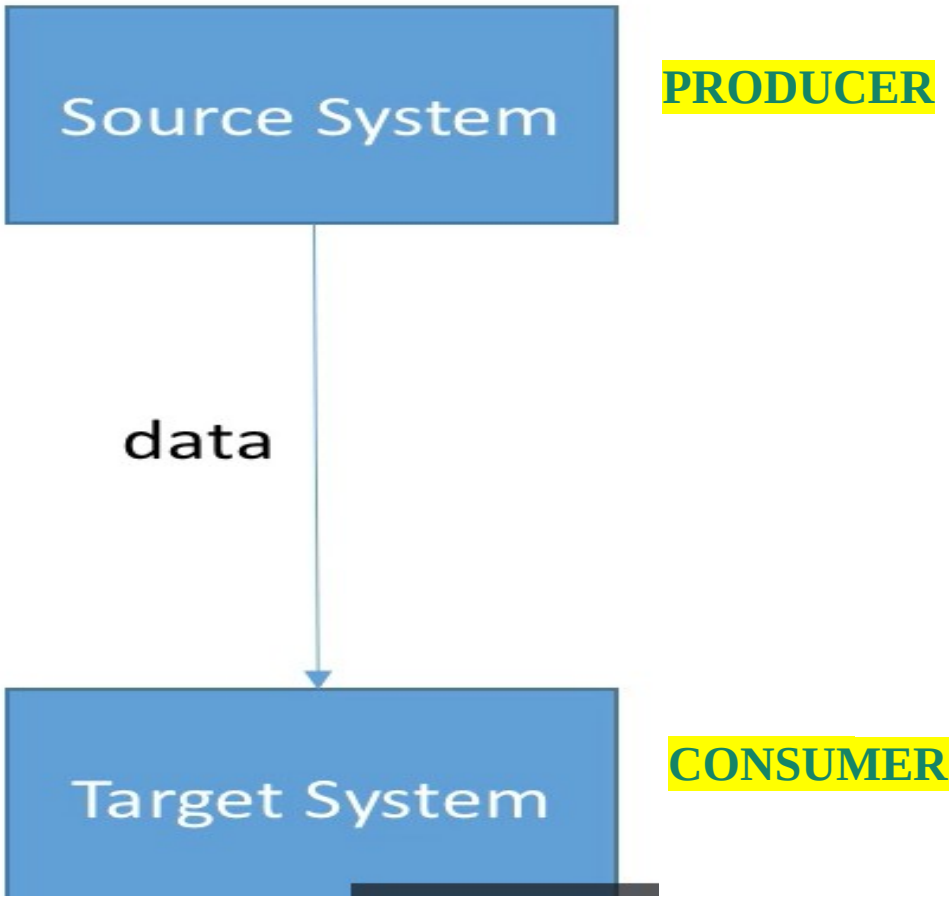
Apache Kafka Beginner

1.0 Introduzione

Supponiamo di avere il seguente caso d’uso, ovvero 2 sistemi che interagiscono:

- **sistema sorgente**
  (che ad esempio genera/produce dei dati e li scambia con un altro e quindi svolge il ruolo di Producer di dati)

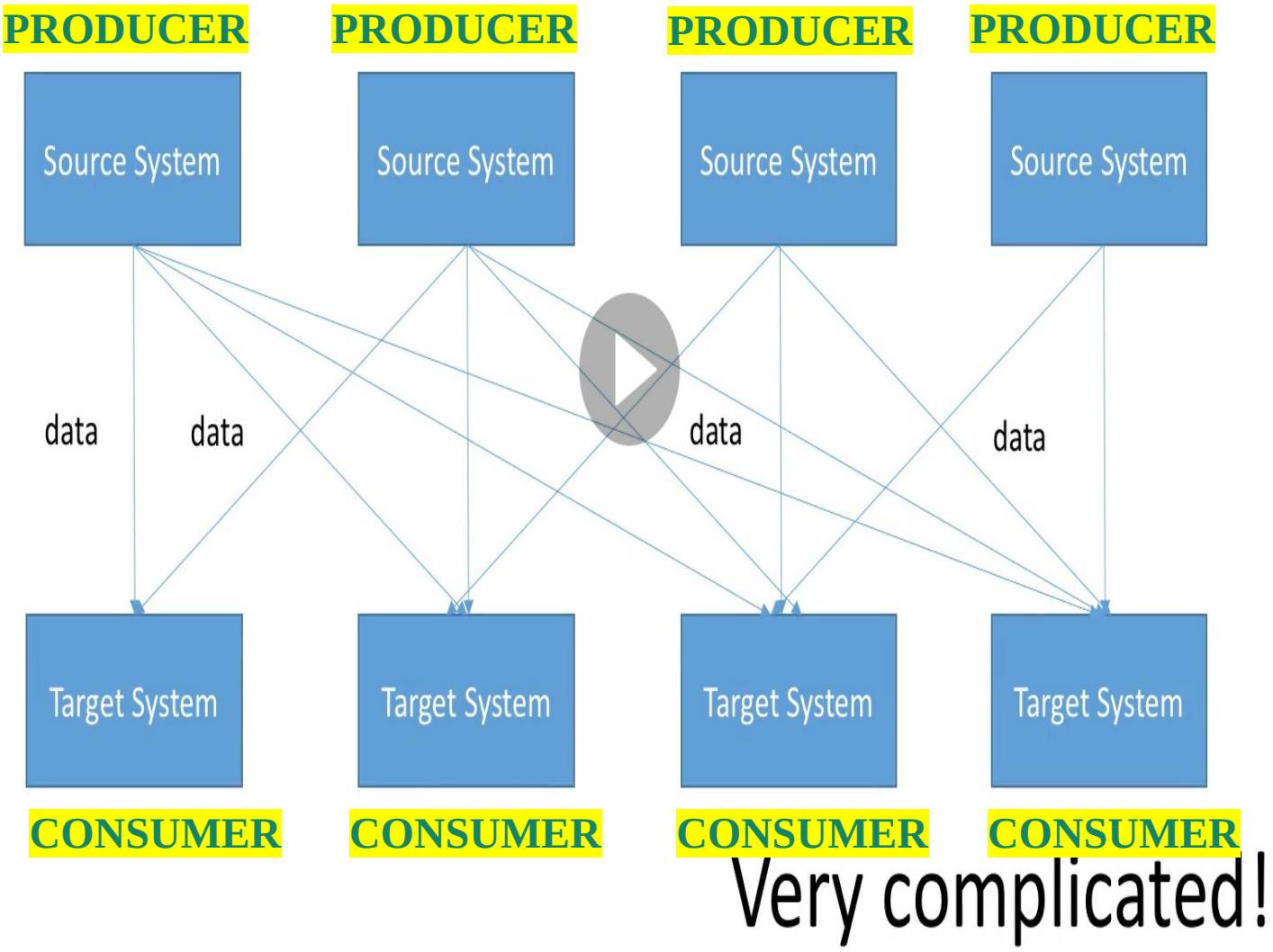
- **sistema destinazione**
  (che riceve i dati dal primo e li elabora in qualche modo e quindi svolge il ruolo di Consumer di dati)



Questo schema sembra ed è abbastanza semplice.

Ma vediamo quello che succede nella vita reale...

Supponiamo che gli affari della nostra azienda crescono e abbiamo quindi bisogno di avere maggiori risorse che generano dei dati e che li elaborano.
Trovandoci quindi in uno schema simile a questo:



In questo schema possiamo notare:

- Diversi sistemi sorgenti

- Diversi sistemi di destinazione

oggetto fondamentale è che tutti quanti i sistemi devono poter scambiare dati tra loro.

Possiamo subito notare che questo schema di scambio dei dati è molto più complesso rispetto a quello precedente dove avevamo solo 2 sistemi.

Le cose potrebbero diventare ancora più complicate se avessimo in generale:

- N sistemi sorgenti

- M sistemi di destinazione

a quel punto per fare in modo che tutti i sorgenti comunicano con tutte le destinazione dovremmo avere N*M integrazioni, dove per singola integrazione intendiamo il singolo collegamento che esiste tra sorgente e destinazione e che permette loro di scambiare dati.

Inoltre per ogni integrazione si possono avere difficoltà diverse che possono riguardare:

1. il protocollo da usare per la comunicazione
(TCP, HTTP, REST, FTP, ...)

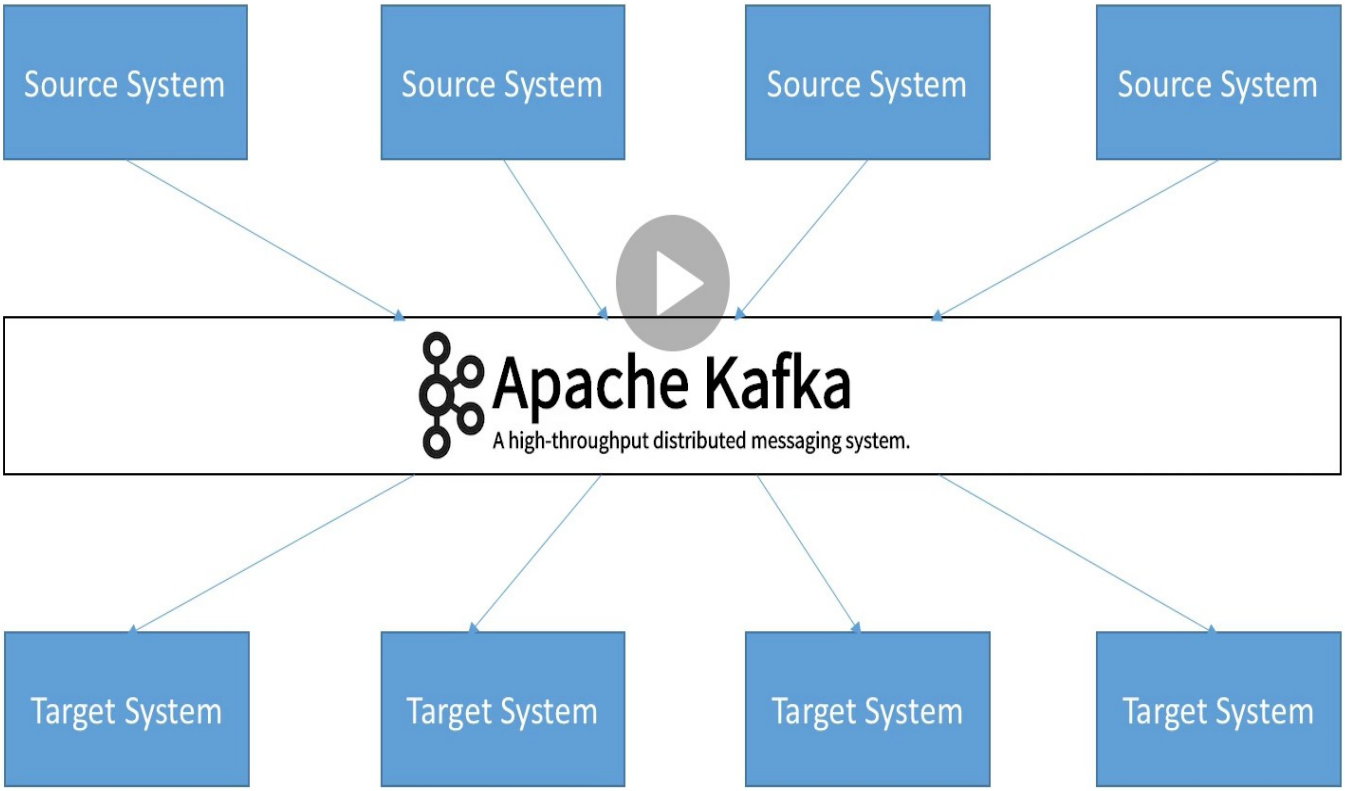
2. il formato dei dati di scambio
(JSON, XML, CSV, Binary, ...)

3. Lo schema dei dati (db) e la loro evoluzione futura

Inoltre ancora, ogni volta che si integra un sistema sorgente con un sistema destinazione, ci sarà un incremento del carico delle connessioni in quanto stiamo aggiungendo un altro flusso.

Abbiamo quindi capito che integrare diversi sistemi non è banale e che in fase di integrazione possono sorgere diverse problematiche come quelle accennate sopra.

Occore quindi trovare una soluzioni a tale problematiche, ed ecco qui che entra in gioco Apache Kafka.



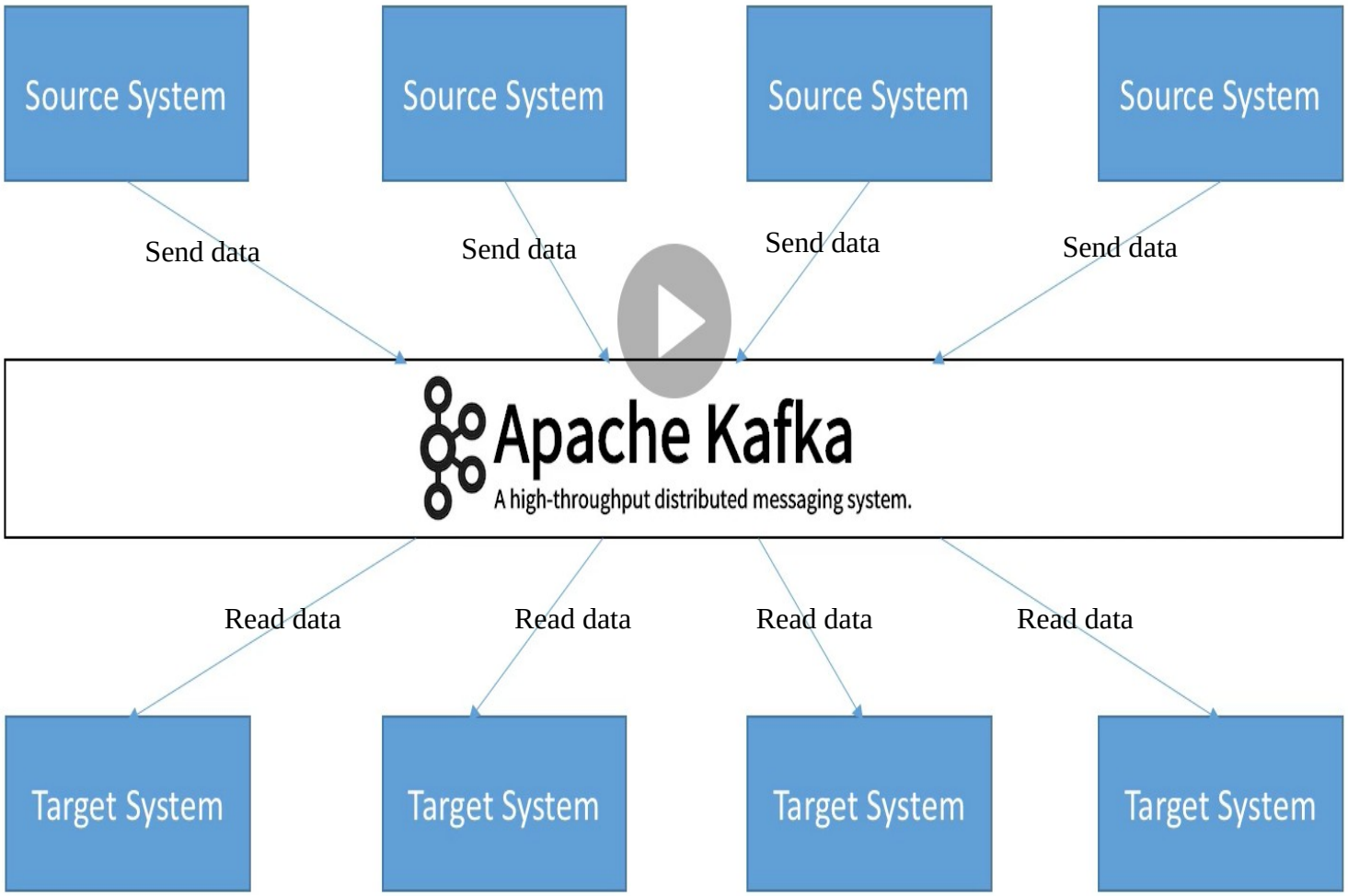
Grazie ad Apache Kafka possiamo:

→ **DISACCOPIARE** i flussi dei dati dai sistemi

Questo significa che un sistema sorgente non deve conoscere un sistema di destinazione per poter scambiare dei dati e quindi sono 2 sistemi indipendenti l'uno dall'altro.
Questo non era assolutamente vero nello schema precedente, infatti la freccia che partiva dalla sorgente e arriva alla destinazione rappresenta bene l'accoppiamento tra i 2, cioè i 2 sistemi per scambiare dati devono conoscersi.

In questo nuovo scenario Apache Kafka si posiziona nel mezzo e fa da ponte tra i sistemi sorgenti e quelli di destinazione.
Quindi un sistema sorgente non invierà più dei dati al diretto interessato sistema di destinazione, ma lo invierà a Kafka che lo memorizzerà in qualche modo.
D'altra parte i sistemi di destinazione prenderanno i dati di interesse prodotti dai sistemi sorgenti da Kafka.

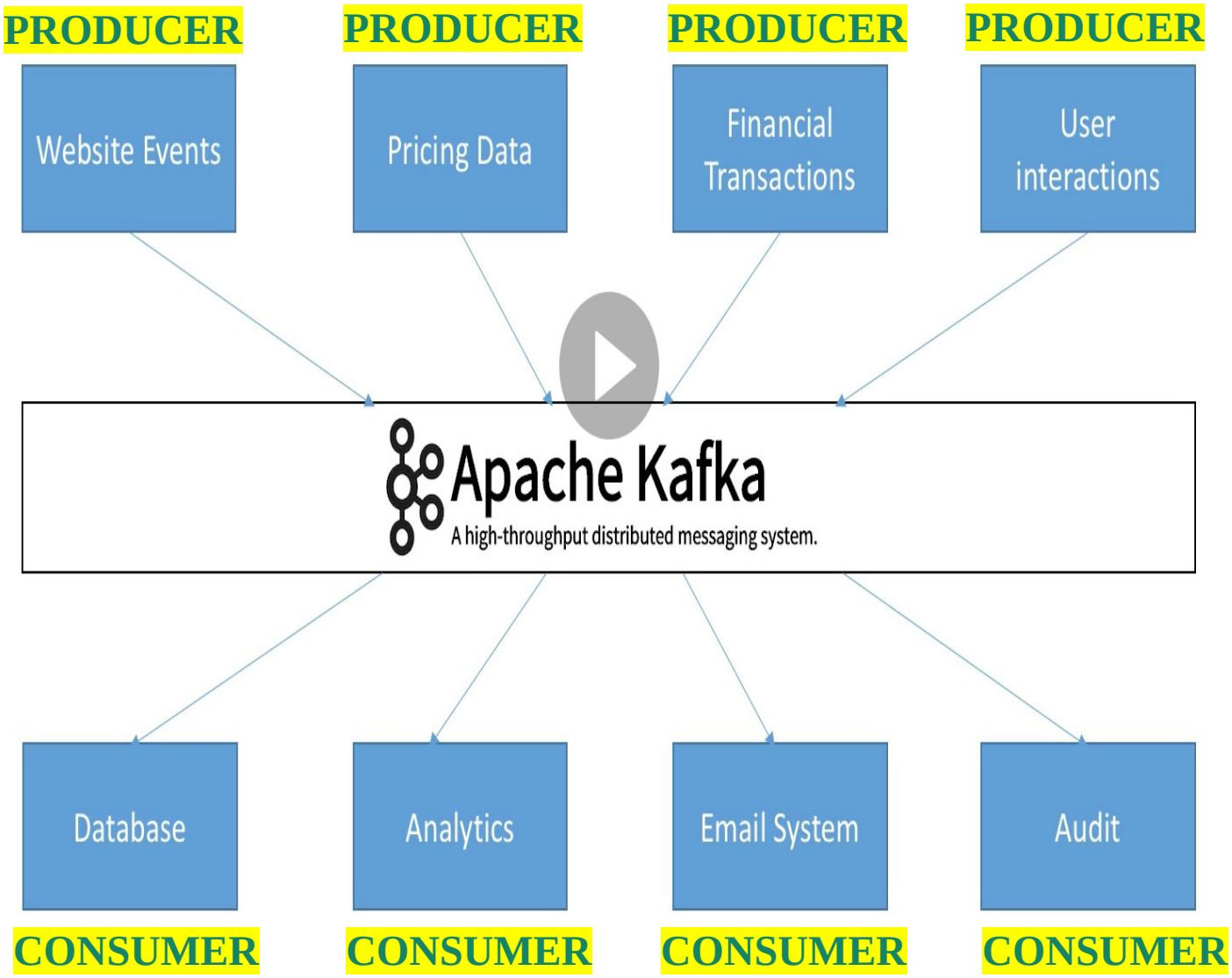
Sistemi sorgenti non sono a conoscenza dei sistemi di destinazione, loro solo producono dati e li inviano a Kafka,
Per loro il target dei dati è Kafka e non altri sistemi



Sistemi destinazione non sono a conoscenza dei sistemi sorgente, loro solo leggono dati da kafka,
per loro la fonte dati è kafka e non altri sistemi che producono dati.

In questo modo è possibile sostituire un qualsiasi sistema di sorgente o destinazione senza avere nessun impatto in quanto i sistemi sono tutti totalmente disaccoppiati, grazie a Kafka.

Schema reale di sistemi sorgente e destinazione che scambiano dati tra loro:



Ma perché usare Kafka come soluzione di integrazione?
Perchè ha i seguenti vantaggi:

1. Architettura distribuita e fault-tollerance (tolleranza ai guasti)

2. Scalabilità orizzontale
   → possiamo scalare più di 100 broker che lavorano in parallelo
   → possiamo scalare milioni di messaggi per sec.
   E questo è dimostrato da società che usano Kafka come:
   Linkedin, Netflix, Airbnb, Uber, ...

3. Estremamente performante con una latenza di meno di 10ms per questo è considerato un sistema di trasmissione di mssaggi real-time

4. Usato da moltissime aziende e quindi è un prodotto affermato

Casi d’uso di apache Kafka sono i segeunti:

1. Sistema di messaggistica
2. Tracciamento delle attività
3. Raccolta di metriche da molti luoghi diversi per dispositivi IoT
4. Logging di applicazioni
5. Stream processing (con Kafka stream API o SPARK)
6. Disaccoppiamento di sistemi dipendenti

.....
.....

Storie d’uso reali di Kafka:

1. Netflix usa Kafka per applicare consigli in tempo reale mentre stai guardando i suoi programmi TV.
   Ecco spiegato il motivo per cui, quando lasciamo una serie o un film riceviamo subito nuove raccomandazioni sul prossimo da vedere.

2. Uber usa Kafka per raccogliere in tempo reale dati su: utenti, taxi e percorsi, in modo da aggiornare in tempo reale i prezzi.

3. Linkedin usa Kafka per raccogliere dati sulle interazioni degli utenti in modo da consigliare in tempo reale nuove interazioni

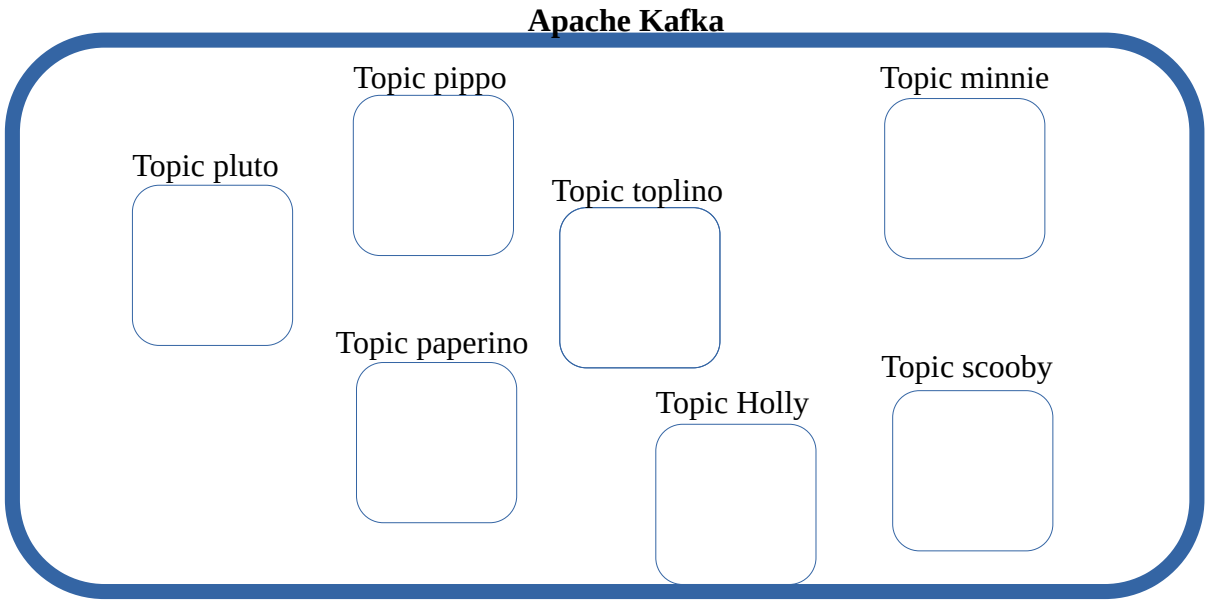
2.0 Teoria su Kafka

In questo capitolo parleremo degli argomenti core di apache kafka che sono sotanzialmente 3:

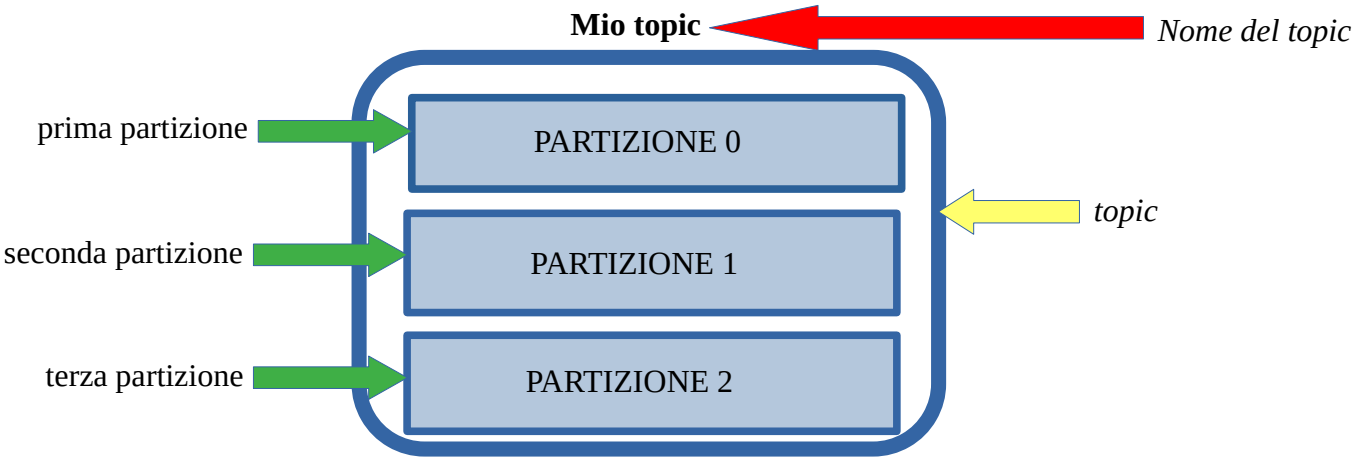
i. ***Topic***
ii. ***Partizione***
iii. ***Offset***

In apache Kafka, il topic è alla base di tutto.
Ma Cosa è un Topic ?
Un topic è uno specifico stream (flusso) di dati/messaggi.

Nella pratica un topic può essere considerato molto simile ad una tabella di un database.
Così come le tabelle di un database hanno un nome, lo stesso vale per i topic e così come per i database possiamo creare diverse tabelle, lo stesso vale per i topic.
Quindi in kafka possiamo avere un insieme di topic e ciascuno viene identificato dal suo nome.



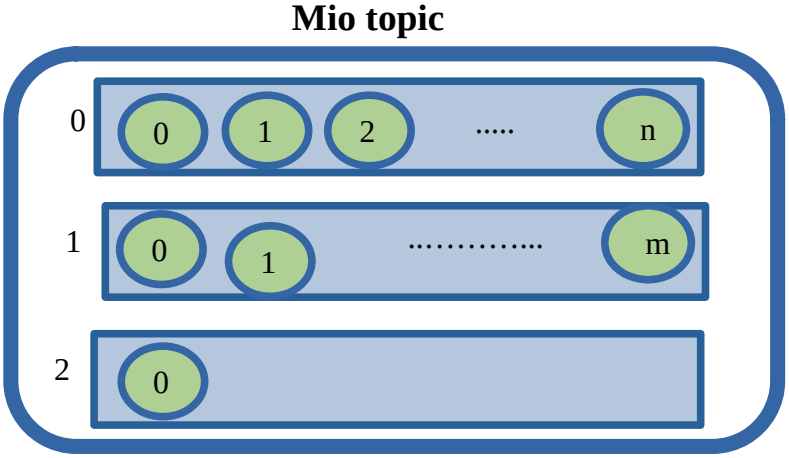
Come per i database le tabelle sono divise in più record, in kafka un topic è diviso in più partizioni.
Di seguito abbiamo una raffigurazione di un topic con 3 partizioni, ma in generale un topic può avere n partizioni.



L'immagine mostra chiaramente come un topic è diviso in più partizioni (nel nostro esmpio 3).

NOTA BENE) Le partizioni sono numerate e partono sempre dal numero 0.
Quindi in generale se avremmo avuto un topic con all'interno n partizioni, la prima avrebbe avuto sempre numero 0 e l'ultima avrebbe avuto numero n-1.

Ogni partizione contiene al suo interno un numero variabile di messaggi e ogni messaggio è identificato da un id che è un numero incrementale che viene molto comunemente chiamato offset.
I messaggi situati all'interno di una specifica partizione non sono disposti in modo casuale ma sono ordinati.



NOTA BENE 1) All’interno di ciascuna partizione, i messaggi sono ordinati nel senso che verranno depositati/scritti nella partizione in ordine crescente di id, ovvero prima arriva nella partizione messaggio con id=1, poi arriva messaggio con id=2, poi arriva messaggio con id=3, e così via…
Non capiterà mai in una partizione che arrivi ad esempio il messaggio con id=4 e dopo quello con id=3 in quanto kafka ci garantisce che i messaggi si depositeranno/scritti in ordine crescente di id

NOTA BENE 2) Ogni partizione può avere un numero di messaggi che è “potenzialmente infinito”

NOTA BENE 3) Ogni partizione può avere un numero di messaggi diverso dalle altre partizioni

NOTA BENE 4) l’offset o id serve per individuare un particolare messaggio all’interno di una specifica partizione.

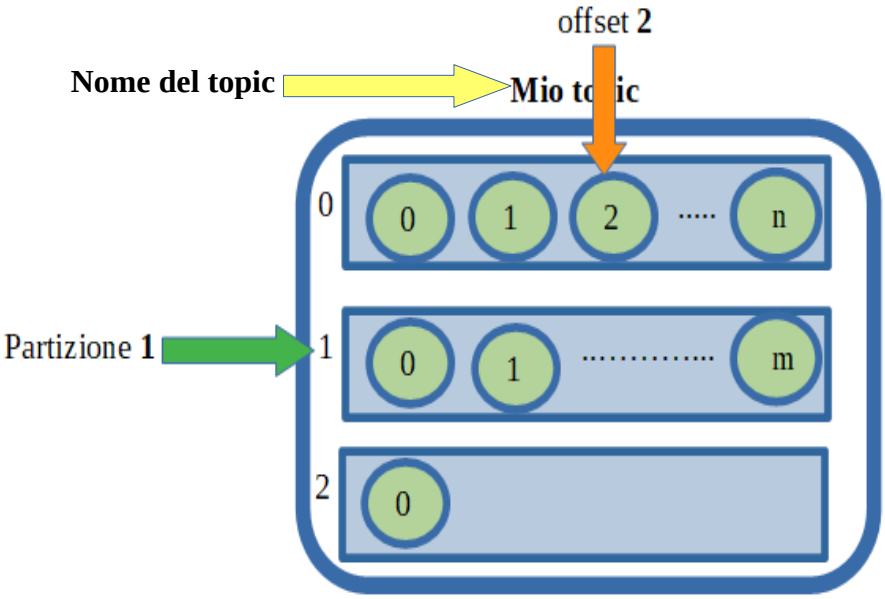
NOTA BENE 5) l’offset ha significato solo per una specifica partizione
Questo implica che partizione 1 offset 2 individua un messaggio completamente diverso da partizione 2 offset 2

Quindi se io volessi individuare un qualsiasi messaggio all’interno di una partizione di un particolare topi mi basterebbero solamente 3 informazioni:

→ nome del topic
→ il numero della partizione
→ l’offset (o id del messaggio)

Ad esempio il 3° messaggio della 2° partizione del topic “mio topic” e identificato dalla terna (“mio topic”, 1, 2) dove:

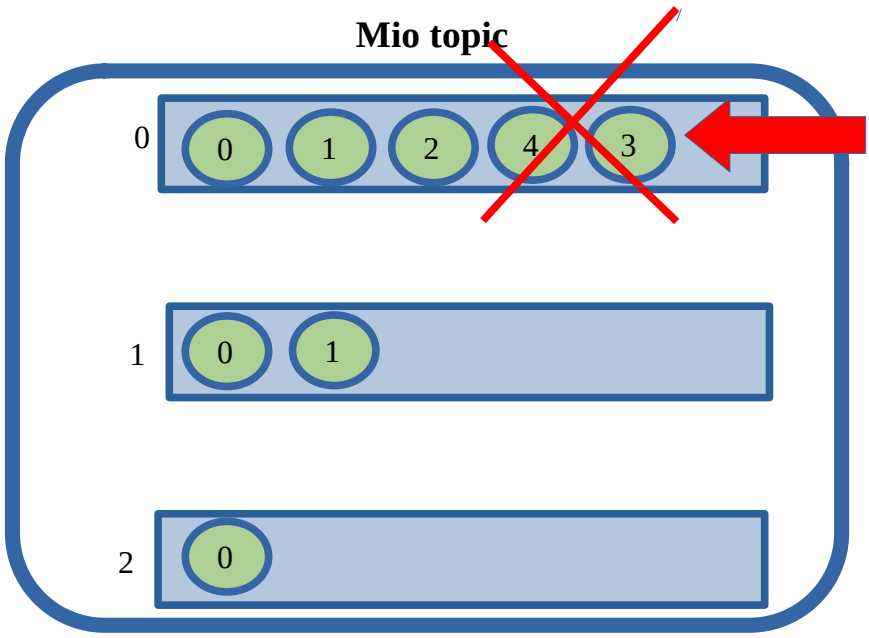
- “mio topic” individua un particoalare topic
- 1 individua la partizione
- 2 individua il messaggio all’interno della partizione 1, che poi coincide anche con lo scostamento che ho fatto all’interno della partizione 1 per trovare il messaggio e infatti offset vuol dire proprio scostamento.



Abbiamo detto prima che i messaggi all'interno di una partizione sono ordinati.
Ma in che senso sono ordinati?
Nel senso che kafka ci garantisce che il messaggio con id=0 arriverà nella partizione prima del messaggio con id=1, e che il messaggio con id=1 arriverà nella partizione prima del messaggio con id=2, e che il messaggio con id=2 arriverà nella partizione prima del messaggio con id=3, ecc...
Così facendo i messaggi si depositeranno nella partizione rispettando sempre l'ordinamento crescente per id.

Abbiamo quindi capito che l'ordine di arrivo dei messaggi nella singola partizione è garantito da kafka.
Tuttavia l'ordine di arrivo dei messaggi attraverso le partizioni di un topic non è garantito da kafka. Questo significa che: potrebbe ad esempio succedere di ricevere prima il messaggio con id=5 nella partizione 1 e dopo ricevere il messaggio con id=4 nella partizione 2.

RICORDA L'ORDINE DI ARRIVO DEI MESSAGGI E' GARANTITO DA KAFKA SOLO A LIVELLO DI PARTIZIONE E NON CROSS PARTIZIONE.

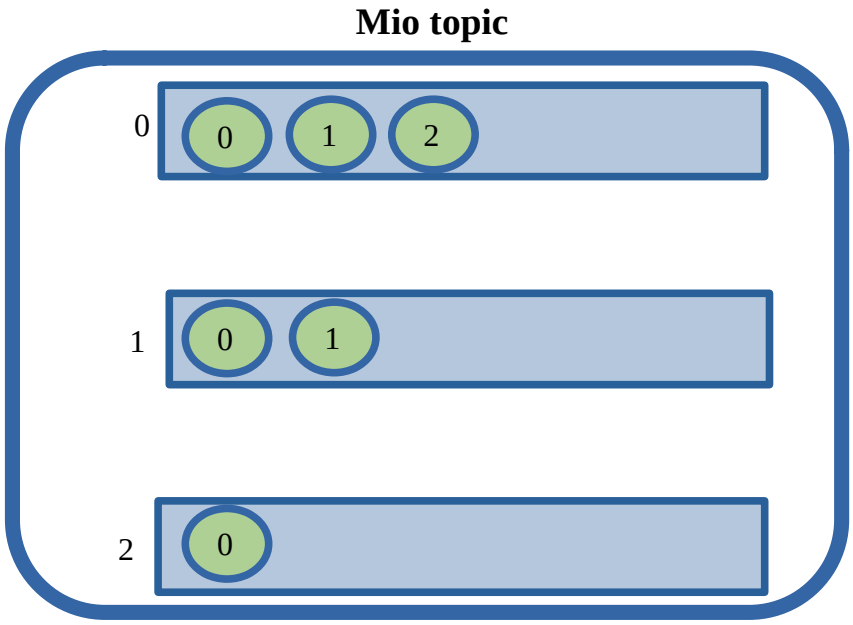


Questa situazione in una partizione non potrà mai verificarsi: ovvero che venga depositato nella partizione prima il messaggio con id=4 e poi quello con id=3

Vedendo l'immagine nella partizione 0 possiamo notare che il messaggio con id=4 è arrivato nella partizione prima del messaggio

con id=3 non rispettando quindi l'ordine di arrivo dei messaggi.
Ecco questo scenario non potrà mai capitare all'interno di una partizione in quanto Kafka ci garantisce che i messaggi arriveranno nella specifica partizione in modo ordinato.
Ovvero che il messaggio con id=3 arrivi prima del messaggio con id=4.

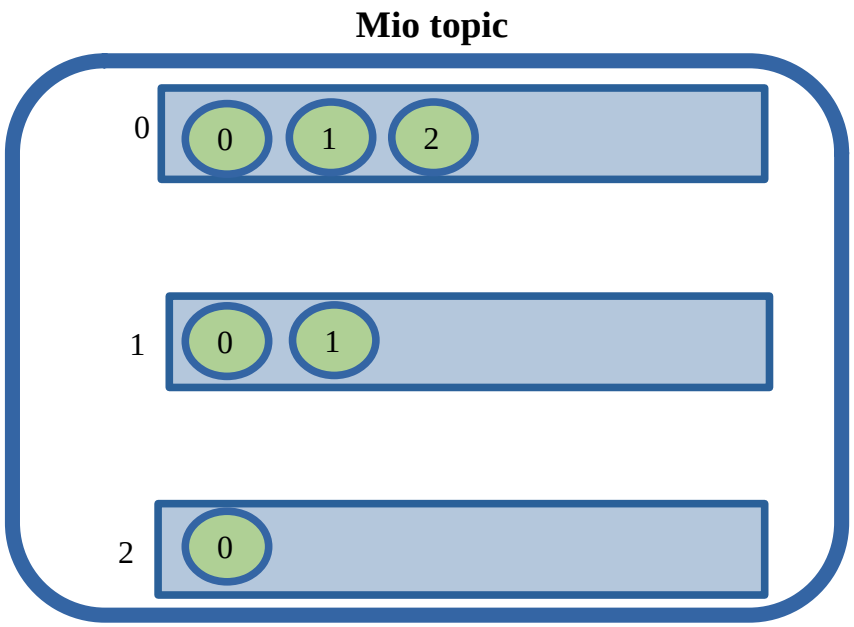
Diversa è invece la situazione della figura seguente:



Potrebbe invece verificarsi il caso che:
all'istante t_0 arrivi il messaggio con id=2 nella partizione 0 e successivamente all'istante t_1 arrivi il messaggio con id=0 nella partizione 2.
Questo è lecito perché kafka non garantisce invece l'arrivo ordinato dei messaggi cross partizioni.

I dati all’interno di Kafka non sono memorizzati in eterno ma sono memorizzati solo per un certo periodo di tempo.
Questo vuol dire che ad un certo punto questi dati verranno cancellati.
Per default Kafka memorizza i dati per un periodo che equivale ad una settimana dopo di che verranno cancellati.
Tuttavia gli id (offset) dei messaggi all’interno di una partizione non verranno ripristinati a 0 ma continueranno ad incrementare.

Altra cosa molto importante è che una volta che un dato/messaggio viene scritto su una partizione, questo è immutabile, che vuol dire che non può più essere modificabile.



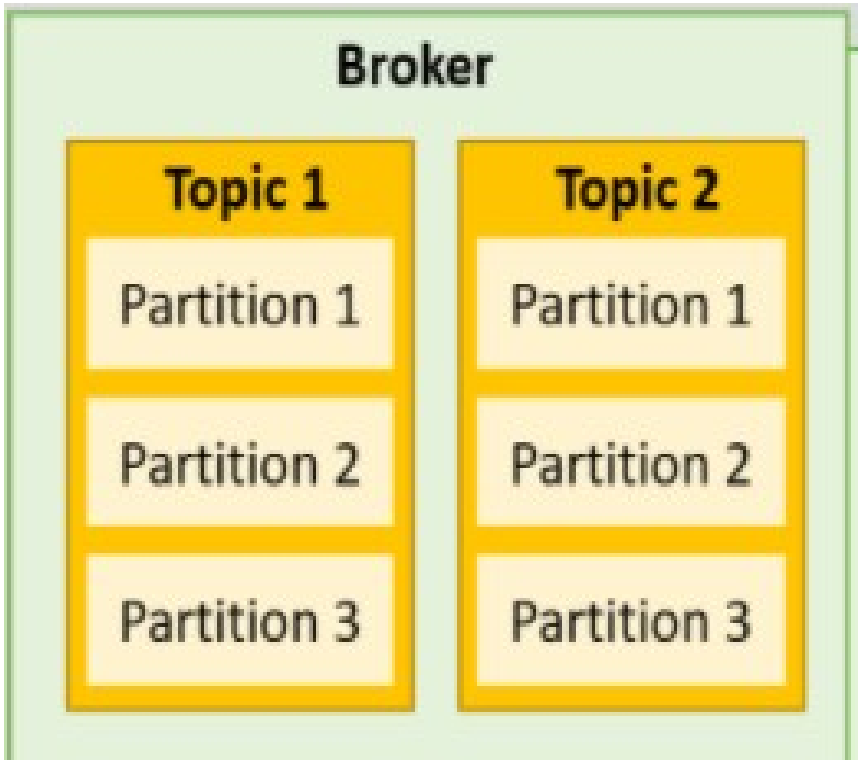
Ogni messaggio/dato una volta scritto sulla partizione di un topic, non può più essere modificato

Ma con quale logica kafka decide se un messaggio è destinato ad andare in una partizione piuttosto che in un’altra?
Se il messaggio non possiede una chiave, questo verrà memorizzato in una partizione casuale che verrà scelto secondo l’algoritmo Round-Robin
Round-Robin vuol dire che se abbiamo 5 partizioni, i messaggi che arrivano vengono memorizzati una volta nella partizione 0, poi nella partizione 1, poi nella partizione 2, poi nella partizione 3, poi nella partizione 4, poi nella partizione 5, e successivamente si comincia a depositare/scrivere il messaggio nella partizione 0, poi nella partizione 1 e così via.

3.0 Brokers e topics

Abbiamo parlato prima dei topic e abbiamo visto come sono strutturati

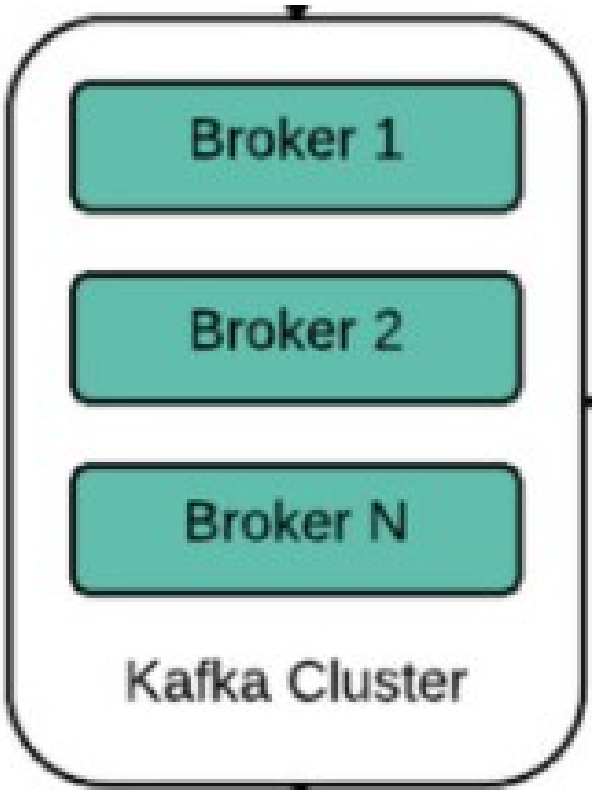
Ma una domanda che ci sorge spontanea adesso è:
Chi è che si occupa, contiene, gestisce i topics?
La risposta è i Brokers, anche conosciuti come Kafka server o nodi Kafka.



Nell'immagine sopra è raffigurato un broker con all'interno due topic (topic 1, topic 2) con 3 partizioni ciascuno, ma in generale un broker può contenere n topic.

La cosa importante da capire è che un broker è un'applicazione server deployata in una macchina fisica che permette di gestire 1 oppure n topics.

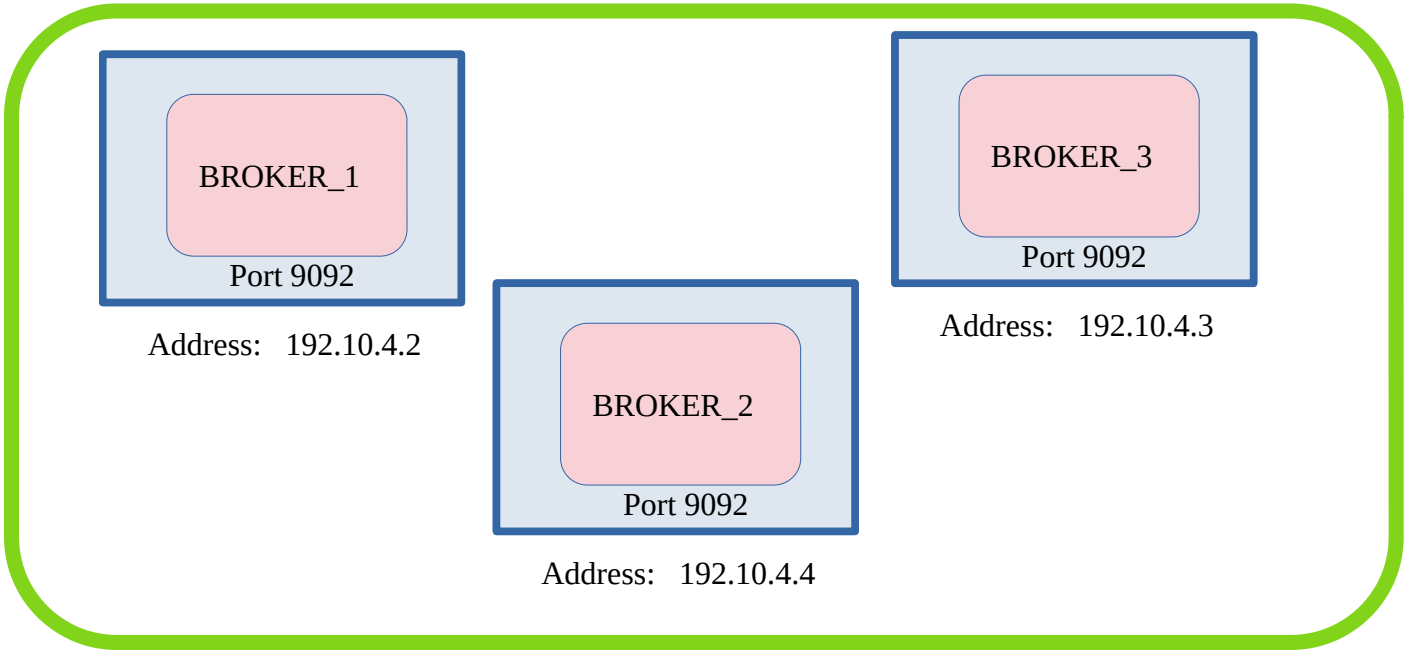
Nella realtà di solito non abbiamo solamente un broker ma abbiamo diversi brokers e quindi in questi casi si parla di cluster di broker (o cluster di nodi kafka).



Nell’immagine è raffigurato un Kafka Cluster con N Broker

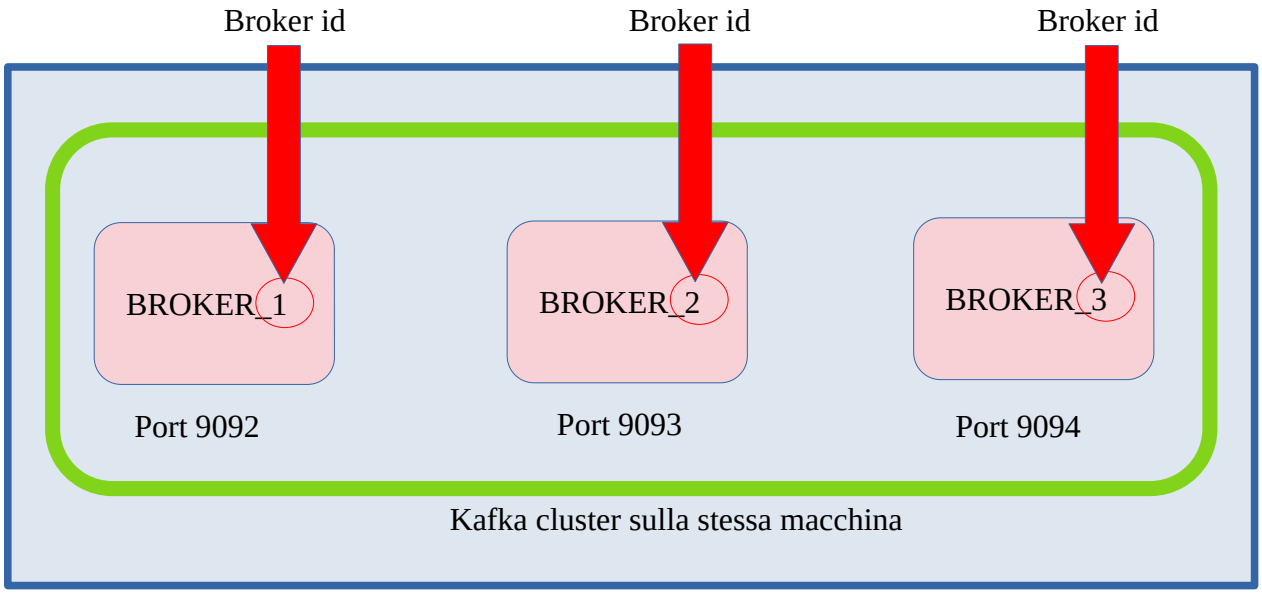
Abbiamo prima detto che un Broker Kafka di fatto è un’applicazione server che gestisce 1 o n topic.
Noi potremmo avere ciascun broker deployato su una macchina diversa oppure sulla stessa assegnando a ciascuno una porta diversa.

Esempio di un cluster con 3 broker ciascuno deployato su una macchina fisica diversa:



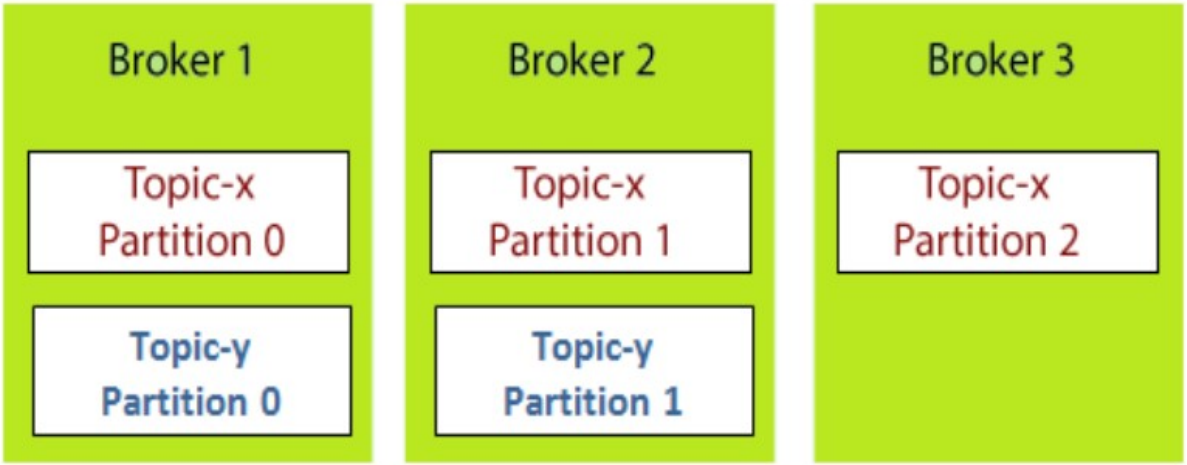
Kafka cluster distribuito su più macchine fisiche

Esempio di un cluster con 3 broker deployati sulla stessa macchina fisica:



NOTA BENE) Ogni Broker è identificato da un id univoco che deve essere necessariamente un numero intero.

NOTA BENE BENE) Abbiamo prima detto che ogni broker contiene uno o più topic nella sua interezza.
In realtà non è proprio così in quanto un broker non contiene un topic nella sua interezza, ma contiene solo una parte di tutte le sue partizioni.
Questo significa che in un determinato broker possiamo avere 3 partizioni su 5 esistenti del topic “Pippo” e 2 partizioni su 4 esistenti del topic “pluto”, ma non conterrà mai tutte le partizioni di un dato topic in quanto Kafka è distribuito.



L’immagine mostra tre broker: Broker 1, Broker 2, Broker 3 e due Topic: Topic X con 3 partizioni, Topic Y con 2 partizioni.
Nota come ciascun broker non contiene tutte le partizioni di un topic ma solo una parte.
Infatti il broker 1 contiene solo 1 partizione su 3 del topic X e 1 partizione su 2 e del topic Y, stessa cosa broker 2, mentre broker 3 contiene solo 1 partizione su 3 del topic X.

NOTA BENE) Ogni Broker (server kafka) all’interno di un cluster Kafka conosce tutti gli altri broker.