

Programma

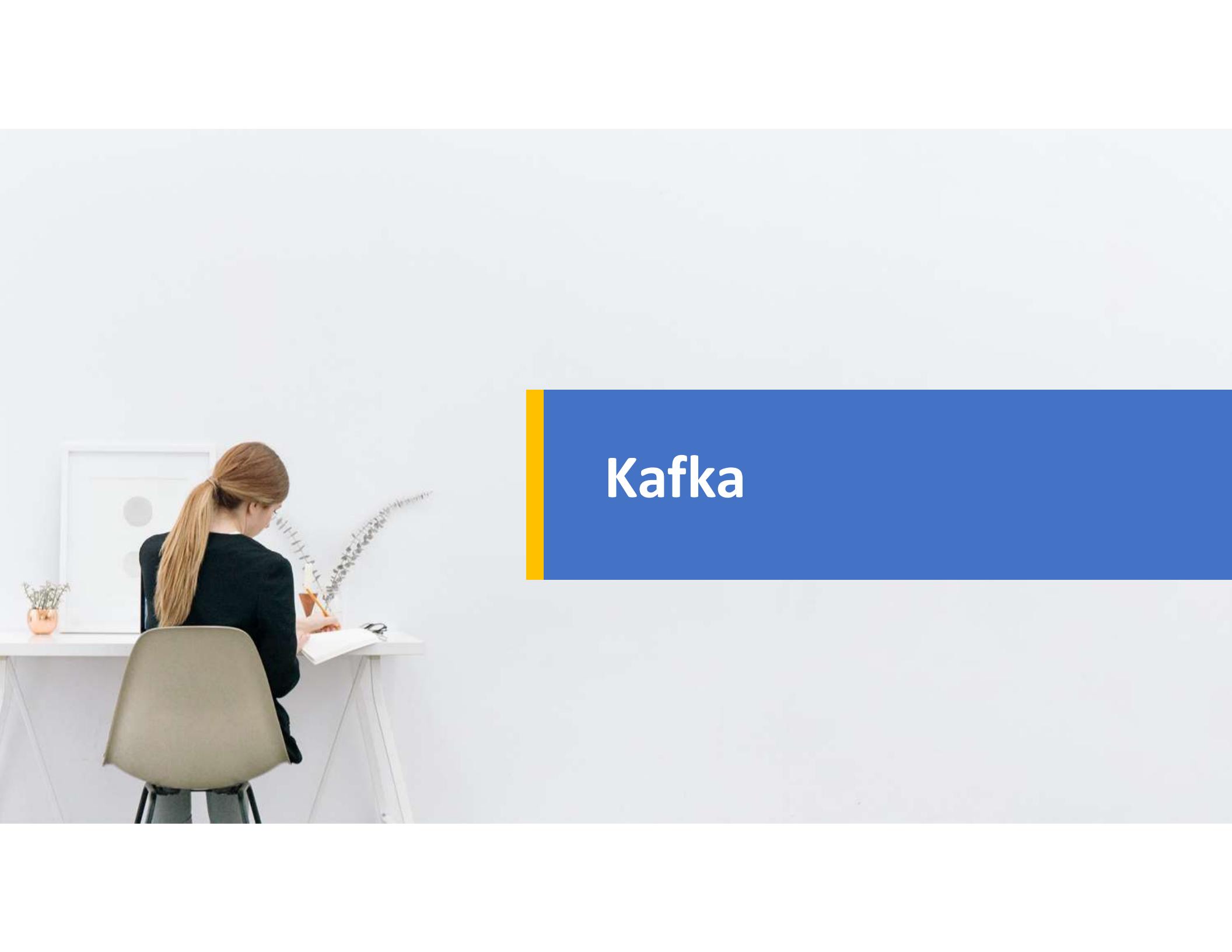


01 Introduzione a Kafka

02 Installazione Kafka

03 Produttore

04 Consumatore

A photograph of a woman with long blonde hair tied back, wearing a black top, sitting at a white desk. She is facing away from the camera, writing in a small notebook with a pen. On the desk, there is a small potted plant in a copper-colored pot, a pair of scissors, and some dried branches. In the background, there is a framed abstract painting on the wall and a small vase with flowers on a shelf.

Kafka



Kafka

Che cos'è?

Capiamone le basi

- Prima di discutere i dettagli di Apache **Kafka**, è importante comprendere il concetto di messaggistica publisher/subscriber e il motivo per cui è importante.
- Un sistema Publish-Subscribe è essenzialmente costituito da un insieme di client che scambiano eventi tra di loro in modo asincrono comportandosi, secondo il caso, da **produttori** o **consumatori** di informazione.
- Un'entità logica, chiamata event notification service (ENS), agisce come mezzo di interconnessione, garantendo il disaccoppiamento tra i publisher ed i subscriber.



Kafka

Che cos'è?

Capiamone le basi

- Apache Kafka è una piattaforma di streaming di dati open source che consente a un gran numero di applicazioni di elaborare e analizzare i dati in tempo reale.
- La sua architettura è basata su un modello **publisher-subscriber**, dove i produttori pubblicano messaggi su un determinato topic e i consumatori si iscrivono a quel topic per ricevere i messaggi.
- Kafka è composto da diversi componenti principali, tra cui il broker, il topic, il producer e il consumer.



Kafka

Che cos'è?

Capiamone le basi

- In Kafka, l'unità di base è il messaggio.
- Un messaggio è semplicemente un array di byte per Kafka, il che significa che i dati contenuti al suo interno non hanno un formato o un significato specifico per Kafka.
- Un messaggio può contenere un bit di metadati opzionale, chiamato chiave.
- Le chiavi vengono utilizzate quando i messaggi devono essere scritti in specifiche partizioni.



Kafka

Che cos'è?

Messaggi e batch

- Per efficienza, i messaggi vengono scritti in Kafka in batch.
- Un batch è semplicemente una raccolta di messaggi, tutti i quali vengono prodotti per lo stesso topic e partizione.
- Naturalmente, l'utilizzo dei batch è un compromesso tra latenza e throughput: l'aumento delle dimensioni dei batch consente di gestire più messaggi per unità di tempo, ma aumenta il tempo necessario per la propagazione di un singolo messaggio.



Kafka

Che cos'è?

Schemi

- Ci sono molte opzioni disponibili per lo schema dei messaggi, a seconda delle esigenze specifiche dell'applicazione.
- Sistemi semplici, come JSON (JavaScript Object Notation) e XML (Extensible Markup Language), sono facili da usare e leggibili dall'uomo.
- Tuttavia, mancano di funzionalità come la gestione robusta dei tipi e la compatibilità tra le diverse versioni dello schema.
- Molti sviluppatori di Kafka preferiscono l'uso di Apache Avro, che è un framework di serializzazione originariamente sviluppato per Hadoop.

Json vs Avro

```
{  
    "persona": {  
        "nome": "Mario",  
        "cognome": "Rossi",  
        "età": 30,  
        "indirizzo": {  
            "via": "Via Roma, 123",  
            "città": "Roma",  
            "CAP": "00100"  
        },  
        "contatti": [  
            {  
                "tipo": "email",  
                "valore": "mario.rossi@example.com"  
            },  
            {  
                "tipo": "telefono",  
                "valore": "+39 123456789"  
            }  
        ]  
    }  
}
```

```
{  
    "type": "record",  
    "name": "Persona",  
    "fields": [  
        {"name": "nome", "type": "string"},  
        {"name": "cognome", "type": "string"},  
        {"name": "età", "type": "int"},  
        {  
            "name": "indirizzo",  
            "type": {  
                "type": "record",  
                "name": "Indirizzo",  
                "fields": [  
                    {"name": "via", "type": "string"},  
                    {"name": "città", "type": "string"},  
                    {"name": "CAP", "type": "string"}  
                ]  
            }  
        },  
        {  
            "name": "contatti",  
            "type": {  
                "type": "array",  
                "items": {  
                    "type": "record",  
                    "name": "Contatto",  
                    "fields": [  
                        {"name": "tipo", "type": "string"},  
                        {"name": "valore", "type": "string"}  
                    ]  
                }  
            }  
        }  
    ]  
}
```



Kafka

Che cos'è?

Schemi

- Un formato dati consistente è importante in Kafka, poiché consente di gestire meglio la trasmissione dei messaggi.
- Ecco alcune ragioni per cui un formato di dati consistente è importante in Kafka:
 - **Interoperabilità.**
 - **Affidabilità del Consumatore.**
 - **Evolutività del Sistema.**
 - **Debugging e Monitoraggio**
 - **Scalabilità.**



Kafka

Che cos'è?

Topic e partizioni

- I messaggi in Kafka sono categorizzati in topic.
- I topic sono suddivisi in un certo numero di partizioni.
- In Kafka, una partizione è una porzione di un topic. I topic sono categorie logiche che raggruppano i messaggi all'interno di Kafka. Ogni partizione di un topic è un registro di eventi ordinati e immutabili.
- Ogni messaggio in una partizione ha un offset univoco, che rappresenta la sua posizione all'interno della partizione.

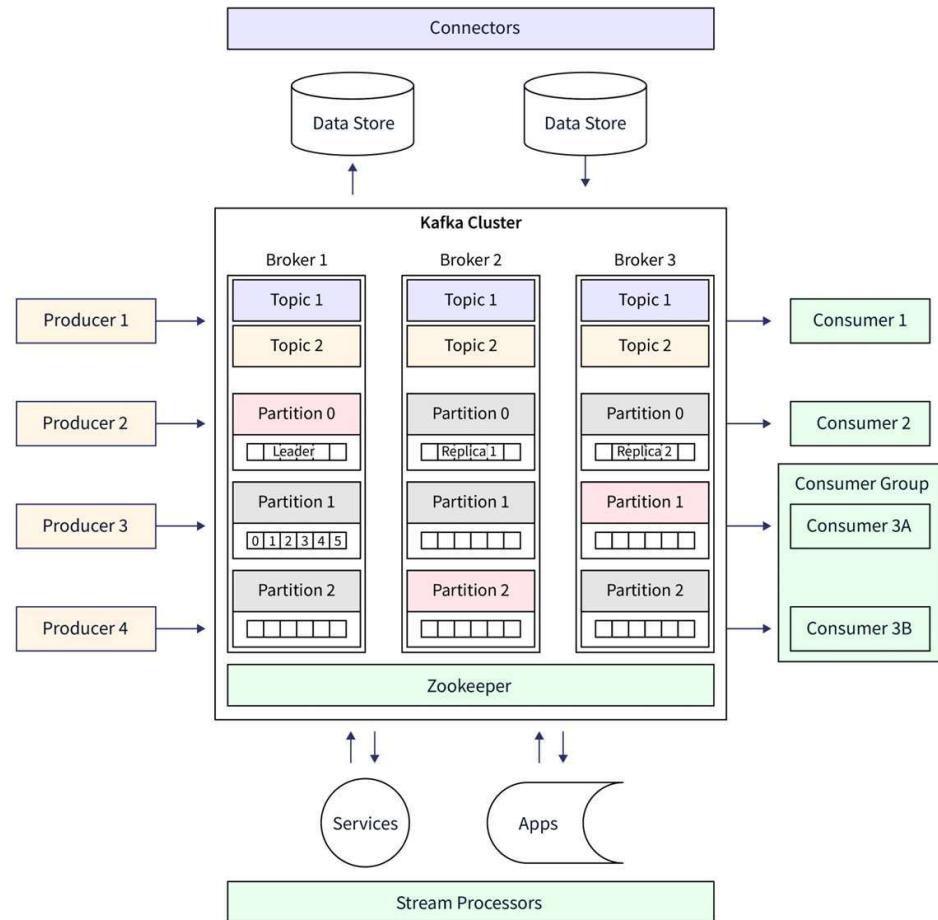


Kafka

Che cos'è?

[Topic e partizioni](#)

- I messaggi vengono scritti in esso in modo append-only e vengono letti in ordine dall'inizio alla fine.
- È importante notare che, poiché un topic ha tipicamente più partizioni, non c'è alcuna garanzia sull'ordinamento temporale dei messaggi sull'intero topic, ma solo all'interno di una singola partizione.
- Le partizioni sono anche il modo in cui Kafka fornisce ridondanza e scalabilità. Ciascuna partizione può essere ospitata su un server diverso, il che significa che un singolo topic può essere scalato in modo orizzontale su più server per fornire prestazioni ben oltre le capacità di un singolo server.





Kafka

Che cos'è?

Producer e Consumer

- I client di Kafka sono gli utenti del sistema e ci sono due tipi fondamentali: **produttori**(Producer) e **consumatori**(Consumer).
- I produttori creano nuovi messaggi.
- In generale, un messaggio verrà prodotto per un topic specifico.
- Per impostazione predefinita, al produttore non interessa a quale partizione venga scritto un messaggio specifico.



Kafka

Che cos'è?

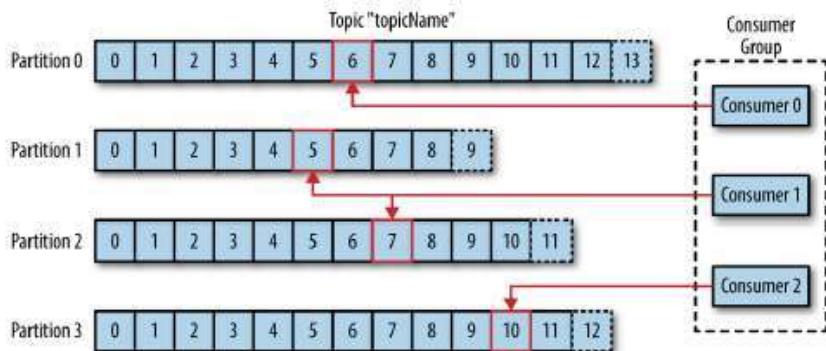
Producer e Consumer

- I consumatori(Consumer) leggono i messaggi.
- Il consumatore si sottoscrive a uno o più topic e legge i messaggi nell'ordine in cui sono stati prodotti.
- Il consumatore tiene traccia dei messaggi che ha già consumato tenendo traccia dell'offset degli incrementi che Kafka aggiunge a ciascun messaggio durante la produzione.
- Ogni messaggio in una data partizione ha un **offset univoco**.

Kafka

Che cos'è?

Producer e Consumer

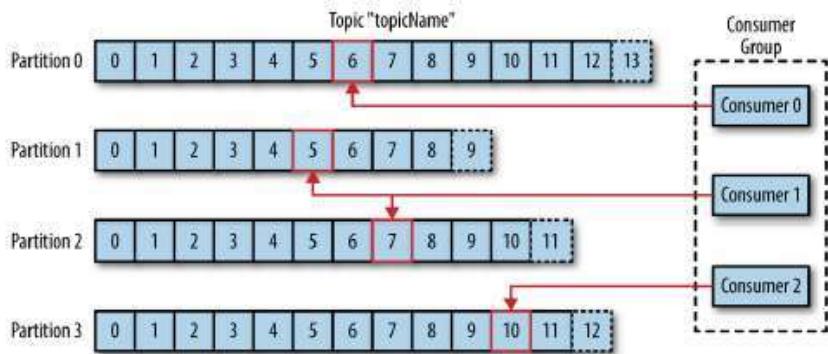


- Memorizzando l'offset dell'ultimo messaggio consumato per ciascuna partizione, sia in Zookeeper che in Kafka stesso, un consumatore può interrompersi e riprendere senza perdere la propria posizione.
- I consumatori lavorano come parte di un gruppo di consumatori, che è costituito da uno o più consumatori che lavorano insieme per consumare un argomento.

Kafka

Che cos'è?

Broker e Cluster



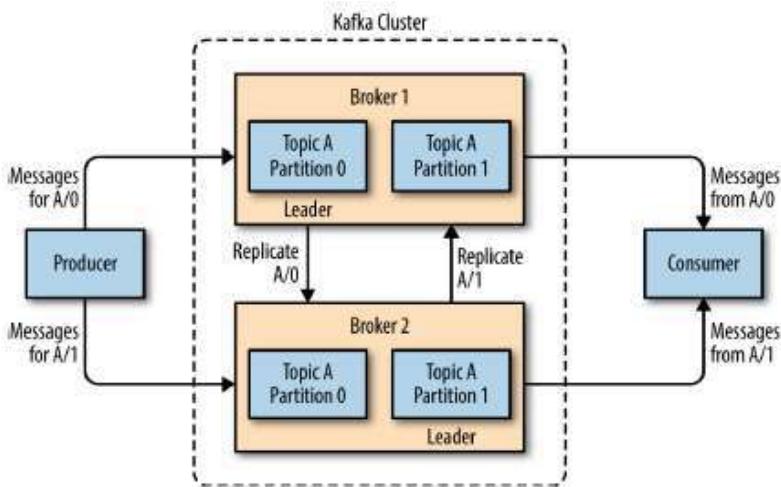
- Un singolo nodo Kafka è chiamato **broker**. Il broker riceve messaggi dai produttori, assegna loro degli offset e li archivia su disco. Si occupa anche dei consumatori, rispondendo alle richieste di recupero per le partizioni e fornendo i messaggi che sono stati archiviati su disco.
- I broker Kafka sono progettati per operare come parte di un cluster. All'interno di un cluster di broker, un broker fungerà anche da controller del cluster (eletto automaticamente tra i membri attivi del cluster).

Kafka

Che cos'è?

Broker e Cluster

- Il controller è responsabile delle operazioni amministrative, comprese l'assegnazione delle partizioni ai broker e il monitoraggio dei fallimenti dei broker.
- Una partizione è di proprietà di un singolo broker nel cluster e tale broker è chiamato il leader della partizione.
- Una partizione può essere assegnata a più broker, il che comporterà la replica della partizione.

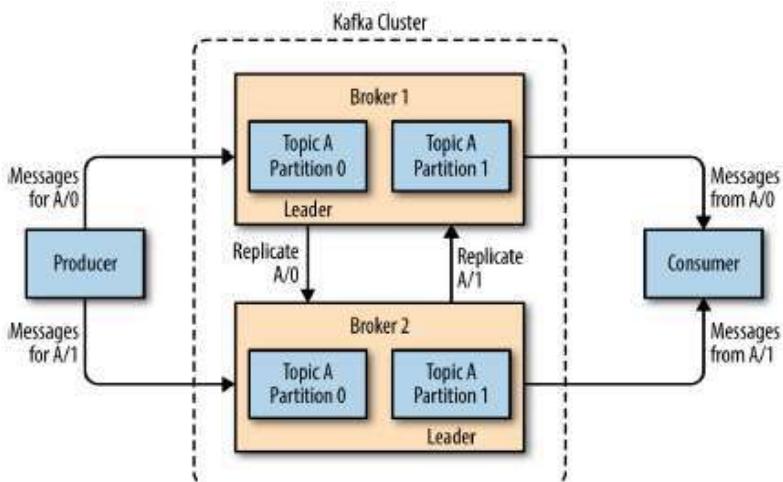


Kafka

Che cos'è?

Broker e Cluster

- Una caratteristica chiave di Apache Kafka è quella della retention, ovvero la conservazione duratura dei messaggi per un certo periodo di tempo.
- I broker di Kafka sono configurati con un'impostazione di retention predefinita per i topic, conservando i messaggi per un certo periodo (ad esempio, 7 giorni) o fino a quando l'argomento raggiunge una determinata dimensione in byte (ad esempio, 1 GB).
- Una volta raggiunti questi limiti, i messaggi vengono cancellati in modo che la configurazione di retention rappresenti la quantità minima di dati disponibile in qualsiasi momento.





Kafka

Perché utilizzare Kafka?

Funzionalità

- **Produttori Multipli:** Kafka è in grado di gestire senza problemi più produttori, che utilizzino molteplici topic o lo stesso topic. Ciò rende il sistema ideale per aggregare dati da molti sistemi FrontEnd/BackEnd e renderli consistenti.
- **Consumatori Multipli:** Oltre a supportare più produttori, Kafka è progettato per consentire a più consumatori di leggere un singolo flusso di messaggi senza interferire tra loro.
- **Mantenimento su disco:** Non solo Kafka può gestire più consumatori, ma la conservazione duratura dei messaggi significa che i consumatori non devono sempre lavorare in tempo reale. I messaggi vengono registrati su disco e saranno archiviati con regole di retention configurabili.

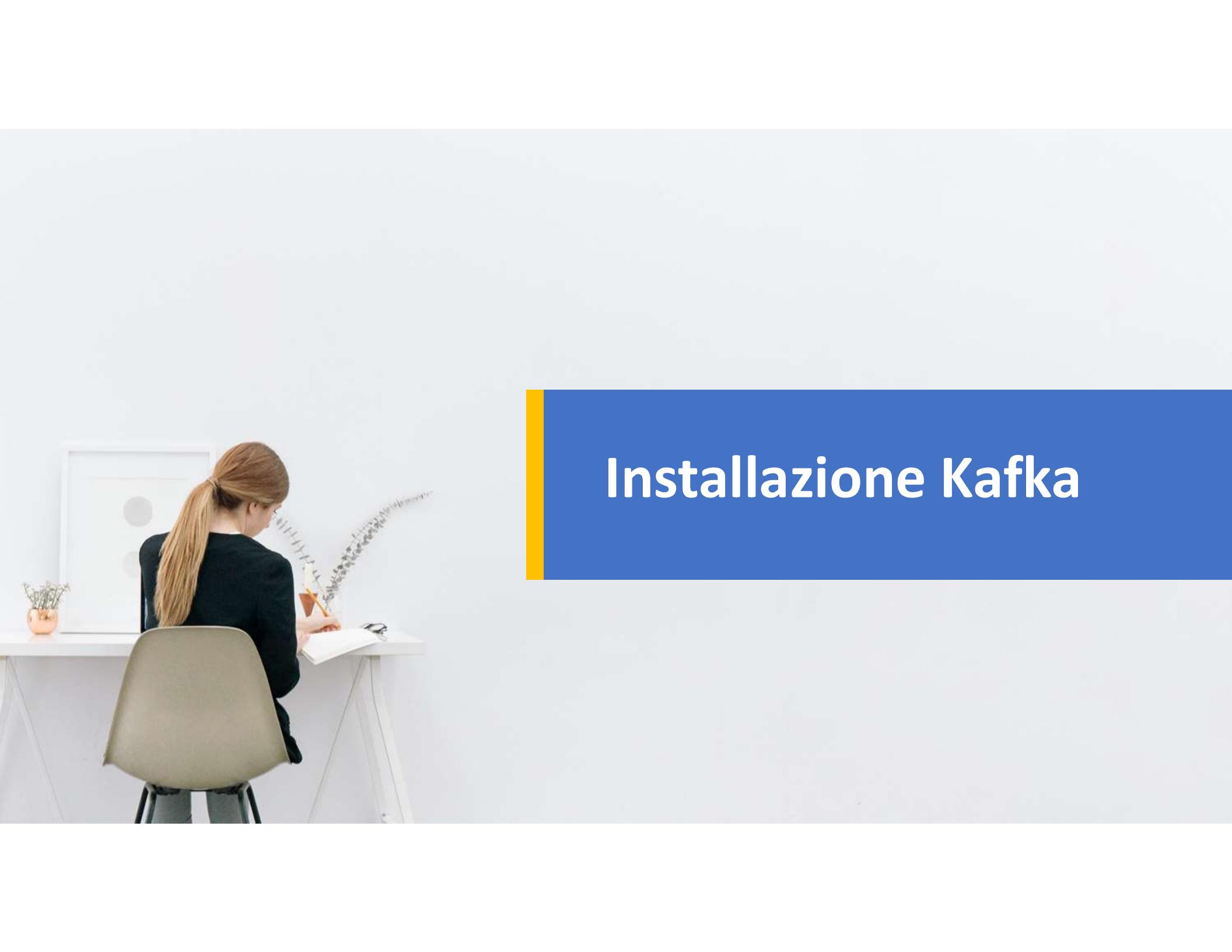


Kafka

Perché utilizzare Kafka?

Funzionalità

- **Scalabilità:** La scalabilità flessibile di Kafka facilita la gestione di qualsiasi quantità di dati.
- **Alte performance:** Tutte queste caratteristiche si uniscono per rendere Apache Kafka un sistema di messaggistica publish/subscribe con eccellenti prestazioni sotto carichi elevati. Produttori, consumatori e broker possono essere tutti scalati per gestire flussi di messaggi molto ampi con facilità.

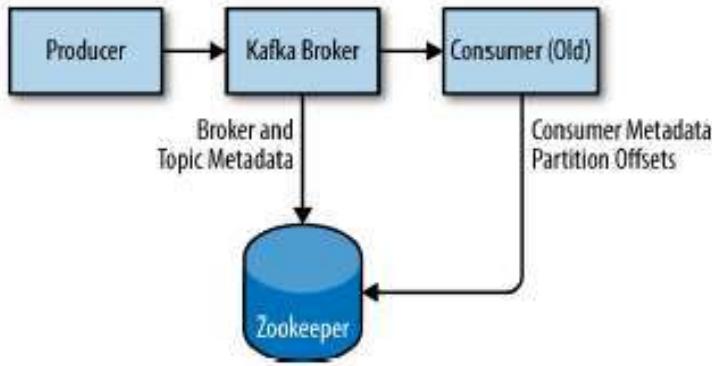
A photograph of a woman with long blonde hair tied back, wearing a black top, sitting at a white desk. She is facing away from the camera, writing in a small notebook with a pen. On the desk in front of her is a small potted plant in a copper-colored pot. In the background, there is a framed abstract painting on the wall and a small white shelf. To the right of the image, there is a solid blue vertical bar with a thin yellow vertical bar next to it.

Installazione Kafka

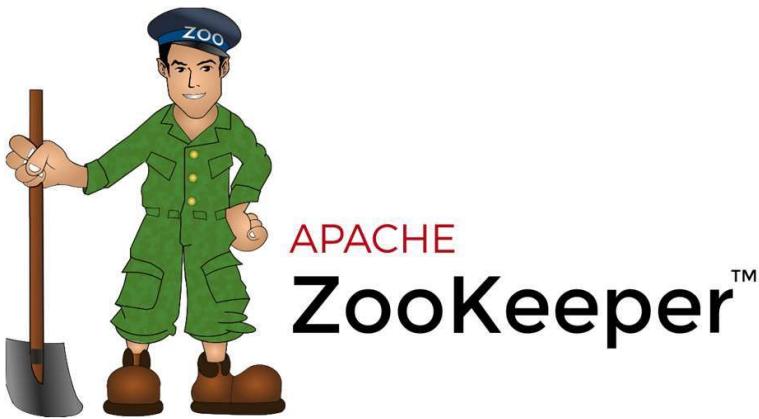
Installazione Kafka

Come procedere?

Prerequisiti



- **Scegliere un sistema operativo:** Apache Kafka è un'applicazione Java e può essere eseguita su molti sistemi operativi, tra cui Windows, MacOS, Linux e altri. Linux è il sistema operativo consigliato per distribuire Kafka per un utilizzo generale.
- **Java:** Java è un linguaggio di programmazione ampiamente utilizzato per codificare le applicazioni Web. Per oltre vent'anni è stata una scelta molto diffusa tra gli sviluppatori, tanto che a oggi vengono utilizzate milioni di applicazioni Java. Java è un linguaggio multi-piattaforma, orientato agli oggetti. Si necessita versione di Java ≥ 8
- **Zookeeper:** Apache Kafka utilizza Zookeeper per memorizzare i metadati sul cluster Kafka, nonché i dettagli dei client consumer, come mostrato nella figura sulla sinistra.

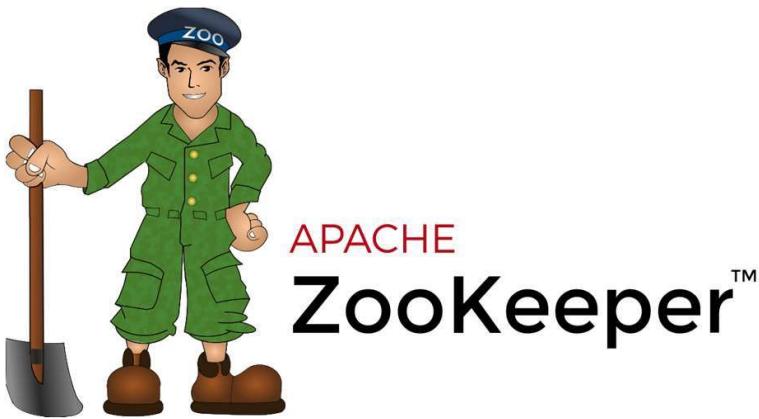


Installazione Kafka

Cos'è ZooKeeper

A cosa serve?

- **ZooKeeper** è utilizzato da diversi progetti open-source per fornire un piano di controllo altamente affidabile per la coordinazione distribuita di applicazioni cluster attraverso uno store gerarchico di chiavi e valori.
- La serie di servizi offerti da ZooKeeper include servizi di configurazione distribuita, servizi di sincronizzazione, servizi di elezione di leadership e un registro di denominazione.
- Un concetto fondamentale di ZooKeeper è lo **znode**.

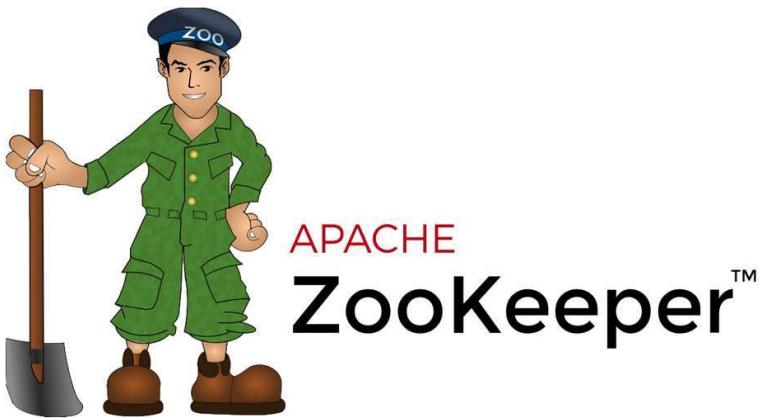


Installazione Kafka

Cos'è ZooKeeper

zNode

- Uno znode di ZooKeeper è un nodo dati che è tracciato da una struttura di stato che include modifiche dei dati, modifiche delle ACL, un numero di versione e un timestamp.
- Il numero di versione, insieme al timestamp, consente a ZooKeeper di coordinare gli aggiornamenti e memorizzare informazioni nella cache.
- Ogni modifica ai dati di uno znode comporta una modifica della versione, che viene utilizzata per assicurarsi che le modifiche allo znode siano applicate correttamente.

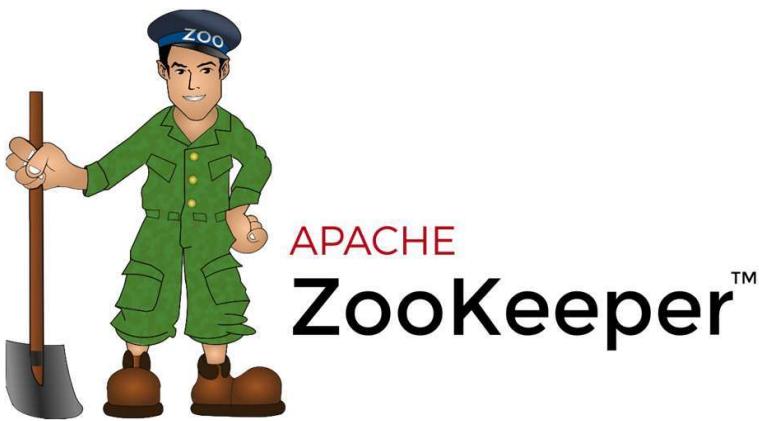


Installazione Kafka

Cos'è ZooKeeper

[Come lavorano Kafka e ZooKeeper?](#)

- Kafka e ZooKeeper lavorano in collaborazione per formare un cluster Kafka completo, con ZooKeeper che fornisce i suddetti servizi di clustering distribuito, mentre Kafka gestisce i flussi di dati effettivi e la connettività con i client.
- A un livello dettagliato, ZooKeeper si occupa dell'elezione della leadership dei broker Kafka e gestisce la topologia del cluster, in modo che ogni broker sappia quando i broker entrano o escono dal cluster, quando un broker fallisce e chi è il nodo leader preferito per una coppia di topic/partizione specifica.

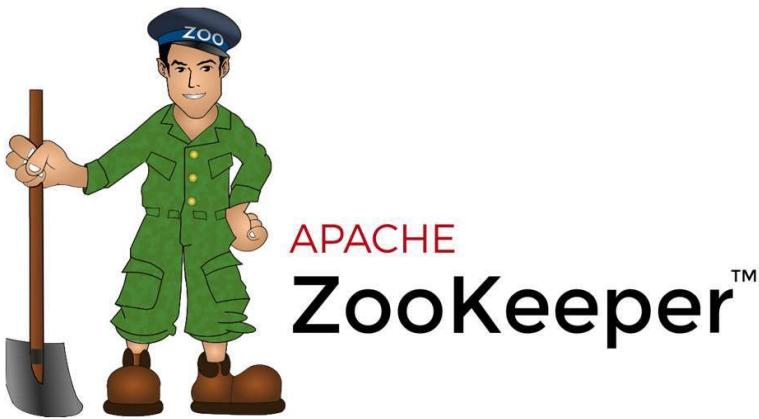


Installazione Kafka

Cos'è ZooKeeper

Funzionalità ZooKeeper

- **Elezione del controllore:** Il controller di Kafka è un broker eletto assegnato da ZooKeeper, responsabile della gestione delle partizioni, della leadership delle coppie partizione-log e delle repliche, oltre a svolgere mansioni generali di manutenzione del cluster, come assegnazioni delle partizioni e la gestione di una lista di repliche in sincronia (ISR).
- **ACL:** Apache Kafka è dotato di un autorizzatore integrato che sfrutta ZooKeeper per memorizzare liste di controllo degli accessi o ACL. Le ACL sono un modo conveniente per gestire l'accesso alle risorse di Kafka, come i topic e i gruppi di consumatori.



Installazione Kafka

Cos'è ZooKeeper

Funzionalità ZooKeeper

- **Gestione risorse (quote)** : I broker di Kafka hanno la capacità di controllare le risorse del broker utilizzate dai client attraverso le quote.

Kafka implementa due principali tipi di quote per i client a questo scopo.

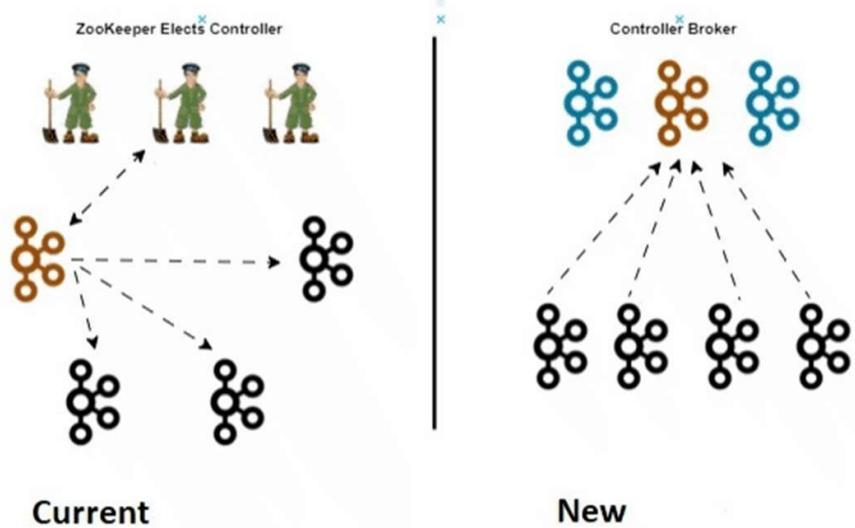
Le quote sulla larghezza di banda di rete sono definite da una soglia di byte-rate e una quota sulla frequenza delle richieste che è definita da una soglia di utilizzo della CPU come percentuale dei thread di I/O di rete.

Installazione Kafka

Perché utilizzare Kafka?

Ma Kafka senza ZooKeeper?

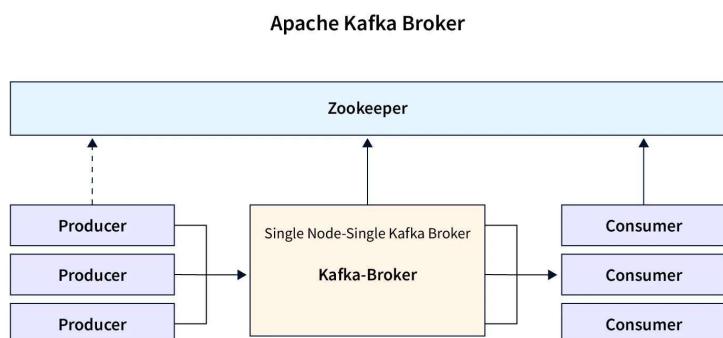
- Con le versioni più recenti di Kafka, è stato introdotto il KIP-500 (o "ZooKeeper Removal") nelle versioni principali.
- Tuttavia, è ancora considerato un elemento in "accesso anticipato" e non è pronto per la produzione a causa di alcuni problemi persistenti. Al momento, non è ancora supportato l'aggiornamento di un cluster basato su ZooKeeper, e manca ancora il supporto per alcune configurazioni dei topic.
- Tenendo presente ciò, ZooKeeper continuerà ad essere rilevante per i cluster di produzione per un po' di tempo.



Installazione Kafka

Perché utilizzare Kafka?

Broker Configuration?



- **Broker.id:** Ogni broker di Kafka deve avere un identificatore intero, che viene impostato utilizzando la configurazione broker.id. Per impostazione predefinita, questo intero è impostato su 0, ma può essere qualsiasi valore. **NB: Il valore deve essere univoco all'interno del cluster Kafka altrimenti non sarà possibile gestire il cluster.**
- **Porta:** Il file di configurazione di default avvia Kafka con un listener sulla porta **TCP 9092**. Questo può essere impostato su qualsiasi porta disponibile cambiando il parametro di configurazione della porta.

Installazione Kafka

Perché utilizzare Kafka?

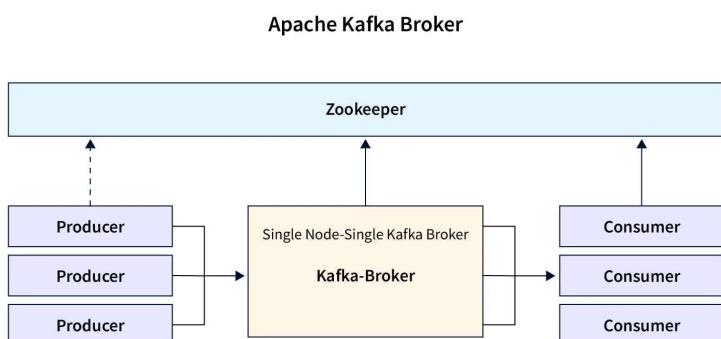
Broker Configuration?

- **Zookeeper.connect:** La posizione di Zookeeper utilizzata per memorizzare i metadati del broker è impostata utilizzando il parametro di configurazione **zookeeper.connect**.

La configurazione di default utilizza un Zookeeper in esecuzione sulla porta 2181 sull'host locale, specificato come `localhost:2181`.

Il formato per questo parametro è una lista di stringhe delimitate da punto e virgola di `hostname:port/path`, che includono:

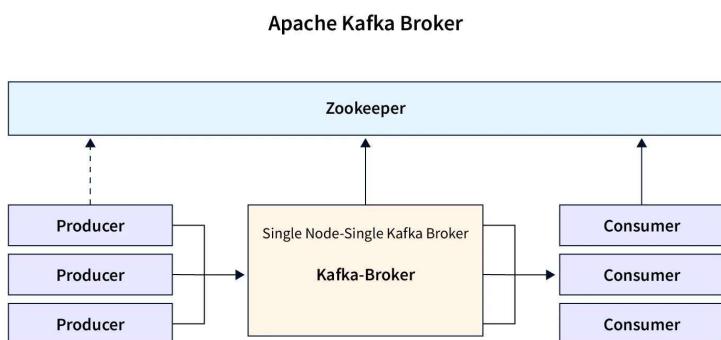
- ✓ `hostname`, il nome host o l'indirizzo IP del server Zookeeper.
- ✓ `port`, il numero di porta del client per il server.



Installazione Kafka

Perché utilizzare Kafka?

Broker Configuration?



- **Log.dirs:** Kafka persiste tutti i messaggi su disco e questi segmenti di log vengono memorizzati nelle directory specificate nella configurazione **log.dirs**.
- **Log.retention:** La configurazione più comune per la durata della conservazione dei messaggi in Kafka è quella basata sul tempo. Il valore predefinito è specificato nel file di configurazione utilizzando il parametro `log.retention.hours` ed è impostato su 168 ore, ovvero una settimana.
- **Log.retention.bytes:** Questo valore è impostato utilizzando il parametro `log.retention.bytes` e viene applicato per partizione. Ciò significa che se si ha un topic con 8 partizioni e `log.retention.bytes` è impostato su 1 GB, la quantità di dati conservati per il topic sarà al massimo di 8 GB.

Installazione Kafka

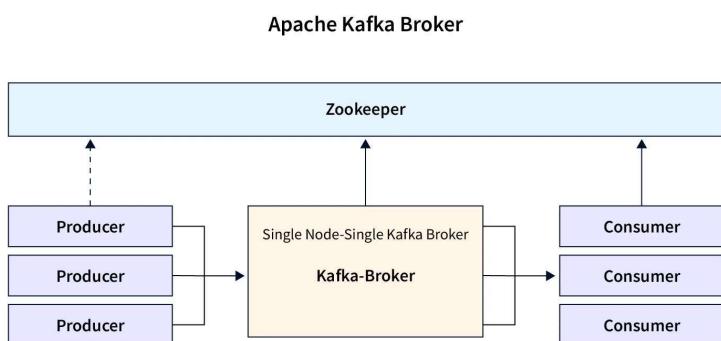
Perché utilizzare Kafka?

Broker Configuration?

- **Message.max.bytes:** Il broker Kafka limita la dimensione massima di un messaggio che può essere prodotto, configurata dal parametro `message.max.bytes`, che di default è impostato su 1000000, ovvero 1 MB. Un produttore che tenta di inviare un messaggio più grande di questo riceverà un errore dal broker, e il messaggio non verrà accettato.

Selezione Hardware

- Kafka stesso non ha requisiti rigidi per una configurazione hardware specifica e funzionerà senza problemi su qualsiasi sistema.
- Tuttavia, ci sono diversi fattori che contribuiranno alle prestazioni complessive: throughput e capacità del disco, memoria, rete e CPU.

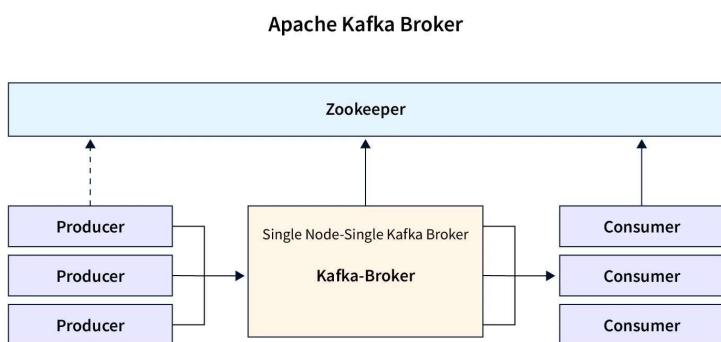


Installazione Kafka

Perché utilizzare Kafka?

Selezione Hardware

- **Disk Throughput:** Le prestazioni dei client producer saranno influenzate più direttamente dal throughput del disco del broker utilizzato per memorizzare i segmenti di log.
- I messaggi di Kafka devono essere memorizzati localmente quando vengono prodotti, e la maggior parte dei consumer attende almeno che almeno un broker confermi che i messaggi sono stati memorizzati prima di considerare l'invio riuscito. Ciò significa che scritture su disco più veloci equivalgono a una latenza di produzione inferiore.
- La decisione ovvia quando si tratta del throughput del disco è se utilizzare dischi rigidi tradizionali (HDD) o dischi a stato solido (SSD).

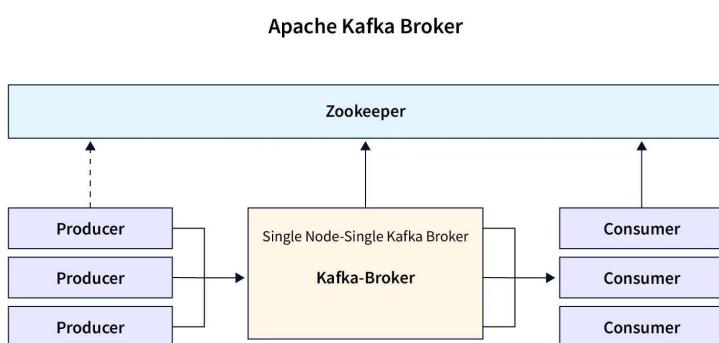


Installazione Kafka

Perché utilizzare Kafka?

Selezione Hardware

- **Disk Capacity:** La quantità di capacità di disco necessaria è determinata dal numero di messaggi che devono essere conservati in ogni momento. Se ci si aspetta che il broker riceva 1 TB di traffico al giorno, con una conservazione di 7 giorni, allora il broker avrà bisogno di almeno 7 TB di storage utilizzabile.
- **CPU:** La potenza di elaborazione non è così importante quanto il disco, ma influenzerà in qualche misura le prestazioni complessive del broker. Idealmente, i client dovrebbero comprimere i messaggi per ottimizzare l'utilizzo della rete e del disco.
Tuttavia, il broker Kafka deve decomprimere tutti i batch di messaggi per convalidare il checksum dei singoli messaggi e assegnare gli offset. Successivamente, deve ricomprimere il batch di messaggi per memorizzarlo su disco. È qui che proviene la maggior parte del requisito di potenza di elaborazione di Kafka.



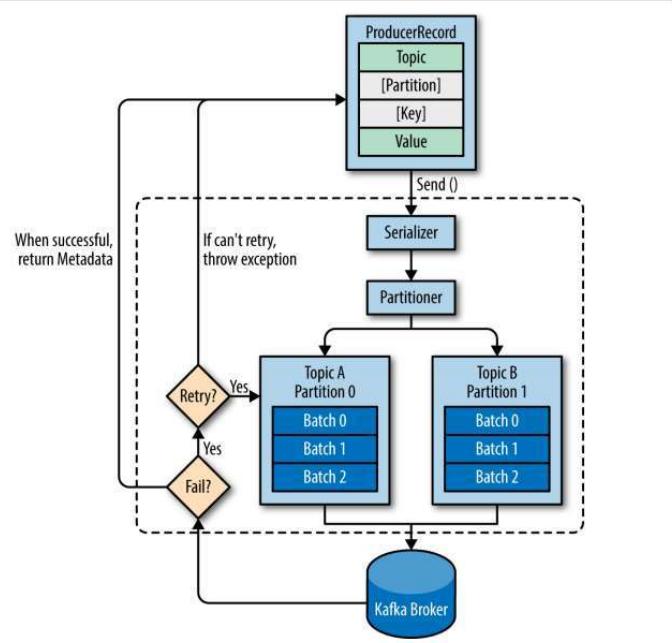
A photograph of a woman with long blonde hair tied back, wearing a black blazer, sitting at a white desk. She is facing away from the camera, writing in a small notebook with a pen. On the desk, there is a small potted plant in a copper-colored pot, a pair of scissors, and some dried botanical specimens. The background is a plain white wall.

Produttori (Producer)

Produttori

Kafka producer?

Iniziamo!



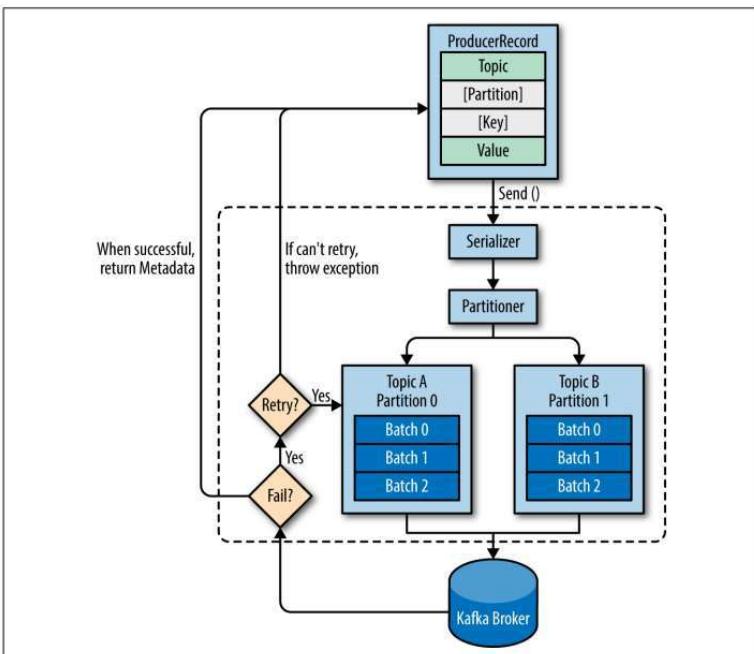
- Quando si inizia a produrre messaggi su kafka si utilizza un **ProducerRecord**, che deve includere il topic al quale si vuole inviare il record e un valore. Opzionalmente, è possibile anche specificare una chiave e/o una partizione.
- Successivamente, i dati vengono inviati a un **partizionatore**.
- Se è stato specificato una partizione nel ProducerRecord, il partizionatore non fa nulla e restituisce semplicemente la partizione che specificata. Se non è stato specificato il partizionatore allora, il partizionatore sceglierà una partizione automaticamente, di solito in base alla chiave del ProducerRecord.

Produttori

Kafka producer?

Iniziamo!

- Quando il broker riceve i messaggi, invia una risposta. Se i messaggi sono stati scritti con successo su Kafka, restituirà un oggetto RecordMetadata con il topic, la partizione e l'offset del record all'interno della partizione. Se il broker non è riuscito a scrivere i messaggi, restituirà un errore. Quando il produttore riceve un errore, può tentare di inviare il messaggio ancora qualche volta prima di rinunciare e restituire un errore.

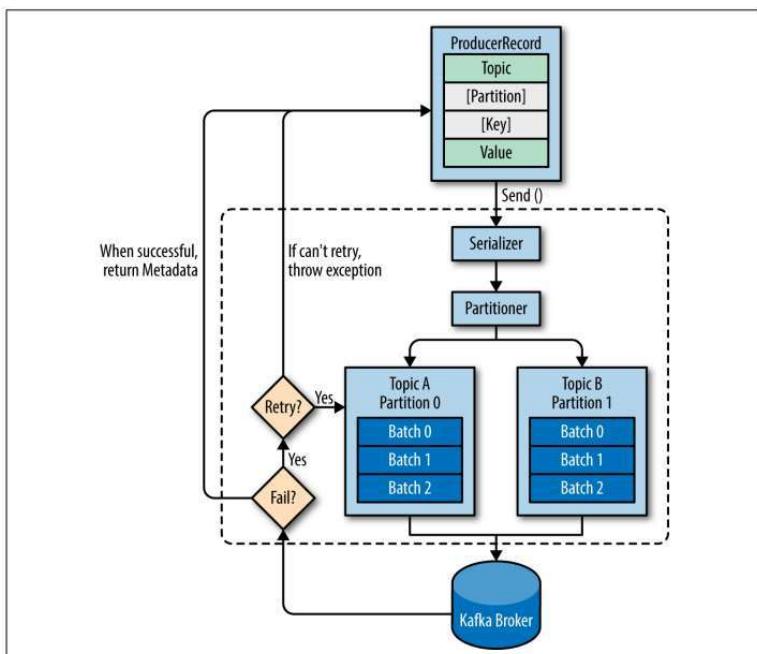


Produttori

Kafka producer?

Costruiamo un primo producer

- Il primo passo nella scrittura di messaggi su Kafka è creare un oggetto produttore con le proprietà che si desidera passare al produttore. Un produttore Kafka ha tre proprietà obbligatorie:
 - Bootstrap.servers**
 - Key.serializer**
 - Value.serializer**



Produttori

Kafka producer?

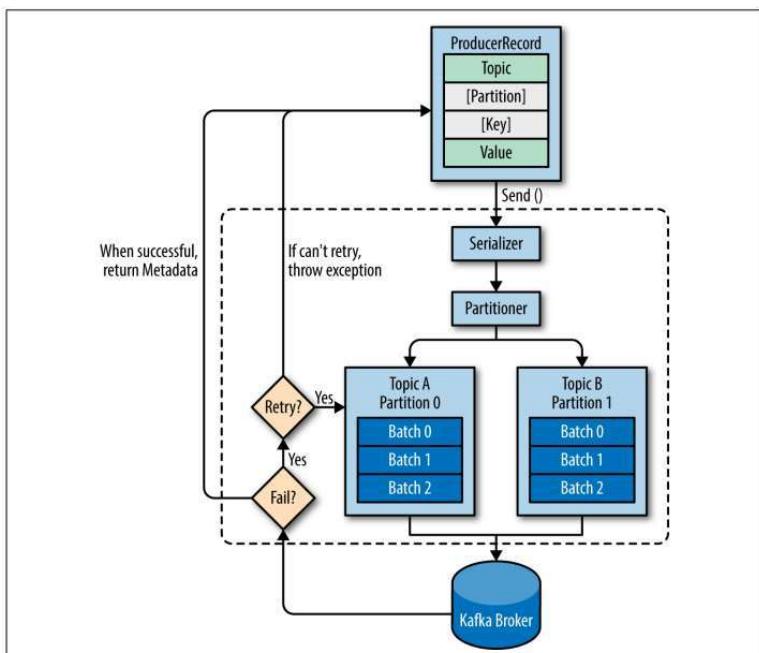
```
// create Producer properties
Properties properties = new Properties();
properties.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
properties.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
properties.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());

// create the producer
KafkaProducer<String, String> producer = new KafkaProducer<>(properties);
```

Produttori

Kafka producer?

Costruiamo un primo producer

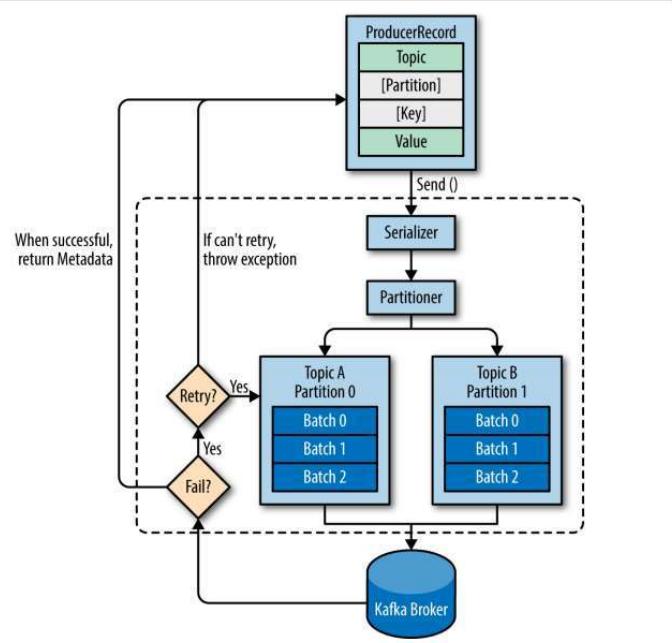


- Una volta istanziato un produttore, è il momento di iniziare a inviare messaggi. Ci sono tre metodi principali per inviare messaggi:
 - (**Fire-and-forget**): Viene inviato un messaggio al server e non ci interessa realmente se arriva con successo o meno. La maggior parte delle volte arriverà con successo, poiché Kafka è altamente disponibile e il produttore ritenterà automaticamente l'invio dei messaggi. Tuttavia, alcuni messaggi si perderanno utilizzando questo metodo.
 - (**Synchronous send**): Inviamo un messaggio, il metodo `send()` restituisce un oggetto `Future`, e utilizziamo `get()` per attendere il futuro e vedere se il `send()` è stato eseguito con successo o meno.
 - (**Asynchronous send**): Chiamiamo il metodo `send()` con una funzione di callback, che viene attivata quando riceve una risposta dal broker Kafka.

Produttori

Kafka producer?

Configurazioni aggiuntive



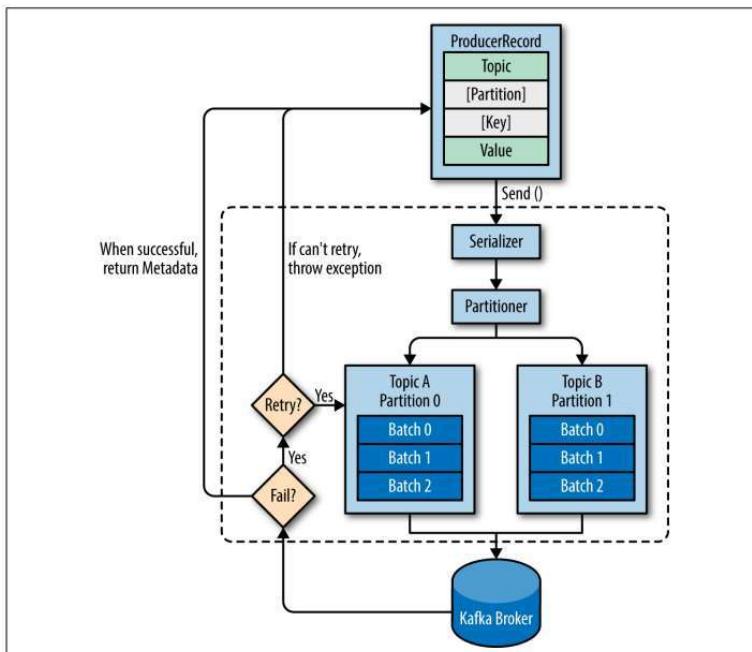
- **Buffer.memory:** Questo parametro imposta la quantità di memoria che il produttore utilizzerà per memorizzare i messaggi in attesa di essere inviati ai broker. Se i messaggi vengono inviati dall'applicazione più velocemente di quanto possano essere consegnati al server, il produttore potrebbe esaurire lo spazio e ulteriori chiamate `send()` bloccheranno o genereranno un'eccezione.
- **Compression.type:** Per impostazione predefinita, i messaggi vengono inviati non compressi. Questo parametro può essere impostato su snappy, gzip o lz4, in tal caso gli algoritmi di compressione corrispondenti verranno utilizzati per comprimere i dati prima di inviarli ai broker. Abilitando la compressione, si riduce l'utilizzo della rete e dello storage, che spesso è un collo di bottiglia nell'invio di messaggi a Kafka.

Produttori

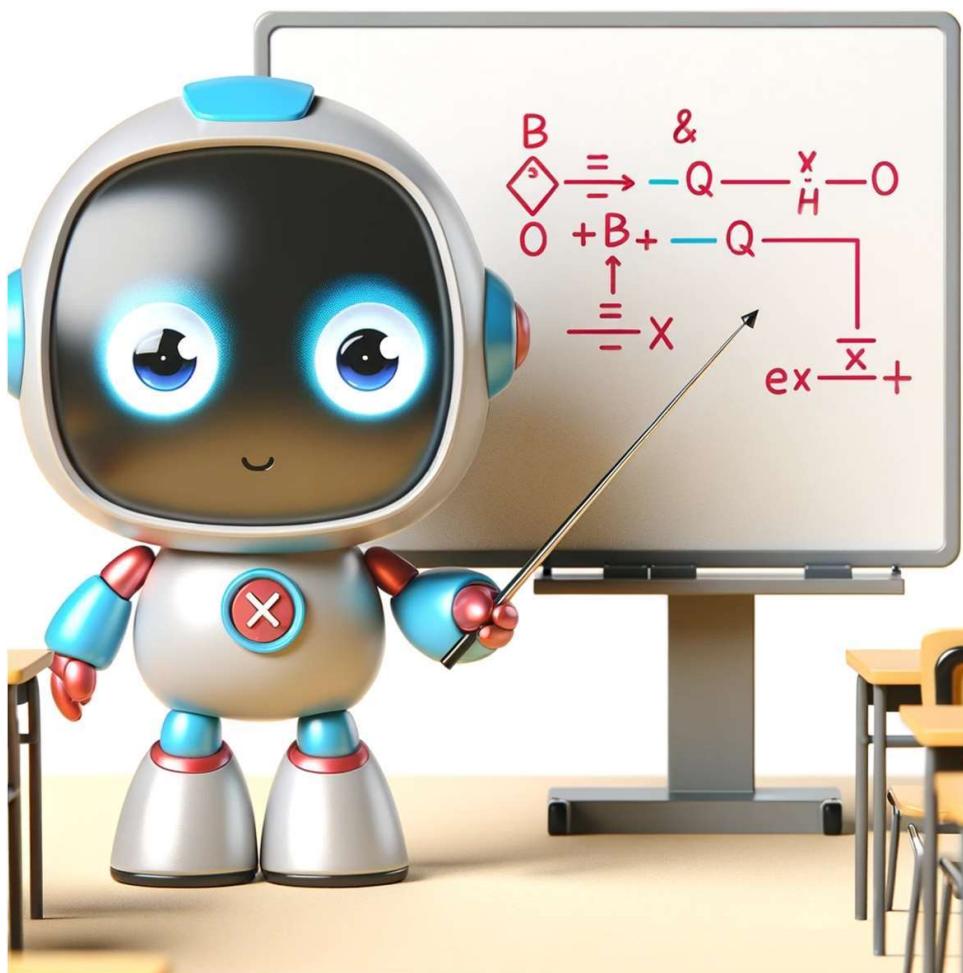
Kafka producer?

Configurazioni aggiuntive

- **Retries:** Quando il produttore riceve un messaggio di errore dal server, l'errore potrebbe essere transitorio (ad esempio, mancanza di un leader per una partizione). In questo caso, il valore del parametro **retries** controllerà quante volte il produttore ritenterà l'invio del messaggio prima di rinunciare e notificare al client un problema. Per impostazione predefinita, il produttore attende 100 ms tra i tentativi, ma è possibile controllare questo valore utilizzando il parametro `retry.backoff.ms`.



Esempio



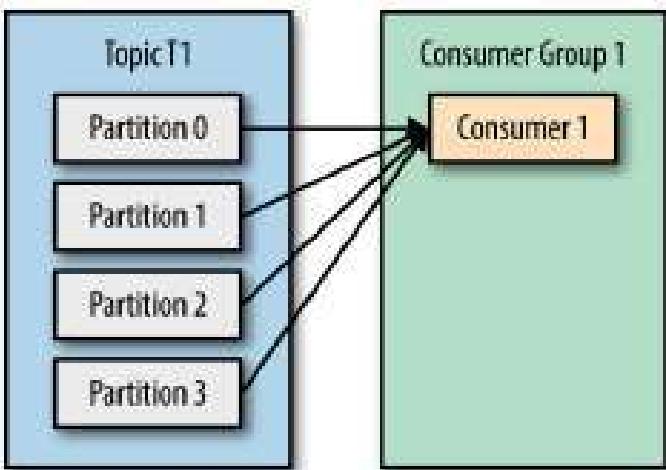
A photograph of a woman with long blonde hair tied back, wearing a black blazer, sitting at a white desk. She is facing away from the camera, writing in a small notebook with a pen. On the desk, there is a small potted plant in a copper-colored pot, a pair of scissors, and some dried botanical specimens. Behind her is a white wall with a framed abstract painting. To the right of the image, there is a blue vertical bar containing the text.

Consumatori (Consumer)

Consumatori

Kafka Consumer?

Iniziamo



- Al cuore dell'API dei consumer c'è un semplice ciclo per richiedere dati aggiuntivi al server. Una volta che il consumatore si iscrive ai topic, il ciclo di poll gestisce tutti i dettagli della coordinazione, dei riequilibri delle partizioni, degli "heartbeats" e dell'acquisizione dei dati, lasciando al programmatore un'API pulita che restituisce semplicemente i dati disponibili dalle partizioni assegnate.

Consumatori

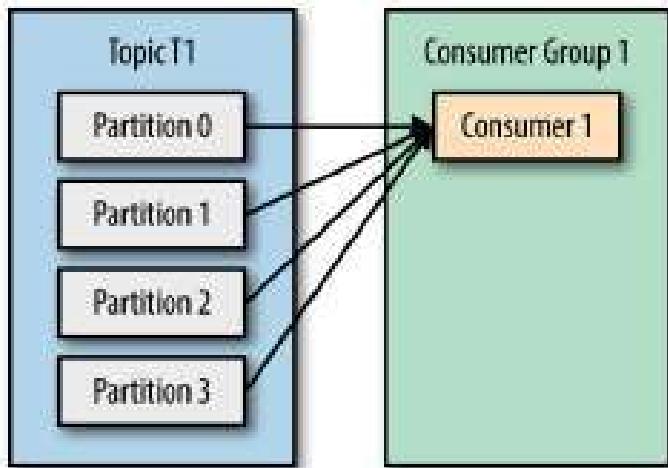
Kafka Consumer?

Iniziamo

- Le applicazioni che hanno bisogno di leggere dati da Kafka utilizzano un KafkaConsumer per iscriversi ai topic di Kafka e ricevere messaggi da questi topic. Leggere dati da Kafka è un po' diverso rispetto alla lettura di dati da altri sistemi di messaggistica, e ci sono pochi concetti e idee uniche coinvolti.

Concetti base

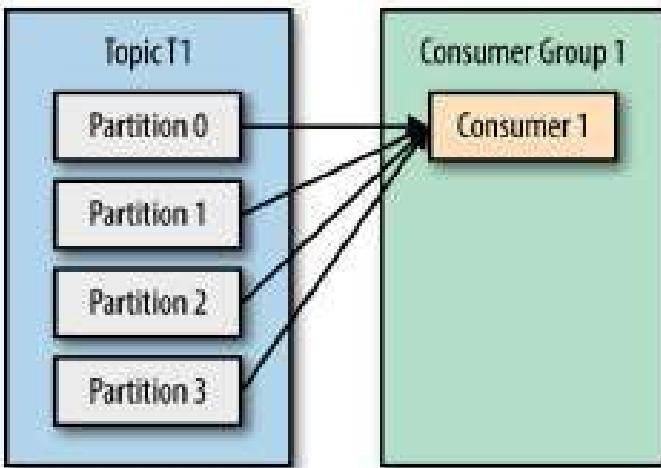
- Consumer e Gruppi di consumer:** L'applicazione creerà un oggetto consumatore, si iscriverà al topic appropriato e inizierà a ricevere messaggi, convalidandoli e scrivendo i risultati.



Consumatori

Kafka Consumer?

Concetti base



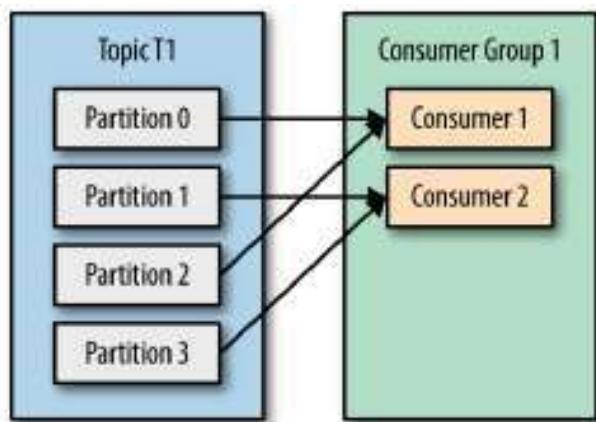
- I consumatori di Kafka sono tipicamente parte di un gruppo di consumatori.
- Quando diversi consumatori sono iscritti a un topic e fanno parte dello stesso gruppo di consumatori, ogni consumatore nel gruppo riceverà messaggi da un sottoinsieme diverso delle partizioni nel topic.
- Si prenda ad esempio il topic T1 con quattro partizioni raffigurato sulla sinistra.
- Si supponga ora di aver creato un nuovo consumatore, C1, che è l'unico consumatore nel gruppo G1, e lo utilizziamo per iscriverci al topic T1.

Consumatori

Kafka Consumer?

Concetti base

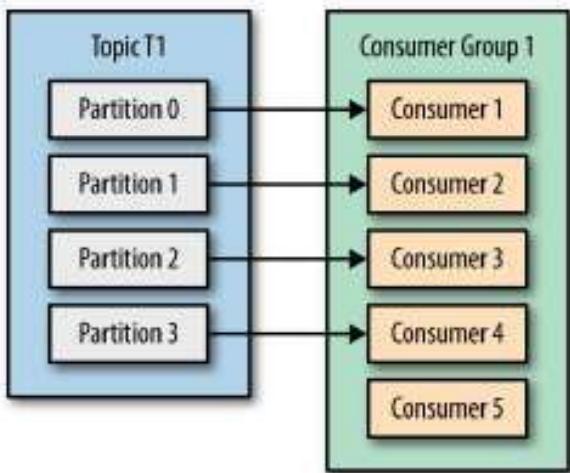
- Se si aggiunge un altro consumatore, C2, al gruppo G1, ogni consumatore riceverà solo i messaggi da due partizioni.



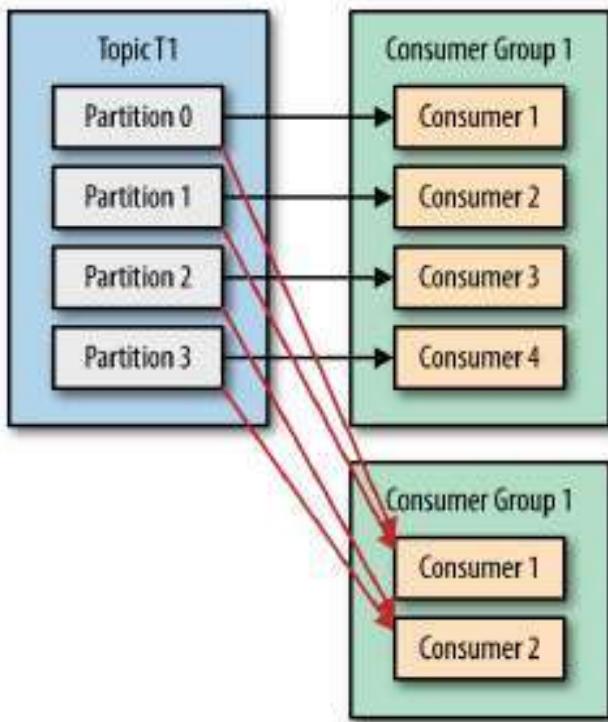
Consumatori

Kafka Consumer?

Concetti base



- Se si aggiunge un altro consumatore, C2, al gruppo G1, ogni consumatore riceverà solo i messaggi da due partizioni.
- Se si aggiungono più consumatori a un singolo gruppo con un singolo topic rispetto al numero di partizioni disponibili, alcuni dei consumatori saranno inattivi e non riceveranno nessun messaggio.
- Il modo principale per scalare il consumo di dati da un topic Kafka è aggiungere più consumatori a un gruppo di consumatori. È comune che i consumatori di Kafka eseguano operazioni ad alta latenza, come la scrittura su un database o un calcolo che richiede molto tempo sui dati.

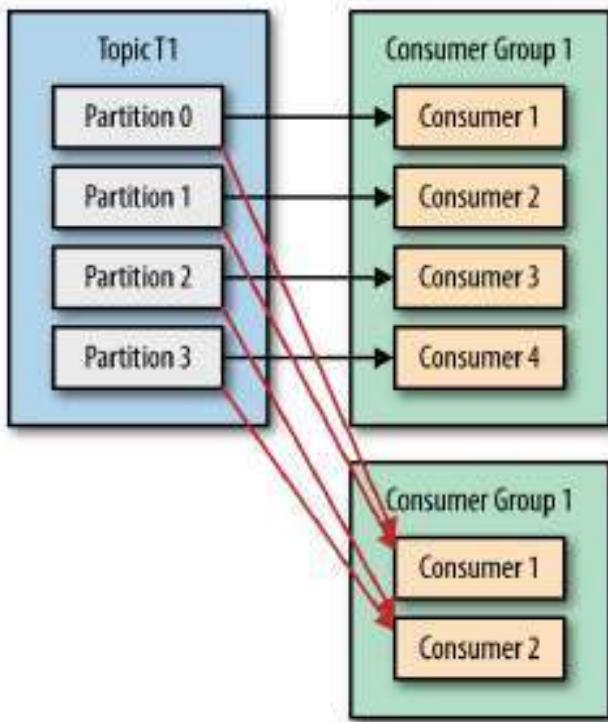


Consumatori

Kafka Consumer?

Concetti base

- Oltre ad aggiungere consumatori per scalare una singola applicazione, è molto comune avere più applicazioni che devono leggere dati dallo stesso topic.
- In effetti, uno degli obiettivi principali nel design di Kafka era rendere i dati prodotti nei topic di Kafka disponibili per molti casi d'uso in tutta l'organizzazione.
- Presa l'immagine precedente, se venisse aggiunto un nuovo gruppo di consumatori G2 con un singolo consumatore, questo consumatore riceverà tutti i messaggi nel topic T1 indipendentemente da ciò che fa G1. G2 può avere più di un singolo consumatore, in tal caso ognuno di essi riceverà un sottoinsieme delle partizioni, proprio come abbiamo mostrato per G1, ma G2 nel complesso riceverà comunque tutti i messaggi indipendentemente dagli altri gruppi di consumatori.



Consumatori

Kafka Consumer?

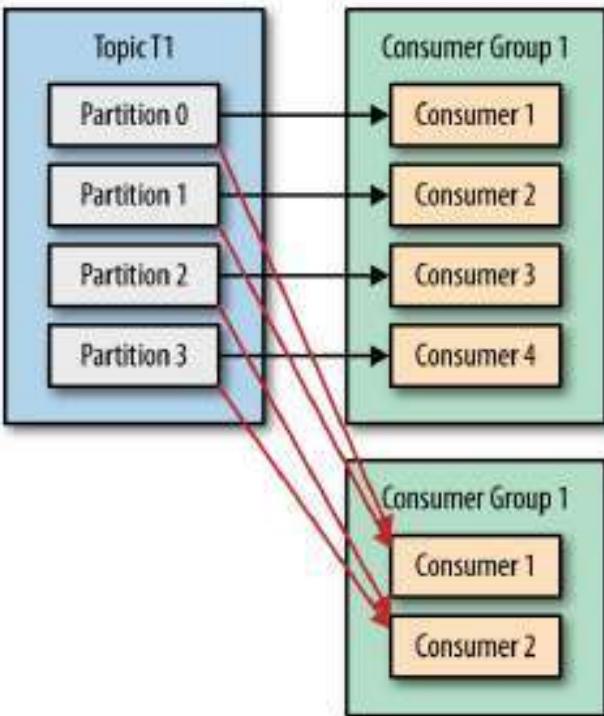
Concetti base

- Il trasferimento della proprietà delle partizioni da un consumatore a un altro è chiamato riequilibrio (**rebalance**).
- I riequilibri sono importanti perché forniscono al gruppo di consumatori elevata disponibilità e scalabilità (consentendoci di aggiungere e rimuovere consumatori facilmente e in modo sicuro).
- Durante un rebalance, i consumatori non possono consumare messaggi, quindi un riequilibrio è essenzialmente una breve finestra di non disponibilità dell'intero gruppo di consumatori.

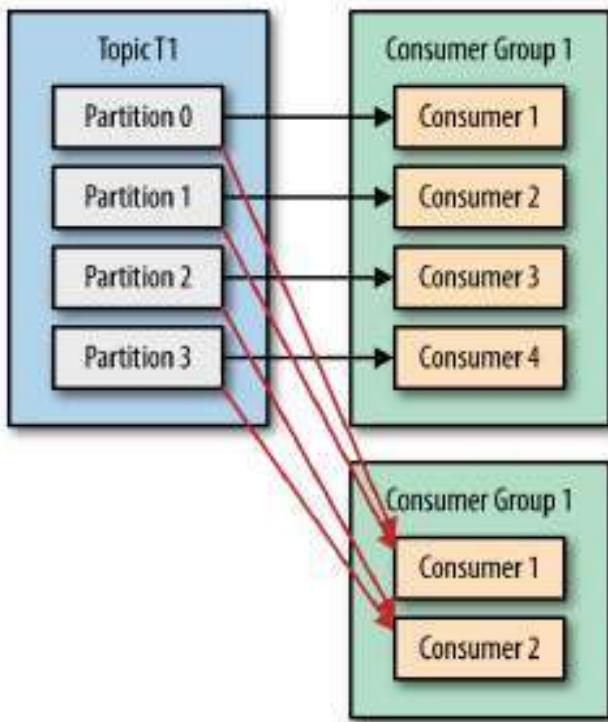
Consumatori

Kafka Consumer?

Concetti base



- I consumatori mantengono la loro appartenenza a un gruppo di consumatori e la proprietà delle partizioni loro assegnate inviando "heartbeats" a un broker Kafka designato come coordinatore del gruppo (questo broker può essere diverso per gruppi di consumatori diversi). Finché il consumatore invia "heartbeats" a intervalli regolari, si presume che sia vivo, in buona salute e che stia elaborando i messaggi dalle sue partizioni.
- Gli "heartbeats" vengono inviati quando il consumatore interroga (cioè recupera i record) e quando conferma i record che ha consumato.

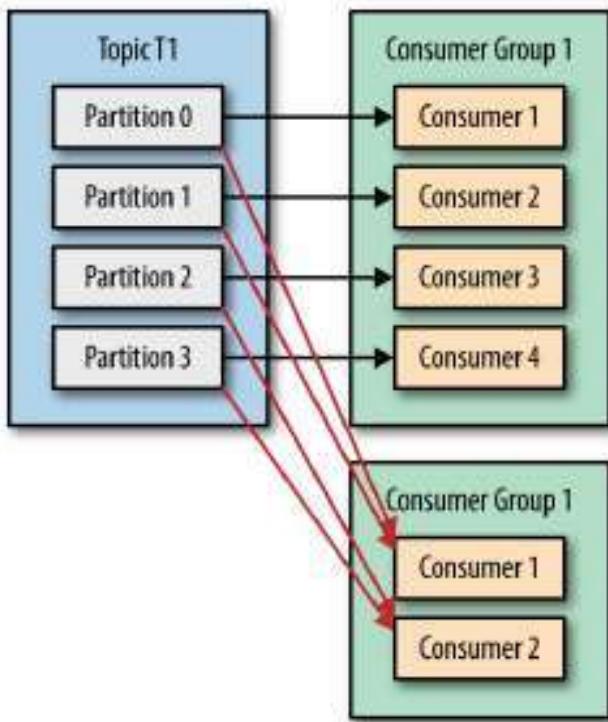


Consumatori

Kafka Consumer?

Concetti base

- Se il consumer smette di inviare "heartbeats" per un periodo sufficientemente lungo, la sua sessione scadrà e il coordinatore del gruppo lo considererà morto, innescando un rebalance.
- Se un consumer si interrompe improvvisamente e smette di elaborare messaggi, al coordinatore del gruppo ci vorranno alcuni secondi senza "heartbeats" per decidere che è morto e innescare il riequilibrio.
- Quando un consumatore viene chiuso correttamente, il consumatore notificherà al coordinatore del gruppo che sta lasciando, e il coordinatore del gruppo innesterà immediatamente un riequilibrio, riducendo il divario nel processo.



Consumatori

Kafka Consumer?

Creiamo un Consumer

- Creare un KafkaConsumer è molto simile a creare un KafkaProducer: si crea un'istanza di Java Properties con le proprietà che si desidera passare al consumatore.
- Per iniziare, è sufficiente utilizzare le tre proprietà obbligatorie: **bootstrap.servers**, **key.deserializer** e **value.deserializer**.
- C'è una quarta proprietà, che non è strettamente obbligatoria. La proprietà è **group.id** e specifica il gruppo di consumatori a cui appartiene l'istanza di KafkaConsumer. **Sebbene sia possibile creare consumatori che non appartengono a nessun gruppo di consumatori, questa è un'opzione non comune, quindi per la maggior parte del capitolo supporremo che il consumatore faccia parte di un gruppo.**



Consumatori

Kafka Consumer?

Consumer attributi

- **fetch.min.bytes:** Questa proprietà consente a un consumatore di specificare la quantità minima di dati che desidera ricevere dal broker durante il recupero dei record. Se un broker riceve una richiesta di record da un consumatore ma i nuovi record ammontano a meno byte di quanto specificato in min.fetch.bytes, il broker attende finché non sono disponibili più messaggi prima di inviare i record al consumatore.
- **Session.timeout.ms:** La quantità di tempo che un consumatore può rimanere senza contatto con i broker pur essendo considerato vivo è predefinita a 3 secondi. Se passa più tempo di quanto specificato in session.timeout.ms senza che il consumatore invii un "heartbeat" al coordinatore del gruppo, viene considerato morto e il coordinatore del gruppo innesterà un riequilibrio del gruppo di consumatori per allocare le partizioni dal consumatore morto agli altri consumatori nel gruppo.