

RABBIT-MQ AMQP

Claudio Biancalana

Fonti e riferimenti

- <https://www.rabbitmq.com/tutorials/amqp-concepts.html>
- Advanced Message Queuing Protocol AMQP: Upgrader's Guide – Gerald Blokdyk - 2017
- RabbitMQ in Depth – Gavin M. Roy – 2017
- RabbitMQ in Action: Distributed Messaging for Everyone – Alvaro Videla e Jason J. W. Williams - 2012

AMQP

Advanced Message Queuing Protocol

- AMQP (Advanced Message Queuing Protocol) è un protocollo che consente alle applicazioni client di comunicare con i broker middleware di messaggistica.
- I Broker di messaggistica ricevono messaggi dai produttori e li indirizzano ai consumatori.
- Poiché si tratta di un protocollo di rete, i produttori e il broker possono risiedere su macchine diverse.

AMPQ

- Secondo il sito ufficiale di AMPQ (<http://www.amqp.org>):

AMQP è uno standard aperto per il passaggio di messaggi tra applicazioni o organizzazioni.

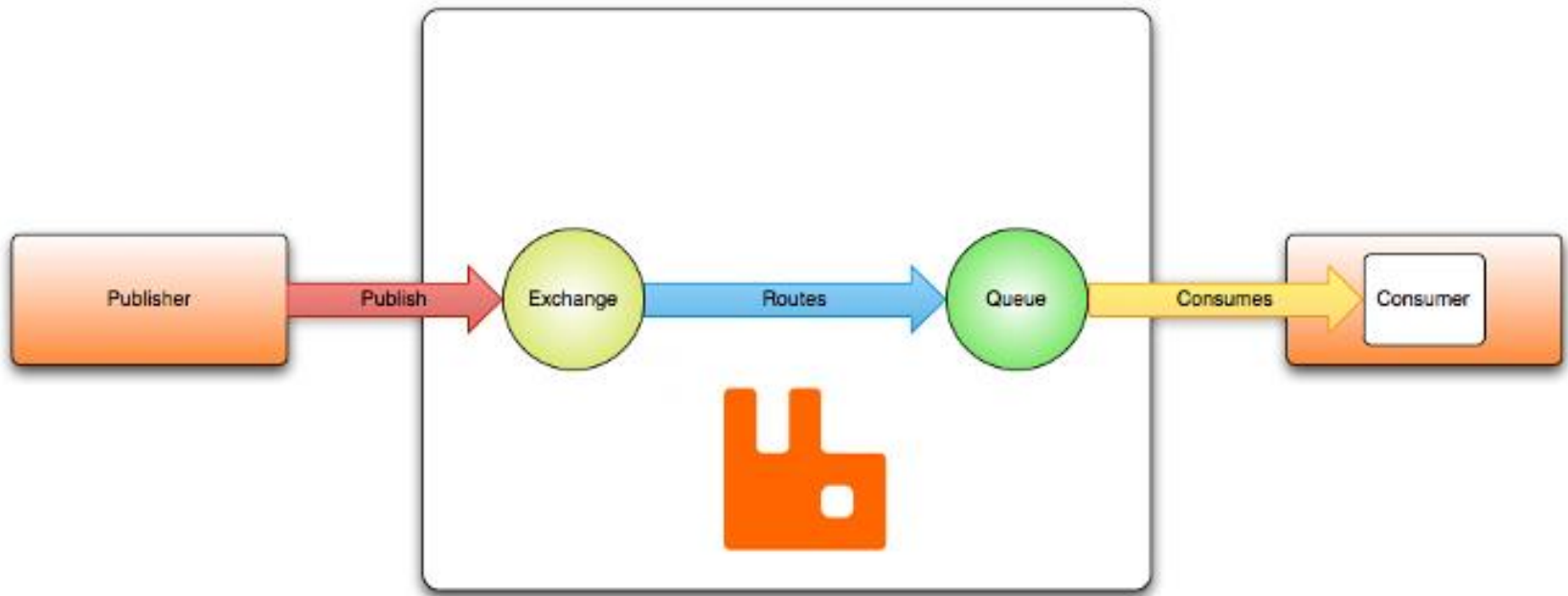
- AMQP definisce
 - le proprietà dei messaggi
 - le proprietà delle code
 - le modalità di instradamento dei messaggi tra applicazioni e client
 - le modalità con cui i broker di messaggi assicurano che il messaggio sia ricevuto o inviato
 - altri problemi simili come **affidabilità** e **sicurezza**.

AMQP

- Il protocollo AMQP ha la seguente visione del mondo:
 - i messaggi vengono pubblicati negli **exchange**, che vengono spesso comparati con gli uffici postali o le cassette postali.
 - Gli **exchange** quindi distribuiscono le copie dei messaggi alle code usando le regole chiamate **bind**
 - Il **broker** consegna i messaggi ai consumatori abbonati alle code, oppure i consumatori recuperano/estraggono i messaggi dalle code su richiesta (Push vs Pull)

AMQP

Esempio 'semplice'



RabbitMQ

- RabbitMQ è un software di Message Broker open source che cerca di risolvere i problemi di messaggistica implementando **AMQP**.
- RabbitMQ è concesso in licenza con Mozilla Public License. RabbitMQ è diventato parte di GoPivotal a maggio 2013 e la comunità ha aiutato nello sviluppo di RabbitMQ.

 RabbitMQ by Pivotal



AMQP in pillole (1)

- Quando si pubblica un messaggio, i publisher possono specificare vari attributi del messaggio (metadati del messaggio).
- Alcuni di questi metadati possono essere utilizzati dal broker, tuttavia il resto è completamente opaco per il broker e viene utilizzato solo dalle applicazioni che ricevono il messaggio.

Fallacies

Le reti sono inaffidabili e le applicazioni potrebbero non riuscire a elaborare i messaggi

- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn't change
- There is one administrator
- Transport cost is zero
- The network is homogeneous

AMQP in pillole (2)

Acknowledgement

- Le reti sono inaffidabili e le applicazioni potrebbero non riuscire a elaborare i messaggi, pertanto il modello AMQP introduce la nozione di 'ack' dei messaggi:
 - quando un messaggio viene consegnato a un consumatore il consumatore avvisa il broker (ack).
 - con questa modalità, un broker rimuoverà completamente un messaggio da una coda solo quando riceve una notifica per quel messaggio (o gruppo di messaggi).

AMQP in pillole (3)

- In alcune situazioni, ad esempio, quando un messaggio non può essere instradato, i messaggi possono essere *restituiti* ai publisher, eliminati o, se il broker implementa un'opportuna estensione, inseriti in una cosiddetta "coda di messaggi non instradabili".
- I publisher scelgono come gestire situazioni come questa pubblicando messaggi con determinati parametri.
- Le **queue**, gli **exchange** e i **bind** vengono definiti come *entità AMQP*.

AMQP è un protocollo programmabile

- AMQP è un protocollo programmabile nel senso che le entità e gli schemi di routing di AMQP sono definiti principalmente dalle applicazioni stesse, non da un amministratore di broker. Di conseguenza, sono previste operazioni di protocollo che dichiarano **queue** ed **exchange**, definiscono collegamenti tra loro, si iscrivono a **queue** e così via.
- Ciò offre agli sviluppatori di applicazioni molta libertà ma richiede anche che siano consapevoli dei potenziali conflitti di definizione. I conflitti di definizione sono rari e spesso indicano una configurazione errata.
- Le applicazioni dichiarano le entità AMQP di cui hanno bisogno, definiscono gli schemi di routing necessari e possono scegliere di eliminare le entità AMQP quando non vengono più utilizzate.

Exchange (1)

- Gli **exchange** sono entità AMQP a cui vengono inviati i messaggi. Gli **exchange** accettano un messaggio e lo instradano a zero o più code.
- L'algoritmo di routing utilizzato dipende dal tipo di **exchange** e dalle regole chiamate **bind**.
- I broker AMQP offrono quattro tipi di exchange:

Nome	Nomi predefiniti
Direct Exchange	(Stringa vuota) o amq.direct
Fanout Exchange	amq.fanout
Topic Exchange	amq.topic
Headers Exchange	amq.match – amq.headers

Exchange (2)

- Oltre al tipo, gli exchange, sono caratterizzati da un numero di attributi, i più importanti dei quali sono:
 - **Nome [Name]**
 - **Durabilità [Durability]** (sopravvivono al riavvio del broker)
 - **Eliminazione automatica [Auto-delete]** (viene eliminato quando l'ultima coda non è associata)
 - **Argomenti [Arguments]** (facoltativo, utilizzato da plugin e funzionalità specifiche del broker)
- Gli exchange possono essere **durevoli** o **transitori**. Gli exchange durevoli sopravvivono al riavvio del broker, mentre gli exchange transitori non lo fanno (devono essere dichiarati nuovamente quando il broker torna online).
- Non tutti gli scenari e i casi d'uso richiedono che gli scambi siano durevoli.

Default Exchange

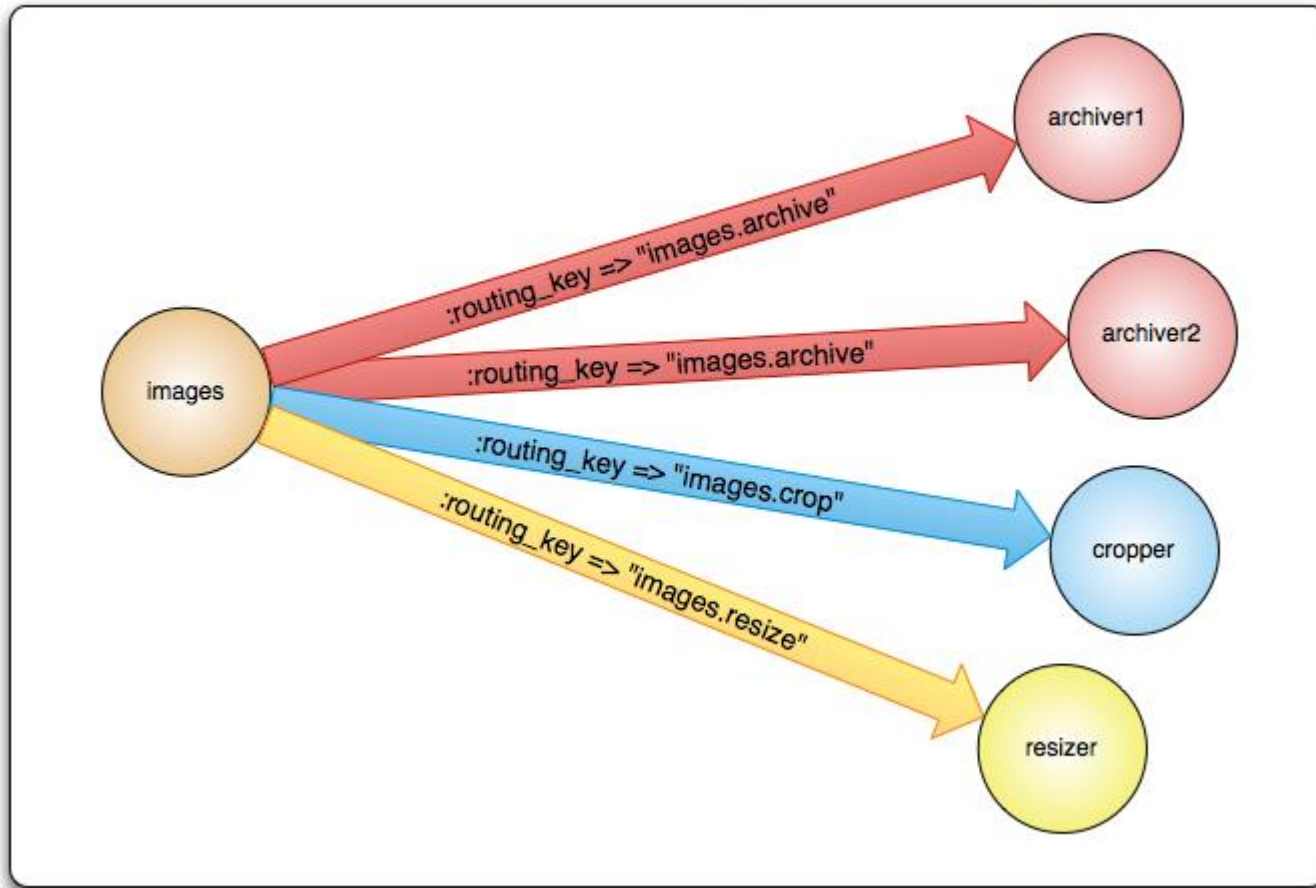
- Il **default exchange** è un **direct exchange** senza nome (stringa vuota) pre-dichiarato dal broker. Ha una proprietà speciale che lo rende molto utile per le applicazioni semplici: **ogni coda creata viene automaticamente associata al default exchange con una chiave di routing che è uguale al nome della coda.**
- Ad esempio, quando si dichiara una coda con il nome di 'hello-world', il broker AMQP lo assocerà allo scambio predefinito utilizzando 'hello-world' come chiave di routing.
- Pertanto, un messaggio pubblicato del default exchange con la chiave di routing 'hello-world' verrà instradato alla coda 'hello-world'. In altre parole, **il default exchange fa sembrare che sia possibile recapitare i messaggi direttamente alle code, anche se tecnicamente non è ciò che sta accadendo.**

Direct Exchange

- Un **direct exchange** consegna i messaggi alle code in base alla chiave di routing dei messaggi. Uno scambio diretto (direct exchange) è ideale per l'instradamento unicast di messaggi (sebbene possano essere utilizzati anche per l'instradamento multicast).
- Ecco come funziona:
 - Una coda si lega (bind) allo scambio con una chiave di routing K
 - Quando un nuovo messaggio con chiave di routing R arriva al direct exchange, viene inoltrato alla coda se $K = R$
- Gli scambi diretti vengono spesso utilizzati per distribuire le attività tra più worker (istanze della stessa applicazione) in modo circolare (round-robin).
- Attenzione! in **AMQP**, i messaggi sono bilanciati in base al carico tra i consumatori e non tra le code.

Esempio

Direct exchange routing



Fanout Exchange

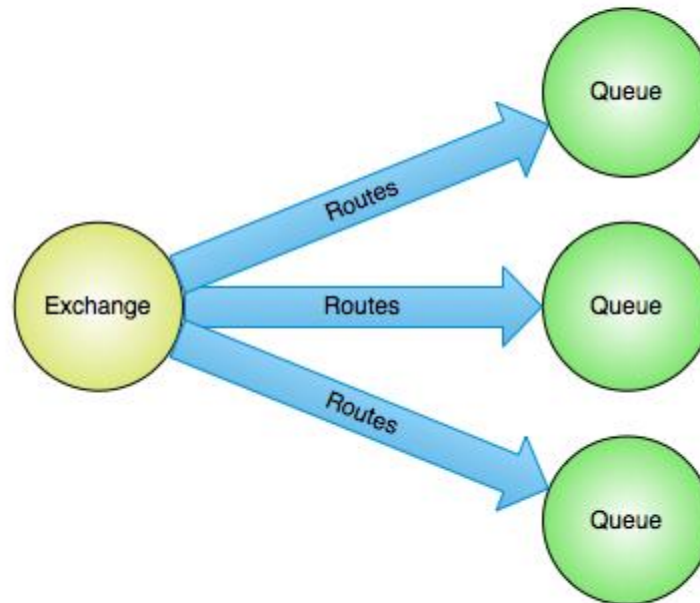
- Un **fanout exchange** instrada i messaggi a tutte le code ad esso associate e **la chiave di routing viene ignorata**.
- Se N code sono legate (**bind**) ad un fanout exchange, quando un nuovo messaggio viene pubblicato in tale exchange, una copia del messaggio viene recapitata a tutte le N code.
- I fanout exchange sono ideali per implementare meccanismi di instradamento dei messaggi.

Esempi

- Un fanout exchange consegna una copia di un messaggio ad ogni coda associata ad esso.
- Esempi d'utilizzo:
 - I giochi online multigiocatore di massa (MMO) possono utilizzarlo per gli aggiornamenti della classifica o altri eventi globali
 - I siti di notizie sportive possono utilizzarlo per distribuire gli aggiornamenti sulle partite in corso ai client mobili (quasi) in tempo reale
 - I sistemi distribuiti possono trasmettere vari aggiornamenti di stato e configurazione
 - Le chat di gruppo possono distribuire i messaggi tra i partecipanti

Esempio

Fanout exchange routing



Topic Exchange

- Il **topic exchange** indirizza i messaggi a una o più code in base alla corrispondenza tra una chiave di routing dei messaggi e il modello utilizzato per associare una coda a un exchange.
- Il topic exchange viene spesso utilizzato per implementare varianti del modello di pubblicazione/sottoscrizione (publish subscribe).
- I topic exchange sono comunemente utilizzati per l'instradamento multicast di messaggi.

Casi d'uso

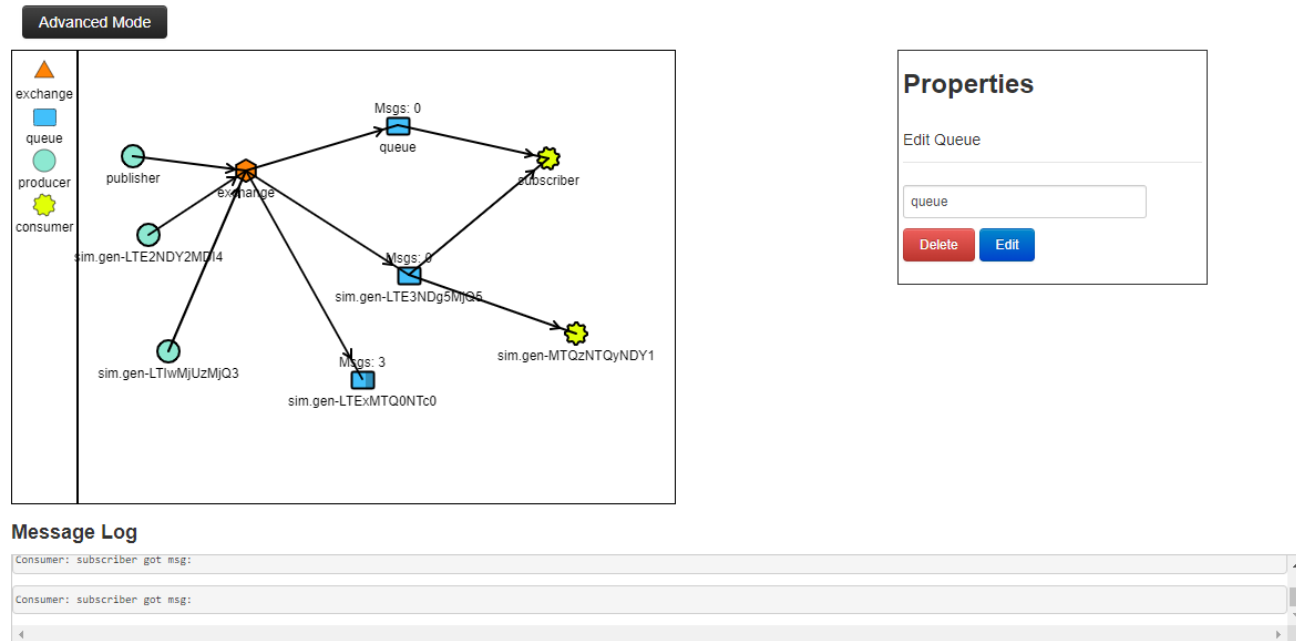
- I topic exchange hanno una vasta gamma di casi d'uso.
- Ogni volta che un problema coinvolge più consumatori/applicazioni che scelgono selettivamente quale tipo di messaggi vogliono ricevere, ad esempio:
 - Distribuzione di dati relativi a specifiche aree geografiche, ad esempio punti vendita
 - Elaborazione di attività in background eseguita da più worker, ciascuno in grado di gestire un set specifico di attività
 - Aggiornamenti dei prezzi delle scorte (e aggiornamenti su altri tipi di dati finanziari)
 - Aggiornamenti di notizie che comportano la categorizzazione o la codifica (ad esempio, solo per un determinato sport o squadra)

Esercizio

- Provare i concetti e le topologie con uno strumento on-line
 - <http://tryrabbitmq.com/>

RabbitMQ Simulator

Use the drawing area below to represent your messaging topology. Drag messaging elements from the toolbox on the left to the canvas. To connect nodes, hold the ALT key (or SHIFT key) and drag from a source node to connect it to a destination node.



Header Exchange

- Un **header exchange** è progettato per il routing su più attributi che sono più facilmente espressi come intestazioni di messaggi rispetto a una chiave di routing. **Gli headers exchange ignorano l'attributo chiave di routing. Invece, gli attributi utilizzati per il routing sono presi dall'attributo headers.**
- Un messaggio viene considerato corrispondente se il valore dell'intestazione è uguale al valore specificato al momento dell'associazione.
- È possibile associare una coda ad un headers exchange utilizzando più di un'intestazione per la corrispondenza.

Coda (1)

- Le code nel modello AMQP sono molto simili alle code in altri sistemi di accodamento di messaggi e attività: memorizzano i messaggi consumati dalle applicazioni.
- Le code condividono alcune proprietà con gli exchange, ma hanno anche alcune proprietà aggiuntive:
 - **Nome [Name]**
 - **Durabilità [Durability]** (la coda sopravviverà al riavvio del broker)
 - **Esclusivo [Exclusive]** (utilizzato da una sola connessione e la coda verrà eliminata alla chiusura della connessione)
 - **Eliminazione automatica [Auto-Delete]** (la coda che ha avuto almeno un consumatore viene eliminata quando l'ultimo consumatore annulla l'iscrizione)

Coda (2)

- Prima che una coda possa essere utilizzata, deve essere dichiarata.
- La dichiarazione di una coda provocherà la sua creazione se non esiste già.
- La dichiarazione non avrà alcun effetto se la coda esiste già e i suoi attributi sono gli stessi di quelli nella dichiarazione (idempotenza).
- Quando gli attributi della coda esistenti non coincidono con quelli della dichiarazione, verrà sollevata un'eccezione a livello di canale con codice 406 (PRECONDITION_FAILED).

Durata della coda

- Le code permanenti vengono mantenute sul disco e quindi sopravvivono al riavvio del broker.
- Le code non durevoli vengono chiamate transitorie.
- Non tutti gli scenari e i casi d'uso richiedono che le code siano permanenti.
- La durabilità di una coda non rende durevoli i messaggi instradati verso quella coda. Se il broker viene rimosso e quindi ripristinato, la coda verrà nuovamente dichiarata all'avvio del broker, tuttavia verranno recuperati solo i messaggi persistenti .

Binding (o bind)

- I **binding (o bind)** sono regole che gli exchange utilizzano (tra le altre cose) per instradare i messaggi alle code.
- Per indicare a un exchange E di instradare i messaggi a una coda Q, Q deve essere associato a E.
- I binding possono avere un attributo chiave di routing (opzionale) utilizzato da alcuni tipi di exchange.
- Lo scopo della chiave di routing è fornire la capacità di selezionare determinati messaggi pubblicati in un exchange da instradare alla coda associata.
- In altre parole, la chiave di routing si comporta come un filtro.

Analogia... con un po' di fantasia

- La coda è come la tua destinazione a Roma
- L'exchange è come l'aeroporto Leonardo Da Vinci
- I binding sono percorsi da Leonardo Da Vinci a destinazione. Ci possono essere zero o molti modi per raggiungerla.



Perché non solo code?

- Avere questo livello di indirizzione consente scenari di routing impossibili o molto difficili da implementare usando la pubblicazione direttamente nelle code ed elimina anche una certa quantità di lavoro duplicato che gli sviluppatori di applicazioni devono fare.
- Se un messaggio non può essere instradato a nessuna coda (ad esempio, perché non ci sono vincoli per lo scambio in cui è stato pubblicato), viene eliminato o restituito al publisher , a seconda degli attributi del messaggio impostati dal publisher.

Consumatori

- Nel modello AMQP, ci sono due modi in cui le applicazioni possono eseguire consumare i messaggi in coda:
 - Ricevi messaggi recapitati ("API push")
 - Recupera i messaggi secondo necessità ("API pull")
- Con "API push", le applicazioni devono indicare l'interesse a consumare messaggi da una determinata coda. Quando lo fanno, diciamo che *registrano un consumatore* o, semplicemente, si *iscrivono a una coda*. È possibile avere più di un consumatore per coda o registrare un *consumatore esclusivo* (esclude tutti gli altri consumatori dalla coda mentre sta consumando).
- Ogni consumatore possiede un identificatore chiamato *tag consumer*. Può essere utilizzato per annullare l'iscrizione ai messaggi. I tag consumer sono solo stringhe.

Acknowledgement

- I consumer, ovvero le applicazioni che ricevono ed elaborano i messaggi, possono occasionalmente non riuscire a elaborare i singoli messaggi o a volte si arrestano in modo anomalo.
- Ciò solleva una domanda: quando il broker dovrebbe rimuovere i messaggi dalle code?
- La specifica AMQP offre ai consumatori due modalità di avvenuta consegna:
 - dopo che il broker invia un messaggio a un'applicazione (usando il metodo `basic.deliver` o `basic.get-ok`).
 - dopo che l'applicazione invia un riconoscimento (usando il metodo `basic.ack`).
- **La prima scelta si chiama modello di riconoscimento automatico, mentre la seconda si chiama modello di riconoscimento esplicito.**

Acknowledgement

- Con il modello esplicito l'applicazione sceglie quando è il momento di inviare un riconoscimento (ack). Può accadere subito dopo aver ricevuto un messaggio o dopo averlo memorizzato in un archivio dati prima dell'elaborazione o dopo aver elaborato completamente il messaggio (ad esempio, recuperare correttamente una pagina Web, elaborarlo e archivarlo in un archivio dati persistente).
- Se un consumatore muore senza inviare un ack, il broker lo riconsegnerà a un altro consumatore o, se nessuno è disponibile al momento, il broker attenderà fino a quando almeno un consumatore non sarà registrato per la stessa coda prima di tentare la riconsegna.

Rifiutare i messaggi

- Quando un'applicazione del consumatore riceve un messaggio, l'elaborazione di quel messaggio potrebbe non riuscire.
- Un'applicazione può indicare al broker che l'elaborazione dei messaggi non è riuscita (o non può essere eseguita in quel momento) rifiutando un messaggio.
- Quando si rifiuta un messaggio, un'applicazione può chiedere al broker di scartarlo o sostituirlo di nuovo.

Quando è presente un solo consumatore su una coda, assicurarsi di non creare infiniti cicli di consegna dei messaggi rifiutando e rielaborando ripetutamente un messaggio dallo stesso consumatore.

Negative Acknowledgement (nack)

- I messaggi vengono rifiutati con il metodo `basic.reject` .
- C'è una limitazione che ha `basic.reject`: **non c'è modo di rifiutare più messaggi come è possibile fare con gli `ack`.**
- Tuttavia, se si utilizza RabbitMQ, esiste una soluzione: fornisce un'estensione AMQP nota come *negative acknowledgement* o *nacks* .

Prefetching Messages

- Per i casi in cui più consumatori condividono una coda, è utile poter specificare quanti messaggi possono essere inviati prima di inviare il successivo ack.
- Questo può essere usato come una semplice tecnica di bilanciamento del carico o per migliorare il throughput se i messaggi tendono ad essere pubblicati in batch.
 - Ad esempio, se un'applicazione di produzione invia messaggi ogni minuto a causa della natura del lavoro che sta svolgendo.
- Si noti che RabbitMQ supporta solo il prefetch-count a livello di canale, non il prefetching basato sulla connessione o sulla dimensione.

Attributi e payload del messaggio

- I messaggi nel modello AMQP hanno *attributi*.
- Alcuni attributi sono utilizzati dai broker AMQP, ma la maggior parte sono aperti all'interpretazione da parte delle applicazioni che li ricevono.
- Alcuni attributi sono facoltativi e noti come *header*. Sono simili alle intestazioni X in HTTP. Gli attributi del messaggio vengono impostati quando viene pubblicato un messaggio.
- I messaggi hanno anche un *payload* (i dati che trasportano), che i broker AMQP considerano un array di byte opaco. Il broker non ispezionerà né modificherà il payload.
- È possibile che i messaggi contengano solo attributi e nessun payload.

Persistenza dei messaggi

- I messaggi possono essere pubblicati come persistenti, il che rende il broker persistente su disco.
- Se il server viene riavviato, il sistema garantisce che i messaggi persistenti ricevuti non vengano persi.
- La semplice pubblicazione di un messaggio in un exchange durevole o il fatto che le code a cui viene instradato siano durevoli non rendono persistente un messaggio: tutto dipende dalla modalità di persistenza del messaggio stesso.
- La pubblicazione dei messaggi come persistente influisce sulle prestazioni (proprio come con gli archivi di dati, la durata ha un certo costo in termini di prestazioni)

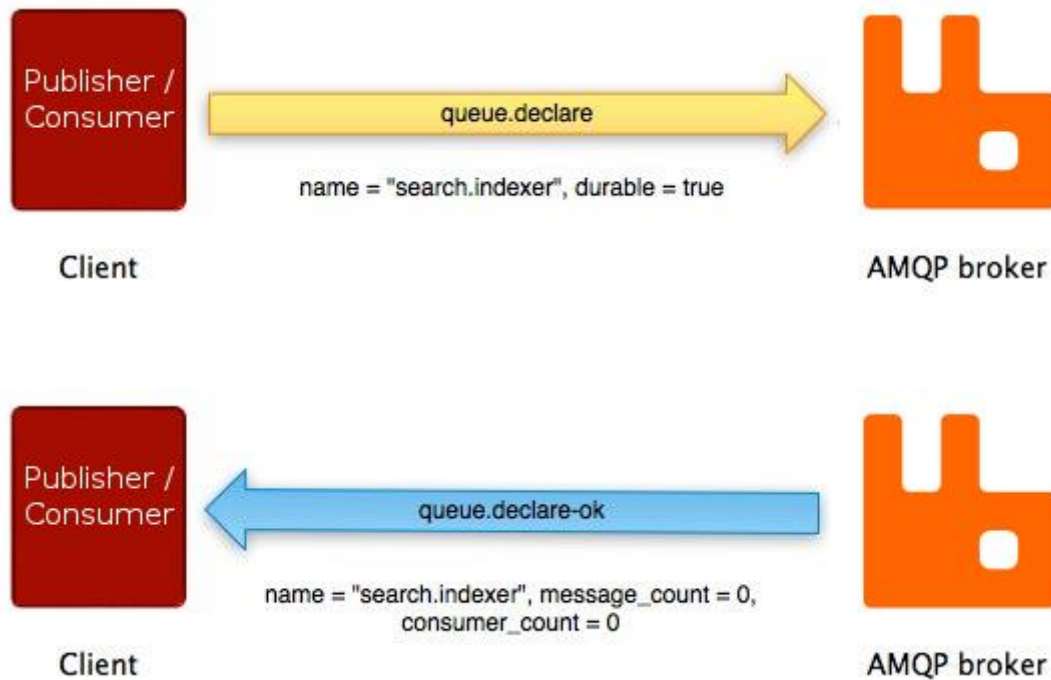
Metodi di AMQP

- AMQP è strutturato con un numero di *metodi*.
- I metodi sono operazioni (come i metodi HTTP) e non hanno nulla in comune con i metodi nei linguaggi di programmazione orientati agli oggetti.
- I metodi di protocollo in AMQP sono raggruppati in *classi*.
- Le classi sono solo raggruppamenti logici di metodi AMQP.
- Per esempio negli exchange:
 - exchange.declare
 - exchange.declare-ok
 - exchange.delete
 - exchange.delete-ok

Esempi



Esempi



Connessioni

- Le connessioni AMQP sono in genere di lunga durata. AMQP è un protocollo a livello di applicazione che utilizza TCP per una consegna affidabile.
- Le connessioni utilizzano l'autenticazione e possono essere protette tramite TLS.
- **ATTENZIONE:** quando un'applicazione non deve più essere connessa al server, dovrebbe chiudere la sua connessione AMQP invece di chiudere bruscamente la connessione TCP sottostante.

Ecosistema

- Esistono molti client AMQP per molti linguaggi e piattaforme di programmazione popolari. Alcuni di essi seguono da vicino la terminologia AMQP e forniscono solo l'implementazione dei metodi AMQP. Alcuni altri hanno funzionalità aggiuntive, metodi di convenienza e astrazioni. Alcuni client sono asincroni (non bloccanti), altri sono sincroni (bloccanti), altri supportano entrambi i modelli. Alcuni client supportano estensioni specifiche del fornitore (ad esempio, estensioni specifiche di RabbitMQ).
- Poiché uno dei principali obiettivi di AMQP è l'interoperabilità, è una buona idea per gli sviluppatori comprendere le operazioni del protocollo e non limitarsi alla terminologia di una particolare libreria client. In questo modo comunicare con gli sviluppatori utilizzando librerie diverse sarà notevolmente più semplice.