```

Figura 47. Programma di pubblicazione client MQTT v3

Linux Windows AIX Pubblicazione di un messaggio nel programma di utilità del client MQTT da IBM MQ Explorer

Attenersi alla procedura descritta in questa attività per pubblicare un messaggio utilizzando IBM MQ Explorere sottoscriverlo con il programma di utilità del client MQTT . Un'attività aggiuntiva mostra come

configurare un alias del gestore code piuttosto che impostare la coda di trasmissione predefinita su SYSTEM.MQTT.TRANSMIT.QUEUE.

Prima di iniziare

L'attività presuppone che l'utente abbia dimestichezza con IBM MQ e IBM MQ Explorere che IBM MQ e MQ Telemetry siano installati.

L'utente che crea le risorse del gestore code per questa attività deve disporre di autorizzazioni sufficienti. A scopo dimostrativo, si presuppone che l'ID utente IBM MQ Explorer sia membro del gruppo mqm .

Informazioni su questa attività

Nell'attività, si crea un topic in IBM MQ e si sottoscrive l'topic utilizzando il programma di utilità del client MQTT . Quando si pubblica l'argomento utilizzando IBM MQ Explorer, il client MQTT riceve la pubblicazione.

Procedura

Effettuare una delle seguenti attività:

- È stato installato MQ Telemetry, ma non è stato ancora avviato. Eseguire l'attività: "Avvia attività senza alcun servizio di telemetria (MQXR) ancora definito" a pagina 125.
- Si è già eseguita la telemetria IBM MQ , ma si desidera utilizzare un nuovo gestore code per eseguire la dimostrazione. Eseguire l'attività: "Avvia attività senza alcun servizio di telemetria (MQXR) ancora definito" a pagina 125.
- Si desidera eseguire l'attività utilizzando un gestore code esistente che non dispone di risorse di telemetria definite. Non si desidera eseguire la procedura guidata **Definisci configurazione di esempio** .
  a. Eseguire una delle seguenti attività per configurare la telemetria:
     – Configurazione di un gestore code per la telemetria su Linux e AIX
     – Configurazione di un gestore code per la telemetria su Windows
  b. Eseguire l'attività: "Avvia attività con un servizio di telemetria in esecuzione (MQXR)" a pagina 126
- Se si desidera eseguire l'attività utilizzando un gestore code esistente che dispone già di risorse di telemetria definite, eseguire l'attività: "Avvia attività con un servizio di telemetria in esecuzione (MQXR)" a pagina 126.

Operazioni successive

Eseguire "Invio di un messaggio a un client MQTT utilizzando IBM MQ Explorer" a pagina 129 per inviare un messaggio direttamente al programma di utilità client.

Avvia attività senza alcun servizio di telemetria (MQXR) ancora definito

Creare un gestore code ed eseguire **Definisci configurazione di esempio** per definire le risorse di telemetria di esempio per il gestore code. Pubblicare un messaggio utilizzando IBM MQ Explorere sottoscriverlo con il programma di utilità client MQTT .

Informazioni su questa attività

Quando si impostano le risorse di telemetria di esempio utilizzando **Definisci configurazione di esempio**, la procedura guidata imposta le autorizzazioni ID utente guest. Considerare attentamente se si desidera che l'ID utente guest sia autorizzato in questo modo. A guest su Windowse a nobody su Linuxviene concessa l'autorizzazione a pubblicare e sottoscrivere la root della struttura ad albero degli argomenti e a inserire i messaggi in SYSTEM.MQTT.TRANSMIT.QUEUE.

La procedura guidata imposta anche la coda di trasmissione predefinita su SYSTEM.MQTT.TRANSMIT.QUEUE, che potrebbe interferire con le applicazioni in esecuzione su un gestore code esistente. È possibile, ma laborioso, configurare la telemetria e non utilizzare la coda di

trasmissione predefinita; eseguire la seguente attività: “Utilizzo di un alias del gestore code” a pagina 127. In questa attività, si crea un gestore code per evitare la possibilità di interferire con qualsiasi coda di trasmissione predefinita esistente.

Procedura

1. Utilizzando IBM MQ Explorer, creare e avviare un nuovo gestore code.
   a) Fare clic con il tasto destro del mouse su `Queue Managers` cartella> **Nuovo** > **Gestore code ...**. Immettere un nome gestore code> **Fine**.

      Creare un nome gestore code; ad esempio, `MQTTQMGR`.
2. Creare e avviare il servizio di telemetria (MQXR) e creare un canale di telemetria di esempio.
   a) Aprire la cartella `Queue Managers\`*`QmgrName`*`\Telemetry` .
   b) Fare clic su **Definisci configurazione di esempio ...** > **Fine**

      Lasciare la casella di spunta **Avvia MQTT Programma di utilità client** selezionata.
3. Creare una sottoscrizione per `MQTT Example` utilizzando il programma di utilità client MQTT .
   a) Fare clic su **Connetti**.

      La **Cronologia client** registra un evento `Connected` .
   b) Immettere `MQTT Example` nel campo **Sottoscrizione \ Argomento** > **Sottoscrivi**.

   La **Cronologia client** registra un evento `Subscribed` .
4. Creare `MQTTExampleTopic` in IBM MQ.
   a) Fare clic con il tasto destro del mouse sulla cartella `Queue Managers\`*`QmgrName`*`\Topics` in **MQ Explorer**> **Nuovo** > **Argomento**.
   b) Immettere `MQTTExampleTopic` come **Nome** > **Avanti**.
   c) Immettere `MQTT Example` come **Stringa argomento** > **Fine**.
   d) Fare clic su **OK** per chiudere la finestra di conferma.
5. Pubblicare `Hello World!` nell'argomento `MQTT Example` utilizzando IBM MQ Explorer.
   a) Fare clic sulla cartella `Queue Managers\`*`QmgrName`*`\Topics` in IBM MQ Explorer.
   b) Fare clic con il tasto destro del mouse su `MQTTExampleTopic` > **Verifica pubblicazione ...**
   c) Immettere `Hello World!` nel campo **Dati del messaggio** > **Pubblica messaggio** > Passa alla finestra Programma di utilità del client MQTT .

   La **Cronologia client** registra un evento `Received` .

Avvia attività con un servizio di telemetria in esecuzione (MQXR)

Creare un canale di telemetria e un argomento. Autorizzare l'utente ad utilizzare l'argomento e la coda di trasmissione della telemetria. Pubblicare un messaggio utilizzando IBM MQ Explorere sottoscriverlo con il programma di utilità client MQTT .

Prima di iniziare

In questa versione dell'attività, un gestore code, *`QmgrName`*, è definito e in esecuzione. Un servizio di telemetria (MQXR) è definito e in esecuzione. Il servizio di telemetria (MQXR) potrebbe essere stato creato manualmente oppure eseguendo la procedura guidata **Definisci configurazione di esempio** .

Informazioni su questa attività

In questa attività si configurerà un gestore code esistente per inviare una pubblicazione al programma di utilità del client MQTT .

Il passo “1” a pagina 127 dell'attività imposta la coda di trasmissione predefinita su `SYSTEM.MQTT.TRANSMIT.QUEUE`, che potrebbe interferire con le applicazioni in esecuzione su un gestore code esistente. È possibile, ma laborioso, configurare la telemetria e non utilizzare la coda di

trasmissione predefinita; eseguire la seguente attività: “Utilizzo di un alias del gestore code” a pagina 127.

Procedura

1. Impostare SYSTEM.MQTT.TRANSMIT.QUEUE come coda di trasmissione predefinita.
   a) Fare clic con il tastino destro del mouse su Queue Managers*QmgrName* folder > **Proprietà**
   b) Fare clic su **Comunicazione** nel navigatore.
   c) Fare clic su **Seleziona ...** > Seleziona SYSTEM.MQTT.TRANSMIT.QUEUE > **OK** > **OK**.
2. Creare un canale di telemetria MQTTExampleChannel per connettere il programma di utilità client MQTT a IBM MQe avviare il programma di utilità client MQTT .
   a) Fare clic con il tasto destro del mouse sulla cartella Queue Managers*QmgrName* \Telemetry\Channels in **MQ Explorer**> **Nuovo** > **Canale di telemetria ...**.
   b) Immettere MQTTExampleChannel nel campo **Nome canale** > **Avanti** > **Avanti**.
   c) Modificare l' **ID utente fisso** nel pannello di autorizzazione del client nell'ID utente che pubblicherà e sottoscrivi MQTTExample > **Avanti**.
   d) Lasciare selezionato **Avvia programma di utilità client** > **Fine**.
3. Creare una sottoscrizione per MQTT Example utilizzando il programma di utilità client MQTT .
   a) Fare clic su **Connetti**.

      La **Cronologia client** registra un evento Connected .
   b) Immettere MQTT Example nel campo **Sottoscrizione \ Argomento** > **Sottoscrivi**.

   La **Cronologia client** registra un evento Subscribed .
4. Creare MQTTExampleTopic in IBM MQ.
   a) Fare clic con il tasto destro del mouse sulla cartella Queue Managers*QmgrName*\Topics in **MQ Explorer**> **Nuovo** > **Argomento**.
   b) Immettere MQTTExampleTopic come **Nome** > **Avanti**.
   c) Immettere MQTT Example come **Stringa argomento** > **Fine**.
   d) Fare clic su **OK** per chiudere la finestra di conferma.
5. Se si desidera che un utente, non appartenente al gruppo mqm , pubblichi e sottoscriva l'argomento MQTTExample , effettuare le seguenti operazioni:
   a) Autorizzare l'utente a pubblicare e sottoscrivere l'argomento MQTTExampleTopic:

      ```
      setmqaut -m qMgrName -t topic -n MQTTExampleTopic -p User ID -all +pub +sub
      ```

   b) Autorizzare l'utente a inserire un messaggio in SYSTEM.MQTT.TRANSMIT.QUEUE:

      ```
      setmqaut -m qMgrName -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p User ID -all +put
      ```

6. Pubblicare Hello World! nell'argomento MQTT Example utilizzando IBM MQ Explorer.
   a) Fare clic sulla cartella Queue Managers*QmgrName*\Topics in IBM MQ Explorer.
   b) Fare clic con il tasto destro del mouse su MQTTExampleTopic > **Verifica pubblicazione ...**
   c) Immettere Hello World! nel campo **Dati del messaggio** > **Pubblica messaggio** > Passa alla finestra Programma di utilità del client MQTT .

   La **Cronologia client** registra un evento Received .

Utilizzo di un alias del gestore code

Pubblicare un messaggio nel programma di utilità del client MQTT utilizzando IBM MQ Explorer senza impostare la coda di trasmissione predefinita su SYSTEM.MQTT.TRANSMIT.QUEUE.

L'attività è una continuazione dell'attività precedente e utilizza un alias del gestore code per evitare di impostare la coda di trasmissione predefinita su SYSTEM.MQTT.TRANSMIT.QUEUE.

Prima di iniziare

Completare l'attività, "Avvia attività senza alcun servizio di telemetria (MQXR) ancora definito" a pagina 125 o l'attività, "Avvia attività con un servizio di telemetria in esecuzione (MQXR)" a pagina 126.

Informazioni su questa attività

Quando un MQTT client crea una sottoscrizione, IBM MQ invia la risposta utilizzando `ClientIdentifier`, come nome del gestore code remoto. In questa attività, utilizza `ClientIdentifier`, `MyClient`.

Se non esiste una coda di trasmissione o un alias del gestore code denominato `MyClient`, la risposta viene inserita nella coda di trasmissione predefinita. Impostando la coda di trasmissione predefinita su SYSTEM.MQTT.TRANSMIT.QUEUE, il client MQTT ottiene la risposta.

È possibile evitare di impostare la coda di trasmissione predefinita su SYSTEM.MQTT.TRANSMIT.QUEUE utilizzando gli alias del gestore code. È necessario impostare un alias del gestore code per ogni `ClientIdentifier`. Di solito, ci sono troppi client per rendere pratico l'utilizzo degli alias dei gestori code. Spesso `ClientIdentifier` è imprevedibile, rendendo impossibile configurare la telemetria in questo modo.

Tuttavia, in alcune circostanze potrebbe essere necessario configurare la coda di trasmissione predefinita su un valore diverso da SYSTEM.MQTT.TRANSMIT.QUEUE. I passaggi in Procedura configurano un alias del gestore code invece di impostare la coda di trasmissione predefinita su SYSTEM.MQTT.TRANSMIT.QUEUE.

Procedura

1. Rimuovere SYSTEM.MQTT.TRANSMIT.QUEUE come coda di trasmissione predefinita.
   a) Fare clic con il tastino destro del mouse su `Queue Managers\`*`QmgrName`* `folder` > **Proprietà**
   b) Fare clic su **Comunicazione** nel navigatore.
   c) Rimuovere SYSTEM.MQTT.TRANSMIT.QUEUE dal campo **Coda di trasmissione predefinita** > **OK**.
2. Verificare che non sia più possibile creare una sottoscrizione con l'utilità client MQTT :
   a) Fare clic su **Connetti**.

      La **Cronologia client** registra un evento `Connected` .
   b) Immettere `MQTT Example` nel campo **Sottoscrizione \ Argomento** > **Sottoscrivi**.

   La **Cronologia client** registra un `Subscribe failed` e un evento `Connection lost` .
3. Creare un alias del gestore code per `ClientIdentifier`, `MyClient`.
   a) Fare clic con il pulsante destro del mouse sulla cartella `Queue Managers\`*`QmgrName`*`\Queues` > **Nuovo** > **Definizione coda remota**.
   b) Denominare la definizione, `MyClient` > **Avanti**.
   c) Immettere `MyClient` nel campo **Gestore code remoto** .
   d) Immettere SYSTEM.MQTT.TRANSMIT.QUEUE nel campo **Coda di trasmissione** > **Fine**.
4. Connettere nuovamente il programma di utilità del client MQTT .
   a) Verificare che **Identificativo client** sia impostato su `MyClient`.
   b) **Connetti**

   La **Cronologia client** registra un evento `Connected` .
5. Creare una sottoscrizione per `MQTT Example` utilizzando il programma di utilità client MQTT .
   a) Fare clic su **Connetti**.

La **Cronologia client** registra un evento `Connected` .

b) Immettere `MQTT Example` nel campo **Sottoscrizione \ Argomento** > **Sottoscrivi**.

La **Cronologia client** registra un evento `Subscribed` .

6. Pubblicare `Hello World!` nell'argomento `MQTT Example` utilizzando IBM MQ Explorer.
   a) Fare clic sulla cartella `Queue Managers\`*`QmgrName`*`\Topics` in IBM MQ Explorer.
   b) Fare clic con il tasto destro del mouse su `MQTTExampleTopic` > **Verifica pubblicazione ...**
   c) Immettere `Hello World!` nel campo **Dati del messaggio** > **Pubblica messaggio** > Passa alla finestra Programma di utilità del client MQTT .

   La **Cronologia client** registra un evento `Received` .

Linux Windows AIX Invio di un messaggio a un client MQTT utilizzando IBM MQ Explorer

Inviare un messaggio al programma di utilità del client MQTT inserendo un messaggio in una coda IBM MQ utilizzando IBM MQ Explorer. L'attività mostra come configurare una definizione di coda remota per l'invio di messaggi direttamente a un client MQTT .

Prima di iniziare

Eseguire l'attività, "Pubblicazione di un messaggio nel programma di utilità del client MQTT da IBM MQ Explorer" a pagina 124. Lasciare connesso il programma di utilità del client MQTT .

Informazioni su questa attività

L'attività dimostra l'invio di un messaggio a un client MQTT utilizzando la coda piuttosto che la pubblicazione in un argomento. Non si crea una sottoscrizione nel client. Il passo "2" a pagina 129 dell'attività dimostra che la precedente sottoscrizione è stata eliminata.

Procedura

1. Eliminare eventuali sottoscrizioni esistenti disconnettendo e ricollegando il programma di utilità del client MQTT .

   La sottoscrizione viene eliminata perché, a meno che non si modifichino i valori predefiniti, il programma di utilità del client MQTT si connette con una sessione pulita; consultare Sessione pulita.

   Per rendere più semplice l'esecuzione dell'attività, immettere il proprio `ClientIdentifier`, piuttosto che utilizzare il `ClientIdentifier` generato dal programma di utilità client MQTT .

   a) Fare clic su **Disconnetti** per disconnettere il programma di utilità del client MQTT dal canale di telemetria.

      La **Cronologia client** registra un evento `Disconnected`

   b) Modificare l' **Identificativo client** in `MyClient`.

   c) Fare clic su **Connetti**.

   La **Cronologia client** registra un evento `Connected`

2. Controllare che l'utilità client MQTT non riceva più la pubblicazione per `MQTTExampleTopic`.
   a) Fare clic sulla cartella `Queue Managers\`*`QmgrName`*`\Topics` in IBM MQ Explorer.
   b) Fare clic con il tasto destro del mouse su `MQTTExampleTopic` > **Verifica pubblicazione ...**
   c) Immettere `Hello World!` nel campo **Dati del messaggio** > **Pubblica messaggio** > Passa alla finestra Programma di utilità del client MQTT .

   Nessun evento viene registrato nella **Cronologia client**.

3. Creare una definizione di coda remota per il client.

Impostare `ClientIdentifier`, `MyClient`, come nome del gestore code remoto nella definizione della coda remota. Utilizzare qualsiasi nome che si desidera come nome della coda remota. Il nome della coda remota viene inoltrato a un client MQTT come nome argomento.

a) Fare clic con il pulsante destro del mouse sulla cartella `Queue Managers\`*`QmgrName`*`\Queues` > **Nuovo** > **Definizione coda remota**.

b) Denominare la definizione, `MyClientRemoteQueue` > **Avanti**.

c) Immettere `MQTTExampleQueue` nel campo **Coda remota** .

d) Immettere `MyClient` nel campo **Gestore code remoto** .

e) Immettere `SYSTEM.MQTT.TRANSMIT.QUEUE` nel campo **Coda di trasmissione** > **Fine**.

4. Inserire un messaggio di test in `MyClientRemoteQueue`.

a) Fare clic con il tasto destro del mouse su **MyClientRemoteQueue** > **Inserisci messaggio di prova ...**

b) Immettere `Hello queue!` nel campo Dati messaggio> **Inserisci messaggio** > **Chiudi**

La **Cronologia client** registra un evento `Received` .

5. Rimuovere `SYSTEM.MQTT.TRANSMIT.QUEUE` come coda di trasmissione predefinita.

a) Fare clic con il tastino destro del mouse su `Queue Managers\`*`QmgrName`* `folder` > **Proprietà**

b) Fare clic su **Comunicazione** nel navigatore.

c) Rimuovere `SYSTEM.MQTT.TRANSMIT.QUEUE` dal campo **Coda di trasmissione predefinita** > **OK**.

6. Ripetere il passo "4" a pagina 130.

`MyClientRemoteQueue` è una definizione di coda remota che denomina esplicitamente la coda di trasmissione. non è necessario definire una coda di trasmissione predefinita per inviare un messaggio a `MyClient`.

Operazioni successive

Con la coda di trasmissione predefinita non più impostata su `SYSTEM.MQTT.TRANSMIT.QUEUE`, il programma di utilità client MQTT non è in grado di creare una nuova sottoscrizione a meno che non sia stato definito un alias del gestore code per `ClientIdentifier`, `MyClient`. Ripristinare la coda di trasmissione predefinita su `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Linux Windows AIX Pubblicazione di un messaggio in un client MQTT v3 specifico

Pubblicare un messaggio da un client MQTT v3 ad un altro, utilizzando `ClientIdentifier` come nome argomento e IBM MQ come broker di pubblicazione / sottoscrizione.

Prima di iniziare

Eseguire l'attività, "Pubblicazione di un messaggio nel programma di utilità del client MQTT da IBM MQ Explorer" a pagina 124. Lasciare connesso il programma di utilità del client MQTT .

Informazioni su questa attività

L'attività mostra due cose:

1. Sottoscrizione di un argomento in un client MQTT e ricezione di una pubblicazione da un altro client MQTT .
2. Impostazione delle sottoscrizioni "point - to - point" utilizzando `ClientIdentifier` come stringa argomento.

Procedura

1. Eliminare eventuali sottoscrizioni esistenti disconnettendo e ricollegando il programma di utilità del client MQTT .

   La sottoscrizione viene eliminata perché, a meno che non si modifichino i valori predefiniti, il programma di utilità del client MQTT si connette con una sessione pulita; consultare Sessione pulita.

   Per rendere più semplice l'esecuzione dell'attività, immettere il proprio `ClientIdentifier`, piuttosto che utilizzare il `ClientIdentifier` generato dal programma di utilità client MQTT .

   a) Fare clic su **Disconnetti** per disconnettere il programma di utilità del client MQTT dal canale di telemetria.

      La **Cronologia client** registra un evento `Disconnected`

   b) Modificare l' **Identificativo client** in `MyClient`.

   c) Fare clic su **Connetti**.

   La **Cronologia client** registra un evento `Connected`
2. Creare una sottoscrizione all'argomento, `MyClient`

   `MyClient` è `ClientIdentifier` di questo client.

   a) Immettere `MyClient` nel campo **Sottoscrizione \ Argomento** > **Sottoscrivi**.

   La **Cronologia client** registra un evento `Subscribed` .
3. Avviare un altro programma di utilità client MQTT .

   a) Aprire la cartella `Queue Managers\`*`QmgrName`*`\Telemetry\channels` .

   b) Fare clic con il tasto destro del mouse sul canale **PlainText** > **Esegui programma di utilità client MQTT ...**

   c) Fare clic su **Connetti**.

   La **Cronologia client** registra un evento `Connected`
4. Pubblicare `Hello MyClient!` nell'argomento `MyClient`.

   a) Copiare l'argomento della sottoscrizione, `MyClient`, dal programma di utilità client MQTT in esecuzione con `ClientIdentifier`, `MyClient`.

   b) Incollare `MyClient` nel campo **Pubblicazione / Argomento** di ciascuna istanza del programma di utilità client di MQTT .

   c) Immettere `Hello MyClient!` nel campo **Pubblicazione \ messaggio** .

   d) Fare clic su **Pubblica** in entrambe le istanze.

Risultati

La **Cronologia client** nel programma di utilità del client MQTT con `ClientIdentifier`, `MyClient`, registra due eventi **Ricevuti** e un evento **Pubblicato** . L'altra istanza del programma di utilità client MQTT registra un evento **Pubblicato** .

Se viene visualizzato un solo evento **Ricevuto** , verificare le seguenti possibili cause:

1. La coda di trasmissione predefinita per il gestore code è impostata su `SYSTEM.MQTT.TRANSMIT.QUEUE` ?
2. Sono stati creati alias del gestore code o definizioni di code remote che fanno riferimento a `MyClient` durante gli altri esercizi? Nel caso in cui si verifichi un problema di configurazione, eliminare le risorse che fanno riferimento a `MyClient`, come ad esempio gli alias di un gestore code o le code di trasmissione. Disconnettere le utilità client, arrestare e riavviare il servizio di telemetria (MQXR).

Linux Windows AIX Invio di un messaggio a un'applicazione IBM MQ da un client MQTT

Un'applicazione IBM MQ può ricevere un messaggio da un client MQTT v3 sottoscrivendo un argomento. Il client MQTT si connette a IBM MQ utilizzando un canale di telemetria e invia un messaggio all'applicazione IBM MQ pubblicando nello stesso argomento.

Eseguire l'attività “Pubblicazione di un messaggio in IBM MQ da un client MQTT” a pagina 132per informazioni su come inviare una pubblicazione da un client MQTT a una sottoscrizione definita in IBM MQ.

Se l'argomento è in cluster o distribuito utilizzando una gerarchia di pubblicazione / sottoscrizione, la sottoscrizione può trovarsi su un gestore code differente rispetto al gestore code a cui è connesso il client MQTT .

Linux Windows AIX Pubblicazione di un messaggio in IBM MQ da un client MQTT

Creare una sottoscrizione a un argomento utilizzando IBM MQ Explorer e pubblicarla nell'argomento utilizzando l'utilità del client MQTT .

Prima di iniziare

Eseguire l'attività, “Pubblicazione di un messaggio nel programma di utilità del client MQTT da IBM MQ Explorer” a pagina 124. Lasciare connesso il programma di utilità del client MQTT .

Informazioni su questa attività

L'attività illustra la pubblicazione di un messaggio con un cliente MQTT e la ricezione della pubblicazione utilizzando una sottoscrizione durevole non gestita creata utilizzando IBM MQ Explorer.

Procedura

1. Creare una sottoscrizione durevole alla stringa di argomenti `MQTT Example`.

   Effettuare le seguenti operazioni per creare la coda e la sottoscrizione utilizzando IBM MQ Explorer.

   a) Fare clic con il pulsante destro del mouse su `Queue Managers\`*`QmgrName`*`\Queues` in IBM MQ Explorer> **Nuovo** > **Coda locale ...**.

   b) Immettere `MQTTExampleQueue` come nome coda> **Fine**.

   c) Fare clic con il pulsante destro del mouse sulla cartella `Queue Managers\`*`QmgrName`*`\Subscriptions` in IBM MQ Explorer> **Nuovo** > **Sottoscrizione**.

   d) Immettere `MQTTExampleSubscription` come nome coda> **Avanti**.

   e) Fare clic su **Seleziona ...** > `MQTTExampleTopic` > **OK**.

   L'argomento è già stato creato, `MQTTExampleTopic` nel passo “4” a pagina 126 di “Pubblicazione di un messaggio nel programma di utilità del client MQTT da IBM MQ Explorer” a pagina 124.

   f) Immettere `MQTTExampleQueue` come nome destinazione> **Fine**.

2. Come passo facoltativo, impostare la coda per l'utilizzo da parte di un utente differente, senza l'autorità `mqm` .

   Se si sta configurando la configurazione per gli utenti con meno autorizzazioni di `mqm`, è necessario fornire l'autorizzazione `put` e `get` a `MQTTExampleQueue`. L'accesso all'argomento e alla coda di trasmissione è stato configurato in “Pubblicazione di un messaggio nel programma di utilità del client MQTT da IBM MQ Explorer” a pagina 124.

   a) Autorizzare un utente a inserire e richiamare la coda `MQTTExampleQueue`:

```
setmqaut -m qMgrName -t queue -n MQTTExampleQueue -p User ID -all +put +get
```

3. Pubblicare `Hello IBM MQ!` nell'argomento `MQTT Example` utilizzando l'utilità client MQTT .

   Se il programma di utilità del client MQTT non è stato ancora connesso, fare clic con il tasto destro del mouse sul canale **PlainText** > **Esegui programma di utilità del client MQTT ...** > **Connetti**.

   a) Immettere `MQTT Example` nel campo **Pubblicazione / Argomento** .

   b) Immettere `Hello IBM MQ!` nel campo **Pubblicazione \ Messaggio** > **Pubblica**.
4. Aprire la cartella `Queue Managers\`*`QmgrName`*`\Queues` e trovare `MQTTExampleQueue`.

   Il campo **Profondità coda corrente** è 1
5. Fare clic con il tasto destro del mouse su `MQTTExampleQueue` > **Sfoglia messaggi ...** ed esaminare la pubblicazione.

Linux Windows AIX MQTT applicazioni di pubblicazione/ sottoscrizione

Utilizzare la pubblicazione / sottoscrizione basata sull'argomento per scrivere applicazioni MQTT .

Quando il client MQTT è connesso, le pubblicazioni passano in entrambe le direzioni tra il client e il server. Le pubblicazioni vengono inviate dal client quando le informazioni vengono pubblicate sul client. Le pubblicazioni vengono ricevute sul client quando un messaggio viene pubblicato su un argomento che corrisponde a una sottoscrizione creata dal client.

Il broker di pubblicazione / sottoscrizione IBM MQ gestisce gli argomenti e le sottoscrizioni creati dai client MQTT . Gli argomenti creati dai clienti MQTT condividono lo stesso spazio argomenti creato dalle applicazioni IBM MQ .

Le pubblicazioni che corrispondono alla stringa argomento in una sottoscrizione client MQTT vengono collocate in `SYSTEM.MQTT.TRANSMIT.QUEUE` con il nome del gestore code remoto impostato su `ClientIdentifier` del client. Il servizio di telemetria (MQXR) inoltra le pubblicazioni al client che ha creato la sottoscrizione. Utilizza `ClientIdentifier`, che è stato impostato come nome del gestore code remoto per identificare il client.

Generalmente, `SYSTEM.MQTT.TRANSMIT.QUEUE` deve essere definito come coda di trasmissione predefinita. È possibile, ma oneroso, configurare MQTT in modo che non utilizzi la coda di trasmissione predefinita; consultare Configurazione dell'accodamento distribuito per inviare messaggi ai client MQTT.

Un client MQTT può creare una sessione persistente; consultare “MQTT sessioni stateless e stateful” a pagina 136. Le sottoscrizioni create in una sessione persistente sono durevoli. Le pubblicazioni che arrivano per un client con una sessione persistente vengono memorizzate in `SYSTEM.MQTT.TRANSMIT.QUEUE`e inoltrate al client quando si riconnette.

Un client MQTT può anche pubblicare e sottoscrivere pubblicazioni conservate; consultare Pubblicazioni conservate e client MQTT. Un sottoscrittore a un argomento di pubblicazione conservato riceve l'ultima pubblicazione per l'argomento. Il sottoscrittore riceve la pubblicazione conservata quando crea una sottoscrizione o quando si riconnette alla sessione precedente.

Linux Windows AIX Applicazioni di telemetria

Scrivere applicazioni di telemetria utilizzando flussi di messaggi IBM MQ o IBM Integration Bus .

Utilizzare JMS, MQI o altre interfacce di programmazione IBM MQ per programmare le applicazioni di telemetria in IBM MQ.

Il servizio di telemetria (MQXR) converte tra i messaggi MQTT v3 e i messaggi IBM MQ . Crea sottoscrizioni e pubblicazioni per conto dei client MQTT e inoltra le pubblicazioni ai client MQTT . Una pubblicazione è il payload di un messaggio MQTT v3 . Il payload comprende le intestazioni del messaggio e un array di byte in formato `jms-bytes` . Il server di telemetria associa le intestazioni tra un messaggio MQTT v3 e uno IBM MQ ; consultare “Integrazione di MQ Telemetry con i gestori code” a pagina 134.

Utilizzare i nodi Publication, MQInput e JMSInput per inviare e ricevere pubblicazioni tra client IBM Integration Bus e MQTT .

Utilizzando i flussi di messaggi è possibile integrare la telemetria con i siti Web utilizzando HTTP e con altre applicazioni utilizzando IBM MQ e WebSphere Adapters.

Linux Windows AIX Integrazione di MQ Telemetry con i gestori code

Il client MQTT è integrato con IBM MQ come applicazione di pubblicazione / sottoscrizione. Può pubblicare o sottoscrivere argomenti in IBM MQ, creare nuovi argomenti o utilizzare argomenti esistenti. Riceve pubblicazioni da IBM MQ come risultato di client MQTT , incluso se stesso, o altre applicazioni IBM MQ che pubblicano gli argomenti delle relative sottoscrizioni. Le regole vengono applicate per decidere gli attributi di una pubblicazione.

Molti degli attributi associati ad argomenti, pubblicazioni, sottoscrizioni e messaggi forniti da IBM MQnon sono supportati. "Client MQTT per IBM MQ broker di pubblicazione / sottoscrizione" a pagina 134 e "IBM MQ su un client MQTT" a pagina 135 descrivono come vengono impostati gli attributi delle pubblicazioni. Le impostazioni dipendono dal fatto che la pubblicazione venga eseguita o meno dal broker di pubblicazione / sottoscrizione IBM MQ .

In IBM MQ , gli argomenti di pubblicazione / sottoscrizione sono associati agli oggetti argomento di gestione. Gli argomenti creati dai client MQTT non sono diversi. Quando un client MQTT crea una stringa di argomento per una pubblicazione, il broker di pubblicazione / sottoscrizione IBM MQ la associa a un oggetto argomento di amministrazione. Il broker associa la stringa argomento nella pubblicazione all'oggetto argomento di gestione principale più vicino. L'associazione è uguale a quella delle applicazioni IBM MQ . Se non è presente alcun argomento creato dall'utente, l'argomento di pubblicazione viene associato a `SYSTEM.BASE.TOPIC`. Gli attributi applicati alla pubblicazione derivano dall'oggetto argomento.

Quando un'applicazione IBM MQ o un amministratore crea una sottoscrizione, la sottoscrizione viene denominata. Elencare le sottoscrizioni utilizzando IBM MQ Explorero utilizzando i comandi **runmqsc** o PCF. Tutte le sottoscrizioni dei client MQTT vengono denominate. Viene fornito un nome del formato: *`ClientIdentifier:Topic name`*

Client MQTT per IBM MQ broker di pubblicazione / sottoscrizione

Un client MQTT ha inviato una pubblicazione a IBM MQ. Il servizio di telemetria (MQXR) converte la pubblicazione in messaggio IBM MQ . Il messaggio IBM MQ contiene tre parti:

1. MQMD
2. RFH2
3. Messaggio

Le proprietà MQMD sono impostate sui valori predefiniti, tranne dove indicato in Tabella 9 a pagina 134.

Tabella 9. Impostazioni per proprietà MQMD

campo MQMD	Tipo	Valore
Format	MQCHAR8	MQFMT_RF_HEADER_2

UserIdentifier	MQCHAR12	Impostare su uno tra:
		`MqttClient.ClientIdentifier`
		`MqttConnectOptions.UserName`
		Un ID utente impostato dall'amministratore IBM MQ per il canale di telemetria.

Priority	MQLONG	MQPRI_PRIORITY_AS_Q_DEF (Impostazione predefinita per IBM MQ, che è diverso da JMS che ha un valore predefinito di 4.)

Tabella 9. Impostazioni per proprietà MQMD (Continua)		
campo MQMD	**Tipo**	**Valore**
Persistence	MQLONG	QoS=0→MQPER_NOT_PERSISTENT QoS=1→MQPER_PERSISTENT QoS=2→MQPER_PERSISTENT

L'intestazione RFH2 non contiene una cartella <msd> per definire il tipo di messaggio JMS . Il servizio di telemetria (MQXR) crea il messaggio IBM MQ come messaggio predefinito JMS . Il tipo di messaggio JMS predefinito è un jms-bytes . Un'applicazione può accedere ad ulteriori informazioni di intestazione come proprietà del messaggio; consultare Proprietà del messaggio.

I valori RFH2 sono impostati come mostrato in Tabella 10 a pagina 135. La proprietà Formato è impostata nell'intestazione fissa RFH2 e gli altri valori sono impostati nelle cartelle di RFH2 .

Tabella 10. Impostazioni per proprietà RFH2		
proprietà RFH2	**Tipo / Cartella**	**Intestazione**
Format	MQCHAR8	MQFMT_NONE
ClientIdentifier	mqtt/clientId	Copiare MqttClient.ClientIdentifier con una lunghezza di 1 ... 23 byte.
QoS	mqtt/qos	Copiare QoS dal messaggio MQTT in entrata.
ID messaggio	mqtt/msgid	Copiare ID messaggio dal messaggio MQTT in entrata, se QoS è 1 o 2.
MQIsRetained	mqps/Ret	Impostare se la pubblicazione MQTT originale è stata inviata con la proprietà RETAIN impostata e il messaggio viene ricevuto come pubblicazione conservata.
MQTopicString	mqps/Top	L'argomento in cui è pubblicato il messaggio MQTT .

Il payload in una pubblicazione MQTT viene associato al contenuto di un messaggio IBM MQ :

Tabella 11. Modalità di associazione del payload in una pubblicazione MQTT al contenuto del messaggio IBM MQ		
Contenuto messaggio	**Tipo**	**Contenuto del messaggio IBM MQ**
Memorizza nel buffer	MQBYTE *n*	Copia dei byte dal messaggio MQTT in entrata. La lunghezza può essere zero.

IBM MQ su un client MQTT

Un client ha sottoscritto un argomento di pubblicazione. Un'applicazione IBM MQ è stata pubblicata nell'argomento, determinando l'invio di una pubblicazione al sottoscrittore MQTT da parte del broker di pubblicazione / sottoscrizione IBM MQ . In alternativa, un'applicazione di IBM MQ ha inviato un messaggio non richiesto direttamente a un client MQTT . Tabella 12 a pagina 136 descrive il modo in cui le intestazioni del messaggio fisso sono impostate nel messaggio inviato al client MQTT . Tutti gli altri dati nell'intestazione del messaggio IBM MQ , o qualsiasi altra intestazione, vengono eliminati. I dati del messaggio nel messaggio IBM MQ vengono inviati come payload del messaggio nel messaggio MQTT , senza alcuna modifica. Il messaggio MQTT viene inviato al client MQTT dal servizio di telemetria (MQXR).

Tabella 12. Modalità di impostazione delle intestazioni di messaggi fissi in un messaggio IBM MQ inviato al client MQTT

MQTT campo	Tipo	Valore
DUP	booleano	Impostare se QoS = 1 o 2e il messaggio è stato inviato a questo client in una trasmissione precedente e il messaggio non è stato riconosciuto dopo un periodo di tempo.
QoS	int	Il modo in cui viene impostato il valore di QoS in una pubblicazione in uscita dal broker di pubblicazione / sottoscrizione in IBM MQ dipende dalla pubblicazione in entrata. Dipende se la pubblicazione in entrata è stata inviata da un client MQTT o da un'applicazione IBM MQ . **MQTT** Valore inferiore di QoS nella pubblicazione in entrata e in QoS richiesto dal sottoscrittore. **IBM MQ** Valore inferiore del QoS derivato dalla pubblicazione in entrata: MQPER_NOT_PERSISTENT→QoS=0 MQPER_PERSISTENT→QoS=2 e il QoS richiesto dal sottoscrittore. Se il messaggio viene inviato al client senza una sottoscrizione, per impostazione predefinita QoS è impostato su 2. Un client può modificare questo valore sottoscrivendo DEFAULT.QoS con un QoSdifferente.
RETAIN	booleano	Impostare se la pubblicazione in entrata ha la proprietà conservata impostata.

Tabella 13 a pagina 136 descrive il modo in cui sono impostate le intestazioni del messaggio variabile nel messaggio MQTT inviato al client MQTT .

Tabella 13. Modalità di impostazione delle proprietà dell'intestazione della variabile MQTT in un messaggio MQTT inviato al client MQTT

MQTT campo	Tipo	Valore
Topic name	Stringa	La stringa di argomenti con cui è stato pubblicato il messaggio.
Message ID	Stringa	Gli ultimi 2 byte di MQMD.MsgId proprietà della pubblicazione quando viene inserita in SYSTEM.MQTT.TRANSMIT.QUEUE.
Payload	byte[]	Copia diretta dei byte dalla pubblicazione in entrata al broker di pubblicazione / sottoscrizione. La lunghezza può essere zero.

Linux Windows AIX

MQTT sessioni stateless e stateful

I client MQTT possono creare una sessione con stato con il gestore code. Quando un client MQTT con stato si disconnette, il gestore code conserva le sottoscrizioni create dal client e i messaggi in corso. Quando il client si riconnette, risolve il messaggio incompleto. Invia tutti i messaggi che sono in coda per la consegna e riceve tutti i messaggi pubblicati per le sue sottoscrizioni mentre è stato disconnesso.

Quando un client MQTT si connette a un canale di telemetria, avvia una nuova sessione o riprende una vecchia sessione. Una nuova sessione non ha messaggi in sospeso che non sono stati riconosciuti,

nessuna sottoscrizione e nessuna pubblicazione in attesa di consegna. Quando un client si connette, specifica se iniziare con una sessione di ripulitura o riprendere una sessione esistente; consultare Sessione di ripulitura.

Se il client riprende una sessione esistente, continua come se la connessione non fosse stata interrotta. Le pubblicazioni in attesa di consegna vengono inviate al client e tutti i trasferimenti di messaggi di cui non è stato eseguito il commit vengono completati. Quando un client in una sessione persistente si disconnette dal servizio di telemetria (MQXR), tutte le sottoscrizioni create dal client rimangono. Le pubblicazioni per le sottoscrizioni vengono inviate al client quando si riconnette. Se si riconnette senza riprendere la vecchia sessione, le pubblicazioni vengono eliminate dal servizio di telemetria (MQXR).

Le informazioni sullo stato della sessione vengono salvate dal gestore code nella coda `SYSTEM.MQTT.PERSISTENT.STATE` .

L'amministratore IBM MQ può disconnettere ed eliminare una sessione.

Linux Windows AIX Quando un client MQTT non è connesso

Quando un client non è connesso, il gestore code può continuare a ricevere pubblicazioni per suo conto. Vengono inoltrati al client quando si riconnette. Un client può creare un "Ultimo testamento", che il gestore code pubblica per conto del client, se il client si disconnette inaspettatamente.

Se si desidera ricevere una notifica quando il client si disconnette inaspettatamente, è possibile registrare una pubblicazione Ultimo testamento e testamento; consultare Pubblicazione Ultimo testamento e testamento. Viene inviato dal servizio di telemetria (MQXR), se rileva che la connessione al client si è interrotta senza che il client lo richieda.

Un client può pubblicare una pubblicazione conservata in qualsiasi momento; consultare Pubblicazioni conservate e client MQTT. Una nuova sottoscrizione a un argomento può richiedere l'invio di qualsiasi pubblicazione conservata associata all'argomento. Se si crea l'ultimo testamento come pubblicazione conservata, è possibile utilizzarlo per monitorare lo stato di un client.

Ad esempio, il client pubblica una pubblicazione conservata, quando si collega, pubblicizzando la relativa disponibilità. Allo stesso tempo, crea una pubblicazione conservata di ultima volontà e testamento che annuncia la sua indisponibilità. Inoltre, prima di effettuare una disconnessione pianificata, pubblica la sua indisponibilità come pubblicazione conservata. Per scoprire se il client è disponibile, è necessario sottoscrivere l'argomento della pubblicazione conservata. Riceverai sempre una delle tre pubblicazioni.

Se il client deve ricevere i messaggi pubblicati quando è disconnesso, riconnettere il client alla sessione precedente; consultare “MQTT sessioni stateless e stateful” a pagina 136. Le relative sottoscrizioni sono attive fino a quando non vengono eliminate o fino a quando il client non crea una sessione pulita.

Linux Windows AIX Accoppiamento debole tra client di MQTT e applicazioni IBM MQ

Il flusso delle pubblicazioni tra client MQTT e applicazioni IBM MQ è accoppiato in modo flessibile. Le pubblicazioni potrebbero provenire da un client MQTT o da un'applicazione IBM MQ e non in un ordine impostato. Gli editori e i sottoscrittori sono debolmente accoppiati. Interagiscono tra loro indirettamente tramite pubblicazioni e sottoscrizioni. È anche possibile inviare messaggi direttamente a un client MQTT da un'applicazione IBM MQ .

I client MQTT e le applicazioni IBM MQ sono accoppiati in due modi:

1. Gli autori (publisher) e i sottoscrittori (subscriber) sono associati liberamente da una pubblicazione e da una sottoscrizione a un argomento. Gli editori e i sottoscrittori normalmente non sono a conoscenza dell'indirizzo o dell'identità dell'altra fonte di una pubblicazione o di un abbonamento.
2. I client MQTT pubblicano, sottoscrivono, ricevono pubblicazioni ed elaborano riconoscimenti di consegna su thread separati.

Un'applicazione client MQTT non attende la consegna di una pubblicazione. L'applicazione passa un messaggio al client MQTT , quindi l'applicazione continua sul proprio thread. Un token di consegna viene utilizzato per sincronizzare l'applicazione con la consegna di una pubblicazione; vedi Token di consegna.

Dopo aver passato un messaggio al client MQTT , l'applicazione ha la possibilità di attendere il token di consegna. Invece di attendere, il client può fornire un metodo di callback che viene richiamato quando la pubblicazione viene consegnata a IBM MQ. Può anche ignorare il token di consegna.

A seconda della qualità del servizio associata al messaggio, il token di consegna viene restituito immediatamente al metodo di callback o, eventualmente, dopo un periodo di tempo considerevole. Il token di consegna potrebbe essere restituito anche dopo che il client si è scollegato e riconnesso. Se la QoS (quality of service) è *fire and forget*, il token di distribuzione viene restituito immediatamente. Negli altri due casi, il token di consegna viene restituito solo quando il client riceve il riconoscimento che la pubblicazione è stata inviata ai sottoscrittori.

Le pubblicazioni inviate a un client MQTT come risultato di una sottoscrizione client vengono consegnate al metodo di callback `messageArrived` . `messageArrived` viene eseguito su un thread differente rispetto all'applicazione principale.

Invio di messaggi direttamente a un client MQTT

È possibile inviare un messaggio a uno specifico client MQTT in due modi.

1. Un'applicazione IBM MQ può inviare un messaggio direttamente a un client MQTT senza una sottoscrizione; consultare Invio di un messaggio direttamente a un client.
2. Un approccio alternativo consiste nell'utilizzare la convenzione di denominazione `ClientIdentifier` . Fare in modo che tutti i sottoscrittori MQTT creino sottoscrizioni utilizzando il proprio `ClientIdentifier` univoco come argomento. Pubblicare in *ClientIdentifier*. La pubblicazione viene inviata al client che ha sottoscritto l'argomento *ClientIdentifier*. Utilizzando questa tecnica, è possibile inviare una pubblicazione a un particolare sottoscrittore MQTT .

Linux Windows AIX Sicurezza di MQ Telemetry

Proteggere i dispositivi di telemetria può essere importante, in quanto è probabile che siano portatili e utilizzati in luoghi che non possono essere controllati con attenzione. Puoi utilizzare la VPN per proteggere la connessione dal dispositivo MQTT al servizio di telemetria (MQXR). MQ Telemetry fornisce altri due meccanismi di sicurezza, TLS e JAAS.

TLS viene utilizzato principalmente per codificare le comunicazioni tra il dispositivo e il canale di telemetria e per autenticare il dispositivo che si connette al server corretto; consultare Autenticazione del canale di telemetria utilizzando TLS. Puoi inoltre utilizzare TLS per controllare che la periferica client sia autorizzata a connettersi al server; vedi Autenticazione clientMQTT tramite TLS.

JAAS viene utilizzato principalmente per controllare che l'utente della periferica sia autorizzato a utilizzare un'applicazione server; consultare MQTT autenticazione client utilizzando una parola d'ordine. JAAS può essere utilizzato con LDAP per controllare una parola d'ordine utilizzando una directory SSO (single sign - on).

TLS e JAAS possono essere utilizzati insieme per fornire l'autenticazione a due fattori. È possibile limitare le cifrature utilizzate da TLS alle cifrature che soddisfano standard FIPS.

Con almeno decine di migliaia di utenti, non è sempre pratico fornire profili di sicurezza individuali. Inoltre, non è sempre pratico utilizzare i profili per autorizzare i singoli utenti ad accedere agli oggetti IBM MQ . Raggruppare invece gli utenti in classi per autorizzare la pubblicazione e la sottoscrizione agli argomenti e inviare pubblicazioni ai client.

Configurare ciascun canale di telemetria per associare i client agli ID utente client comuni. Utilizzare un ID utente comune per ogni client che si connette su un canale specifico; consultare Identità e autorizzazione clientMQTT.

L'autorizzazione di gruppi di utenti non compromette l'autenticazione di ogni individuo. Ogni singolo utente può essere autenticato, sul client o sul server, con i relativi `Nome utente` e `Password`e quindi autorizzato sul server utilizzando un ID utente comune.

Linux Windows AIX MQ Telemetry globalizzazione

Il payload del messaggio nel protocollo MQTT v3 è codificato come array di byte. In genere, le applicazioni che gestiscono il testo creano il payload del messaggio in UTF-8. Il canale di telemetria descrive il payload del messaggio come UTF-8, ma non esegue alcuna conversione di codepage. La stringa dell'argomento di pubblicazione deve essere UTF-8.

L'applicazione è responsabile della conversione dei dati alfabetici nella codepage corretta e dei dati numerici nella codifica numerica corretta.

Il client MQTT Java ha un metodo `MqttMessage.toString` conveniente. Il metodo considera il payload del messaggio come codificato nella serie di caratteri predefinita della piattaforma locale, generalmente UTF-8. Converte il payload in una stringa Java . Java dispone di un metodo String, `getBytes` che converte una stringa in una schiera di byte codificata utilizzando la serie di caratteri predefinita della piattaforma locale. Due programmi MQTT Java che scambiano testo nel payload del messaggio, tra piattaforme con la stessa serie di caratteri predefinita, lo fanno in modo semplice ed efficiente in UTF-8.

Se la serie di caratteri predefinita di una delle piattaforme non è UTF-8, le applicazioni devono stabilire una convenzione per lo scambio di messaggi. Ad esempio, il publisher specifica la conversione da una stringa a UTF-8 utilizzando il metodo `getBytes("UTF8")` . Per ricevere il testo di un messaggio, l'utente assume che il messaggio sia codificato nella serie di caratteri UTF-8 .

Il servizio di telemetria (MQXR) descrive la codifica di tutte le pubblicazioni in entrata dai messaggi dei client MQTT come UTF-8. Imposta `MQMD.CodedCharSetId` su UTF-8e `RFH2.CodedCharSetId` su `MQCCSI_INHERIT` ; consultare “Integrazione di MQ Telemetry con i gestori code” a pagina 134.
Il formato della pubblicazione è impostato su `MQFMT_NONE`, quindi nessuna conversione può essere eseguita dai canali o da `MQGET`.

Linux Windows AIX Prestazioni e scalabilità di MQ Telemetry

Considerare i seguenti fattori quando si gestiscono un numero elevato di client e si migliora la scalabilità di MQ Telemetry.

Pianificazione della capacità

Per informazioni sui report delle prestazioni per MQ Telemetry, consultare MQ Documenti delle prestazioni.

Connessioni

I costi relativi alle connessioni includono:

- Il costo dell'impostazione di una connessione in termini di tempo e utilizzo del processore.
- Costi di rete.
- Memoria utilizzata quando si mantiene una connessione aperta ma non la si utilizza.

C'è un carico aggiuntivo che si verifica quando i client rimangono connessi. Se una connessione viene mantenuta aperta, i flussi TCP/IP e i messaggi MQTT utilizzano la rete per controllare che la connessione sia ancora presente. Inoltre, la memoria viene utilizzata nel server per ogni connessione client mantenuta aperta.

Se si stanno inviando messaggi più di uno al minuto, mantenere la connessione aperta per evitare il costo di avviare una nuova connessione. Se si inviano messaggi meno di uno ogni 10-15 minuti, si consiglia di eliminare la connessione per evitare il costo di mantenerla aperta. Potresti voler mantenere una connessione TLS aperta, ma inattiva, per periodi più lunghi perché è più costosa da configurare.

Inoltre, considerare la capacità del client. Se sul client è presente una funzione di memorizzazione e inoltro, è possibile eseguire il batch dei messaggi ed eliminare la connessione tra l'invio dei batch. Tuttavia, se il client è disconnesso, non è possibile per il client ricevere un messaggio dal server. Pertanto, lo scopo della tua domanda ha un impatto sulla decisione.

Se il sistema dispone di un client che invia molti messaggi, ad esempio i trasferimenti file, non attendere una risposta del server per messaggio. Invece, inviare tutti i messaggi e verificare alla fine che siano stati tutti ricevuti. In alternativa, utilizzare Quality of Service (QoS).

È possibile variare il QoS in base al messaggio, consegnando messaggi non importanti utilizzando QoS 0 e messaggi importanti utilizzando un QoS di 2. La velocità di trasmissione dei messaggi può essere circa il doppio di quella con un QoS pari a 0 rispetto a QoS pari a 2.

Convenzioni di denominazione

Se si sta progettando l'applicazione per molti client, implementare una convenzione di denominazione efficace. Per associare ciascun client al `ClientIdentifier`corretto, rendere `ClientIdentifier` significativo. Una buona convenzione di denominazione rende più semplice per l'amministratore capire quali client sono in esecuzione. Una convenzione di denominazione aiuta l'amministratore a filtrare un lungo elenco di client in IBM MQ Explorer e aiuta nella determinazione dei problemi; consultare Identificatore client.

Velocità di trasmissione

La lunghezza dei nomi argomento influisce sul numero di byte che passano attraverso la rete. Durante la pubblicazione o la sottoscrizione, il numero di byte in un messaggio potrebbe essere importante. Pertanto, limitare il numero di caratteri in un nome argomento. Quando un MQTT client effettua la sottoscrizione per un argomento IBM MQ gli fornisce un nome del formato:

```
ClientIdentifier: TopicName
```

Per visualizzare tutte le sottoscrizioni per un client MQTT , è possibile utilizzare il comando IBM MQ MQSC **DISPLAY** :

```
DISPLAY SUB(' ClientID1:*')
```

Definizione delle risorse in IBM MQ per l'utilizzo da parte dei client MQTT

Un client MQTT si connette a un gestore code remoto IBM MQ . Esistono due metodi di base per un'applicazione IBM MQ per inviare i messaggi a un client MQTT : impostare la coda di trasmissione predefinita su `SYSTEM.MQTT.TRANSMIT.QUEUE` o utilizzare gli alias del gestore code. Definire la coda di trasmissione predefinita di un gestore code, se è presente un numero elevato di client MQTT . L'uso dell'impostazione della coda di trasmissione predefinita semplifica l'amministrazione; consultare Configurazione dell'accodamento distribuito per inviare messaggi ai client MQTT ..

Migliorare la scalabilità evitando le sottoscrizioni.

Quando un client MQTT V3 sottoscrive un argomento, viene creata una sottoscrizione dal servizio di telemetria (MQXR) in IBM MQ. La sottoscrizione instrada le pubblicazioni per il client su `SYSTEM.MQTT.TRANSMIT.QUEUE`. Il nome del gestore code remoto nell'intestazione di trasmissione di ogni pubblicazione è impostato su `ClientIdentifier` del client MQTT che ha eseguito la sottoscrizione. Se vi sono molti client, ognuno dei quali effettua le proprie sottoscrizioni, ciò comporta la gestione di molte sottoscrizioni proxy in tutta la gerarchia o il cluster di pubblicazione / sottoscrizione IBM MQ . Per informazioni su come non utilizzare la pubblicazione / sottoscrizione, ma utilizzare invece una soluzione basata punto a punto, consultare Invio di un messaggio direttamente a un client.

Gestione di un gran numero di client

Per supportare molti client connessi simultaneamente, aumentare la memoria disponibile per il servizio di telemetria (MQXR) impostando i parametri JVM **-Xms** e **-Xmx**. Eseguire queste operazioni:

1. Individuare il file `java.properties` nella directory di configurazione del servizio di telemetria; consultare Directory di configurazione del servizio di telemetria (MQXR) su Windows o Directory di configurazione del servizio di telemetria su Linux .
2. Seguire le istruzioni nel file; un heap di 1 GB è sufficiente per 50.000 client connessi contemporaneamente.

```
# Heap sizing options - uncomment the following lines to set the heap to 1G
#-Xmx1024m
#-Xms1024m
```

3. Aggiungere altri argomenti della riga comandi da passare alla JVM che esegue il servizio di telemetria (MQXR) nel file `java.properties` ; consultare Passaggio dei parametri JVM al servizio di telemetria (MQXR).

Per aumentare il numero di descrittori di file aperti su Linux, aggiungere le seguenti righe a `/etc/security/limits.conf`/ed eseguire nuovamente l'accesso.

```
@mqm soft nofile 65000
@mqm hard nofile 65000
```

Ogni socket richiede un descrittore di file. Il servizio di telemetria richiede alcuni descrittori di file aggiuntivi, quindi questo numero deve essere maggiore del numero di socket aperti richiesti.

Il gestore code utilizza un handle di oggetto per ogni sottoscrizione non durevole. Per supportare molte sottoscrizioni attive e non durevoli, aumentare il numero massimo di handle attivi nel gestore code; ad esempio:

```
echo ALTER QMGR MAXHANDS(999999999) | runmqsc qMgrName
```

Figura 48. Modifica numero massimo di handle su Windows

```
echo "ALTER QMGR MAXHANDS(999999999)" | runmqsc qMgrName
```

Figura 49. Modifica numero massimo di handle su Linux

Altre considerazioni

Quando si pianificano i requisiti di sistema, considerare il tempo impiegato per riavviare il sistema. Il tempo di inattività pianificato potrebbe avere implicazioni per il numero di messaggi in coda, in attesa di essere elaborati. Configurare il sistema in modo che i messaggi possano essere elaborati correttamente in un tempo accettabile. Esaminare l'archiviazione disco, la memoria e la potenza di elaborazione. Con alcune applicazioni client, è possibile eliminare i messaggi quando il client si riconnette. Per eliminare i messaggi, impostare `CleanSession` nei parametri di collegamento client; consultare Sessione pulita. In alternativa, pubblica e sottoscrivi utilizzando il QoS (Quality of Service) più efficace, `0`, in un client MQTT ; vedi QoS (Quality of service). Utilizzare messaggi non `persistenti` quando si inviano messaggi da IBM MQ. I messaggi con queste QoS (quality of service) non vengono ripristinati al riavvio del sistema o della connessione.

Linux Windows AIX Dispositivi supportati da MQ Telemetry

I clienti MQTT possono essere eseguiti su una vasta gamma di dispositivi, da sensori e attuatori, a dispositivi portatili e sistemi di veicoli.

I client MQTT sono di piccole dimensioni e vengono eseguiti su dispositivi con poca memoria e bassa potenza di elaborazione. Il MQTT protocol è affidabile e ha intestazioni piccole, che si adattano alle reti limitate da una larghezza di banda bassa, costi elevati e disponibilità intermittente.

MQ Telemetry comunica con i dispositivi di telemetria tramite applicazioni client MQTT . Queste applicazioni utilizzano le seguenti risorse, che implementano il protocollo MQTT v3 :

- Le seguenti librerie client:
  - *MQTT client for Java*, utilizzato per la creazione di applicazioni native per (ad esempio) dispositivi Android, OS X, Linux o Windows . Le applicazioni che utilizzano questa libreria client possono essere eseguite su tutte le varianti di Java dal più piccolo CLDC (Connected Limited Device Configuration) / MIDP (Mobile Information Device Profile) al CDC (Connected Device Configuration) /Foundation, J2SE (Java Platform, Standard Edition) e J2EE (Java Platform, Enterprise Edition). È supportata anche la libreria di classi personalizzata IBM jclRM . La piattaforma Java ME viene generalmente utilizzata su dispositivi di piccole dimensioni, come attuatori, sensori, telefoni cellulari e altri dispositivi incorporati. La piattaforma Java SE è generalmente installata su dispositivi integrati di fascia superiore, come computer desktop e server.
  - *MQTT client for C*, utilizzato per la creazione di applicazioni native per (ad esempio) periferiche iOS, OS X, Linux o Windows . Questa libreria del client fornisce un'implementazione di riferimento C insieme al client nativo precostruito per sistemi Windows e Linux . L'implementazione del riferimento C consente a MQTT di essere portato su una vasta gamma di dispositivi e piattaforme. Alcuni sistemi Windows su Intel, inclusi Windows 7, RedHat, Ubuntue alcuni sistemi Linux su piattaforme ARM come Eurotech Viper implementano versioni di Linux che eseguono il client C, ma IBM non fornisce il supporto di servizio per le piattaforme. È necessario riprodurre i problemi con il client su una piattaforma supportata se si intende chiamare il centro di supporto IBM .
  - *MQTT client for Java*, utilizzato per la creazione di applicazioni Web basate su browser.

Le librerie client MQTT sono disponibili gratuitamente da Eclipse Paho e MQTT.org. Vedere IBM MQ Telemetry Transport programmi di esempio.

Sicurezza in IBM MQ

In IBM MQ, esistono diversi metodi per fornire la sicurezza: l'interfaccia del servizio di autorizzazione, le uscite del canale scritte dall'utente o di terze parti, la sicurezza del canale utilizzando TLS (Transport Layer Security), i record di autenticazione del canale e la sicurezza dei messaggi.

Interfaccia servizio di autorizzazione

L'autorizzazione per l'utilizzo di chiamate MQI, comandi e accesso agli oggetti è fornita da **gestore autorizzazioni oggetto** (OAM), che per impostazione predefinita è abilitata. L'accesso alle entità IBM MQ è controllato tramite i gruppi di utenti IBM MQ e OAM. Gli amministratori possono utilizzare una CLI (command - line interface) per concedere o revocare le autorizzazioni come richiesto.

Per ulteriori informazioni sulla creazione dei componenti del servizio di autorizzazione, consultare Impostazione della sicurezza sui sistemi AIX, Linux, and Windows.

Uscite canale di terze parti o scritte dall'utente

I canali possono utilizzare uscite di canali scritte dall'utente o di terze parti. Per ulteriori informazioni, consultare Channel - exit programs for messaging channels.

Sicurezza del canale utilizzando TLS

Il protocollo TLS (Transport Layer Security) fornisce la sicurezza del canale standard del settore, con protezione da intercettazione, manomissione e impersonificazione.

TLS utilizza tecniche di chiave pubblica e simmetriche per fornire riservatezza e integrità dei messaggi e autenticazione reciproca.

Per una revisione completa della sicurezza in IBM MQ incluse informazioni dettagliate su TLS, vedi Sicurezza. Per una panoramica di TLS, inclusi i puntatori ai comandi descritti in questa sezione, consultare Protocolli di sicurezza crittografici: TLS.

Record di autenticazione di canale

Utilizzare i record di autenticazione di canale per esercitare un controllo preciso sull'accesso concesso ai sistemi di collegamento a livello di canale. Per ulteriori informazioni, consultare Record di autenticazione di canale.

Sicurezza messaggi

Utilizzare Advanced Message Security, che è un componente con licenza e installato separatamente di IBM MQ, per fornire una protezione crittografica ai messaggi inviati e ricevuti utilizzando IBM MQ. Vedere Advanced Message Security.

Attività correlate

Protezione

Pianificazione dei requisiti di sicurezza

Supporto TLS del client gestito IBM MQ.NET

Il client IBM MQ.NET completamente gestito fornisce il supporto TLS (Transport Layer Security) basato sul kit Microsoft.NET SSLStreams. È diverso dagli altri client IBM MQ , che si basano su IBM Global Security Kit (GSKit).

È possibile sviluppare applicazioni IBM MQ.NET da eseguire in modalità gestita o non gestita.

- In modalità gestita, le applicazioni .NET funzionano all'interno di .NET CLR (Common Language Runtime) senza alcun richiamo tra piattaforme, come il richiamo di C MQI.
- In modalità non gestita, C MQI viene richiamata per le operazioni MQI sottostanti. In pratica, l'interfaccia in modalità non gestita comprende le classi wrapper .NET in cima alla C MQI.

Il client IBM MQ.NET gestito utilizza le librerie Microsoft.NET Framework per implementare i protocolli socket sicuri TLS. Il sistema.NET.Security.SSLStream da Microsoft viene utilizzata per implementare la sicurezza (TLS) in IBM MQ.NET.

La modalità client IBM MQ.NET non gestita supporta già la funzione TLS, che è basata su C MQI (e GSKit). In altre parole, le operazioni TLS sono gestite da C MQI. In questo caso, GSKit implementa i protocolli socket protetti TLS.

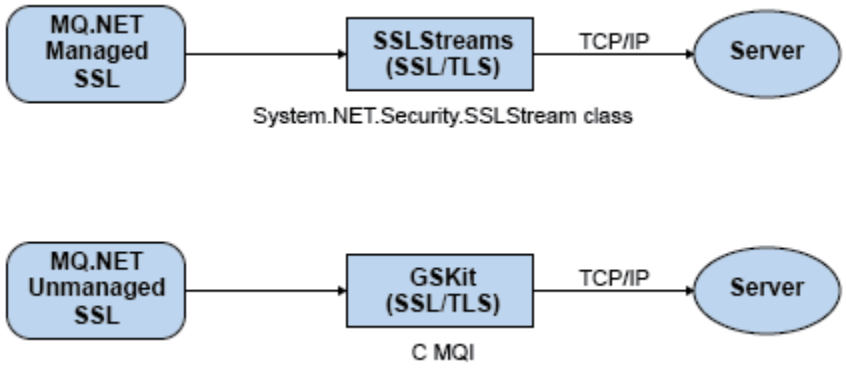


Figura 50. Confronto TLS gestito e non gestito di IBM MQ.NET

La seguente tabella riepiloga le differenze tra le implementazioni gestite e non gestite:

Tabella 14. Differenze tra implementazioni gestite e non gestite			
Modalità	**Protocolli**	**Implementazione**	**Commenti**
SSL gestito da IBM MQ.NET	TLS	Sistema.NET.Security.SSLStream SSLStream La classe SSLStream funziona come un flusso su un socket TCP connesso	TLS 1.0 TLS 1.2 (solo con Microsoft.NET Framework v4.5)
IBM MQ.NET SSL non gestito	TLS	GSKit e C-MQI	Protocolli socket sicuri TLS

Concetti correlati

Supporto SSL (Secure Sockets Layer) e TLS (Transport Layer Security) per .NET

IBM MQ MQI clients

Un IBM MQ MQI client è un componente del prodotto IBM MQ che può essere installato su un sistema su cui non è in esecuzione alcun gestore code.

Un IBM MQ client MQI è un componente che permette a un'applicazione in esecuzione su un sistema di emettere chiamate MQI a un gestore code in esecuzione su un altro sistema. L'output della chiamata viene inviato di nuovo al client, che lo restituisce all'applicazione.

Utilizzando un IBM MQ MQI client, un'applicazione in esecuzione sullo stesso sistema del client può connettersi a un gestore code in esecuzione su un altro sistema. L'applicazione può emettere chiamate MQI a tale gestore code. Tale applicazione è denominata IBM MQ MQI client e il gestore code è denominato *gestore code server*.

Un server IBM MQ è un gestore code che fornisce servizi di accodamento a uno o più client. Tutti gli oggetti IBM MQ , ad esempio le code, esistono solo sulla macchina del gestore code (la macchina server IBM MQ) e non sul client. Un server IBM MQ può anche supportare applicazioni IBM MQ locali.

La differenza tra un server IBM MQ e un gestore code ordinario è che un server ha un collegamento di comunicazioni dedicato con ciascun client. Per ulteriori informazioni sulla creazione di canali per client e server, consultare Configurazione dell'accodamento distribuito.

Un'applicazione IBM MQ MQI client e un gestore code del server comunicano tra loro utilizzando un canale *MQI*. Un canale MQI viene avviato quando l'applicazione client emette una chiamata **MQCONN** o **MQCONNX** per connettersi al gestore code e termina quando l'applicazione client emette una chiamata **MQDISC** per disconnettersi dal gestore code. I parametri di input di un flusso di chiamate MQI in una direzione su un canale MQI e i parametri di output nella direzione opposta.

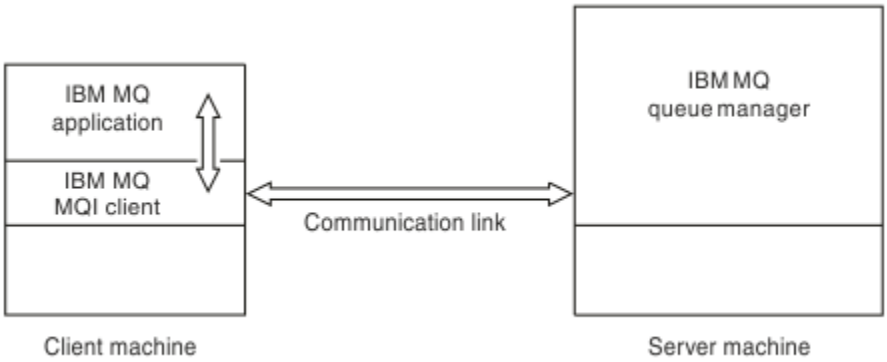


Figura 51. Collegamento tra un client e un server

È possibile utilizzare le piattaforme riportate di seguito. Le combinazioni dipendono da quale prodotto IBM MQ si sta utilizzando e sono descritte in “Supporto della piattaforma per client IBM MQ” a pagina 146.

IBM MQ MQI client	IBM MQ server
AIX and Linux Windows IBM i	AIX and Linux Windows IBM i z/OS

La MQI è disponibile per le applicazioni in esecuzione sulla piattaforma client; le code e gli altri oggetti IBM MQ si trovano su un gestore code installato su un server.

Un'applicazione che si desidera eseguire nell'ambiente IBM MQ MQI client deve essere prima collegata alla libreria client pertinente. Quando l'applicazione emette una chiamata MQI, IBM MQ MQI client indirizza la richiesta a un gestore code, dove viene elaborata e da dove viene inviata una risposta a IBM MQ MQI client.

Il collegamento tra l'applicazione e IBM MQ MQI client viene stabilito dinamicamente in fase di runtime.

È anche possibile sviluppare applicazioni client utilizzando IBM MQ classes for .NET, IBM MQ classes for Java o IBM MQ classes for Java Message Service (JMS). È possibile utilizzare i client Java e JMS sulle seguenti piattaforme:

- IBM i
- AIX
- Linux
- Windows

L'utilizzo di Java e JMS non è descritto qui. Per i dettagli completi su come installare, configurare e utilizzare IBM MQ classes for Java e IBM MQ classes for JMS consultare Utilizzo di IBM MQ classes for Java e Utilizzo di IBM MQ classes for JMS.

Applicazioni IBM MQ in un ambiente client-server

Quando sono collegati a un server, le applicazioni IBM MQ del client possono emettere la maggior parte delle chiamate MQI allo stesso modo delle applicazioni locali. L'applicazione client emette una chiamata `MQCONN` per connettersi a un gestore code specificato. Tutte le chiamate MQI aggiuntive che specificano l'handle di connessione restituito dalla richiesta di connessione vengono quindi elaborate da questo gestore code.

È necessario collegare le applicazioni alle librerie client appropriate. Consultare Creazione di applicazioni per IBM MQ MQI clients.

Concetti correlati

“Perché utilizzare i client IBM MQ ?” a pagina 146
L'uso dei client IBM MQ è un modo efficiente di implementare la messaggistica e l'accodamento IBM MQ .

“Cos' è un client transazionale esteso?” a pagina 147
Un client transazionale esteso IBM MQ può aggiornare le risorse gestite da un altro gestore risorse, sotto il controllo di un gestore transazioni esterno.

“Modalità di connessione del client al server” a pagina 149
Un client si connette a un server utilizzando MQCONN o MQCONNX e comunica attraverso un canale.

“Gestione e supporto delle transazioni” a pagina 150
Introduzione alla gestione delle transazioni e al modo in cui IBM MQ supporta le transazioni.

“Estensione delle funzioni del gestore code” a pagina 151

È possibile estendere le funzioni del gestore code utilizzando le uscite utente, le uscite API o i servizi installabili.

Informazioni correlate

Come configurare un IBM MQ MQI client

Perché utilizzare i client IBM MQ ?

L'uso dei client IBM MQ è un modo efficiente di implementare la messaggistica e l'accodamento IBM MQ .

È possibile avere un'applicazione che utilizza MQI in esecuzione su una macchina e il gestore code in esecuzione su una macchina differente (fisica o virtuale). I vantaggi di questa operazione sono:

- Non è necessaria un'implementazione IBM MQ completa sulla macchina client.
- I requisiti hardware sul sistema client sono ridotti.
- I requisiti di amministrazione del sistema sono ridotti.
- Un'applicazione IBM MQ in esecuzione su un client può connettersi a più gestori code su sistemi differenti.
- Possono essere utilizzati canali alternativi che utilizzano protocolli di trasmissione differenti.

Supporto della piattaforma per client IBM MQ

IBM MQ su tutte le piattaforme server supportate accetta le connessioni client da IBM MQ MQI clients su un numero di piattaforme.

IBM MQ installato come prodotto di base *e server* su tutte le piattaforme server supportate può accettare connessioni da IBM MQ MQI clients sulle piattaforme seguenti:

- IBM i
- AIX
- Linux
- Windows

I collegamenti client sono soggetti a differenze nel CCSID (Coded Character Set Identifier) e nel protocollo di comunicazioni.

Quali applicazioni vengono eseguite su un IBM MQ MQI client?

MQI completo è supportato nell'ambiente client. Ciò consente a quasi tutte le applicazioni IBM MQ di essere configurate per essere eseguite su un sistema IBM MQ MQI client collegando l'applicazione su IBM MQ MQI client alla libreria MQIC, piuttosto che alla libreria MQI. Le eccezioni sono:

- MQGET con segnale
- Un'applicazione che richiede il coordinamento del punto di sincronizzazione con altri gestori risorse deve utilizzare un client transazionale esteso

Se la lettura anticipata è abilitata, per migliorare le prestazioni della messaggistica non persistente, non sono disponibili tutte le opzioni MQGET. La tabella mostra le opzioni consentite e se è possibile modificarle tra le chiamate MQGET.

Tabella 15. Opzioni MQGET consentite quando la lettura anticipata è abilitata

Valori	**Consentito quando la lettura anticipata è abilitata e può essere modificato tra le chiamate MQGET**	**Consentito quando la lettura anticipata è abilitata ma non può essere modificato tra chiamate MQGET** [1]	**Le opzioni MQGET non consentite quando la lettura anticipata è abilitata** [2]
Valori MQGET MD	MsgId [3] CorrelId [3]	Codifica CodedCharSetId	

Tabella 15. Opzioni MQGET consentite quando la lettura anticipata è abilitata (Continua)			
Valori	**Consentito quando la lettura anticipata è abilitata e può essere modificato tra le chiamate MQGET**	**Consentito quando la lettura anticipata è abilitata ma non può essere modificato tra chiamate MQGET [1]**	**Le opzioni MQGET non consentite quando la lettura anticipata è abilitata [2]**
Opzioni MQGET MQGMO	MQGMO_WAIT MQGMO_NO_WAIT MQGMO_FAIL_IF_QUIESCING MQGMO_BROWSE_FIRST [4] MQGMO_BROWSE_NEXT [4] MESSAGGIO_BROWSE_MQGMO_ _UNDER_CURSOR [4]	MQGMO_SYNCPOINT_IF_PERSISTENT MQGMO_NO_SYNCPOINT MQGMO_ACCEPT_TRUNCATED_MSG MQGMO_CONVERT ORDER LOGICAL_MQGMO_ 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 SYNCPOINT MQGMO SKIP_MARK_MQGMO _BACKOUT MQGMO_MSG_UNDER _CURSORE [4] LOCK_MQGMO MQGMO_UNLOCK
Valori MQGMO		MsgHandle	

1. Se queste opzioni vengono modificate tra le chiamate MQGET, viene restituito un codice motivo MQRC_OPTIONS_CHANGED.
2. Se queste opzioni vengono specificate nella prima chiamata MQGET, il read ahead è disabilitato. Se queste opzioni vengono specificate in una successiva chiamata MQGET, viene restituito il codice motivo MQRC_OPTIONS_ERROR.
3. Le applicazioni client devono essere consapevoli che se i valori MsgId e CorrelId vengono modificati tra le chiamate MQGET, i messaggi con i valori precedenti potrebbero essere già stati inviati al client e rimanere nel buffer di lettura anticipata del client finché non vengono consumati (o automaticamente eliminati).
4. La prima chiamata MQGET determina se i messaggi devono essere ricercati o richiamati da una coda quando il read ahead è abilitato. Se l'applicazione tenta di utilizzare una combinazione di BROWSE e GET, viene restituito il codice motivo MQRC_OPTIONS_CHANGED.
5. MQGMO_MSG_UNDER_CURSOR non è consentito con il read ahead. Quando il read ahead è abilitato, i messaggi possono essere sfogliati o richiamati, ma non entrambe le cose.

Un'applicazione in esecuzione su un IBM MQ MQI client può connettersi a più di un gestore code contemporaneamente oppure utilizzare un nome gestore code con un asterisco (*) su una chiamata MQCONN o MQCONNX (consultare gli esempi in Connessione di applicazioni IBM MQ MQI client ai gestori code).

Cos' è un client transazionale esteso?

Un client transazionale esteso IBM MQ può aggiornare le risorse gestite da un altro gestore risorse, sotto il controllo di un gestore transazioni esterno.

Se non si ha familiarità con i concetti di gestione delle transazioni, consultare “Gestione e supporto delle transazioni” a pagina 150.

Notare che il client transazionale XA viene ora fornito come parte di IBM MQ.

Un'applicazione client può partecipare a un'unità di lavoro gestita da un gestore code a cui è connessa. All'interno dell'unità di lavoro, l'applicazione client può inserire e richiamare messaggi dalle code di proprietà di tale gestore code. L'applicazione client può quindi utilizzare la chiamata **MQCMIT** per eseguire il commit dell'unità di lavoro o la chiamata **MQBACK** per eseguire il backout dell'unità di lavoro. Tuttavia, all'interno della stessa unità di lavoro, l'applicazione client non può aggiornare le risorse di un altro gestore risorse, ad esempio le tabelle di un database Db2 . L'uso di un client transazionale esteso IBM MQ elimina questa limitazione.

Un client transazionale esteso IBM MQ è un IBM MQ MQI client con alcune funzioni aggiuntive. Utilizzando questa funzione un'applicazione client, all'interno della stessa unità di lavoro, può eseguire le seguenti attività:

- Inserire e richiamare i messaggi dalle code di proprietà del gestore code a cui è connesso
- Aggiornare le risorse di un gestore risorse diverso da un gestore code IBM MQ

Questa unità di lavoro deve essere gestita da un gestore transazioni esterno in esecuzione sullo stesso sistema dell'applicazione client. L'unità di lavoro non può essere gestita dal gestore code a cui è connessa l'applicazione client. Ciò significa che il gestore code può agire solo come gestore risorse e non come gestore transazioni. Ciò significa anche che l'applicazione client può eseguire il commit o il backout dell'unità di lavoro utilizzando solo l'API (application programming interface) fornita dal gestore transazioni esterno. L'applicazione client, pertanto, non può utilizzare le chiamate MQI, **MQBEGIN**, **MQCMIT**e **MQBACK**.

Il gestore transazioni esterno comunica con il gestore code come gestore risorse utilizzando lo stesso canale MQI utilizzato dall'applicazione client connessa al gestore code. Tuttavia, in una situazione di ripristino in seguito a un errore, quando nessuna applicazione è in esecuzione, il gestore transazioni può utilizzare un canale MQI dedicato per ripristinare le unità di lavoro incomplete a cui il gestore code partecipava al momento dell'errore.

In questa sezione, un IBM MQ MQI client che non dispone della funzione transazionale estesa viene indicato come client di base IBM MQ . È possibile considerare, quindi, un client transazionale esteso IBM MQ composto da un client di base IBM MQ con aggiunta della funzione transazionale estesa.

Nota: IBM i IBM MQ MQI client su IBM i non supporta la funzione transazionale estesa IBM MQ .

Supporto della piattaforma per i client transazionali estesi

Multi

I client transazionali estesi sono disponibili per tutte le piattaforme multiple che supportano un client di base. I client non sono disponibili per z/OS.

Un'applicazione client che utilizza un client transazionale esteso può connettersi solo a un gestore code dei seguenti prodotti IBM MQ 9.0o successivi:

- AIX IBM MQ for AIX
- IBM i IBM MQ for IBM i
- Linux IBM MQ per Linux
- Windows IBM MQ for Windows

z/OS Anche se non ci sono client transazionali estesi in esecuzione su z/OS, un'applicazione client che utilizza un client transazionale esteso può connettersi a un gestore code in esecuzione su z/OS.

Per ogni piattaforma, i requisiti hardware e software per il client transazionale esteso corrispondono a quelli per il client di base IBM MQ . Un linguaggio di programmazione è supportato da un client transazionale esteso se è supportato dal client di base IBM MQ e dal gestore transazioni che si sta utilizzando.

Per informazioni sui gestori transazioni esterni per tutte le piattaforme, consultare Requisiti di sistema per IBM MQ.

Modalità di connessione del client al server

Un client si connette a un server utilizzando MQCONN o MQCONNX e comunica attraverso un canale.

Un'applicazione in esecuzione nell'ambiente client IBM MQ deve mantenere una connessione attiva tra le macchine client e server.

La connessione viene effettuata da un'applicazione che emette una chiamata MQCONN o MQCONNX. I client e i server comunicano tramite i *canali MQI*oppure, quando si utilizzano conversazioni condivise, ogni conversazione condivide un'istanza del canale MQI. Quando la chiamata ha esito positivo, la conversazione o l'istanza del canale MQI rimane connessa finché l'applicazione non emette una chiamata MQDISC. Questo è il caso di ogni gestore code a cui un'applicazione deve connettersi.

Client e gestore code sulla stessa macchina

È inoltre possibile eseguire un'applicazione nell'ambiente IBM MQ MQI client quando sulla macchina è installato anche un gestore code.

In questa situazione, si ha la possibilità di collegarsi alle librerie del gestore code o alle librerie del client, ma tenere presente che se si collega alle librerie del client, è comunque necessario definire le connessioni del canale. Ciò può essere utile durante la fase di sviluppo di una applicazione. È possibile testare il programma sulla propria macchina, senza dipendere da altri, ed essere certi che funzionerà ancora quando lo si sposta in un ambiente IBM MQ MQI client indipendente.

Client su diverse piattaforme

In questo esempio, la macchina server comunica con tre IBM MQ MQI clients su piattaforme differenti.

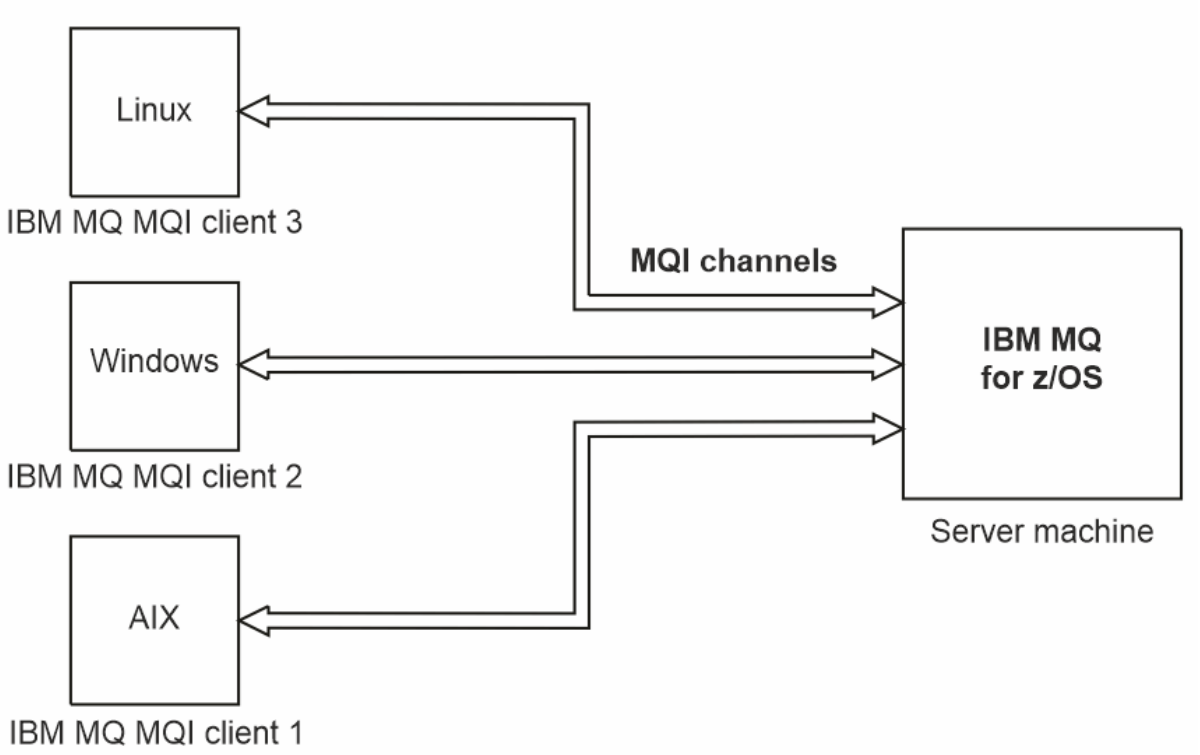


Figura 52. Server IBM MQ connesso a client su piattaforme differenti

Altri ambienti più complessi sono possibili. Ad esempio, un client IBM MQ può connettersi a più di un gestore code o a qualsiasi numero di gestori code connessi come parte di un gruppo di condivisione code.

Utilizzo di versioni differenti di software client e server

Se si stanno utilizzando versioni precedenti di prodotti IBM MQ , assicurarsi che la conversione del codice dal CCSID del proprio client sia supportato dal server.

Un client IBM MQ può connettersi a tutte le versioni supportate del gestore code. Se ci si connette a un gestore code di una versione precedente, non è possibile utilizzare le funzioni e le strutture di una versione successiva del prodotto nell'applicazione IBM MQ sul client.

Un gestore code IBM MQ può comunicare con i client di versioni diverse con se stesso negoziando fino al livello di protocollo supportato reciprocamente più alto. Ciò significa che i client più vecchi possono essere utilizzati con livelli di gestore code successivi. Si consiglia che sia il client che il server siano alle versioni di IBM MQ attualmente supportate per facilitare la diagnostica dei problemi e abilitare il supporto da parte di IBM.

Per ulteriori informazioni, consultare i linguaggi di programmazione supportati in Sviluppo delle applicazioni.

Gestione e supporto delle transazioni

Introduzione alla gestione delle transazioni e al modo in cui IBM MQ supporta le transazioni.

Un *gestore risorse* è un sottosistema di computer che possiede e gestisce le risorse a cui le applicazioni possono accedere e che possono essere aggiornate. Di seguito sono riportati esempi di gestori risorse:

- Un gestore code IBM MQ , con le relative risorse
- Un database Db2 , con le risorse che sono le relative tabelle

Quando un'applicazione aggiorna le risorse di uno o più gestori risorse, potrebbe esserci un requisito di business per garantire che alcuni aggiornamenti vengano completati correttamente come un gruppo o che nessuno di essi venga completato. Il motivo per questo tipo di requisito è che i dati di business verrebbero lasciati in uno stato incongruente se alcuni di questi aggiornamenti fossero stati completati correttamente, ma altri no.

Gli aggiornamenti alle risorse gestite in questo modo si verificano all'interno di un' *unità di lavoro*o di *transazione*. Un programma applicativo può raggruppare una serie di aggiornamenti in un'unità di lavoro.

Durante un'unità di lavoro, un'applicazione invia richieste ai gestori risorse per aggiornare le relative risorse. L'unità di lavoro termina quando l'applicazione emette una richiesta di commit di tutti gli aggiornamenti. Fino a quando non viene eseguito il commit degli aggiornamenti, nessuna di esse diventa visibile alle altre applicazioni che accedono alle stesse risorse. In alternativa, se l'applicazione decide di non poter completare l'unità di lavoro per qualsiasi motivo, può emettere una richiesta di backout di tutti gli aggiornamenti che ha richiesto fino a quel momento. In questo caso, nessuno degli aggiornamenti diventa mai visibile ad altre applicazioni. Questi aggiornamenti sono di solito correlati logicamente e devono essere tutti corretti per preservare l'integrità dei dati. Se un aggiornamento ha esito positivo mentre un altro ha esito negativo, l'integrità dei dati viene persa.

Quando un'unità di lavoro viene completata correttamente, viene detto *commit*. Una volta eseguito il commit, tutti gli aggiornamenti effettuati all'interno di tale unità di lavoro sono resi permanenti e irreversibili. Tuttavia, se l'unità di lavoro non riesce, tutti gli aggiornamenti vengono *ripristinati*. Questo processo, in cui le unità di lavoro vengono sottoposte a commit o a backout con integrità, è noto come *coordinamento del punto di sincronizzazione*.

Il momento in cui viene eseguito il commit o il backout di tutti gli aggiornamenti all'interno di un'unità di lavoro viene definito *punto di sincronizzazione*. Un aggiornamento all'interno di un'unità di lavoro si verifica *all'interno del controllo del punto di sincronizzazione*. Se un'applicazione richiede un aggiornamento *esterno al controllo del punto di sincronizzazione*, il gestore risorse esegue il commit dell'aggiornamento immediatamente, anche se è presente un'unità di lavoro in corso e non è possibile eseguire il backout dell'aggiornamento in un secondo momento.

Il sottosistema del computer che gestisce le unità di lavoro è denominato *gestore transazioni*o *coordinatore del punto*.

Un'unità di lavoro *locale* è un'unità in cui le uniche risorse aggiornate sono quelle del gestore code IBM MQ . Qui il coordinamento del punto di sincronizzazione viene fornito dal gestore code stesso utilizzando un processo di commit a fase singola.

Un'unità di lavoro *globale* è un'unità di lavoro in cui vengono aggiornate anche le risorse appartenenti ad altri gestori risorse, ad esempio i database compatibili con XA. Qui, è necessario utilizzare una procedura di commit a due fasi e l'unità di lavoro può essere coordinata dal gestore code stesso o esternamente da un altro gestore transazioni compatibile con XA come IBM TXSerieso BEA Tuxedo.

Un gestore transazioni è responsabile di garantire che tutti gli aggiornamenti alle risorse all'interno di un'unità di lavoro vengano completati correttamente o che nessuno di essi venga completato. È a un gestore transazioni che un'applicazione emette una richiesta per eseguire il commit o il backout di un'unità di lavoro. Esempi di gestori transazioni sono CICS e WebSphere Application Server, sebbene entrambi dispongano di altre funzioni.

Alcuni gestori risorse forniscono la propria funzione di gestione transazioni. Ad esempio, un gestore code IBM MQ può gestire unità di lavoro che includono aggiornamenti alle proprie risorse e aggiornamenti alle tabelle Db2 . Il gestore code non ha bisogno di un gestore transazioni separato per eseguire questa funzione, anche se è possibile utilizzarne uno se si tratta di un requisito utente. Se viene utilizzato un gestore transazioni separato, viene indicato come *gestore transazioni esterno*.

Affinché un gestore transazioni esterno gestisca un'unità di lavoro, deve esistere un'interfaccia standard tra il gestore transazioni e ogni gestore risorse che partecipa all'unità di lavoro. Questa interfaccia consente al gestore transazioni e a un gestore risorse di comunicare tra loro. Una di queste interfacce è l' *Interfaccia XA*, che è un'interfaccia standard supportata da diversi gestori transazioni e gestori risorse. L'interfaccia XA viene pubblicata da Open Group in *Distributed Transaction Processing: The XA Specification*.

Quando più di un gestore risorse partecipa a un'unità di lavoro, un gestore transazioni deve utilizzare un protocollo di *commit a due fasi* per garantire che tutti gli aggiornamenti all'interno dell'unità di lavoro vengano completati correttamente o nessuno di essi venga completato, anche se si verifica un errore di sistema. Quando un'applicazione invia una richiesta a un gestore transazioni per eseguire il commit di un'unità di lavoro, il gestore transazioni effettua le seguenti operazioni:

Fase 1 (preparazione al commit)
: Il gestore transazioni richiede a ciascun gestore risorse che partecipa all'unità di lavoro di verificare che tutte le informazioni relative agli aggiornamenti previsti per le relative risorse siano in uno stato ripristinabile. Un gestore risorse normalmente esegue questa operazione scrivendo le informazioni in un log e verificandone la scrittura sul disco fisso. La fase 1 viene completata quando il gestore transazioni riceve la notifica da ciascun gestore risorse che le informazioni relative agli aggiornamenti previsti per le relative risorse si trovano in uno stato ripristinabile.

Fase 2 (commit)
: Al termine della fase 1, il gestore della transazione prende la decisione irrevocabile di impegnare l'unità di lavoro. Chiede a ciascun gestore risorse che partecipa all'unità di lavoro di eseguire il commit degli aggiornamenti alle proprie risorse. Quando un gestore risorse riceve questa richiesta, deve eseguire il commit degli aggiornamenti. Non ha la possibilità di eseguirne il backout in questo momento. La fase 2 viene completata quando il gestore transazioni riceve una notifica da ciascun gestore risorse che ha eseguito il commit degli aggiornamenti alle proprie risorse.

L'interfaccia XA utilizza un protocollo di commit a due fasi.

Per ulteriori informazioni, consultare Scenari di supporto transazionale.

IBM MQ fornisce inoltre il supporto per Microsoft Transaction Server (COM +). L'utilizzo di Microsoft Transaction Server (COM +) fornisce informazioni su come configurare IBM MQ per sfruttare il supporto COM +.

Estensione delle funzioni del gestore code

È possibile estendere le funzioni del gestore code utilizzando le uscite utente, le uscite API o i servizi installabili.

Uscite Utente

Le uscite utente forniscono un meccanismo per inserire il proprio codice in una funzione del gestore code. Le uscite utente supportate includono:

Uscite canale
Queste uscite cambiano il modo in cui operano i canali. Le uscite canale sono descritte in Programmi di uscita canale per i canali di messaggistica.

Uscite di conversione dati
Queste uscite creano frammenti di codice sorgente che possono essere inseriti in programmi applicativi per convertire i dati da un formato all'altro. Le uscite di conversione dati sono descritte in Scrittura delle uscite di conversione dati.

L'uscita del workload del cluster
La funzione eseguita da questa uscita è definita dal provider dell'uscita. Le informazioni sulla definizione della chiamata vengono fornite in MQ_CLUSTER_WORKLOAD_EXIT - Descrizione chiamata.

Uscite API

Le uscite API consentono di scrivere il codice che modifica il funzionamento delle chiamate API IBM MQ , come MQPUT e MQGET, e quindi inserire tale codice immediatamente prima o subito dopo tali chiamate. L'inserimento è automatico; il gestore code guida il codice di uscita nei punti registrati. Per ulteriori informazioni sulle uscite API, vedi Utilizzo e scrittura delle uscite API.

Servizi installabili

I servizi installabili hanno interfacce formalizzate (un'API) con più punti di ingresso.

Un'implementazione di un servizio installabile è denominata *componente del servizio*. È possibile utilizzare i componenti forniti con IBM MQoppure è possibile scrivere il proprio componente per eseguire le funzioni richieste.

Attualmente, vengono forniti i seguenti servizi installabili:

Servizio di autorizzazione
Il servizio di autorizzazione consente di creare la propria funzione di protezione.

Il componente di servizio predefinito che implementa il servizio è OAM (object authority manager). Per impostazione predefinita, OAM è attivo e non è necessario eseguire alcuna operazione per configurarlo. È possibile utilizzare l'interfaccia del servizio di autorizzazione per creare altri componenti per sostituire o aumentare l'OAM. Per ulteriori informazioni su OAM, vedi Configurazione della sicurezza sui sistemi AIX, Linux, and Windows.

Servizio di denominazione
Il servizio dei nomi consente alle applicazioni di condividere le code identificando le code remote come se fossero code locali.

È possibile scrivere il proprio componente servizio nomi. Si potrebbe voler eseguire questa operazione se si intende utilizzare il servizio nomi con IBM MQ, ad esempio. Per utilizzare il servizio nomi, è necessario disporre di un componente scritto dall'utente o fornito da un altro fornitore di software. Per impostazione predefinita, il servizio nomi è inattivo.

Concetti correlati
Uscite utente, uscite API e servizi installabili IBM MQ

Interfacce di lingua IBM MQ Java

IBM MQ fornisce tre API (application programming interface) da utilizzare nelle applicazioni Java : IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMSe IBM MQ classes for Java.

IBM supporta ed è un partecipante attivo in standard aperti.

- Da IBM MQ 8.0, il prodotto implementa lo standard JMS 2.0 , che ha introdotto una nuova API semplificata insieme a funzioni come le sottoscrizioni condivise.
- Da IBM MQ 9.3.0, è supportato anche Jakarta Messaging 3.0 .
- Inoltre, WebSphere Liberty supporta JMS 2.0 e Jakarta Messaging 3.0 con IBM MQ.

All'interno di IBM MQ ci sono tre API da usare nelle applicazioni Java :

JM 3.0 **IBM MQ classes for Jakarta Messaging**
IBM MQ classes for Jakarta Messaging è un provider Jakarta Messaging che implementa le interfacce Jakarta Messaging per IBM MQ come sistema di messaggistica. Jakarta Connectors Architecture fornisce una modalità standard di connessione delle applicazioni in esecuzione in un ambiente Jakarta EE a un EIS (Enterprise Information System) come IBM MQ o Db2.

JMS 2.0 **IBM MQ classes for JMS**
IBM MQ classes for JMS è un provider JMS che implementa le interfacce JMS per IBM MQ come sistema di messaggistica. Java Platform, Enterprise Edition Connector Architecture (JCA) fornisce un modo standard per collegare le applicazioni in esecuzione in un ambiente Java EE a un EIS (Enterprise Information System) come IBM MQ o Db2.

IBM MQ classes for Java
IBM MQ classes for Java consente di utilizzare IBM MQ in un ambiente Java . IBM MQ classes for Java consenti a un'applicazione Java di connettersi a IBM MQ come client IBM MQ o di connettersi direttamente a un gestore code IBM MQ .

Nota:

- JMS 2.0 è stato sostituito da Jakarta Messaging. IBM MQ classes for JMS continua a supportare lo standard JMS 2.0 , ma i futuri miglioramenti alla messaggistica Java emergeranno solo in Jakarta Messaging, quindi in IBM MQ classes for Jakarta Messaging. IBM MQ classes for JMS sono consigliati solo per la manutenzione e l'estensione di applicazioni JMS 2.0 esistenti. IBM MQ classes for Jakarta Messaging dovrebbe essere la tecnologia preferita per il nuovo sviluppo.
- Stabilized IBM MQ classes for Java sono funzionalmente stabilizzati al livello fornito in IBM MQ 8.0. Le applicazioni esistenti che utilizzano il IBM MQ classes for Java continueranno ad essere completamente supportate, ma questa API è stabilizzata, quindi le nuove funzioni non saranno aggiunte e le richieste di miglioramenti rifiutate. Completamente supportato significa che i difetti verranno corretti insieme a tutte le modifiche richieste dalle modifiche ai requisiti di sistema IBM MQ .

JM 3.0 Da IBM MQ 9.3, IBM MQ classes for Java, IBM MQ classes for JMSe IBM MQ classes for Jakarta Messaging vengono creati con Java 8. Gli ambienti di runtime Java a questi livelli o superiori devono essere utilizzati per eseguire le applicazioni utilizzando queste interfacce.

Concetti correlati
Accesso a IBM MQ da Java - Scelta dell'API
Perché utilizzare le classi IBM MQ per Jakarta Messaging?
Perché dovrei utilizzare IBM MQ classes for JMS?
Perché dovrei utilizzare IBM MQ classes for Java?

IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging sono provider di messaggistica forniti con IBM MQ. Ciascuno di questi provider fornisce anche due serie di estensioni all'API di messaggistica. Le applicazioni Java Platform, Standard Edition (Java SE) e Java Platform, Enterprise Edition (Java EE) possono utilizzare questi fornitori di messaggistica.

JM 3.0 IBM MQ 9.3.0 ha introdotto il supporto per Jakarta Messaging 3.0. JMS 2.0 è ancora completamente supportato.

Le specifiche JMS e Jakarta Messaging definiscono una serie di interfacce che le applicazioni possono utilizzare per eseguire operazioni di messaggistica. Il prodotto supporta la versione JMS 2.0 dello

standard JMS . Questa implementazione offre tutte le funzioni dell'API classica ma richiede meno interfacce ed è più semplice da utilizzare. Per ulteriori informazioni, consultare “Modello JMS e Jakarta Messaging” a pagina 157 e la specifica JMS 2.0 all'indirizzo Java.net. JM 3.0 Da IBM MQ 9.3.0, è supportato anche Jakarta Messaging .

Il package jakarta.jms (Jakarta Messaging 3.0) o javax.jms (JMS 2.0) specifica i dettagli delle interfacce di messaggistica e un provider di messaggistica implementa tali interfacce per un prodotto di messaggistica specifico. Ad esempio:

- IBM MQ classes for JMS è un fornitore JMS che implementa le interfacce JMS per IBM MQ e fornisce anche le seguenti due serie di estensioni all'API JMS :
  - Estensioni IBM MQ JMS
  - Estensioni IBM JMS
- Un oggetto argomento, coda o factory di connessione creato utilizzando javax.dimso jakarta.jms, le interfacce o una serie di estensioni JMS possono essere indirizzate utilizzando una qualsiasi di queste API; ovvero, può essere assegnato a una qualsiasi delle interfacce. Per mantenere la portabilità dell'applicazione al livello più alto, utilizzare l'API più generica adatta ai propri requisiti.

Poiché JMS e Jakarta Messaging condividono molto in comune, ulteriori riferimenti a JMS in questo argomento possono essere considerati come riferimenti ad entrambi. Eventuali differenze vengono evidenziate come necessario.

Estensioni IBM MQ JMS

IBM MQ classes for JMS fornisce inoltre le estensioni all'API JMS . IBM MQ classes for JMS contiene le estensioni implementate negli oggetti MQConnectionFactory, MQQueue e MQTopic. Questi oggetti hanno proprietà e metodi specifici di IBM MQ. Gli oggetti possono essere gestiti oppure un'applicazione può creare gli oggetti in modo dinamico al runtime. Queste estensioni sono definite estensioni IBM MQ JMS . Notare che, in questa documentazione, gli oggetti creati dinamicamente da un'applicazione in fase di runtime non sono considerati oggetti gestiti.

Estensioni IBM JMS

Oltre alle estensioni IBM MQ JMS , IBM MQ classes for JMS fornisce una serie più generica di estensioni all'API JMS o Java come linguaggio di programmazione utilizzato. Queste estensioni sono definite estensioni IBM JMS e hanno i seguenti obiettivi generali:

- Per fornire un livello maggiore di coerenza tra i provider IBM JMS .
- Per semplificare la scrittura di un'applicazione bridge tra due sistemi di messaggistica IBM .
- Per rendere più semplice la porta di un'applicazione da un provider IBM JMS a un altro.

Il focus principale di queste estensioni riguarda la creazione e la configurazione di factory di connessione e destinazioni in modo dinamico al runtime, ma le estensioni forniscono anche funzioni che non sono direttamente correlate alla messaggistica, come la funzione per la determinazione dei problemi.

Attività correlate

Utilizzo delle classi IBM MQ per JMS/Jakarta Messaging

Configurazione delle risorse JMS e Jakarta Messaging

JM 3.0 IBM MQ classes for Jakarta Messaging: una panoramica

IBM MQ 9.3.0 introduce il supporto per Jakarta Messaging. Per Jakarta Messaging 3.0, il controllo della specifica JMS è stato spostato da Oracle al processo della comunità Java . Tuttavia, Oracle conserva il controllo del nome "javax", che viene utilizzato in altre tecnologie Java . Quindi, sebbene Jakarta Messaging 3.0 sia funzionalmente equivalente a JMS 2.0, ci sono alcune differenze nella denominazione. Il nome ufficiale per la versione 3.0 non è Java Message Service, ma Jakarta Messaging e i nomi del pacchetto e della costante hanno come prefisso `jakarta` piuttosto che `javax`.

Sfondo

Per molti anni, la piattaforma Java è disponibile in due formati: Standard Edition e Enterprise Edition.

Java Platform, Standard Edition (a volte abbreviato come Java SE) è la lingua principale e le librerie di classe, in grado di essere eseguite in un contesto autonomo. La maggior parte dei package Java in Java SE ha nomi che iniziano con "java.".

Java Platform, Enterprise Edition (Java EE) estende questa funzione, aggiungendo funzionalità come messaggistica, vari bean, transazionalità e così via. Alcune di queste tecnologie possono essere utilizzate anche in un contesto Java SE . La maggior parte dei pacchetti Java in Java EE storicamente hanno nomi che iniziano con "javax." - c'è qualche crossover, tuttavia, alcuni pacchetti Java SE hanno "javax." come prefisso del nome.

Java Message Service (JMS) fa parte di Java Platform, Enterprise Edition. Java EE 7 integra JMS 2.0.

Fino a Java EE 7, le tecnologie erano sotto la direzione di Oracle.

Le tecnologie Java EE sono state recentemente trasferite dalla gestione di Oracle a un processo della community supervisionato da Eclipse Foundation.

Come il "javax". non è stato possibile spostare il nome nel nuovo progetto, è stato adottato un nuovo nome - tutti i package e i nomi delle proprietà hanno ora il prefisso "jakarta". e Java Platform, Enterprise Edition verrà chiamato "Jakarta EE" in futuro. La numerazione della versione è continuata: la versione 8 era una versione provvisoria che può essere in gran parte ignorata, e Jakarta EE 9 è il punto in cui il "jakarta". il prefisso ha effetto.

La tecnologia Jakarta EE principale che si applica nel contesto IBM MQ è Jakarta Messaging 3.0 - il successore di Java Message Service (JMS) 2.0. Quindi Jakarta EE 9 incorpora Jakarta Messaging 3.0.

IBM MQ continua a supportare Java EE 7 e JMS 2.0, introducendo il supporto per Jakarta EE 9 e Jakarta Messaging 3.0.

Cosa viene consegnato: Java SE

Per Java Platform, Standard Edition, oltre a IBM MQ classes for JMS (che supporta le operazioni JMS 2.0 con IBM MQ) IBM MQ 9.3.0 e versioni successive, fornisce IBM MQ classes for Jakarta Messaging. Queste classi forniscono un provider Jakarta Messaging 3.0 che si integra con IBM MQ, consentendo l'uso dei gestori code IBM MQ per facilitare le operazioni Jakarta Messaging .

Tali file vengono forniti come file JAR standard, `com.ibm.mq.jakarta.client.jar`, nella directory secondaria `java/lib` dell'installazione IBM MQ .

Per l'utilizzo in contenitori OSGi, come Apache Felix o Eclipse Equinox, IBM MQ fornisce anche una coppia di bundle OSGi:

- `com.ibm.mq.osgi.jms30.clientprereqs_`*`V.R.M.F`*`.jar`
- `com.ibm.mq.osgi.jms30.client_`*`V.R.M.F`*`.jar`

dove *V.R.M.F* rappresenta la versione di IBM MQ, ad esempio 9.3.0.0. Questi bundle si trovano nella sottodirectory `java/lib/OSGi` dell'installazione IBM MQ .

Cosa viene consegnato: Jakarta EE 9

Per supportare la messaggistica basata su IBM MQin un server delle applicazioni compatibile con Jakarta EE 9 , IBM MQ fornisce un adattatore di risorse compatibile con Jakarta EE 9: `wmq.jakarta.jmsra.rar`. Si trova nella sottodirectory `java/lib/jca` dell'installazione di IBM MQ .

IBM MQ continua a fornire un adattatore di risorse compatibile con Java EE 7 , `wmq.jmsra.rar`, nella directory secondaria `java/lib/jca` dell'installazione di IBM MQ .

Modalità di consegna di queste risorse utente

Questi JAR e il file RAR per l'adattatore risorse sono forniti con le risorse preesistenti nel normale supporto di installazione di IBM MQ - sia il supporto di installazione specifico della piattaforma, come i file "`.rpm`", che il supporto ridistribuibile, come i file JAR del client ridistribuibili autoestraenti.

Cosa è cambiato tra JMS 2.0 e Jakarta Messaging 3.0

Jakarta EE 9 e Jakarta Messaging 3.0 non introducono nuove funzioni. Tutto ciò che cambia sono i nomi. Ad esempio, quando si utilizza "javax.jms.Connection" in JMS 2.0, si utilizza "jakarta.jms.Connection" in Jakarta Messaging 3.0.

Poiché Eclipse Foundation porta avanti la piattaforma Jakarta EE , si baserà su questa base e questa convenzione di denominazione verrà utilizzata per le nuove funzionalità introdotte in futuro.

Cosa è cambiato tra IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging

Riepilogo

IBM MQ classes for JMS, che fornisce supporto per JMS 2.0, rimane disponibile e si consiglia principalmente di gestire ed estendere le applicazioni esistenti. Essi sono pienamente supportati.

IBM MQ classes for Jakarta Messaging, che fornisce supporto per Jakarta Messaging 3.0, è consigliato per il nuovo sviluppo.

In IBM MQ 9.3.0, queste due offerte erano funzionalmente equivalenti. Solo la denominazione è diversa. Tuttavia, è più probabile che la nuova funzionalità di messaggistica emerga in IBM MQ classes for Jakarta Messaging che in IBM MQ classes for JMS.

Le due offerte sono interoperabili. I messaggi prodotti da IBM MQ classes for JMS possono essere utilizzati da IBM MQ classes for Jakarta Messaginge viceversa. Ma le due offerte non devono coesistere in un'unica applicazione.

Modifiche di denominazione

Tabella 16. Modifiche ai nomi dei package

IBM MQ classes for JMS nome pacchetto	**IBM MQ classes for Jakarta Messaging nome pacchetto**
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.*

I pacchetti relativi ai servizi comuni (traccia, registrazione, supporto lingua nazionale ecc.) e le implementazioni JMQI (locale e remota) sono comuni sia a IBM MQ classes for JMS che a IBM MQ classes for Jakarta Messaging, pertanto non sono necessarie modifiche in queste aree.

Si noti che anche i nomi delle proprietà sono stati modificati. Ad esempio, la proprietà per abilitare le estensioni IBM MQ in IBM MQ classes for Jakarta Messaging è **`com.ibm.mq.jakarta.jms.SupportMQExtensions`**.

I nomi delle proprietà che sono indipendenti da IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging, come le diverse proprietà **`com.ibm.msg.client.commonservices.trace.*`** , si applicano ugualmente a entrambe le offerte.

Programmi di utilità amministrativi

I programmi di utilità **`crtmqenv`** e **`setmqenv`** accettano ora un'opzione per specificare se il percorso classi deve essere configurato per IBM MQ classes for JMS (`-j 2.0`) o IBM MQ classes for Jakarta

Messaging (`-j 3.0`) e vi sono IBM MQ classes for Jakarta Messaging varianti dei programmi di utilità **runjms** , denominati **runjms30** e nomi simili.

Il programma di utilità **dspmqver** , quando viene richiesto di creare un report sui componenti Java , include IBM MQ classes for Jakarta Messaging nel relativo output.

Per configurare gli oggetti IBM MQ classes for Jakarta Messaging da recuperare tramite JNDI, il nuovo programma di utilità **JMS30Admin** è equivalente al programma di utilità **JMSAdmin** per IBM MQ classes for JMS.

Notare che gli oggetti sottostanti provengono da pacchetti differenti. le definizioni JNDI create da **JMSAdmin** non possono essere utilizzate da IBM MQ classes for Jakarta Messaging, né quelle create da **JMS30Admin** possono essere utilizzate da IBM MQ classes for JMS.

Nota: Non vi è alcun supporto per gli oggetti IBM MQ classes for Jakarta Messaging forniti da IBM MQ Explorer; la sua integrazione JNDI è solo per IBM MQ classes for JMS .

Concetti correlati

Perché utilizzare le classi IBM MQ per Jakarta Messaging?

Modello JMS e Jakarta Messaging

Il modello JMS e Jakarta Messaging definisce una serie di interfacce che le applicazioni Java possono utilizzare per eseguire operazioni di messaggistica. IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging sono provider di messaggistica che definiscono il modo in cui gli oggetti di messaggistica Java sono correlati ai concetti IBM MQ . Le specifiche JMS e Jakarta Messaging prevedono che alcuni oggetti di messaggistica siano oggetti gestiti.

Da IBM MQ 8.0, il prodotto supporta la versione JMS 2.0 dello standard JMS, che ha introdotto un'API semplificata, conservando anche l'API classica, da JMS 1.1.

JM 3.0 IBM MQ 9.3.0 ha introdotto il supporto per Jakarta Messaging 3.0. JMS 2.0 è ancora completamente supportato. Poiché JMS e Jakarta Messaging condividono molto in comune, ulteriori riferimenti a JMS in questo argomento possono essere considerati come riferimenti ad entrambi. Eventuali differenze vengono evidenziate come necessario.

API semplificata

JMS 2.0 ha introdotto l'API semplificata, mantenendo anche le interfacce specifiche del dominio e indipendenti dal dominio da JMS 1.1. L'API semplificata riduce il numero di oggetti necessari per inviare e ricevere messaggi e consiste nelle seguenti interfacce:

ConnectionFactory
Un ConnectionFactory è un oggetto gestito utilizzato da un client JMS per creare una connessione. Questa interfaccia viene utilizzata anche nell'API classica.

JMSContesto
Questo oggetto combina gli oggetti Connection e Session dell'API classica. JMSGli oggetti di contesto possono essere creati da altri JMSoggetti di contesto, con la connessione sottostante duplicata.

JMSMittente
Un Producer JMSviene creato da un contesto JMSe viene utilizzato per inviare messaggi a una coda o a un argomento. L'oggetto Producer JMSprovoca la creazione degli oggetti richiesti per inviare il messaggio.

JMSDestinatario
Un consumer JMSviene creato da un contesto JMSe viene utilizzato per ricevere messaggi da un argomento o da una coda.

L'API semplificata ha una serie di effetti:

- L'oggetto Context JMSavvia sempre automaticamente la connessione sottostante.
- JMSI produttori e JMSi consumatori possono ora lavorare direttamente con il corpo del messaggio, senza dover ottenere l'intero oggetto del messaggio, utilizzando il metodo getBody del messaggio.

- Le proprietà dei messaggi possono essere impostati sull'oggetto JMSProducer, utilizzando il concatenamento di metodo, prima di inviare un 'corpo ', un contenuto di messaggi. Il Producer JMSgestirà la creazione di tutti gli oggetti necessari per inviare il messaggio. Utilizzando JMS 2.0, è possibile impostare le proprietà e inviare un messaggio nel modo seguente:

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

JMS 2.0 ha anche introdotto sottoscrizioni condivise in cui i messaggi possono essere condivisi tra più consumer. Tutte le sottoscrizioni JMS 1.1 vengono considerate sottoscrizioni non condivise.

API classica

Il seguente elenco riepiloga le principali interfacce JMS dell'API classica:

Destinazione
Una destinazione è il punto in cui un'applicazione invia i messaggi o è un'origine da cui un'applicazione riceve i messaggi o entrambi.

ConnectionFactory
Un oggetto ConnectionFactory contiene una serie di proprietà di configurazione per una connessione. Un'applicazione utilizza una produzione connessioni per creare una connessione.

Connessione
Un oggetto Connection incapsula una connessione attiva dell'applicazione a un server di messaggistica. Un'applicazione utilizza una connessione per creare sessioni.

Sessione
Una sessione è un contesto a thread singolo per inviare e ricevere messaggi. Un'applicazione utilizza una sessione per creare messaggi, produttori di messaggi e utenti di messaggi. Una sessione è sottoposta a transazione o non è stata sottoposta a transazione.

Messaggio
Un oggetto Message incapsula un messaggio che un'applicazione invia o riceve.

MessageProducer
Un'applicazione utilizza un produttore di messaggi per inviare messaggi a una destinazione.

MessageConsumer
Un'applicazione utilizza un utente di messaggi per ricevere i messaggi inviati a una destinazione.

Figura 53 a pagina 158 mostra questi oggetti e le relazioni.

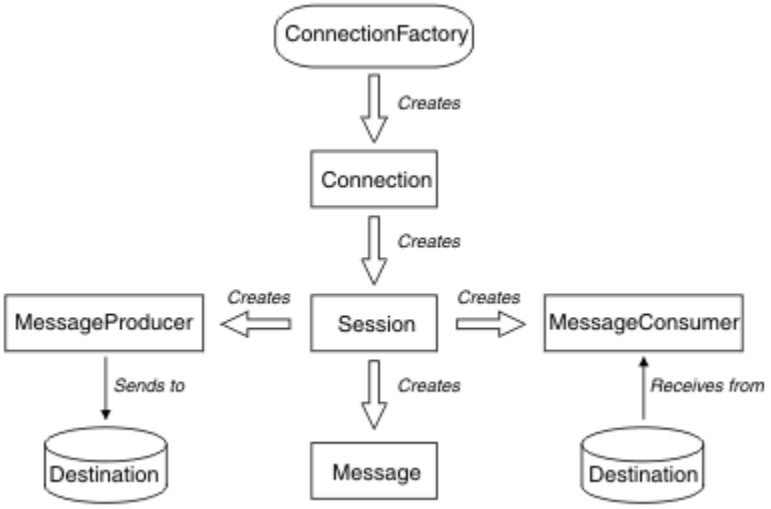


Figura 53. Oggetti JMS e relative relazioni

Il diagramma mostra le interfacce principali: ConnectionFactory, Connection, Session, MessageProducer, MessageConsumer, Message e Destination. Un'applicazione utilizza una produzione connessioni per creare una connessione e una connessione per creare sessioni. L'applicazione può quindi utilizzare una sessione per creare messaggi, produttori di messaggi e utenti di messaggi. L'applicazione utilizza un producer di messaggi per inviare messaggi a una destinazione e utilizza un consumer di messaggi per ricevere i messaggi inviati a una destinazione.

Un oggetto di destinazione, ConnectionFactoryo Connection può essere utilizzato contemporaneamente da thread differenti di un'applicazione a più thread, ma un oggetto Session, MessageProducero MessageConsumer non può essere utilizzato contemporaneamente da thread differenti. Il modo più semplice per garantire che un oggetto Session, MessageProducero MessageConsumer non venga utilizzato contemporaneamente consiste nel creare un oggetto Session separato per ciascun thread.

JMS supporta due stili di messaggistica:

- Messaggistica point-to-point
- Pubblicazione/sottoscrizione della messaggistica

Questi stili di messaggistica vengono anche indicati come *domini di messaggistica*ed è possibile combinare entrambi gli stili di messaggistica in un'applicazione. Nel dominio point-to-point, una destinazione è una coda e, nel dominio di pubblicazione / sottoscrizione, una destinazione è un argomento.

Con le versioni di JMS precedenti a JMS 1.1, la programmazione per il dominio point-to-point utilizza una serie di interfacce e metodi, mentre la programmazione per il dominio di pubblicazione / sottoscrizione utilizza un'altra serie. I due set sono simili, ma separati. Da JMS 1.1, è possibile utilizzare una serie comune di interfacce e metodi che supporta entrambi i domini di messaggistica. Le interfacce comuni forniscono una vista indipendente dal dominio di ciascun dominio di messaggistica. Tabella 17 a pagina 159 elenca le interfacce indipendenti del dominio JMS e le relative interfacce specifiche del dominio corrispondenti.

Tabella 17. Il dominio JMS indipendente e le relative interfacce specifiche del dominio corrispondenti

Interfacce indipendenti dal dominio	**Interfacce specifiche del dominio per il dominio point - to - point**	**Interfacce specifiche del dominio per il dominio di pubblicazione / sottoscrizione**
ConnectionFactory	Factory QueueConnection	Factory TopicConnection
Connessione	QueueConnection	TopicConnection
Destinazione	Coda	Argomento
Sessione	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 IBM MQ classes for JMS 2.0 supporta sia le precedenti interfacce specifiche del dominio JMS 1.1 che l'API semplificata di JMS 2.0. IBM MQ classes for JMS 2.0 può quindi essere utilizzato per gestire le applicazioni esistenti, incluso lo sviluppo di nuove funzioni in applicazioni esistenti.

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 supporta le versioni Jakarta Messaging delle stesse interfacce ed è consigliato per lo sviluppo di nuove applicazioni.

In IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, gli oggetti JMS sono correlati a concetti IBM MQ nei seguenti modi:

- Un oggetto Connection ha proprietà derivate dalle proprietà del factory di connessione utilizzato per creare la connessione. Queste proprietà controllano il modo in cui un'applicazione si connette a un gestore code. Esempi di queste proprietà sono il nome del gestore code e, per un'applicazione che

si connette al gestore code in modalità client, il nome host o l'indirizzo IP del sistema su cui è in esecuzione il gestore code.

- Un oggetto Session incapsula un handle di connessione IBM MQ , che definisce quindi l'ambito transazionale della sessione.
- Un oggetto MessageProducer e un oggetto MessageConsumer incapsulano un handle di oggetto IBM MQ .

Quando si utilizza IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging, vengono applicate tutte le normali regole di IBM MQ . Notare, in particolare, che un'applicazione può inviare un messaggio a una coda remota, ma può ricevere un messaggio solo da una coda di proprietà del gestore code a cui è connessa l'applicazione.

La specifica JMS prevede che gli oggetti ConnectionFactory e Destination siano oggetti gestiti. Un amministratore crea e gestisce gli oggetti gestiti in un repository centrale e un'applicazione JMS richiama tali oggetti utilizzando JNDI (Java Naming and Directory Interface).

In IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging, l'implementazione dell'interfaccia Destinazione è una superclasse astratta di Coda e Argomento, e quindi un'istanza di Destinazione è un oggetto Coda o un oggetto Argomento. Le interfacce indipendenti dal dominio trattano una coda o un argomento come una destinazione. Il dominio di messaggistica per un oggetto MessageProducer o MessageConsumer è determinato se la destinazione è una coda o un argomento.

In IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging , quindi, gli oggetti dei seguenti tipi possono essere oggetti gestiti:

- ConnectionFactory
- Factory QueueConnection
- Factory TopicConnection
- Coda
- Argomento
- XAConnectionFactory
- Factory XAQueueConnection
- Factory XATopicConnection

Architettura IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging hanno un'architettura a livelli. Il livello di codice più alto è un livello comune che può essere utilizzato da qualsiasi provider di messaggistica IBM Java .

JM 3.0 IBM MQ 9.3.0 ha introdotto il supporto per Jakarta Messaging 3.0. JMS 2.0 è ancora completamente supportato.

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging hanno un'architettura a livelli come mostrato nel diagramma Figura 54 a pagina 161. Il livello più alto di codice è un livello comune che può essere utilizzato da qualsiasi provider IBM JMS o Jakarta Messaging. Quando un'applicazione chiama un metodo JMS o Jakarta Messaging, qualsiasi elaborazione della chiamata che non è specifica di un sistema di messaggistica viene eseguita dal livello comune, che fornisce anche una risposta congruente alla chiamata. Qualsiasi elaborazione della chiamata specifica di un sistema di messaggistica viene delegata ad un livello inferiore. Nel seguente diagramma, il provider di messaggistica IBM MQ viene visualizzato nel livello inferiore, insieme a due ulteriori provider di messaggistica (provider di messaggistica A e provider di messaggistica B.)

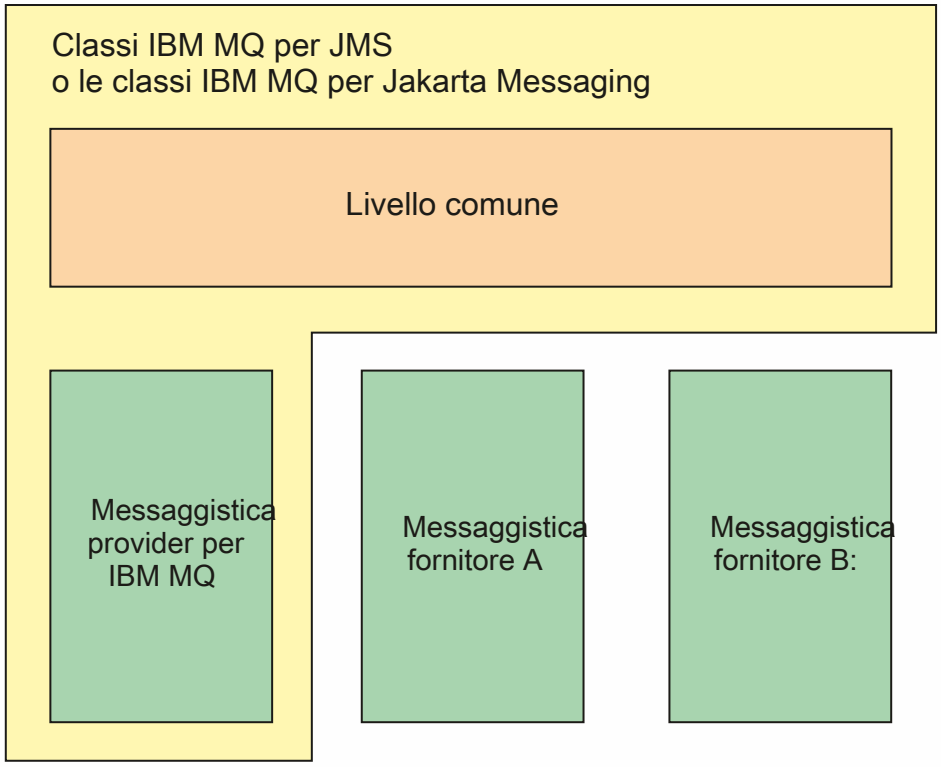


Figura 54. L'architettura a livelli per provider IBM JMS e Jakarta Messaging

Un'architettura a livelli soddisfa i seguenti obiettivi:

- Per migliorare la coerenza del comportamento dei diversi provider IBM JMS e Jakarta Messaging
- Per semplificare la scrittura di un'applicazione bridge tra due sistemi di messaggistica IBM
- Per semplificare la porta di un'applicazione da un provider IBM JMS o Jakarta Messaging a un altro

Attività correlate
Utilizzo delle classi IBM MQ per JMS/Jakarta Messaging

Supporto per gli oggetti gestiti

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging supportano l'utilizzo di oggetti gestiti.

JM 3.0 Da IBM MQ 9.3.0, Jakarta Messaging 3.0 è supportato per lo sviluppo di nuove applicazioni. IBM MQ 9.3.0 e successive continuano a supportare JMS 2.0 per le applicazioni esistenti. Non è supportato utilizzare sia l'API Jakarta Messaging 3.0 che l'API JMS 2.0 nella stessa applicazione. Per ulteriori informazioni, fare riferimento a Utilizzo delle classi IBM MQ per JMS/Jakarta Messaging.

Il flusso di logica all'interno di un'applicazione JMS o IBM MQ classes for Jakarta Messaging inizia con gli oggetti ConnectionFactory e Destination. L'applicazione utilizza un oggetto ConnectionFactory per creare un oggetto Connection, che rappresenta la connessione attiva dall'applicazione a un server di messaggistica. L'applicazione utilizza l'oggetto Connection per creare un oggetto Session, un contesto a thread singolo per la produzione e l'utilizzo di messaggi. L'applicazione può quindi utilizzare l'oggetto Session e un oggetto Destination per creare un oggetto MessageProducer , che l'applicazione utilizza per inviare messaggi alla destinazione specificata. La destinazione è una coda o un argomento nel sistema di messaggistica ed è incapsulata dall'oggetto Destinazione. L'applicazione può anche utilizzare l'oggetto Session e un oggetto Destination per creare un oggetto MessageConsumer , che l'applicazione utilizza per ricevere i messaggi inviati alla destinazione specificata.

Le specifiche JMS e Jakarta Messaging prevedono che gli oggetti gestiti siano ConnectionFactory e Destination. Un amministratore crea e gestisce oggetti gestiti in un repository centrale e un'applicazione

JMS o Jakarta Messaging richiama tali oggetti utilizzando Java Naming Directory Interface (JNDI). Il repository di oggetti gestiti può variare da un file semplice a una directory LDAP (Lightweight Directory Access Protocol).

IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging supportano l'utilizzo di oggetti gestiti. Un'applicazione può utilizzare tutte le funzioni di IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging esposte tramite IBM MQ senza avere alcuna informazione specifica di IBM MQcodificata nell'applicazione stessa. Questa disposizione fornisce all'applicazione un grado di indipendenza dalla configurazione IBM MQ sottostante.

Per ottenere questa indipendenza, l'applicazione può utilizzare JNDI per richiamare factory di connessione e destinazioni memorizzate come oggetti gestiti e utilizzare solo le interfacce definite nel pacchetto `javax.jms` (JMS 2.0) o `jakarta.jms` (Jakarta Messaging 3.0) per eseguire operazioni di messaggistica.

JMS 2.0 Per JMS 2.0, un amministratore può utilizzare lo strumento di amministrazione IBM MQ JMS **JMSAdmin**o IBM MQ Explorer, per creare e gestire gli oggetti gestiti in un repository centrale.

JM 3.0 Per Jakarta Messaging 3.0, non è possibile gestire JNDI utilizzando IBM MQ Explorer. La gestione JNDI è supportata dalla variante Jakarta Messaging 3.0 di **JMSAdmin**, che è **JMS30Admin**.

Un server delle applicazioni generalmente fornisce il proprio repository per gli oggetti gestiti e i propri strumenti per la creazione e la gestione degli oggetti. Un'applicazione Java EE JM 3.0 o Jakarta EE può quindi utilizzare JNDI per richiamare gli oggetti gestiti dal repository del server delle applicazioni o da un repository centrale.

Attività correlate

Configurazione delle risorse JMS e Jakarta Messaging

Tipi di comunicazione supportati su piattaforme Java EE e Jakarta EE

Sulle piatteforme Java EE e Jakarta EE , IBM MQ classes for JMS e IBM MQ classes for Jakarta Messaging supportano due tipi di comunicazione tra un componente di un'applicazione e un gestore code IBM MQ .

JM 3.0 IBM MQ 9.3.0 ha introdotto il supporto per Jakarta Messaging 3.0. JMS 2.0 è ancora completamente supportato. Poiché JMS e Jakarta Messaging condividono molto in comune, ulteriori riferimenti a JMS in questo argomento possono essere considerati come riferimenti ad entrambi. Eventuali differenze vengono evidenziate come necessario.

Sono supportati i seguenti due tipi di comunicazione tra un componente di un'applicazione e un gestore code IBM MQ :

- Comunicazione in uscita
- Comunicazione in ingresso

Comunicazione in uscita

Utilizzando direttamente l'API JMS o Jakarta Messaging , un componente dell'applicazione crea una connessione a un gestore code, quindi invia e riceve messaggi.

Ad esempio, il componente dell'applicazione pu essere un client dell'applicazione, un servlet, un JSP (Java Server Page), un EJB (enterprise Java bean) o un MDB (message driven bean). In questo tipo di comunicazione, il contenitore del server delle applicazioni fornisce solo funzioni di basso livello a supporto delle operazioni di messaggistica, come il pool di connessioni e la gestione thread.

Comunicazione in ingresso

Nel caso di una comunicazione in entrata, un messaggio che arriva a una destinazione viene consegnato a un MDB, che quindi elabora il messaggio.

Le applicazioni Java EE JM 3.0 e Jakarta EE utilizzano MDB per elaborare i messaggi in modo asincrono. Un MDB funge da listener di messaggi JMS e viene implementato da un metodo onMessage(), che definisce il modo in cui un messaggio viene elaborato. Un MDB viene distribuito nel contenitore EJB di un application server. Il modo preciso in cui un MDB è configurato dipende dal server delle applicazioni che si sta utilizzando, ma le informazioni di configurazione devono specificare a quali gestori code connettersi, come connettersi al gestore code, quale destinazione monitorare per i messaggi e il comportamento transazionale di MDB. Queste informazioni vengono quindi utilizzate dal contenitore EJB. Quando un messaggio che soddisfa i criteri di selezione dell'MDB arriva alla destinazione specificata, il contenitore EJB utilizza IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging per richiamare il messaggio dal gestore code e, quindi, consegna il messaggio all'MDB richiamando il relativo metodo onMessage().

Relazione con IBM MQ classes for Java

IBM MQ classes for Java, IBM MQ classes for Jakarta Messaging e IBM MQ classes for JMS sono peer che utilizzano un'interfaccia Java comune per MQI.

Figura 55 a pagina 163 mostra la relazione tra IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging e IBM MQ classes for Java.

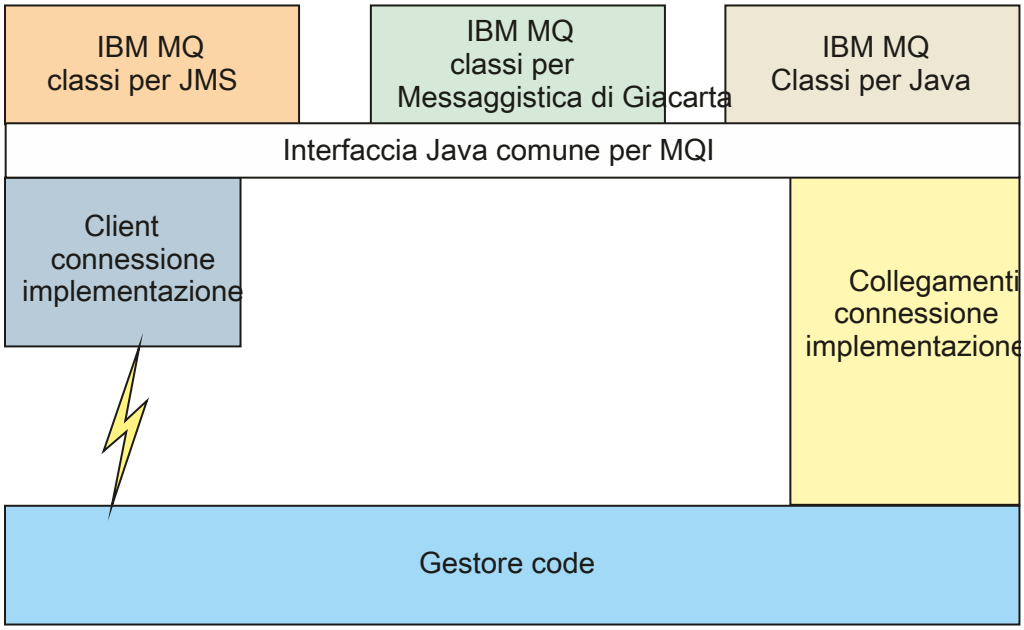


Figura 55. La relazione tra IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging e IBM MQ classes for Java

In generale, i programmi Java devono utilizzare una sola interfaccia per interfacciarsi con IBM MQ - IBM MQ classes for Java, IBM MQ classes for Jakarta Messaging o IBM MQ classes for JMS. La combinazione di interfacce non è supportata, con un'eccezione. Per mantenere la compatibilità con le release precedenti a IBM WebSphere MQ 7.0, le classi di uscita del canale scritte in Java possono ancora utilizzare le interfacce IBM MQ classes for Java , anche se le classi di uscita del canale vengono richiamate da IBM MQ classes for JMS. Tuttavia, l'utilizzo delle interfacce IBM MQ classes for Java significa che le applicazioni dipendono ancora da:

- JMS 2.0 Il file JAR IBM MQ classes for Java , `com.ibm.mq.jar`. Se non si desidera `com.ibm.mq.jar` nel percorso classe, è possibile utilizzare la serie di interfacce nel pacchetto `com.ibm.mq.exits` .
- JM 3.0 Utilizzo di `com.ibm.mq.jakarta.client.jar`, quando si interagisce con IBM MQ classes for Jakarta Messaging.

Concetti correlati

Perché utilizzare le classi IBM MQ per Jakarta Messaging?

Perché utilizzare le classi IBM MQ per JMS?

Perché utilizzare le classi IBM MQ per Java?

IBM MQ Provider di messaggistica

Il provider di messaggistica IBM MQ ha tre modalità operative: modalità normale, modalità normale con limitazioni e modalità di migrazione.

Il provider di messaggistica IBM MQ ha tre modi di funzionamento:

- Modalità normale del provider di messaggistica IBM MQ
- Modalità normale del provider di messaggistica IBM MQ con restrizioni
- Modalità di migrazione del provider di messaggistica IBM MQ

La modalità normale del provider di messaggistica IBM MQ utilizza tutte le funzioni di un gestore code IBM MQ per implementare JMS. Questa modalità è ottimizzata per utilizzare l'API e la funzionalità JMS 2.0 JM 3.0 o Jakarta Messaging 3.0 .

If:

- Il client specifica una versione del provider 6 su un **ConnectionFactory**, il client si comporta in modo compatibile con il client fornito con IBM WebSphere MQ 6.0. Sono supportate solo le interfacce JMS 1.1 e JMS 2, ma alcune funzionalità JMS 2, come le sottoscrizioni condivise, il ritardo di consegna e l'invio asincrono, sono disabilitate. Non esiste alcuna connessione condivisa.
- Il client specifica una versione del provider 7 su un **ConnectionFactory**, entrambe le interfacce JMS 1.1 e JMS 2 sono completamente supportate.
- Non è stata specificata alcuna versione del provider, viene effettuato un tentativo di connessione con il provider versione 7. Se ciò non riesce, viene effettuato un ulteriore tentativo con il fornitore versione 6.

Se si desidera connettersi a IBM Integration Bus utilizzando IBM MQ Enterprise Transport, utilizzare la modalità di migrazione. Se si utilizza IBM MQ Real-Time Transport, la modalità di migrazione viene selezionata automaticamente perché sono state selezionate esplicitamente le proprietà nell'oggetto factory di connessione. La connessione a IBM Integration Bus utilizzando IBM MQ Enterprise Transport segue le regole generali per selezionare la modalità descritte in Configurazione della proprietà JMS **PROVIDERVERSION**.

Attività correlate

Configurazione delle risorse 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 165
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 166
The channel initiator provides and manages resources that enable IBM MQ distributed queuing. IBM MQ uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

"Terms and tasks for managing IBM MQ for z/OS" on page 168
Use this topic as an introduction to the terminology, and tasks that are specific to IBM MQ for z/OS.

"Shared queues and queue sharing groups" on page 171
You can use shared queues and queue sharing groups, to implement high availability of IBM MQ resources. Shared queues and queue sharing groups are functions unique to IBM MQ for z/OS on the z/OS platform.

"Intra-group queuing" on page 215
This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

"Storage management on z/OS" on page 228
IBM MQ for z/OS requires permanent and temporary data structures and uses page sets and memory buffers to store this data. These topics give more details on how IBM MQ uses these page sets and buffers.

"Logging in IBM MQ for z/OS" on page 232
IBM MQ maintains *logs* of data changes and significant events as they occur. These logs can be used to recover data to a previous state if required.

"Recovery and restart on z/OS" on page 253
Use the links in this topic to find out about the features of IBM MQ for z/OS for restart and recovery.

"Security concepts in IBM MQ for z/OS" on page 269
Use this topic to understand the importance of security for IBM MQ, and the implications of not having adequate security settings on your system.

"Availability on z/OS" on page 275
IBM MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

"Unit of recovery disposition on z/OS" on page 280
Certain transactional applications can use a GROUP, rather than a QMGR, unit of recovery disposition when connected to a queue manager in a queue sharing group (QSG) by specifying the QSG name when they connect instead of the queue manager name. This allows transaction recovery to be more flexible and robust by removing the requirement to reconnect to the same queue manager in the QSG.

Related reference

"System definition on z/OS" on page 243
IBM MQ for z/OS uses many default object definitions, and provides sample JCL to create those default objects. Use this topic to understand these default objects, and the sample JCL.

"Monitoring and statistics on IBM MQ for z/OS" on page 279
IBM MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

z/OS The queue manager on z/OS

Before you can let your application programs use IBM MQ on your z/OS system, you must install the IBM MQ for z/OS product and start a queue manager. The queue manager owns and manages the set of resources that are used by IBM MQ.

The queue manager

A *queue manager* is a program that provides messaging services to applications. Applications that use the Message Queue Interface (MQI) can put messages on queues and get messages from queues. The queue manager ensures that messages are sent to the correct queue or are routed to another queue manager. The queue manager processes both the MQI calls that are issued to it, and the commands that are submitted to it (from whatever source). The queue manager generates the appropriate completion codes for each call or command.

The resources managed by the queue manager include:

- Page sets that hold the IBM MQ object definitions and message data
- Logs that are used to recover messages and objects in the event of queue manager failure
- Processor storage
- Connections through which different application environments (CICS, IMS, and Batch) can access the IBM MQ API
- The IBM MQ channel initiator, which allows communication between IBM MQ on your z/OS system and other systems

The queue manager has a name, and applications can connect to it using this name.

Figure 56 on page 166 illustrates a queue manager, showing connections to different application environments, and the channel initiator.

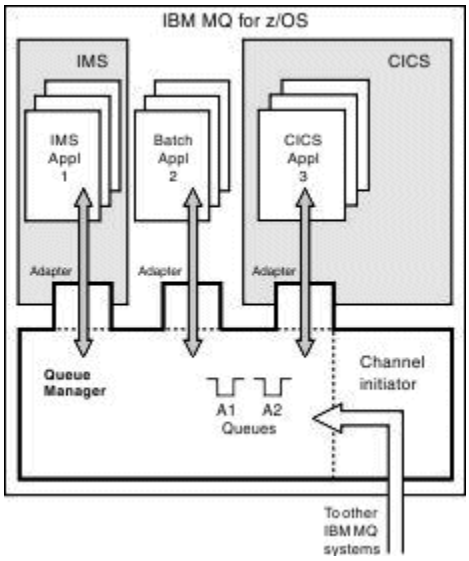


Figure 56. Overview of IBM MQ for z/OS

The queue manager subsystem on z/OS

On z/OS, IBM MQ runs as a z/OS subsystem that is started at IPL time. In the subsystem, the queue manager is started by executing a JCL procedure that specifies the z/OS data sets that contain information about the logs, and that hold object definitions and message data (the page sets). The subsystem and the queue manager have the same name, of up to four characters. All queue managers in your network must have unique names, even if they are on different systems, sysplexes, or platforms.

z/OS The channel initiator on z/OS

The channel initiator provides and manages resources that enable IBM MQ distributed queuing. IBM MQ uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

To send messages from queue manager A to queue manager B, a *sending* MCA on queue manager A must set up a communications link to queue manager B. A *receiving* MCA must be started on queue manager B to receive messages from the communications link. This one-way path consisting of the sending MCA, the communications link, and the receiving MCA is known as a *channel*. The sending MCA takes messages from a transmission queue and sends them down a channel to the receiving MCA. The receiving MCA receives the messages and puts them on to the destination queues.

In IBM MQ for z/OS, the sending and receiving MCAs all run inside the channel initiator (the channel initiator is also known as the *mover*). The channel initiator runs as a z/OS address space under the control of the queue manager. There can be only a single channel initiator connected to a queue manager and it is run inside the same z/OS image as the queue manager. There can be thousands of MCA processes running inside the channel initiator concurrently.

Figure 57 on page 167 shows two queue managers within a sysplex. Each queue manager has a channel initiator and a local queue. Messages sent by queue managers on AIX and Windows are placed on the local queue, from where they are retrieved by an application. Reply messages are returned by a similar route.

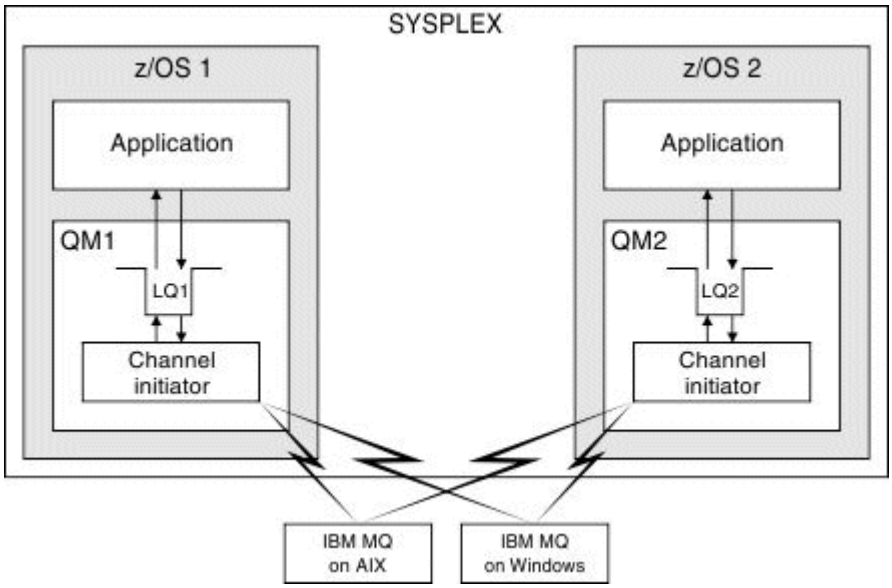


Figure 57. Communication between queue managers

The channel initiator also contains other processes concerned with the management of the channels. These processes include:

Listeners
These processes listen for inbound channel requests on a communications subsystem such as TCP, and start a named MCA when an inbound request is received.

Supervisor
This manages the channel initiator address space, for example it is responsible for restarting channels after a failure.

Name server
This is used to resolve TCP names into addresses.

TLS tasks
These are used to perform encryption and decryption and check certificate revocation lists.

z/OS SMF records for the channel initiator

The channel initiator (CHINIT) can produce SMF statistics records and accounting records with information on tasks and channels.

The CHINIT can produce SMF statistics records and accounting records with the following types of information:

- The tasks: dispatcher, adapter, Domain Name Server (DNS), and SSL. These tasks form what is called CHINIT statistics.
- Channels: provides accounting information similar to that available with the DIS CHSTATUS command. This is called channel accounting.

IBM MQ for Multiplatforms provides similar information by writing PCF messages to the SYSTEM.ADMIN.STATISTICS.QUEUE. See Channel statistics message data for further information on how statistics information is recorded on IBM MQ for Multiplatforms.

Statistics data

You can use this information to find out the following information:

- Whether you need more of the CHINIT tasks, such as number of SSL TCBs and how much CPU is used by these tasks.
- The average time for requests on these tasks.
- The longest duration request in the interval, and the time of day this occurred, for DNS and SSL tasks. You can correlate this time of day with problems you may experience with the channel.

Accounting data

You can use this information to monitor channel usage and find out the following information:

- The channels with the highest throughput.
- The rate at which messages were sent, and the rate of sending data in MB/second.
- The achieved batch size. If the achieved batch size is close to the batch size specified for the channel, the channel might be close to its limit for sending messages.

You use the START TRACE and STOP TRACE commands to control the collection of the accounting trace and the statistics trace. You can use the STATCHL and STATACLS options on the channel and queue manager to control whether channels produce SMF data.

z/OS Terms and tasks for managing IBM MQ for z/OS

Use this topic as an introduction to the terminology, and tasks that are specific to IBM MQ for z/OS.

Some of the terms and tasks required for managing IBM MQ for z/OS are specific to the z/OS platform. The following list contains some of these terms and tasks.

- Shared queues
- Page sets and buffer pools
- Logging
- Tailoring the queue manager environment
- Restart and recovery
- Security
- Availability
- Manipulating objects
- Monitoring and statistics
- Application environments

Shared queues

Queues can be *non-shared*, owned by and accessible to only one queue manager, or *shared*, owned by a *queue sharing group*. A queue sharing group consists of a number of queue managers, running within a single z/OS sysplex, that can access the same IBM MQ object definitions and message data concurrently. Within a queue sharing group, the shareable object definitions are stored in a shared Db2 database. The shared queue messages are held inside one or more coupling facility structures (CF structures). If the message data is too large to store directly in the structure (more than 63 KB in size), or if the message is large enough that installation-defined rules select it for offloading, the message control information is still stored in the coupling facility entry, but the message data is offloaded to a shared message data set (SMDS) or to a shared Db2 database. The shared message data sets, the shared Db2 database, and the coupling facility structures are resources that are jointly managed by all of the queue managers in the group.

Pages sets and buffer pools

When a message is put on to a non-shared queue, the queue manager stores the data on a page set in such a way that it can be retrieved when a subsequent operation gets a message from the same queue. If the message is removed from the queue, space in the page set that holds the data is later freed for reuse. As the number of messages held on a queue increases, so the amount of space used in the page set increases, and as the number of messages on a queue reduces, the space used in the page set reduces.

To reduce the performance cost of writing data to and reading data from the page sets, the queue manager buffers the updates into processor storage. The amount of storage used to buffer the page set access is controlled through IBM MQ objects called *buffer pools*.

For more information about page sets and buffer pools, see Storage management.

Logging

Any changes to objects held on page sets, and operations on persistent messages, are recorded as log records. These log records are written to a log data set called the *active log*. The name and size of the active log data set is held in a data set called the *bootstrap data set* (BSDS).

When the active log data set fills up, the queue manager switches to another log data set so that logging can continue, and copies the content of the full active log data set to an *archive log* data set. Information about these actions, including the name of the archive log data set, is held in the bootstrap data set. Conceptually, there is a ring of active log data sets that the queue manager cycles through; when an active log is filled, the log data is offloaded to an archive log, and the active log data set is available for reuse.

For more information about the log and bootstrap data sets, see “Logging in IBM MQ for z/OS” on page 232.

Tailoring the queue manager environment

When the queue manager is started, a set of initialization parameters that control how the queue manager operates are read. In addition, data sets containing IBM MQ commands are read, and the commands they contain are executed. Typically, these data sets contain definitions of the system objects required for IBM MQ to run, and you can tailor these to define or initialize the IBM MQ objects necessary for your operating environment. When these data sets have been read, any objects defined by them are stored, either on a page set or in Db2.

For more information about initialization parameters and system objects, see “System definition on z/OS” on page 243.

Recovery and restart

At any time during the operation of IBM MQ, there might be changes held in processor storage that have not yet been written to the page set. These changes are written out to the page set that is the least recently used by a background task within the queue manager.

If the queue manager terminates abnormally, the recovery phase of queue manager restart can recover the lost page set changes because persistent message data is held in log records. This means that IBM MQ can recover persistent message data and object changes right up to the point of failure.

If a queue manager that is a member of a queue sharing group encounters a coupling facility failure, the persistent messages on that queue can be recovered only if you have backed up your coupling facility structure.

For more information about recovery and restart, see "Recovery and restart on z/OS" on page 253.

Security

You can use an external security manager, such as Security Server (previously known as RACF) to protect the resources that IBM MQ owns and manages from access by unauthorized users. You can also use Transport Layer Security (TLS) for channel security. TLS is included as part of the IBM MQ product.

For more information about IBM MQ security, see "Security concepts in IBM MQ for z/OS" on page 269.

Availability

There are several features of IBM MQ that are designed to increase system availability in the event of queue manager or communications subsystem failure. For more information about these features, see "Availability on z/OS" on page 275.

Manipulating objects

When the queue manager is running, you can manipulate IBM MQ objects either through a z/OS console interface, or through an administration utility that uses ISPF services under TSO. Both mechanisms enable you to define, alter, or delete IBM MQ objects. You can also control and display the status of various IBM MQ and queue manager functions.

For more information about these facilities, see Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS.

You can also manipulate IBM MQ objects using the IBM MQ Explorer, a graphical user interface that provides a visual way of working with queues, queue managers, and other objects.

Monitoring and statistics

Several facilities are available to monitor your queue managers and channel initiators. You can also collect statistics for performance evaluation and accounting purposes.

For more information about these facilities, see "Monitoring and statistics on IBM MQ for z/OS" on page 279.

Application environments

When the queue manager has started, applications can connect to it and start using the IBM MQ API. These can be CICS, IMS, Batch, or WebSphere Application Server applications. IBM MQ applications can also access applications on CICS and IMS systems that are not aware of IBM MQ, using the CICS and IMS bridges.

For more information about these facilities, see "IBM MQ and other z/OS products" on page 282.

For information about writing IBM MQ applications, see the following documentation:

- Developing applications

- Using C++
- Using IBM MQ classes for Java

z/OS Shared queues and queue sharing groups

You can use shared queues and queue sharing groups, to implement high availability of IBM MQ resources. Shared queues and queue sharing groups are functions unique to IBM MQ for z/OS on the z/OS platform.

This section describes the attributes and benefits, and offers information about how several queue managers can share the same queues and the messages on those queues.

What is a shared queue?

A shared queue is a type of local queue. The messages on that queue can be accessed by one or more queue managers that are in a sysplex.

A queue sharing group

The queue managers that can access the same set of shared queues form a group called a *queue sharing group*.

Any queue manager can access messages

Any queue manager in the queue sharing group can access a shared queue. This means that you can put a message on to a shared queue on one queue manager, and get the same message from the queue from a different queue manager. This provides a rapid mechanism for communication within a queue sharing group that does not require channels to be active between queue managers.

IBM MQ supports the offloading of messages to Db2 or a shared message data set (SMDS). The offloading of messages of any size is configurable.

Figure 58 on page 172 shows three queue managers and a coupling facility, forming a queue sharing group. All three queue managers can access the shared queue in the coupling facility.

An application can connect to any of the queue managers within the queue sharing group. Because all the queue managers in the queue sharing group can access all the shared queues, the application does not depend on the availability of a specific queue manager; any queue manager in the queue sharing group can service the queue.

This gives greater availability because all the other queue managers in the queue sharing group can continue processing the queue if one of the queue managers has a problem.

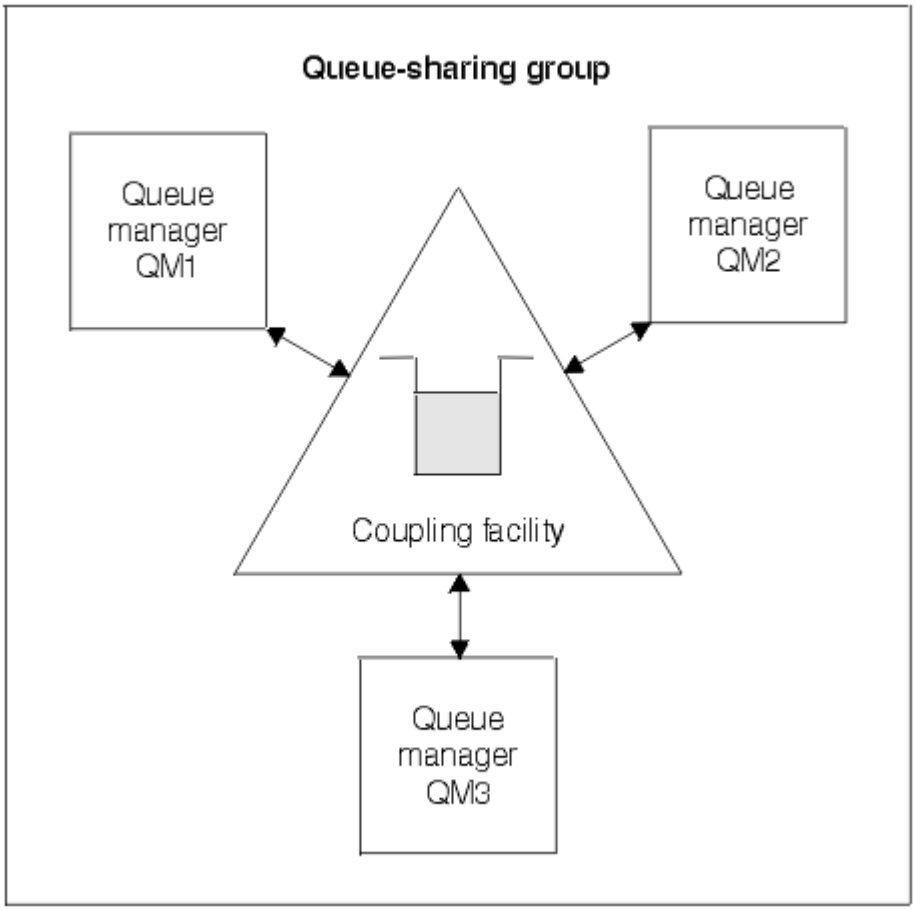


Figure 58. A queue sharing group

Queue definition is shared by all queue managers

Shared queue definitions are stored in the Db2 database table OBJ_B_QUEUE. Because of this, you need to define the queue only once and then it can be accessed by all the queue managers in the queue sharing group. This means that there are fewer definitions to make.

By contrast, the definition of a non-shared queue is stored on page set zero of the queue manager that owns the queue (as described in Page sets).

You cannot define a shared queue if a queue with that name has already been defined on the page sets of the defining queue manager. Likewise, you cannot define a local version of a queue on the queue manager page sets if a shared queue with the same name exists.

What is a queue sharing group?

A group of queue managers that can access the same shared queues is called a queue sharing group. Each member of the queue sharing group has access to the same set of shared queues.

Queue sharing groups have a name of up to four characters. The name must be unique in your network, and must be different from any queue manager names.

Figure 59 on page 173 illustrates a queue sharing group that contains two queue managers. Each queue manager has a channel initiator and its own local page sets and log data sets.

Each member of the queue sharing group must also connect to a Db2 system. The Db2 systems must all be in the same Db2 data-sharing group so that the queue managers can access the Db2 shared repository used to hold shared object definitions. These are definitions of any type of IBM MQ object (for example,

queues and channels) that are defined only once and then any queue manager in the group can use them. These are called *global* definitions and are described in Private and global definitions.

More than one queue sharing group can reference a particular data-sharing group. You specify the name of the Db2 subsystem and which data-sharing group a queue manager uses in the IBM MQ system parameters at startup.

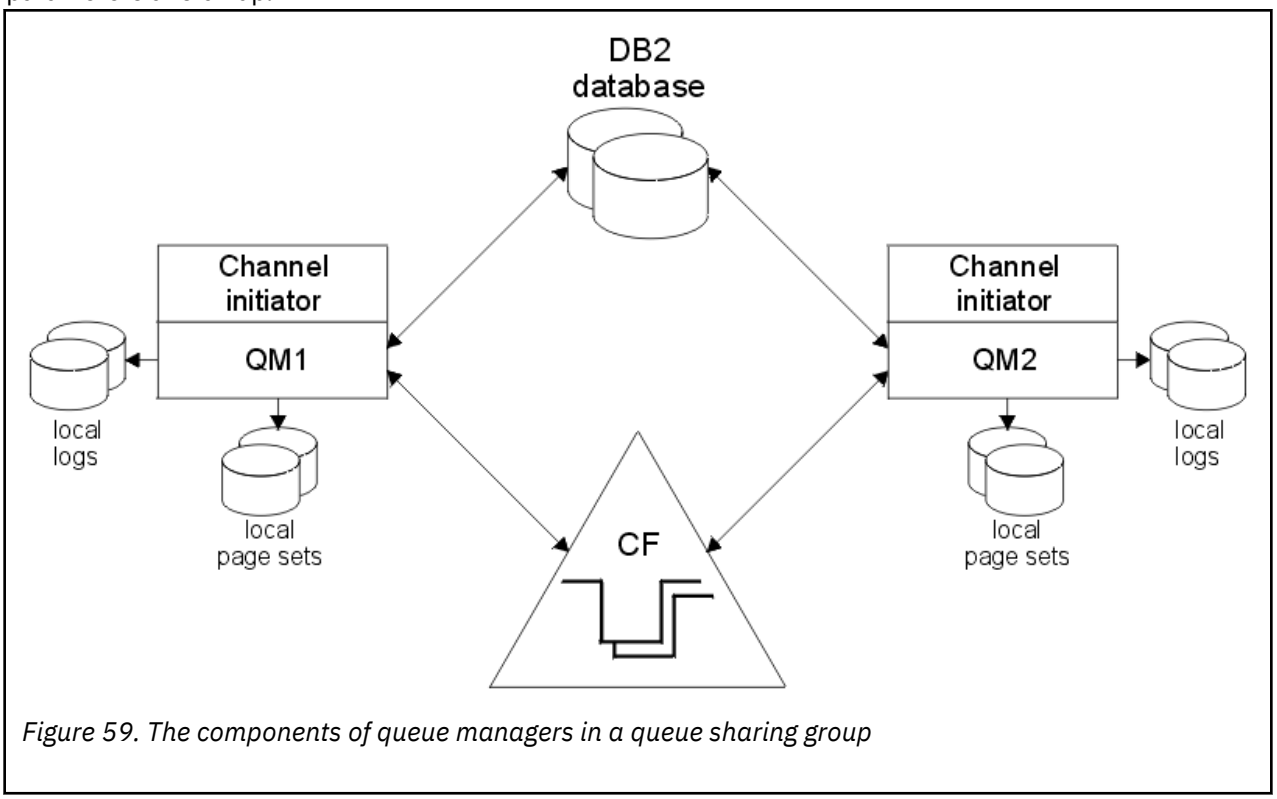


Figure 59. The components of queue managers in a queue sharing group

When a queue manager has joined a queue sharing group, it has access to the shared objects defined for that group, and you can use that queue manager to define new shared objects within the group. If shared queues are defined within the group, you can use this queue manager to put messages to and get messages from those shared queues. Any queue manager in the group can retrieve the messages held on a shared queue.

You can enter an MQSC command once, and have it executed on all queue managers within the queue sharing group as if it had been entered at each queue manager individually. The *command scope* attribute is used for this. This attribute is described in Directing commands to different queue managers.

When a queue manager runs as a member of a queue sharing group it must be possible to distinguish between IBM MQ objects defined privately to that queue manager and IBM MQ objects defined globally that are available to all queue managers in the queue sharing group. The *queue sharing group disposition* attribute is used for this. This attribute is described in Private and global definitions.

You can define a single set of security profiles that control access to IBM MQ objects anywhere within the group. This means that the number of profiles you have to define is greatly reduced.

A queue manager can belong to only one queue sharing group, and all queue managers in the group must be in the same sysplex. You specify which queue sharing group the queue manager belongs to in the system parameters at startup.

Related concepts

"Where are shared queue messages held?" on page 174
Each message on a shared queue is represented by an entry in a z/OS coupling facility list structure. If the message data is too large to fit in the same entry, it is offloaded either to a shared message data set (SMDS) or to Db2.

"Advantages of using shared queues" on page 190

Shared queue allows for IBM MQ applications to be scalable, highly available, and allows workload balancing to be implemented.

“Distributed queuing and queue sharing groups” on page 209
Distributed queuing and queue sharing groups are two techniques that you can use to increase the availability of your application systems. Use this topic to find further information about these techniques.

“Influencing workload distribution with shared queues” on page 213
Use this topic to understand the factors that affect workload distribution with shared queues in a queue sharing group.

Related reference

“Where to find more information about shared queues and queue sharing groups” on page 214
Use the table in this topic to find more information about how IBM MQ for z/OS uses shared queues and queue sharing groups.

z/OS Where are shared queue messages held?

Each message on a shared queue is represented by an entry in a z/OS coupling facility list structure. If the message data is too large to fit in the same entry, it is offloaded either to a shared message data set (SMDS) or to Db2.

If the CF structure has been configured to use System Class Memory (SCM), IBM MQ can use this with no additional configuration.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

Shared queue message storage

Messages that are put onto shared queues are not stored on page sets and do not use buffer pools.

The messages in shared queues have entries on list structures in the z/OS coupling facility (CF). Many queue managers in the same sysplex can access those messages using the CF list structure.

The message data for small shared queue messages is normally included in the coupling facility entry. For larger messages, the message data can be stored either in a shared message data set (SMDS), or as one or more binary large objects (BLOBs) in a Db2 table which is shared by a Db2 data sharing group. Message data exceeding 63 KB is always offloaded to SMDS or Db2. Smaller messages can also optionally be offloaded in the same way to save space in the coupling facility structure. See “Specifying offload options for shared messages” on page 176 for more details.

Messages put on a shared queue are referenced in a coupling facility structure until they are retrieved by an MQGET. Coupling facility operations are used to:

- Search for the next retrievable message
- Lock uncommitted messages on shared queues
- Notify interested queue managers about the arrival of committed messages

MQPUT and MQGET operations on persistent messages are recorded on the log of the queue manager performing that operation. This minimizes the risk of data loss in the event of a coupling facility failure.

The coupling facility

The messages held on shared queues are referenced inside a coupling facility. The coupling facility lies outside any of the z/OS images in the sysplex and is typically configured to run on a different power supply. The coupling facility is therefore resilient to software failures and you can configure it so that it is resilient to hardware failures or power-outages. This means that messages stored in the coupling facility are highly available.

Each coupling facility list structure used by IBM MQ is dedicated to a specific queue sharing group, but a coupling facility can hold structures for more than one queue sharing group. Queue managers in different queue sharing groups cannot share data. Up to 32 queue managers in a queue sharing group can connect to a coupling facility list structure at the same time.

A single coupling facility list structure can contain up to 512 shared queues. The total amount of message data stored in the structure is limited by the structure capacity. However, with **CFLEVEL(5)** you can use the offload parameters to offload data for messages less than 63 KB thereby increasing the number of messages which can be stored in the structure, although each message still requires at least a coupling facility entry plus at least 768 bytes of data, made up of 256 bytes for the entry and 512 bytes for the two elements of header and descriptor.

The size of the list structure is restricted by the following factors:

- It must lie within a single coupling facility.
- It might share the available coupling facility storage with other structures for IBM MQ and other products.

Coupling facility list structures can have storage class memory associated with them. In certain situations this storage class memory can be useful when used with shared queues. See “Use of storage class memory with shared queues” on page 191 for more information.

Planning the CF structure size

If you require guidance on the sizing of your CF structures you can use the MP16: IBM MQ for z/OS Capacity planning and tuning supportpac. You can also use the web-based tool CFSizer, which is provided by IBM to assist with CF sizes.

The CF structure object

The queue manager's use of a coupling facility structure is specified in a CF structure (CFSTRUCT) IBM MQ object.

These structure objects are stored in Db2.

When using z/OS commands or definitions relating to a coupling facility structure, the first four characters of the name of the queue sharing group are required. However, an IBM MQ CFSTRUCT object always exists within a single queue sharing group, and so its name does not include the first four characters of the name of the queue sharing group. For example, CFSTRUCT(MYDATA) defined in queue sharing group starting with SQ03 would use coupling facility list structure SQ03MYDATA.

CF structures have a CFLEVEL attribute that determines their functional capability:

- 1, 2 - can be used for nonpersistent messages less than 63 KB
- 3 - can be used for persistent and nonpersistent messages less than 63 KB
- 4 - can be used for persistent and nonpersistent messages up to 100 MB
- 5 - can be used for persistent and nonpersistent messages up to 100 MB and selectively offloaded to shared message data sets (SMDS) or Db2.

Note: When using IBM MQ you can encrypt a coupling facility structure. See Encrypting coupling facility structure data for more information.

Backup and recovery of the coupling facility

You can back up coupling facility list structures using the IBM MQ command BACKUP CFSTRUCT. This puts a copy of the persistent messages currently within the CF structure onto the active log data set of the queue manager making the backup, and writes a record of the backup to Db2.

If coupling facility fails, you can use the IBM MQ command RECOVER CFSTRUCT. This uses the backup record from Db2 to locate and restore persistent messages from the backup of the CF structure. Any activity since the last backup is replayed using the logs of all the queue managers in the queue sharing group, and the CF structure is then restored up to the point before the failure.

See the BACKUP CFSTRUCT and RECOVER CFSTRUCT commands for more details.

Related concepts

"Specifying offload options for shared messages" on page 176
You can choose where the message data for a shared queue message is stored, either in a Db2 table or a shared message data set (SMDS). You can also select which messages are offloaded, based on the size of the message and the current usage of the coupling facility structure (CF).

"Managing your shared message data set (SMDS) environment" on page 178
If you select shared message data sets to offload large messages then you must also be aware of the information that IBM MQ uses to manage these data sets and the commands used to work with this information. Use this topic to understand how to manage shared message data sets.

z/OS *Specifying offload options for shared messages*

You can choose where the message data for a shared queue message is stored, either in a Db2 table or a shared message data set (SMDS). You can also select which messages are offloaded, based on the size of the message and the current usage of the coupling facility structure (CF).

The message data for shared queues can be offloaded from the coupling facility and stored in either a Db2 table or in an IBM MQ managed data set called a *shared message data set* (SMDS).

For messages larger than the coupling facility entry size of 63 KB, offloading message data to a SMDS can have a significant performance improvement compared with offloading to Db2.

Every shared queue message is still managed using a list entry in a coupling facility structure, but when the message data is offloaded to the SMDS, the coupling facility entry only contains some control information and a list of references to the relevant disk blocks where the message is stored. Using this mechanism means the amount of coupling facility element storage required for each message is only a fraction of the actual size of the message.

Selecting where the shared queue messages are stored

The selection of SMDS or Db2 shared message storage is controlled with the **OFFLOAD(SMDS|DB2)** parameter on the **CFSTRUCT** definition. **OFFLOAD(SMDS)** is the default value.

This parameter also requires the **CFSTRUCT** to use **CFLEVEL(5)** or greater.

The **OFFLOAD** parameter is only valid from **CFLEVEL(5)**. See DEFINE CFSTRUCT for more details.

OFFLOAD(DB2) is supported primarily for migration purposes.

Selecting which shared queue messages are offloaded

Message data is offloaded to SMDS or Db2 based on the size of the message data, and the current usage of the coupling facility structure. There are three rules, and each rule specifies a matching pair of parameters. These parameters are a corresponding coupling facility structure usage threshold percentage (**OFFLDnTH**) and a message size limit (**OFFLDnSZ**).

The current implementation of the three rules is specified using the following pairs of keywords:

- OFFLD1TH and OFFLD1SZ
- OFFLD2TH and OFFLD2SZ
- OFFLD3TH and OFFLD3SZ

Rule pair	Default value	Description
Rule pair 1	OFFLD1TH(70) and OFFLD1SZ(32K)	If the coupling facility structure is more than 70% full offload data for messages exceeding 32 KB
Rule pair 2	OFFLD2TH(80) and OFFLD2SZ(4K)	If the coupling facility structure is more than 80% full offload data for messages exceeding 4 KB
Rule pair 3	OFFLD3TH(90) and OFFLD3SZ(0K)	If the coupling facility structure is more than 90% full offload data for messages exceeding 0 KB (all messages)

If an offload rule has the OFFLD *x* SZ value of 64K this indicates that the rule is not in effect. In this case messages will only be offloaded if another offload rule is in effect, or if the message is greater than 63.75 KB and so, too large to store in the structure.

Each message which is offloaded still requires 0.75 KB of storage in the coupling facility.

The three offload rules which can be specified for each structure are intended to be used as follows.

- Performance
  - When there is plenty of space in the application structure, message data should only be offloaded if it is too large to store in the structure, or if it exceeds some lower message size threshold such that the performance value of storing it in the structure is not worth the amount of structure space that it would need.
  - If a specific message size threshold is required, it is conventionally specified using the first offload rule.
- Capacity
  - When there is very little space in the application structure, the maximum amount of message data should be offloaded so as to make the best use of the remaining space.
  - The third offload rule is conventionally used to indicate that when the structure is nearly full, most messages should be offloaded, so the entries in the application structure will be typically of the minimum size (requiring about 0.75K bytes).
  - The usage threshold parameter should be chosen based on the application structure size and the maximum anticipated backlog. For example, if the maximum anticipated backlog is 1M messages, then the amount of structure storage required for this number of messages is about 0.75G bytes. This means for example that if the structure is about 10G bytes, the usage threshold for offloading all messages must be set to 92% or lower.
  - Structure space is divided into elements and entries, and even though there may be enough space overall, one of these may run out before the other. The system provides AUTOALTER capabilities to adjust the ratio when necessary, but this is not very sensitive, so the amount of space actually available may be somewhat less. It may be better therefore to aim to use not more than 90% of the maximum structure space, so in the previous example, the usage threshold for offloading all messages would be better set around 80%.
- Cushioned transition:
  - As the amount of space left in the coupling facility structure decreases, it would be undesirable to have a large sudden change in the performance characteristics. It is also undesirable for coupling facility management to have a sudden threshold change in the typical ratio of entries to elements being used.
  - The second offload rule is conventionally used to provide some intermediate cushion between the performance and capacity biased offload rules. It can be set to cause a significant increase

in offload activity when the space used in the coupling facility structure exceeds an intermediate threshold. This means that the remaining space is used up more slowly, and gives the coupling facility automatic alter processing more time to adapt to the higher usage levels.

If the coupling facility structure cannot be expanded, and there is a need to store at least some predetermined number of messages, the third rule can be modified as necessary to ensure that offloading of data for all messages starts at an appropriate threshold to ensure space is reserved for that predetermined number of messages.

For example, if the coupling facility structure size is 4 GB, and the predetermined number of messages is 1 million, then 1,000,000 * 0.75 KB are needed, which is 768 MB, 18.75% of 4 GB. In this case the threshold for offloading all messages needs to be set around 80% rather than 90%. This gives parameters OFFLD3TH(80) and OFFLD3SZ(0K) . The other offload parameters would also need to be adjusted.

If it is found that offloading very small messages has a significant performance impact, but the relative impact is less for larger messages, then the usage thresholds for the other rules can be reduced to offload larger messages earlier, leaving more space in the structure for small messages before they need to be offloaded.

For example, if messages exceeding 32KB occur frequently but the elapsed time performance for offloading them (as determined from RMF statistics or application performance) is very similar to that for keeping them in the coupling facility, then the threshold for the first rule could be set to 0% to offload all such messages. This gives parameters OFFLD1TH(0) and OFFLD1SZ(32K). Again the other offload parameters would need to be adjusted.

If there are many messages around specific intermediate sizes, such as 16 KB and 6 KB, then it might be useful to change the message size option for the second rule so that the larger ones get offloaded at a fairly low usage threshold, saving a significant amount of space, but the smaller ones still get stored only in the coupling facility.

z/OS *Managing your shared message data set (SMDS) environment*

If you select shared message data sets to offload large messages then you must also be aware of the information that IBM MQ uses to manage these data sets and the commands used to work with this information. Use this topic to understand how to manage shared message data sets.

SMDS objects

The properties and status of each shared message data set are tracked in a shared SMDS object which can be updated through any queue manager in the queue sharing group.

There is one shared message data set for each queue manager that can access each coupling facility application structure. The shared message data set is identified by the owning queue manager name, specified using the SMDS keyword, and by the application structure name, specified using the CFSTRUCT keyword.

Note: When defining SMDS data sets for a structure, you must have one for each queue manager.

The SMDS object is stored in an array (with one entry per queue manager in the group) which forms an extension of the corresponding CFSTRUCT object stored in Db2.

There is no command to DEFINE or DELETE the SMDS object because it is created or deleted as part of the CFSTRUCT object, but there is a command to ALTER it to change settings for an individual owning queue manager.

For further information on SMDS commands, see “SMDS related commands” on page 189

SMDSCONN information

It is possible for a shared message data set to be in a normal state, but for one or more queue managers to be unable to connect to it, for example because of a problem with a security definition or with direct access device connectivity. It is therefore necessary for each queue manager to keep track of connection

status, and availability information for each shared message data set, indicating for example whether it can currently connect to it, and if not why not.

The SMDSCONN information represents a queue manager connection to a shared message data set. As for the shared message data set itself, it is identified by the queue manager which owns the shared message data set (as specified on the SMDS keyword for the shared object itself) combined with the CFSTRUCT name.

There is no parameter to identify the connecting queue manager because commands addressed to a specific queue manager can only refer to SMDSCONN information for that same queue manager.

The SMDSCONN information entries are maintained in main storage in the owning queue manager, and are re-created when the queue manager is restarted. However, if a connection from an individual queue manager has been explicitly stopped, this information is also stored as a flag in a connection array in the corresponding CFSTRUCT or SMDS object, so that it persists across a queue manager restart.

Status and availability information

Status information indicates the state of a resource or connection (for example, whether it is not yet being used, is in normal use or is in need of recovery). It is usually described using the STATUS keyword. The possible values depend on the type of object.

Status information is normally updated automatically, for example when an error is detected while using the resource or connection. However, in some cases a command can also be used to update the status, to allow for cases when it is not possible for a queue manager to determine the correct status automatically.

Availability information indicates whether the resource or connection can be used, and is usually primarily determined by the status information. For the resource or connection types used in shared message data set support, three levels of availability are implemented:

Available
This means that the resource is available to be used normally. This does not necessarily mean that it is in use at present (which can be determined instead from the STATUS value). For a data set, if it requires restart processing, this allows the owning queue manager to open it, but other queue managers must wait until the data set is back in the ACTIVE state.

Unavailable because of error
This means that the resource has been made unavailable automatically because of an error and is not expected to be available again until some form of repair or recovery processing has been performed. However, attempts to make it available again are permitted without operator intervention. Such an attempt can also be triggered by a command to mark the resource as enabled, or a command which changes the status in such a way as to indicate that recovery processing has been completed.

The reason that the resource has been made unavailable is normally obvious from the related STATUS value, but in some cases there may be other reasons to make the resource unavailable, in which case a separate REASON value is provided to indicate the reason.

Unavailable because of operator command
This means that access to the resource has been explicitly disabled by a command. It can only be made available by using a command to enable it again.

SMDS availability

For the shared SMDS object, the availability is described by the ACCESS keyword, with the possible values ENABLED, SUSPENDED and DISABLED.

The availability can be updated using a **RESET SMDS** command for the relevant shared object from any queue manager in the group to set ACCESS(ENABLED) or ACCESS(DISABLED).

If the availability was previously ACCESS(SUSPENDED), changing it to ACCESS(ENABLED) will trigger a new attempt to use the shared message data set, but if the previous error is still present, the availability will be reset back to ACCESS(SUSPENDED).

SMDSCONN availability

For a local SMDSCONN information entry, the availability is described by the AVAIL keyword, with the possible values NORMAL, ERROR or STOPPED. The availability can be updated using a **START SMDSCONN** or **STOP SMDSCONN** command addressed to a specific queue manager to enable or disable its connection.

If the availability was previously AVAIL(ERROR), changing it to AVAIL(NORMAL) will trigger a new attempt to use the shared message data set, but if the previous error is still present, the availability will be reset back to AVAIL(ERROR).

Shared message data set shared status and availability

The availability of each shared message data set is managed within the group using shared status information, which can be displayed using the **DISPLAY CFSTATUS** command with TYPE(SMDS). This displays status information for each queue manager that has activated a data set for each structure. Each data set can be in one of the following states:

NOTFOUND
This means that the corresponding data set has not yet been activated. This status only appears when a specific queue manager is specified, as data sets which have not been activated are skipped when all queue managers are selected.

NEW
The data set is being opened and initialized for the first time, ready to be made active.

ACTIVE
This means that the data set is fully available and should be allocated and opened by all active queue managers for the structure.

FAILED
This means the data set is not available at all (except for recovery processing) and must be closed and deallocated by all queue managers.

INRECOVER
This means that media recovery (using RECOVER CFSTRUCT) is in progress for this data set.

RECOVERED
This indicates that a command has been issued to switch a failed data set back to the active state, but further restart processing is required which is not yet complete, so the data set can only be opened by the owning queue manager for restart processing.

EMPTY
The data set contains no messages. The data set is put into this state if it is closed normally by the owning queue manager, at a time when it does not contain any messages. It can also be put into EMPTY state when the previous data set contents are to be discarded because the application structure has been emptied (using **RECOVER CFSTRUCT** with TYPE PURGE or, for a nonrecoverable structure only, by deleting the previous instance of the structure). The next time the data set is opened by its owning queue manager, the space map is reset to empty, and the status is changed to ACTIVE. As the previous data set contents are no longer required, a data set in this state can be replaced with a newly allocated data set, for example to change the space allocation or move it to another volume.

The command output includes the date and time at which recovery logging was enabled, if any, and the date and time at which the data set failed, if it is not currently active.

A shared message data set can be put into a FAILED state either by a **RESET SMDS** command or automatically when any of the following types of error are detected:

- The data set cannot be allocated or opened by the owning queue manager.
- Validation of the data set header fails after it has been successfully opened by any queue manager.
- A permanent I/O error occurs when the owning queue manager is reading or writing data.
- A permanent I/O error occurs when another queue manager is reading data from a data set which had successfully completed open processing and validation.

When a data set is in the FAILED or INRECOVER state, it not available for normal use, so if the availability state is ACCESS(ENABLED) it is changed to ACCESS(SUSPENDED).

If a data set has been put into the the FAILED state but no media recovery is required, for example because the data was still valid but the storage device was temporarily offline, then the **RESET SMDS** command can be used to request changing the status directly to the RECOVERED state.

When the data set enters the RECOVERED state, either on completion of recovery processing or as a result of the **RESET SMDS** command, then it is ready to be used again once restart processing has been completed. If it was in the ACCESS(SUSPENDED) state, it is automatically switched back to the ACCESS(ENABLED) state, which allows the owning queue manager to perform restart processing. When restart processing completes, the state is changed to ACTIVE and all other queue managers can then connect to the data set again.

Shared message data set connection status and availability

Each queue manager maintains local status and availability information for its connection to each shared message data set owned by itself and by other queue managers in the group. This information can be displayed using the **DISPLAY SMDSCONN** command.

If it is unable to access a shared message data set in the ACTIVE state which belongs to another queue manager it flags the connection as being unavailable from its own point of view.

If the error definitely indicates a problem with the data set itself, the queue manager also automatically changes the shared status to indicate that the data set is now in a FAILED state. However, if the error could be caused by an environmental problem, such as not being authorized to open the data set, the queue manager issues error messages and treats the data set as being unavailable, but it does not modify the shared data set status. If the environmental error turns out to be a problem with the data set anyway (for example it has been allocated on a device which cannot be accessed by some of the queue managers) then an operator can use the RESET SMDS command specifying STATUS(FAILED) to allow the data set to be recovered or repaired as necessary.

If a connection to a shared message data set could not be established but the data set appears to be valid, a new attempt to use it can be triggered by issuing a **START SMDSCONN** command for the owning queue manager.

If there is an operational need to terminate the connection between a specific queue manager and a data set temporarily, but the data set itself is not damaged, then the data set can be closed and deallocated using the **STOP SMDSCONN** command. If the data set is in use, the queue manager will close it normally (although any requests for data in that data set will be rejected with a return code). If it is the owned data set, the queue manager will save the space map during CLOSE processing, avoiding the need for restart processing.

If a data set needs to be taken out of service temporarily from all queue managers (for example to move it) but is not damaged, then it is best to use **STOP SMDSCONN** for the relevant data set with the option CMDSCOPE(*) to stop the queue managers using it first, as this will avoid the need for restart processing when the data set is brought back into service. In contrast, if the data set is marked as FAILED this tells queue managers that they must stop using it immediately, which means that the space map will not be saved and will need to be rebuilt by restart processing.

Access to any shared message data sets previously in the ACCESS(SUSPENDED) state will be retried if the queue manager is restarted.

Shared message data set recovery logging

Persistent shared messages are logged for media recovery purposes. This means that the messages can be recovered after any failure of coupling facility structures or shared message data sets, provided that the recovery logs are still intact. Persistent messages can also be re-created from the recovery logs at another site for disaster recovery purposes.

When the message data is written to a shared message data set, each block written to the data set is logged separately followed by the message entry (including the data map) as written to the coupling

facility. The recovery process always recovers the coupling facility structure, but it does not need to recover individual shared message data sets except when the data set status is FAILED, or when the status is ACTIVE but the data set header record is no longer valid, indicating that the data set has been re-created. A data set is not selected for recovery if its status is ACTIVE and the data set header is still valid, nor if its status is EMPTY, indicating that no messages were stored in it at the time of the failure.

Shared message data set backups

When BACKUP CFSTRUCT is used to make a backup of the shared messages in an application structure, any data for persistent messages stored in shared message data sets is backed up at the same time, as for persistent shared messages previously stored in DB.

Shared message data set recovery

If a shared message data set is corrupted or lost, then it needs to be put into the FAILED state to stop the queue managers from using it until it has been repaired. This normally happens automatically, but can also be done using the **RESET SMDS** command specifying STATUS(FAILED).

If the shared message data set contained any persistent messages, these can be recovered using the RECOVER CFSTRUCT command. This command first restores any persistent message data for that shared message data set from the most recent BACKUP CFSTRUCT command, then applies all logged changes since that time. If no **BACKUP CFSTRUCT** command has been performed since the time that the data set was first activated, it is reset to empty then all changes since activation are applied.

If the CFSTRUCT contents and all of the shared message data sets are unavailable, for example in a disaster recovery situation, they can all be recovered in a single **RECOVER CFSTRUCT** command.

If a shared message data set is damaged but recovery was not active for the CFSTRUCT, or the log containing the latest BACKUP CFSTRUCT is unavailable or unusable, then the messages offloaded to that data set cannot be recovered. In this case, the **RECOVER CFSTRUCT** command with the parameter TYPE(PURGE) can be used to mark the shared message data set as empty and delete any messages from the structure which had data stored in that data set.

When the **RECOVER CFSTRUCT** command is issued, the shared message data set status is changed from FAILED to INRECOVER. If recovery completes successfully, the status is automatically changed to RECOVERED, otherwise it changes back to FAILED.

When the data set is changed to the RECOVERED state, this tells the owning queue manager that it can now try to open the data set and perform restart processing.

Shared message data set recovery and syncpoints

The shared message data set recovery process reapplies the changes for all complete log records up to the end of the log, regardless of syncpoints.

If changes were made within syncpoint, restart or recovery processing for the CFSTRUCT may result in backing out of uncommitted requests, so some of the recovered changes may not actually be used, but there is no harm in recovering them anyway.

It is also possible that an uncommitted MQPUT message may have been written to the structure but the corresponding data may not have been written to the data set or the log (as I/O completion is only forced at the start of syncpoint processing). This is harmless because restart processing will back out the message entry in the structure, so the fact that it refers to unrecovered data does not matter.

Shared message data set restart processing

If a queue manager connection to a CFSTRUCT terminates normally, the queue manager writes out the free block space map for each shared message data set to a checkpoint area within the data set, just before the data set is closed. The space map can then be read in again at connection restart time, provided that neither the CFSTRUCT nor the shared message data set require any recovery processing before the next restart.

However, if a queue manager terminates abnormally, or the structure or data set require any recovery processing, then additional processing is required to rebuild the space map dynamically when the queue manager connection to the structure is restarted.

Provided that the data set itself did not need to be recovered, queue manager restart simply scans the current contents of the structure to locate references to message data owned by the current queue manager, and marks the relevant data blocks as owned in the space map. Other queue managers can continue to use the structure and read the data owned by the restarting queue manager while the space map is being rebuilt.

Shared message data set restart after recovery

If a shared message data set had to be recovered from a backup, then all nonpersistent messages stored in the data set will have been lost, and if the data set was recovered using TYPE(PURGE) then all messages stored in the data set will have been lost. Until recovery has completed, the data set will be marked as FAILED or INRECOVER so any attempt to read one of the affected messages from another queue manager returns an error code indicating that the data set is temporarily unavailable.

When the data set has been recovered, the status is changed to RECOVERED, which allows the owning queue manager to open it for restart processing, but the data set remains unavailable to other queue managers. Queue manager restart scans the structure to rebuild the space map for any remaining messages. The scan also checks for messages for which the data has been lost, and deletes them from the structure (or if necessary flags them as lost, to be deleted later).

The data set status is automatically changed from RECOVERED to ACTIVE when this restart scan completes, at which point other queue managers can start using it again.

Shared message data set usage information

The DISPLAY USAGE command now also shows information about shared message data set space and buffer pool usage for any currently open shared message data sets. This information is displayed if either the new option TYPE(SMDS) or the existing option TYPE(ALL) is specified.

Shared message data performance and capacity considerations

Monitoring data set usage

The current percentage full of each owned shared message data set can be displayed by the **DISPLAY USAGE** command with the option **TYPE(SMDS)**.

The queue manager will normally automatically expand a shared message data set when it reaches 90% full, provided that the option **DSEXPAND(YES)** is in effect for the SMDS definition. This applies when either the SMDS option is set to **DSEXPAND(YES)** or the SMDS option is set to **DSEXPAND(DEFAULT)** and the CFSTRUCT default option is set to **DSEXPAND(YES)**.

If the expansion attempt fails because no secondary allocation size was specified when the data set was created (giving message IEC070I with reason code 203) the queue manager repeats the expansion request using an override secondary allocation of approximately 20% of the current size.

When a data set is expanded, the new data set extents are formatted as part of the expansion processing, which can take tens of seconds, or even minutes for very large extents. The new space becomes available for use after formatting is complete and the catalog has been updated to show the new high used control interval.

If new messages are being created very rapidly, it is possible for the existing data set to become full before expansion processing completes. In this case, any request which could not allocate space is temporarily suspended until the expansion attempt completes and the new space becomes available for use. If the expansion was successful the request is retried automatically.

If an expansion attempt fails, because of a lack of available space or because the maximum extents have already been reached, a message is issued giving the reason for the failure, then the override option for the affected SMDS is automatically altered to **DSEXPAND(NO)** to prevent further expansion

attempts. In this case, there is a risk that the data set may become full, in which case further action may be needed as described in Data set becomes full.

Monitoring application structure usage

The usage level of an application structure can be displayed using the MVS **DISPLAY XCF,STRUCTURE** command specifying the full name of the application structure (including the queue sharing group prefix). The IXC360I response message shows current usage of elements and entries.

When the structure usage exceeds the **FULLTHRESHOLD** value specified in the CFRM policy, the system issues message IXC585E and may perform automatic **ALTER** actions if specified, which may either alter the entry to element ratio or increase the structure size.

Optimising buffer pool sizes

Each buffer in a shared buffer pool is used to read or write a contiguous range of pages for one message of up to the logical block size. If the message spills over into further blocks, each range of pages in a separate block requires a separate buffer.

Buffers containing message data after a write or read operation are retained in storage and reused using a least-recently-used (LRU) cache scheme so that a request to read the same data again shortly afterwards will not need to go to disk. This provides a significant optimization when shared messages are written and then read back soon afterwards by applications running on the same system. If messages owned by another queue manager are browsed for selection purposes then retrieved, this also avoids the need to reread the message from disk.

This means that the number of buffers required for each application structure is one for each concurrent API request which reads or writes large messages for that application structure plus some number of additional buffers which will be used to save recently accessed data in order to optimize subsequent read accesses.

For shared buffer pools, if there are insufficient buffers, API requests will simply wait if a buffer is not immediately available. However, this situation should be avoided as it can cause significantly degraded performance.

The statistics from the **DISPLAY USAGE** command for shared buffer pools show whether there have been any buffer waits within the current statistics interval, and also shows the lowest number of free buffers (or a negative value indicating the maximum number of threads which waited for a buffer at any time), the number of buffers which have saved data, and the percentage of the times that a buffer request has successfully found saved data on the LRU chain ("LRU hits") instead of having to read it ("LRU misses") [1] .

- If there have been any waits, the number of buffers should be increased.
- If there are many unused buffers, the number of buffers may be reduced to make more storage available in the region for other purposes.
- If there are many buffers containing saved data but the proportion of reads which were hits against that saved data is very small, the number of buffers may be reduced if the storage could be better used for other purposes. The number of buffers should not however be reduced by more than the lowest number of free buffers, as that could trigger waits, and it should preferably be high enough that the lowest free buffer count is normally well above zero.

Deleting shared message data sets

The DELETE CFSTRUCT command (which is only allowed when all shared queues in the structure are empty and closed) does not delete the shared message data sets themselves, but they can be deleted in the usual way after this command has completed. If the same data set is to be reused as a shared message data set, it must be reformatted first to reset it to the empty state.

[1] `(Hits / (Hits+Misses))* 100`

Exception situations for shared message data sets

There are a number of exception situations which can occur during normal use, even when no software or hardware error is present.

Data set becomes full

If a data set becomes full but cannot be expanded, or the expansion attempt fails, applications using the corresponding queue manager to write large messages to the corresponding application structure will receive error 2192, `MQRC_STORAGE_MEDIUM_FULL` (also known as `MQRC_PAGESET_FULL`).

A data set could become full because of a failure in the application which is supposed to process the data, causing a large backlog of messages to accumulate. If so, expanding the data set any further will only be a temporary solution, and it is important to get the processing application going again as soon as possible.

If more space can be made available the **ALTER SMDS** command can then be used to set **DSEXPAND(YES)** or **DSEXPAND(DEFAULT)** (assuming that YES has been set or assumed as the **DSEXPAND** default for the CFSTRUCT definition) to trigger a retry. If the reason for the failure was however that maximum extents had been reached, the new expansion attempt will be rejected with a message and **DSEXPAND(NO)** will be set again. In this case, the only way to expand it any further is to reallocate it, which involves making it temporarily unavailable, as described next.

Data set needs to be moved or reallocated

If a data set needs to be moved or expanded but is otherwise in normal use, it can be taken out of use temporarily to allow it to be moved or reallocated. Any API request which attempts to use the data set while it is unavailable will receive the reason code `MQRC_DATA_SET_NOT_AVAILABLE.`

1. Use the **RESET SMDS** command to mark the data set as **ACCESS(DISABLED)**. This will cause it to be closed normally and deallocated by all currently connected queue managers.
2. Move or reallocate the data set as necessary, copying the old contents to the newly allocated data set, for example using the Access Method Services (AMS) **REPRO** command.

   Do not attempt to preformat the new data set before copying the old data into it, as this would result in the copied data being appended to the end of the formatted data set.
3. Use the **RESET SMDS** command to mark the data set as **ACCESS(ENABLED)** again, to bring it back into use.

   If the old contents are smaller than the size of the new data set, the rest of the space will be preformatted automatically when the new data set is opened.

   If the old contents were larger than the size of the new data set then the queue manager has to scan the messages in the coupling facility structure and rebuild the space map to ensure that none of the active data has been lost. If any reference is found to a data block which is outside the new extents, the data set is marked as **STATUS(FAILED)** and must be repaired by replacing the data set with one of the correct size and either copying the old data set into it again or using **RECOVER CFSTRUCT** to recover any persistent messages.

Coupling facility structure is low on space

If the coupling facility structure is running out of space, causing message IXC585E, it is worth checking whether the offload rules have been set to ensure that the maximum amount of data is being offloaded in this case. If not, the offload rules can be modified using the **ALTER CFSTRUCT** command.

Error situations for shared message data sets

There are a number of problems to be aware of, which can only be caused by errors and not occur in normal operational situations.

Owned data set cannot be opened

If the queue manager which owns a shared message data set cannot allocate it or open it, or the data set attributes are not supported, the queue manager sets an appropriate **SMDSCONN** status value of **ALLOCFAIL** or **OPENFAIL** and sets the **SMDSCONN** availability to **AVAIL(ERROR)**. It also sets the SMDS availability to **ACCESS(SUSPENDED)**. When the error has been corrected, use the **RESET SMDS** command to set **ACCESS(ENABLED)** to trigger a retry, or issue the **START SMDSCONN** command to the owning queue manager.

Read-only data set cannot be opened

If a queue manager cannot allocate or open a shared message data set owned by another queue manager and marked as **STATUS(ACTIVE)**, it assumes that this is probably due to a specific problem with its connection to the data set (represented by the **SMDSCONN** object) rather than a problem with the data set itself.

It marks the **SMDSCONN** as **STATUS(ALLOCFAIL)** or **STATUS(OPENFAIL)** as appropriate and marks the **SMDSCONN** availability as AVAIL(ERROR) to prevent further attempts to use it.

If the problem can been corrected without affecting the status of the data set itself, use the **START SMDSCONN** command to trigger a retry.

If the problem turns out to be a problem with the data set itself, then the **RESET SMDS** command can be used to mark the data set as **STATUS(FAILED)** until it it has been recovered. When the data set has been recovered, the action of changing the status back to **STATUS(ACTIVE)** will cause other queue managers to be notified. If the **SMDSCONN** is marked as **AVAIL(ERROR)**, it will automatically be changed back to **AVAIL(NORMAL)** to trigger a new attempt to open the data set.

Data set header is corrupt

If the data set was successfully opened but the format of the header information is incorrect, the queue manager closes and deallocates the data set and sets the status set to **STATUS(FAILED)** and the availability to **ACCESS(SUSPENDED)**. This allows **RECOVER CFSTRUCT** to be used to recover the contents.

If the error arose because the data set contained residual data from another use and had not been subsequently preformatted, then preformat the data set and use the **RESET SMDS** command to change the status to **STATUS(RECOVERED)**.

Otherwise, the data set must be recovered.

Data set is unexpectedly empty

If the queue manager opens a data set which is marked as **STATUS(ACTIVE)** but finds that it is uninitialized or newly preformatted but otherwise valid, the queue manager closes and deallocates the shared message data set then sets the status to **STATUS(FAILED)** and the availability to **ACCESS(SUSPENDED)**.

Data set has permanent I/O errors

If a data set has permanent I/O errors after successful **OPEN** processing, it probably needs recovery. The queue manager will mark the data set as **STATUS(FAILED)** so that all currently connected queue managers will close and deallocate it.

Data set has recoverable I/O errors

If there are hardware problems with the data set, it is possible that this might result in recoverable I/O errors which are not reflected back to the queue manager but which cause significant performance degradation, and also indicate a risk of permanent I/O errors in the near future.

In this case, the data set may be taken off line for recovery by using the **RESET SMDS** command to mark it as **STATUS(FAILED)**. This will cause it to be closed and deallocated by all queue managers, so for example it could be moved to a new volume before being made available again.

When a data set is made unavailable in this way, the space map is not saved so the queue manager connection restart processing will need to scan the coupling facility structure to locate messages in the data set and rebuild the space map before the data set can be made available again. As an

alternative, if the shared message data set is still usable, it set can be made unavailable more gently by using the **RESET SMDS** command to mark the data set **ACCESS(DISABLED)** until it is ready to be made available again.

Data set contents are incorrect

The queue manager cannot detect directly that a data set contains incorrect data or is not up to date, for example because a volume including that data set had to be restored from backups. However, it performs integrity checks which make it very unlikely that any such errors could result in incorrect message data being seen by application programs.

For integrity checking purposes, each message block in the data set is prefixed with a copy of the corresponding coupling facility entry ID, including a unique time stamp, which is checked whenever the message block is read, before the message data is passed to the user program. If the message block prefix does not match the entry ID (and the coupling facility entry was not deleted in the mean time) the message block is assumed to be damaged and unusable.

If the damaged message was persistent, the data set is marked as **STATUS(FAILED)** and the structure contents must be recovered using the **RECOVER CFSTRUCT** command. If the damaged message was non-persistent, there is no way to recover it, so a diagnostic message is issued and the corresponding coupling facility message entry is deleted.

If no saved space map is available when the data set is opened, it is rebuilt by scanning the coupling facility structure for references to data in the data set. During this scan, the queue manager performs a number of actions:

1. The queue manager determines the location of the most recent message (if any) currently remaining in the data set.
2. The queue manager then reads that message from the data set to ensure that the block prefix matches the message entry id

These actions ensure that the queue manager detects any case where the data set is down-level, and marks the data set as FAILED. This check does however tolerate the case where the data set was restored from a previous copy and either no new messages had been added since then or all messages added since that copy had been subsequently read and deleted.

To protect against down-level data in the case where the data set was closed normally, the queue manager performs a number of actions:

1. The queue manager saves a copy of the space map time stamp in the SMDS object within Db2 when the data set is closed normally.
2. The queue manager then checks the space map time stamp is the same, when the data set is opened again

If the time stamp does not match, this suggests that a down-level copy of the data set might have been used, so the queue manager ignores the existing space map and rebuilds it, which will succeed only if no message data was actually lost.

Note: These integrity checks do not guarantee to detect a down-level or damaged data set in all theoretically possible cases. For example, they will not detect a case where the start of a message block is valid but the rest of the data has been partly overwritten.

Recovery scenarios for shared message data sets

This section described shared message data set recovery scenarios.

Data set recovery where no data was lost

In some cases, the correct contents of a failed data set can be restored without needing actual recovery. One example is where a data set contains residual data from a previous use and has not been preformatted again, which can be fixed by preformatting it. Another case is when a data set has been moved, but there was an error in the process of copying the data across, which can be fixed by copying the data again correctly.

In such cases, the corrected data set can be made available again by using the **RESET SMDS** command to set **STATUS(RECOVERED)**. If the availability is currently **ACCESS(SUSPENDED)** this will automatically set it back to **ACCESS(ENABLED)**.

When the owning queue manager is notified that the data set has been recovered, it scans the structure contents to reconstruct the space map, then changes the status to **STATUS(ACTIVE)**. The other queue managers can then start reading the data set again.

Data set recovery with TYPE(NORMAL)

If the contents of a data set have been lost, but the application structure was defined with **RECOVER(YES)** and the appropriate recovery logs are available, the **RECOVER CFSTRUCT** command can be used to recover any persistent messages stored in the structure including persistent message data offloaded to shared message data sets. This command restores the current state using information logged by the **BACKUP CFSTRUCT** command plus all logged changes to persistent messages since the backup time.

The **RECOVER CFSTRUCT** command always recovers all persistent messages in the coupling facility structure together with offloaded message data stored in Db2. For offloaded data stored in shared message data sets, each data set is only selected for recovery processing if it is already marked as **STATUS(FAILED)** or if it is found to be unexpectedly empty or otherwise invalid when opened by recovery processing. Any shared message data set which is marked as active and which passes the validation checks does not need to be recovered, as the existing message data is already correct, but the header is updated to indicate that any saved space map will need to be rebuilt after recovery.

Recovery processing is only possible when the structure has been marked as failed, as the complete contents of the structure need to be reconstructed by recovery processing. However, if at least one shared message data set has been marked as failed the **RECOVER CFSTRUCT** command will automatically mark the structure as failed if necessary to allow recovery processing to proceed.

Recovery may be performed from any queue manager in the queue sharing group, provided that it has been given write access to the relevant data sets.

Only persistent messages are backed up and logged, so normal recovery processing will restore all persistent messages, but will cause any non-persistent messages in the structure to be lost.

When recovery has completed, any data set which was selected for recovery is automatically changed to **STATUS(RECOVERED)**, and if the availability was **ACCESS(SUSPENDED)** it is changed to **ACCESS(ENABLED)**. The queue manager rebuilds the space map for each data set by scanning the messages in the coupling facility, then marks the data set as **STATUS(ACTIVE)** so that it can be used again.

Data set recovery with TYPE(PURGE)

For a recoverable structure, if the data set contents have been lost, but recovery is not possible for some reason, for example because recovery logs are not available or recovery would take too long, the **RECOVER CFSTRUCT** command can be used with **TYPE(PURGE)** to get the structure back to a usable state. This resets the structure to the empty state and marks all of the associated data sets as **STATUS(EMPTY)**.

Deleting the application structure

If a non-recoverable application structure is deleted using the MVS **SETXCF FORCE** command, or as a result of structure failure, then the next time the structure is connected, message CSQE028I is issued to say that the structure has been reset and all existing messages have been discarded, and any existing data sets are automatically reset to **STATUS(EMPTY)** as well. This action makes a non-recoverable structure usable again after loss of data either in the structure or in any of the associated data sets.

If a recoverable application structure is deleted, it will be treated in the same way as if the structure had failed.

Data set recovery fails

If **RECOVER CFSTRUCT** cannot complete for some reason, for example because a log data set is no longer available, or because the queue manager terminated while recovery was in progress, then any data set for which recovery was at least started will be marked in the header to show that partial recovery has been attempted, and the data set will be left in the **STATUS(FAILED)** state.

In this case, the options are to repeat the original recovery request or to recover with **TYPE(PURGE)** instead, discarding the existing data.

If an attempt is made to mark the data set as **STATUS(RECOVERED)** without actually recovering it, then the next time it is opened the queue manager will see that the header indicates incomplete recovery and mark it as **STATUS(FAILED)** again.

Off site disaster recovery

For off site disaster recovery, persistent shared messages can be re-created using only the logs and the Db2 shared objects containing the CFSTRUCT definitions and associated SMDS status information.

After setting up the Db2 tables containing the definitions, the application structure and the shared message data sets can be set up as empty. When a queue manager connects to them and finds that they are unexpectedly empty, it will mark them as failed, after which a single **RECOVER CFSTRUCT** command can be used to recover all persistent messages for all affected structures.

z/OS *SMDS related commands*

This topic describes and provides access to the commands relating to shared message data sets.

Display and alter the **CFSTRUCT** options relating to large message offload (**OFFLOAD** and offload rules) and shared message data sets (**DSGROUP, DSBLOCK, DSBUFS, DSEXPAND)**:

- DISPLAY CFSTRUCT
- DEFINE CFSTRUCT
- ALTER CFSTRUCT
- DELETE CFSTRUCT

Display **CFSTRUCT** status relating to large message offload **(OFFLDUSE)**:

- DISPLAY CFSTATUS

Display and alter override data set options (**DSEXPAND** and **DSBUFS**) for individual queue managers:

- DISPLAY SMDS
- ALTER SMDS

Display or modify the status and availability of the data sets within the queue sharing group:

- DISPLAY CFSTATUS TYPE(SMDS)
- RESET SMDS

Display SMDS data set space usage and buffer usage information for a queue manager:

- DISPLAY USAGE TYPE(SMDS)

Display or modify the status and availability of the connections (**SMDSCONN**) to the data sets from an individual queue manager:

- DISPLAY SMDSCONN
- START SMDSCONN
- STOP SMDSCONN

Backup and recover shared messages, including large message data in SMDS when necessary:

- BACKUP CFSTRUCT
- RECOVER CFSTRUCT

z/OS *Advantages of using shared queues*

Shared queue allows for IBM MQ applications to be scalable, highly available, and allows workload balancing to be implemented.

The advantages of shared queues

The shared queue architecture, where cloned servers pull work from a single shared queue, has some useful properties:

- It is scalable, by adding new instances of the server application, or even adding a new z/OS image with a queue manager (in the queue sharing group) and a copy of the application.
- It is highly available.
- It naturally performs *pull* workload balancing, based on the available processing capacity of each queue manager in the queue sharing group.

Using shared queues for high availability

The following examples illustrate how you can use a shared queue to increase application availability.

Consider an IBM MQ scenario where client applications running in the network want to make requests of server applications running on z/OS. The client application constructs a request message and places it on a request queue. The client then waits for a reply from the server, sent to the reply-to queue named in the message descriptor of the request message.

IBM MQ manages the transportation of the request message from the client machine to the server's input queue on z/OS and of the response from the server back to the client. By defining the server's input queue as a shared queue, any messages put to the queue can be retrieved on any queue manager in the queue sharing group. This means that you can configure a queue manager on each z/OS image in the sysplex and, by connecting them all to the same queue sharing group, any one of them can access messages on the server's input queue.

Messages on the input queue of the server are still available, even if one of the queue managers terminates abnormally or you have to stop it for administrative reasons. You can take an entire z/OS image offline and the messages will still be available.

To take advantage of this availability of messages on a shared queue, run an instance of the server application on each z/OS image in the sysplex to provide higher server application capacity and availability, as shown in Figure 60 on page 191.

One instance of the server application retrieves a request message from the shared queue and, based on the content, performs its processing, producing a result that is sent back to the client as an IBM MQ message. The response message is destined for the reply-to queue and reply-to queue manager named in the message descriptor of the request message.

There are a number of options that you can use to configure the return path. For more information about these options, see “Distributed queuing and queue sharing groups” on page 209.

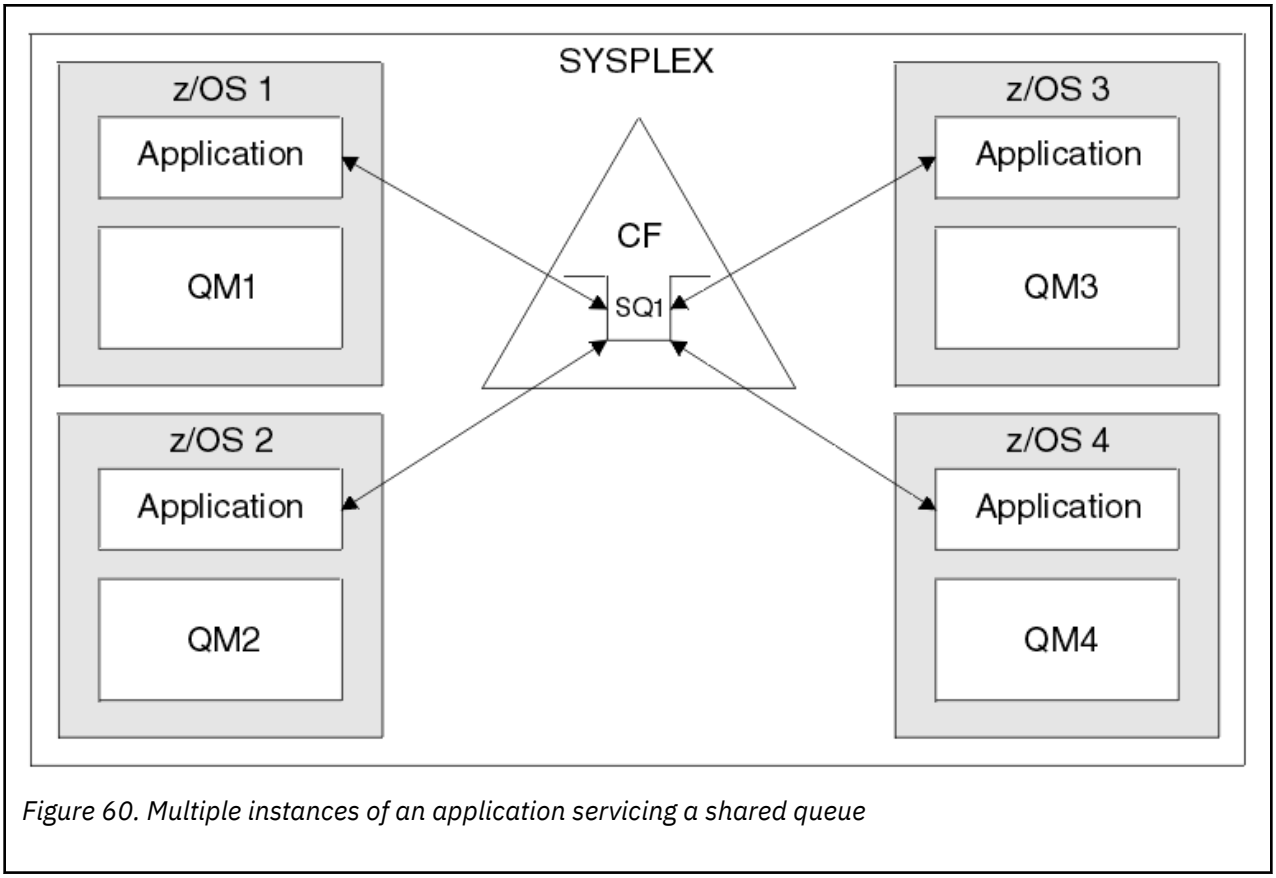


Figure 60. Multiple instances of an application servicing a shared queue

Peer recovery

To further enhance the availability of messages in a queue sharing group, IBM MQ detects if another queue manager in the group disconnects from the coupling facility abnormally and completes units of work for that queue manager that are still pending, where possible. This feature is known as *peer recovery.*

Suppose a queue manager terminates abnormally at a point where an application has retrieved a request message from a queue in sync point, but has not yet put the response message or committed the unit of work. Another queue manager in the queue sharing group detects the failure, and backs out the in-flight units of work being performed on the failed queue manager. This means that the request message is put back on to the request queue and is available for one of the other server instances to process, without waiting for the failed queue manager to restart.

If IBM MQ cannot resolve a unit of work automatically, you can resolve the shared portion manually to enable another queue manager in the queue sharing group to continue processing that work.

z/OS *Use of storage class memory with shared queues*

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

The z13, zEC12, and zBC12 machines allow the installation of Flash Express cards. These cards contain flash solid-state drives (SSD). After installation, flash storage from the cards can be allocated to one or more LPARs where it is typically known as SCM.

SCM sits between real storage and direct access storage device (DASD) in terms of both I/O latency and cost. Because SCM has no moving parts, it exhibits much lower I/O latencies than DASD.

SCM is also much cheaper than real storage. As a result, a large amount of storage can be installed for a relatively low cost; for example, a pair of Flash Express cards contains 1424 GB of usable storage.

These characteristics mean that SCM is useful when a large amount of data must be taken from real storage in a short period of time, because that data can be written to SCM much quicker than it can be written to DASD. This specific point can be very useful when using coupling facility (CF) list structures containing IBM MQ shared queues.

Why list structures fill up

When a CF structure is defined, it is configured with a SIZE attribute that describes the maximum size of the structure. Because CF structures are always permanently resident in real storage, the sum of the SIZE attributes of the structures that are defined on a CF should be less than the amount of the real storage allocated to the CF.

As a result, there is a constant pressure to keep the SIZE value of any given structure to the minimum value possible, so that more structures can fit into the CF. However, ensuring structures are large enough to achieve their purpose can result in a conflicting pressure, because making a structure too small means that it might fill up, disrupting the applications or subsystems making use of it.

There is a strong need to accurately size a structure based on its expected usage. However, this task is difficult to do because workloads can change over time and accounting for their fluctuations is not easy.

IBM MQ shared queues use CF list structures to store messages. IBM MQ calls CF structures, which contain messages and application structures.

Application structures are referenced using the information stored in IBM MQ CFSTRUCT objects. When a message that is smaller than 63 KB is put on a shared queue, the message is stored entirely in an application structure as a single list entry, and zero or more list elements.

Because IBM MQ shared queues use list structures, the pressures described also affect shared queues. In this case, the maximum number of messages that can be stored on a shared queue is a function of the:

- Size of the messages on the queue
- Maximum size of the structure
- Number of entries and elements available in the structure

Because up to 512 shared queues can use the same structure, and effectively compete for entries and elements, this complicates matters even further

IBM MQ queues are used for the transfer of data between applications, so a common situation is an application putting messages to a queue, when the partner application, which should be getting those messages, is not running.

When this happens the number of messages on the queue increase over time until one or more of the following situations occur:

- The putting application stops putting messages.
- The getting application starts getting messages.
- Existing messages on the queue start expiring, and are removed from the queue.
- The queue reaches its maximum depth in which case an MQRC_Q_FULL reason code is returned to the putting application.
- The structure containing the shared queue reaches its maximum size, or the CF containing the structure runs out of available storage. In either case, an MQRC_STORAGE_MEDIUM_FULL reason code is returned to the putting application.

In the last three cases the queue is full. At this point the putting application has a problem because there is nowhere for its messages to go. The putting application typically solves this problem by using one or more of the following solutions:

- Repeatedly retry putting the message, optionally with a delay between retries.
- Put the messages somewhere else, such as a database or a file. The messages can be accessed later and put to the queue as normal.
- Discard the message if it is nonpersistent.

However, for some classes of applications, for example those with a large volume of incoming messages, or no access to a file system, these solutions are not practical. There is a real need to ensure that queues never, or are extremely unlikely to, fill up in the first place and this is especially pertinent to shared queues.

SMDS and offload rules

The offload rules introduced in IBM WebSphere MQ 7.1 provide a way of reducing the likelihood of an application structure filling up.

Each application structure has three rules associated with it, specified using three pairs of keywords:

- OFFLD1SZ and OFFLD1TH
- OFFLD2SZ and OFFLD2TH
- OFFLD3SZ and OFFLD3TH

Each rule specifies the conditions that must be met for message data to be offloaded to the storage mechanism that is associated with the application structure. Two types of storage mechanisms are currently available:

- Db2
- Group of Virtual Storage Access Method (VSAM) linear data sets, which IBM MQ calls a shared message data set (SMDS).

The following example shows the MQSC command to create an application structure named LIST1, using the DEFINE CFSTRUCT command.

This structure has the default offload rules in place, and uses SMDS as the offload mechanism. This means that when the structure is 70% full (OFFLD1TH), all messages that are 32 KB or larger (OFFLD1SZ) are offloaded to SMDS.

Similarly, when the structure is 80% full (OFFLD2TH) all messages that are 4 KB or larger (OFFLD2SZ) are offloaded. When the structure is 90% full (OFFLD3TH) all messages (OFFLD3SZ) are offloaded.

```
DEFINE CFSTRUCT(LIST1)
CFLEVEL(5)
OFFLOAD(SMDS)
OFFLD1SZ(32K) OFFLD1TH(70)
OFFLD2SZ(4K) OFFLD2TH(80)
OFFLD3SZ(0K) OFFLD3TH(90)
```

An offloaded message is stored in the offload medium, and a pointer to the message is stored in the structure. While the offload rules reduce the chance of the structure filling up, by putting less message data in the structure as it runs out of storage, some data is still written to the structure for each message. That is, the pointer to the offloaded message.

Additionally, the offload rules come with a performance cost. Writing a message to a structure is relatively quick and is largely dominated by the time spent to send the request for the write to the CF. The actual writing to the structure is fast, happening at real storage speeds.

Writing a message to SMDS is much slower because it includes writing to the structure for the message pointer, and writing the message data to SMDS. This second write operation is done at DASD speed and has the potential to add latency. If Db2 is used as the offload mechanism the performance cost is much greater.

z/OS *How storage class memory works with IBM MQ for z/OS*
An overview of the use of storage class memory (SCM) with IBM MQ for z/OS shared queues.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

A coupling facility (CF) that is at CFLEVEL 19, or greater, can have SCM allocated to it. The structures defined in that CF can then be configured to make use of SCM to reduce the chances of the structures filling up (known as a structure full condition). When a structure configured to make use of SCM fills up past a system-determined point, the CF starts moving data from the structure into SCM, which frees space in the structure for new data.

Note: Because SCM itself can fill up, allocating SCM to a structure reduces only the likelihood of a structure full condition, but does not entirely remove the chance of one occurring.

A structure is configured to use SCM by specifying both the **SCMALGORITHM** and **SCMMAXSIZE** keywords in the coupling facility resource manager (CFRM) policy, containing the definition of that structure.

Note that after these keywords are specified, and the CFRM policy is applied, the structure must be rebuilt, or deallocated so that they can take effect.

SCMALGORITHM keyword

Because the input/output speed of SCM is slower than that of real storage, the CF uses an algorithm that is tailored to the expected use of the structure in order to reduce the impact of writing to, or reading from, SCM.

The algorithm is configured by the **SCMALGORITHM** keyword in the CFRM policy for the structure, using the *KEYPRIORITY1* value. Note that you should use the *KEYPRIORITY1* value only with list structures used by IBM MQ shared queues.

The *KEYPRIORITY1* algorithm works by assuming that most applications will get messages from a shared queue in priority order; that is, when an application gets a message, it gets the oldest message with the highest priority.

When a structure starts to fill past the system-defined threshold of 90%, the CF starts asynchronously migrating messages that are least likely to be got next. These are messages with lower priorities that were more recently put on the queue.

This asynchronous migration of messages from the structure into SCM is known as "pre-staging".

Pre-staging reduces the performance cost of using SCM because it reduces the chance of an application being blocked during the occurrence of synchronous input/output to SCM.

In addition to pre-staging, the *KEYPRIORITY1* algorithm also asynchronously brings back messages from SCM and into the structure when sufficient free space is available. For the *KEYPRIORITY1* algorithm, this means that when the structure is less than or equal to 70% full.

The act of bringing messages from SCM into the structure is known as "pre-fetching".

Pre-fetching reduces the likelihood of an application trying to get a message that has been pre-staged to SCM and having to wait while the CF synchronously brings back the message into the structure.

SCMMAXSIZE keyword

The **SCMMAXSIZE** keyword defines the maximum amount of SCM that can be used by a structure. Because SCM is allocated to the structure by the CF when it is required, it is possible to specify a **SCMMAXSIZE** that is greater than the total amount of free SCM available. This is known as "over-committing".

Important: Never over-commit SCM. If you do, the applications that are relying on it will not obtain the behavior that they expect. For example, IBM MQ applications using shared queues might get unexpected MQRC_STORAGE_MEDIUM_FULL reason codes.

The CF uses various data structures to track its use of SCM. These data structures reside in the real storage that is allocated to the CF and, as a result, reduce the amount of real storage that can be used by structures. The storage used by these data structures is known as "augmented space".

When a structure is configured with SCM, a small amount of real storage is allocated from the CF to the structure known as fixed augmented space. This is allocated even if the structure never actually uses any SCM. As data from the structure is stored into SCM, extra dynamic augmented space will be allocated from the spare real storage in the CF.

When the data is removed from SCM, the dynamic augmented space is returned to the CF. Augmented space, either fixed or dynamic, is never taken from the real storage that is allocated to a structure.

In addition to augmented storage, when a structure is configured to use SCM, the amount of control storage used by that structure increases. This means that a list structure configured with SCM can contain fewer entries and elements than a structure of the same size without SCM being configured.

To understand the impact of SCM on new or existing structures, use the CFSizer tool.

A final important point to note is that after data is moved from the structure into SCM, and dynamic augmented space has been used, the structure cannot be altered either manually or automatically.

That is, the amount of storage allocated to the structure cannot be increased or decreased, the entry-to-element ratio that is used by the structure cannot be changed, and so on. To make the structure alterable again, the structure must not have any data stored in SCM and must not be making use of dynamic augmented storage.

z/OS *Why use SCM*

Emergency storage and improved performance are two use cases for using SCM with IBM MQ for z/OS.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This section introduces the theory behind the two possible scenarios. For further details on how you set up the scenarios, see:

- “Emergency storage - basic configuration” on page 198
- “Improved performance - basic configuration” on page 204

Important: The use of SCM with CF structures is not dependent on any specific version of IBM MQ. However the emergency storage scenario works only with IBM WebSphere MQ 7.1 and later, because it requires SMDS and the offload rules.

Emergency storage

SMDS and message offloading can be used in conjunction with SCM to reduce the likelihood of an MQRC_STORAGE_MEDIUM_FULL reason code being returned to an IBM MQ application during an extended outage.

Overview

A single shared queue is configured on an application structure. The putting application puts messages onto the shared queue; the getting application gets messages from the shared queue.

During normal running, the queue depth is expected to be close to zero, but a business requirement indicates that the system must be able to tolerate a two-hour outage of the getting application. This means that the shared queue must be able to contain two hours of messages from the putting application.

Currently, this process is achieved by using the default offload rules, and SMDS, so that the size of the structure is minimized, while reducing the performance cost that is associated with offloading.

The rate of messages being sent to the shared queue is expected to double in the short to medium term. Although the requirement that the system be able to tolerate a two-hour outage still exists, not enough real storage is available in the CF to double the size of the structure.

Because the CF, which contains the application structure, resides on a zEC12 machine, the possibility exists to associate sufficient SCM with the structure to store enough messages so that a two-hour outage can be tolerated.

Consider what happens over a period of time:

1. Initially, the system is in its steady state. Both the putting and getting application are running normally and the queue depth is close to, or at, zero. The result is that the application structure is largely empty.
2. At a certain time, the getting application suffers an unexpected failure and stops. The putting application continues to put messages to the queue and the application structure starts to fill up.
3. After the structure reaches 70% full, the conditions of the first offload rule are met and all messages with a size greater than or equal to 32 KB are offloaded to SMDS.

   See "SMDS and offload rules" on page 193 for an overview of the offload rules.
4. As messages continue to be put to the shared queue, the structure continues to fill (either because of the message data being stored in the structure, or as a result of the pointers to the offloaded messages being stored in the structure).

   When the structure reaches 80% full, the second offload rule starts to apply and messages that are 4 KB or greater are offloaded to SMDS.
5. When the structure is past 90% full, all messages are offloaded to SMDS and only the message pointers are being placed in the structure.

   About this time, the pre-staging algorithm starts to run, and begins moving data from the structure into SCM. Assuming all messages on the queue are the same priority, the newest messages are pre-staged.

   Because all messages are now being offloaded to SMDS, the data being moved into SCM is not actual message data, but instead the pointers to the messages on SMDS.

   As a result, the number of messages that can be stored on the combination of the structure, and the SCM and SMDS associated with the structure, is very large.

   Performance: During this stage of the outage, the putting application can suffer a degree of performance degradation because of having to write to SMDS. In this case, the use of SCM should not be a limiting factor on the putting application in terms of performance. SCM provides extra space to prevent the structure filling up.
6. Eventually the getting application is available again and the outage is over.

   However SCM is still being used by the structure. The getting application starts reading messages off the queue, getting the oldest, highest priority messages first.

   Because these messages were written before the structure started to fill up, they come out entirely from the real storage portion of the structure.
7. As the structure starts to empty, it goes below the threshold at which pre-staging is active, and so pre-staging stops.
8. The structure usage reduces below the point at which the offload rules take effect, so messages are no longer offloaded to SMDS unless they are more than 63 KB.

   At about this time, the pre-fetch algorithm starts moving data from SCM in to the structure. Because the getting application gets messages from the queue in the order expected by the SCM algorithms, messages are brought in before the getting application needs them.

   The result is that the getting application never needs to wait for messages to be brought in synchronously from SCM.

9. As the getting application continues to move down the queue, it starts retrieving messages that were offloaded to SMDS.
10. Finally, the system is in a steady state again. No messages are stored in SCM or SMDS, and the queue depth is close to zero.

Improved performance

This scenario describes using SCM to increase the number of messages that can be stored on a shared queue without incurring the performance cost of using SMDS.

Description

For this scenario, a putting and getting application communicate through a shared queue which is stored in application structure.

The putting application tends to run in bursts, when it puts a large number of messages in a short amount of time. Then, in an extended period of time, it produces no messages at all.

The getting application sequentially processes each message, and performs complex processing on each one. As a result, most of the time the queue depth is zero, except for when the putting application starts to run, where the queue depth starts increasing as messages are being put faster than they are being got.

The queue depth increases until the putting application stops, and the getting application has enough time to process all the messages on the queue.

Notes:

1. In this scenario, the key factor is performance. The messages being sent to the queue are always less than 63 KB and so never need to be offloaded to SMDS.
2. The application structure has been sized so that it is large enough to contain all of the messages that will be placed on it by the putting application in a single "burst."
3. The offload rules must all be disabled so that, even when the structure starts to fill, the messages are not offloaded to SMDS. This is because the performance costs that are associated with writing messages to, and reading messages from, SMDS are deemed unacceptable.

Over time, the number of messages that the putting application send in a burst must increase by several orders of magnitude. Because the getting application must process each message sequentially, the number of messages on the queue increases to the point where the structure fills up.

At this point, the putting application receives a reason code (MQRC_STORAGE_MEDIUM_FULL) when putting a message, and the put operation fails. The putting application can only briefly tolerate periods when it is unable to put messages to the queue. If the period is too long, the application ends.

Assuming that you do not have the time, or skills available, to rewrite either the putting application or getting application, this problem has three possible solutions:

1. Increase the size of the application structure.
2. Add offload rules to the application structure so that messages are offloaded to SMDS as the queue starts to fill up.
3. Associate SCM with the structure.

The first solution is quick to implement, but not enough real storage is available on the CF.

The second solution might also be quick to implement, but the performance impact of offloading to SMDS is considered too significant to use this option.

The third solution, associating SCM with the structure, provides an acceptable balance of cost and performance.

Associating SCM with a structure results in a higher use of real storage in the CF because of the augmented storage that get operations used. However, the actual amount of real storage will be less than the amount used in the first option.

Another consideration is the cost of SCM. However this cost is much cheaper than real storage. These factors combine to make the third option cheaper than the first option.

Although the third option, potentially, might not perform as well as the first option, the pre-fetch and pre-staging algorithms used by the CF can combine to make the differences in performance acceptable, or in some cases negligible.

Certainly the performance can be much better than using SMDS to offload messages.

Consider what happens over a period of time:

1. Initially, the getting application is active and waiting for messages to be delivered to the shared queue. The putting application is not active and the shared queue is empty.
2. At a certain time, the putting application becomes active, and starts putting a large number of messages to the shared queue. The getting application starts getting the messages, but the queue depth rapidly starts to increase because the getting application is slower than the putting application.

   As a result, the application structure starts to fill up.
3. As the time increases, the putting application is still active. The application structure fills up to approximately 90%.

   This is when the SCM pre-staging algorithm starts to move messages from the structure into SCM, freeing space in the structure.

   Because the getting application gets the oldest, highest priority messages from the queue first, it is always getting messages from the structure and does not need to wait for messages to be brought synchronously from SCM in to the structure.
4. The putting application is still active and putting messages to the shared queue. However, the application never receives an MQRC_STORAGE_MEDIUM_FULL reason code, because enough space exists in SCM to store all the messages that do not fit in the structure.
5. Eventually, the putting application stops because it has no more messages to put.

   The pre-staging algorithm stops because the structure falls below 90% in use, and the getting application continues processing the messages in the queue.
6. As the getting application starts to free space in the structure, the pre-fetch algorithm starts to bring messages back from SCM in to the structure.

   Because the getting application processes messages in the order expected by the pre-fetch algorithm, the getting application never becomes blocked waiting for message data to be brought synchronously from SCM in to the structure.
7. Finally, the getting application processes all the messages on the shared queue, and waits until the next message is available. The structure and SCM are empty of messages.

z/OS *Emergency storage - basic configuration*

How you set up a basic scenario for emergency storage on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

SMDS and message offloading can be used in conjunction with SCM to reduce the likelihood of an MQRC_STORAGE_MEDIUM_FULL reason code being returned to an IBM MQ application during an extended outage.

For example, your enterprise has an application that puts messages onto the queue and an application that gets messages from the queue. During normal running, you expect the queue depth to be close to

zero, but a business requirement indicates that the system be able to tolerate a two-hour outage of the application that gets the messages.

This means that the shared queue being used must be able to contain two hours of messages from the putting application. Currently you achieve this by using the default offload rules, and SMDS.

You expect the rate of messages being sent to the shared queue to double in the short to medium term. Although your requirement that the system be able to tolerate a two-hour outage still exists, not enough real storage is available in the CF to double the size of the structure. Because the CF containing the application structure resides on a zEC12 machine, you have the capability of associating sufficient SCM with the structure to store enough messages, so that a two-hour outage can be tolerated

This initial scenario uses a:

- Queue sharing group, IBM1, that contains a single queue manager, CSQ3. In addition to the administration structure, the queue sharing group defined a single application structure, SCEN1.
- Coupling facility (CF) CF01, in which the SCEN1 application structure is stored as the IBM1SCEN1 structure. This structure has a maximum size of 1 GB.
- Single shared queue, SCEN1.Q that the application structure uses.

This configuration is illustrated in Figure 61 on page 199.

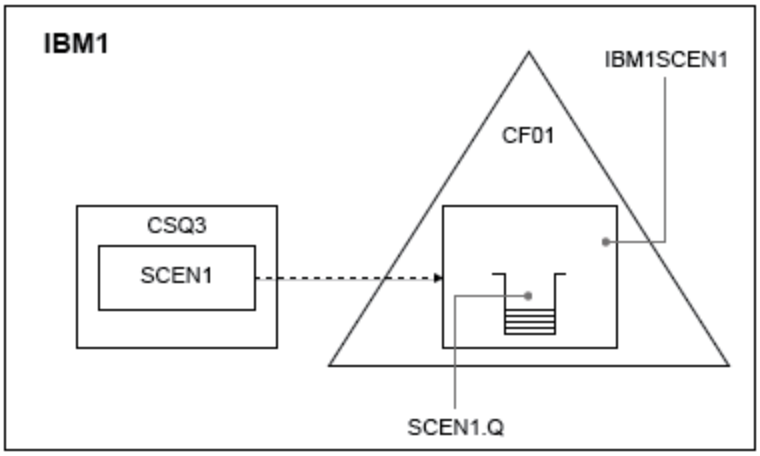


Figure 61. Basic configuration

Furthermore, assume that queue manager CSQ3 is already the only member of queue sharing group IBM1.

You must add the definition for structure IBM1SCEN1 to the coupling facility resource manager (CFRM) policy. For simplicity, the structure is defined so that it can be created in only a single coupling facility, CF01, by specifying PREFLIST(CF01).

Attention: To allow for high availability in your production system, you should include at least two CFs in the PREFLIST for any structures that are used by IBM MQ.

Procedure

1. Refresh the CFRM policy by using the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN1
```

   Sample CFRM policy for structure IBM1SCEN1:

```
STRUCTURE
NAME(IBM1SCEN1)
SIZE(1024M)
INITSIZE(512M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
```

```
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
```

2. Verify that the structure has been created correctly, by using the following command:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

   At this point, your structure has not been allocated, shown by the STATUS line, to the queue sharing group.

3. Configure IBM MQ to make use of the structure defined in the CFRM policy.

   a. Use the DEFINE CFSTRUCT command, with the structure name of SCEN1 to create an IBM MQ CFSTRUCT object:

```
DEFINE CFSTRUCT(SCEN1)
CFCONLOS(TOLERATE)
CFLEVEL(5)
DESCR('Structure for SCM scenario 1')
RECOVER(NO)
RECAUTO(YES)
OFFLOAD(DB2)
OFFLD1SZ(64K) OFFLD1TH(70)
OFFLD2SZ(64K) OFFLD2TH(80)
OFFLD3SZ(64K) OFFLD3TH(90)
```

   .

   b. Validate the structure, using the DISPLAY CFSTRUCT command.

   c. Define the SCEN1.Q shared queue, to use the SCEN1 structure, using the following MQSC command:

```
DEFINE QLOCAL(SCEN1.Q) QSGDISP(SHARED) CFSTRUCT(SCEN1) MAXDEPTH(999999999)
```

4. Use IBM MQ Explorer to put a single message to the queue SCEN1.Q and take the message off again.

5. Issue the following command to check that the structure is now allocated:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

   Check in the output from the command, that the STATUS line shows ALLOCATED.

Results

You have created the basic configuration. You can now obtain an idea of the baseline performance of your configuration using whatever method you select.

What to do next

Add SMDS and SCM to the initial structure

Related concepts

“Use of storage class memory with shared queues” on page 191

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

z/OS *Adding SMDS and SCM to the initial structure*

How you add SMDS and SCM for emergency storage on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This part of the task uses the basic configuration described in “Emergency storage - basic configuration” on page 198. The scenario describes the addition of shared message data sets (SMDS), and then of SCM to the initial structure.

This final configuration is illustrated in Figure 62 on page 201.

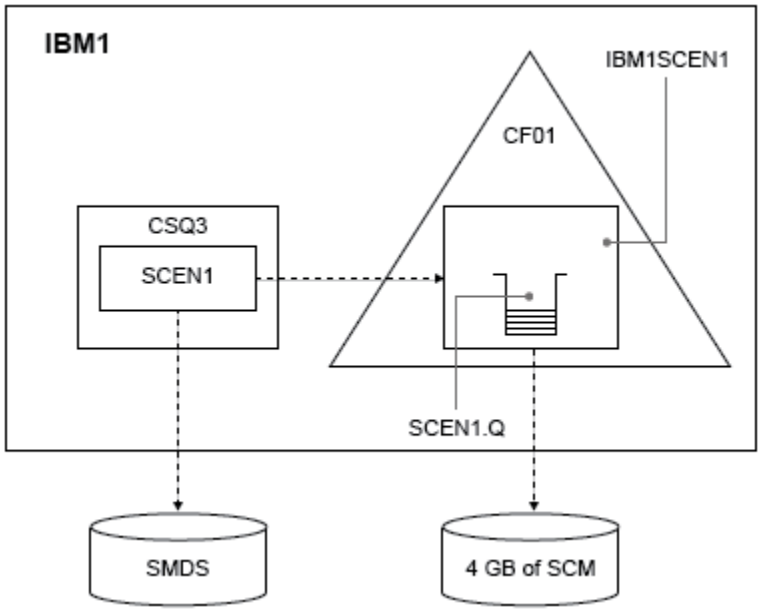


Figure 62. Configuration adding SMDS and SCM for emergency storage

Procedure

1. Create the SMDS data set that the SCEN1 application structure uses, by editing the **CSQ4SMDS** sample JCL, as shown:

```
//CSQ4SMDS JOB NOTIFY=&SYSUID
//*
//* Allocate SMDS
//*
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER           -
(NAME(CSQSMDS.SCEN1.CSQ3.SMDS) -
MEGABYTES(5000 3000)     -
LINEAR             -
SHAREOPTIONS(2 3) )      -
DATA                -
(NAME(CSQSMDS.SCEN1.CSQ3.SMDS.DATA) )
/*
//*
//* Format the SMDS
//*
//FORM   EXEC PGM=CSQJUFMT,COND=(0,NE),REGION=0M
//STEPLIB DD DSN=MQ800.SCSQANLE,DISP=SHR
//     DD DSN=MQ800.SCSQAUTH,DISP=SHR
//SYSUT1  DD DISP=OLD,DSN=CSQSMDS.SCEN1.CSQ3.SMDS
//SYSPRINT DD SYSOUT=*
```

2. Issue the ALTER CFSTRUCT command to change the SCEN1 application structure, to use SMDS for offloading, implementing the default offload rules:

```
ALTER CFSTRUCT(SCEN1) OFFLOAD(SMDS) OFFLD1SZ(32K) OFFLD2SZ(4K) OFFLD3SZ(0K)
DSGROUP('CSQSMDS.SCEN1.*.SMDS') DSBLOCK(1M)
```

   Note the following:

   - Because SCEN1.Q is the only shared queue on the SCEN1 application structure, the **DSBLOCK** value has been set to 1M, the largest value possible. This should be the most efficient setting for our scenario.
   - Because the messages sent by the putting application are 30 KB, offloading to SMDS does not start until the second offload rule is met, when the structure is 80% full.

3. Run your test application again.

   Note the increased storage of messages on the queue.

4. Add 4 GB of SCM to structure IBM1SCEN1 by carrying out the following procedure:

   a) Check how much SCM is installed, and allocated to CF01, by issuing the following command:

```
D CF,CFNAME=CF01
```

   b) Check the STORAGE-CLASS MEMORY figures in the STORAGE CONFIGURATION section of the displayed output to see the available storage.

   c) Update the CFRM policy with the SCMMAXSIZE and SCMALGORITHM keywords as shown:

```
STRUCTURE
NAME(IBM1SCEN1)
SIZE(1024M)
INITSIZE(512M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
SCMMAXSIZE(4G)
SCMALGORITHM(KEYPRIORITY1)
```

5. Activate the CFRM policy by issuing the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=polname
```

6. Rebuild the IBM1SCEN1 structure.

   You must carry out this procedure because the structure was allocated when you made the previous changes.

   Issue the following command to rebuild the structure:

```
SETXCF START,REBUILD,STRNM=IBM1SCEN1
```

Results

You have successfully added SCM to your configuration.

What to do next

Optimize the performance of your system. See “Optimizing storage class memory usage” on page 203 for more information.

z/OS *Optimizing storage class memory usage*

How you improve your use of storage class memory (SCM).

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

Run the following command:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

As the structure was already full with message data, because of the previous tests, part of the rebuild involved pre-staging some of the messages from the structure into SCM. This process was initiated by using the previous command.

The output from this command produces, for example:

```
ACTIVE STRUCTURE
----------------
ALLOCATION TIME: 06/17/2014 09:28:50
CFNAME : CF01
COUPLING FACILITY: 002827.IBM.02.00000000B8D7
PARTITION: 3B CPCID: 00
STORAGE CONFIGURATION ALLOCATED MAXIMUM %
ACTUAL SIZE: 1024 M 1024 M 100
AUGMENTED SPACE: 3 M 142 M 2
STORAGE-CLASS MEMORY: 88 M 4096 M 2
ENTRIES: 120120 1089536 11
ELEMENTS: 240240 15664556 1
SPACE USAGE IN-USE TOTAL %
ENTRIES: 84921 219439 38
ELEMENTS: 2707678 3149050 85
EMCS: 2 282044 0
LOCKS: 1024
SCMHIGHTHRESHOLD : 90
SCMLOWTHRESHOLD : 70
ACTUAL SUBNOTIFYDELAY: 5000
PHYSICAL VERSION: CD5186A0 2BD8B85C
LOGICAL VERSION: CD515C50 CE2ED258
SYSTEM-MANAGED PROCESS LEVEL: 9
XCF GRPNAME : IXCLO053
DISPOSITION : KEEP
ACCESS TIME : NOLIMIT
MAX CONNECTIONS: 32
# CONNECTIONS : 1
CONNECTION NAME ID VERSION  SYSNAME JOBNAME  ASID STATE
--------------- -- -------  ------- -------  ---- ----------------
CSQEIBM1CSQ301  01 00010059 SC61    CSQ3MSTR 0091 ACTIVE
```

Note the following from the output of the command:

- That STORAGE_CLASS MEMORY provides confirmation that a **MAXIMUM** of 4096 MB of SCM has been added to the structure.
- The ALLOCATED figure for the amount of STORAGE-CLASS MEMORY used for pre-staging. There is now free space in the structure where there was none before SCM was added.
- The amount of AUGMENTED SPACE used to track SCM usage.
- The point at which the pre-staging algorithm starts to move data from the structure into SCM is when the structure is 90% full. This is indicated by the non-configurable **SCMHIGHTHRESHOLD** property.
- The point below which the prefetching algorithm starts to move data from SCM into the structure is when the structure is 70% full. This is indicated by the non-configurable **SCMLOWTHRESHOLD** property.

You can now test various ways to optimize the use of SCM. Note the following:

- After SCM is used to store messages, you cannot alter the structure until you have removed all the data from SCM.

  In this case, that means that the entry-to-element ratio is frozen at the value that was in place when SCM was first used. You must carefully ensure that the structure is in the state you want, before the pre-staging algorithm starts moving data into SCM.

- Is the current structure size correct before using SCM?

  For example, have you increased **INITSIZE** from 512 MB to a SIZE of 1 GB?

  If you do not do this, it is possible that although you enabled your structure for auto-alteration, the pre-staging algorithm will start to move data into SCM before the alteration has a chance to start. As a result, the structure is frozen using 512 MB of real storage.
- Is the entry-to-element ratio correct before using SCM?

  The goal of this scenario is to increase the number of offloaded message pointers that can be stored in the structure and SCM as a whole, as well as keeping as many messages entirely in structure storage as possible. Accessing these messages is faster than accessing messages on SMDS.

  Therefore, you need to have a structure that starts with an entry-to-element ratio that is good for storing messages, and then transitions to a ratio that is good for storing message pointers before the prestage algorithm first starts. This transition can be achieved, in part, by making use of the IBM MQ offload rules.

Change the offload rules by issuing the following command:

```
ALTER CFSTRUCT(SCEN1) OFFLD1SZ(0K)
```

You might have to carry out several runs to optimize the entry-to-element ratios.

The following table shows possible improvements in the number of messages put on the queue during the different phases of the emergency storage scenario.

Table 18. Comparison of results for the emergency storage scenario

Test description	Number of messages	Time to fill queue (seconds)
Basic configuration	27,850	3.2
SMDS with default offload rules	205,000	158
SCM with default offload rules	828,610	469
SCM with adjusted offload rules	1,135,775	679

The last row in the table shows that adjusting the offload rules had the required effect.

You need to examine your system to see if you can improve on these figures in any way. For example, you might run out of available SMDS storage. If you can allocate more SMDS storage you should be able to increase the number of messages on the queue quite significantly.

z/OS *Improved performance - basic configuration*

How you set up a basic scenario for improved performance using shared queues on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This scenario describes the use of SCM to increase the number of messages that can be stored on a shared queue without incurring the performance cost of using SMDS.

This initial scenario is very similar to that used for emergency storage and uses a:

- Queue sharing group, IBM1, that contains a single queue manager, CSQ3. In addition to the administration structure, the queue sharing group defined a single application structure, SCEN2.

- Coupling facility (CF) CF01, in which the SCEN2 application structure is stored as the IBM1SCEN2 structure. This structure has a maximum size of 2 GB.
- Single shared queue, SCEN2.Q, which is configured to use the application structure.

This configuration is illustrated in Figure 63 on page 205.

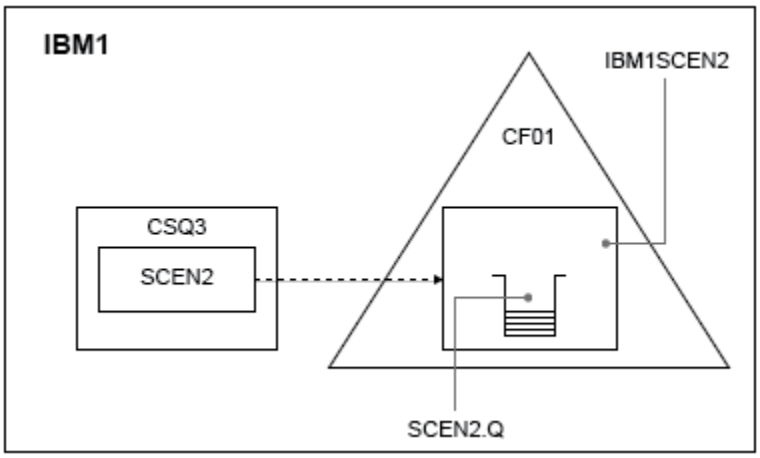


Figure 63. Basic configuration

Furthermore, assume that queue manager CSQ3 is already the only member of queue sharing group IBM1.

You must add the definition for structure IBM1SCEN2 to the coupling facility resource manager (CFRM) policy. For simplicity, the structure is defined so that it can be created in only a single coupling facility, CF01, by specifying PREFLIST(CF01).

Sample CFRM policy for structure IBM1SCEN2:

```
STRUCTURE
NAME(IBM1SCEN2)
SIZE(2048M)
INITSIZE(2048M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
```

Both the **INITSIZE** and **SIZE** keywords have the value 2048M so that the structure cannot resize.

Procedure

1. Refresh the CFRM policy by using the following command:

   ```
   SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN2
   ```

2. Verify that the structure has been created correctly, by using the following command:

   ```
   D XCF,STR,STRNAME=IBM1SCEN2
   ```

   Issuing the preceding command gives the following output:

   ```
   RESPONSE=SC61
   IXC360I 07.58.51 DISPLAY XCF 581
   STRNAME: IBM1SCEN2
   STATUS: NOT ALLOCATED
   POLICY INFORMATION:
   POLICY SIZE : 2048 M
   POLICY INITSIZE: 2048 M
   ```

```
POLICY MINSIZE : 1536 M
FULLTHRESHOLD : 85
ALLOWAUTOALT : YES
REBUILD PERCENT: N/A
DUPLEX : DISABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF01
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY

EVENT MANAGEMENT: MESSAGE-BASED   MANAGER SYSTEM NAME: SC53
MANAGEMENT LEVEL : 01050107
```

At this point, your structure has not been allocated, shown by the STATUS line, to the queue sharing group.

3. Configure IBM MQ to make use of the structure defined in the CFRM policy.
   a. Use the DEFINE CFSTRUCT command, with the structure name of SCEN2 to create an IBM MQ CFSTRUCT object.

```
DEFINE CFSTRUCT(SCEN2)
CFCONLOS(TOLERATE)
CFLEVEL(5)
DESCR('Structure for SCM scenario 2')
RECOVER(NO)
RECAUTO(YES)
OFFLOAD(DB2)
OFFLD1SZ(64K) OFFLD1TH(70)
OFFLD2SZ(64K) OFFLD2TH(80)
OFFLD3SZ(64K) OFFLD3TH(90)
```

   b. Check the structure, using the DISPLAY CFSTRUCT command.
   c. Define the SCEN2.Q shared queue, to use the SCEN2 structure, using the following MQSC command:

```
DEFINE QLOCAL(SCEN2.Q) QSGDISP(SHARED) CFSTRUCT(SCEN2) MAXDEPTH(999999999)
```

4. Use IBM MQ Explorer to put a single message to the queue SCEN2.Q and take the message off again.
5. Issue the following command to check that the structure is now allocated:

```
D XCF,STR,STRNAME=IBM1SCEN2
```

Review the output from the command, a portion of which is shown, and ensure that the STATUS line shows ALLOCATED.

```
RESPONSE=SC61
IXC360I 08.31.27 DISPLAY XCF 703
STRNAME: IBM1SCEN2
STATUS: ALLOCATED
EVENT MANAGEMENT: MESSAGE-BASED
TYPE: SERIALIZED LIST
POLICY INFORMATION:
POLICY SIZE : 2048 M
POLICY INITSIZE: 2048 M
POLICY MINSIZE : 1536 M
FULLTHRESHOLD : 85
ALLOWAUTOALT : YES
REBUILD PERCENT: N/A
DUPLEX : DISABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF01
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY
```

Additionally, note the values of the fields in the SPACE USAGE section:

- ENTRIES
- ELEMENTS

- EMCS
- LOCKS

An example of the values follows:

```
SPACE USAGE   IN-USE      TOTAL    %
ENTRIES:      344686     345242   99
ELEMENTS:    6548455    6548467   99
EMCS:              2     780318    0
LOCKS:          1024
```

Results

You have created the basic configuration. You can now obtain an idea of the baseline performance of your configuration using whatever method you select.

What to do next

You should test the basic scenario. As an example, you can use the following three applications, starting the applications in the order shown and, running them concurrently.

1. Use a PCF application to request the current depth (**CURDEPTH**) value for SCEN2.Q every five seconds. The output can be used to plot the depth of the queue over time.
2. A single threaded getting application repeatedly gets messages from SCEN2.Q, using a get with an infinite wait. To simulate processing of the messages that were removed, the getting application pauses for four milliseconds for every ten messages that it removed.
3. A single threaded putting application puts a total of one million 4 KB non-persistent messages to SCEN2.Q. This application does not pause between putting each message so messages are put on SCEN2.Q faster than the getting application can get them.

   As a result, when the putting application is running, the depth of SCEN2.Q increases.

   When structure IBM1SCEN2 is filled, and the putting application receives a MQRC_STORAGE_MEDIUM_FULL reason code, the putting application sleeps for five seconds before attempting to put the next message to the queue.

You can plot the results of the CURDEPTH application over a period of time. You obtain some form of saw-tooth wave output as the putting application pauses to allow the queue to partially empty.

Go to “Adding SCM to the initial structure” on page 207.

Related concepts

“Use of storage class memory with shared queues” on page 191

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

z/OS *Adding SCM to the initial structure*

How you add SCM for improved performance on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This part of the task uses the basic configuration described in “Improved performance - basic configuration” on page 204. The scenario describes the addition of SCM to the initial structure.

This final configuration is illustrated in Figure 64 on page 208.

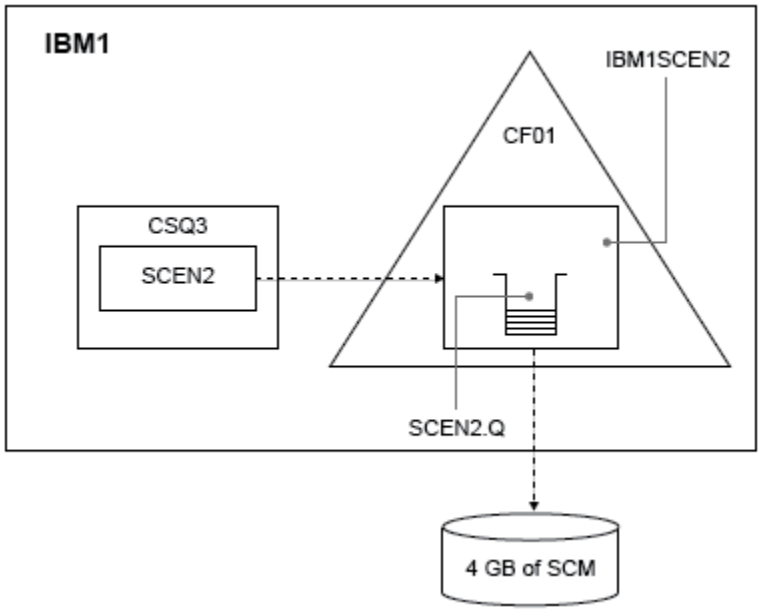


Figure 64. Configuration adding SCM for improved performance

Procedure

1. Add 4 GB of SCM to structure IBM1SCEN2 by carrying out the following procedure:
   a) Check how much SCM is installed, and allocated to CF01, by issuing the following command:

```
D CF,CFNAME=CF01
```

   b) Check the STORAGE-CLASS MEMORY figures in the STORAGE CONFIGURATION section of the displayed output to see the available storage.
   c) Update the CFRM policy with the SCMMAXSIZE and SCMALGORITHM keywords as shown:

```
STRUCTURE
NAME(IBM1SCEN2)
SIZE(2048M)
INITSIZE(2048M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
SCMMAXSIZE(4G)
SCMALGORITHM(KEYPRIORITY1)
```

2. Activate the CFRM policy by issuing the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN2
```

3. Rebuild the IBM1SCEN2 structure.

   You must carry out this procedure because the structure was allocated when you made the previous changes.

   Issue the following command to rebuild the structure:

```
SETXCF START,REBUILD,STRNM=IBM1SCEN2
```

4. Issue the following command to confirm the new configuration of the structure:

```
D XCF,STR,STRNAME=IBM1SCEN2
```

Review the output of the command, a portion of which follows:

```
SPACE USAGE     IN-USE       TOTAL      %
ENTRIES:          33         342684     0
ELEMENTS:         48        6503697     0
EMCS:              2         575600     0
LOCKS:                         1024
```

Results

Calculate the change in the use of real storage by the increase in control storage required to use SCM.

- Before SCM is added to the structure, the structure has these totals as shown in "Improved performance - basic configuration" on page 204:
  - 345,242 entries
  - 6,548,467 elements
  - 780,318 EMCS
- After SCM is added to the structure, the structure has these totals:
  - 342,684 entries
  - 6,503,697 elements
  - 575,600 EMCS

Using these figures, after the SCM was added, the structure is reduced in size by:

- 2558 entries
- 44,770 elements
- 204,718 EMCS

The amount of structure storage that is used to manage SCM, is as follows for a 2 GB structure with 4 GB of SCM allocated:

```
(2558 + 44,770 + 204,718) * 256 = 61.5 MB
```

Note that adding more SCM is likely to achieve only a marginal reduction of the size of the structure, because the amount of control storage used to track SCM increases, both as the structure size, and the amount of allocated SCM increases.

What to do next

Repeat the tests described in the final section of "Improved performance - basic configuration" on page 204.

You can plot the results of the revised application over a period of time. Comparing the plot to the one obtained previously, you now obtain an output without a saw-tooth wave, as the putting application no longer has to wait for the queue to partially empty.

For more information, refer to MP16: WebSphere MQ for z/OS - Capacity planning & tuning.

z/OS Distributed queuing and queue sharing groups

Distributed queuing and queue sharing groups are two techniques that you can use to increase the availability of your application systems. Use this topic to find further information about these techniques.

To complement the high availability of messages on shared queues, the distributed queuing component of IBM MQ has additional functions to provide the following:

- Higher availability to the network.
- Increased capacity for inbound network connections to the queue sharing group.

Figure 65 on page 210 illustrates distributed queuing and queue sharing groups. It shows two queue managers within a sysplex, both of which belong to the same queue sharing group. They can both access shared queue SQ1. Queue managers in the network (on AIX and Windows for example) can put messages onto this queue through the channel initiator of either queue manager. Cloned applications on both queue managers service the queue.

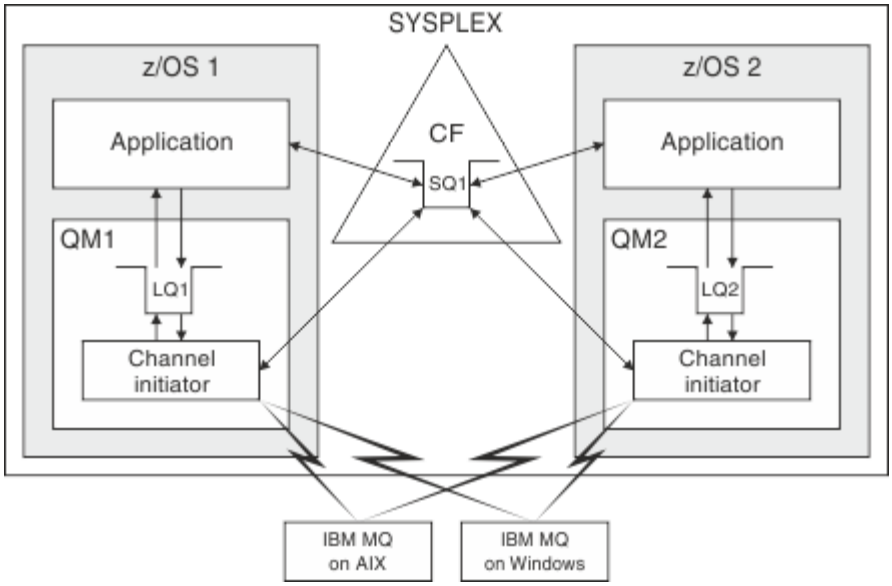


Figure 65. Distributed queuing and queue sharing groups

Related concepts

"Shared channels" on page 210
Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

"Intra-group queuing" on page 215
This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

"Clusters and queue sharing groups" on page 212
Use this topic to understand how you can use queue sharing groups with clusters.

z/OS *Shared channels*

Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

A number of networking products provide a mechanism to hide server failures from the network, or to balance inbound network requests across a set of eligible servers. The network products make a *generic port* available for inbound network connection requests, and the inbound request can be satisfied by connecting to one of the eligible servers.

These networking products include:

- VTAM generic resources
- SYSPLEX Distributor

The channel initiator takes advantage of these products to use the capabilities of shared queues

There are two types of shared channels, *shared inbound channel*, and the *shared outbound channel*.

- Shared inbound channels
- Shared outbound channels

For further information about channels see

- Shared channel summary
- Shared channel status

Shared inbound channels

Each channel initiator in the queue sharing group starts an additional listener task to listen on a *generic port*. This generic port is made available to the network by one of the supporting technologies (VTAM, TCP/IP). Inbound network attach requests to the generic port are dispatched by the network technology, to any one of the listeners in the queue sharing group (QSG) that are listening on the generic port.

You can start a channel on the channel initiator to which the inbound attach is directed if the channel initiator has access to a channel definition for a channel with that name. You can define a channel definition to be private to a queue manager or stored on the shared repository and so available anywhere (a global definition). This means that you can make a channel definition available on any channel initiator in the queue sharing group by defining it as a global definition.

There is an additional difference when starting a channel through the generic port; channel synchronization is with the queue sharing group and not with an individual queue manager. For example, consider a remote queue manager starting a channel through the generic port. When the channel first starts, it might start on queue manager QM1 and messages flow. If the channel stops and is restarted on queue manager QM2, information about the number of messages that have flowed is still correct because the synchronization is with the queue sharing group.

You can use an inbound channel started through the generic port to put messages to any queue. The remote queue manager does not know whether the target queue is shared or not. If the target queue is a shared queue, the remote queue manager connects through any available channel initiator in a load-balanced fashion and the messages are put to the shared queue.

If the target queue is a private queue, the messages are put to the private queue owned by which ever queue manager the current instance of the channel is connected to. In this environment, known as *replicated local queues*, each queue manager must have the same set of private queues defined.

Configuring SVRCONN channels for a queue sharing group

The optimal configuration for SVRCONN channels in a queue sharing group is to set up private listeners in each CHINIT which use a different port number from the point to point channels. These listener ports are then used as the 'back-end' resources for a new workload distribution mechanism such as Sysplex Distributor using Virtual IP addresses (VIPA). The external VIPA address is then used as the target address for the CLNTCONN definitions in the network. The SVRCONN channel can be defined with QSGDISP(GROUP) so the same definition is available to all queue managers in the QSG. This configuration avoids using a shared listener, and therefore reduces the performance effect of the queue sharing group maintaining shared channel state, which is not needed for client/server channels.

Shared outbound channels

An outbound channel is considered to be a shared channel if it is taking messages from a shared transmission queue. If it is shared, it holds synchronization information at queue sharing group level. This means that the channel can be restarted on a different queue manager and channel initiator instance within the queue sharing group if the communications subsystem, channel initiator, or queue manager fails. Restarting failed channels in this way is a feature of shared channels called *peer channel recovery*.

Workload balancing for shared outbound channels

An outbound shared channel is eligible for starting on any channel initiator within the queue sharing group, if you have not specified that you want it to be started on a particular channel initiator. The channel initiator selected by IBM MQ is determined using the following criteria:

- Is the communications subsystem required currently available to the channel initiator?
- Is a Db2 connection available to the channel initiator?
- Which channel initiator has the lowest current workload? The workload includes channels that are active and retrying.

Shared channel summary

Shared channels differ from private channels in the following ways:

Private channel
Tied to a single channel initiator.

- Outbound channel uses a local transmission queue.
- Inbound channel started through a local port.
- Synchronization information held in SYSTEM.CHANNEL.SYNCQ queue.

Shared Channel
Workload balanced with high availability.

- Outbound channel uses a shared transmission queue.
- Inbound channel started through a generic port.
- Synchronization information held in SYSTEM.QSG.CHANNEL.SYNCQ queue.

You specify whether a channel is private or shared when you start the channel by using CHLDISP options with the START CHANNEL command. A shared channel can be started by triggering in the same way as a private channel. However, when a shared channel is started, IBM MQ performs workload balancing and starts the channel on the most appropriate channel initiator within the queue sharing group. (If required, you can specify that a shared channel is to be started on a particular channel initiator.)

Shared channel status

The channel initiators in a queue sharing group maintain a shared channel-status table in Db2. This records which channels are active on which channel initiators. The shared channel-status table is used if there is a channel initiator or communications system failure. It indicates which channels need to be restarted on a different channel initiator in the queue sharing group.

z/OS *Clusters and queue sharing groups*

Use this topic to understand how you can use queue sharing groups with clusters.

You can make your shared queues available to a cluster in a single definition. To do this you specify the name of the cluster when you define the shared queue.

Users in the network see the shared queue as being hosted by each queue manager within the queue sharing group (the shared queue is not advertised as being hosted by the queue sharing group). Clients can start sessions with any members of the queue sharing group to put messages to the same shared queue.

Figure 66 on page 213 shows how members of a cluster can access a shared queue through any member of the queue sharing group.

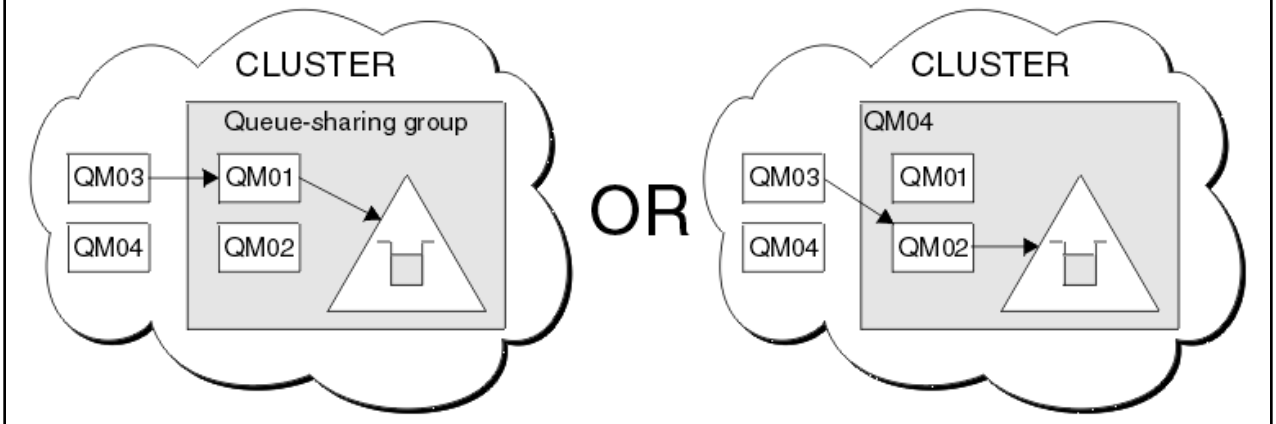


Figure 66. A queue sharing group as part of a cluster

z/OS Influencing workload distribution with shared queues

Use this topic to understand the factors that affect workload distribution with shared queues in a queue sharing group.

IBM MQ does not provide workload balancing for shared queues. However, workload distribution in a queue sharing group (QSG) can be influenced in *a pull based fashion*. The choice of which queue manager services a queue (receives a message written to a shared queue) is affected by the available processing capacity of each queue manager in the queue sharing group and the workload management goals defined across the sysplex.

However, it is important to appreciate, the queue manager that performs the MQPUT of a message can also have a large influence in deciding which queue manager gets the message.

A local queue manager is more likely to perform the MQGET

For an application performing an MQPUT, the local queue manager is said to be the queue manager to which the application is connected.

Exactly which queue manager services an MQPUT of a message by performing an MQGET on behalf of a getting application is influenced by the following considerations.

When a message is put to an empty shared queue, the local queue manager is typically posted before any of the other queue manager in the queue sharing group is notified. If the local queue manager is in a position to process the message, it receives a list transition notification from the coupling facility (CF) before any other queue manager in the QSG. (A list transition notification is a notification that the shared queue has changed state from empty to non-empty.)

The possible scenarios, in this case, are as follows:

1. MQPUT of nonpersistent message out of sync point and *fast put to waiting getter*.

   If there is an application with an *MQGET with wait* on the local queue manager for the queue, then the MQPUT of the message is passed directly to the getting application's buffer and not written to the queue. This is true for shared and non-shared queues. This feature is often called *fast put to a waiting getter* mechanism. In the case of shared queues, no other queue manager in the QSG is notified as there is no transition from empty to non-empty of the queue. This means, for example, that provided this queue manager can service all the puts from this application and assuming that no other applications are putting messages to the queue, then no other queue manager in the queue sharing group assists in draining this queue. If however there is no MQGET with wait on the local queue manager, and a message is put to the shared queue then the CF will notify other queue managers in the queue sharing group according to its rules for notifications of list transitions.
2. MQPUT of a persistent or in-syncpoint message.

In this case, if there is an application with an *MQGET with wait* on the local queue manager, then the message is put to the shared queue and the CF notifies other queue managers in the queue sharing group according to its rules for notifications of list transitions. However, the local queue manager does not wait for a transition notification from the CF but honors any local *MQGET with wait* first and usually performs the get of this message on behalf of the application before any other queue manager in the queue sharing group can respond to a CF notification. This is dependent on how busy the local queue manager is. Otherwise, any queue manager notified by the CF due to the arrival of the message on the empty queue will try to service the get first. The first queue manager to respond processes the new message.

3. Finally, if the queue is not drained of messages, where the CF has sent a notification of a state change from empty to non-empty for the queue, all connected queue managers will have an opportunity to assist in the processing of the queue. In this event, the workload is said to be *pull based*.

This design allows for the improved performance over a purely pull based workload distribution. The aim is to take advantage of the high availability offered by queues held in the CF while allowing the queue manager, where possible, to perform the MQGET without needing to reference the CF and so to process the message workload as efficiently as possible.

Alternative approaches can be adopted where emphasis on balance of the workload is more important than the previously described performance enhancements. For example, ensuring that none of the getting applications are connected to the same queue manager that the putting application is connected to. Using this design all messages are put to the queue and all queue managers in the QSG are notified when the queue moves from empty to non-empty, in accordance with the CF algorithm for handling such transitions. In addition, the *fast put to waiting getter* mechanism is not applicable.

z/OS Where to find more information about shared queues and queue sharing groups

Use the table in this topic to find more information about how IBM MQ for z/OS uses shared queues and queue sharing groups.

Table 19. Where to find more information about shared queues and queue sharing groups

Topic	Where to look
Queue sharing group recovery	“Recovery and restart on z/OS” on page 253
Queue sharing group security	“Security concepts in IBM MQ for z/OS” on page 269
Private and global object definitions Directing commands to different queue managers	Sources from which you can issue commands on z/OS
Planning your coupling facility environment	Defining coupling facility resources
Planning your SMDS environment	Planning your shared message data set (SMDS) environment
Planning your Db2 environment	Planning your Db2 environment
Setting up your shared queues System parameters	“Shared queues and queue sharing groups” on page 171
Utility programs Migrating queues	IBM MQ utilities on z/OS reference

Table 19. Where to find more information about shared queues and queue sharing groups (continued)	
Topic	**Where to look**
Console messages	Messages for IBM MQ for z/OS
MQSC commands	MQSC commands
IBM MQ clusters	Configuring a queue manager cluster
IBM MQ distributed queuing Channel names	Introduction to distributed queue management
Writing applications	Overview of application design
MQCONNX call	MQCONNX

z/OS Intra-group queuing

This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

For information about queue sharing groups, see “Shared queues and queue sharing groups” on page 171.

Intra-group queuing concepts

You can perform fast message transfer between queue managers in a queue sharing group without defining channels. This uses a system queue called the SYSTEM.QSG.TRANSMIT.QUEUE, which is a shared transmission queue. Each queue manager in the queue sharing group starts a task called the intra-group queuing agent, which waits for messages to arrive on this queue that are destined for their queue manager. When such a message is detected, it is removed from the queue and placed on the correct destination queue.

Standard name resolution rules are used but, if intra-group queuing (IGQ) is enabled and the target queue manager is within the queue sharing group, the SYSTEM.QSG.TRANSMIT.QUEUE is used to transfer the message to the correct destination queue manager instead of using a transmission queue and channel.

You enable intra-group queuing through a queue manager attribute. Intra-group queuing moves nonpersistent messages outside sync point, and persistent messages within sync point. If it finds a problem delivering messages to the target queue, intra-group queuing tries to put them to the dead-letter queue. If the dead-letter queue is full or undefined, nonpersistent messages are discarded, but persistent messages are backed out and returned to the SYSTEM.QSG.TRANSMIT.QUEUE, and the IGQ agent tries to deliver the messages until it is successful.

An inbound shared channel that receives a message destined for a queue on a different queue manager in the queue sharing group can use intra-group queuing to *hop* the message to the correct destination.

There might be times when you want the local queue manager to put a message directly to the target queue if the target queue is a shared queue, rather than the message first being transferred to the target queue manager. You can use the queue manager attribute SQQMNAME to control this. If you set the value of SQQMNAME to USE, the MQOPEN command is performed on the queue manager specified by the **ObjectQMgrName**.

However, if the target queue is a shared queue and you set the value of SQQMNAME to IGNORE, and the **ObjectQMgrName** is that of another queue manager in the queue sharing group, the shared queue is opened on the local queue manager. If the local queue manager cannot open the target queue, or put a

message to the queue, the message is transferred to the specified **ObjectQMgrName** through either IGQ or an IBM MQ channel.

Intra-group queuing can be used to deliver, more efficiently, small messages to queues residing on remote queue managers within a queue sharing group. Intra-group queuing also supports large messages, the largest being 100 MB *minus* the length of the transmission queue header.

Note: If you use this feature, users must have the same access to the queues on each queue manager in the queue sharing group.

The following diagram shows a typical example of intra-group queuing.

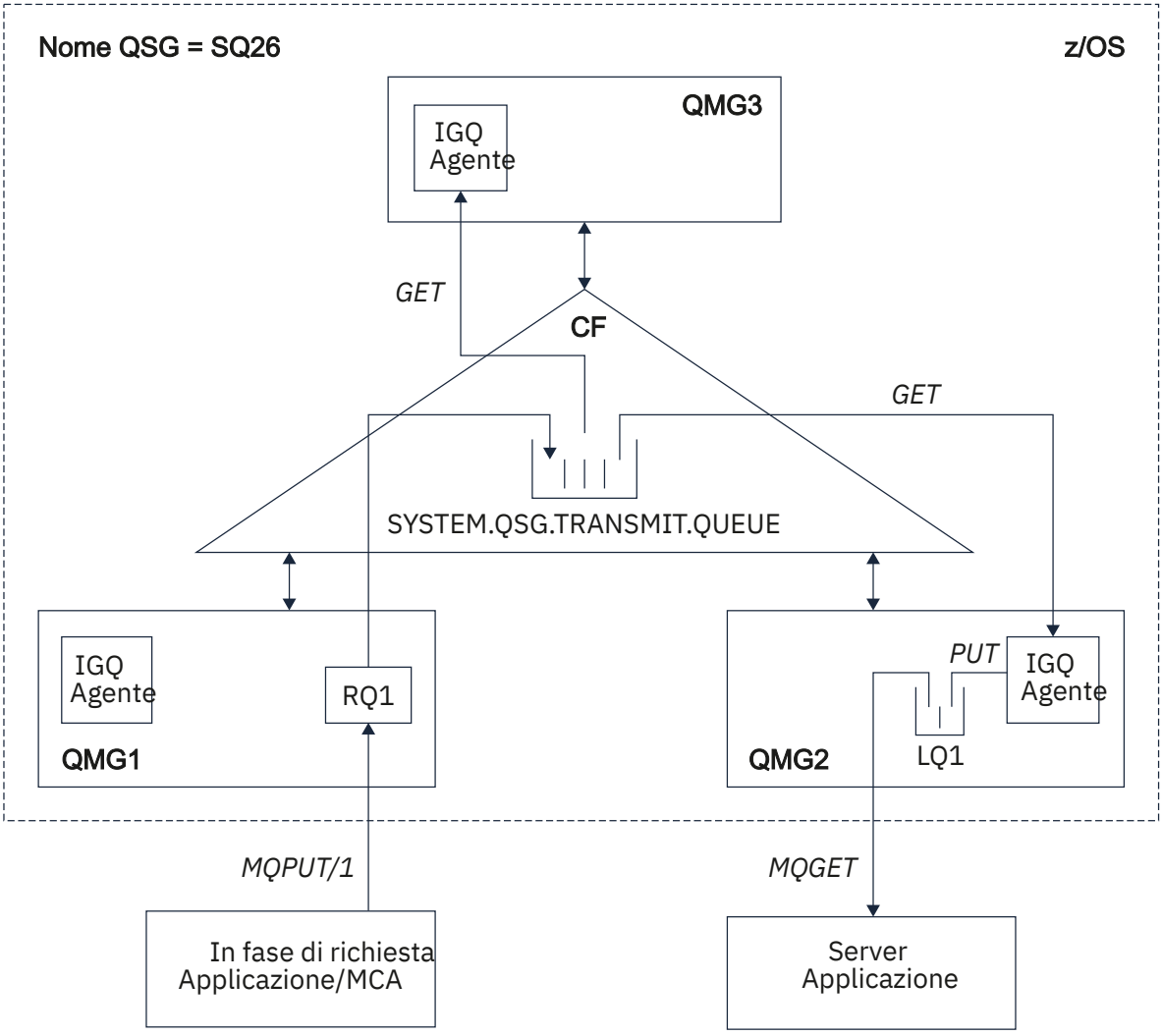


Figure 67. An example of intra-group queuing

The diagram shows:

- IGQ agents running on three queue managers (QMG1, QMG2, and QMG3) that are defined to a queue sharing group called SQ26.
- Shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE that is defined in the coupling facility (CF).
- A remote queue definition that is defined in queue manager QMG1.
- A local queue that is defined in queue manager QMG2.
- A requesting application (this application could be a Message Channel Agent (MCA)) that is connected to queue manager QMG1.

- A server application that is connected to queue manager QMG2.
- A request message being placed on to the SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing and the intra-group queuing agent

An IGQ agent is started during queue manager initialization. When applications open and put messages to remote queues, the local queue manager determines whether intra-group queuing is used for message transfer. If intra-group queuing is to be used, the local queue manager places the message on to the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent on the target remote queue manager retrieves the message and places it on to the destination queue.

Intra-group queuing terminology

Explanations of the terminology: intra-group queuing, shared transmission queue for use by intra-group queuing, and intra-group queuing agent.

Intra-group queuing

Intra-group queuing can effect potentially fast and less expensive message transfer between queue managers in a queue sharing group, without the need to define channels.

Shared transmission queue for use by intra-group queuing

Each queue sharing group has a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE for use by intra-group queuing. If intra-group queuing is enabled, SYSTEM.QSG.TRANSMIT.QUEUE appears in the name resolution path when opening remote queues. When applications (including Message Channel Agents (MCAs)) put messages to a remote queue, the local queue manager determines the eligibility of messages for fast transfer and places them on SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing agent

The IGQ agent is the task, started at queue manager initialization, that waits for suitable messages to arrive on the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent retrieves suitable messages from this queue and delivers them to the destination queues.

The IGQ agent for each queue manager is always started because intra-group queuing is used by the queue manager itself for its own internal processing.

z/OS Benefits of intra-group queuing

The benefits of intra-group queuing are: reduced system definitions, reduced system administration, improved performance, supports migration, and delivery of messages when multi-hopping between queue managers in a queue sharing group.

The benefits of intra-group queuing are:

Reduced system definitions
Intra-group queuing removes the need to define channels between queue managers in a queue sharing group.

Reduced system administration
Because there are no channels defined between queue managers in a queue sharing group, there is no requirement for channel administration.

Improved performance
Because there is only one IGQ agent needed for the delivery of a message to a target queue (instead of two intermediate sender and receiver agents), the delivery of messages using intra-group queuing can be less expensive than the delivery of messages using channels. In intra-group queuing there is only a receiving component, because the need for the sending component has been removed. This saving is because the message is available to the IGQ agent at the destination queue manager for

delivery to the destination queue once the put operation at the local queue manager has completed and, in the case of messages put in sync point scope, committed.

Supports migration

Applications external to a queue sharing group can deliver messages to a queue residing on any queue manager in the queue sharing group, while being connected only to a particular queue manager in the queue sharing group. This is because messages arriving on a receiver channel, destined for a queue on a remote queue manager, can be transparently sent to the destination queue using intra-group queuing. This facility allows applications to be deployed among the queue sharing group without the need to change any systems that are external to the queue sharing group.

A typical configuration is illustrated by the following diagram, in which:

- A requesting application connected to queue manager QMG1 needs to send a message to a local queue on queue manager QMG3.
- Queue manager QMG1 is connected only to queue manager QMG2.
- Queue managers QMG2 and QMG3, which were previously connected using channels, are now members of queue sharing group SQ26.

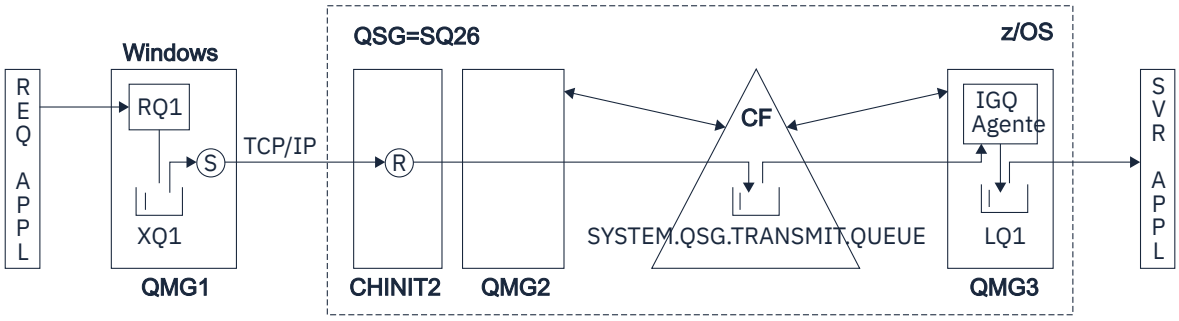


Figure 68. An example of migration support

The flow of operations is as follows:

1. The requesting application puts a message, destined for local queue LQ1 at remote queue manager QMG3, on to remote queue definition RQ1.
2. Queue manager QMG1, running on a Windows NT workstation, places the message on to the transmission queue XQ1.
3. Sender MCA (S) on QM1 transmits the message, using TCP/IP, to the receiver MCA (R) on channel initiator CHINIT2.
4. Receiver MCA (R) on channel initiator CHINIT2 places the message on to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
5. IGQ agent on queue manager QMG3 retrieves the message from the SYSTEM.QSG.TRANSMIT.QUEUE and places it on to the target local queue LQ1.
6. The server application retrieves the message from the target local queue and processes it.

Delivery of messages when multi-hopping between queue managers in a queue sharing group

The previous diagram in Supports migration also illustrates the delivery of messages when multi-hopping between queue managers in a queue sharing group. Messages arriving on a queue manager within the queue sharing group, but destined for a queue on another queue manager in the queue sharing group, can be easily transmitted to the destination queue on the destination queue manager, using intra-group queuing.

z/OS Limitations of intra-group queuing

The limitations of intra-group queuing are: messages eligible for transfer using intra-group queuing, number of intra-group queuing agents per queue manager, and starting and stopping the intra-group queuing agent.

This topic describes the limitations of intra-group queuing.

Messages eligible for transfer using intra-group queuing
Because intra-group queuing uses a shared transmission queue that is defined in the coupling facility (CF), intra-group queuing is limited to delivering messages of the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Number of intra-group queuing agents per queue manager
Only one IGQ agent is started per queue manager in a queue sharing group.

Starting and stopping the intra-group queuing agent
The IGQ agent is started during queue manager initialization and terminated during queue manager shut-down. It is designed to be a long running, self recovering (in the event of abnormal termination), task. If there is an error with the definition of the SYSTEM.QSG.TRANSMIT.QUEUE (for example, if this queue is Get inhibited) the IGQ agent keeps retrying. If the IGQ agent encounters an error that results in normal termination of the agent while the queue manager is still active, it can be restarted by issuing an ALTER QMGR IGQ(ENABLED) command. This command avoids the need to recycle the queue manager.

Setting the queue manager attribute IGQ to ENABLED or DISABLED
If the queue manager attribute IGQ is set to ENABLED or DISABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See Getting started with intra-group queueing for more information.

z/OS Getting started with intra-group queuing

You can enable, disable, and use intra-group queuing as described in this topic.

Enabling intra-group queuing
To enable intra-group queuing on your queue managers, you need to do the following:

- Define a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE. The definition of this queue can be found in thlqual.SCSQPROC(CSQ4INSS), the CSQINP2 sample for SYSTEM objects for queue sharing groups. This queue must be defined with the correct attributes, as stated in thlqual.SCSQPROC(CSQ4INSS), for intra-group queuing to work properly.
- Because the IGQ agent is always started at queue manager initialization, intra-group queuing is always available for inbound message processing. The IGQ agent processes any messages that are placed on the SYSTEM.QSG.TRANSMIT.QUEUE. However, to enable intra-group queuing for outbound processing, the queue manager attribute IGQ must be set to ENABLED.

  Important: If the queue manager attribute IGQ is set to ENABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See “Specific properties of intra-group queuing” on page 227 for more information. As described in the 'Programmer response' section for this reason code, applications need to be coded to handle this situation (see 2041 (07F9) (RC2041): MQRC_OBJECT_CHANGED for more details).

Additionally, as IGQ is designed as a long running and self-recovering task, which starts during initialization and terminates with shutdown, see “Limitations of intra-group queuing” on page 219 for further information.

Disabling intra-group queuing

To disable intra-group queuing for outbound message transfer, set the queue manager attribute IGQ to DISABLED. If intra-group queuing is disabled for a particular queue manager, the IGQ agent on that queue manager can still process inbound messages that have been placed on the SYSTEM.QSG.TRANSMIT.QUEUE by a queue manager that does have intra-group queuing enabled for outbound transfer.

Important: If the queue manager attribute IGQ is set to ENABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See “Specific properties of intra-group queuing” on page 227 for more information. As described in the 'Programmer response' section for this reason code, applications need to be coded to handle this situation (see 2041 (07F9) (RC2041): MQRC_OBJECT_CHANGED for more details).

Additionally, as IGQ is designed as a long running and self-recovering task, which starts during initialization and terminates with shutdown, see “Limitations of intra-group queuing” on page 219 for further information.

Using intra-group queuing

Once intra-group queuing is enabled, it is available for use and a queue manager uses it whenever possible. That is, when an application puts a message to a remote queue definition, to a fully qualified remote queue, or to a cluster queue, the queue manager determines if the message is eligible to be delivered using intra-group queuing and if it is, places the message on to SYSTEM.QSG.TRANSMIT.QUEUE. There is no need to change user applications, or to application queues, because for eligible messages the queue manager uses the SYSTEM.QSG.TRANSMIT.QUEUE, in preference to any other transmission queue.

z/OS Intra-group queuing configurations

In addition to the typical intra-group queuing configuration, other configurations are possible.

“Intra-group queuing concepts” on page 215 describes the typical configuration.

Related concepts

“Distributed queuing with intra-group queuing (multiple delivery paths)” on page 220
For applications that process short messages it might be feasible to configure intra-group queuing only for delivering messages between queue managers in a queue sharing group.

“Clustering with intra-group queuing (multiple delivery paths)” on page 222
It is possible to configure queue managers so that they are in a cluster as well as in a queue sharing group.

“Clustering, intra-group queuing and distributed queuing” on page 224
It is possible to configure a queue manager that is a member of a cluster as well as a queue sharing group and is connected to a distributed queue manager using a sender/receiver channel pair.

z/OS *Distributed queuing with intra-group queuing (multiple delivery paths)*

For applications that process short messages it might be feasible to configure intra-group queuing only for delivering messages between queue managers in a queue sharing group.

The choice of intra-group queuing over channel communications can be controlled by the CFSTRUCT type level. (3 instead of 4 or 5). The maximum message length as set on the SYSTEM.QSQ.TRANSMIT.QUEUE.

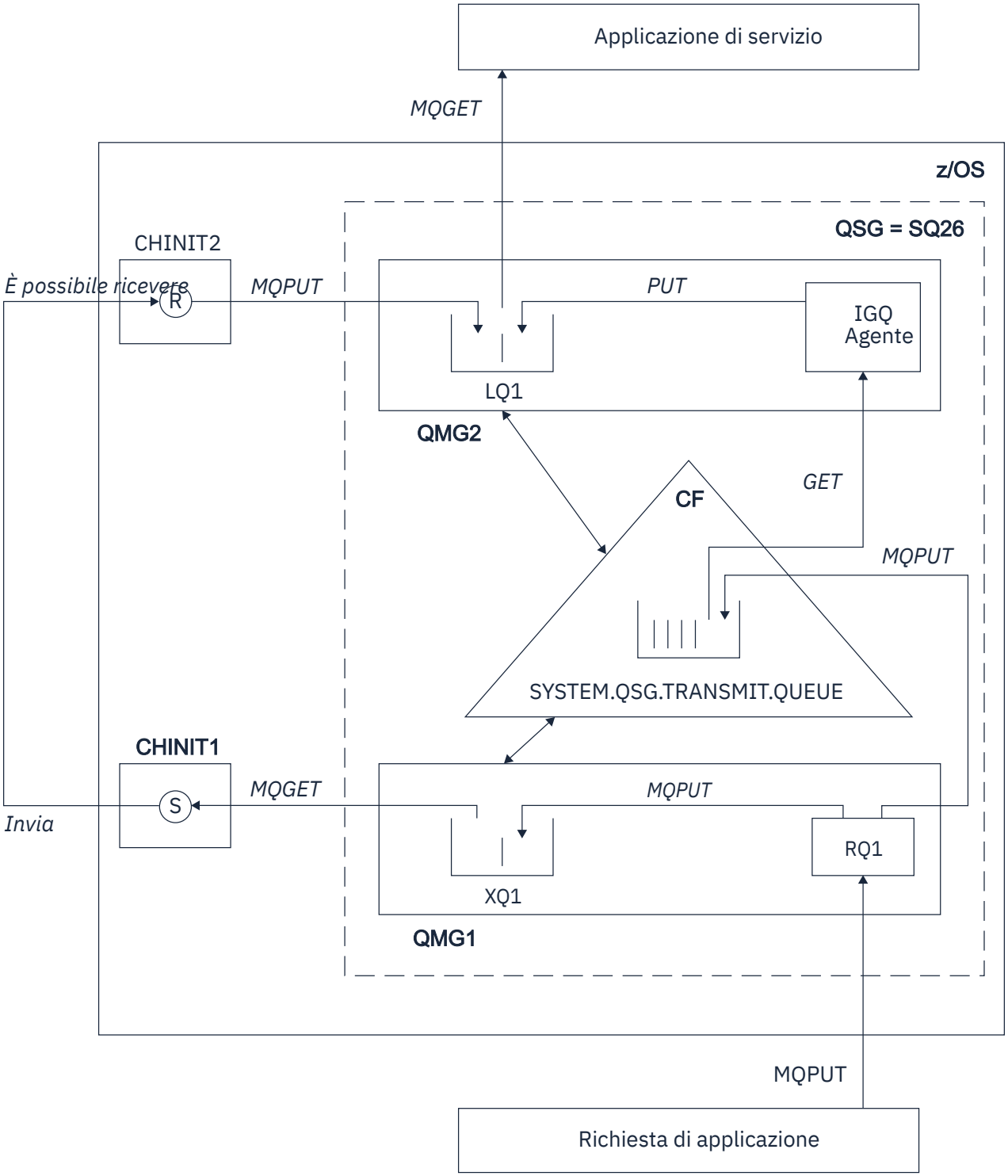


Figure 69. An example configuration

Open/Put processing

1. It is important to note that when the requesting application opens remote queue RQ1, name resolution occurs for both the non-shared transmission queue XQ1 and the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
2. When the requesting application puts a message on to the remote queue, based on whether intra-group queuing is enabled for outbound transfer on the queue manager and on the message characteristics, the message is put to transmission queue XQ1, or to transmission queue

SYSTEM.QSG.TRANSMIT.QUEUE. The queue manager places all large messages on to transmission queue XQ1, and all small messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.

3. If transmission queue XQ1 is full, or is not available, put requests for large messages fail synchronously with a suitable return and reason code. However, put requests for small messages continue to succeed and are placed on transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
4. If transmission queue SYSTEM.QSG.TRANSMIT.QUEUE is full, or cannot be put to, put requests for small messages fail synchronously with a suitable return and reason code. However, put requests for large messages continue to succeed and are placed on transmission queue XQ1. In this case, no attempt is made to put the small messages on to a transmission queue.

Flow for large messages

1. The requesting application puts large messages to remote queue RQ1.
2. Queue manager QMG1 puts the messages on to transmission queue XQ1.
3. Sender MCA (S) on queue manager QMG1 retrieves the messages from transmission queue XQ1 and sends them to queue manager QMG2.
4. Receiver MCA (R) on queue manager QMG2 receives the messages and places them on to destination queue LQ1.
5. The serving application retrieves and then processes the messages from queue LQ1.

Flow for small messages

1. The requesting application puts small messages on to remote queue RQ1.
2. Queue manager QMG1 puts the messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
3. IGQ on queue manager QMG2 retrieves the messages and places them on to the destination queue LQ1.
4. The serving application retrieves the messages from queue LQ1.

Points to note

1. The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
2. A potentially faster message delivery mechanism can be achieved for small messages.
3. Multiple paths are available for message delivery (that is, the normal channel route and the intra-group queuing route).
4. The intra-group queuing route, being potentially faster, is selected in preference to the normal channel route. Depending on the message characteristics, message delivery might be divided across the two paths. Hence, messages might be delivered out of sequence (though this delivery is also possible if messages are delivered using only the normal channel route).
5. When a route has been selected, and messages have been placed on to the transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to transmission queue XQ1.

z/OS *Clustering with intra-group queuing (multiple delivery paths)*

It is possible to configure queue managers so that they are in a cluster as well as in a queue sharing group.

When messages are sent to a cluster queue and the local and remote destination queue managers are in the same queue sharing group, intra-group queuing is used for the delivery of small messages (using the SYSTEM.QSG.TRANSMIT.QUEUE), and the delivery of large messages if intra-group queuing supports the size of the message. Also, the SYSTEM.CLUSTER.TRANSMIT.QUEUE is used for the delivery of messages to any queue manager that is in the cluster, but outside the queue sharing group. The following diagram illustrates this configuration (the channel initiators are not shown).

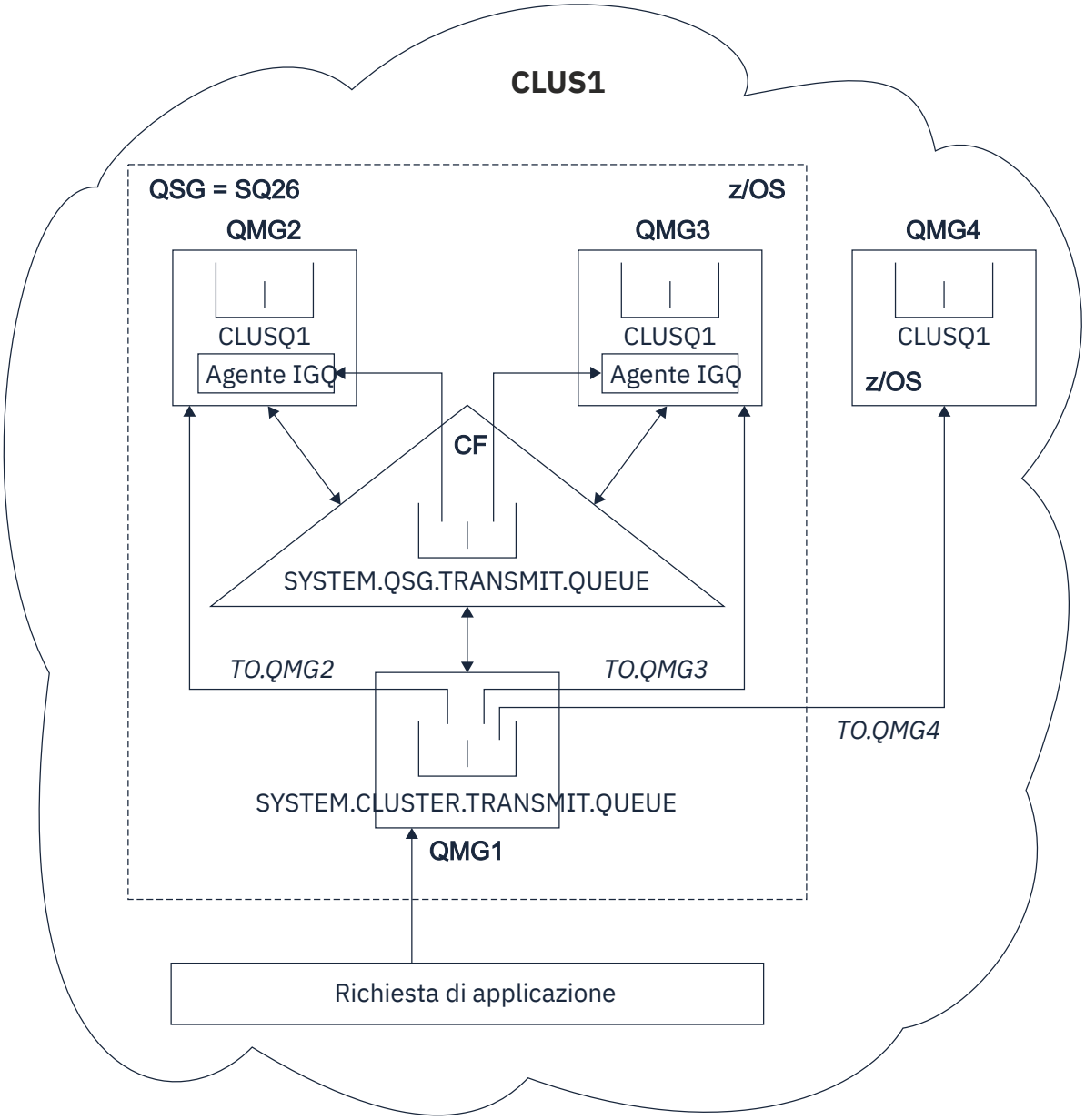


Figure 70. An example of clustering with intra-group queuing

The diagram shows:

- Four z/OS queue managers QMG1, QMG2, QMG3, and QMG4 configured in a cluster CLUS1.
- Queue managers QMG1, QMG2, and QMG3 configured in a queue sharing group SQ26.
- IGQ agents running on queue managers QMG2 and QMG3.
- The local SYSTEM.CLUSTER.TRANSMIT.QUEUE defined in QMG1.

  Note: For clarity, the SYSTEM.CLUSTER.TRANSMIT.QUEUE on the other queue managers not shown.
- The shared SYSTEM.QSG.TRANSMIT.QUEUE defined in the CF, which is in an IBM MQ structure configured with the CFLEVEL(3) RECOVER(YES) attribute.
- Cluster channels TO.QMG2 (connecting QMG1 to QMG2), TO.QMG3 (connecting QMG1 to QMG3), and TO.QMG4 (connecting QMG1 to QMG4).
- Cluster queue CLUSQ1 being hosted on queue managers QMG2, QMG3, and QMG4.

Assume that the requesting application opens the cluster queue with the MQOO_BIND_NOT_FIXED option, so that the target queue manager for the cluster queue is selected at put time.

If the selected target queue manager is QMG2:

- All large messages put by the requesting application are:
  - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1, because SYSTEM.QSG.TRANSMIT.QUEUE is in a CFLEVEL(3) structure; therefore supports messages only up to 63 KB in size.
  - Transferred to cluster queue CLUSQ1 on QMG2 using cluster channel TO.QMG2
- All small messages put by the requesting application are
  - Put to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE. This queue is in a structure configured with the RECOVER(YES) attribute, so is used for both persistent, and non-persistent, small messages.
  - Retrieved by the IGQ agent on QMG2
  - Put to the cluster queue CLUSQ1 on QMG2

If the selected target queue manager is QMG4:

- Because QMG4 is not a member of queue sharing group SQ26, all messages put by the requesting application are
  - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1
  - Transferred to cluster queue CLUSQ1 on QMG4 using cluster channel TO.QMG4

Points to note

- The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
- A potentially faster delivery mechanism is achieved for the transfer of small non-persistent messages between queue managers in a queue sharing group (even if the same queue managers are in a cluster).
- Multiple paths are available for message delivery (that is, both the cluster route and the intra-group queuing route).
- The intra-group queuing route, being potentially faster, is selected in preference to the cluster route. Depending on the message characteristics, message delivery might be divided across the two paths. Hence, messages might be delivered out of sequence. It is important to note that this delivery is possible without regard to the MQOO_BIND_* option specified by the application. Intra-group queuing distributes messages in the same way as clustering does, depending on whether the MQOO_BIND_NOT_FIXED, MQOO_BIND_ON_OPEN, MQOO_BIND_ON_GROUP, or MQOO_BIND_AS_Q_DEF is specified on open.
- When a route has been selected, and messages have been placed on to the transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

z/OS *Clustering, intra-group queuing and distributed queuing*

It is possible to configure a queue manager that is a member of a cluster as well as a queue sharing group and is connected to a distributed queue manager using a sender/receiver channel pair.

This configuration is a combination of distributed queuing with intra-group queuing and clustering with intra-group queuing.

Intra-group queuing is described in “Distributed queuing with intra-group queuing (multiple delivery paths)” on page 220.

Clustering with intra-group queuing is described in “Clustering with intra-group queuing (multiple delivery paths)” on page 222.

z/OS Intra-group queuing messages

This section describes the messages put to the SYSTEM.QSG.TRANSMIT.QUEUE.

Message structure

Like all other messages that are put to transmission queues, messages that are put to the SYSTEM.QSG.TRANSMIT.QUEUE are prefixed with the transmission queue header (MQXQH).

Message persistence

In IBM WebSphere MQ 5.3 and above, shared queues support both persistent and non-persistent messages.

If the queue manager terminates while the IGQ agent is processing non-persistent messages, or if the IGQ agent terminates abnormally while in the middle of processing messages, non-persistent messages being processed might be lost. Applications must make arrangements for the recovery of non-persistent messages if their recovery is required.

If a put request for a non-persistent message, issued by the IGQ agent, fails unexpectedly, the message being processed is lost.

Delivery of messages

The IGQ agent retrieves and delivers all nonpersistent messages outside of sync point scope, and all persistent messages within sync point scope. In this case, the IGQ agent acts as the sync point coordinator. The IGQ agent therefore processes nonpersistent messages like the way fast, nonpersistent messages are processed on a message channel. See Fast, nonpersistent messages.

Batching of messages

The IGQ agent uses a fixed batch size of 50 messages. Any persistent messages retrieved within a batch are committed at intervals of 50 messages. The agent commits a batch consisting of persistent messages when there are no more messages available for retrieval on the SYSTEM.QSG.TRANSMIT.QUEUE.

Message size

The maximum size of message that can be put to the SYSTEM.QSG.TRANSMIT.QUEUE is the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Default message persistence and default message priority

If the SYSTEM.QSG.TRANSMIT.QUEUE is in the queue name resolution path established at open time, then for messages that are put with default persistence and default priority (or with default persistence or default priority), the normal rules are applied in the selection of the queue that has default priority and persistence values that are used. (See the IBM MQ messages section for more information about the rules of queue selection).

Related concepts

“Undelivered/unprocessed messages” on page 225
This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

“Report messages - Intra Group Queuing” on page 226
This topic describes the report messages: Confirmation of arrival, confirmation of delivery, expiry report, and exception report.

z/OS *Undelivered/unprocessed messages*

This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

If an IGQ agent cannot deliver a message to the destination queue, the IGQ agent:

- Honors the MQRO_DISCARD_MSG report option (if the Report options field of the MQMD for the undelivered message indicates that it must) and discards the undelivered message.

- Attempts to place the undelivered message on to the dead letter queue for the destination queue manager, if the message has not already been discarded. The IGQ agent prefixes the message with a dead letter queue header (MQDLH).

If a dead letter queue is not defined, or if an undelivered message cannot be put to the dead letter queue, and if the undelivered message is:

- persistent, the IGQ agent backs out the current batch of persistent messages that it is processing, and enters a state of retry. For more information, see "Specific properties of intra-group queuing" on page 227.
- non-persistent, the IGQ agent discards the message and continues to process the next message.

If a queue manager in a queue sharing group is terminated before its associated IGQ agent has had time to process all its messages, the unprocessed messages remain on the SYSTEM.QSG.TRANSMIT.QUEUE until the queue manager is next started. The IGQ agent then retrieves and delivers the messages to the destination queues.

If the coupling facility fails before all the messages on the SYSTEM.QSG.TRANSMIT.QUEUE have been processed, any unprocessed non-persistent messages are lost.

IBM recommends that applications do not put messages directly to transmission queues. If an application does put messages directly to the SYSTEM.QSG.TRANSMIT.QUEUE, the IGQ agent might not be able to process these messages and they remain on the SYSTEM.QSG.TRANSMIT.QUEUE. Users then have to use their own methods to deal with these unprocessed messages.

z/OS *Report messages - Intra Group Queuing*

This topic describes the report messages: Confirmation of arrival, confirmation of delivery, expiry report, and exception report.

Confirmation of arrival (COA)/confirmation of delivery (COD) report messages
COA and COD messages are generated by the queue manager, when intra-group queuing is used.

Expiry report messages
Expiry report messages are generated by the queue manager.

Exception report messages

Depending on the MQRO_EXCEPTION_* report option specified in the *Report options* field of the message descriptor for the undelivered message, the IGQ agent generates the required exception report and places it on the specified reply-to queue. Intra-group queuing can be used to deliver the exception report to the destination reply-to queue.

The persistence of the report message is the same as the persistence of the undelivered message. If the IGQ agent fails to resolve the name of the destination reply-to queue, or if it fails to put the reply message to a transmission queue (for subsequent transfer to the destination reply-to queue) it attempts to put the exception report to the dead letter queue of the queue manager on which the report message is generated. If it is not possible, then if the undelivered message is:

- persistent, the IGQ agent discards the exception report, backs out the current batch of messages, and enters a state of retry. For more information, see "Specific properties of intra-group queuing" on page 227.
- non-persistent, the IGQ agent discards the exception report and continues processing the next message on the SYSTEM.QSG.TRANSMIT.QUEUE.

z/OS Security for intra-group queuing

This topic describes the security arrangements for intra-group queuing.

Queue manager attributes IGQAUT (IGQ authority) and IGQUSER (IGQ agent user ID) can be set to control the level of security checking that is performed when the IGQ agent opens destination queues.

Intra-group queuing authority (IGQAUT)

The IGQAUT attribute can be set to indicate the type of security checks to be performed, and hence to determine the userids to be used by the IGQ agent when it establishes the authority to put messages on to the destination queue.

The IGQAUT attribute is analogous to the PUTAUT attribute that is available on channel definitions.

Intra-group queuing user identifier (IGQUSER)

The IGQUSER attribute can be used to nominate a user ID to be used by the IGQ agent when it establishes the authority to put messages on to a destination queue.

The IGQUSER attribute is analogous to the MCAUSER attribute that is available on channel definitions.

z/OS Specific properties of intra-group queuing

This section describes the specific properties of intra-group queuing including Invalidation of object handles, self recovery and retry capability of the intra-group queuing agent, and the intra-group queuing agent and serialization.

Invalidation of object handles (MQRC_OBJECT_CHANGED)

If the attributes of an object are found to have changed after the object is opened, the queue manager invalidates the object handle with MQRC_OBJECT_CHANGED on its next use.

Intra-group queuing introduces the following rules for object handle invalidation:

- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was ENABLED at open time, but intra-group queuing is found to be DISABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was not included in the name resolution path during open processing because intra-group queuing was DISABLED at open time, but intra-group queuing is found to be ENABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was enabled at open time, but the SYSTEM.QSG.TRANSMIT.QUEUE definition is found to have changed by put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.

Self recovery of the intra-group queuing agent

If the IGQ agent terminates abnormally, message CSQM067E is issued and the IGQ agent starts again.

Retry capability of the intra-group queuing agent

If the IGQ agent encounters a problem accessing the SYSTEM.QSG.TRANSMIT.QUEUE (because it is not defined, for example, or is defined with incorrect attributes, or is inhibited for Gets, or for some other reason), the IGQ agent goes into the state of retry.

The IGQ agent observes short and long retry counts and intervals. The values for these counts and intervals, which cannot be changed, are as follows:

Table 20. Short and long retry counts and intervals values

Constant	**Value**
Short retry count	10
Short retry interval	60 seconds = 1 min
Long retry count	999,999,999

Table 20. Short and long retry counts and intervals values (continued)	
Constant	**Value**
Long retry interval	1200 seconds = 20 min

The intra-group queuing agent and serialization

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail.

If there is a failure of a queue manager in a queue sharing group while the IGQ agent is dealing with uncommitted messages on a shared queue or queues, the IGQ agent ends, and shared queue peer recovery takes place for the failing queue manager. Because shared queue peer recovery is an asynchronous activity, it leaves the possibility for the failing queue manager, and also the IGQ agent for that queue manager, to restart before shared queue peer recovery is complete. Which in turn leaves the possibility for any committed messages to be processed ahead of and out of sequence with the messages still being recovered. To ensure that messages are not processed out of sequence, the IGQ agent serializes access to shared queues by issuing the MQCONNX API call.

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail. An error message is issued and the IGQ agent is put into retry state. When queue manager peer recovery is complete, for example at the time of the next retry, the IGQ agent can start.

z/OS Storage management on z/OS

IBM MQ for z/OS requires permanent and temporary data structures and uses page sets and memory buffers to store this data. These topics give more details on how IBM MQ uses these page sets and buffers.

Related concepts

"Page sets for IBM MQ for z/OS" on page 228
Use this topic to understand how IBM MQ for z/OS uses pages sets to store messages.

"Storage classes for IBM MQ for z/OS" on page 229
A storage class is an IBM MQ for z/OS concept that allows the queue manager to map queues to page sets. You can use storage classes to control which data sets are used by which queues.

"Buffers and buffer pools for IBM MQ for z/OS" on page 231
IBM MQ for z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

Related reference

"Where to find more information about storage management for IBM MQ for z/OS" on page 232
Use this topic as a reference to find further information about storage management for IBM MQ for z/OS.

z/OS Page sets for IBM MQ for z/OS

Use this topic to understand how IBM MQ for z/OS uses pages sets to store messages.

A *page set* is a VSAM linear data set that has been specially formatted to be used by IBM MQ. Page sets are used to store most messages and object definitions.

The exceptions to this are global definitions, which are stored in a shared repository on Db2, and the messages on shared queues. These are not stored on the queue manager page sets. For more information about shared queues, see "Shared queues and queue sharing groups" on page 171, and for more information about global definitions, see Private and global definitions.

IBM MQ page sets can be up to 64 GB in size. Each page set is identified by a page set identifier (PSID), an integer in the range 00 through 99. Each queue manager must have its own page sets.

IBM MQ uses page set zero (PSID=00) to store object definitions and other important information relevant to the queue manager. For normal operation of IBM MQ it is essential that page set zero does not become full, so do not use it to store messages.

To improve the performance of your system, you should also separate short-lived messages from long-lived messages by placing them on different page sets.

You must format page sets, and IBM MQ provides a FORMAT utility for this; see Formatting page sets (FORMAT). Page sets must also be defined to the IBM MQ subsystem.

IBM MQ for z/OS can be configured to expand a page set dynamically if it becomes full. IBM MQ continues to expand the page set if required until 123 logical extents exist, if there is sufficient disk storage space available. The extents can span volumes if the linear data set is defined in this way, however, IBM MQ cannot expand the page sets beyond 64 GB.

You cannot use page sets from one IBM MQ queue manager on a different IBM MQ queue manager, or change the queue manager name. If you want to transfer the data from one queue manager to another, you must unload all the objects and messages from the first queue manager and reload them onto another.

It is not possible to use page sets greater than 4 GB in a queue manager running a release earlier than V6. During the migration period, when it is likely that you might need to fall back to a previous release of code:

- Do not change page set 0 to be greater than 4 GB.
- Other page sets greater than 4 GB will be left offline when restarting a queue manager with a previous release.

For further information about migrating existing page sets capable of expanding beyond 4 GB, see Defining a page set to be larger than 4 GB.

It is possible for an administrator to dynamically add page sets to a running queue manager, or remove page sets from a running queue manager (except for page set zero). The DEFINE PSID command can run after the queue manager restart has completed, only if the command contains the DSN keyword.

z/OS Storage classes for IBM MQ for z/OS

A storage class is an IBM MQ for z/OS concept that allows the queue manager to map queues to page sets. You can use storage classes to control which data sets are used by which queues.

Introducing storage classes

A *storage class* maps one or more queues to a page set. This means that messages for that queue are stored on that page set.

Storage classes allow you to control where non-shared message data is stored for administrative, data set space and load management, or application isolation purposes. You can also use storage classes to define the XCF group and member name of an IMS region if you are using the IMS bridge (described in “IBM MQ and IMS” on page 284).

Shared queues do not use storage classes to obtain a page set mapping because the messages on them are not stored on page sets.

How storage classes work

- You define a storage class, using the DEFINE STGCLASS command, specifying a page set identifier (PSID).
- When you define a queue, you specify the storage class in the STGCLASS attribute.

In the following example, the local queue QE5 is mapped to page set 21 through storage class ARC2.

```
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE QLOCAL(QE5) STGCLASS(ARC2)
```

This means that messages that are put on the queue QE5 are stored on page set 21 (if they stay on the queue long enough to be written to DASD).

More than one queue can use the same storage class, and you can define as many storage classes as you like. For example, you can extend the previous example to include more storage class and queue definitions, as follows:

```
DEFINE STGCLASS(ARC1) PSID(05)
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE STGCLASS(MAXI) PSID(05)
DEFINE QLOCAL(QE1) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE2) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE3) STGCLASS(MAXI) ...
DEFINE QLOCAL(QE4) STGCLASS(ARC2) ...
DEFINE QLOCAL(QE5) STGCLASS(ARC2) ...
```

In Figure 71 on page 230, both storage classes ARC1 and MAXI are associated with page set 05. Therefore, the queues QE1, QE2, and QE3 are mapped to page set 05. Similarly, storage class ARC2 associates queues QE4 and QE5 with page set 21.

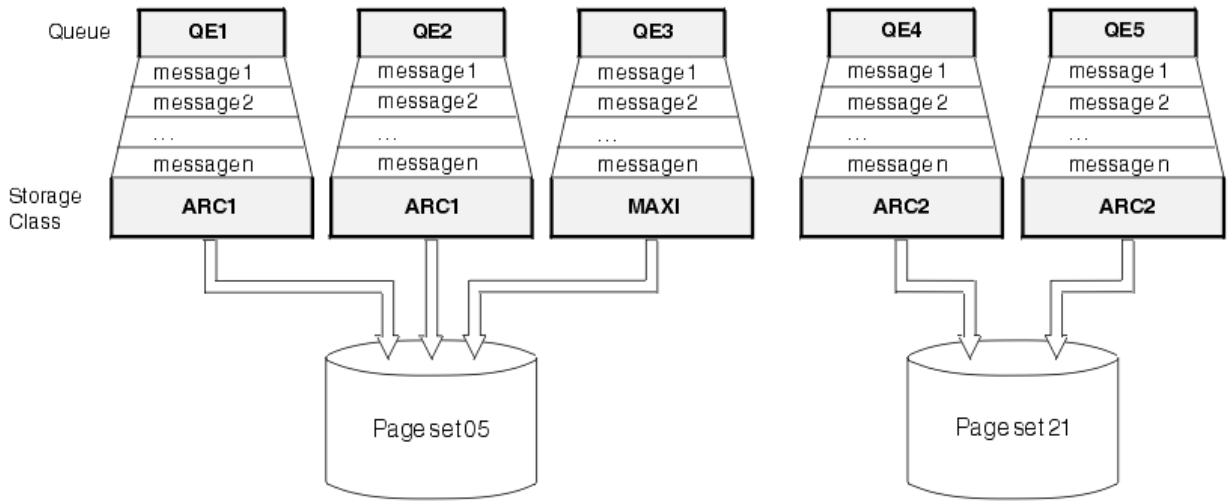


Figure 71. Mapping queues to page sets through storage classes

If you define a queue without specifying a storage class, IBM MQ uses a default storage class.

If a message is put on a queue that names a nonexistent storage class, the application receives an error. You must alter the queue definition to give it an existing storage class name, or create the storage class named by the queue.

You can change a storage class only when:

- All queues that use this storage class are empty, and have no uncommitted activity.
- All queues that use this storage class are closed.

z/OS Buffers and buffer pools for IBM MQ for z/OS

IBM MQ for z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

For efficiency, IBM MQ uses a form of caching whereby messages (and object definitions) are stored temporarily in buffers before being stored in page sets on DASD. Short-lived messages, that is, messages that are retrieved from a queue shortly after they are received, might only ever be stored in the buffers. This caching activity is controlled by a buffer manager, which is a component of IBM MQ.

The buffers are organized into *buffer pools*. You can define up to 100 buffer pools (0 through 99) for each queue manager.

You are recommended to use the minimal number of buffer pools consistent with the object and message type separation outlined in Figure 72 on page 231, and any data isolation requirements your application might have. Each buffer is 4 KB long. Buffer pools use 31 bit storage by default, in this mode, the maximum number of buffers is determined by the amount of 31 bit storage available in the queue manager address space; do not use more than about 70% for buffers. Alternatively, buffer pool storage allocation can be made from 64 bit storage (use the LOCATION attribute of the **DEFINE BUFFPOOL** command). Using LOCATION(ABOVE) so that 64 bit storage is used has two benefits. Firstly, there is much more 64 bit storage available so buffer pools can be much bigger, and secondly, 31 bit storage is made available for use by other functions. Typically, the more buffers you have, the more efficient the buffering and the better the performance of IBM MQ.

Figure 72 on page 231 shows the relationship between messages, buffers, buffer pools, and page sets. A buffer pool is associated with one or more page sets; each page set is associated with a single buffer pool.

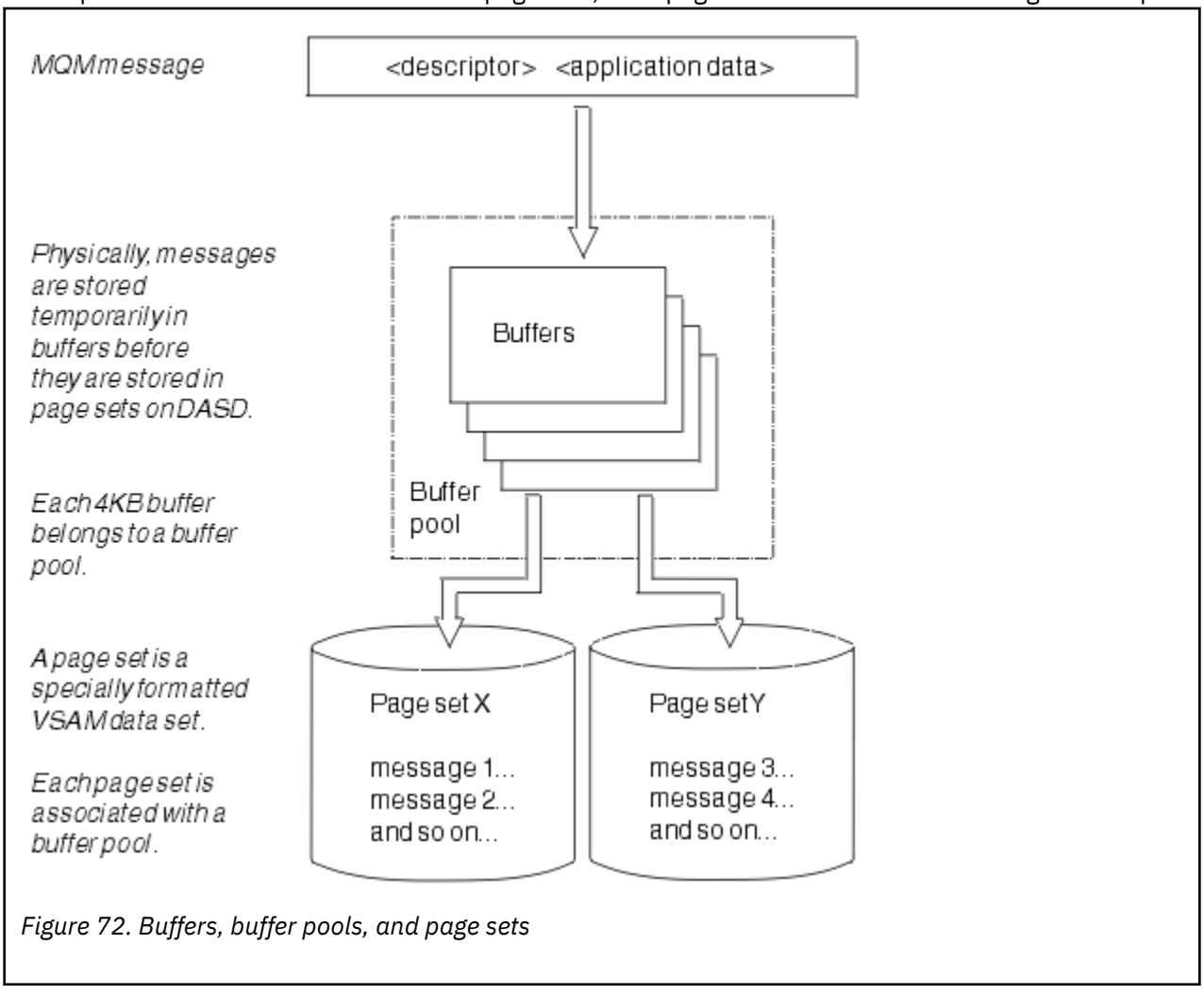


Figure 72. Buffers, buffer pools, and page sets

You can dynamically issue commands to modify buffer pool size, and location, using the **ALTER BUFFPOOL** command. Page sets can be dynamically added by using the **DEFINE PSID** command, or deleted by using the **DELETE PSID** command.

If a buffer pool is too small, IBM MQ issues message CSQP020E. You can then dynamically add more buffers to the affected buffer pool (note that you may have to remove buffers from other buffer pools to do this).

You specify the number of buffers in a pool with the **DEFINE BUFFPOOL** command, and you can dynamically resize buffer pools with the **ALTER BUFFPOOL** command. You determine the current number of buffers in a pool dynamically by displaying a page set that uses the buffer pool, using the **DISPLAY USAGE** command.

For performance reasons, do not put messages and object definitions in the same buffer pool. Use one buffer pool (say number zero) exclusively for page set zero, where the object definitions are kept. Similarly, keep short-lived messages and long-lived messages in different buffer pools and therefore on different page sets, and in different queues.

The **DEFINE BUFFPOOL** command cannot be used after restart to create a new buffer pool. Instead, if a **DEFINE PSID** command uses the DSN keyword, it can explicitly identify a buffer pool that is not currently defined. That new buffer pool will then be created.

z/OS Where to find more information about storage management for IBM MQ for z/OS

Use this topic as a reference to find further information about storage management for IBM MQ for z/OS.

You can find more information about the topics in this section from the following sources:

Table 21. Where to find more information about storage management

Topic	Where to look
How much storage you need	Planning your storage and performance requirements on z/OS
How large to make your page sets and buffer pools	Plan your page sets and buffer pools
Managing page sets	Managing page sets
MQSC commands	The MQSC commands

z/OS Logging in IBM MQ for z/OS

IBM MQ maintains *logs* of data changes and significant events as they occur. These logs can be used to recover data to a previous state if required.

The *bootstrap data set* (BSDS) stores information about the data sets that contain the logs.

The log does not contain information for statistics, traces, or performance evaluation. For further details about the statistical and monitoring information that IBM MQ collects, see Monitoring and statistics.

For more information about logging, see the following topics:

- “Log files in IBM MQ for z/OS” on page 233
- “How the log is structured” on page 236
- “How the IBM MQ for z/OS logs are written” on page 237
- “Larger log Relative Byte Address” on page 240
- “The bootstrap data set” on page 241

Related tasks

Planning your logging environment

Setting logs using the system parameter module

z/OS Administering z/OS

Related reference

z/OS Messages for IBM MQ for z/OS

z/OS Log files in IBM MQ for z/OS

Log files contain information needed for transaction recovery. Active log files can be archived so that you can keep log data for a long period.

What is a log file

IBM MQ records all significant events as they occur in an *active log*. The log contains the information needed to recover:

- Persistent messages
- IBM MQ objects, such as queues
- The IBM MQ queue manager

The active log comprises a collection of data sets (up to 310) which are used cyclically.

You can enable log archiving so that when an active log fills a copy is made in an archive data set. Using archiving allows you to keep log data for an extended period. If you do not use archiving, the logs wrap and earlier data is overwritten. To recover a page set, or recover data in a CF structure, you need log data from when the backup of the page set or structure was taken. An archive log can be created on disk or on tape.

Archiving

Because the active log has a fixed size, IBM MQ copies the contents of each log data set periodically to an *archive log*, which is normally a data set on a direct-access storage device (DASD) or a magnetic tape. If there is a subsystem or transaction failure, IBM MQ uses the active log and, if necessary, the archive log for recovery.

The archive log can contain up to 1000 sequential data sets. You can catalog each data set using the z/OS integrated catalog facility (ICF).

Archiving is an essential component of IBM MQ recovery. If a unit of recovery is a long-running one, log records within that unit of recovery might be found in the archive log. In this case, recovery requires data from the archive log. However, if archiving is disabled, the active log with new log records wraps, overwriting earlier log records. This means that IBM MQ might not be able to back out the unit of recovery and messages might be lost. The queue manager then terminates abnormally.

Therefore, in a production environment, **never switch archiving off**. If you do, you run the risk of losing data after a system or transaction failure. Only if you are running in a test environment can you consider switching archiving off. If you need to do this, use the CSQ6LOGP macro, which is described in Using CSQ6LOGP.

To help prevent problems with unplanned long-running units of work, IBM MQ issues a message (CSQJ160I or CSQJ161I) if a long-running unit of work is detected during active log offload processing.

Dual logging

In dual logging, each log record is written to two different active log data sets to minimize the likelihood of data loss problems during restart.

You can configure IBM MQ to run with either *single logging* or *dual logging*. With single logging, log records are written once to an active log data set. Each active log data set is a single-extent VSAM linear data set (LDS). With dual logging, each log record is written to two different active log data sets. Dual logging minimizes the likelihood of data loss problems during restart.

Log shunting

Log shunting causes the log records for some units of work to be written further down the log. This reduces the amount of log data that must be read at queue manager restart, or backout, for long running or long term in-doubt units of work.

When a unit of work is considered to be long, a representation of each log record is written further down the log. This technique is known as *shunting*. When the whole of the unit of work has been processed, the unit of work is in a *shunted* state. Any backout or restart activity relating to the shunted unit of work can use the shunted log records instead of using the original unit of work log records.

Detecting a long-running unit of work is a function of the checkpoint process. At checkpoint time, each active unit of work is checked to establish whether it needs to be shunted. If the unit of work has been through two prior checkpoints since it was created, or since it was last shunted, the unit of work is suitable to be shunted. This means that a single unit of work might be shunted more than once. This is known as a *multi-shunted* unit of work.

A unit of work is shunted every three checkpoints. However the checkpoint is performed asynchronously to the log-switch (or the writing of the log record which caused LOGLOAD to be exceeded).

There is only a single checkpoint taking place at a time, so there might be multiple log-switches before a checkpoint completes.

This means that if there are not enough active logs, or if they are too small, then shunting of a large unit of work might not complete before all the logs are filled.

Message CSQR027I results if shunting is unable to complete.

If log archiving is turned off, ABEND 5C6 with reason 00D1032A occurs if there is an attempt to back out the unit of work for which shunting failed. To avoid this problem you should use OFFLOAD=YES.

Log shunting is always active, and runs whether log archiving is enabled or not.

Note: Although all log records for a unit of work are shunted, the entire content of each record is not shunted, only the part that is necessary for backout. This means that the amount of log data written is kept to a minimum, and that shunted records cannot be used if a page set failure occurs. A long running unit of work is one that has been running for more than three queue manager checkpoints.

For more information about log shunting, see Managing the logs.

Log compression

You can configure IBM MQ for z/OS to compress and decompress log records as they are written and read from the log data set.

Log compression can be used to reduce the amount of data written to the log for persistent messages on private queues. The amount of compression that is achieved depends on the type of data contained within messages. For example, Run Length Encoding (RLE) works by compacting repeated instances of bytes which can give good results efficiently for structured or record oriented data.

Attention: Persistent messages that are being put to a shared queue are not subject to log compression.

You can use fields within the Log manager section of the System Management Facility 115 (SMF) records to monitor how much data compression is achieved. For more information about SMF, see Using the System Management Facility and Accounting and statistics messages.

Log compression increases the processor utilization of the system. You should only consider using compression if throughput of your queue manager is constrained by the IO bandwidth writing to the

log data sets or you are constrained by the disk storage needed to hold log data sets. If you are using shared queues then IO bandwidth constraints can be relieved by adding additional queue managers to the queue sharing group and distributing the workload across more queue managers.

The log compression option can be enabled and disabled as required without the need to stop and restart the queue manager. The queue manager can read any compressed log records regardless of the current log compression setting.

The queue manager supports 3 settings for log compression.

NONE
No log data compression is used. This is the default value.

RLE
Log data compression is performed using run-length encoding (RLE).

ANY
Enable the queue manager to select the compression algorithm that gives the greatest degree of log record compression. This option results in RLE compression.

You can control the compression of log records using one of the following:

- The SET and DISPLAY LOG commands in MQSC; see SET LOG and DISPLAY LOG
- The Set Log and Inquire Log functions in the PCF interface; see Set log and Inquire log
- The CSQ6LOGP macro in the system parameter module; see Using CSQ6LOGP

In addition the Log Print utility CSQ1LOGP has support for expanding any compressed log records.

Log data

The log can contain up to 18 million million million ($1.8*10^{19}$) bytes. Each byte can be addressed by its offset from the beginning of the log, and that offset is known as its *relative byte address* (RBA).

The RBA is referenced by a 6-byte or 8-byte field giving a total addressable range of 2^{48} bytes, or 2^{64} bytes, depending on whether 6-byte or 8-byte log RBAs are in use.

However, when IBM MQ detects that the used range is beyond F00000000000 (if 6-byte RBAs are in use) or FFFF800000000000 (if 8-byte log RBAs are in use), messages CSQI045, CSQI046, CSQI047, and CSQJ032 are issued, warning you to reset the log RBA.

If the RBA value reaches FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFC000000000 (if 8-byte log RBAs are in use) the queue manager terminates with reason code 00D10257.

Once the warning messages about the used log range are being issued, you should plan a queue manager outage during which the queue manager can be converted to use 8-byte log RBAs, or the log can be reset. The procedure to reset the log is documented in Resetting the queue manager's log.

If your queue manager is using 6-byte log RBAs, consider converting the queue manager to use 8-byte log RBAs rather than resetting the queue manager's log, following the procedure documented in Implementing the larger log Relative Byte Address.

The log consists of *log records*, each of which is a set of log data treated as a single unit. A log record is identified either by the RBA of the first byte of its header, or by its log record sequence number (LRSN). The RBA or LRSN uniquely identifies a record that starts at a particular point in the log.

Whether you use the RBA or LRSN to identify log points depends on whether you are using queue sharing groups. In a queue-sharing environment, you cannot use the relative byte address to uniquely identify a log point, because multiple queue managers can update the same queue at the same time, and each has its own log. To solve this, the log record sequence number is derived from a timestamp value, and does not necessarily represent the physical displacement of the log record within the log.

Each log record has a header that gives its type, the IBM MQ subcomponent that made the record, and, for unit of recovery records, a unit of recovery identifier.

There are four types of log record, described under the following headings:

- Unit of recovery log records
- Checkpoint records
- Page set control records
- CF structure backup records

Unit of recovery log records

Most of the log records describe changes to IBM MQ queues. All such changes are made within units of recovery.

IBM MQ uses special logging techniques involving *undo/redo* and *compensating log records* to reduce restart times and improve system availability.

One effect of this is that the restart time is bounded. If a failure occurs during a restart so that the queue manager has to be restarted a second time, all the recovery activity that completed to the point of failure in the first restart does not need to be reapplied during a second restart. This means that successive restarts do not take progressively longer times to complete.

Checkpoint records

To reduce restart time, IBM MQ takes periodic checkpoints during normal operation. These occur as follows:

- When a predefined number of log records has been written. This number is defined by the checkpoint frequency operand called LOGLOAD of the system parameter macro CSQ6SYSP, described in Using CSQ6SYSP.
- At the end of a successful restart.
- At normal termination.
- Whenever IBM MQ switches to the next active log data set in the cycle.

At the time a checkpoint is taken, IBM MQ issues the DISPLAY CONN command (described in DISPLAY CONN) internally so that a list of connections currently in doubt is written to the z/OS console log.

Page set control records

These records register the page sets and buffer pools known to the IBM MQ queue manager at each checkpoint, and record information about the log ranges required to perform media recovery of the page set at the time of the checkpoint.

Certain dynamic changes to page sets and buffer pools are also written as page set control records, so that the changes can be recovered and automatically reinstated at the next queue manager restart.

CF structure backup records

These records hold data read from a coupling facility list structure in response to a BACKUP CFSTRUCT command. In the unlikely event of a coupling facility structure failure, these records are used, together with unit of recovery records, by the RECOVER CFSTRUCT command to perform media recovery of the coupling facility structure to the point of failure.

Related tasks

Implementing the larger log Relative Byte Address

z/OS How the log is structured

Use this topic to understand the terminology used to describe log records.

Each active log data set must be a VSAM linear data set (LDS). The physical output unit written to the active log data set is a 4 KB control interval (CI). Each CI contains one VSAM record.

Physical and logical log records

One VSAM CI is a *physical* record. The information logged at a particular time forms a *logical* record, with a length that varies independently of the space available in the CI. So one physical record might contain:

- Several logical records
- One or more logical records and part of another logical record
- Part of one logical record only

The term *log record* refers to the *logical* record, regardless of how many *physical* records are needed to store it.

z/OS How the IBM MQ for z/OS logs are written

Use this topic to understand how IBM MQ processes log file records.

IBM MQ writes each log record to a DASD data set called the *active log*. When the active log is full, IBM MQ copies its contents to a DASD or tape data set called the *archive log*. This process is called *offloading*.

Figure 73 on page 238 illustrates the process of logging. Log records typically go through the following cycle:

1. IBM MQ notes changes to data and significant events in recovery log records.
2. IBM MQ processes recovery log records and breaks them into segments, if necessary.
3. Log records are placed sequentially in *output log buffers*, which are formatted as VSAM Controls Intervals (CI). Each log record is identified by a relative byte address in the range zero through 2^{64} -1.
4. The CIs are written to a set of predefined DASD active log data sets, which are used sequentially and recycled.
5. If archiving is active, as each active log data set becomes full, its contents are automatically offloaded to a new archive log data set.

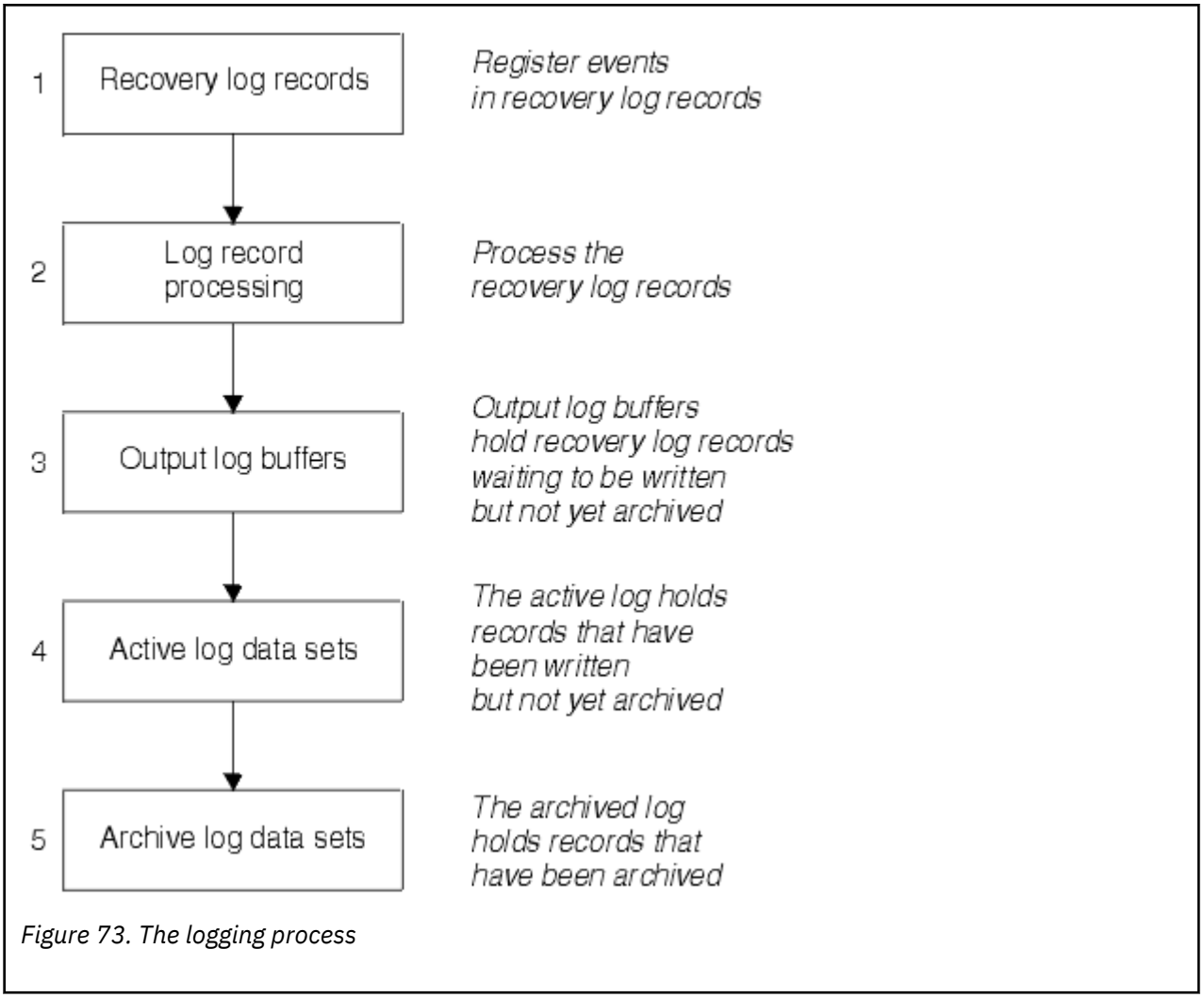


Figure 73. The logging process

When the active log is written

The in-storage log buffers are written to an active log data set whenever any of the following occur:

- The log buffers become full.
- The write threshold is reached (as specified in the CSQ6LOGP macro).
- Certain significant events occur, such as a commit point, or when an IBM MQ BACKUP CFSTRUCT command is issued.

When the queue manager is initialized, the active log data sets named in the BSDS are dynamically allocated for exclusive use by the queue manager and remain allocated exclusively to IBM MQ until the queue manager terminates.

Dynamically adding log data sets

It is possible to dynamically define new active log data sets while the queue manager is running. This feature alleviates the problem of a queue manager hang when archiving is not able to offload active logs due to a transient problem. See the DEFINE LOG command for more information.

Note: To redefine or remove active logs you must terminate and restart the queue manager.

IBM MQ and Storage Management Subsystem

IBM MQ parameters enable you to specify Storage Management Subsystem (MVS™/DFP SMS) storage classes when allocating IBM MQ archive log data sets dynamically. IBM MQ initiates the archiving of log data sets, but you can use SMS to perform allocation of the archive data set.

Related reference

“When the IBM MQ for z/OS archive log is written” on page 239
Use this topic to understand the process of copying active logs to archive logs, and when the process occurs.

z/OS *When the IBM MQ for z/OS archive log is written*

Use this topic to understand the process of copying active logs to archive logs, and when the process occurs.

The process of copying active logs to archive logs is called *offloading*. The relation of offloading to other logging events is shown schematically in Figure 74 on page 239.

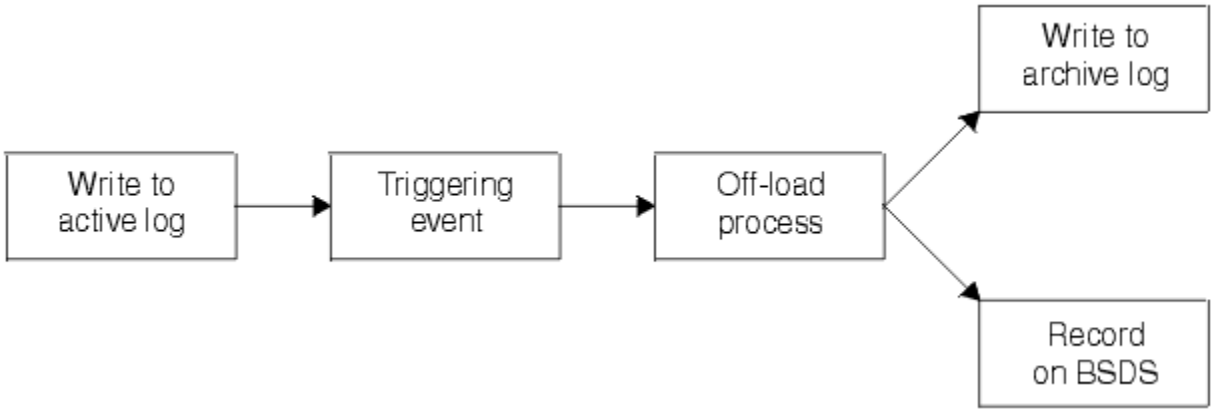


Figure 74. The offloading process

Triggering the offloading process

The offload process of an active log to an archive log can be triggered by several events. For example:

- Filling an active log data set.
- Using the MQSC ARCHIVE LOG command.
- An error occurring while writing to an active log data set.

The data set is truncated before the point of failure, and the record that was not written becomes the first record of the new data set. Offloading is triggered for the truncated data set as for a normal full log data set. If there are dual active logs, both copies are truncated so that the two copies remain synchronized.

Message CSQJ110E is issued when the last available active log is 5% full and at 5% increments thereafter, stating the percentage of the log's capacity that is in use. If all the active logs become full, IBM MQ stops processing, until offloading occurs, and issues this message:

```
CSQJ111A +CSQ1 OUT OF SPACE IN ACTIVE LOG DATA SETS
```

The offload process

When all the active logs become full, IBM MQ runs the offloading process and halts processing until the offloading process has been completed. If the offload processing fails when the active logs are full, IBM MQ abends.

When an active log is ready to be offloaded, a request is sent to the z/OS console operator to mount a tape or prepare a DASD unit. The value of the ARCWTOR logging option (for further information, see Using CSQ6ARVP) determines whether the request is received. If you are using tape for offloading, specify ARCWTOR=YES. If the value is YES, the request is preceded by a WTOR (message number CSQJ008E) telling the operator to prepare an archive log data set to be allocated.

The operator need not respond to this message immediately. However, delaying the response delays the offload process. It does not affect IBM MQ performance unless the operator delays the response for so long that IBM MQ runs out of active logs.

The operator can respond by canceling the offload process. In this case, if the allocation is for the first copy of dual archive data sets, the offload process is merely delayed until the next active log data set becomes full. If the allocation is for the second copy, the archive process switches to single copy mode, but for this data set only.

Interruptions and errors while offloading

A request to stop the queue manager does not take effect until offload processing has finished. If IBM MQ fails while offloading is in progress, offloading begins again when the queue manager is restarted.

Messages during offload processing

Offloaded messages are sent to the z/OS console by IBM MQ and the offloading process. You can use these messages to find the RBA ranges in the various log data sets.

z/OS Larger log Relative Byte Address

This function improves the availability of the queue manager by increasing the period of time before you have to reset the log.

Recovery data is written to the log so that persistent messages are available when the queue manager is restarted. The term log Relative Byte Address (log RBA) is used to refer to the location of data as an offset from the beginning of the log.

Before IBM MQ 8.0, the 6 byte log RBA could address up to 256 terabytes of data. Before this quantity of log records has been written, you have to reset the queue manager's log by following the procedure documented in Resetting the queue manager's log.

Resetting the logs of queue managers is not a quick process, and can require an extended outage, due to the need to reset the page sets as part of the process. For a high use queue manager this operation might typically be done once a year.

From IBM MQ 8.0, the log RBA can be 8 bytes long and the queue manager can now address over 64,000 times as much data (16 exabytes) before the log RBA needs to be reset. The impact of using the larger log RBA is that the size of the log data written increases by a few bytes.

When is this function enabled?

Queue managers created at IBM MQ 9.3.0 or later already have this function enabled.

If the current log RBA is approaching the end of the log RBA range, consider converting the queue manager to use an 8 byte log RBA rather than resetting the queue manager's log. Converting a queue manager to use 8 byte log RBAs requires a shorter outage than resetting the log, and significantly increases the period of time before you have to reset the log.

Message CSQJ034I, issued during queue manager initialization, indicates the end of the log RBA range for the queue manager as configured, and can be used to determine whether 6 byte or 8 byte log RBAs are in use.

How is this function enabled?

8 byte log RBA is enabled by starting the queue manager with a version 2 format BSDS. In summary, this is achieved by:

1. Ensuring that all queue managers in the queue sharing group meet the requirements for enabling 8 byte log RBA
2. Shutting down the queue manager cleanly
3. Running the BSDS conversion utility to create a copy of the BSDS in version 2 format.
4. Restarting the queue manager with the converted BSDS.

Once a queue manager has been converted to use 8 byte log RBAs, it cannot go back to using 6 byte log RBA.

See Implementing the larger log Relative Byte Address for the detailed procedure on how to enable 8 byte log RBAs.

Related tasks

Planning to increase the maximum addressable log range

Related reference

The BSDS conversion utility (CSQJUCNV)

z/OS The bootstrap data set

The bootstrap data set is required by IBM MQ as a mechanism to reference log data sets, and log records. This information is required during normal processing, and restart recovery.

What the bootstrap data set is for

The *bootstrap data set* (BSDS) is a VSAM key-sequenced data set (KSDS) that holds information needed by IBM MQ. It contains the following:

- An inventory of all active and archived log data sets known to IBM MQ. IBM MQ uses this inventory to:
  - Track the active and archived log data sets
  - Locate log records so that it can satisfy log read requests during normal processing
  - Locate log records so that it can handle restart processing

  IBM MQ stores information in the inventory each time an archive log data set is defined or an active log data set is reused. For active logs, the inventory shows which are full and which are available for reuse. The inventory holds the relative byte address (RBA) of each portion of the log held in that data set.
- A *wrap-around* inventory of all recent IBM MQ activity. This is needed if you have to restart the queue manager.

The BSDS is required if the queue manager has an error and you have to restart it. IBM MQ **must** have a BSDS. To minimize the likelihood of problems during a restart, you can configure IBM MQ with dual BSDSs, each recording the same information. Using dual BSDSs is known as running in *dual mode*. If possible, place the copies on separate volumes. This reduces the risk of them both being lost if the volume is corrupted or destroyed. Use dual BSDSs rather than dual write to DASD.

The BSDS is set up when IBM MQ is customized and you can manage the inventory using the change log inventory utility (CSQJU003). For more information about this utility, see Amministrazione di 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 Amministrazione di IBM MQ for z/OS.

The active logs are first registered in the BSDS when IBM MQ is installed. You cannot replace the active logs without terminating and restarting the queue manager.

Archive log data sets are allocated dynamically. When one is allocated, the data set name is registered in the BSDS. The list of archive log data sets expands as archives are added, and wraps when a user-determined number of entries has been reached. The maximum number of entries is 1000 for single archive logging and 2000 for dual logging.

You can use a tape management system to delete the archive log data sets (IBM MQ does not have an automated method). Therefore, the information about an archive log data set can be in the BSDS long after the archive log data set has been deleted by the system administrator.

Conversely, the maximum number of archive log data sets could have been exceeded, and the data from the BSDS dropped long before the data set has reached its expiry date.

You can use the following MQSC command to determine the extent of the log, and the name of the active or archive log data set holding the earliest log RBA, required for various types of media or queue manager recovery:

```
DISPLAY USAGE TYPE(DATASET)
```

If the system parameter module specifies that archive log data sets are cataloged when allocated, the BSDS points to the integrated catalog facility (ICF) catalog for the information needed for later allocations. Otherwise, the BSDS entries for each volume register the volume serial number and unit information that is needed for later allocations.

The BSDS version

The format of the BSDS varies according to its version. Increasing the version of the BSDS allows new features to be used. The following BSDS versions are supported by IBM MQ:

Version 1
Supported by all releases of IBM MQ. A version 1 BSDS supports 6-byte log RBA values.

Version 2
Supported by IBM MQ 8.0 and later. A version 2 BSDS enables 8-byte log RBA values, and up to 310 data sets in each active log copy.

Enabled by default for queue managers created at IBM MQ 9.3.0 or later.

Version 3
Supported by IBM MQ 8.0 and later. The BSDS is automatically converted to version 3, from version 2, when more than 31 data sets are added to either active log copy.

You can determine the version of a BSDS by running the print log map utility (CSQJU004). To convert a BSDS from Version 1 to Version 2, run the BSDS conversion utility (CSQJUCNV).

See “Larger log Relative Byte Address” on page 240 for more information on 6-byte and 8-byte log RBAs.

Archive log data sets and BSDS copies

Each time a new archive log data set is created, a copy of the BSDS is also created. If the archive log is on tape, the BSDS is the first data set on the first output volume. If the archive log is on DASD, the BSDS is a separate data set.

The data set names of the archive log and the BSDS copy are the same, except that the lowest-level qualifier of the archive log name begins with A and the BSDS copy begins with B, for example:

Archive log name
CSQ.ARCHLOG1.E00186.T2336229. *A* 0000001

BSDS copy name
CSQ.ARCHLOG1.E00186.T2336229. *B* 0000001

If there is a read error while copying the BSDS, the copy is not created, message CSQJ125E is issued, and the offloading to the new archive log data set continues without the BSDS copy.

z/OS System definition on z/OS

IBM MQ for z/OS uses many default object definitions, and provides sample JCL to create those default objects. Use this topic to understand these default objects, and the sample JCL.

Setting system parameters

In IBM MQ for z/OS, a system parameter module controls the logging, archiving, tracing, and connection environments that IBM MQ uses in its operation. The system parameters are specified by three assembler macros, as follows:

CSQ6SYSP
System parameters, including setting the connection and tracing environment.

CSQ6LOGP
Logging parameters.

CSQ6ARVP
Log archive parameters.

Default parameter modules are supplied with IBM MQ for z/OS. If these do not contain the values that you want to use, you can create your own parameter modules using the sample supplied with IBM MQ. The sample is `thlqual.SCSQPROC(CSQ4ZPRM)`.

You can alter some system parameters while a queue manager is running. See the SET SYSTEM, SET LOG, and SET ARCHIVE commands in The MQSC commands.

For more information about defining , see the following topics:

- “Defining system objects for IBM MQ for z/OS” on page 243
- “Tuning your queue manager on IBM MQ for z/OS” on page 248
- “Sample definitions supplied with IBM MQ for z/OS” on page 249

Related concepts
Customize the sample initialization input data sets
Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS
Related tasks
Administering z/OS
Configuring clusters
Monitoring IBM MQ

z/OS Defining system objects for IBM MQ for z/OS

IBM MQ for z/OS requires additional predefined objects for publish/subscribe applications, cluster, and channel control and other system administration functions.

The system objects required by IBM MQ for z/OS can be divided into the following categories:

- Publish/subscribe objects
- System default objects
- System command objects
- System administration objects
- Channels queues
- Cluster queues

- Queue sharing group queues
- Storage classes
- Defining the system object dead-letter queue
- Default transmission queue
- Internal queues
- “Channel authentication queue” on page 247

Publish/subscribe objects

There are several system objects that you need to define before you can use publish/subscribe applications with IBM MQ for z/OS. Sample definitions are supplied with IBM MQ to help you define these objects. These samples are described in CSQ4INSG.

To use publish/subscribe you need to define the following objects:

- A local queue called SYSTEM.RETAINED.PUB.QUEUE, which is used to hold a copy of each retained publication in the queue manager. Each full topic name could have up to one retained publication stored on this queue. If your applications will make use of retained publications on many different topics, or if your retained publication messages are large messages, the requirements for storage for this queue should be carefully planned, including assigning it to its own page set if the storage requirements for it are large. To improve performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definition).
- A local queue called SYSTEM.DURABLE.SUBSCRIBER.QUEUE, which is used to hold a persistent copy of the durable subscriptions in the queue manager. To improve performance, you should define this queue with an index type of CORRELID (as shown in the supplied sample queue definition).
- A local queue called SYSTEM.DURABLE.MODEL.QUEUE, which is used as a model for managed durable subscriptions.
- A local queue called SYSTEM.NDURABLE.MODEL.QUEUE, which is used as a model for managed non-durable subscriptions.
- A namelist called SYSTEM.QPUBSUB.QUEUE.NAMELIST, which contains a list of queue names monitored by the queued publish/subscribe interface.
- A namelist called SYSTEM.QPUBSUB.SUBPOINT.NAMELIST, which contains a list of topic objects used by the queued publish/subscribe interface to match topic objects to subscription points.
- A topic called SYSTEM.BASE.TOPIC, which is used as a base topic for resolving attributes.
- A topic called SYSTEM.BROKER.DEFAULT.STREAM, which is the default stream used by the queued publish/subscribe interface.
- A topic called SYSTEM.BROKER.DEFAULT.SUBPOINT, which is the default RFH2 subscription point used by the queued publish/subscribe interface.
- A topic called SYSTEM.BROKER.ADMIN.STREAM, which is the admin stream used by the queued publish/subscribe interface.
- A subscription called SYSTEM.DEFAULT.SUB, which is a default subscription object used to provide default values on DEFINE SUB commands.

System default objects

System default objects are used to provide default attributes when you define an object and do not specify the name of another object to base the definition on.

The names of the default system object definitions begin with the characters "SYSTEM.DEFAULT" or "SYSTEM.DEF." For example, the system default local queue is named SYSTEM.DEFAULT.LOCAL.QUEUE.

These objects define the system defaults for the attributes of these IBM MQ objects:

- Local queues

- Model queues
- Alias queues
- Remote queues
- Processes
- Namelists
- Channels
- Storage classes
- Authentication information

Shared queues are a special type of local queue, so when you define a shared queue, the definition is based on the SYSTEM.DEFAULT.LOCAL.QUEUE. You need to remember to supply a value for the Coupling Facility structure name because one is not specified in the default definition. Alternatively, you could define your own default shared queue definition to use as a basis for shared queues so that they all inherit the required attributes. Remember that you need to define a shared queue on one queue manager in the queue sharing group only.

System command objects

The names of the system command objects begin with the characters SYSTEM.COMMAND. You must define these objects before you can use the IBM MQ operations and control panels to issue commands to an IBM MQ subsystem.

There are two system command objects:

1. The system-command input queue is a local queue on which commands are put before they are processed by the IBM MQ command processor. It must be called SYSTEM.COMMAND.INPUT. An alias named SYSTEM.ADMIN.COMMAND.QUEUE should also be defined, for compatibility with IBM MQ for Multiplatforms, and for use by the IBM MQ Console and administrative REST API.
2. SYSTEM.COMMAND.REPLY.MODEL is a model queue that defines the system-command reply-to queue.

There are two extra objects for use by the IBM MQ Explorer:

- SYSTEM.MQEXPLORER.REPLY.MODEL queue
- SYSTEM.ADMIN.SVRCONN channel

SYSTEM.REST.REPLY.QUEUE is the reply queue used by the IBM MQ administrative REST API.

Commands are normally sent using nonpersistent messages so both the system command objects should have the DEFPSIST(NO) attribute so that applications using them (including the supplied applications like the utility program and the operations and control panels) get nonpersistent messages by default. If you have an application that uses persistent messages for commands, set the DEFTYPE(PERMDYN) attribute for the reply-to queue, because the reply messages to such commands are persistent.

System administration objects

The names of the system administration objects begin with the characters SYSTEM.ADMIN.

There are seven system administration objects:

- The SYSTEM.ADMIN.CHANNEL.EVENT queue
- The SYSTEM.ADMIN.COMMAND.EVENT queue
- The SYSTEM.ADMIN.CONFIG.EVENT queue
- The SYSTEM.ADMIN.PERFM.EVENT queue
- The SYSTEM.ADMIN.QMGR.EVENT queue

- The SYSTEM.ADMIN.TRACE.ROUTE.QUEUE queue
- The SYSTEM.ADMIN.ACTIVITY.QUEUE queue

Channels queues

To use distributed queuing, you need to define the following objects:

- A local queue with the name SYSTEM.CHANNEL.SYNCQ, which is used to maintain sequence numbers and logical units of work identifiers (LUWID) of channels. To improve channel performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definition).
- A local queue with the name SYSTEM.CHANNEL.INITQ, which is used for channel commands.

You cannot define these queues as shared queues.

Cluster queues

To use IBM MQ clusters, you need to define the following objects:

- A local queue called the SYSTEM.CLUSTER.COMMAND.QUEUE, which is used to communicate repository changes between queue managers. Messages written to this queue contain updates to the repository data to be applied to the local copy of the repository, or requests for repository data.
- A local queue called SYSTEM.CLUSTER.REPOSITORY.QUEUE, which is used to hold a persistent copy of the repository.
- A local queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE, which is the transmission queue for all destinations in the cluster. For performance reasons, you should define this queue with an index type of CORRELID (as shown in the sample queue definition).

These queues typically contain large numbers of messages.

You cannot define these queues as shared queues.

Queue sharing group queues

To use shared channels and intra-group queuing, you need to define the following objects:

- A shared queue with the name SYSTEM.QSG.CHANNEL.SYNCQ, which is used to hold synchronization information for shared channels.
- A shared queue with the name SYSTEM.QSG.TRANSMIT.QUEUE, which is used as the transmission queue for intra-group queuing. If you are running in a queue sharing group, you must define this queue, even if you are not using intra-group queuing.

Storage classes

You are recommended to define the following six storage classes. You must define four of them because they are required by IBM MQ. The other storage class definitions are recommended because they are used in the sample queue definitions.

DEFAULT (required)
This storage class is used for all message queues that are not performance critical and that don't fit in to any of the other storage classes. It is also the supplied default storage class if you do not specify one when defining a queue.

NODEFINE (required)
This storage class is used if the storage class specified when you define a queue is not defined.

REMOTE (required)
This storage class is used primarily for transmission queues, that is, system related queues with short-lived performance-critical messages.

SYSLNGLV
This storage class is used for long-lived, performance-critical messages.

SYSTEM (required)
This storage class is used for performance critical, system related message queues, for example the SYSTEM.CHANNEL.SYNQ and the SYSTEM.CLUSTER.* queues.

SYSVOLAT
This storage class is used for short-lived, performance-critical messages.

You can modify their attributes and add other storage class definitions as required.

Defining the system object dead-letter queue

The dead-letter queue is used if the message destination is not valid. IBM MQ puts such messages on a local queue called the dead-letter queue. Although having a dead-letter queue is not mandatory, you should regard it as essential, especially if you are using distributed queuing or one of the IBM MQ bridges.

Do **not** define the dead-letter queue as a shared queue. A put to a local queue on one queue manager might get put to the dead letter queue. If the dead letter queue was a shared queue, a dead letter queue handler on a different system could process the message and put it on a queue with the same name, but because this is on a different queue manager, it would be the wrong queue, or have a different security profile. If the queue did not exist, it would fail to reprocess it.

If you decide to define a dead-letter queue, you must also tell the queue manager its name. To do this use the ALTER QMGR DEADQ(*queue-name*) command. For more information see Displaying and altering queue manager attributes.

Default transmission queue

The default transmission queue is used when no other suitable transmission queue is available for sending messages to another queue manager. If you define a default transmission queue, you must also define a channel to serve the queue. If you do not do this, messages that are put on to the default transmission queue are not transmitted to the remote queue manager and remain on the queue.

If you decide to define a default transmission queue, you must also tell the queue manager its name. To do this use the ALTER QMGR command.

Internal queues

- **Pending data queue**
  - A queue defined for internal use, SYSTEM.PENDING.DATA.QUEUE, supports the use of durable subscriptions in a JMS publish/subscribe environment.
- **JMS 2.0 delivery delay staging queue**
  - If the delivery delay functionality provided by JMS 2.0 is used then an internal staging queue, SYSTEM.DDELAY.LOCAL.QUEUE, must be defined. This queue is used by the queue manager to temporarily store messages sent with a non-zero delivery delay until the delivery delay is completed, and the message is put to its target destination. A sample definition for this queue is provided, commented out, in CSQ4INSG.
  - When you define the SYSTEM.DDELAY.LOCAL.QUEUE queue, you must set the STGCLASS, MAXMSGL and MAXDEPTH attributes for the anticipated number of messages that will be sent with a delivery delay. Additionally when defining the SYSTEM.DDELAY.LOCAL.QUEUE queue ensure that only the queue manager can put messages to this queue. Care should be taken to ensure that no user identifier has the authority to put messages to this queue.

Channel authentication queue

For internal use of channel authentication the SYSTEM.CHLAUTH.DATA.QUEUE queue is required. Sample definitions are supplied with IBM MQ to help you define these objects. This sample is described in CSQ4INSA, which also defines some default rules.

z/OS Tuning your queue manager on IBM MQ for z/OS

There are few simple steps that you can take to ensure that your queue manager is tuned to avoid basic performance problems.

There are a number of ways in which you can improve the performance of your queue manager, which are controlled by queue manager attributes set by the ALTER QMGR command. This section contains information about how you can do this by setting the maximum number of messages allowed on the queue manager, or by performing 'housekeeping' on the queue manager. IBM MQ SupportPac MP16 - IBM MQ per z/OS Capacity planning & tuning gives more information on performance and tuning.

Syncpoints

One of the roles of the queue manager is syncpoint control within an application. An application constructs a unit of work containing any number of MQPUT or MQGET calls terminated with an MQCMIT call.

As the number of MQPUT or MQGET calls within the scope of one MQCMIT increases, the performance cost of the commit increases significantly. Applications, in general, should be designed to not MQPUT/MQGET a large number of messages in a single synchpoint.

You can administratively limit the number of messages within any single syncpoint by using the MAXUMSGS queue manager attribute. If an application exceeds this limit it receives MQRC_SYNCPOINT_LIMIT_REACHED on the MQPUT,MQPUT1, or MQGET call which exceeds the limit. The application should then issue MQCMIT or MQBACK as appropriate.

The default value of MAXUMSGS is 10000. This value can be lowered if you want to enforce a lower limit, which can also help protect against looping applications. Before reducing MAXUMSGS make sure you understand your existing applications to ensure they do not exceed the limit, or can tolerate the MQRC_SYNCPOINT_LIMIT_REACHED return code

Expired messages

Messages that have expired are discarded by the next appropriate MQGET call. However, if no such call occurs, the expired messages are not discarded, and, for some queues, particularly those where message retrieval is done by MessageId, CorrelId, or GroupId and the queue is indexed for performance, many expired messages can accumulate. The queue manager can periodically scan any queue for expired messages, which are then deleted. You can choose how often this scanning takes place, if at all. There are two ways of doing this:

Explicit request
: You can control which queues are scanned and when. Issue the REFRESH QMGR TYPE(EXPIRY) command, specifying the queue or queues that you want to be scanned.

Periodic scan
: You can specify an expiry interval in the queue manager object by using the EXPRYINT attribute. The queue manager maintains information about the expired messages on each queue, and knows at what time a scan for expired messages is worthwhile. Each time that the EXPRYINT interval is reached, the queue manager looks for candidate queues that are worth scanning for expired messages, and scans only those queues that it deems to be worthwhile. It does not scan all queues. This avoids any processor time being wasted on unnecessary scans.

  Shared queues are only scanned by one queue manager in the queue sharing group. Generally, the first queue manager to restart or the first to have the EXPRYINT set performs the scan.

Note: You must set the same EXPRYINT value for all queue managers within a queue sharing group.

z/OS Sample definitions supplied with IBM MQ for z/OS

Use this topic as a reference for the sample JCL, and code supplied with IBM MQ for z/OS.

The following sample definitions are supplied with IBM MQ in the thlqual.SCSQPROC library. You can use them to define the system objects and to customize your own objects. You can include some of them in the initialization input data sets (described in Initialization commands).

Table 22. IBM MQ sample definitions for system objects

Initialization input data set	Sample name
CSQINP1	CSQ4INP1 CSQ4INPR
CSQINP2	CSQ4INSA CSQ4INYS [1] CSQ4INSX CSQ4INSG CSQ4INSR CSQ4INSS CSQ4INSJ CSQ4INSM CSQ4INYG CSQ4INYR CSQ4INYC CSQ4INYD CSQ4INSC
CSQINPT	CSQ4INST CSQ4INYT
Other	CSQ4DISP CSQ4INPX CSQ4IVPQ CSQ4IVPG CSQ4MSTR CSQ4MSRR CSQ4QMIN

Note:

1. The order of these sample definitions is important: an error occurs if INYS, INSX, and INSG are ordered incorrectly.

CSQINP1 samples

Use the sample CSQINP1 data set thlqual.SCSQPROC(CSQ4INP1) when you are using one page set for each class of message, or thlqual.SCSQPROC(CSQ4INPR) when using multiple page sets for the major classes of message. It contains definitions of buffer pools, page set to buffer pool associations, and an ALTER SECURITY command. Include the sample in the CSQINP1 concatenation of your queue manager started task procedure.

CSQINP2 samples

CSQ4INSG system object sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSG) contains definitions for the following system objects for general use:

- System default objects
- System command objects
- System administration objects
- Other objects for system use

You must define the objects in this sample, but you need to do it only once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across queue manager shutdown and restart. You must not change the object names, but you can change their attributes if required.

When the following conditions are met, one message is put to the SYSTEM.DURABLE.SUBSCRIBER.QUEUE queue (even if publish subscribe is not active):

- The QMGR attribute PSMODE is set to DISABLED
- The sample object CSQ4INST statement `DEFINE SUB('SYSTEM.DEFAULT.SUB')` is present.

To avoid this, delete or comment out the `DEFINE SUB('SYSTEM.DEFAULT.SUB')` statement.

The JMS 2.0 delivery delay staging queue, SYSTEM.DDELAY.LOCAL.QUEUE only need be defined if JMS 2.0 delivery delay is used. By default, the queue definition is commented out, which you can uncomment if required.

CSQ4INSA system object and authentication sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSA) contains the channel authentication system queue definition. This queue holds the channel authentication records. It also contains the default channel authentication rules.

You must define the objects in this sample if CHLAUTH is ENABLED on the queue manager and you want to run channels, or you want to SET or DISPLAY CHLAUTH record. You only need to define them once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across a queue manager shutdown and restart, you must not change the queue name.

CSQ4INSS system object sample

You can define additional system objects if you are using queue sharing groups.

Sample data set thlqual.SCSQPROC(CSQ4INSS) contains sample commands for use with CF structures and a set of definitions for the system objects required for shared channels and intra-group queuing.

You cannot use this sample as is; you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required commands.

When you are defining group or shared objects, you need to include them in the CSQINP2 DD concatenation for only one queue manager in the queue sharing group.

CSQ4INSX system object sample

You must define additional system objects if you are using distributed queuing and clustering.

Sample data set thlqual.SCSQPROC(CSQ4INSX) contains the queue definitions required. You can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or

you can use it as input to the COMMAND function in CSQUTIL utility to issue the required DEFINE commands.

There are two types of object definitions:

- SYSTEM.CHANNEL.xx, needed for any distributed queuing
- SYSTEM.CLUSTER.xx, needed for clustering

CSQ4INSJ system JMS object sample

Defines queues used in the JMS publish/subscribe domain.

CSQ4INSM system object sample

If you are using advanced message security you must define additional system objects. Sample data set thlqual.SCSQPROC(CSQ4INSM) contains the queue definitions required.

CSQ4INSR object sample

Defines queues used by WebSphere Application Server and brokers.

CSQ4INYD object sample

If you are using distributed queuing and you need to set up your own queues, processes, and channels.

Sample data set thlqual.SCSQPROC(CSQ4INYD) contains sample definitions that you can use for customizing your distributed queuing objects. It comprises:

- A set of definitions for the sending end
- A set of definitions for the receiving end
- A set of definitions for using clients

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. (This is preferable because it means that you don't have to redefine these objects each time you restart the queue manager).

CSQ4INYC object sample

If you are using clustering, definitions equivalent to the channel definitions and remote queue definitions of distributed queuing are created automatically, when needed. However, some manual channel definitions are needed - a cluster-receiver channel for the cluster and a cluster-sender definition to at least one cluster repository queue manager.

The sample data set: thlqual.SCSQPROC(CSQ4INYC) contains the following sample definitions that you can use for customizing your clustering objects:

- Definitions for the queue manager
- Definitions for the receiving channel
- Definitions for the sending channel
- Definitions for cluster queues
- Definitions for lists of clusters

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. This is preferable because it means that you don't have to redefine these objects each time that you restart IBM MQ.

CSQ4INYG object sample

The sample data set: thlqual.SCSQPROC(CSQ4INYG) contains the following sample definitions that you can use for customizing your own objects for general use:

- Dead-letter queue
- Default transmission queue
- CICS adapter objects

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. This is preferable because it means that you don't have to redefine these objects each time that you restart IBM MQ.

In addition to the sample definitions here, you can use the system object definitions as the basis for your own resource definitions. For example, you can make a working copy of SYSTEM.DEFAULT.LOCAL.QUEUE and name it MY.DEFAULT.LOCAL.QUEUE. You can then change any of the parameters in this copy as required. You could then issue a DEFINE command by whichever method you choose, provided you have the authority to create resources of that type.

Default transmission queue
Read the Default transmission queue description before you decide whether you want to define a default transmission queue.

- If you decide that you do want to define a default transmission queue, remember that you must also define a channel to serve it.
- If you decide that you do not want to define one, remember to remove the DEFXMITQ statement from the ALTER QMGR command in the sample.

CICS adapter objects
The sample defines an initiation queue named CICS01.INITQ. This queue is used by the IBM MQ -supplied CKTI transaction. You can change the name of this queue; however it must match the name specified in the CICS system initialization table (SIT) or SYSIN override in the INITPARM statement.

CSQ4INYS/CSQ4INYR object samples

Storage class definitions for using:

- one page set for each class of message
- multiple page sets for major classes of message

For example, SYSTEM.COMMAND.INPUT uses STGCLASS('SYSVOLAT'), and SYSTEM.CLUSTER.TRANSMIT.QUEUE uses STGCLASS('REMOTE'). In CSQ4INYS, both of those storage classes use the same page set. In CSQ4INYR, those storage classes use different page sets in order to lessen the impact of the transmission queue filling.

CSQINPT samples

CSQ4INST
The sample data set: thlqual.SCSQPROC(CSQ4INST) contains the definition for the system default subscription.

You must define the object in this sample, but you need to do it only once when the publish/subscribe engine is first started. Including the definition in the CSQINPT data set is the best way to do this. It is maintained across queue manager shutdown and restart. You must not change the object name, but you can change their attributes if required.

CSQ4INYT
The sample data set: thlqual.SCSQPROC(CSQ4INYT) contains a set of commands that you might want to run when the publish/subscribe engine is started. This sample displays Topic and Subscription information.

Other

CSQ4DISP display sample

The sample data set: thlqual.SCSQPROC(CSQ4DISP) contains a set of generic DISPLAY commands that display all the defined resources on your queue manager. This includes the definitions for all IBM MQ objects and definitions such as storage classes and trace. These commands can generate a large amount of output. You can use this sample in the CSQINP2 data set or as input to the COMMAND function of the CSQUTIL utility.

CSQ4INPX sample

The sample data set: thlqual.SCSQPROC(CSQ4INPX) contains a set of commands that you might want to execute each time the channel initiator starts. You must customize this sample before use; you can then include it in the CSQINPX data set for the channel initiator.

CSQ4IVPQ and CSQ4IVPG samples

The sample data sets: thlqual.SCSQPROC(CSQ4IVPQ) and thlqual.SCSQPROC(CSQ4IVPG) contain sets of DEFINE commands that are required to run the installation verification programs (IVPs).

You can include these samples in the CSQINP2 data set. When you have run the IVPs successfully, you do not need to run them again each time the queue manager is restarted. Therefore, you do not need to keep these samples in the CSQINP2 concatenation permanently.

CSQ4MSTR and CSQ4MSRR samples

These are sample started task procedures for the queue manager: thlqual.SCSQPROC(CSQ4MSTR) and thlqual.SCSQPROC(CSQ4MSRR).

CSQ4MSRR uses CSQ4INYR in the CSQINP2 concatenation so that important queues are spread across different page sets.

You can remove the comments, so that you can use the CSQMINI card for newly created queue managers if required.

CSQ4QMIN sample

A sample QMINI data set, thlqual.SCSQPROC(CSQ4QMIN).

See QMINI data set for details of the QMINI data set and the **TransportSecurity** stanza.

z/OS Recovery and restart on z/OS

Use the links in this topic to find out about the features of IBM MQ for z/OS for restart and recovery.

IBM MQ for z/OS has robust features for restart and recovery. For information about how a queue manager recovers after it has stopped, and what happens when it is restarted, see the following subtopics:

- “How changes are made to data in IBM MQ for z/OS” on page 254
- “How consistency is maintained in IBM MQ for z/OS” on page 255
- “What happens during termination in IBM MQ for z/OS” on page 257
- “What happens during restart and recovery in IBM MQ for z/OS” on page 258
- “How in-doubt units of recovery are resolved” on page 260
- “Shared queue recovery” on page 263

Related concepts

z/OS IBM MQ for z/OS recovery actions

Related tasks

Planning for backup and recovery

z/OS Administering z/OS

Related reference

z/OS Messages for IBM MQ for z/OS

z/OS How changes are made to data in IBM MQ for z/OS

IBM MQ for z/OS must interact with other subsystems to keep all the data consistent. This topic contains information about *units of recovery*, what they are and how they are used in *back outs*.

Units of recovery

A *unit of recovery* is the processing done by a single queue manager for an application program, that changes IBM MQ data from one point of consistency to another. A *point of consistency* - also called a *syncpoint* or *commit point* - is a point in time when all the recoverable data that an application program accesses is consistent.

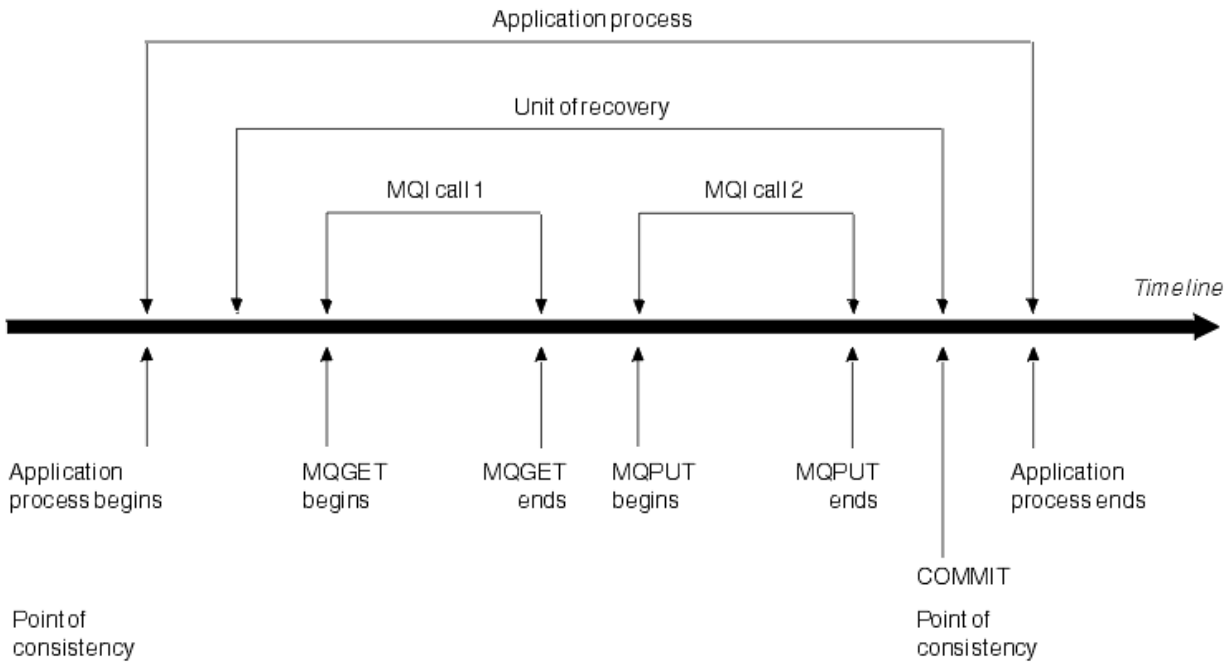


Figure 75. A unit of recovery within an application program

A unit of recovery begins with the first change to the data after the beginning of the program or following the previous point of consistency; it ends with a later point of consistency. Figure 75 on page 254 shows the relationship between units of recovery, the point of consistency, and an application program. In this example, the application program makes changes to queues through MQI calls 1 and 2. The application program can include more than one unit of recovery or just one. However, any complete unit of recovery ends in a commit point.

For example, a bank transaction transfers funds from one account to another. First, the program subtracts the amount from the first account, account A. Then, it adds the amount to the second account, B. After subtracting the amount from A, the two accounts are inconsistent and IBM MQ cannot commit. They become consistent when the amount is added to account B. When both steps are complete, the program can announce a point of consistency through a commit, making the changes visible to other application programs.

Normal termination of an application program automatically causes a point of consistency. Some program requests in CICS and IMS programs also cause a point of consistency, for example, EXEC CICS SYNCPOINT.

Backing out work

If an error occurs within a unit of recovery, IBM MQ removes any changes to data, returning the data to its state at the start of the unit of recovery; that is, IBM MQ backs out the work. The events are shown in Figure 76 on page 255.

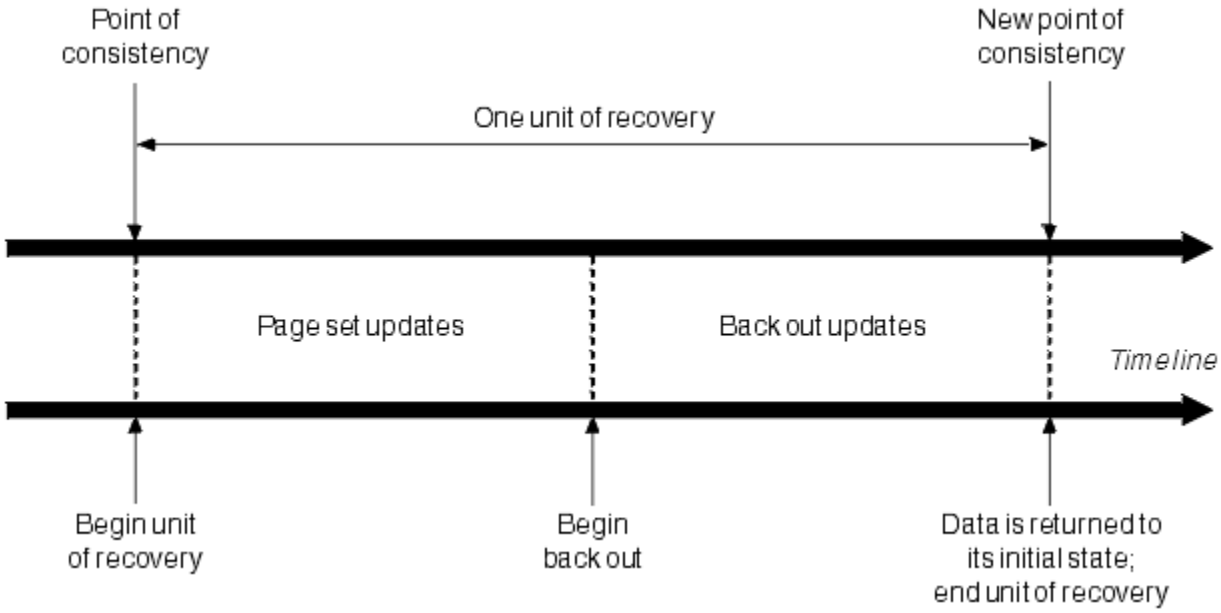


Figure 76. A unit of recovery showing back out

z/OS How consistency is maintained in IBM MQ for z/OS

Data in IBM MQ for z/OS must be consistent with batch, CICS, IMS, or TSO. Any data changed in one must be matched by a change in the other.

Before one system commits the changed data, it must know that the other system can make the corresponding change. So, the systems must communicate.

During a *two-phase commit* (for example under CICS), one subsystem coordinates the process. That subsystem is called the *coordinator*; the other is the *participant*. CICS or IMS is always the coordinator in interactions with IBM MQ, and IBM MQ is always the participant. In the batch or TSO environment, IBM MQ can participate in two-phase commit protocols coordinated by z/OS RRS.

During a *single-phase commit* (for example under TSO or batch), IBM MQ is always the coordinator in the interactions and completely controls the commit process.

In a WebSphere Application Server environment, the semantics of the JMS session object determine whether single-phase or two-phase commit coordination is used.

Consistency with CICS or IMS

The connection between IBM MQ and CICS or IMS supports the following syncpoint protocols:

- Two-phase commit - for transactions that update resources owned by more than one resource manager.

This is the standard distributed syncpoint protocol. It involves more logging and message flows than a single-phase commit.

- Single-phase commit - for transactions that update resources owned by a single resource manager (IBM MQ).

  This protocol is optimized for logging and message flows.

- Bypass of syncpoint - for transactions that involve IBM MQ but which do nothing in the queue manager that requires a syncpoint (for example, browsing a queue).

In each case, CICS or IMS acts as the syncpoint manager.

The stages of the two-phase commit that IBM MQ uses to communicate with CICS or IMS are as follows:

1. In phase 1, each system determines independently whether it has recorded enough recovery information in its log, and can commit its work.

   At the end of the phase, the systems communicate. If they agree, each begins the next phase.
2. In phase 2, the changes are made permanent. If one of the systems abends during phase 2, the operation is completed by the recovery process during restart.

Illustration of the two-phase commit process

Figure 77 on page 256 illustrates the two-phase commit process. Events in the CICS or IMS coordinator are shown on the upper line, events in IBM MQ on the lower line.

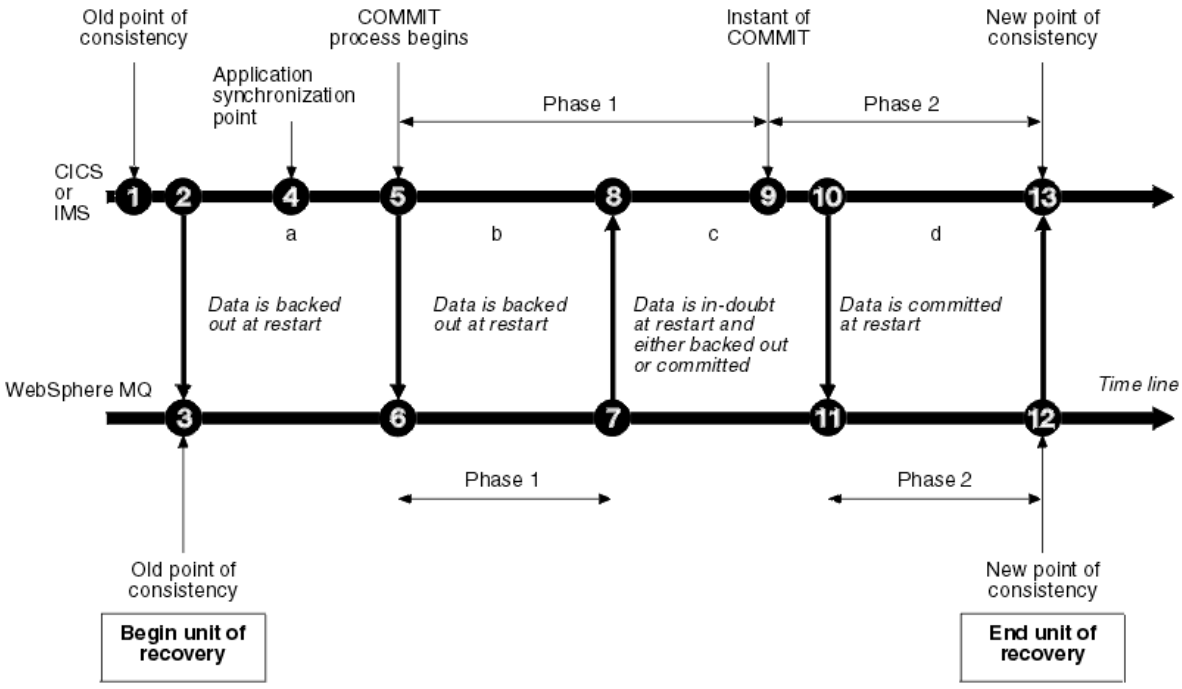


Figure 77. The two-phase commit process

The numbers in the following section are linked to those shown in the figure.

1. The data in the coordinator is at a point of consistency.
2. An application program in the coordinator calls IBM MQ to update a queue by adding a message.
3. This starts a unit of recovery in IBM MQ.
4. Processing continues in the coordinator until an application synchronization point is reached.
5. The coordinator then starts commit processing. CICS programs use a SYNCPOINT command or a normal application termination to start the commit. IMS programs can start the commit by using

a CHKP call, a SYNC call, a GET UNIQUE call to the IOPCB, or a normal application termination. Phase 1 of commit processing begins.

6. As the coordinator begins phase 1 processing, so does IBM MQ.
7. IBM MQ successfully completes phase 1, writes this fact in its log, and notifies the coordinator.
8. The coordinator receives the notification.
9. The coordinator successfully completes its phase 1 processing. Now both subsystems agree to commit the data changes, because both have completed phase 1 and could recover from any errors. The coordinator records in its log the instant of commit - the irrevocable decision of the two subsystems to make the changes.

   The coordinator now begins phase 2 of the processing - the actual commitment.
10. The coordinator notifies IBM MQ to begin its phase 2.
11. IBM MQ logs the start of phase 2.
12. Phase 2 is successfully completed, and this is now a new point of consistency for IBM MQ. IBM MQ then notifies the coordinator that it has finished its phase 2 processing.
13. The coordinator finishes its phase 2 processing. The data controlled by both subsystems is now consistent and available to other applications.

How consistency is maintained after an abnormal termination

When a queue manager is restarted after an abnormal termination, it must determine whether to commit or to back out units of recovery that were active at the time of termination. For some units of recovery, IBM MQ has enough information to make the decision. For others, it does not, and must get information from the coordinator when the connection is reestablished.

Figure 77 on page 256 shows four periods within the two phases: a, b, c, and d. The status of a unit of recovery depends on the period in which termination happened. The status can be one of the following:

In flight
The queue manager ended before finishing phase 1 (period a or b); during restart, IBM MQ backs out the updates.

In doubt
The queue manager ended after finishing phase 1 and before starting phase 2 (period c); only the coordinator knows whether the error happened before or after the commit (point 9). If it happened before, IBM MQ must back out its changes; if it happened after, IBM MQ must make its changes and commit them. At restart, IBM MQ waits for information from the coordinator before processing this unit of recovery.

In commit
The queue manager ended after it began its own phase 2 processing (period d); it makes committed changes.

In backout
The queue manager ended after a unit of recovery began to be backed out but before the process was complete (not shown in the figure) during restart, IBM MQ continues to back out the changes.

z/OS What happens during termination in IBM MQ for z/OS

A queue manager terminates normally in response to the STOP QMGR command. If a queue manager stops for any other reason, the termination is abnormal.

Note, that during queue manager termination, IBM MQ internally issues the command

```
DISPLAY CONN(*) TYPE(CONN) ALL WHERE (APPLTYPE NE SYSTEMAL)
```

so that you are aware of what threads might prevent the queue manager from completing shutdown.

SYSTEMAL matches APPLTYPES of either SYSTEM or CHINIT, so the DISPLAY CONN command filtering application types not matching SYSTEMAL, returns to the joblog information about threads that could be preventing normal shutdown.

Normal termination

In a normal termination, IBM MQ stops all activity in an orderly way. You can stop IBM MQ using either quiesce, force, or restart mode. The effects are given in Table 23 on page 258.

Table 23. Termination using QUIESCE, FORCE, and RESTART

Thread type	QUIESCE	FORCE	RESTART
Active threads	Run to completion	Back out	Back out
New threads	Can start	Not permitted	Not permitted
New connections	Not permitted	Not permitted	Not permitted

Batch applications are notified if a termination occurs while the application is still connected.

With CICS, a current thread runs only to the end of the unit of recovery. With CICS, stopping a queue manager in quiesce mode stops the CICS adapter, and so if an active task contains more than one unit of recovery, the task does not necessarily run to completion.

If you stop a queue manager in force or restart mode, no new threads are allocated, and work on connected threads is rolled back. Using these modes can create in-doubt units of recovery for threads that are between commit processing phases. They are resolved when IBM MQ is reconnected with the controlling CICS, IMS, or RRS subsystem.

When you stop a queue manager, in any mode, the steps are:

1. Connections are ended.
2. IBM MQ ceases to accept commands.
3. IBM MQ ensures that any outstanding updates to the page sets are completed.
4. The DISPLAY USAGE command is issued internally by IBM MQ so that the restart RBA is recorded on the z/OS console log.
5. The shutdown checkpoint is taken and the BSDS is updated.

Terminations that specify quiesce mode do not affect in-doubt units of recovery. Any unit that is in doubt remains in doubt.

Abnormal termination

An abnormal termination can leave data in an inconsistent state, for example:

- A unit of recovery has been interrupted before reaching a point of consistency.
- Committed data has not been written to page sets.
- Uncommitted data has been written to page sets.
- An application program has been interrupted between phase 1 and phase 2 of the commit process, leaving the unit of recovery in doubt.

IBM MQ resolves any data inconsistencies arising from abnormal termination during restart and recovery.

z/OS What happens during restart and recovery in IBM MQ for z/OS

IBM MQ uses its recovery log and the bootstrap data set (BSDS) to determine what to recover when it restarts. The BSDS identifies the active and archive log data sets, and the location of the most recent IBM MQ checkpoint in the log.

Introduction to restart and recovery

After IBM MQ has been initialized, the queue manager restart process takes place as follows:

- Log initialization
- Current status rebuild
- Forward log recovery
- Backward log recovery
- Queue index rebuilding

When recovery has been completed:

- Committed changes are reflected in the data.
- In-doubt activity is reflected in the data. However, the data is locked and cannot be used until IBM MQ recognizes and acts on the in-doubt decision.
- Interrupted in-flight and in-abort changes have been removed from the queues. The messages are consistent and can be used.
- A new checkpoint has been taken.
- New indexes have been built for indexed queues containing persistent messages (described in “Rebuilding queue indexes” on page 260).

If dual BSDSs are in use, IBM MQ checks the consistency of the time stamps in the BSDS:

- If both copies of the BSDS are current, IBM MQ tests whether the two time stamps are equal. If they are not, IBM MQ issues message CSQJ120E and terminates. This can happen when the two copies of the BSDS are maintained on separate DASD volumes and one of the volumes was restored while the queue manager was stopped. IBM MQ detects the situation at restart.
- If one copy of the BSDS was de-allocated, and logging continued with a single BSDS, a problem could arise. If both copies of the BSDS are maintained on a single volume, and the volume was restored, or if both BSDS copies were restored separately, IBM MQ might not detect the restoration. In that case, log records not noted in the BSDS would be unknown to the system.

Batch applications are not notified when restart occurs *after* the application has requested a connection.

Understanding the log range required for recovery

During restart, the range of log data which must be read is dependent on many factors:

- At the time of an abnormal termination, there are typically many incomplete units of work in the system. As described earlier, restart processing will bring the system to a state of consistency, which may involve backing out inflight units of work, or recovering locks on indoubt units of work. Unit of work recovery requires that all unit of work log records for inflight, in-backout, and in-doubt units of work are available. IBM MQ will 'shunt' old units of work, so that unit of work recovery can be performed using a much smaller range of log data.
- At the time of an abnormal termination, there are typically many persistent updates which are only held in the buffer pool cache. They have not yet been written to disk. These changes must be read from the log, and reapplied to the data held in page sets. Page set recovery RBAs in the checkpoint describe the lowest log RBA required for updating the page sets to a consistent state.
- If old page sets have been introduced into the system, for example, a page set backup has been introduced to recover from a media failure, all the changes must be read from the log from the time the backup was taken. These changes are reapplied to the data held in the page set being recovered. Page set recovery RBAs held in page 0 of the page set describe the lowest log RBA required for media recovery of a page set.
- If using persistent messages on shared queues, a range of log data is required to recover CFSTRUCTs which are holding persistent messages. The earliest log data that would be required to perform a CFSTRUCT recovery, is from around the time of the old CFSTRUCT BACKUP.

During normal running, the DISPLAY USAGE TYPE(DATASET) command can be used to view the recovery log range associated with these factors (it is unable to provide information due to reintroducing old page sets, of course). To avoid any issues that might prolong a queue manager restart in the event of an abnormal termination, regularly monitor the values output from DISPLAY USAGE TYPE(DATASET).

In addition, the queue manager issues informational messages relating to these factors:

- CSQJ160I and CSQJ161I warn of long running units of work.
- CSQR026I and CSQR027I provide information about whether these long running units of work have been successfully shunted.
- CSQE040I and CSQE041E warn that structure backups are getting old, and consequently a RECOVER CFSTRUCT operation would take a long time.

Determining which application has a long running unit of work

It is possible to determine the application with the long-running unit of work. To do this, you use the DISPLAY CONN command.

The DISPLAY CONN command returns connection information for all the applications connected to the queue manager, together with additional information that helps you determine which application(s) currently have a long-running unit of work. The information returned by the DISPLAY CONN command is similar to the information returned by the DISPLAY QSTATUS command, but the main difference is that DISPLAY CONN displays information about objects, and transactional information for a particular connection, rather than details of which connections are associated with a particular object.

For each connected application, the DISPLAY CONN command returns the following information:

- Basic information including the Connection Id and PID.
- Transactional information for that connection, including the time and date when the transaction was created (that is, when the first MQGET/PUT was made under syncpoint), and when the transaction first wrote to the log.
- Log time information indicating which application still has a long running unit of work.
- A list of all objects that the connection currently has open. Details for each object are returned as a separate message, with the Connection Id used as a key. Because there are different types of object such as queues and queue managers, the information displayed with the object is specific to its object type.

Rebuilding queue indexes

To increase the speed of MQGET operations on a queue where messages are not retrieved sequentially, you can specify that you want IBM MQ to maintain an index of the message or correlation identifiers or groupid for all the messages on that queue.

When a queue manager is restarted, these indexes are rebuilt for each queue. This applies only to persistent messages; nonpersistent messages are deleted at restart. If your indexed queues contain large numbers of persistent messages, this increases the time taken to restart the queue manager.

You can choose to have indexes rebuilt asynchronously to queue manager startup by using the QINDXBLD parameter of the CSQ6SYSP macro. If you set QINDXBLD=NOWAIT, IBM MQ restarts without waiting for indexes to rebuild.

z/OS How in-doubt units of recovery are resolved

If IBM MQ loses its connection to another resource manager, it typically attempts to recover all inconsistent objects at restart.

If IBM MQ loses its connection to CICS, IMS, or RRS, it normally attempts to recover all inconsistent objects at restart. The information required to resolve in-doubt units of recovery must come from the coordinating system. The next sections describe the process of resolution for different environments.

- How in-doubt units of recovery are resolved from CICS
- How in-doubt units of recovery are resolved from IMS
- How in-doubt units of recovery are resolved from RRS
- How in-doubt units of recovery with a GROUP unit of recovery disposition are resolved

How in-doubt units of recovery are resolved from CICS

Under some circumstances, CICS cannot run the IBM MQ process to resolve in-doubt units of recovery. When this happens, IBM MQ sends one of the following messages:

- CSQC404E
- CSQC405E
- CSQC406E
- CSQC407E

followed by the message CSQC408I.

For details of what these messages mean, see the Messaggi IBM MQ for z/OS , codici di completamento e di errore 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 Amministrazione di 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 Amministrazione di 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 Messaggi IBM MQ for z/OS , codici di completamento e di errore 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 Amministrazione di IBM MQ for z/OS.

How in-doubt units of recovery with a GROUP unit of recovery disposition are resolved

In-doubt transactions that have a GROUP unit of recovery disposition can be resolved by the transaction coordinator by any queue manager in the queue sharing group (QSG) where the GROUPUR queue manager attribute is enabled. Whenever a transaction coordinator reconnects it typically requests a list of any outstanding in-doubt transactions and then resolves them using information from its logs.

When a transaction coordinator, that has connected with a GROUP unit of recovery disposition, requests the list of in-doubt transactions, the list returned comprises all in-doubt transactions with a GROUP unit of recovery disposition that exist throughout the queue sharing group. This list is not dependent on which queue manager those in-doubt transactions were started on. A queue manager processing such a request compiles the list by communicating with all other active queue managers in the queue sharing group using the SYSTEM.QSG.UR.RESOLUTION.QUEUE. The queue manager then reads the logs of any inactive queue

managers, from their last checkpoint, to identify any additional in-doubt transactions that they would have reported had they been active.

When a transaction coordinator requests the resolution of an in-doubt transaction, the queue manager to which it is connected identifies whether the transaction was originated on itself and if so resolves it in the same way as transactions with a QMGR unit of recovery disposition. If the transaction was originated on another active queue manager in the QSG, a request to complete the resolution is routed to that queue manager using the SYSTEM.QSG.UR.RESOLUTION.QUEUE. In the case where the transaction was originated on an inactive queue manager in the QSG, any shared-queue work is resolved immediately, and a request to resolve any remaining private queue work is placed on the SYSTEM.QSG.UR.RESOLUTION.QUEUE. The inactive queue manager processes this request upon start-up before accepting new work. In this scenario, the original queue manager's logs still reflect that the unit of recovery is in doubt until it has restarted and processed the request.

z/OS Shared queue recovery

Use this topic to understand IBM MQ recovery and resilience of various components in the queue sharing group environment.

Transactional recovery

When an application issues a MQBACK call or terminates abnormally (for example, because of an EXEC CICS ROLLBACK or an IMS abend) thread-level information stored in the queue manager ensures that the in-flight unit of work is rolled back. MQPUT and MQGET operations within syncpoint on shared queues are rolled back in the same way as updates to non-shared queues.

Peer recovery

If a queue manager fails, it disconnects abnormally from the coupling facility structures that it is currently connected to. If the connection between the z/OS instance and the coupling facility fails (for example, physical link failure or power-off of a coupling facility or partition) this is also detected as an abnormal termination of the connection between the queue manager and the coupling facility structures involved. Other queue managers in the same queue sharing group that remain connected to that structure detect the abnormal disconnection and all attempt to initiate *peer recovery* for the failed queue manager on that structure. Only one of these queue managers initiates peer recovery successfully, but all the other queue managers cooperate in the recovery of units of work that were owned by the queue manager that failed.

If a queue manager fails when there are no peers connected to a structure, recovery is performed when another queue manager connects to that structure, or when the queue manager that failed restarts.

Peer recovery, often referred to as Peer Level Recovery (PLR), is performed on a structure by structure basis and it is possible for a single queue manager to participate in the recovery of more than one structure at the same time. However, the set of peers cooperating in the recovery of different structures might vary depending on which queue managers were connected to the different structures at the time of failure.

When the failed queue manager restarts, it reconnects to the structures that it was connected to at the time of failure, and recovers any remaining unresolved units of work that were not recovered by peer recovery.

Peer recovery is a multi-phase process. During the first phase, units of work that had progressed beyond the in-flight phase are recovered; this might involve committing messages for units of work that are in-commit and locking messages for units of work that are in-doubt. During the second phase, queues that had threads active against them in the failing queue manager are checked, uncommitted messages related to in-flight units of work are rolled back, and information about active handles on shared queues in the failed queue manager are reset. This means that IBM MQ resets any indicators that the failing queue manager had a shared queue open for input-exclusive, allowing other active queue managers to open the queue for input.

Shared queue definitions

The queue objects that represent the attributes of a shared queue are held in the shared Db2 repository used by the queue sharing group. Ensure that adequate procedures are in place for the backup and recovery of the Db2 tables used to hold IBM MQ objects. You can also use the IBM MQ CSQUTIL utility to create MQSC commands for replay into a queue manager to redefine IBM MQ objects, including shared queue and group definitions stored in Db2.

Logging

Queue sharing groups can support persistent messages, because the messages on shared queues can be logged in the queue manager logs.

Coupling facility and structure failures

There are two types of failure that can be reported for a coupling facility (CF) structure: structure failure and loss of connectivity. Sysplex services for data sharing (XES) inform IBM MQ of a CF structure failure or a CF failure with a structure failure event. If XES creates a loss of connectivity event this does not necessarily indicate that there is a problem with the structure, it might be that there is no connection available to communicate with the structure. It is possible that not all of the queue managers receive a loss of connectivity event for the structure; it depends on the configuration of connections to the CF. A loss of connectivity event can also be received because of operator commands, for example VARY PATH OFFLINE or CONFIG CHP OFFLINE.

The CF structures that are used by IBM MQ can be configured to use system-managed duplexing. This means that if there is a single failure, system-managed failover processing hides the failure of a structure or the loss of connectivity, and the queue manager is not informed of the failure. If there is a failure of both instances of a duplexed structure or connection, the queue manager receives the appropriate event and handles it in the same way as a failure event for a simplex structure. Details of how the queue manager handles the events are described in Scenarios.

In the unlikely event of a CF or structure failure, any nonpersistent messages stored in the affected application structures are lost. You can recover persistent messages using the RECOVER CFSTRUCT command. If a recoverable application structure has failed, any further application activity to this structure is prevented until the structure has been recovered.

To ensure that you can recover a CF structure in a reasonable period of time, take frequent backups, using the BACKUP CFSTRUCT command. You can choose to perform the backups on any queue managers in the queue sharing group or dedicate one queue manager to perform all the backups. Automate the process of taking backups to ensure that they are taken on a regular basis.

Each backup is written to the active log data set of the queue manager taking the backup. The shared queue Db2 repository records the name of the CF structure being backed up, the name of the queue manager doing the backup, the RBA range for this backup on that queue manager's log, and the backup time.

The administration structure contains information about incomplete units of work on shared queues at the time of any application structure failure so the administration structure must be available during RECOVER CFSTRUCT processing. If the administration structure has failed, all the queue managers in the queue sharing group must have rebuilt their administration structure entries before you can issue the RECOVER CFSTRUCT command.

Queue managers rebuild their administration structure entries automatically and without terminating. If a queue manager is not running at the time of the failure, its administration structure entries can be rebuilt by another queue manager in the queue sharing group that is running at the same or higher level.

To recover an application structure, issue a RECOVER CFSTRUCT command to the queue manager that you want to perform the recovery. You can recover a single CF structure or you can recover several CF structures simultaneously. You can recover using any queue manager in the queue sharing group, it does not have to be the one that performed the backup, or one that has been connected previously to the failed structure.

The RECOVER CFSTRUCT command uses the backup, located through the Db2 repository information (Db2 must therefore be available on the queue manager where recovery is being carried out), and recovers this to the point of failure.

The RECOVER CFSTRUCT command does this by applying log records from every queue manager in the queue sharing group that has performed an MQPUT or MQGET between the start of the backup and the time of failure, to any shared queue that maps to the CF structure. The resulting merging of the logs might require reading a considerable amount of log data because all the log data written by participating queue managers since the backup is read. You are strongly recommended to make frequent (for example, hourly) backups, especially if there are large messages within the backup.

Structure failure scenarios

Scenarios

If a failure is reported for a CF structure, the action taken by connected queue managers depends on the following:

- The type of failure reported by the XES component of z/OS to IBM MQ.
- The structure type (application or administration)
- The queue manager level
- The CFLEVEL of the IBM MQ CFSTRUCT object (2, 3, 4 or 5. This is not the CFLEVEL of the CFCC microcode)
- The RECAUTO attribute of an IBM MQ CFSTRUCT object at CFLEVEL(5)

The following scenarios describe what happens when a failure is reported for the administration structure:

- If a structure failure event is received for the administration structure, the structure is reallocated and rebuilt automatically without the queue manager terminating. A new instance of the structure is allocated by XES when a queue manager attempts to connect to it.

  When the queue manager has connected to the new instance of the structure, the queue manager writes the entries for itself into the structure. This processing is carried out by the queue manager and is not part of XES rebuild processing.

  If a queue manager was not running at the time of the failure, or terminates before recovery of its part of the administration structure has been completed, its administration structure entries can be rebuilt by another queue manager in the queue sharing group.

  Administration structure entries of a queue manager can only be rebuilt by another queue manager running at the same level or higher. If administration structure entries of a queue manager cannot be rebuilt by another queue manager in the queue sharing group, restart the queue manager so that it can complete the rebuild of its part of the structure.

Certain actions are suspended until the administration structure entries for all queue managers have been rebuilt. The suspended actions include the following:

- Opening and closing of shared queues.
- Committing or backing out units of recovery.
- Serialized applications connecting to or disconnecting from the queue manager.
- Backing up or recovering an application structure.

Any serialized applications that have already connected to the queue manager can continue processing. Any serialized application attempting to connect with the MQCNO_SERIALIZE_CONN_TAG_QSG or MQCNO_RESTRICT_CONN_TAG_QSG parameters receive the MQRC_CONN_TAG_NOT_USABLE return code.

When the administration structure entries for the queue manager have been rebuilt, the suspended actions are resumed.

The following scenarios describe what happens when a failure is reported for an application structure:

- If a structure failure event is received for an application structure, and the CFLEVEL is 1 or 2, the queue manager terminates. Restart the queue manager. The first queue manager to attempt to connect to the structure again causes XES to allocate a new instance of the structure.
- If a structure failure event is received for an application structure, and the CFLEVEL is 3, 4, or 5 the queue managers connected to the structure continue to run. Applications that do not use the queues in the failed structure can continue normal processing.

  However, applications that attempt operations on queues in the failed structure receive an MQRC_CF_STRUC_FAILED error until the structure has been successfully rebuilt, at which point the application can open the queues again.

  Structure rebuild is started automatically for CFLEVEL(5) application structures defined with RECAUTO(YES). Otherwise, the structure will be rebuilt when the RECOVER CFSTRUCT command is issued.

Resilience to coupling facility connectivity failures

What is resilience to coupling facility connectivity failures?

Resilience to coupling facility connectivity failures refers to the ability of queue managers in a queue sharing group to tolerate loss of connectivity to a coupling facility structure without terminating. This function also attempts to rebuild the structure in another coupling facility with better connectivity in order to regain access to shared queues as soon as possible.

What is partial loss of connectivity?

IBM MQ defines partial loss of connectivity as a situation where one or more systems in the sysplex lose connectivity to the coupling facility where the structure being accessed by the system is allocated, but at least one system in the sysplex maintains connectivity to the same coupling facility.

What is total loss of connectivity?

IBM MQ defines a total loss of connectivity as a situation where no systems in the sysplex have connectivity to the coupling facility and the structure allocated within it.

Why would you enable this function?

Resilience to coupling facility connectivity failures improves the availability of IBM MQ, allowing non-shared queues to remain available after a queue manager has lost connectivity to one or more coupling facility structures. Additionally, queue managers that lose connectivity to a coupling facility structure automatically attempt to rebuild the structure in another available coupling facility, improving the availability of the shared queues within the queue sharing group.

Considerations when enabling this function

A queue manager that tolerates loss of connectivity to coupling facility structures without terminating may not be able to reconnect to a coupling facility structure for some time if there is no alternative coupling facility available. Shared queues defined on a structure that has suffered loss of connectivity remain unavailable until connectivity to the structure is restored. In this situation, applications that connect into members of the queue sharing group in order to perform shared queue work may find that the shared queues they need to access are not available. To avoid this situation it is recommended that queue managers should be configured to terminate when connectivity to a coupling facility structure is lost. This termination forces applications to connect to another member of the queue sharing group that has connectivity to the coupling facility structures where the shared queues the application requires are defined.

Managing Resilience to coupling facility connectivity failures

How do I enable this functionality?

The following steps must be performed in order to enable resilience to coupling facility connectivity

1. Ensure that the CFRM couple data set has been formatted to support system-managed rebuild. This allows queue managers to initiate a system-managed rebuild to re-create a structure into an available coupling facility. Use the **DISPLAY XCF,COUPLE,TYPE=CFRM** command to determine the format of the CFRM couple data set. To support system-managed rebuild, the CFRM couple data set should be formatted by specifying:

   ```
   "ITEM NAME(SMREBLD) NUMBER(1)"
   ```

   Refer to the z/OS MVS Setting Up a Sysplex documentation for more information on formatting a CFRM couple data set.
2. Ensure that an alternative coupling facility is available and is in the CFRM preference list for all IBM MQ coupling facility structures. This enables the queue managers to attempt to rebuild structures into an alternative available coupling facility to restore access to the structures as soon as possible.

   IBM MQ structures must be defined with ENFORCEORDER(NO) in CFRM policy, so that XCF is able to choose the optimum CF in the configuration if IBM MQ needs to reallocate the structure.

   Refer to the z/OS MVS Setting Up a Sysplex documentation for more information about structure preference lists.
3. Alter all application coupling facility structures that need to tolerate loss of connectivity to CFLEVEL(5). This is the minimum level that can tolerate a loss of connectivity.
4. Determine the values required for the **QMGR CFCONLOS** and the **CFSTRUCT CFCONLOS** attributes and alter these accordingly. The **QMGR CFCONLOS** attribute controls whether loss of connectivity to the administration structure is tolerated, and the **CFSTRUCT CFCONLOS** attribute controls whether loss of connectivity is tolerated by each application coupling facility structure. If the default values for these attributes are retained, the queue manager terminates following loss of connectivity to any coupling facility structure.
5. Determine the values required for the **CFSTRUCT RECAUTO** attribute for each application coupling facility structure, and alter these accordingly. This attribute controls whether coupling facility structures should be automatically recovered using logged data following total loss of connectivity. If the default value for this attribute is retained, no automatic recovery is performed for application structures following total loss of connectivity.

Scenario 1 - Loss of connectivity to the administration structure

Queue managers can tolerate loss of connectivity to the administration structure without terminating.

When connectivity to the administration structure is lost by any queue manager that has been configured to tolerate loss of connectivity to the administration structure, all members of the queue sharing group disconnect from the administration structure. All active queue managers in the queue

sharing group then attempt to reconnect to the administration structure, causing it to be reallocated in the coupling facility with the best connectivity to all systems in the sysplex, and rebuild the administration structure data.

Note: This may not necessarily be the coupling facility which has the best connectivity to all systems that have active queue managers.

If a queue manager cannot reconnect to the administration structure, for example because none of the coupling facilities in the CFRM preference list for the administration structure are available, some shared queue operations remain unavailable until the queue manager can successfully reconnect to the administration structure and rebuild its administration structure data. Reconnection occurs automatically when a suitable coupling facility becomes available on the system.

Failure to connect to the administration structure during queue manager startup as a result of a lack of connectivity to the coupling facility, or no suitable coupling facility available to allocate the structure, is not tolerated. All active queue managers in the queue sharing group then attempt to reconnect to the administration structure, causing it to be reallocated in another coupling facility if one is available, and rebuild the administration structure data.

Scenario 2- Loss of connectivity to the application structure

Loss of connectivity to application structures at **CFLEVEL(5)** or higher can be tolerated without the queue manager terminating. Queue managers connected to application structures at **CFLEVEL(4)** or lower, or structures at **CFLEVEL(5)** that have not been configured to tolerate loss of connectivity, abend with reason code 00C510AB when connectivity to the structure is lost.

When connectivity is lost to an application structure that has been configured to tolerate loss of connectivity, all queue managers that lost connectivity to the structure disconnect. The subsequent behavior of the queue manager depends on whether the loss of connectivity is partial or total.

Partial loss of connectivity to an application structure

If the loss of connectivity is determined to be partial, queue managers that have lost connectivity to the structure attempt to initiate a system-managed rebuild in order to move the structure to another coupling facility with improved connectivity. If this rebuild is successful, both persistent and non-persistent messages in the structure are copied to the other coupling facility, and access to queues on the structure is restored. Queue managers that did not lose connectivity remain connected to the structure, however, operations that access the structure are delayed during the system-managed rebuild process.

If an application structure cannot be rebuilt to another coupling facility with improved connectivity, or some queue managers still do not have connectivity to the structure after it has been rebuilt in another coupling facility, queues defined on the structure remain unavailable on the queue managers that do not have connectivity to the structure until connectivity is restored to the coupling facility. Queue managers automatically reconnect to the structure when it becomes available and access to the shared queues defined on the structure are restored.

Total loss of connectivity to an application structure

If all MVS systems in the sysplex have lost connectivity to the coupling facility that the application structure is allocated in, z/OS deallocates the structure from the coupling facility whenever an attempt is made to reconnect to the structure. It is possible for the queue manager to attempt to reconnect to the structure for several reasons, such as an attempt by an application to open a shared queue, or a notification from the system that new coupling facility resources may have become available. It is therefore likely that all non-persistent messages in the affected structure are lost following total loss of connectivity to an application structure.

Recoverable application structures are automatically recovered following total loss of connectivity, if they have been defined with **RECAUTO(YES)**. The recovery starts almost immediately if an alternative coupling facility is available to allocate the structure in, or whenever such a coupling facility becomes available. If a structure has not been defined with **RECAUTO(YES)**, recovery can be started by issuing the **RECOVER CFSTRUCT** command. This recovers all persistent messages in the structure, but all non-persistent messages are lost. As this process involves reading the queue manager log it can take

some time to complete, therefore it is recommended that structure backups be taken regularly to reduce the time until access to the shared queues on the structure is restored.

Queue managers attempt to reconnect to non-recoverable application structures as soon as an application attempts to open a shared queue that is defined on the structure or a notification is received from the system that new coupling facility resources have become available. If a suitable coupling facility is available to allocate the structure in, a new structure is allocated and access to the shared queues defined on the structure is restored. As persistent messages cannot be put to queues defined in non-recoverable structures, all messages on the shared queues are lost.

Operational behavior

If an IBM WebSphere MQ 7.1, or later, queue manager, configured to tolerate loss of connectivity to a particular coupling facility structure loses connectivity, the members of the queue sharing group attempt to automatically recover from the failure and reconnect to the structure. This activity may involve reallocating the structure in another coupling facility with better connectivity if one is available. However, operator intervention may still be required to recover from the loss of connectivity.

Typically the required operator action is to:

1. Resolve the cause of the failure that resulting in the loss of connectivity.
2. Ensure that a coupling facility where the IBM MQ structures can be allocated is available on all systems in the sysplex

Any structures that have been automatically reallocated in another coupling facility after the loss of connectivity event, can be moved to the coupling facility with the optimal connectivity to all queue managers in the queue sharing group. If required, this can be done by initiating the system-managed rebuild command **SETXCF START,REBUILD** as documented in z/OS MVS Riferimento comandi di sistema.

In the case of a partial loss of connectivity to an application structure, the queue managers that lost connectivity to the structure attempt to initiate a system-managed rebuild. This process only allocates the structure in another coupling facility if that coupling facility has connectivity to all active queue managers currently connected to the structure. Therefore, it is possible that where the majority of queue managers in a queue sharing group have lost connectivity to an application structure, they are unable to rebuild the structure into another coupling facility due to the queue managers that are still connected to the original structure. In this situation the queue managers that are still connected to the original structure can be shut down to allow the structure to be rebuilt, or the **RESET CFSTRUCT ACTION(FAIL)** command can be issued to fail the structure. Recovery can be initiated on applicable structures by issuing the **RECOVER CFSTRUCT** command.

Note: When failing and recovering the structure, all non-persistent messages on the structure are lost.

z/OS Security concepts in IBM MQ for z/OS

Use this topic to understand the importance of security for IBM MQ, and the implications of not having adequate security settings on your system.

Why you must protect IBM MQ resources

IBM MQ handles the transfer of information that is potentially valuable. Applying security ensures that the resources IBM MQ owns and manages are protected from unauthorized access. Such access might lead to the loss or disclosure of the information.

You should ensure that none of the following resources are accessed or changed by any unauthorized user or process:

- Connections to IBM MQ
- IBM MQ objects such as queues, processes, and namelists
- IBM MQ transmission links, that is, IBM MQ channels

- IBM MQ system control commands
- IBM MQ messages
- Context information associated with messages

To provide the necessary security, IBM MQ uses the z/OS system authorization facility (SAF) to route authorization requests to an External Security Manager (ESM), for example Security Server (previously known as RACF). IBM MQ does no security verification of its own. Where distributed queuing or clients are being used, you might require additional security measures, for which IBM MQ provides channel authentication records, channel exits, the MCAUSER channel attribute, and TLS.

The decision to allow access to an object is made by the ESM and IBM MQ follows that decision. If the ESM cannot make a decision, IBM MQ prevents access to the object.

What happens if you do not protect IBM MQ resources

If you do nothing about security, the most likely effect is that *all* users can access and change *every* resource. This includes not only local users, but also those on remote systems using distributed queuing or clients, where the logon security controls might be less strict than is normally the case for z/OS.

To enable security checking you must do the following:

- Install and activate an ESM (for example, Security Server).
- Define the MQADMIN class if you are using an ESM other than Security Server.
- Activate the MQADMIN class.

You must consider whether using mixed-case resource names would be beneficial to your enterprise. If you do use mixed-case resource names in your ESM profiles you must define and activate the MXADMIN class.

z/OS Data Set Encryption

Data Set Encryption (DSE) provides the capability to encrypt z/OS data sets, so that the data they contain can only be viewed or modified by user IDs granted the specific permission. This provides encryption of data at rest in the file system, and prevents inadvertent disclosure of sensitive information to users who have a legitimate business need and permissions to manage the data sets themselves.

Prior to IBM MQ for z/OS 9.1.4, IBM MQ for z/OS does not support use of DSE with the active logs, page sets, and shared message data sets (SMDS) that provide the primary persistence mechanisms for IBM MQ messages.

Instead, Advanced Message Security provides an end-to-end encryption solution for IBM MQ messaging, which encompasses the entire IBM MQ network, encryption of data in flight, at rest, and even inside the runtime IBM MQ processes.

Other VSAM and sequential data sets used in an IBM MQ subsystem can be encrypted using DSE. For example:

- Bootstrap data set (BSDS)
- Sequential files holding system configuration (MQSC) commands read at startup using CSQINPx DDNAMEs
- IBM MQ archive logs, often used for long term archival of IBM MQ log data for audit purposes.

You can encrypt using DSE by allocating a dataclass that is defined with a data set key label. For more information, see Planning your log archive storage.

From IBM MQ for z/OS 9.1.4, IBM MQ for z/OS supports use of DSE with the active logs and page sets in addition to the support provided in earlier releases.

IBM MQ for z/OS does not support use of DSE for shared message data sets (SMDS).

See the section, confidentiality for data at rest on IBM MQ for z/OS with data set encryption. for more information.

Related concepts
Security concepts
Channel authentication records
Authority to work with IBM MQ objects on z/OS
Cryptographic security protocols: TLS
Related tasks
Setting up security on z/OS
Comparing link level security and application level security
Related reference
Messages for IBM MQ for z/OS

z/OS Security controls and options in IBM MQ for z/OS

You can specify whether security is turned on for the whole IBM MQ subsystem, and whether you want to perform security checks at queue manager or queue sharing group level. You can also control the number of user IDs checked for API-resource security.

Subsystem security

Subsystem security is a control that specifies whether any security checking is done for the whole queue manager. If you do not require security checking (for example, on a test system), or if you are satisfied with the level of security on all the resources that can connect to IBM MQ (including clients and channels), you can turn security checking off for the queue manager or queue sharing group so that no further security checking takes place.

This is the only check that can turn security off completely and determine whether any other security checks are performed or not. That is, if you turn off checking for the queue manager or queue sharing group, no other IBM MQ checking is done; if you leave it turned on, IBM MQ checks your security requirements for other IBM MQ resources.

You can also turn security on or off for particular sets of resources, such as commands.

Queue manager or queue sharing group level checking

You can implement security at queue manager level or at queue sharing group level. If you implement security at queue sharing group level, all the queue managers in the group share the same profiles. This means that there are fewer profiles to define and maintain, making security management easier. It also makes it easy to add a new queue manager to the queue sharing group because it inherits the existing security profiles.

It is also possible to implement a combination of both if your installation requires it, for example, during migration or if you have one queue manager in the queue sharing group that requires different levels of security to the other queue managers in the group.

Queue sharing group level security

Queue sharing group level security checking is performed for the entire queue sharing group. It enables you to simplify security administration because it requires you to define fewer security profiles. The authorization of a user ID to use a particular resource is handled at the queue sharing group level, and is independent of which queue manager that user ID is using to access the resource.

For example, say a server application runs under user ID SERVER and wants access to a queue called SERVER.REQUEST, and you want to run an instance of SERVER on each z/OS image in the sysplex. Rather than permitting SERVER to open SERVER.REQUEST on each queue manager individually (queue manager level security), you can permit access only at the queue sharing group level.

You can use queue sharing group level security profiles to protect all types of resource, whether local or shared.

Queue manager level security
You can use queue manager level security profiles to protect all types of resource, whether local or shared.

Combination of both levels
You can use a combination of both queue manager and queue sharing group level security.

You can override queue sharing group level security settings for a particular queue manager that is a member of that group. This means that you can perform a different level of security checks on an individual queue manager to those performed on the other queue managers in the group.

For more information, see Profiles to control queue sharing group or queue manager level security.

Controlling the number of user IDs checked

RESLEVEL is a Security Server profile that controls the number of user IDs checked for IBM MQ resource security. Normally, when a user attempts to access an IBM MQ resource, Security Server checks the relevant user ID or IDs to see if access is allowed to that resource. By defining a RESLEVEL profile you can control whether zero, one or, where applicable, two user IDs are checked.

These controls are done on a connection by connection basis, and last for the life of the connection.

There is only one RESLEVEL profile for each queue manager. Control is implemented by the access that a user ID has to this profile.

Mixed case or uppercase IBM MQ RACF classes

You can now use mixed case RACF profile support, which allows you to use mixed case resource names and define IBM MQ RACF profiles to protect them.

You can choose to either:

- Continue using uppercase only IBM MQ RACF Classes as in previous releases, or
- Use the new mixed case IBM MQ RACF classes.

Without the use of mixed case RACF profiles, you can still use mixed case resource names in IBM MQ for z/OS ; however, these resource names can only be protected by generic RACF profiles in the uppercase IBM MQ classes. When using mixed case IBM MQ RACF profile support you can provide a more granular level of protection by defining IBM MQ RACF profiles in the mixed case IBM MQ classes.

z/OS Resources you can protect in IBM MQ for z/OS

When a queue manager starts, or when instructed by an operator command, IBM MQ for z/OS determines which resources you want to protect.

You can control which security checks are performed for each individual queue manager. For example, you can implement a number of security checks on a production queue manager, but none on a test queue manager.

Connection security

Connection security checking is carried out either when an application program tries to connect to a queue manager. It is done by issuing an MQCONN or MQCONNX request, or when the channel initiator, or CICS or IMS adapter issues a connection request.

If you are using queue manager level security, you can turn connection security checking off for a particular queue manager. However, if you do this any user can connect to that queue manager.

For the CICS adapter, only the CICS address space user ID is used for the connection security check, not the individual CICS terminal user ID. For the IMS adapter, when the IMS control or dependent regions connect to IBM MQ, the IMS address space user ID is checked. For the channel initiator, the user ID used by the channel initiator address space is checked.

You can turn connection security checking on or off at either queue manager or queue sharing group level.

Command security

Command security checking is carried out when a user issues an MQSC command from any of the sources described in Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS. You can make a separate check on the resource specified by the command as described in “Command resource security” on page 273.

If you turn off command checking, issuers of commands are not checked to see whether they have the authority to issue the command.

If MQSC commands are entered from a console, the console must have the z/OS SYS console authority attribute. Commands that are issued from the CSQINP1 or CSQINP2 data sets, or internally by the queue manager, are exempt from all security checking while those for CSQINPX use the user ID of the channel initiator address space. You must control who is allowed to update these data sets through normal data set protection.

You can turn command security checking on or off at either queue manager or queue sharing group level.

Command resource security

Some MQSC commands, for example defining a local queue, involve the manipulation of IBM MQ resources. When command resource security is active, each time a command involving a resource is issued, IBM MQ checks to see if the user is allowed to change the definition of that resource.

You can use command resource security to help enforce naming standards. For example, a payroll administrator might be allowed to delete and define only queues with names beginning "PAYROLL".
If command resource security is inactive, no security checks are made on the resource that is being manipulated by the command. Do not confuse command resource security with command security; the two are independent.

Turning off command resource security checking does not affect the resource checking that is done specifically for other types of processing that do not involve commands.

You can turn command resource security checking on or off at either queue manager or queue sharing group level.

Channel security considerations

Channel security

When you are using channels, the security features available depend on which communications protocol you are going to use. If you use TCP, there are no security features provided with the communications protocol, although you can use TLS. If you are using APPC, you can flow user ID information from the sending MCA through the network to the destination MCA for verification.

For both protocols, you can specify which user IDs you want to check for security purposes, and how many. Again, the choices available to you depend on which protocol you are using, what you specify when you define the channel, and the RESLEVEL settings for the channel initiator.

For more information about the types of channel security available see Channel authentication records and Security exit overview

Related reference

"API-resource security in IBM MQ for z/OS" on page 274
Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on what open options are specified when the queue is opened.

z/OS *API-resource security in IBM MQ for z/OS*

Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on what open options are specified when the queue is opened.

API-resource security is subdivided into the following checks:

- Queue
- Process
- Namelist
- Alternate user
- Context

No security checks are performed when opening the queue manager object or when accessing storage class objects.

Queue

Queue security checking controls who is allowed to open which queue, and what options they are allowed to open it with. For example, a user might be allowed to open a queue called PAYROLL.INCREASE.SALARY to browse the messages on the queue (using the MQOO_BROWSE option), but not to remove messages from the queue (using one of the MQOO_INPUT_* options). If you turn checking for queues off, any user can open any queue with any valid open option (that is, any valid MQOO_* option on an MQOPEN or MQPUT1 call).

You can turn queue security checking on or off at either queue manager or queue sharing group level.

Process

Process security checking is carried out when a user opens a process definition object. If you turn checking for processes off, any user can open any process.

You can turn process security checking on or off at either queue manager or queue sharing group level.

Namelist

Namelist security checking is carried out when a user opens a namelist. If you turn checking for namelists off, any user can open any namelist.

You can turn namelist security checking on or off at either queue manager or queue sharing group level.

Alternate user

Alternate user security controls whether one user ID can use the authority of another user ID to open an IBM MQ object.

For example:

- A server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1.
- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.

- Instead of using its own user ID (PAYSERV) to authorize opening the reply-to queue, the server can specify some other user ID, in this case, USER1. In this example, alternate user security would control whether user ID PAYSERV is allowed to specify user ID USER1 as an alternative user ID when opening the reply-to queue.

The alternative user ID is specified in the *AlternateUserId* field of the object descriptor (MQOD).

You can use alternative user IDs on any IBM MQ object, for example, processes or namelists. It does not affect the user ID used by any other resource managers, for example, for CICS security or for z/OS data set security.

If alternate user security is not active, any user can use any other user ID as an alternative user ID.

You can turn alternate user security checking on or off at either queue manager or queue sharing group level.

Context

Context is information that is applicable to a particular message and is contained in the message descriptor (MQMD) that is part of the message. The context information comes in two sections:

Identity section
The user of the application that first put the message to a queue. It consists of the following fields:

- *UserIdentifier*
- *AccountingToken*
- *ApplIdentityData*

Origin section
The application that put the message on the queue where it is currently stored. It consists of the following fields:

- *PutApplType*
- *PutApplName*
- *PutDate*
- *PutTime*
- *ApplOriginData*

Applications can specify the context data when either an MQPUT or an MQPUT1 call is made. The application might generate the data, the data might be passed on from another message, or the queue manager might generate the data by default. For example, server programs can use context data to check the identity of the requester, that is, did this message come from the correct application? Typically, the *UserIdentifier* field is used to determine the user ID of an alternative user.

You use context security to control whether the user can specify any of the context options on any MQOPEN or MQPUT call. For information about the context options, see the MQOPEN options relating to message context. For descriptions of the message descriptor fields relating to context, see MQMD - Message descriptorMQMD - Message descriptor.

If you turn context security checking off, any user can use any of the context options that the queue security allows.

You can turn context security checking on or off at either queue, queue manager or queue sharing group level.

z/OS Availability on z/OS

IBM MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

Several features of IBM MQ can increase system availability if the queue manager or channel initiator fails. For more information about these features, see the following sections:

- Sysplex considerations

- Shared queues
- Shared channels
- IBM MQ network availability
- Using the z/OS Automatic Restart Manager (ARM)
- Using the z/OS Extended Recovery Facility (XRF)
- Using the z/OS GROUPUR attribute for recovery in a queue sharing group
- Where to find more information about availability

Sysplex considerations

In a *sysplex*, a number of z/OS operating system images collaborate in a single system image and communicate using a coupling facility. IBM MQ can use the facilities of the sysplex environment for enhanced availability.

Removing the affinities between a queue manager and a particular z/OS image allows a queue manager to be restarted on a different z/OS image in the event of an image failure. The restart mechanism can be manual, use ARM, or use system automation, if you ensure the following:

- All page sets, logs, bootstrap data sets, code libraries, and queue manager configuration data sets are defined on shared volumes.
- The subsystem definition has sysplex scope and a unique name within the sysplex.
- The level of *early code* installed on every z/OS image at IPL time is at the same level.
- TCP virtual IP addresses (VIPA) is available on each TCP stack in the sysplex, and you have configured IBM MQ TCP listeners and inbound connections to use VIPAs rather than default host names.

For more information about using TCP in a sysplex, see *TCP/IP in a sysplex*, SG24-5235, an IBM Redbooks® publication.

You can additionally configure multiple queue managers running on different operating system images in a sysplex to operate as a queue sharing group, which can take advantage of shared queues and shared channels for higher availability and workload balancing.

Shared queues

In the queue sharing group environment, an application can connect to any of the queue managers within the queue sharing group. Because all the queue managers in the queue sharing group can access the same set of shared queues, the application does not depend on the availability of a particular queue manager; any queue manager in the queue sharing group can service any queue. This gives greater availability if a queue manager stops because all the other queue managers in the queue sharing group can continue processing the queue. For information about high availability of shared queues, see “Advantages of using shared queues” on page 190.

To further enhance the availability of messages in a queue sharing group, IBM MQ detects if another queue manager in the group disconnects from the coupling facility abnormally, and completes units of work for that queue manager that are still pending, where possible. This is known as *peer recovery* and is described in “Peer recovery” on page 263.

Peer recovery cannot recover units of work that were in doubt at the time of the failure. You can use the Automatic Restart Manager (ARM) to restart all the systems involved in the failure (CICS, Db2, and IBM MQ for example), and to ensure that they are all restarted on the same new processor. This means that they can resynchronize, and gives rapid recovery of in-doubt units of work. This is described in “Using the z/OS Automatic Restart Manager (ARM)” on page 277.

Shared channels

In the queue sharing group environment, IBM MQ provides functions that give high availability to the network. The channel initiator enables you to use networking products that balance network requests across a set of eligible servers and hide server failures from the network (for example, VTAM generic resources). IBM MQ uses a generic port for inbound requests so that attach requests can be routed to any available channel initiator in the queue sharing group. This is described in “Shared channels” on page 210.

Shared outbound channels take the messages they send from a shared transmission queue. Information about the status of a shared channel is held in one place for the whole queue sharing group level. This means that a channel can be restarted automatically on a different channel initiator in the queue sharing group if the channel initiator, queue manager, or communications subsystem fails. This is called *peer channel recovery* and is described in Shared outbound channels.

IBM MQ network availability

IBM MQ messages are carried from queue manager to queue manager in an IBM MQ network using channels. You can change the configuration at a number of levels to improve the network availability of a queue manager, and the ability of an IBM MQ channel to detect a network problem and to reconnect.

TCP *Keepalive* is available for TCP/IP channels. It causes TCP to send packets periodically between sessions to detect network failures. The KAINT channel attribute determines the frequency of these packets for a channel.

AdoptMCA allows a channel, blocked in receive processing as a result of a network outage, to be terminated and replaced by a new connection request. You control AdoptMCA using the ADOPTMCA queue manager property with the MQSC utility or the AdoptNewMCAType property with the Programmable Command Formats interface.

ReceiveTimeout prevents a channel from being permanently blocked in a network receive call. The RCVTIME and RCVTMIN channel initiator parameters, determine the receive timeout characteristics for channels, as a function of their heartbeat interval. See Queue manager parameter for more details.

Using the z/OS Automatic Restart Manager (ARM)

You can use IBM MQ for z/OS in conjunction with the z/OS automatic restart manager (ARM). If a queue manager or a channel initiator has failed, ARM restarts it on the same z/OS image. If z/OS fails, a whole group of related subsystems and applications also fail. ARM can restart all the failed systems automatically, in a predefined order, on another z/OS image within the sysplex. This is called a cross-system restart.

ARM enables rapid recovery of in-doubt transactions in the shared queue environment. It also gives higher availability if you are not using queue sharing groups.

You can use ARM to restart a queue manager on a different z/OS image within the sysplex in the event of z/OS failure.

To enable automatic restart, you must do the following:

1. Set up an ARM coupling data set.
2. Define the automatic restart actions that you want z/OS to perform in an *ARM policy*.
3. Start the ARM policy.

If you want to restart queue managers in different z/OS images automatically, every queue manager in each z/OS image on which that queue manager might be restarted must be defined with a sysplex-wide unique 4-character subsystem name.

Using ARM with IBM MQ is described in Using ARM in an IBM MQ network.

Using the z/OS Extended Recovery Facility (XRF)

You can use IBM MQ in an extended recovery facility (XRF) environment. All IBM MQ-owned data sets (executable code, BSDSs, logs, and page sets) must be on DASD shared between the active and alternative XRF processors.

If you use XRF for recovery, you must stop the queue manager on the active processor and start it on the alternative processor. For CICS, you can do this using the command list table (CLT) provided by CICS, or the system operator can do it manually. For IMS, this is a manual operation and you must do it after the coordinating IMS system has completed the processor switch.

IBM MQ utilities must be completed or terminated before the queue manager can be switched to the alternative processor. Consider the effect of this potential interruption carefully when planning your XRF recovery plans.

Take care to prevent the queue manager starting on the alternative processor before the queue manager on the active processor terminates. A premature start can cause severe integrity problems in data, the catalog, and the log. Using global resource serialization (GRS) helps avoid the integrity problems by preventing simultaneous use of IBM MQ on the two systems. You must include the BSDS as a protected resource, and you must include the active and alternative XRF processors in the GRS ring.

Using the z/OS GROUPUR attribute for recovery in a queue sharing group

Queue sharing groups (QSG) allow additional transactional facilities which are described in this topic. The GROUPUR attribute allows XA client applications to have any in-doubt transaction recovery that may be required, performed on any member of the QSG.

If an XA client application connects to a queue sharing group (QSG) through a Sysplex it cannot guarantee which specific queue manager it connects to. Use of the GROUPUR attribute by queue managers within the queue sharing group can enable any in-doubt transaction recovery that may be necessary to occur on any member of the QSG. Even if the queue manager to which the application was initially connected is not available, transaction recovery can take place.

This feature frees the XA client application from any dependency on specific members of the QSG and thus extends the availability of the queue manager. The queue sharing group appears to the transactional application as a single entity providing all the IBM MQ features and without a single queue manager point of failure.

This functionality is not apparent to the transactional application.

Where to find more information about availability

You can find more information about these topics from the following sources:

Table 24. Where to find more information about availability

Topic	**Where to look**
Queue sharing groups	“Shared queues and queue sharing groups” on page 171
System parameters	Configuring system parameters
Using the Automatic Restart Manager Utility programs	Using ARM in an IBM MQ network
MQSC commands	MQSC commands

z/OS Monitoring and statistics on IBM MQ for z/OS

IBM MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

IBM MQ supplies facilities for monitoring the system and collecting statistics. For further information about these facilities, see the following sections:

- "Online monitoring" on page 279
- "IBM MQ trace" on page 279
- "Events" on page 279

Online monitoring

IBM MQ includes the following commands for monitoring the status of IBM MQ objects:

- DISPLAY CHSTATUS displays the status of a specified channel.
- DISPLAY QSTATUS displays the status of a specified queue.
- DISPLAY CONN displays the status of a specified connection.

For more information about these commands, see The MQSC commands.

IBM MQ trace

IBM MQ supplies a trace facility that you can use to gather the following information while the queue manager is running:

Performance statistics

The statistics trace gathers the following information to help you monitor performance and tune your system:

- Counts of different MQI requests (message manager statistics)
- Counts of different object requests (data manager statistics)
- Information about Db2 usage (Db2 manager statistics)
- Information about Coupling Facility usage (Coupling Facility manager statistics)
- Information about SMDS usage (shared message data set statistics)
- Information about buffer pool usage (buffer manager statistics)
- Information about logging (log manager statistics)
- Information about storage usage (storage manager statistics)
- Information about lock requests (lock manager statistics)

Accounting data

- The accounting trace gathers information about the processor time spent processing MQI calls and about the number of MQPUT and MQGET requests made by a particular user.
- IBM MQ can also gather information about each task using IBM MQ. This data is gathered as a thread-level accounting record. For each thread, IBM MQ also gathers information about each queue used by that thread.

The data generated by the trace is sent to the System Management Facility (SMF) or the generalized trace facility (GTF).

Events

IBM MQ events provide information about errors, warnings, and other significant occurrences in a queue manager. By incorporating these events into your own system management application, you can monitor the activities across many queue managers, for multiple IBM MQ applications. In particular, you can monitor all the queue managers in your system from a single queue manager.

Events can be reported through a user-written reporting mechanism to an administration application that supports the presentation of the events to an operator. Events also enable applications acting as agents for other administration networks, for example NetView®, to monitor reports and create the appropriate alerts.

Related tasks

Using IBM MQ trace

Using IBM MQ events

z/OS Unit of recovery disposition on z/OS

Certain transactional applications can use a GROUP, rather than a QMGR, unit of recovery disposition when connected to a queue manager in a queue sharing group (QSG) by specifying the QSG name when they connect instead of the queue manager name. This allows transaction recovery to be more flexible and robust by removing the requirement to reconnect to the same queue manager in the QSG.

Transactions started by applications that have connected using the queue sharing group name also have a GROUP unit of recovery disposition.

When a transactional application connects with a GROUP unit of recovery disposition it is logically connected to the queue sharing group and does not have an affinity to any specific queue manager. Any 2-phase commit transactions that it has started that have completed phase-1 of the commit process, that is, they are in doubt, can be inquired and resolved, when connected to any queue manager within the QSG. In a recovery scenario this means that the transaction coordinator does not have to reconnect to the same queue manager, which may be unavailable at that time.

Applications that connect with a QMGR unit of recovery disposition have a direct affinity to the queue manager to which they are connected. In a recovery scenario the transaction coordinator must reconnect to the same queue manager to resolve any in-doubt transactions, irrespective of whether the queue manager belongs to a queue sharing group.

When applications specify a queue sharing group name, and thus connect to a queue manager in a QSG with a GROUP unit of recovery disposition, the queue sharing group is logically a separate resource manager. This means that in-doubt transactions are only visible to an application if it reconnects with the same unit of recovery disposition. In-doubt transactions with a QMGR unit of recovery disposition are not visible to applications that have connected with a GROUP unit of recovery disposition and vice versa.

Related concepts

“Enabling GROUP units of recovery” on page 280
A queue sharing group can configure and enable support for GROUP units of recovery.

“Application support” on page 281
Use this page to determine which applications can connect with a GROUP unit of recovery disposition.

z/OS Enabling GROUP units of recovery

A queue sharing group can configure and enable support for GROUP units of recovery.

To use GROUP units of recovery on a queue manager within a QSG, enable the GROUPUR queue manager attribute. For more information about this concept, see “Unit of recovery disposition on z/OS” on page 280 before reading the rest of this topic.

When the GROUPUR queue manager attribute is enabled, the queue manager accepts new connections with a GROUP unit of recovery disposition. If you disable this attribute new connections with this disposition are not accepted, although applications already connected is unaffected until they disconnect.

When an application connects with a GROUP unit of recovery disposition and either inquires what transactions are in doubt or attempts to resolve a transaction that was started elsewhere in the

queue sharing group (QSG), the queue manager to which it is now connected must be able to communicate with the other members of the queue sharing group so that it can process the request. To do this it uses a shared queue called SYSTEM.QSG.UR.RESOLUTION.QUEUE. This queue must be on a recoverable application structure called CSQSYSAPPL. The structure must be recoverable because persistent messages are stored on this queue when processing resolution requests.

Before you can enable GROUP units of recovery, you must ensure that the coupling facility structure and the shared queue are defined. You can use the definitions in the CSQ4INSS sample. When the queue is defined, or detected during startup, each queue manager in the queue sharing group opens the queue so that it can receive incoming requests. If you want to delete or move the queue because it has been defined incorrectly you can request that the queue managers close their open handles on it by updating the queue object to inhibit MQGET requests. When you have made the necessary corrections, permitting applications to get messages from the queue once more directs each queue manager to reopen it. Use the DISPLAY QSTATUS command to identify what handles are open on a queue.

When you have completed this setup you can then enable GROUP units of recovery on each queue manager that you want transactional applications to be able to connect to with a GROUP unit of recovery disposition. This need not be all of the queue managers within the queue sharing group but if you choose to only enable this functionality on a subset of the queue sharing group you must ensure that your applications only attempt to connect to queue managers on which you have enabled it. For more information, see “Application support” on page 281.

When you attempt to enable the GROUPUR queue manager attribute, a number of configuration checks are performed. The queue manager checks that:

- It belongs to a queue sharing group.
- The shared-queue called SYSTEM.QSG.UR.RESOLUTION.QUEUE has been defined, according to the the definition in CSQ4INSS.
- The SYSTEM.QSG.UR.RESOLUTION.QUEUE is on a recoverable CF structure called CSQSYSAPPL.

If any of the above checks fail, the GROUPUR attribute remains disabled and a message code is returned.

These configuration checks are also performed at queue manager startup if the queue manager attribute is enabled. If any of the checks fail during startup GROUP units of recovery is disabled and the queue manager issues a message identifying which check failed. When you have performed the necessary corrective action you must then re-enable the queue manager attribute.

z/OS Application support

Use this page to determine which applications can connect with a GROUP unit of recovery disposition.

Support for the GROUP unit of recovery disposition is limited to certain types of transactional applications for which IBM MQ for z/OS is a resource manager but not the transaction coordinator. Currently supported transactional applications are:

- IBM MQ extended transactional client applications
- IBM MQ classes for JMS applications running in an application server, such as WebSphere Application Server.
- CICS applications running in CICS Transaction Server 4.2 or later, when the CICS MQCONN resource definition is configured with RESYNCMEMBER(GROUPRESYNC).

Related concepts

“ IBM MQ extended transactional client applications” on page 282
Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

“CICS applications” on page 282
Use this page to determine how CICS can use the GROUP unit of recovery disposition.

z/OS *IBM MQ extended transactional client applications*

Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

An example of an IBM MQ extended transactional client application is one that uses JMS and runs in WebSphere Application Server, connecting to IBM MQ over TCP/IP, rather than local bindings. These client applications connect to IBM MQ for z/OS over network connections, such as via TCP/IP. For these applications, it is the value specified for the QMNAME parameter of the xa_info string passed in the xa_open call that specifies whether a QMGR or GROUP unit of recovery disposition is used. For more information about xa_open, see The format of an xa_open string and Additional error processing for xa_open. For JMS applications this is done by specifying the name of the queue sharing group (QSG) in the ConnectionFactory instead of the name of a specific queue manager.

For XA client applications to take advantage of using the GROUP unit of recovery disposition you must configure your TCP/IP setup to allow your client applications to be routed to the queue managers in the queue sharing group that have the GROUPUR attribute enabled, rather than a specific queue manager. One of the dynamic virtual IP address technologies that you can use to do this is the z/OS SysPlex Distributor. See z/OS Communications Server and z/OS Competenze di base: indirizzamento virtuale dinamico for more details. If you want to enable GROUP units of recovery on a subset of the queue managers in your queue sharing group, ensure that your client applications cannot be routed to those on which it is not enabled.

Your client applications do not have to connect to the queue sharing group using shared channels.

z/OS *CICS applications*

Use this page to determine how CICS can use the GROUP unit of recovery disposition.

CICS 4.2 and later provides the group resynchronization option, RESYNCMEMBER(GROUPRESYNC) in an MQCONN resource definition. A CICS configured with this option can connect to any suitable queue manager in a queue sharing group which is running on the same LPAR as that CICS region. To support the CICS GROUPRESYNC option, a queue manager must be running at MQ V7.1 or later, and be enabled for GROUPUR support.

Transactions running within a CICS region connected to MQ using GROUPRESYNC create units of work with GROUP unit of recovery disposition.

You can use RESYNCMEMBER(GROUPRESYNC) to enable faster recovery after a queue manager failure as it enables the CICS region to immediately connect to an alternative eligible queue manager running on the same LPAR, resolving any indoubt transactions as necessary, without waiting for queue manager restart.

RESYNCMEMBER(GROUPRESYNC) also enables more flexible restart options for CICS. A CICS region with its MQ connection configured to use GROUPRESYNC and MQ shared queues can be restarted on any LPAR where there is a queue manager running as a member of the same queue sharing group.

z/OS IBM MQ and other z/OS products

Use this topic to understand how IBM MQ can work with other z/OS products.

Related concepts

“IBM MQ and CICS” on page 283
All the CICS versions supported by IBM MQ 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

“IBM MQ for z/OS and WebSphere Application Server” on page 289
Use this topic to understand the use of IBM MQ for z/OS by the WebSphere Application Server.

Related reference

“IBM MQ and IMS” on page 284
Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

“IBM MQ and the z/OS Batch, TSO, and RRS adapters” on page 287

Use this topic to understand how IBM MQ works with the z/OS Batch, TSO, and RRS adapters.

z/OS IBM MQ and CICS

All the CICS versions supported by IBM MQ 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

For more information about configuring the IBM MQ CICS adapter, and the IBM MQ CICS bridge components, see the Configuring connections to IBM MQ section of the CICS documentation.

Related tasks

Using IBM MQ with CICS

z/OS CICS group attach

CICS group attach provides the ability for a CICS region to connect to any active member of an IBM MQ queue sharing group on the same LPAR rather than specifying an individual queue manager. CICS still connects to a single queue manager at a time.

You require at least two queue managers on the LPAR to support CICS group attach. Using group attach provides higher availability as you do not need a particular queue manager to be active. CICS connects to any queue manager in the queue sharing group on the LPAR.

For more information, see the CICS documentation on the MQCONN resource.

CICS attempts to connect to MQNAME passed as if it were a queue manager:

- If the queue manager exists and is active, the connection will work.
- If the connection fails, CICS queries the status of queue managers in the group to ascertain which are active on same LPAR.
- If multiple queue managers are active, CICS checks for RESYNCMEMBER(YES) and the UOW status to determine whether CICS needs to connect, or should connect, to a particular member, or wait if not active.
- If there is no need to connect to a particular member, CICS selects a queue manager (using a randomizing algorithm).
- CICS attempts to connect to chosen queue manager.
- If the attempt fails then, depending upon the return code, CICS chooses the next member, then goes through the selection loop again.
- If no queue managers are active, CICS issues multiple connections to the list of queue managers and waits on ECBLIST until the first queue manager becomes available.

Related concepts

“Group units of recovery (GROUPUR) for CICS” on page 283
The IBM MQ GROUPUR for CICS provides peer recovery for in-doubt units of work in a queue sharing group (QSG). One IBM MQ queue manager can resolve in-doubt units of work on behalf of another queue manager in the queue sharing group. This means that if CICS reconnects through group attach to a different queue manager in the QSG, it can resolve indoubt transactions from a previous IBM MQ connection.

Related information

Support for IBM MQ queue sharing groups

z/OS Group units of recovery (GROUPUR) for CICS

The IBM MQ GROUPUR for CICS provides peer recovery for in-doubt units of work in a queue sharing group (QSG). One IBM MQ queue manager can resolve in-doubt units of work on behalf of another queue manager in the queue sharing group. This means that if CICS reconnects through group attach to a different queue manager in the QSG, it can resolve indoubt transactions from a previous IBM MQ connection.

If a CICS region is working with a queue manager, and the queue manager ends abnormally, then any indoubt transactions are recovered. This eliminates the need for the CICS region to wait for the queue manager that it was working with to restart, and then resolve any in doubt units of work. This means that you need at least two queue managers on the LPAR, so that CICS can connect to another queue manager in the event of an abnormal termination of the first queue manager.

The new RESYNCMEMBER(GROUPRESYNC) setting on the CICS MQCONN definition:

- Uses the IBM MQ group attach function and peer recovery.
- Requires a queue manager with the GROUPUR attribute enabled.
- Still supports the existing CICS MQCONN RESYNCMEMBER settings (YES and NO):
  - Uses the existing CICS group attach function and no peer recovery.
  - Changing RESYNCMEMBER settings takes effect next time CICS connects to IBM MQ.

Related concepts

"Enabling GROUP units of recovery" on page 280

A queue sharing group can configure and enable support for GROUP units of recovery.

z/OS IBM MQ and IMS

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

The optional additional IBM MQ - IMS bridge enables applications to run an IMS application that does not use the MQI. This means that you can use your legacy applications with IBM MQ, without the need to rewrite them.

For more information about these components, see the following subtopics:

Related concepts

IMS and IMS bridge applications on IBM MQ for z/OS

Related tasks

Setting up the IMS adapter

Setting up the IMS bridge

Operating the IMS adapter

Related reference

MQIIH - IMS information header

z/OS The IMS adapter

The IMS adapter is an interface between IMS application programs and an IBM MQ subsystem.

The IBM MQ adapters enable different application environments to send and receive messages through a message queuing network. The IMS adapter is the interface between IMS application programs and an IBM MQ subsystem. It makes it possible for IMS application programs to use the MQI.

The IMS adapter receives and interprets requests for access to IBM MQ using the External Subsystem Attach Facility (ESAF) provided by IMS. Usually, IMS connects to IBM MQ automatically without operator intervention.

The IMS adapter provides access to IBM MQ resources for programs running in the following modes or states:

- Task (TCB) mode
- Problem state
- Non-cross-memory mode
- Non-access register mode

The adapter provides a connection thread from an application task control block (TCB) to IBM MQ.

The adapter supports a two-phase commit protocol for changes made to resources owned by IBM MQ with IMS acting as the syncpoint coordinator. Conversations where IMS is not the syncpoint coordinator, for example APPC- protected (SYNCLVL=SYNCPT) conversations, are not supported by the IMS adapter.

The adapter also provides a trigger monitor transaction (CSQQTRMN). This is described in "The IMS trigger monitor" on page 285.

You can use IBM MQ with the IMS Extended Recovery Facility (XRF) to aid recovery from a IMS error.

Note: As of IMS 15.2 Extended Recovery Facility (XRF) is no longer supported. See the IMS documentation for more information.

Using the adapter

The application programs and the IMS adapter run in the same address space. The queue manager is separate, in its own address space.

You must link-edit each program that issues one or more MQI calls to a suitable IMS language interface module, and, unless it uses dynamic MQI calls, the IBM MQ-supplied API stub program, CSQQSTUB. When the application issues an MQI call, the stub transfers control to the adapter through the IMS external subsystem interface, which manages the processing of the request by the message queue manager.

System administration and operation with IMS

An authorized IMS terminal operator can issue IMS commands to control and monitor the connection to IBM MQ. However, the IMS terminal operator has no control over the IBM MQ address space. For example, the operator cannot shut down IBM MQ from an IMS address space.

Restrictions

The following IBM MQ API calls are not supported within an application using the IMS adapter:

- MQCB
- MQCB_FUNCTION
- MQCTL

The IMS trigger monitor

The IMS trigger monitor (**CSQQTRMN**) is an IBM MQ-supplied IMS application that starts an IMS transaction when an IBM MQ event occurs, for example, when a message is put onto a specific queue.

How it works

When a message is put onto an application message queue, a trigger is generated if the trigger conditions are met. The queue manager then writes a message (containing some user-defined data), known as a *trigger message*, to the initiation queue that has been specified for that message queue. In an IMS environment, you can start an instance of CSQQTRMN to monitor an initiation queue and to retrieve the trigger messages from it as they arrive. Typically, CSQQTRMN schedules another IMS transaction by an INSERT (ISRT) to the IMS message queue. The started IMS application reads the message from the application message queue and then processes it. CSQQTRMN must run as a non-message BMP.

Each copy of CSQQTRMN services a single initiation queue. When it has started, the trigger monitor runs until IBM MQ or IMS ends.

The APPLCTN macro for CSQQTRMN must specify SCHDTYP=PARALLEL.

Because the trigger monitor is a batch-oriented BMP, IMS transactions that are started by the trigger monitor contain the following:

- Blanks in the LTERM field of the IOPCB
- The PSB name of the trigger monitor BMP in the Userid field of the IOPCB

If the target IMS transaction is protected by Security Server (previously known as RACF), you might need to define CSQQTRMN as a user ID to Security Server.

z/OS The IBM MQ - IMS bridge

The IBM MQ - IMS bridge is the component of IBM MQ for z/OS that allows direct access from IBM MQ applications to applications on your IMS system.

The IBM MQ - IMS bridge enables *implicit MQI support*. This means that you can re-engineer legacy applications that were controlled by 3270-connected terminals to be controlled by IBM MQ messages, without having to rewrite, recompile, or re-link them. The bridge is an IMS *Open Transaction Manager Access* (OTMA) client.

In bridge applications there are no IBM MQ calls within the IMS application. The application gets its input using a GET UNIQUE (GU) to the IOPCB and sends its output using an ISRT to the IOPCB. IBM MQ applications use the IMS header (the MQIIH structure) in the message data to ensure that the applications can execute as they did when driven by nonprogrammable terminals. If you are using an IMS application that processes multi-segment messages, note that all segments should be contained within one IBM MQ message.

The IMS bridge is illustrated in Figure 78 on page 286.

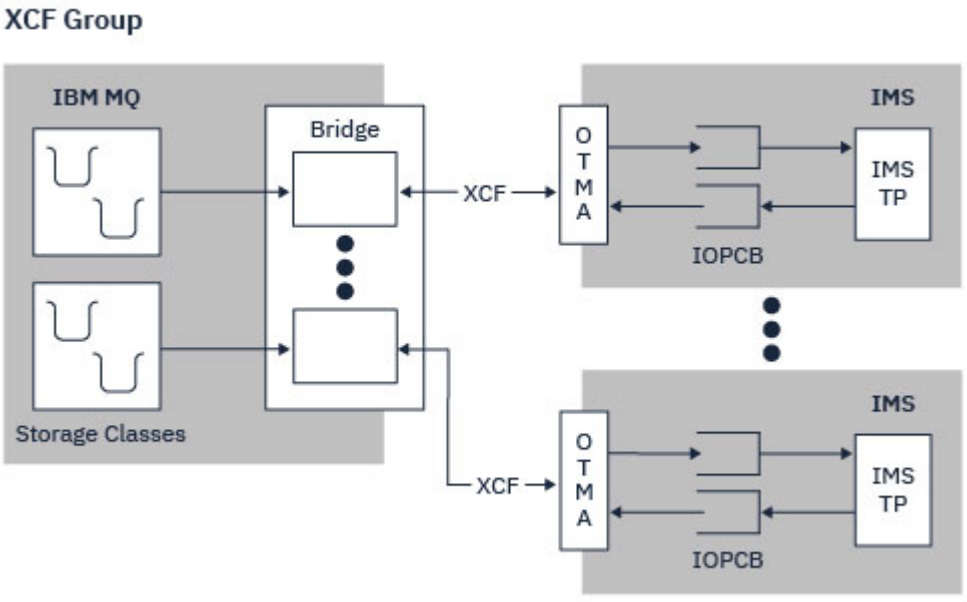


Figure 78. The IBM MQ - IMS bridge

A queue manager can connect to one or more IMS systems, and more than one queue manager can connect to one IMS system. The only restriction is that they must all belong to the same XCF group and must all be in the same sysplex.

See Setting up the IMS bridge for information on setting up an IMS bridge and adding an additional IMS connection to the same queue manager.

What is OTMA?

The IMS OTMA facility is a transaction-based connectionless client/server protocol that runs on IMS. It functions as an interface for host-based communications servers accessing IMS TM applications through the z/OS Cross Systems coupling facility (XCF).

OTMA enables clients to connect to IMS to provide high performance for interactions between clients and IMS for a large network or large number of sessions. OTMA is implemented in a z/OS sysplex environment. Therefore, the domain of OTMA is restricted to the domain of XCF.

OTMA Resource Monitoring

Support for the x'3C' OTMA protocol messages, available in IMS v10 or higher, is included in the IBM MQ - IMS bridge in IBM MQ for z/OS. These messages are sent to OTMA clients by IMS to report its health status.

If an IMS partner is unable to process the volume of transaction requests being sent then it will notify IBM MQ that a flood warning has occurred. In response IBM MQ will slow down the rate at which requests are sent over the bridge.

If IMS is still unable to process the transaction requests and a full flood condition occurs all TPIPEs to the IMS partner are suspended. Upon notification from the IMS partner that the flood or flood-warning condition has been relieved IBM MQ will resume all suspended TPIPEs, if appropriate, and gradually increase the rate at which transaction requests are sent until the maximum rate is achieved. Console messages are issued by IBM MQ in response to a change in the status of IMS partners.

If IMS v10 partners are being used you should ensure that PTF UK45082 has been applied.

Submitting IMS transactions from IBM MQ

To submit an IMS transaction that uses the bridge, applications put messages on an IBM MQ queue as usual. The messages contain IMS transaction data; they can have an IMS header (the MQIIH structure) or allow the IBM MQ - IMS bridge to make assumptions about the data in the message.

IBM MQ then puts the message to an IMS queue (it is queued in IBM MQ first to enable the use of syncpoints to assure data integrity). The storage class of the IBM MQ queue determines whether the queue is an *OTMA queue* (that is, a queue used to transmit messages to the IBM MQ - IMS bridge) and the particular IMS partner to which the message data is sent.

Remote queue managers can also start IMS transactions by writing to these OTMA queues on IBM MQ for z/OS.

Data returned from the IMS system is written directly to the IBM MQ reply-to queue specified in the message descriptor structure (MQMD). (This might be a transmission queue to the queue manager specified in the **ReplyToQMgr** field of the MQMD.)

Related concepts

IMS and IMS bridge applications on IBM MQ for z/OS

Related tasks

Customizing the IMS bridge

Related reference

“IBM MQ and IMS” on page 284

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

z/OS IBM MQ and the z/OS Batch, TSO, and RRS adapters

Use this topic to understand how IBM MQ works with the z/OS Batch, TSO, and RRS adapters.

Introduction to the Batch adapters

The Batch/TSO adapters are the interface between IBM MQ and z/OS application programs running under JES, TSO, or z/OS UNIX System Services. These adaptors enable z/OS application programs to use the MQI.

The adapters provide access to IBM MQ resources for programs running in the following modes or states:

- Task (TCB) mode
- Problem or supervisor state
- Non-cross-memory mode
- Non-access register mode

Connections between application programs and IBM MQ are at the task level. The adapters provide a connection thread from an application task control block (TCB) to IBM MQ.

The Batch/TSO adapter supports a single-phase commit protocol for changes made to resources owned by IBM MQ. It does not support multi-phase commit protocols. The RRS adapter enables IBM MQ applications to participate in two-phase commit protocols with other RRS-enabled products, coordinated by z/OS Resource Recovery Services (RRS).

The adapters use the z/OS STIMERM service to schedule an asynchronous event every second. This event runs an interrupt request block (IRB) that does not involve any waiting by the batch application's task. This IRB checks to see if the IBM MQ termination ECB has been posted. If the termination ECB has been posted, the IRB posts any application ECBs that are waiting on an event in IBM MQ (for example, a signal or a wait).

The Batch/TSO adapter

The IBM MQ Batch/TSO adapter provides IBM MQ support for z/OS Batch and TSO applications. All application programs that run under z/OS Batch or TSO must have the API stub program CSQBSTUB link-edited with them. The stub provides the application with access to all MQI calls. You use single-phase commit and backout for applications by issuing the MQI calls **MQCMIT** and **MQBACK**.

The RRS adapter

Resource Recovery Services (RRS) is a subcomponent of z/OS that provides a system-wide service for coordinating two-phase commit across z/OS products. The IBM MQ Batch/TSO RRS adapter (the RRS adapter) provides IBM MQ support for z/OS Batch and TSO applications that want to use these services. The RRS adapter enables IBM MQ to become a full participant in RRS coordination. Applications can participate in two-phase commit processing with other products that support RRS (for example, Db2).

The RRS adapter provides two stubs; you must link-edit application programs that want to use RRS with one of these stubs.

CSQBRSTB
: This stub allows you to use two-phase commit and backout for applications by using the RRS callable resource recovery services instead of the MQI calls **MQCMIT** and **MQBACK**.

  You must also link-edit module ATRSCSS from library SYS1.CSSLIB with your application. If you use the MQI calls **MQCMIT** and **MQBACK**, you will receive return code MQRC_ENVIRONMENT_ERROR.

CSQBRRSI
: This stub allows you to use MQI calls **MQCMIT** and **MQBACK** ; IBM MQ actually implements these calls as the **SRRCMIT** and **SRRBACK** RRS calls.

For information about building application programs that use the RRS adapter, see The RRS batch adapter.

Where to find more information about the z/OS Batch, TSO, and RRS adapters

You can find more information about the topics in this section in the following sources:

Table 25. Where to find more information about using z/OS Batch with IBM MQ	
Topic	**Where to look**
Setting up the Batch adapters	Task 19: Set up Batch, TSO, and RRS adapters
RRS callable resource recovery services	*MVS Programming: Callable Services for High Level Languages*

z/OS IBM MQ for z/OS and WebSphere Application Server

Use this topic to understand the use of IBM MQ for z/OS by the WebSphere Application Server.

Applications written in Java that are running under WebSphere Application Server can use the Java Message Service (JMS) specification to perform messaging. Point-to-point messaging in this environment can be provided by an IBM MQ for z/OS queue manager.

A benefit of using an IBM MQ for z/OS queue manager to provide the messaging is that connecting JMS applications can participate fully in the functionality of an IBM MQ network. For example, they can use the IMS bridge, or exchange messages with queue managers running on other platforms.

Connection between WebSphere Application Server and a queue manager

See Using IBM MQ and WebSphere Application Server together for more information.

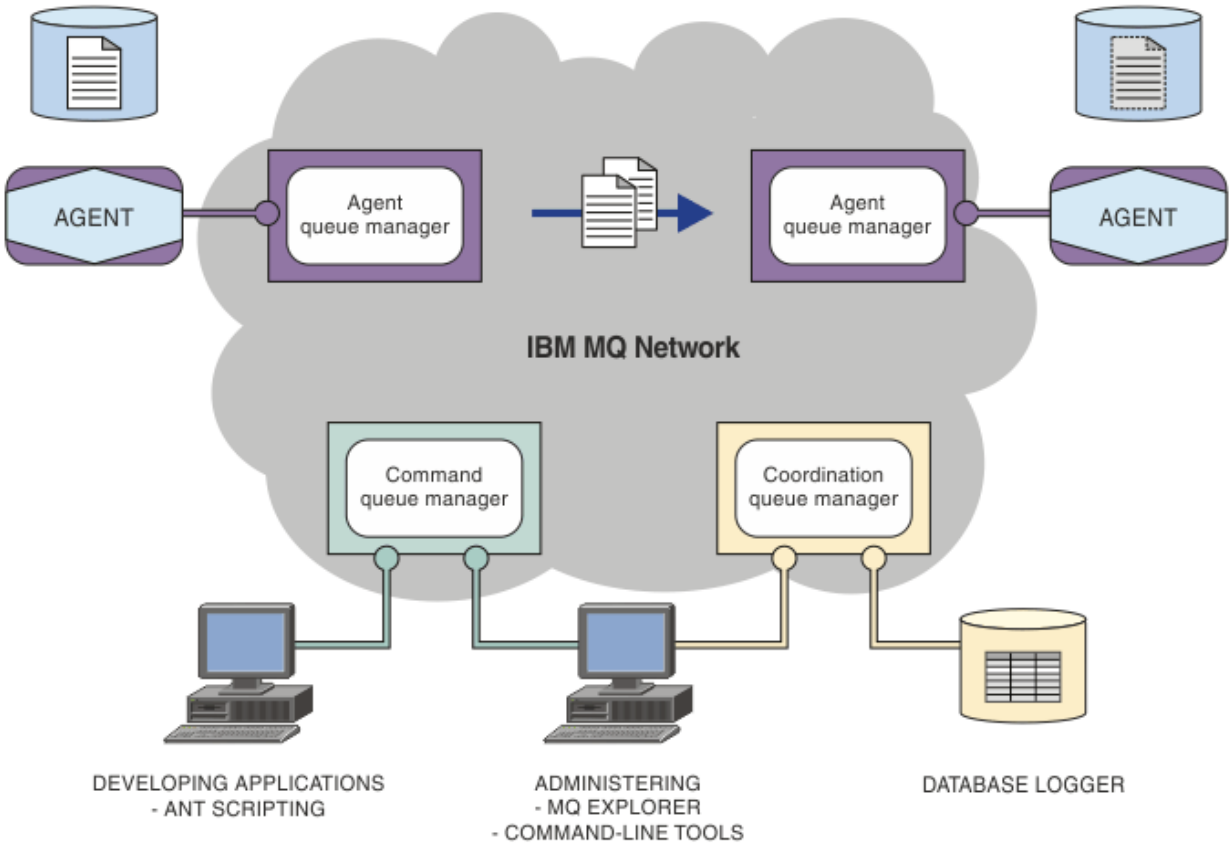
Using IBM MQ functions from JMS applications

By default, JMS messages held on IBM MQ queues use an MQRFH2 header to hold some of the JMS message header information. Many legacy IBM MQ applications cannot process messages with these headers, and require their own characteristic headers, for example the MQCIH for CICS Bridge, or MQWIH for IBM MQ Workflow applications. For more details about these special considerations, see Mapping JMS messages onto IBM MQ messages.

Managed File Transfer

Managed File Transfer trasferisce i file tra i sistemi in modo gestito e controllabile, indipendentemente dalla dimensione del file o dai sistemi operativi utilizzati.

È possibile utilizzare Managed File Transfer per creare una soluzione personalizzata, scalabile e automatizzata che consente di gestire, rendere attendibili e proteggere i trasferimenti file. Managed File Transfer elimina costose ridondanze, riduce i costi di manutenzione e massimizza gli investimenti IT esistenti.



Il diagramma mostra una topologia Managed File Transfer semplice. Ci sono due agenti, ognuno dei quali si connette al proprio gestore code dell'agente in una rete IBM MQ . Un file viene trasferito dall'agente su un lato del diagramma, tramite la rete IBM MQ , all'agente sull'altro lato. Inoltre, nella rete IBM MQ sono presenti il gestore code di coordinamento e un gestore code comandi. Le applicazioni e gli strumenti si collegano a questi gestori code per configurare, amministrare, operare e registrare l'attività Managed File Transfer nella rete IBM MQ .

Managed File Transfer può essere installato come quattro diverse opzioni, a seconda del proprio sistema operativo e della configurazione generale. Queste opzioni sono Managed File Transfer Agent, Managed File Transfer Logger, Managed File Transfer Serviceo Managed File Transfer Tools. Per ulteriori informazioni, consultare Opzioni del prodotto Managed File Transfer.

È possibile utilizzare Managed File Transfer per eseguire queste attività:

- Crea trasferimenti file gestiti
  - Linux Windows Creare nuovi trasferimenti file da IBM MQ Explorer su piattaforme Linux o Windows .
  - Creare nuovi trasferimenti file dalla riga comandi su tutte le piattaforme supportate.
  - Integrare la funzione di trasferimento file nello strumento Apache Ant .
  - Scrivere le applicazioni che controllano Managed File Transfer inserendo i messaggi nelle code comandi dell'agent.
  - Pianificare i trasferimenti di file da eseguire in un secondo momento. È anche possibile attivare trasferimenti di file pianificati in base a un intervallo di eventi del file system, ad esempio un nuovo file creato.
  - Monitorare continuamente una risorsa, ad esempio una directory, e quando il suo contenuto soddisfa alcune condizioni predefinite, avviare un'attività. Questa attività può essere un trasferimento file, uno script Ant o un lavoro JCL.

  - Trasferire i file da e verso le code IBM MQ .
  - Trasferire i file da e verso i server FTP, FTPS o SFTP.
  - Trasferire i file da e verso i nodi Connect:Direct .
  - Trasferire sia il testo che i file binari. I file di testo vengono convertiti automaticamente tra le codepage e le convenzioni di fine riga dei sistemi di origine e di destinazione.
  - I trasferimenti possono essere protetti, utilizzando gli standard industriali per le connessioni basate su SSL (Secure Socket Layer).
- Visualizzare i trasferimenti in corso e registrare le informazioni su tutti i trasferimenti nella rete
  - 

Visualizzare lo stato dei trasferimenti in corso da IBM MQ Explorer su piattaforme Linux o Windows .
  - 

Verificare lo stato dei trasferimenti completati utilizzando IBM MQ Explorer su piattaforme Linux o Windows .
  - Utilizzare la funzione del programma di registrazione database Managed File Transfer per salvare i messaggi di log in un database Db2 o Oracle .

Managed File Transfer è costruito su IBM MQ, che fornisce una consegna certa e unica dei messaggi tra le applicazioni. È possibile usufruire di varie funzionalità di IBM MQ. Ad esempio, è possibile utilizzare la compressione del canale per comprimere i dati che vengono inviati tra gli agent sui canali IBM MQ e utilizzare i canali SSL per proteggere i dati che vengono inviati tra gli agent. I file vengono trasferiti in modo affidabile e possono tollerare il malfunzionamento dell'infrastruttura su cui viene effettuato il trasferimento file. Se si verifica un'interruzione di rete, il trasferimento file viene riavviato dal punto in cui è stato lasciato quando è stata ripristinata la connettività.

Consolidando il trasferimento file con la tua rete IBM MQ esistente, puoi evitare di spendere le risorse necessarie per mantenere due infrastrutture separate. Se non sei già un cliente IBM MQ , creando una IBM MQ rete di supporto Managed File Transfer stai creando il backbone per una futura implementazione SOA. Se sei già un cliente IBM MQ , Managed File Transfer può trarre vantaggio dalla tua infrastruttura IBM MQ esistente, inclusi IBM MQ Internet Pass-Thru e IBM Integration Bus.

Puoi sfruttare le soluzioni di alta disponibilità IBM MQ per migliorare la resilienza della tua configurazione Managed File Transfer . Se gli agent utilizzano gestori code di dati replicati (RDQM), è necessario configurarli per utilizzare la funzione di indirizzo IP mobile. Ciò significa che gli agenti utilizzano lo stesso indirizzo IP per comunicare con una delle tre istanze RDQM attualmente in esecuzione e si riconnettono automaticamente in caso di failover (consultare Alta disponibilità RDQM e Creazione ed eliminazione di un indirizzo IP mobile). Se si utilizza la soluzione del gestore code a più istanze, le applicazioni utilizzano un indirizzo IP differente per comunicare con ciascuna istanza, gestita dalla riconnessione client in caso di failover (consultare Gestori code a più istanze e Riconnessione canale e client).

Managed File Transfer si integra con numerosi altri prodotti IBM :

IBM Integration Bus

Elaborare i file che sono stati trasferiti da Managed File Transfer come parte di un flusso IBM Integration Bus . Per ulteriori informazioni, consultare Utilizzo di MFT da IBM Integration Bus.

IBM Sterling Connect:Direct

Trasferire i file a e da una rete Connect:Direct esistente utilizzando il bridge Managed File Transfer Connect:Direct . Per ulteriori informazioni, consultare Il bridge Connect:Direct.

IBM Tivoli Composite Application Manager

IBM Tivoli Composite Application Manager fornisce un agent che è possibile utilizzare per monitorare le informazioni pubblicate sul gestore code di coordinamento.

Concetti correlati

Opzioni del prodotto Managed File Transfer

“Panoramica della topologia MFT” a pagina 292

Una panoramica su come gli agenti Managed File Transfer sono connessi al gestore code di coordinamento in una rete IBM MQ .

"Come funziona MFT con IBM MQ?" a pagina 292
Managed File Transfer interagisce in diversi modi con IBM MQ.

Come funziona MFT con IBM MQ?

Managed File Transfer interagisce in diversi modi con IBM MQ.

- Managed File Transfer trasferisce i file tra i processi dell'agent dividendo ciascun file in uno o più messaggi e trasmettendo i messaggi attraverso la rete IBM MQ .
- L'agent elabora lo spostamento dei dati del file utilizzando messaggi non persistenti per ridurre al minimo l'impatto sui log IBM MQ . Comunicando tra loro, i processi dell'agent regolano il flusso di messaggi contenenti i dati del file. Ciò impedisce la creazione di messaggi contenenti dati di file sulle code di trasmissione IBM MQ e garantisce che se uno qualsiasi dei messaggi non persistenti non viene consegnato, i dati di file vengono inviati di nuovo.
- Gli agent Managed File Transfer utilizzano un certo numero di code IBM MQ . Per ulteriori informazioni, consultare MFT code di sistema e l'argomento di sistema.
- Sebbene alcune di queste code siano strettamente per uso interno, un agent può accettare le richieste sotto forma di messaggi di comando specialmente formattati inviati a una coda specifica da cui l'agent legge. Sia i comandi della riga comandi che il plugin IBM MQ Explorer inviano IBM MQ messaggi all'agente per indicare all'agente di eseguire l'azione desiderata. È possibile scrivere le applicazioni IBM MQ che interagiscono con l'agent in questo modo. Per ulteriori informazioni, consultare Controllo MFT inserendo i messaggi nella coda comandi dell'agent.
- Gli agent Managed File Transfer inviano informazioni relative al loro stato, all'avanzamento e al risultato dei trasferimenti a un gestore code MQ designato come gestore code di coordinamento. Queste informazioni vengono pubblicate dal gestore code di coordinamento e possono essere sottoscritte dalle applicazioni che desiderano monitorare l'avanzamento del trasferimento o conservare i record dei trasferimenti che si sono verificati. Sia i comandi della riga comandi che il plug-in IBM MQ Explorer possono utilizzare le informazioni pubblicate. È possibile scrivere applicazioni IBM MQ che utilizzano queste informazioni. Per ulteriori informazioni sull'argomento in cui vengono pubblicate le informazioni, consultare SYSTEM.FTE ArgomentoFTE.
- I componenti chiave di Managed File Transfer sfruttano la funzione dei gestori code IBM MQ per memorizzare e inoltrare i messaggi. Ciò significa che se si verifica un'interruzione, le parti non interessate della propria infrastruttura possono continuare a trasferire i file. Ciò si estende al gestore code di coordinamento, dove una combinazione di sottoscrizioni di archiviazione e inoltro e durevoli consente al gestore code di coordinazione di tollerare che diventi non disponibile senza perdere le informazioni chiave sui trasferimenti file che hanno avuto luogo.

Panoramica della topologia MFT

Una panoramica su come gli agenti Managed File Transfer sono connessi al gestore code di coordinamento in una rete IBM MQ .

Gli agent Managed File Transfer inviano e ricevono i file trasferiti. Ogni agent ha la sua serie di code sul suo gestore code associato e l'agent è collegato al suo gestore code in modalità bind o client. Un agent può anche utilizzare il gestore code di coordinamento come proprio gestore code.

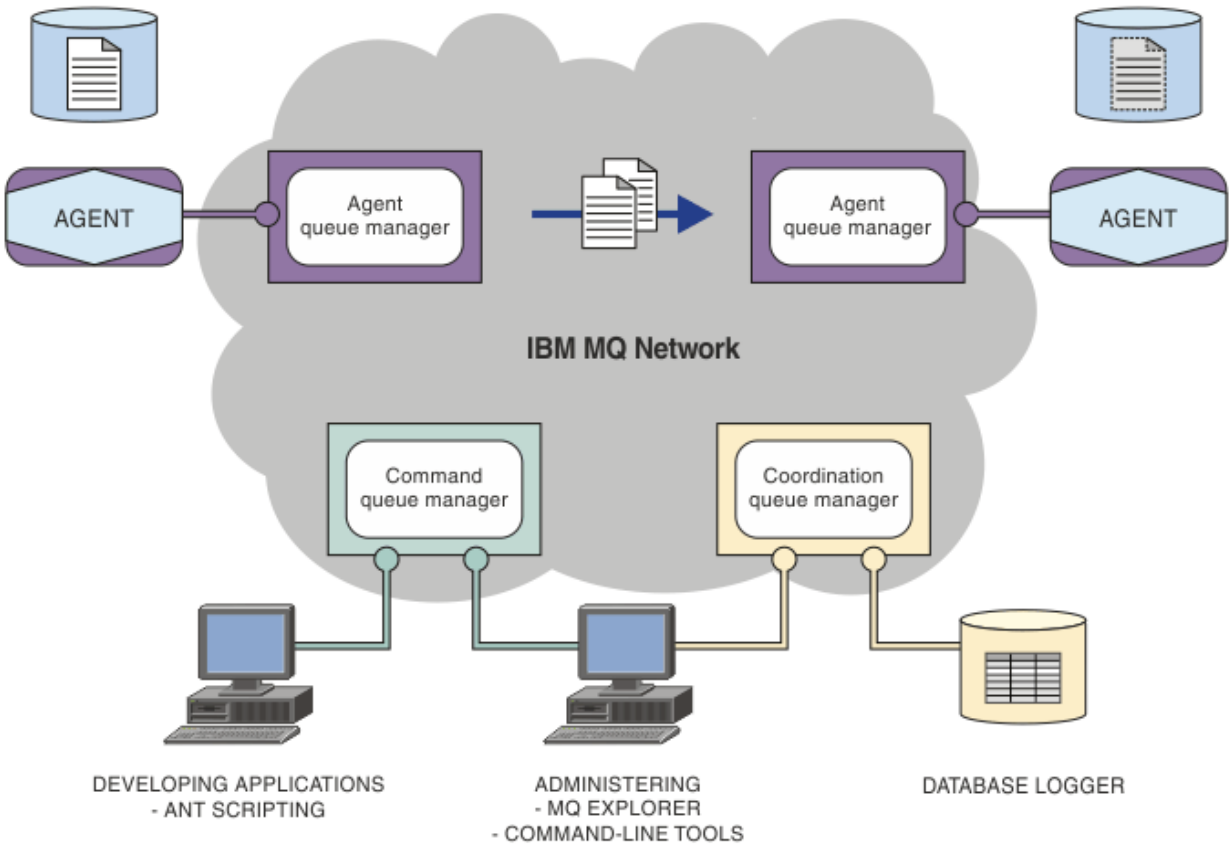
Il gestore code di coordinamento trasmette le informazioni di controllo e di trasferimento file. Il gestore code di coordinamento rappresenta un singolo punto per la raccolta di informazioni sull'agent, sullo stato del trasferimento e sul controllo del trasferimento. Il gestore code di coordinamento non deve essere disponibile per eseguire i trasferimenti. Se il gestore code di coordinamento diventa temporaneamente non disponibile, i trasferimenti continuano normalmente. I messaggi di controllo e stato vengono memorizzati nei gestori code dell'agent fino a quando il gestore code di coordinamento non diventa disponibile e possono essere elaborati normalmente.

Gli agent si registrano con il gestore code di coordinamento e pubblicano i loro dettagli su tale gestore code. Queste informazioni sull'agente vengono usate dal plugin Managed File Transfer per abilitare

l'inizio dei trasferimenti da IBM MQ Explorer. Le informazioni sull'agent raccolte sul gestore code di coordinamento vengono utilizzate dai comandi per visualizzare le informazioni sull'agent e lo stato dell'agent.

Lo stato del trasferimento e le informazioni di controllo del trasferimento vengono pubblicate sul gestore code di coordinamento. Lo stato del trasferimento e le informazioni di controllo del trasferimento vengono utilizzati dal plugin Managed File Transfer per monitorare l'avanzamento dei trasferimenti da IBM MQ Explorer. Le informazioni di controllo trasferimento memorizzate sul gestore code di coordinamento possono essere conservate per fornire la possibilità di controllo.

Il gestore code comandi viene utilizzato per connettersi alla rete IBM MQ ed è il gestore code a cui si è connessi quando si immettono i comandi Managed File Transfer .



Concetti correlati

“Managed File Transfer” a pagina 289
Managed File Transfer trasferisce i file tra i sistemi in modo gestito e controllabile, indipendentemente dalla dimensione del file o dai sistemi operativi utilizzati.

“Come funziona MFT con IBM MQ?” a pagina 292
Managed File Transfer interagisce in diversi modi con IBM MQ.

Managed File Transfer scenario

Panoramica su MFTREST API

REST API supporta alcuni comandi Managed File Transfer , incluso l'elencazione dei trasferimenti e i dettagli sugli agent di trasferimento file.

REST API include opzioni per elencare tutti i trasferimenti Managed File Transfer correnti e per interrogare lo stato degli agent Managed File Transfer . Per ulteriori informazioni, vedi Introduzione a REST API MFT.

IBM MQ Internet Pass-Thru

IBM MQ Internet Pass-Thru (MQIPT) è un componente facoltativo di IBM MQ che può essere utilizzato per implementare soluzioni di messaggistica tra siti remoti su Internet.

Per ottenere i file di installazione MQIPT per IBM MQ 9.4.x, andare all'indirizzo IBM Fix Central per IBM MQ.

È possibile utilizzare MQIPT per collegare qualsiasi versione supportata di IBM MQ. Non è necessario installare altri componenti IBM MQ alla stessa versione di MQIPT.

Se è stata acquistata la titolarità IBM MQ , è possibile installare tutte le copie richieste da MQIPT. Le installazioni MQIPT non vengono conteggiate rispetto alla titolarità IBM MQ acquistata. Per ulteriori informazioni sulla licenza IBM MQ , consultare Informazioni sulla licenzaIBM MQ.

Nota: Questa documentazione è relativa a MQIPT in IBM MQ 9.4. Per la documentazione relativa al pacchetto di supporto MQIPT (versione 2.1) in IBM Documentation, consultare MQIPT (SupportPac MS81) nella documentazione IBM MQ 9.0 .

Nota: Se si utilizza MQIPT 2.1 o precedente, si consiglia di eseguire l'aggiornamento a MQIPT per IBM MQ 9.4, poiché la data di fine del supporto per il pacchetto di supporto MQIPT era il 30th settembre 2020.

IBM MQ Internet Pass-Thru viene eseguito come un servizio autonomo che può ricevere e inoltrare flussi di messaggi IBM MQ , tra due gestori code IBM MQ o tra un client IBM MQ e un gestore code IBM MQ .

MQIPT abilita questa connessione quando client e server non si trovano sulla stessa rete fisica.

Una o più istanze di MQIPT possono essere collocate nel percorso di comunicazione tra due gestori code IBM MQ o tra un client IBM MQ e un gestore code IBM MQ . Le istanze di MQIPT consentono ai due sistemi IBM MQ di scambiare messaggi senza la necessità di una connessione TCP/IP diretta tra i due sistemi. Questa architettura è utile se la configurazione del firewall non consente una connessione TCP/IP diretta tra i due sistemi.

MQIPT è in ascolto su una o più porte TCP/IP per le connessioni in entrata. Queste connessioni possono trasportare messaggi IBM MQ normali, messaggi IBM MQ con tunneling all'interno di HTTP o messaggi codificati utilizzando TLS (Transport Layer Security) o SSL (Secure Sockets Layer). MQIPT può gestire più connessioni simultanee.

Si fa riferimento al canale IBM MQ che effettua la richiesta di connessione TCP/IP iniziale come *chiamante*, al canale a cui sta tentando di connettersi come *risponditore* e al gestore code che sta tentando di contattare come *gestore code di destinazione*.

MQIPT conserva i dati in memoria mentre li inoltra dall'origine alla destinazione. Nessun dato viene salvato su disco (ad eccezione della memoria che il sistema operativo utilizza come pagine su disco). L'unica volta in cui MQIPT accede esplicitamente al disco è per leggere il file di configurazione e scrivere i record di traccia e di log di connessione.

La gamma completa di tipi di canale IBM MQ può connettersi tramite una o più istanze di MQIPT. La presenza di MQIPT in un percorso di comunicazione non ha alcun effetto sulle caratteristiche funzionali dei componenti IBM MQ connessi. Tuttavia, le prestazioni del trasferimento del messaggio potrebbero essere influenzate.

MQIPT può essere utilizzato con IBM MQ come descritto in “Possibili configurazioni di MQIPT” a pagina 298.

Per installare MQIPT, consultare Installazione MQIPT.

Attività correlate

Configurazione di IBM MQ Internet Pass-Thru

Amministrazione e configurazione di IBM MQ Internet Pass-Thru

Riferimenti correlati

IBM MQ Internet Pass-Thru Riferimento di configurazione

Utilizzi di MQIPT

Esistono diversi utilizzi potenziali per IBM MQ Internet Pass-Thru (MQIPT).

MQIPT può essere utilizzato come concentratore di canali

Utilizzando MQIPT in questo modo, i canali verso o da più host separati possono apparire a un firewall come se fossero tutti verso o dall'host MQIPT . Ciò semplifica la definizione e la gestione delle regole di filtro del firewall.

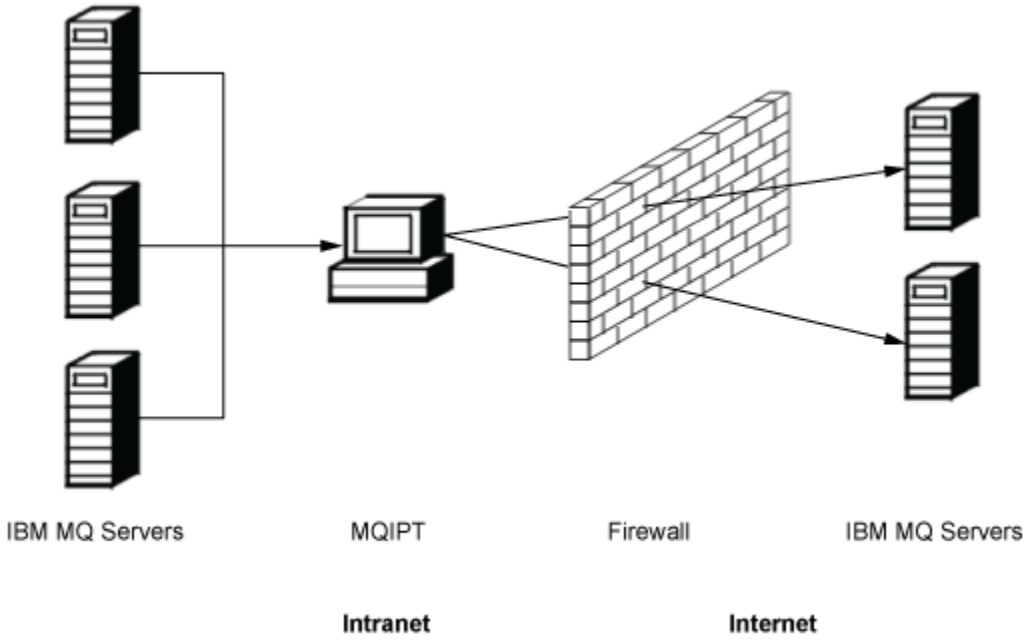


Figura 79. Esempio di MQIPT come concentratore di canale

MQIPT può essere inserito in una DMZ per fornire un singolo punto di accesso

Se MQIPT si trova all'interno di un firewall DMZ (una configurazione del firewall per la protezione delle reti locali), su un computer con un indirizzo IP (Internet Protocol) noto e attendibile, MQIPT può essere utilizzato per ascoltare le connessioni del canale IBM MQ in entrata che può quindi inoltrare all'intranet attendibile; il firewall interno deve consentire a questo computer attendibile di effettuare connessioni in entrata. In questa configurazione, MQIPT impedisce alle richieste esterne di accesso di ricevere i veri indirizzi IP dei computer nell'intranet attendibile. In questo modo, MQIPT fornisce un singolo punto di accesso. Se richiesto, MQIPT può essere configurato per accettare connessioni TLS e inoltrare i dati alla destinazione utilizzando una connessione TLS separata, terminando quindi la sessione TLS nella DMZ.

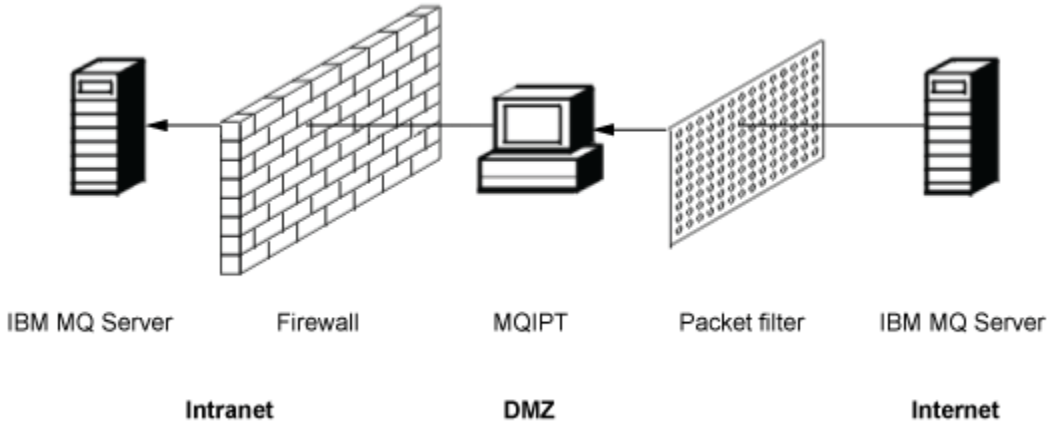


Figura 80. Esempio di MQIPT in un firewall DMZ

MQIPT può comunicare tramite tunneling HTTP

Se due istanze di MQIPT sono distribuite in linea, possono comunicare utilizzando HTTP. La funzione di tunneling HTTP abilita la trasmissione delle richieste attraverso i firewall, utilizzando i proxy HTTP esistenti. Il primo MQIPT inserisce il Protocollo IBM MQ in HTTP e il secondo estrae il protocollo IBM MQ dal relativo wrapper HTTP e lo inoltra al Gestore code di destinazione.

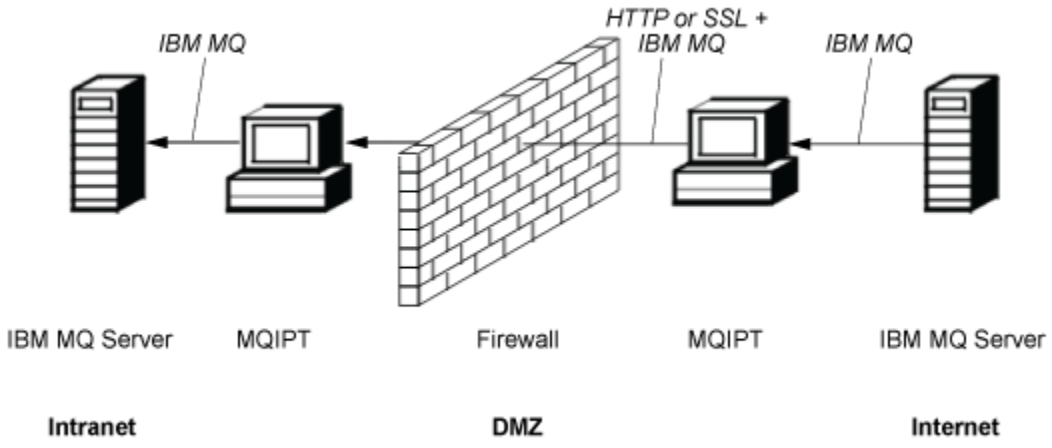


Figura 81. Esempio di tunneling MQIPT e HTTP

MQIPT può codificare i messaggi

Se MQIPT è configurato come nell'esempio precedente, le richieste possono essere codificate prima della trasmissione attraverso i firewall. Il primo MQIPT codifica i dati e il secondo li decodifica utilizzando SSL/TLS prima di inviarli al gestore code di destinazione.

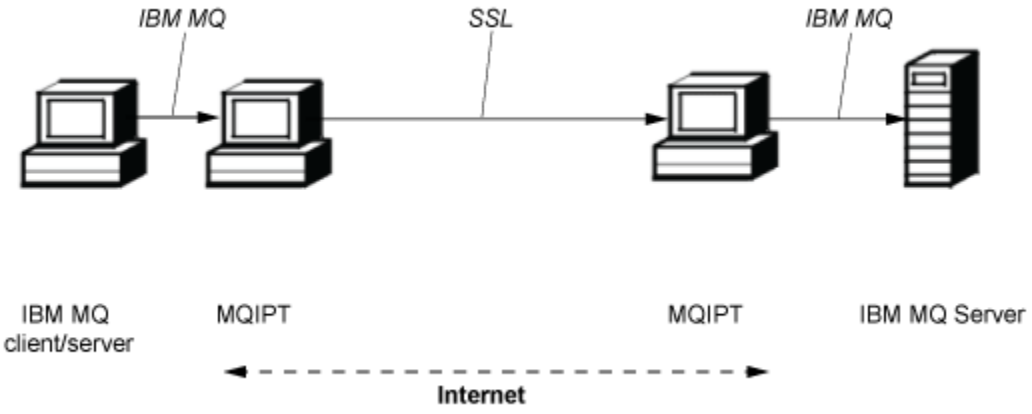


Figura 82. Esempio di MQIPT e SSL/TLS

Come funziona MQIPT

Nella sua configurazione più semplice, MQIPT agisce come un programma di inoltro del protocollo IBM MQ . È in ascolto su una porta TCP/IP e accetta richieste di connessione da canali IBM MQ .

Se viene ricevuta una richiesta con formato corretto, MQIPT stabilisce un'altra connessione TCP/IP tra se stessa e il gestore code IBM MQ di destinazione. Quindi, passa tutti i pacchetti di protocollo che riceve dalla connessione in entrata al gestore code di destinazione e restituisce i pacchetti di protocollo dal gestore code di destinazione alla connessione in entrata originale.

Non è implicata alcuna modifica al protocollo IBM MQ (client/server o gestore code su gestore code) poiché nessuna delle due estremità è direttamente consapevole della presenza dell'intermediario. Le nuove versioni del codice client o server IBM MQ non sono richieste.

Per utilizzare MQIPT, il canale chiamante deve essere configurato in modo da utilizzare il nome host e la porta MQIPT , non il nome host e la porta del gestore code di destinazione. Questo è definito con la proprietà **CONNAME** del canale IBM MQ . MQIPT legge i dati in entrata e li trasmette semplicemente al gestore code di destinazione. Altri campi di configurazione, come l'ID utente e la password in un canale client/server, vengono passati in modo simile al gestore code di destinazione.

Più gestori code

MQIPT può essere utilizzato per consentire l'accesso a più di un gestore code di destinazione. Perché ciò funzioni, deve esistere un meccanismo per indicare a MQIPT a quale gestore code connettersi, quindi MQIPT utilizza il numero di porta TCP/IP in entrata per stabilire a quale gestore code connettersi.

È quindi possibile configurare MQIPT in modo che sia in ascolto su più porte TCP/IP. Ogni porta in ascolto viene associata a un gestore code di destinazione tramite un MQIPT *instradamento*. È possibile definire fino a 100 instradamenti di questo tipo, che associano una porta TCP/IP in ascolto al nome host e alla porta del gestore code di destinazione. Ciò significa che il nome host (indirizzo IP) del gestore code di destinazione non è mai visibile al canale di origine. Ogni instradamento può gestire più connessioni tra la sua porta di ascolto e la destinazione, ogni connessione agisce in modo indipendente.

MQIPT file di configurazione

MQIPT utilizza un file di configurazione denominato `mqipt.conf`. Questo file contiene le definizioni di tutti gli instradamenti e le relative proprietà associate. Consultare Amministrazione e configurazione di IBM MQ Internet Pass-Thru per ulteriori informazioni su `mqipt.conf`.

Quando MQIPT viene avviato, avvia ogni instradamento elencato nel file di configurazione. I messaggi vengono scritti sulla console di sistema mostrando lo stato di ciascun instradamento. Quando viene

visualizzato il messaggio MQCPI078 per un instradamento, tale instradamento è pronto ad accettare richieste di connessione.

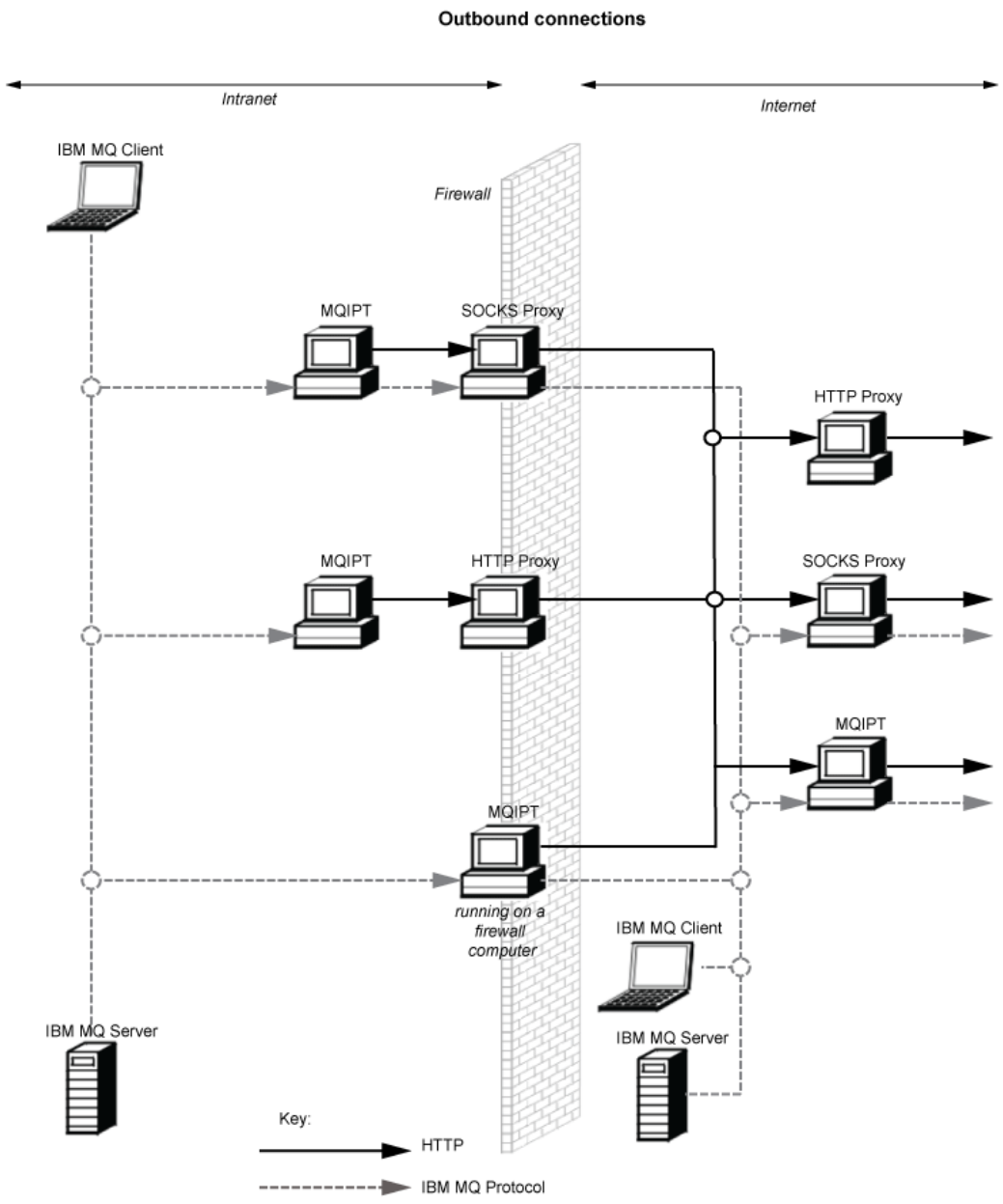
Possibili configurazioni di MQIPT

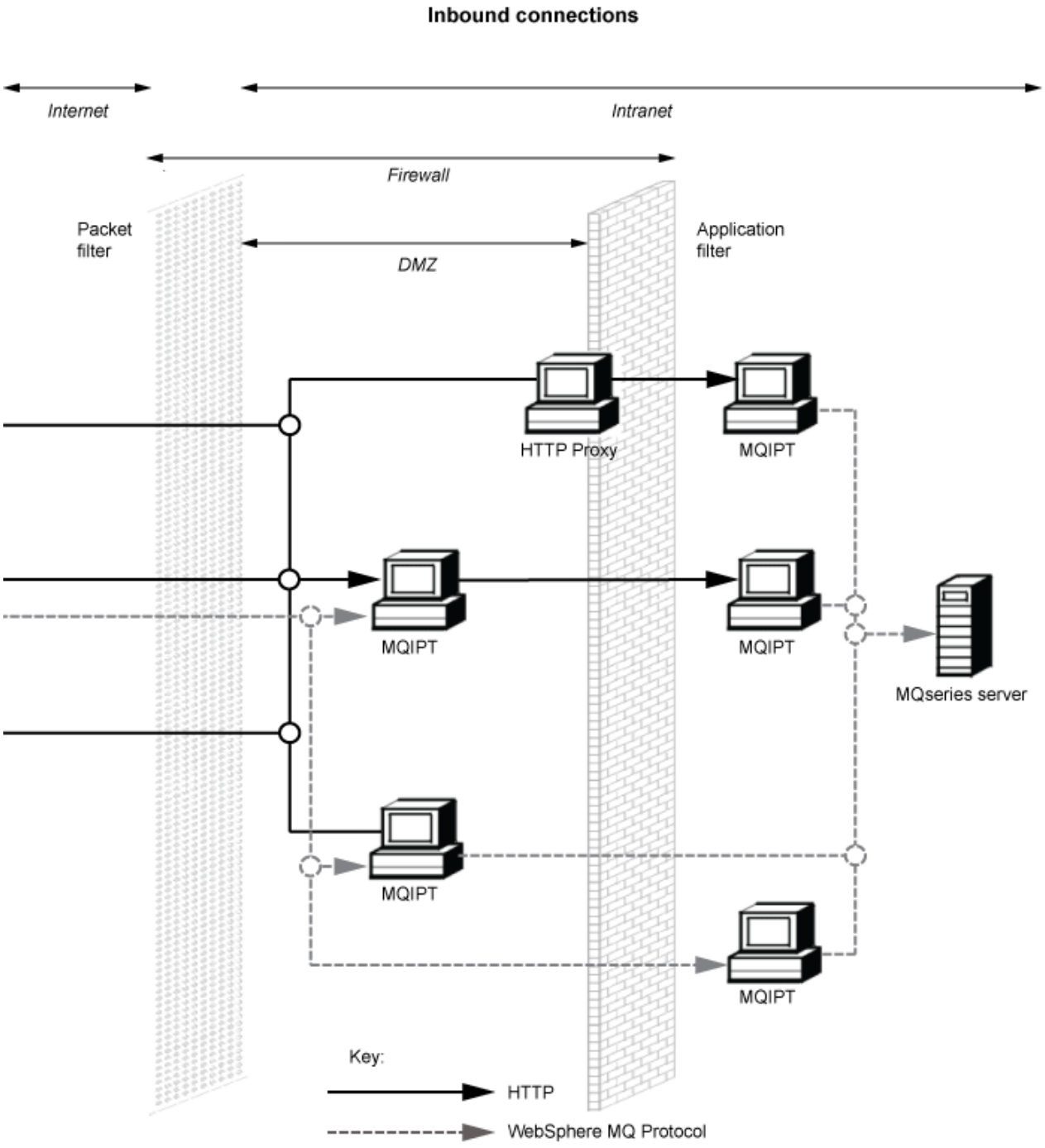
MQIPT può essere utilizzato insieme a IBM MQ e IBM Integration Bus.

La seguente figura a più parti mostra molte delle possibili configurazioni per MQIPT in una topologia IBM MQ . Illustra diversi modi in cui MQIPT può inviare messaggi. Mostra i client e i server su una intranet, all'interno di un firewall e su Internet all'esterno del firewall, passando i messaggi a MQIPT, proxy HTTP o proxy SOCKS, che li inoltra.

I messaggi vengono ricevuti da un proxy MQIPT o da un proxy HTTP in una DMZ prima di passare il messaggio attraverso il firewall in entrata a un server.

Si noti che i computer proxy HTTP, proxy SOCKS e MQIPT sul lato intranet del firewall rappresentano la possibilità di più computer concatenati su Internet. Ad esempio, un computer MQIPT potrebbe comunicare tramite uno o più computer proxy SOCKS o HTTP o ulteriori computer MQIPT , prima di raggiungere la destinazione.


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



Configurazioni compatibili

Scenari di connessione compatibili in cui un client IBM MQ o un gestore code comunica con MQIPT. Lo stesso instradamento o un secondo instradamento MQIPT viene utilizzato per comunicare con un gestore code di destinazione.

Configurazioni compatibili con un singolo instradamento MQIPT

Puoi utilizzare una singola rotta MQIPT per comunicare con IBM MQ.

Le colonne in Tabella 26 a pagina 301 contengono le seguenti informazioni:

1. Il protocollo utilizzato tra IBM MQ e l'instradamento MQIPT . La connessione può essere creata da un client IBM MQ o da un gestore code e può utilizzare FAP (IBM MQ Formats and Protocols) o un protocollo SSL/TLS.
2. La modalità in cui opera l'instradamento MQIPT . Il formato della comunicazione su Internet tra MQIPT e IBM MQ, è determinato dalla configurazione dell'instradamento MQIPT . Nota che dove la tabella cita SSL, puoi anche utilizzare TLS.
3. Il protocollo utilizzato tra l'instradamento MQIPT e il gestore code di destinazione.

Tabella 26. Configurazioni valide con una singola istanza di MQIPT

1. IBM MQ Protocollo di origine	**2. Modalità dell' MQIPT instradamento**	**3. IBM MQ Protocollo di destinazione**
FAP	FAP - proxy (predefinito)	FAP
	Server FAP e client SSL	SSL/TLS
SSL/TLS	Proxy SSL	SSL/TLS
	Server SSL e client FAP	FAP
	Server SSL e client SSL	SSL/TLS

Configurazioni compatibili con più di un instradamento MQIPT

Puoi scegliere di utilizzare più di una rotta, su una o più istanze di MQIPT, per comunicare con IBM MQ.

Le colonne in Tabella 27 a pagina 302 contengono le seguenti informazioni:

1. Il protocollo utilizzato tra IBM MQ e il primo instradamento MQIPT . La connessione può essere creata da un client IBM MQ o da un gestore code e può utilizzare FAP (IBM MQ Formats and Protocols) o un protocollo SSL/TLS.
2. La modalità in cui opera il primo instradamento MQIPT . Il formato della comunicazione su Internet tra MQIPT e IBM MQ, è determinato dalla configurazione dell'instradamento MQIPT . Nota che dove la tabella cita SSL, puoi anche utilizzare TLS.
3. La modalità in cui opera il secondo instradamento MQIPT .
4. Il protocollo utilizzato tra il secondo instradamento MQIPT e il gestore code di destinazione.

Tabella 27. Configurazioni valide con più istanze di MQIPT			
1. IBM MQ Protocollo di origine	**2. Modalità del primo MQIPT instradamento**	**3. Modalità del secondo MQIPT instradamento**	**4. IBM MQ Protocollo di destinazione**
FAP (predefinito)	FAP - proxy (predefinito)	FAP - proxy (predefinito)	FAP
	Server FAP e client SSL	Proxy SSL	SSL/TLS
		Server SSL e client FAP	FAP
		Server SSL e client SSL	SSL/TLS
	Client HTTP	Server HTTP e client SSL	SSL/TLS
	Client HTTPS	HTTPS - server e SSL - client	SSL/TLS
	Client HTTP	server-http	FAP
	Client HTTPS	Server HTTPS	FAP
SSL/TLS	Proxy SSL	Proxy SSL	SSL/TLS
		Server SSL e client FAP	FAP
		Server SSL e client SSL	SSL/TLS
	Client HTTP	server-http	FAP
	Client HTTPS	Server HTTPS	SSL/TLS
	Client HTTP	Server HTTP e client SSL	FAP
	Client HTTPS	HTTPS - server e SSL - client	SSL/TLS

Configurazioni canale supportate

Tutti i tipi di canale IBM MQ sono supportati, ma la configurazione è limitata alle connessioni TCP/IP. Per un client o gestore code IBM MQ , MQIPT viene visualizzato come se fosse il gestore code di destinazione. Laddove la configurazione del canale richiede un host di destinazione e un numero di porta, vengono specificati il numero di porta del listener e il nome host MQIPT .

Canali client/server
MQIPT ascolta le richieste di connessione client in entrata e le inoltra utilizzando il tunneling HTTP, SSL/TLS o come pacchetti di protocollo IBM MQ standard. Se MQIPT utilizza il tunneling HTTP o SSL/TLS, li inoltra su una connessione a un secondo MQIPT. Se non utilizza il tunneling HTTP, li inoltra su una connessione a quello che vede come gestore code di destinazione (anche se questo potrebbe a sua volta essere un ulteriore MQIPT). Quando il gestore code di destinazione ha accettato la connessione client, i pacchetti vengono inoltrati tra client e server.

Canali mittente / ricevente cluster
Se MQIPT riceve una richiesta in entrata da un canale mittente del cluster, presuppone che il gestore code sia stato abilitato a SOCKS e che il vero indirizzo di destinazione venga ottenuto durante il processo di handshake SOCKS. Inoltra la richiesta al successivo MQIPT o al gestore code di destinazione esattamente nello stesso modo dei canali di connessione client. Include anche i canali mittenti del cluster definiti automaticamente.

Mittente / destinatario
Se MQIPT riceve una richiesta in entrata da un canale mittente, la inoltra al successivo MQIPT o al gestore code di destinazione esattamente come per i canali di connessione client. Il gestore code di destinazione convalida la richiesta in entrata e avvia il canale ricevente, se appropriato. Tutte le comunicazioni tra il canale mittente e destinatario (inclusi i flussi di sicurezza) vengono inoltrate.

Richiedente / server
Questa combinazione viene gestita nello stesso modo delle precedenti configurazioni. La convalida della richiesta di connessione viene effettuata dal canale server sul gestore code di destinazione.

Richiedente / mittente
La configurazione di "callback" potrebbe essere utile se ai due gestori code non è consentito stabilire connessioni dirette tra loro, ma è consentito connettersi a MQIPT e accettare connessioni da esso.

Server / richiedente e server / ricevitore
Questi sono gestiti da MQIPT nello stesso modo in cui gestisce la configurazione `Sender/Receiver` .

Condizioni di terminazione e di errore del canale

Quando MQIPT rileva la chiusura (normale o anomala) di un canale IBM MQ , propaga la chiusura del canale. Se si chiude un instradamento utilizzando MQIPT, tutti i canali che attraversano tale instradamento vengono chiusi.

MQIPT fornisce una funzione di timeout di inattività facoltativa. Se MQIPT rileva che un canale è stato inattivo per un periodo di tempo superiore al timeout, esegue una chiusura immediata sulle due connessioni in questione.

I sistemi IBM MQ a entrambe le estremità del canale osservano queste condizioni di arresto anomale come errori di rete o come terminazione del canale da parte del partner. Il canale è quindi in grado di riavviare e ripristinare (se l'errore si verifica durante un periodo di dubbio del protocollo) come se MQIPT non fosse utilizzato.

Sicurezza dei messaggi

La gestione delle code distribuite IBM MQ garantisce che i messaggi vengano consegnati correttamente. Questo è ancora il caso quando MQIPT è presente tra le due estremità del canale. MQIPT non memorizza i dati del messaggio o non partecipa alla procedura del punto di sincronizzazione che garantisce la corretta consegna del messaggio.

Quando si utilizzano messaggi IBM MQ veloci, non persistenti, se l'instradamento MQIPT non riesce o viene riavviato quando un messaggio IBM MQ è in transito, il messaggio potrebbe essere perso. Prima di riavviare la rotta, assicurarsi che tutti i canali di IBM MQ che utilizzano la rotta MQIPT siano inattivi.

Per ulteriori informazioni sulla sicurezza dei messaggi in IBM MQ, consultare Sicurezza dei messaggi.

Gestori code a più istanze e alta disponibilità

MQIPT può essere utilizzato con gestori code a più istanze in ambienti ad alta disponibilità.

MQIPT non ha uno stato persistente e quindi non è possibile eseguire il failover MQIPT su un altro sistema. Disporre invece di più istanze di MQIPT con file di configurazione `mqipt.conf` identici in esecuzione su sistemi differenti. Monitorare la disponibilità di ciascuna istanza di MQIPT e riavviarla (sullo stesso sistema) se necessario. Ciò fornisce una serie di istanze MQIPT identiche che possono essere utilizzate per instradare connessioni. È necessario quindi assicurarsi che IBM MQ possa instradare le connessioni a MQIPT e che MQIPT possa inoltrare tali connessioni al gestore code di destinazione.

I canali IBM MQ in uscita possono essere indirizzati a un'istanza MQIPT disponibile in diversi modi, ad esempio:

- Utilizzare un programma di bilanciamento del carico o un router ad alta disponibilità, ad esempio IBM Network Dispatcher dal prodotto WebSphere Edge Components.
- Specificare più nomi di connessione nella definizione del canale IBM MQ utilizzando un elenco separato da virgole. IBM MQ tenta quindi di connettersi a ciascun indirizzo di MQIPT finché non trova un'istanza MQIPT disponibile.

È inoltre necessario indirizzare le connessioni da MQIPT al gestore code di destinazione. Se la configurazione ad alta disponibilità garantisce il failover dell'indirizzo IP con il gestore code di destinazione, non è richiesta alcuna configurazione MQIPT speciale: specificare l'indirizzo IP di

destinazione nella proprietà di instradamento **Destination** e consentire all'operazione di failover di spostare l'indirizzo IP con il gestore code.

Tuttavia, se l'indirizzo IP del gestore code cambia dopo un failover, è necessario fare in modo che MQIPT inoltra la connessione alla destinazione corretta. Ciò può essere fatto in uno dei modi seguenti:

- Scrivere un'uscita di instradamento che controlli quale indirizzo IP e numero di porta sono accessibili e quindi sovrascrivere la destinazione di instradamento per ogni connessione. Alcune uscite di instradamento di esempio sono fornite con MQIPT; possono essere adattate a questo scopo.
- Utilizzare un programma di bilanciamento del carico ad alta disponibilità per reindirizzare la connessione.
- Definire più instradamenti MQIPT , uno per ogni indirizzo IP e porta in cui il gestore code potrebbe essere in esecuzione. Quindi indirizza le connessioni IBM MQ ai vari instradamenti MQIPT , ad esempio elencando tutti gli indirizzi IP di instradamento e i numeri di porta in un elenco separato da virgole nel nome della connessione del canale in uscita.

È anche importante ottimizzare tutti i componenti end-to-end sul percorso di rete:

1. I tentativi di connessione ai sistemi non disponibili devono avere esito negativo immediatamente in modo che i tentativi di riconnessione possano passare alla prima destinazione disponibile.

   Per gli instradamenti MQIPT SSL, ottimizzare la proprietà di instradamento **SSLClientConnectTimeout** per assicurare un errore di connessione di prompt per destinazioni non disponibili. Fare riferimento alla documentazione IBM MQ per dettagli sui parametri di ottimizzazione IBM MQ . Inoltre, consultare la documentazione del proprio sistema operativo per dettagli sull'ottimizzazione TCP/IP per il sistema operativo. In tutti i casi, i tentativi di connessione non riusciti dovrebbero restituire rapidamente un errore di rete (ad esempio, un pacchetto di ripristino TCP) o dovrebbero andare in timeout senza ritardi ingiustificati.
2. Le connessioni attive a un sistema non riuscito devono essere interrotte prontamente in modo da poter stabilire nuove connessioni.

   È inoltre necessario considerare l'impatto di un failover in un momento in cui le connessioni utilizzano attivamente MQIPT. È probabile che le connessioni di rete vengano interrotte durante un failover. Per le applicazioni client, è possibile utilizzare la funzione di riconnessione client automatica IBM MQ per ristabilire le connessioni interrotte. Per i canali dei messaggi, è possibile specificare un breve intervallo di tentativi in modo che il canale si riconnetta prontamente. Consultare la documentazione di IBM MQ per ulteriori informazioni sulla riconnessione automatica del client e sulla configurazione dei tentativi del canale dei messaggi.

IBM MQ Console e REST API

È possibile utilizzare IBM MQ Console e REST API per gestire IBM MQed eseguire operazioni di messaggistica utilizzando HTTP.

- È possibile utilizzare IBM MQ Console per eseguire attività di gestione di base da un browser web. Per ulteriori informazioni, consultare Amministrazione utilizzando IBM MQ Console.
- È possibile utilizzare administrative REST API per amministrare gli oggetti IBM MQ , come i gestori code e le code, gli agent e i trasferimenti Managed File Transfer . Per ulteriori informazioni, consultare Amministrazione utilizzando REST API.
- È possibile utilizzare messaging REST API per eseguire semplici operazioni point-to-point e pubblicare la messaggistica. Per ulteriori informazioni, consultare Messaggistica utilizzando REST API.

Opzioni di installazione

IBM MQ Console e REST API vengono eseguiti in un server WebSphere Liberty , denominato mqweb. Da IBM MQ 9.3.5, è possibile installare il server mqweb come componente facoltativo in un'installazione IBM MQ o come installazione IBM MQ Web Server autonoma.

V 9.4.0 Linux Installazione autonoma di IBM MQ Web Server

Da IBM MQ 9.4.0, il server mqweb può essere eseguito in una installazione autonoma di IBM MQ Web Server. Un'installazione autonoma di IBM MQ Web Server consente di installare ed eseguire il server mqweb su sistemi separati dalle installazioni di IBM MQ . L'installazione di un IBM MQ Web Server autonomo offre una maggiore flessibilità per quanto riguarda i sistemi e il numero di sistemi su cui si sceglie di eseguire i server mqweb. Se necessario, è possibile eseguire diverse istanze del server mqweb su macchine differenti per fornire la scalabilità e la disponibilità necessarie.

Se è stata acquistata la titolarità IBM MQ , è possibile installare tutte le copie richieste del IBM MQ Web Serverautonomo. Le installazioni IBM MQ Web Server non vengono conteggiate rispetto alla titolarità IBM MQ acquistata. Per ulteriori informazioni sulla licenza IBM MQ , consultare IBM MQ informazioni sulla licenza.

Le seguenti limitazioni si applicano in una installazione IBM MQ Web Server autonoma:

- IBM MQ Console può essere utilizzato per gestire solo i gestori code remoti.
- messaging REST API può essere utilizzato solo con gestori code remoti.
- administrative REST API non è disponibile.

Il IBM MQ Web Server autonomo è supportato solo su piattaforme Linux .

Per ulteriori informazioni sull'installazione del IBM MQ Web Serverautonomo, consultare Installazione del IBM MQ Web Serverautonomo.

Componente facoltativo di un'installazione IBM MQ

È possibile scegliere di installare il componente IBM MQ Console e REST API come parte di un'installazione IBM MQ .

Tutte le funzioni IBM MQ Console e REST API sono disponibili quando il server mqweb viene eseguito in un'installazione di IBM MQ .

- IBM MQ Console può essere utilizzato per gestire gestori code locali e remoti.
- messaging REST API può essere utilizzato con gestori code locali e remoti.
- administrative REST API può essere utilizzato per gestire gestori code locali e remoti.

Per utilizzare il componente IBM MQ Console e REST API , installare il seguente componente come parte dell'installazione di IBM MQ :

- AIX Su AIX, installare il fileset `mqm.web.rte` .
- IBM i Su IBM i, installare il componente WEB.
- Linux Su Linux, installare il componente `MQSeriesWeb` .
- Windows Su Windows, installare la funzione `Web Administration` .
- z/OS Su z/OS, installare la funzione `IBM MQ for z/OS UNIX System Services Web Components` .

Informazioni particolari

Queste informazioni sono state sviluppate per prodotti e servizi offerti negli Stati Uniti.

IBM potrebbe non offrire i prodotti, i servizi o le funzioni descritti in questo documento in altri paesi. Consultare il rappresentante IBM locale per informazioni sui prodotti e sui servizi disponibili nel proprio paese. Ogni riferimento relativo a prodotti, programmi o servizi IBM non implica che solo quei prodotti, programmi o servizi IBM possano essere utilizzati. In sostituzione a quelli forniti da IBM possono essere usati prodotti, programmi o servizi funzionalmente equivalenti che non comportino la violazione dei diritti di proprietà intellettuale o di altri diritti dell'IBM. Tuttavia, è responsabilità dell'utente valutare e verificare il funzionamento di qualsiasi prodotto, programma o servizio non IBM.

IBM potrebbe disporre di applicazioni di brevetti o brevetti in corso relativi all'argomento descritto in questo documento. La fornitura di tale documento non concede alcuna licenza a tali brevetti. Chi desiderasse ricevere informazioni relative a licenze può rivolgersi per iscritto a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Per richieste di licenze relative ad informazioni double-byte (DBCS), contattare il Dipartimento di Proprietà Intellettuale IBM nel proprio paese o inviare richieste per iscritto a:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

Il seguente paragrafo non si applica al Regno Unito o a qualunque altro paese in cui tali dichiarazioni sono incompatibili con le norme locali: INTERNATIONAL BUSINESS MACHINES CORPORATION FORNISCE LA PRESENTE PUBBLICAZIONE "NELLO STATO IN CUI SI TROVA" SENZA GARANZIE DI ALCUN TIPO, ESPRESSE O IMPLICITE, IVI INCLUSE, A TITOLO DI ESEMPIO, GARANZIE IMPLICITE DI NON VIOLAZIONE, DI COMMERCIABILITÀ E DI IDONEITÀ PER UNO SCOPO PARTICOLARE. Alcuni stati non consentono la rinuncia a garanzie esplicite o implicite in determinate transazioni; quindi la presente dichiarazione potrebbe non essere applicabile.

Questa pubblicazione potrebbe contenere imprecisioni tecniche o errori tipografici. Le informazioni incluse in questo documento vengono modificate su base periodica; tali modifiche vengono incorporate nelle nuove edizioni della pubblicazione. IBM si riserva il diritto di apportare miglioramenti o modifiche al prodotto/i e/o al programma/i descritti nella pubblicazione in qualsiasi momento e senza preavviso.

Tutti i riferimenti a siti Web non dell'IBM contenuti in questo documento sono forniti solo per consultazione e non rappresenta in alcun modo un'approvazione di tali siti. I materiali reperibili in tali siti Web non fanno parte dei materiali relativi a questo prodotto IBM e l'utilizzo di tali siti è responsabilità dell'utente.

Tutti i commenti e i suggerimenti inviati potranno essere utilizzati liberamente da IBM e diventeranno esclusiva della stessa.

Coloro che detengono la licenza su questo programma e desiderano avere informazioni su di esso allo scopo di consentire (i) uno scambio di informazioni tra programmi indipendenti ed altri (compreso questo) e (ii) l'uso reciproco di tali informazioni, dovrebbero rivolgersi a:

IBM Corporation
Coordinatore interoperabilità software, Dipartimento 49XA
Autostrada 3605 52 N

Rochester, MN 55901
U.S.A.

Queste informazioni possono essere rese disponibili secondo condizioni contrattuali appropriate, compreso, in alcuni casi, il pagamento di un addebito.

Il programma su licenza descritto in queste informazioni e tutto il materiale su licenza disponibile per esso sono forniti da IBM in base ai termini dell' IBM Customer Agreement, IBM International Program License Agreement o qualsiasi altro accordo equivalente tra le parti.

Tutti i dati relativi alle prestazioni contenuti in questo documento sono stati determinati in un ambiente controllato. Pertanto, i risultati ottenuti in altri ambienti operativi possono variare in modo significativo. Alcune misurazioni potrebbero essere state fatte su sistemi a livello di sviluppo e non vi è alcuna garanzia che queste misurazioni saranno le stesse sui sistemi generalmente disponibili. Inoltre, alcune misurazioni potrebbero essere state stimate mediante estrapolazione. I risultati quindi possono variare. Gli utenti di questo documento dovrebbero verificare i dati applicabili per il loro ambiente specifico.

Le informazioni relative a prodotti non IBM provengono dai fornitori di tali prodotti, dagli annunci pubblicati o da altre fonti pubblicamente disponibili. IBM non ha verificato tali prodotti e, pertanto, non può garantirne l'accuratezza delle prestazioni. Eventuali commenti relativi alle prestazioni dei prodotti non IBM devono essere indirizzati ai fornitori di tali prodotti.

Tutte le dichiarazioni riguardanti la direzione o l'intento futuro di IBM sono soggette a modifica o ritiro senza preavviso e rappresentano solo scopi e obiettivi.

Questa pubblicazione contiene esempi di dati e prospetti utilizzati quotidianamente nelle operazioni aziendali, Per poterli illustrare nel modo più completo possibile, gli esempi riportano nomi di persone, società, marchi e prodotti. Tutti questi nomi sono fittizi e qualsiasi somiglianza con nomi ed indirizzi adoperati da imprese realmente esistenti sono una mera coincidenza.

LICENZA SUL COPYRIGHT:

Queste informazioni contengono programmi applicativi di esempio in lingua originale, che illustrano le tecniche di programmazione su diverse piattaforme operative. È possibile copiare, modificare e distribuire questi programmi di esempio sotto qualsiasi forma senza alcun pagamento alla IBM, allo scopo di sviluppare, utilizzare, commercializzare o distribuire i programmi applicativi in conformità alle API (application programming interface) a seconda della piattaforma operativa per cui i programmi di esempio sono stati scritti. Questi esempi non sono stati testati approfonditamente tenendo conto di tutte le condizioni possibili. IBM, quindi, non può garantire o sottintendere l'affidabilità, l'utilità o il funzionamento di questi programmi.

Se si sta visualizzando queste informazioni in formato elettronico, le fotografie e le illustrazioni a colori potrebbero non apparire.

Informazioni sull'interfaccia di programmazione

Le informazioni sull'interfaccia di programmazione, se fornite, consentono di creare software applicativo da utilizzare con questo programma.

Questo manuale contiene informazioni sulle interfacce di programmazione che consentono al cliente di scrivere programmi per ottenere i servizi di IBM MQ.

Queste informazioni, tuttavia, possono contenere diagnosi, modifica e regolazione delle informazioni. La diagnosi, la modifica e la regolazione delle informazioni vengono fornite per consentire il debug del software applicativo.

Importante: Non utilizzare queste informazioni di diagnosi, modifica e ottimizzazione come interfaccia di programmazione poiché sono soggette a modifica.

Marchi

IBM, il logo IBM , ibm.com, sono marchi di IBM Corporation, registrati in molte giurisdizioni nel mondo. Un elenco aggiornato dei marchi IBM è disponibile sul web in "Copyright and trademark

information"www.ibm.com/legal/copytrade.shtml. Altri nomi di prodotti e servizi potrebbero essere marchi di IBM o altre società.

Microsoft e Windows sono marchi di Microsoft Corporation negli Stati Uniti, in altri paesi o entrambi.

UNIX è un marchio registrato di The Open Group negli Stati Uniti e/o in altri paesi.

Linux è un marchi registrato di Linus Torvalds negli Stati Uniti e/o in altri paesi.

Questo prodotto include il software sviluppato da Eclipse Project (https://www.eclipse.org/).

Java e tutti i marchi e i logo Java sono marchi registrati di Oracle e/o di società affiliate.

Numero parte:

(1P) P/N: