# Rabbit Mq trainig course

httpstraining.cloudamqp.comcourse3

<https://www.youtube.com/watch?v=sN5YpfOpCHA>

<https://www.youtube.com/watch?v=sN5YpfOpCHA>

https://okyrylchuk.dev/blog/mastering-cancellation-in-csharp-with-cancellationtoken/

## Introduzione

Immagine che contiene testo, schermata, Carattere, numero

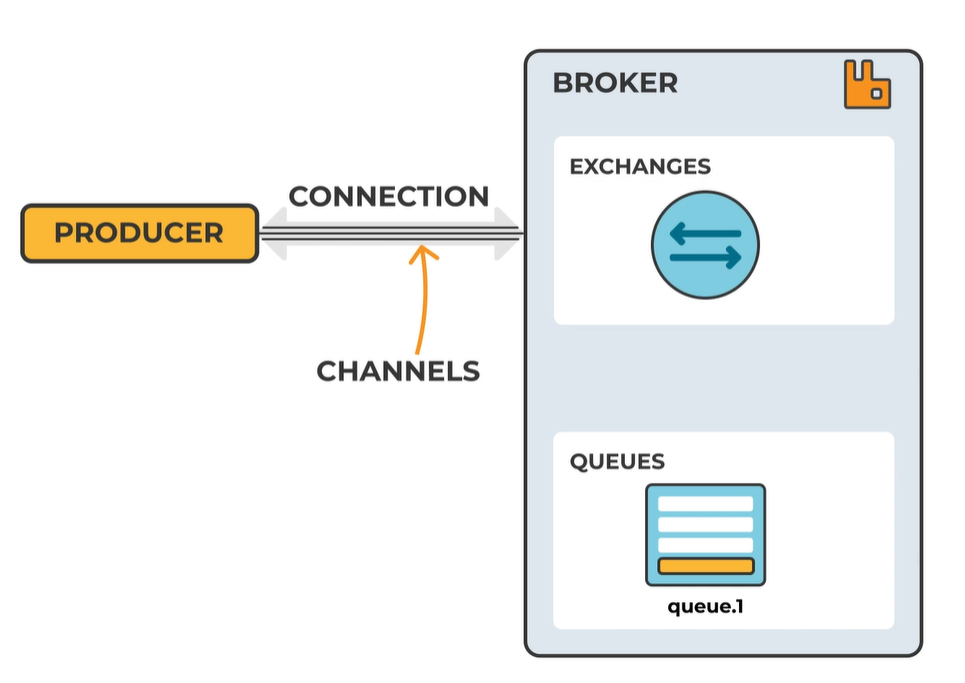
Il contenuto generato dall'IA potrebbe non essere corretto.

Message: bytearray con un header on top

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.Una coda è un message storage buffer, una connessione viene stabilita tra il producer ed il brocker ed **il canale è una connessione virtuale dentro questa connessione**.

Possiamo avere più canali in una connessione ed il producer spedisce il messaggio **all’exchange (che è come il postino che si occupa di consegnare il messaggio alla giusta coda:**



**Il messaggio viene consegnato alla coda in base al binding che vie è tra la coda e l’exchange** e dipende anche dalla **routing key**:

Immagine che contiene testo, schermata, Carattere, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.

Il messaggio contiene info come la durability, priority ed è formato dalle seguenti parti:

Immagine che contiene testo, Carattere, bianco, lavagna

Il contenuto generato dall'IA potrebbe non essere corretto.

**Il payload è il contenuto principale trasportato dal messaggio**. L’ack è fondamentale per confermare la ricezione del messaggio.

Immagine che contiene testo, schermata, Carattere, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.

Quando la gestione di un evento va a buon fine si da l’ack mentre se non va a buon fine si può dare il nak o il reject, per come funzionano le librerie di abletech quando fai nak l’elemento continua a rimanere in testa:



Se C va in errore continua a rimanere li dove è C se do reject:



In realtà quando si fa il reject C viene messo in una coda di retry temporaneamente:

Immagine che contiene testo, schermata, strumento, design

Il contenuto generato dall'IA potrebbe non essere corretto.

E poi quando dopo un minuto arriva il tempo di retry C viene messo qua:



In sostanza se faccio il Nak devo prevedere dei meccanismi che se ho un errore non frullo come un matto.

Se dobbiamo processare i messaggi nell’ordine cronologico con cui ci vengono dati il nak è d’obbligo.

Facciamo un esempio pratico, avevo bisogno durante uno sviluppo in Abletech di recuperare dei messaggi relativi a delle sottoscrizioni proveniente da retail (un altro applicativo) Quelli del reparto retailer ogni volta che c’è un evento di creazione cliente fanno un publish verso l’exchange di rabbit, questa cosa ha una serie di eventi che riveverò con un mio subscriber per consumarli e che dovrò scremare mediante dei filtri. Nello specifico il dto che arrivava dalla coda era VariazioneAnagraficaDto, l’ho filtrato per IdDistributore.

Lexchange di retail forcava verso due code (c’erano due code in produzione che probabilmente servivano già altre applicazioni), per il mio applicativo è stata creata una nuova coda dal devops (nuovo flusso per CR):

Immagine che contiene testo, Carattere, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

Qui sotto possiamo vedere l’exchange nella gui di rabbitmq con le due code preesistenti

Immagine che contiene testo, schermata, software, Pagina Web

Il contenuto generato dall'IA potrebbe non essere corretto.

Possiamo avere più virtualhost sulla macchina:

Immagine che contiene testo, schermata, Viso umano, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.

## Inviamo il primo messaggio

Prima creiamo una coda (image\_scaling\_queue)

Poi creiamo un exchange (image\_scaling\_exchange)

Poi settiamo il binding nell’exchange:

Immagine che contiene testo, schermata, numero, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.

Quindi inviamo un messaggio all’exchange:

Immagine che contiene testo, schermata, linea, software

Il contenuto generato dall'IA potrebbe non essere corretto.

Concetti AMQP

Per inviare un messaggio o gestire le code, è necessario stabilire un canale. Il canale impacchetta i messaggi e gestisce le operazioni di protocollo. I client inviano messaggi tramite il metodo "basic\_publish" del canale.

Anche la creazione e la manutenzione della coda vengono inviate tramite un canale. I comandi AMQP come "queue.create" e "exchange.create" vengono tutti inviati tramite un canale. La chiusura di una connessione chiude tutti i canali associati.

**Code**:

Le code hanno proprietà che definiscono il loro comportamento e queste proprietà vengono passate al broker quando la coda viene dichiarata.

* Una coda può essere contrassegnata come durevole, specificando se la coda deve sopravvivere al riavvio del broker.
* Una coda può essere esclusiva, il che specifica se la coda può essere utilizzata da una sola connessione. Una coda esclusiva viene eliminata quando quella connessione si chiude.
* Una coda può anche essere dichiarata con la proprietà auto-delete, il che significa che una coda che ha avuto almeno un consumatore viene eliminata quando l'ultimo consumatore annulla l'iscrizione.
* Esistono anche alcune proprietà opzionali utilizzate dai plugin e dalle funzionalità specifiche del broker, come TTL, che indica a una coda inutilizzata quando scadere dopo un determinato periodo di tempo.

**Exchange**

Un exchange è responsabile del routing dei messaggi a diverse code con l'aiuto di attributi di intestazione, binding e chiavi di routing.

Gli exchange possono, come le code, essere configurati con parametri quali durevole, temporaneo e auto-eliminati alla creazione.

In RabbitMQ, ci sono quattro diversi tipi di exchange che instradano il messaggio in modo diverso utilizzando parametri e configurazioni di binding differenti. I client possono creare i propri exchange o utilizzare gli exchange predefiniti che vengono creati quando il server si avvia per la prima volta.

Per creare un exchange:

**channel.exchange\_declare(exchange='direct\_exchange', exchange\_type='direct')**

**Binding**

Un binding è un'associazione o una relazione tra una coda e un exchange.

I binding possono accettare un parametro extra chiamato routing\_key. Se ricordi, una chiave di routing può anche essere inviata con un messaggio. La chiave di routing sul binding a volte chiamata chiave di binding e la chiave di routing nel messaggio sono le cose che lo scambio sta guardando durante la consegna dei messaggi.

**channel.queue\_bind(exchange=exchange\_name, queue=queue\_name, routing\_key='black')**

**I tipi di exchange**

Parleremo di come exchange e code sono associati tra loro e daremo esempi di come puoi usare diversi tipi di exchange in quattro scenari diversi.

**Il modo in cui un messaggio viene indirizzato dipende da diversi fattori, tra cui il tipo di exchange che specifica un certo numero di regole di indirizzamento, chiavi di routing e attributi di intestazione. Tutti questi fungono da indirizzi per i messaggi.**

Dal punto di vista di una coda, puoi controllare quali exchange e regole di routing sono collegati a questa specifica coda. Questi collegamenti sono chiamati binding.

Un binding collega la coda a uno exchange, mentre la **chiave di routing** è come un indirizzo per il messaggio.

Questo è principalmente ciò che lo exchange cerca quando decide come instradare il messaggio alle code.

Utilizzeremo un esempio di una compagnia di taxi per spiegare i diversi tipi di exchange.

In RabbitMQ ci sono quattro tipi principali di Exchange:

* **Diretto** (direct)  
  **indirizza il messaggio a una coda specifica esaminando la chiave di routing**  
  Nel nostro esempio, lo scambio diretto viene utilizzato quando un utente richiede un taxi specifico, come il suo autista preferito.
* Argomento (**topic**)  
  **Lo scambio di argomenti indirizza i messaggi a una o più code esaminando la chiave di routing**. **La chiave di routing nel messaggio viene confrontata per le corrispondenze con i pattern di chiavi di routing sui binding**.  
  Nel nostro esempio un cliente con un gruppo di amici chiede un grande taxi ecologico. Questo ordine viene instradato tramite un exchange legato a taxi di questo tipo.  
  In un altro esempio **un cliente ordina un taxi grande ma non gli importa del tipo di taxi. Lo scambio di argomenti indirizza questo messaggio a tutti i taxi vincolati come taxi grande.**
* Uscita a ventaglio (**fanout**)  
  **Lo scambio fanout copia e instrada un messaggio ricevuto a tutte le code ad esso collegate, indipendentemente dalle chiavi di routing**. Una chiave di routing fornita viene semplicemente ignorata.  
  Nel nostro esempio, il Fanout Exchange viene utilizzato quando **i coordinatori dei taxi informano tutti i tassisti di una strada bloccata.**
* Intestazioni (**headers**)  
  **Gli scambi di intestazioni sono molto simili agli scambi di argomenti, ma instradano i messaggi in base ai valori delle intestazioni anziché alle chiavi di instradamento**.  
  In questo esempio un servizio scrive un report per tutti i taxi attorno a Manhattan, New York, tenendo traccia del consumo di carburante e dei chilometri percorsi. Questi dati sono inclusi nel messaggio.   
  Un argomento speciale denominato "x-match", aggiunto nel binding tra scambio e coda, specifica se le intestazioni devono corrispondere a "tutti" o "qualsiasi".  
  Questi dati sono inclusi nel messaggio. Quando "x-match" è impostato su "any", insieme ai seguenti argomenti, il report di New York riceve tutti i messaggi dai taxi con viaggi da, per o all'interno di New York. Un viaggio che inizia a New York e termina a Jersey sarà quindi incluso.

Un altro report potrebbe essere interessato ai viaggi solo all'interno di New York. Questo può essere ottenuto semplicemente impostando "x-match" su "all". Il nuovo report riceverà quindi solo messaggi in cui "from" e "to" sono entrambi impostati su New York, viaggi che non lasciano mai la città.

**L’ack del consumatore**

Gli acknowledgment consentono al server e ai client di sapere quando ritrasmettere i messaggi. Il client può confermare il messaggio quando lo riceve o quando lo ha completamente elaborato.   
Publish confirm è lo stesso concetto, ma per la pubblicazione. Il server conferma quando ha ricevuto un messaggio da un publisher.  
Un messaggio può essere considerato recapitato con successo immediatamente dopo essere stato inviato, una volta scritto su un socket TCP, OPPURE quando viene ricevuto un riconoscimento esplicito dal client. Il riconoscimento inviato manualmente può essere positivo o negativo.

Una volta ricevuto un riconoscimento, il messaggio può essere scartato dalla coda. Se il consumatore non riesce a elaborare un messaggio, il risultato desiderato potrebbe essere quello di rimetterlo in coda e lasciare che un altro consumatore lo riceva e lo gestisca, oppure di riprovare l'elaborazione in un secondo momento.

**Host virtuali**

**Gli host virtuali (o vhost) in RabbitMQ forniscono un modo per separare le applicazioni che utilizzano la stessa istanza di RabbitMQ.**

Quando un client stabilisce una connessione al server RabbitMQ, specifica il vhost all'interno del quale opererà.

Risorse come scambi e code sono risorse denominate all'interno del contenitore vhost, rendendo ogni vhost essenzialmente un mini-server RabbitMQ.

Quando si configura RabbitMQ, è necessario almeno un vhost, che di default è solo una barra "/".