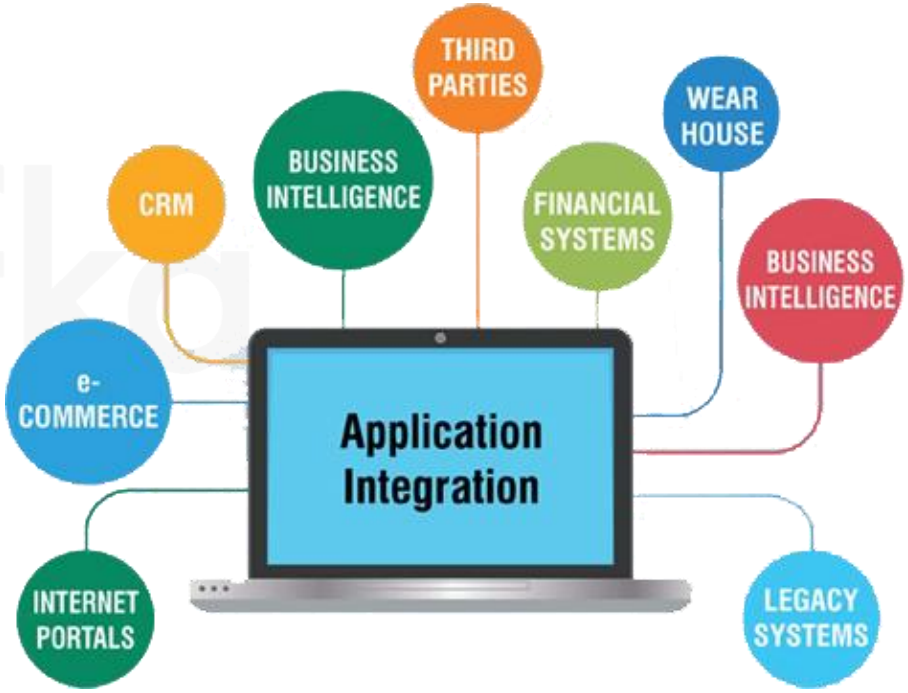

APACHE
kafka®

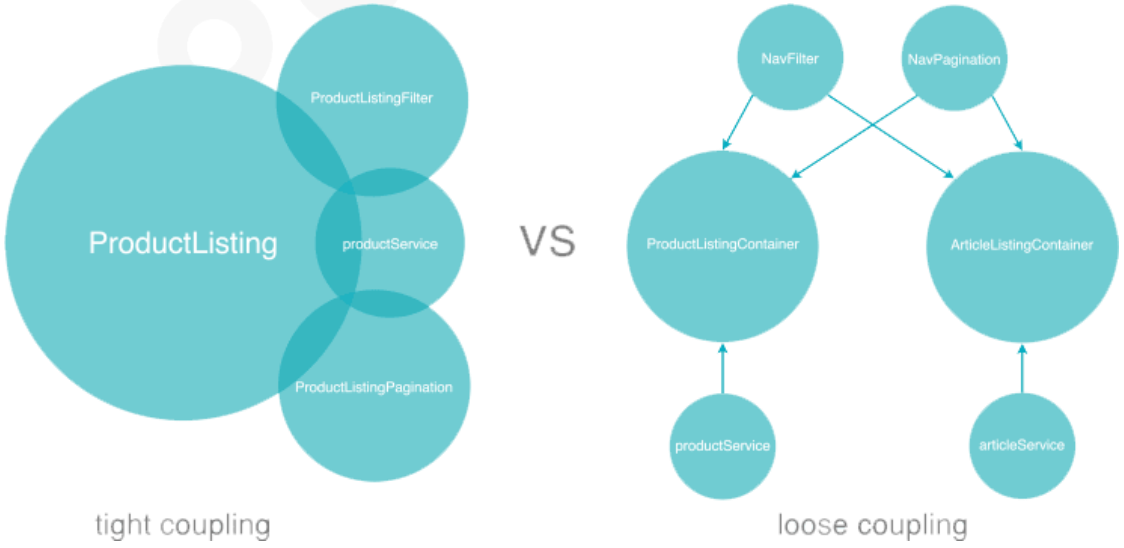
UNIT

Design di un sistema di Messaging

- Nella progettazione e sviluppo di un qualsiasi sistema di integrazione delle applicazioni, ci sono alcuni principi importanti che dovrebbero essere considerati e valutati:
  - Loose coupling (bassa dipendenza)
  - Common Interface Definitions
  - Latency (latenza)
  - Reliability (affidabilità)

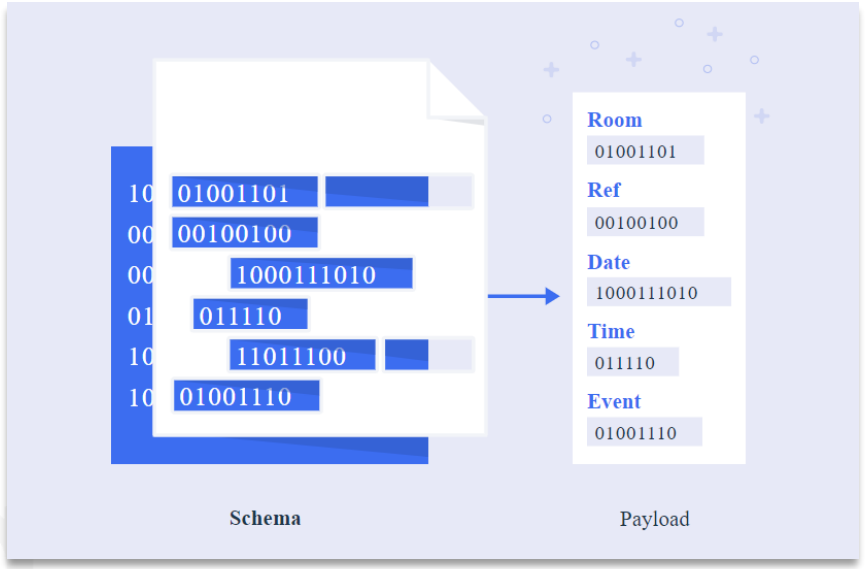


- Il «**Loose Coupling**» è un approccio allo sviluppo che ha come fine il ridurre al minimo le dipendenze tra le varie applicazioni coinvolte nello sviluppo per
  - Garantire che eventuali modifiche in un'applicazione non influiscano su altre applicazioni («evitare di sviluppare applicazioni strettamente accoppiate»)
  - Prevenire che qualsiasi cambiamento nelle specifiche possa «rompere o cambiare o in qualche modo influenzare» le funzionalità di altre applicazioni dipendenti (**regression**).



Common Interface Definitions

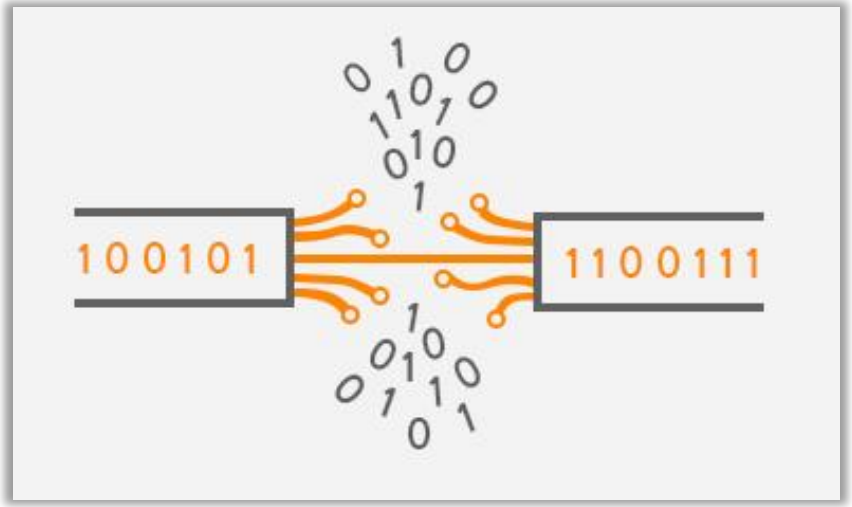
- Le CID garantiscono un formato di dati concordato comune per lo scambio tra le applicazioni.
- Aiuta a stabilire standard di scambio di messaggi tra le applicazioni.
- Assicura che eventuali migliorie pratiche di scambio di informazioni possano essere applicate facilmente.



- Esempio: scelta di Avro come formato dati per scambiare messaggi.
  - Avro è una buona scelta per gli scambi di messaggi in quanto serializza i dati in un formato binario compatto e supporta l'evoluzione dello schema.

Latency

- La latenza è il lasso di tempo che intercorre tra l'invio da parte del mittente fino e la relativa consegna al destinatario.



- Per la maggior parte delle applicazioni lo sviluppo è orientato ad ottenere una bassa latenza come requisito critico.

- Anche in una modalità di comunicazione asincrona, non è desiderabile un'elevata latenza poiché un ritardo significativo nella ricezione dei messaggi potrebbe tradursi in perdite significative per un'azienda.

Reliability

- L'affidabilità garantisce che l'indisponibilità temporanea delle applicazioni non influisca sulle applicazioni dipendenti che devono scambiare informazioni.

- In generale, quando l'applicazione di origine invia un messaggio all'applicazione remota, a volte l'applicazione remota potrebbe essere lenta o potrebbe non essere in esecuzione a causa di un errore.

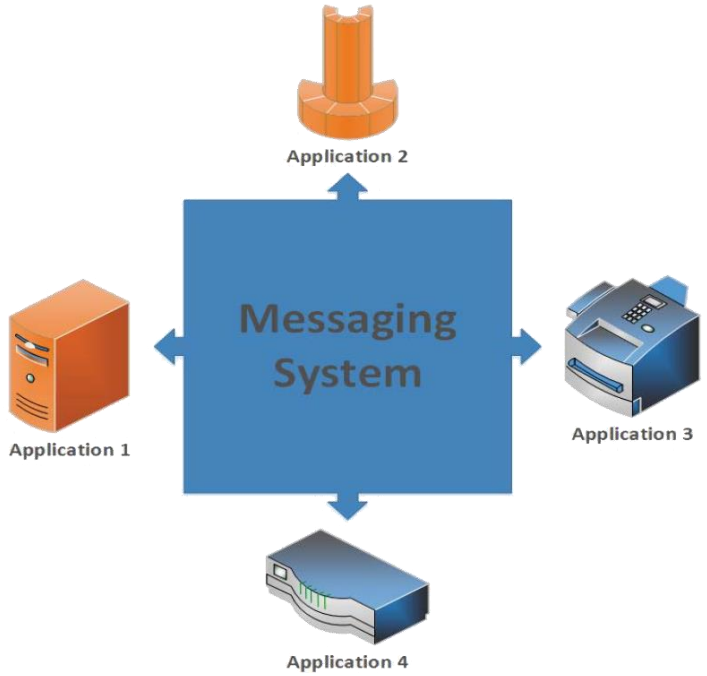
- La comunicazione affidabile e asincrona dei messaggi garantisce che l'applicazione di origine continui a funzionare e sia sicura che l'applicazione remota riprenderà l'attività in un secondo momento.

Comprensione dei sistemi di messaggistica

- Come illustrato nella figura, l'integrazione delle applicazioni basate su messaggi comporta applicazioni aziendali discrete che si colleghino a un sistema di messaggistica comune e inviino o ricevano dati ad esso.

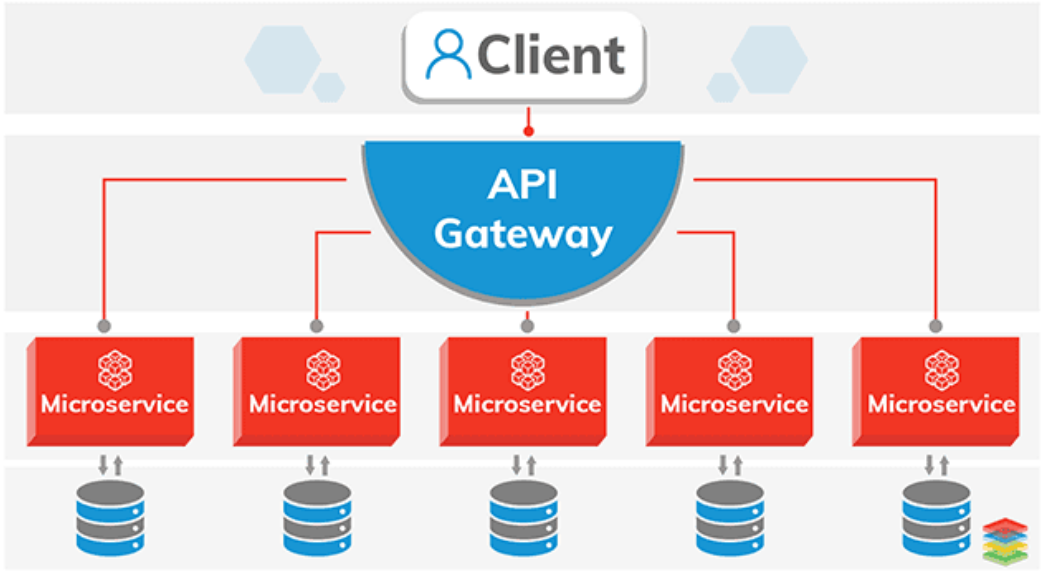
- Un sistema di messaggistica funge da componente di integrazione tra più applicazioni.

- Tale integrazione richiama comportamenti applicativi diversi basati su scambi di informazioni sull'applicazione.



Comprensione dei sistemi di messaggistica

- Le aziende hanno iniziato ad adottare architetture a micro-servizi con l'obiettivo di disaccoppiare sempre di più le proprie applicazioni.



- In tal modo esse comunicano tra loro in modo asincrono al fine di rendere la comunicazione più affidabile dato che le applicazioni non devono essere in esecuzione contemporaneamente.

- Un sistema di messaggistica aiuta a trasferire i dati da un'applicazione all'altra lasciando alle applicazioni la sola logica di gestione di ciò che devono condividere come dati piuttosto che sul come debbano essere condivisi.

«Message Queues»:

- Sono connettori (a volte canali) tra l'invio e la ricezione di apps.
- La loro funzione principale è quella di ricevere pacchetti di messaggi dall'app di origine e inviarli all'app ricevente in modo tempestivo e affidabile.

«Messages» (data packets):

- Un messaggio è un pacchetto di dati atomico che viene trasmesso su una rete a una coda di messaggi.
- L'app mittente suddivide i dati in pacchetti di dati più piccoli e li prepara come un messaggio con le informazioni di protocollo e intestazione e lo invia alla coda dei messaggi.
- In modo simile, un'applicazione ricevente riceve un messaggio ed estrae i dati dal wrapper di messaggi per elaborarlo ulteriormente.

«Sender» (detto anche Producer):

- E’ un’app che funge da fonte per i dati che devono essere inviati a una determinata destinazione.
- I Senders stabiliscono connessioni agli endpoint della coda dei messaggi e inviano i dati in pacchetti di messaggi più piccoli che aderiscono agli standard di interfaccia comuni.
- A seconda del tipo di sistema di messaggistica in uso, le app del mittente possono decidere di inviare i dati uno per uno o in un batch.

«Receiver» (detto anche Consumer):

- Sono i destinatari dei messaggi inviati dall'appl mittente.
- Estraggono i dati dalle code dei messaggi o ricevono dati dalle code dei messaggi attraverso una connessione persistente.
- Alla ricezione dei messaggi, estraggono i dati da quei pacchetti di messaggi e li usano per un'ulteriore elaborazione.

«Data transmission protocols»:

- Determinano le regole per governare gli scambi di messaggi tra app.
- Diversi sistemi di accodamento utilizzano protocolli di trasmissione dati diversi; dipende dall'implementazione tecnica degli endpoint di messaggistica.
- Kafka utilizza protocolli binari su TCP: il client avvia una connessione socket con le code Kafka e quindi scrive i messaggi insieme alla lettura del messaggio di riconoscimento.

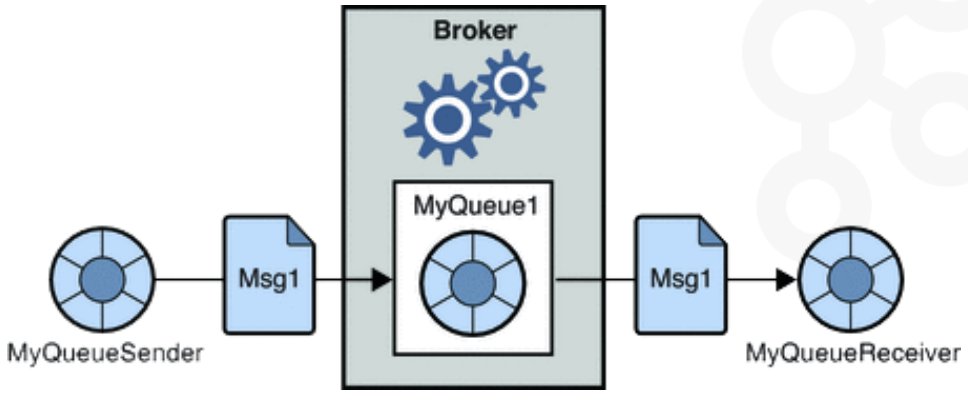
- Alcuni esempi di protocolli di trasmissione dei dati sono
  - AMQP (Advance Message Queuing Protocol)
  - STOMP (Streaming Text Oriented Message Protocol)
  - MQTT (Message Queue Telemetry Protocol)
  - HTTP (Hypertext Transfer Protocol)

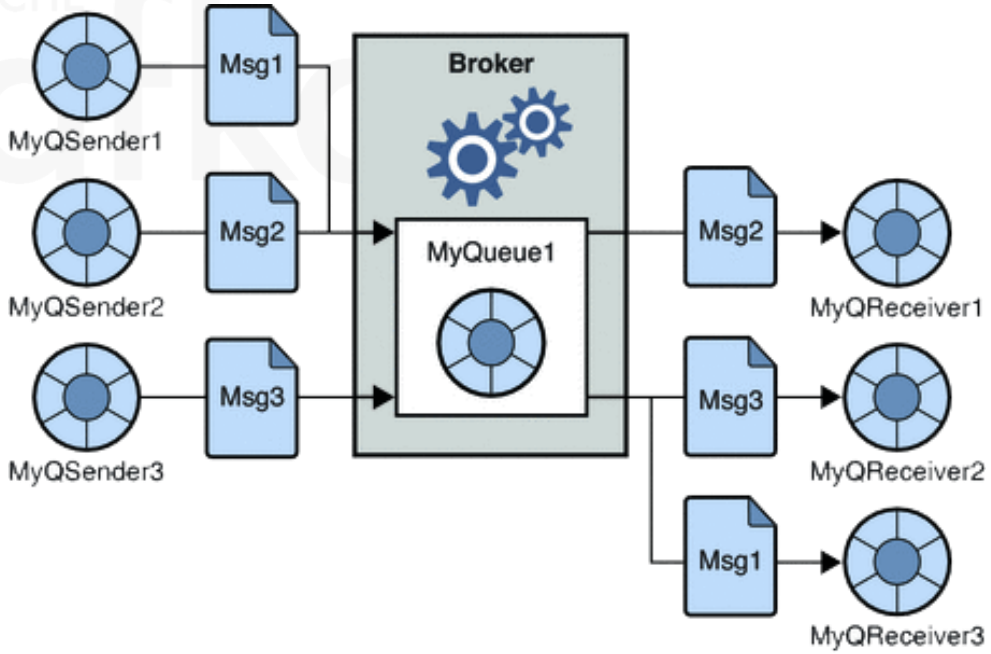
- Transfer mode:
  - La modalità di trasferimento in un sistema di messaggistica può essere intesa come il modo in cui i dati vengono trasferiti dall'applicazione di origine all'applicazione ricevente.
  - Esempi di modalità di trasferimento sono le modalità sincrone, asincrone e batch.

UNIT

Messaging Model

- I principali modelli di «**messaging**» sono:
  - Point-To-Point (PTP) Messaging System
  - Publish-Subscribe (Pub/Sub) Messaging System





PTP Model

- Il Message Producer è chiamato Sender e il Consumer è chiamato Receiver
- Sender e Receiver scambiano messaggi per mezzo di una destinazione chiamata coda.
- I mittenti producono messaggi in una coda e i destinatari consumano messaggi da questa coda.
- Ciò che distingue la messaggistica PTP è che un messaggio può essere consumato da un solo consumatore.
- È possibile che più Consumer ascoltino in coda per lo stesso messaggio, ma solo uno di loro lo riceverà.

PTP Model

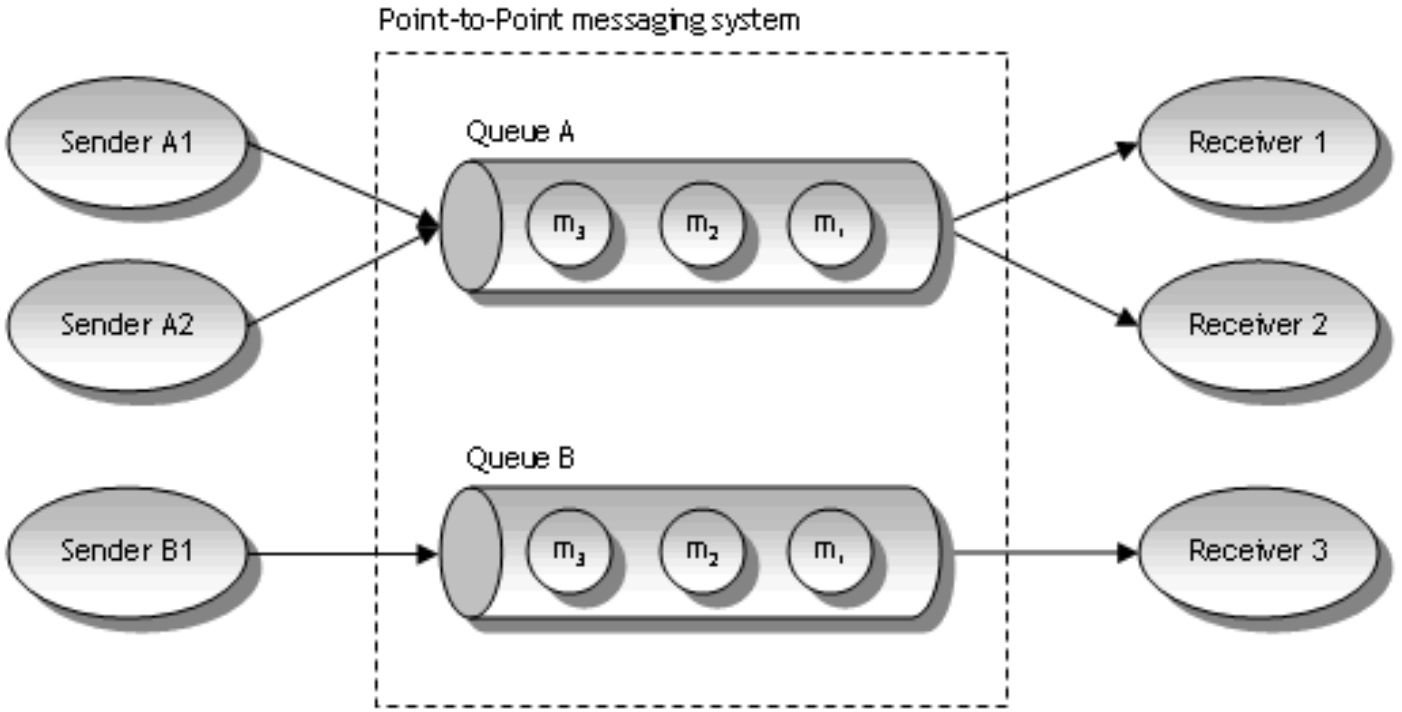
- Un modello PTP si basa sul concetto di invio di un messaggio a una named destination corrispondente all'endpoint della coda messaggi che ascolta i messaggi in arrivo su una porta.

- In genere, nel modello PTP, un destinatario richiede un messaggio che un mittente invia alla coda, piuttosto che iscriversi a un canale e ricevere tutti i messaggi inviati su una determinata coda.

- I modelli di messaggistica PTP gestiscono le code dei messaggi come liste FIFO.

PTP Model

- Nelle code, i messaggi vengono ordinati nell'ordine in cui sono stati ricevuti e, man mano che vengono consumati, vengono rimossi dall'intestazione della coda.



Nota: In Kafka le code mantengono gli offset dei messaggi; invece di eliminare i messaggi, incrementano gli offset per il destinatario. I modelli basati su offset offrono un supporto migliore per la riproduzione dei messaggi.

PTP Model

- Il modello di messaggistica PTP può essere ulteriormente classificato in due tipi:
  - Fire-and-forget model: il produttore si limita ad inviare un messaggio a una coda centralizzata senza attendere alcun tipo di riscontro relativamente alla consegna del messaggio e a chi l'abbia ricevuto; può essere utilizzato in uno scenario in cui si desidera attivare un'azione o inviare un segnale al ricevitore per attivare un'azione che non richiede una risposta.
  - Request/reply model: E' un modello asincrono di richiesta/risposta ove il mittente invia un messaggio su una coda e poi si pone in ascolto su un'altra coda in attesa della risposta dal destinatario. Il modello prevede un elevato grado di disaccoppiamento tra app mittente e app destinataria.

Publish-subscribe (Pub/Sub) Model

- In questo tipo di modello, un Subscriber (**sottoscrittore**) registra il proprio interesse per un particolare argomento o evento e viene successivamente informato dell'evento in modo asincrono.

- I Subscriber hanno la possibilità di esprimere il loro interesse per un evento o un modello di eventi e vengono successivamente informati di qualsiasi evento generato da un Publisher (Editore) che corrisponda al loro interesse registrato.

Publish-subscribe (Pub/Sub) Model

- È diverso dal modello di messaggistica PTP in quanto un argomento può avere più destinatari e ogni destinatario riceve una copia di ciascun messaggio.

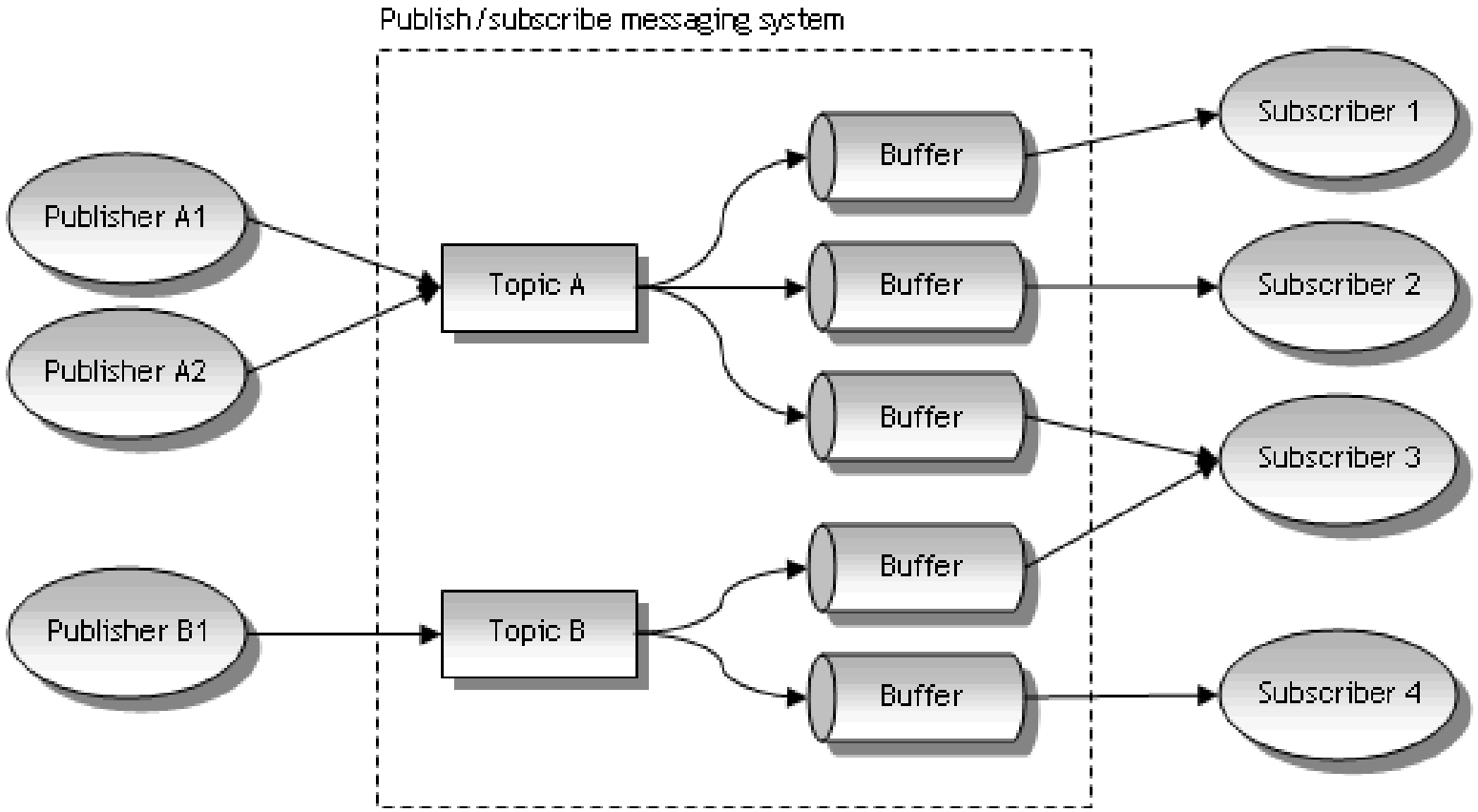
- In altre parole, un messaggio viene trasmesso a tutti i destinatari senza che debbano eseguire il polling dell'argomento.

- Nel modello PTP, il destinatario esegue il polling della coda per i nuovi messaggi.

Publish-subscribe (Pub/Sub) Model

- E’ utilizzato quando è necessario trasmettere un evento o un messaggio a molti consumer.
- A differenza del PTP, tutti i consumer (o subscriber) che ascoltano l'argomento riceveranno il messaggio.
- Garantisce un’alta interoperabilità delle piattaforma con i vari sistemi software che interagiscono con essa.
- Inoltre, i messaggi possono essere conservati nell'argomento (Topic) fino a quando non vengono recapitati ai subscriber attivi.

- Il modello prevede opzioni per avere subscription «durable» per consentire al subscriber di disconnettersi, riconnettersi e raccogliere i messaggi consegnati quando non era attivo.



NOTA: Kafka incorpora alcuni di questi importanti principi di progettazione.

UNIT

Protocolli

Advance Queuing Messaging Protocol (AQMP)

- Esistono diversi protocolli di trasmissione dei dati che utilizzano i messaggi come «oggetto» per lo scambio dati che possono essere trasmessi tra sender, receiver, e message queues.

- E’ importante comprendere come funzionino perché influiscono sulle decisioni di progettazione utili per la definizione dell’architettura di integrazione delle applicazioni orientate ai messaggi. AQMP è uno dei protocolli progettato per la messagistica.

- AQMP è un protocollo aperto per l'accodamento dei messaggi asincrono sviluppato e maturato nel corso di diversi anni.

Advance Queuing Messaging Protocol (AQMP)

- AMQP offre set ricchi di funzionalità di messaggistica che possono essere utilizzate per supportare scenari *molto avanzati*.

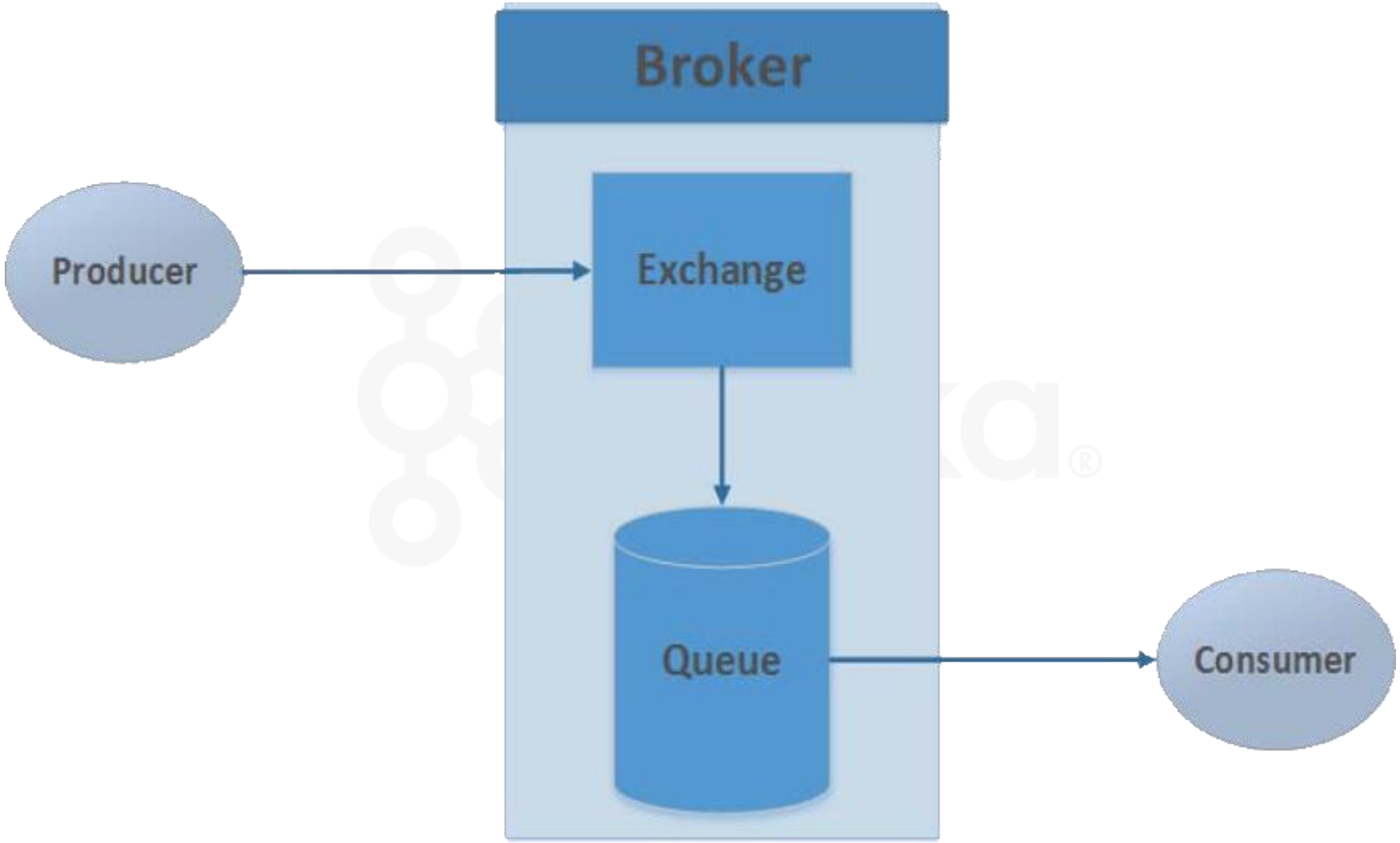
- Il protocollo prevede tre componenti principali:

  - Publisher(s)
  - Consumer(s)
  - Broker/Server(s)

- Ogni componente può essere replicato e installato su host indipendenti.

- Publishers e Consumers comunicano tra loro attraverso le code dei messaggi destinate agli scambi all'interno dei broker. Il recapito è affidabile, garantito e la consegna è di tipo «in-order».

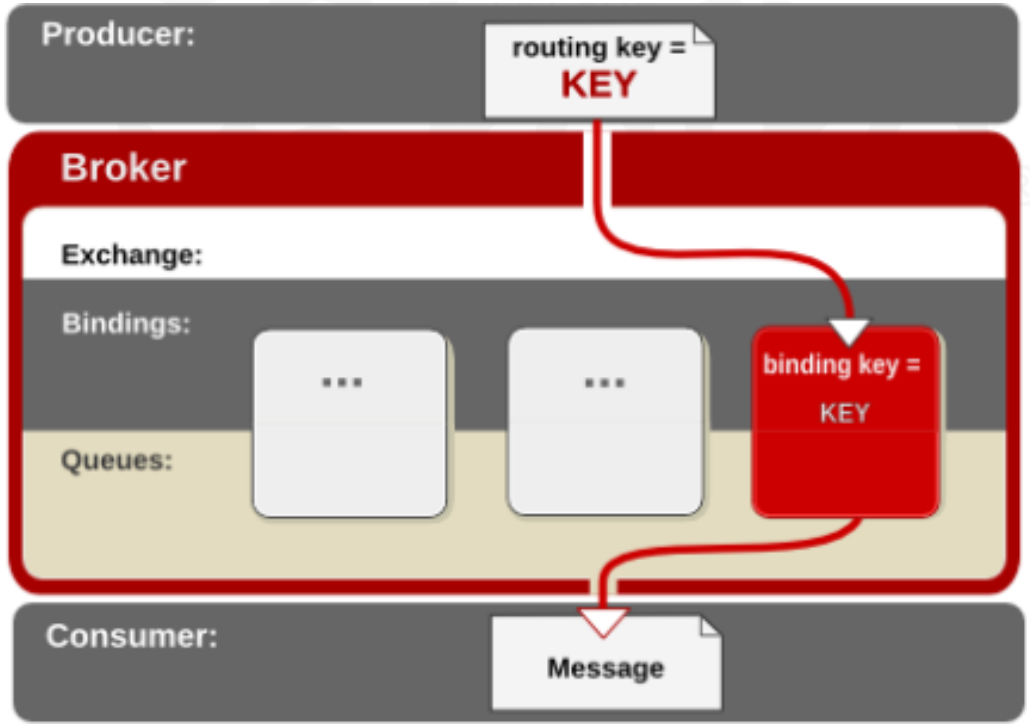
Advance Queuing Messaging Protocol (AQMP)



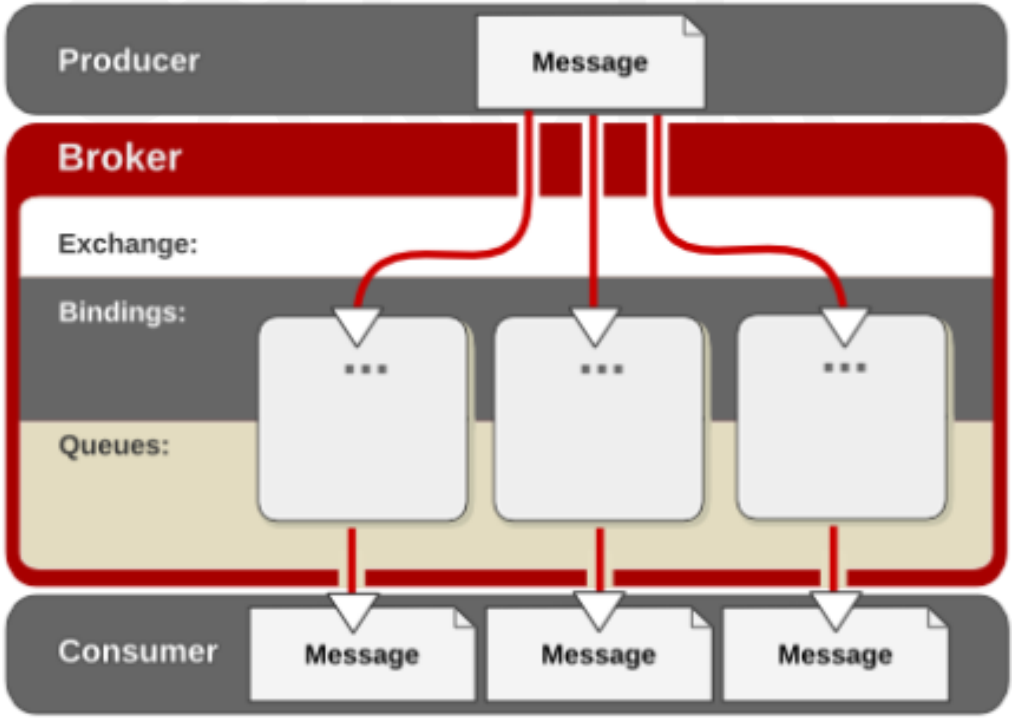
Advance Queuing Messaging Protocol (AQMP) - Direct exchange

- Lo scambio di messaggi può avvenire in diversi modi.

- Direct exchange: è un meccanismo di routing key-based; in pratica un messaggio viene recapitato alla coda ove il suo identificativo è uguale alla chiave di routing del messaggio.

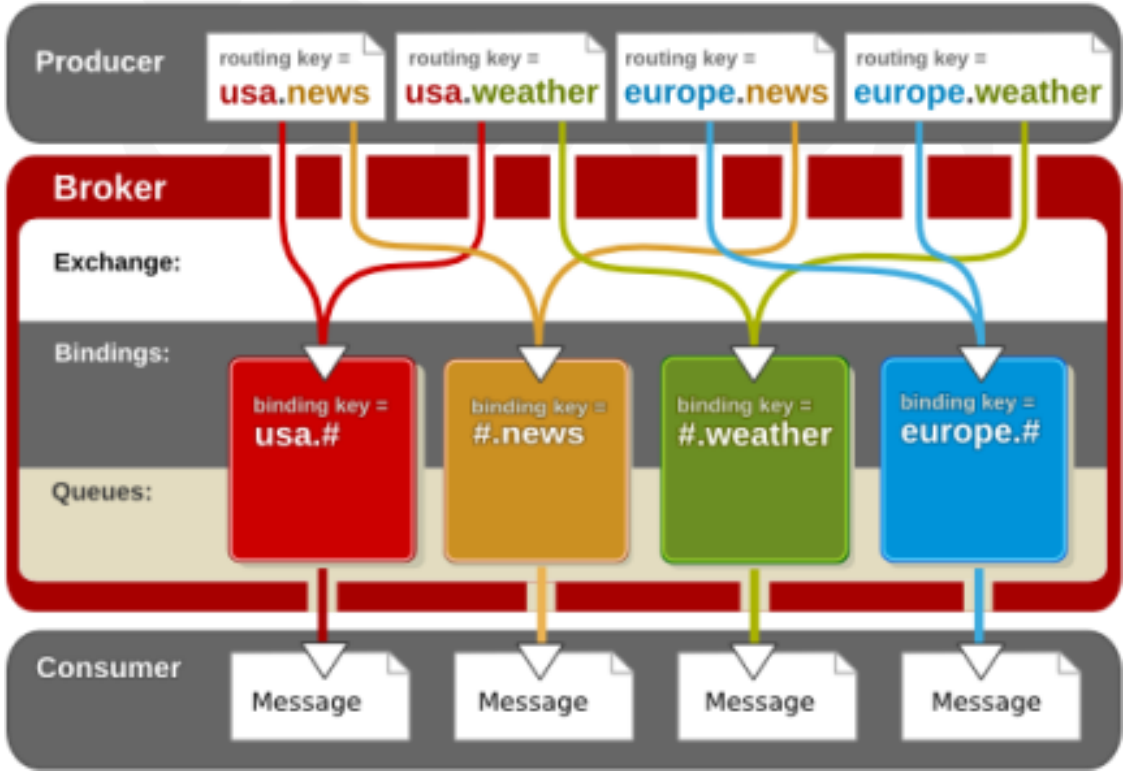


- Fan-out exchange: uno scambio di fan-out instrada i messaggi a tutte le code ad esso associate e la chiave di routing viene ignorata.
  - Se N code sono associate a uno scambio di fan-out, quando un nuovo messaggio viene pubblicato, una copia del messaggio viene recapitata a tutte le N code.
  - Gli scambi di fan-out sono ideali per il routing di trasmissione dei messaggi.
  - In altre parole, il messaggio viene clonato e inviato a tutte le code collegate a questo scambio.



Advance Queuing Messaging Protocol (AQMP) - Topic exchange

- Topic exchange: nello scambio di argomenti, il messaggio può essere instradato ad alcune delle code collegate usando i caratteri jolly (wildcards).
  - Il tipo di scambio di argomenti viene spesso utilizzato per implementare varie varianti del modello di publish/subscribe (pubblicazione/sottoscrizione).
  - Gli scambi di argomenti sono comunemente utilizzati per l'instradamento multicast di messaggi.



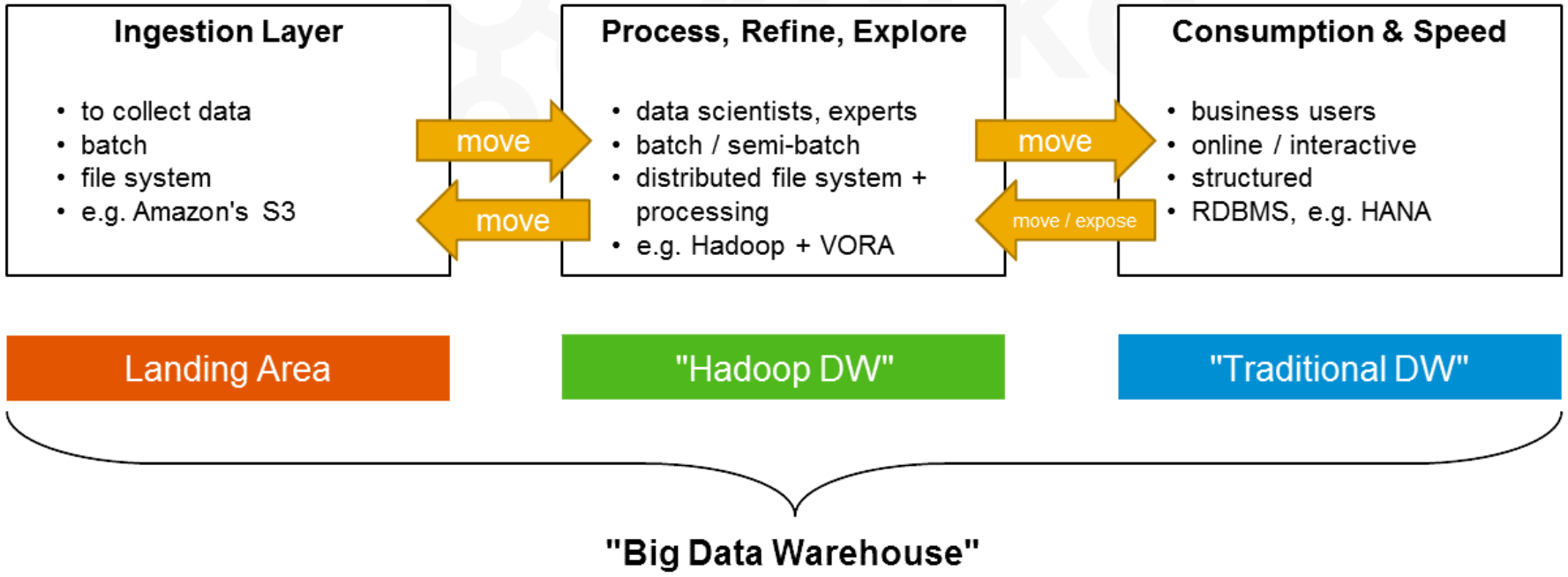
UNIT

Uso di messaging systems in applicazioni di big data streaming

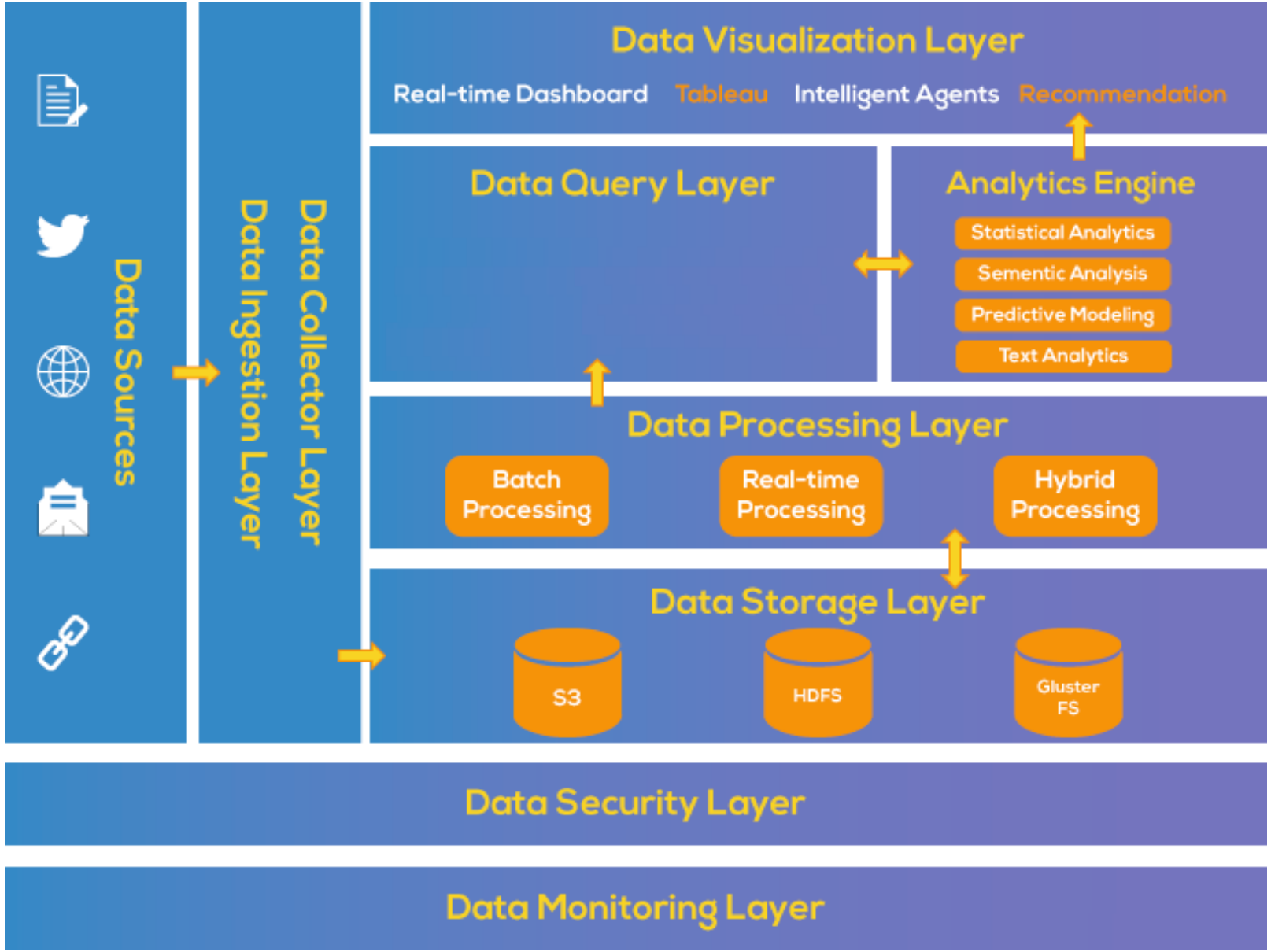
Layer di un'applicazione per big data

- Si parla di big data quando si ha un insieme talmente grande e complesso di dati che richiede la definizione di nuovi strumenti e metodologie per estrapolare, gestire e processare informazioni entro un tempo ragionevole.

- In ambito Big Data, le evoluzioni architetturali hanno portato alla definizione di architetture complesse (come mostrato nella slide successiva) e di cui discuteremo alcuni aspetti.



Layer di un'applicazione per big data



Ingestion layer

- I dati di input richiesti per l'elaborazione vengono «ingeriti» in sistemi di archiviazione; possono esservi molte fonti di dati per le quali è necessario eseguire la stessa o diversa elaborazione.

- Il livello di acquisizione elabora i dati in entrata, dando la priorità alle fonti, convalidando i dati e instradandoli nella posizione migliore per essere memorizzati ed essere pronti per l'accesso immediato.

- L'estrazione dei dati può avvenire in un singolo batch di grandi dimensioni o suddiviso in più piccoli; il livello di inserimento dati sceglierà il metodo in base alla situazione.

- Il layer dà la priorità a un tempo di caricamento più rapido e a ciò che è meglio per il programma.

Ingestion layer

- I dati di input richiesti per l'elaborazione vengono «ingeriti» in sistemi di archiviazione; possono esservi molte fonti di dati per le quali è necessario eseguire la stessa o diversa elaborazione.

- Il livello di acquisizione elabora i dati in entrata, dando la priorità alle fonti, convalidando i dati e instradandoli nella posizione migliore per essere memorizzati ed essere pronti per l'accesso immediato.

- L'estrazione dei dati può avvenire in un singolo batch di grandi dimensioni o suddiviso in più piccoli; il livello di inserimento dati sceglierà il metodo in base alla situazione.

- Il layer dà la priorità a un tempo di caricamento più rapido e a ciò che è meglio per il programma.

Data Collector Layer

- In questo livello, maggiore attenzione è rivolta al trasporto di dati dal livello di importazione al resto della pipeline di dati.

- È lo strato, in cui i componenti sono disaccoppiati in modo che possano inizializzare le capacità analitiche.

Processing layer

- Contiene la business logic che elabora i dati ricevuti nel livello di importazione e applica trasformazioni per portare i dati stessi in una forma utilizzabile.
- Il suo scopo è quindi quello di convertire i dati grezzi in informazioni.
- Possono esistere più applicazioni di elaborazione per dati uguali o diversi.
- Ogni applicazione può avere una logica e capacità di elaborazione diverse.

Consumption layer

- Questo livello contiene i dati elaborati dal livello di elaborazione.
- I dati elaborati sono un unico punto di verità e contengono informazioni importanti per le decisioni aziendali.
- Possono esistere più consumatori che possono utilizzare gli stessi dati per scopi diversi o dati diversi per lo stesso scopo.

Uso di messaging systems in big data streaming applications

- Le applicazioni di streaming probabilmente rientrerebbero nel secondo livello, il livello di elaborazione.

- Gli stessi dati possono essere utilizzati da molte applicazioni contemporaneamente e possono sussistere diversi modi di fornirli alle applicazioni.

- Pertanto, le applicazioni possono essere in streaming, batch o micro-batch.

- Tutte le applicazioni coinvolte consumano i dati in diversi modi:

  - le applicazioni di streaming possono richiedere dati come flusso (stream) continuo
  - le applicazioni batch possono richiedere dati come batch.

Uso di messaging systems in big data streaming applications

- In questo ambito possono esservi molteplici casi d'uso sia per i producer che per i consumer di dati; il contesto favorisce quindi l'adozione di un sistema di messaggistica.

- Lo stesso messaggio può essere utilizzato da più consumer, quindi è necessario conservare il messaggio fino a quando tutti i consumer lo abbiano ricevuto.

- L'esigenza è quella di avere un sistema di messaggistica in grado di

  - Conservare i dati fino a quando non li vogliamo
  - Offrire un alto grado di tolleranza agli errori
  - Fornire un modo diverso di consumare flussi di dati, batch e micro-batch

Punti di forza di un messaging systems - High consuming rate

- Le origini dati in streaming possono avere varia natura e sono inquadrati in un contesto in cui la velocità di produzione dei messaggi è troppo elevata.

- È possibile che le applicazioni di *streaming* possano o meno richiedere un consumo simile.

- È possibile che si desideri disporre di una coda di messaggistica in grado di utilizzare i dati a una velocità maggiore.

Punti di forza di un messaging systems - Guaranteed delivery

- Alcune applicazioni di streaming non possono permettersi di perdere messaggi;

- Abbiamo bisogno di un sistema che garantisca la consegna dei messaggi all'applicazione di streaming ogni volta che è necessario.

Punti di forza di un messaging systems - Persisting capability

- Possono sussistere casi di più applicazioni che consumano dati simili ma a una velocità diversa.

- È possibile che si desideri disporre di un sistema di messaggistica che conservi i dati per un periodo di tempo e li fornisca a un'applicazione diversa in modo asincrono.

- Questo aiuta a disaccoppiare tutte le applicazioni e a progettare un'architettura a micro-servizi.

Punti di forza di un messaging systems - Security

- Alcune applicazioni richiedono sicurezza sui dati che consumano
- Applicazioni potrebbero non voler condividere, come specifica, alcuni dati con altre che utilizzano lo stesso sistema di messaggistica.
- Si desidera un sistema che garantisca la sicurezza in generale.

Punti di forza di un messaging systems - Fault tolerance

- Le applicazioni non possono poggiare su sistemi che non recapitino messaggi o dati ogni qualvolta che li richiedono.

- Un sistema deve garantire la tolleranza agli errori e, in particolare, a fronte del fallimento di un server non vi deve essere perdita di informazioni e quindi i messaggi dovranno e potranno essere consegnati lo stesso.

UNIT 2

Il Messaging

- Parlare di messaging «sembra tanto anni Novanta», eppure molte delle soluzioni e delle architetture «moderne» hanno alla base delle applicazioni frammentate in microservizi dove le soluzioni per far convivere l'IT vecchia e nuova sono basate concettualmente proprio sui paradigmi del messaging (…ovviamente convertiti alla visione del Terzo Millennio…)

- Apache Kafka nasce proprio come soluzione «fatta in casa» per collegare parti diverse dell'IT aziendale in seno a LinkedIn da un'idea di un gruppo di sviluppatori del social network stesso.

- Siamo nel 2008 e l'esigenza da risolvere è combinare la grande quantità di dati che arrivano sulla piattaforma online con le funzioni di analytics.

- La parte transazionale è veloce e quella analitica anche; il problema da risolvere «è il miglioramento della comunicazione fra le due componenti».

- Gli sviluppatori di Likedin pensano che a collegarle debba essere un sistema di messaging che, nelle intenzioni, dovrebbe supportare le varie applicazioni di LinkedIn.

- Sul mercato non trovano soluzioni adeguate, da qui l'idea di creare un prodotto interno nuovo ovvero Kafka.

- Kafka nasce dalla ricerca di superare i limiti sperimentati in LinkedIn dai Team di Sviluppo.

- E’ sempre un sistema di messaging ma secondo i suoi creatori è stato progettato col fine di gestire un flusso continuo di messaggi in grandi volumi

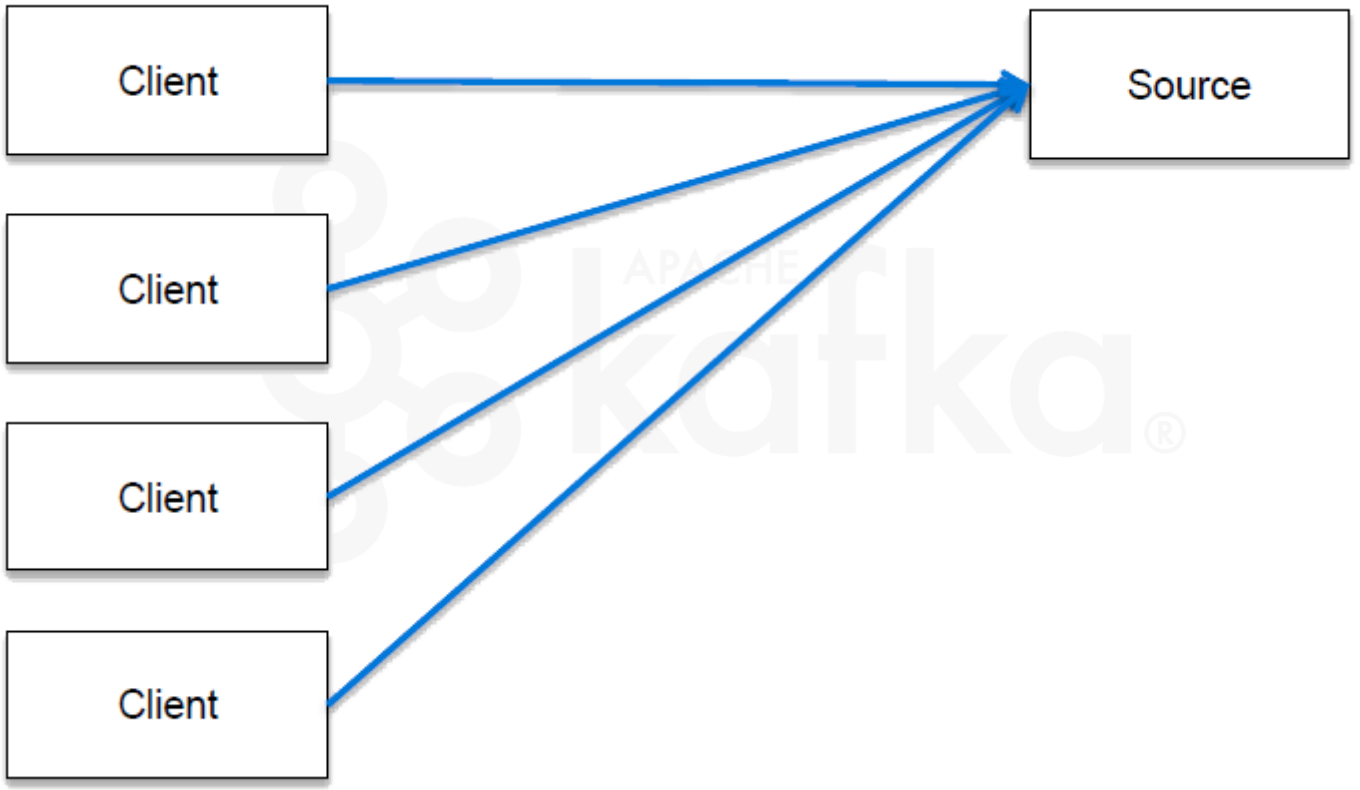
- Un’esigenza che le tradizionali piattaforme di messaging non si erano poste e che rende invece il nuovo progetto adatto ad applicazioni come quelle di IoT e tutte quelle in real-time.

- I suoi progettisti lo hanno pensato con natura estremamente distribuita e con lo storage dei messaggi (necessario a garantirne l’arrivo anche in infrastrutture in cui non sempre i nodi riceventi sono attivi).

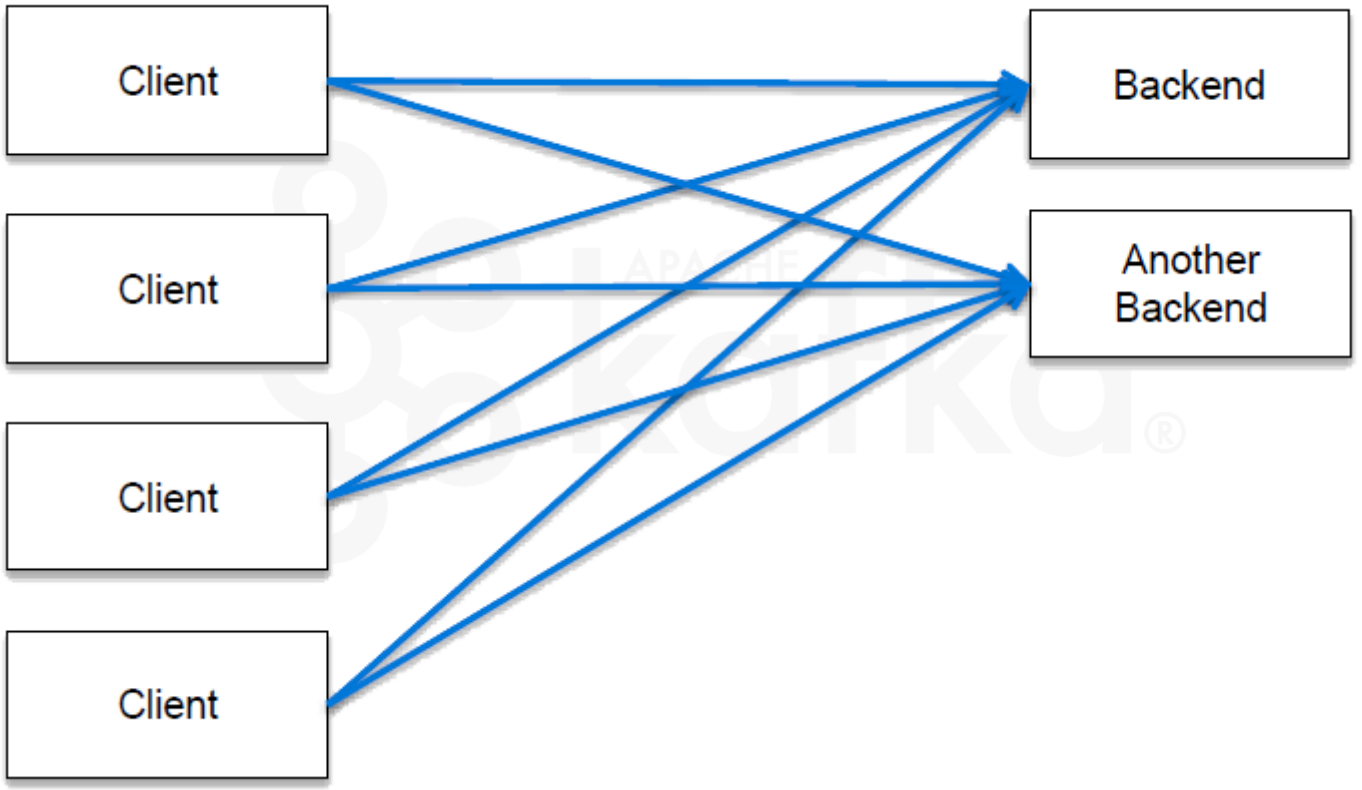
- Inoltre hanno previsto la possibilità di operare direttamente sui messaggi con funzioni di organizzazione, trasformazione e selezione senza richiedere altri elementi esterni.

Il problema di Linkedin

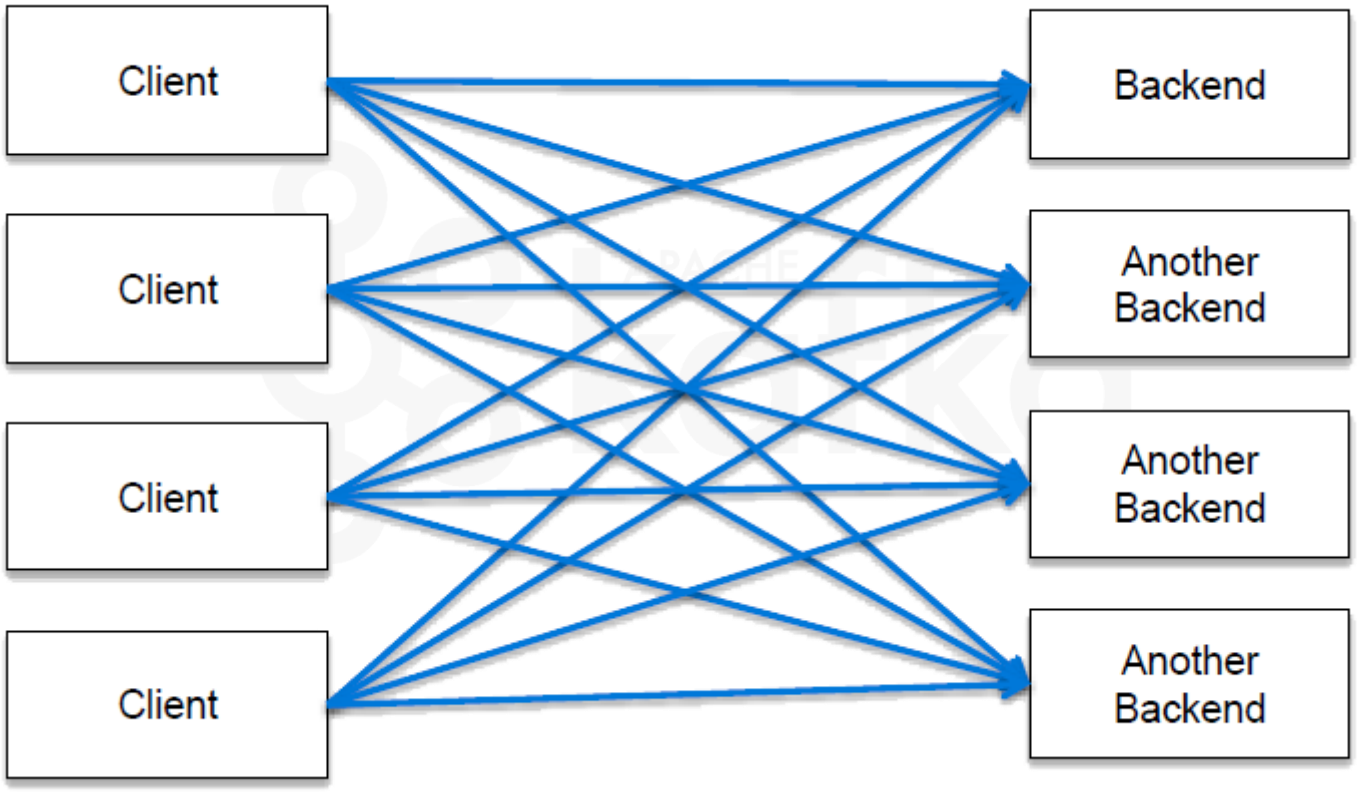
Il problema di Linkedin



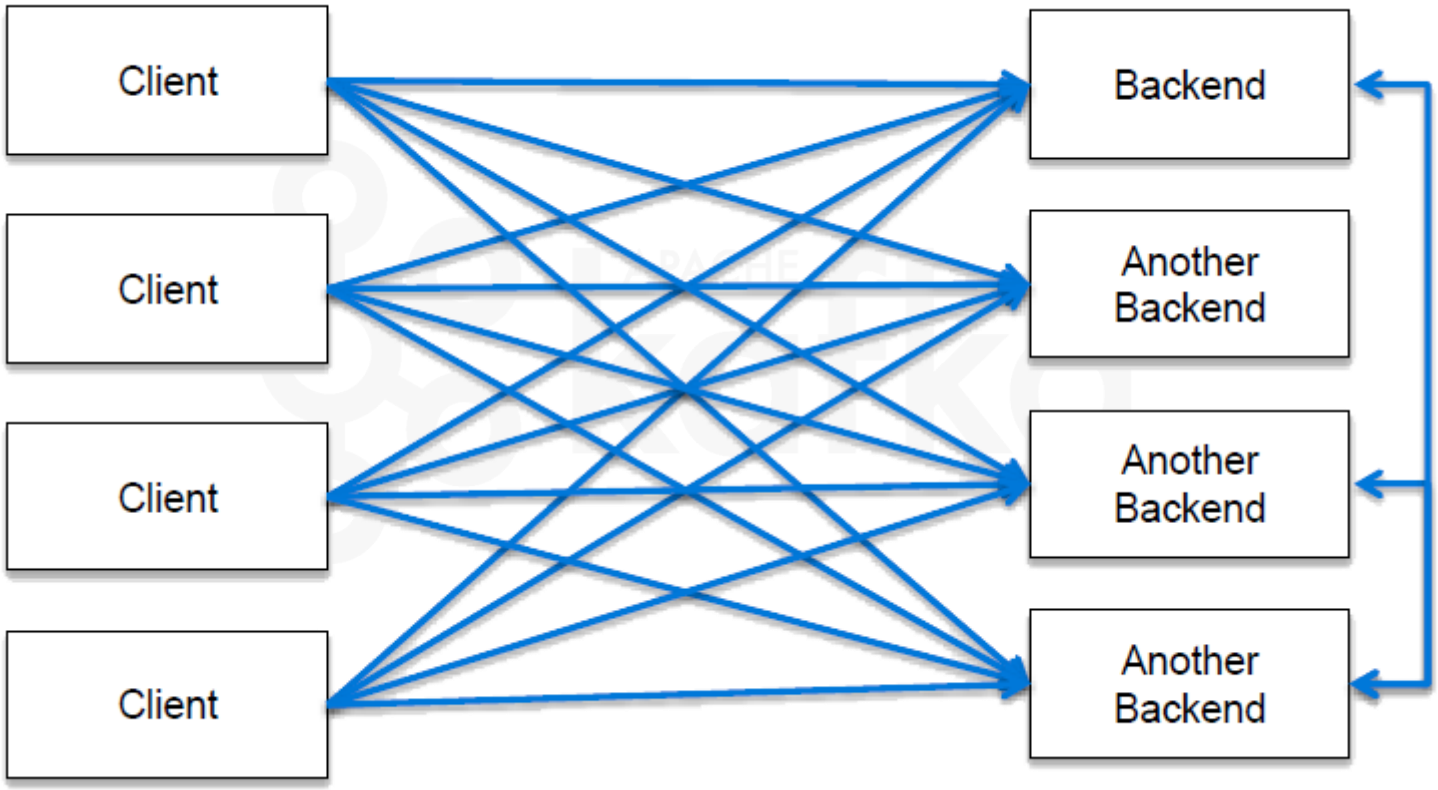
Il problema di Linkedin



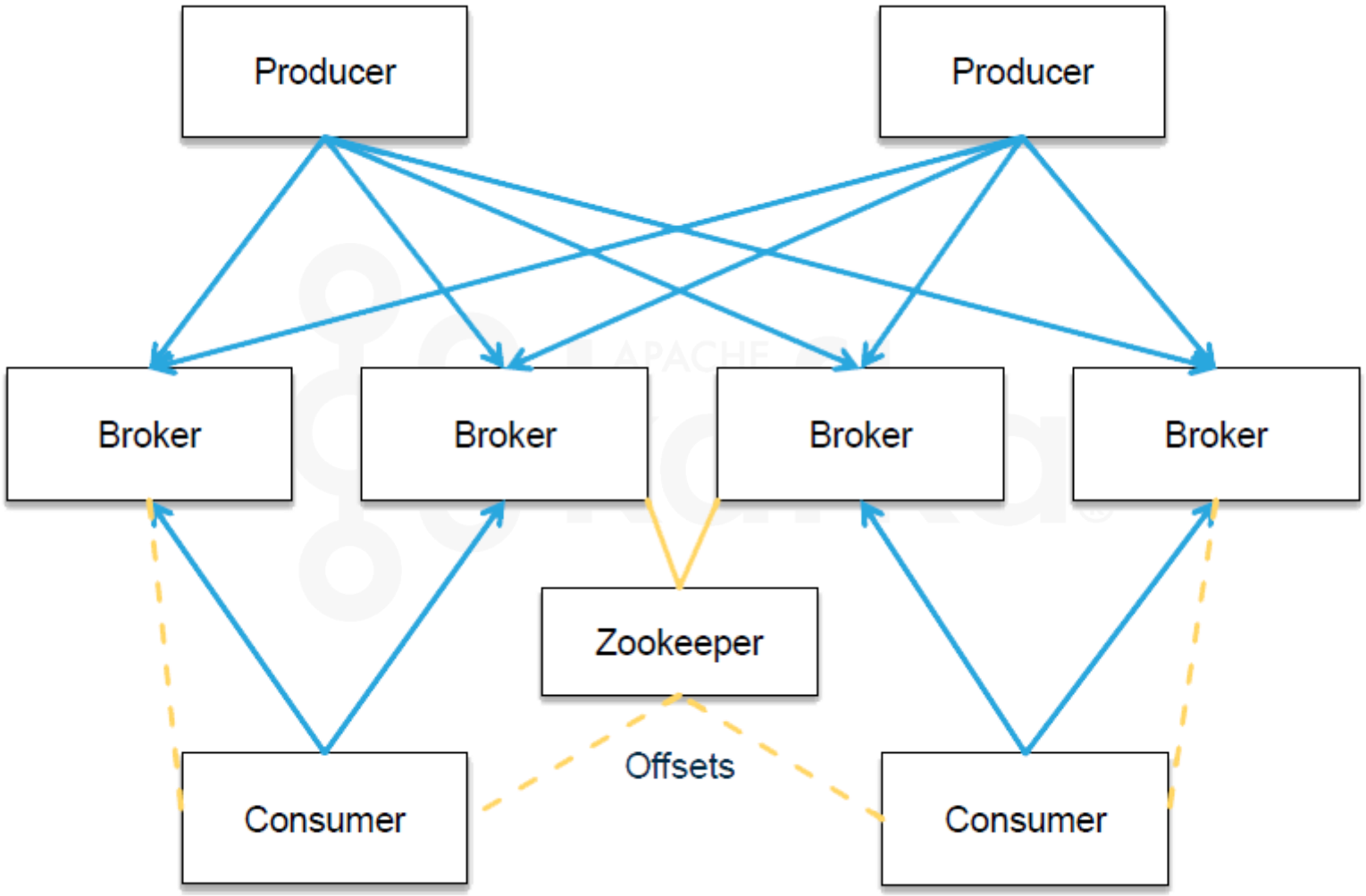
Il problema di Linkedin



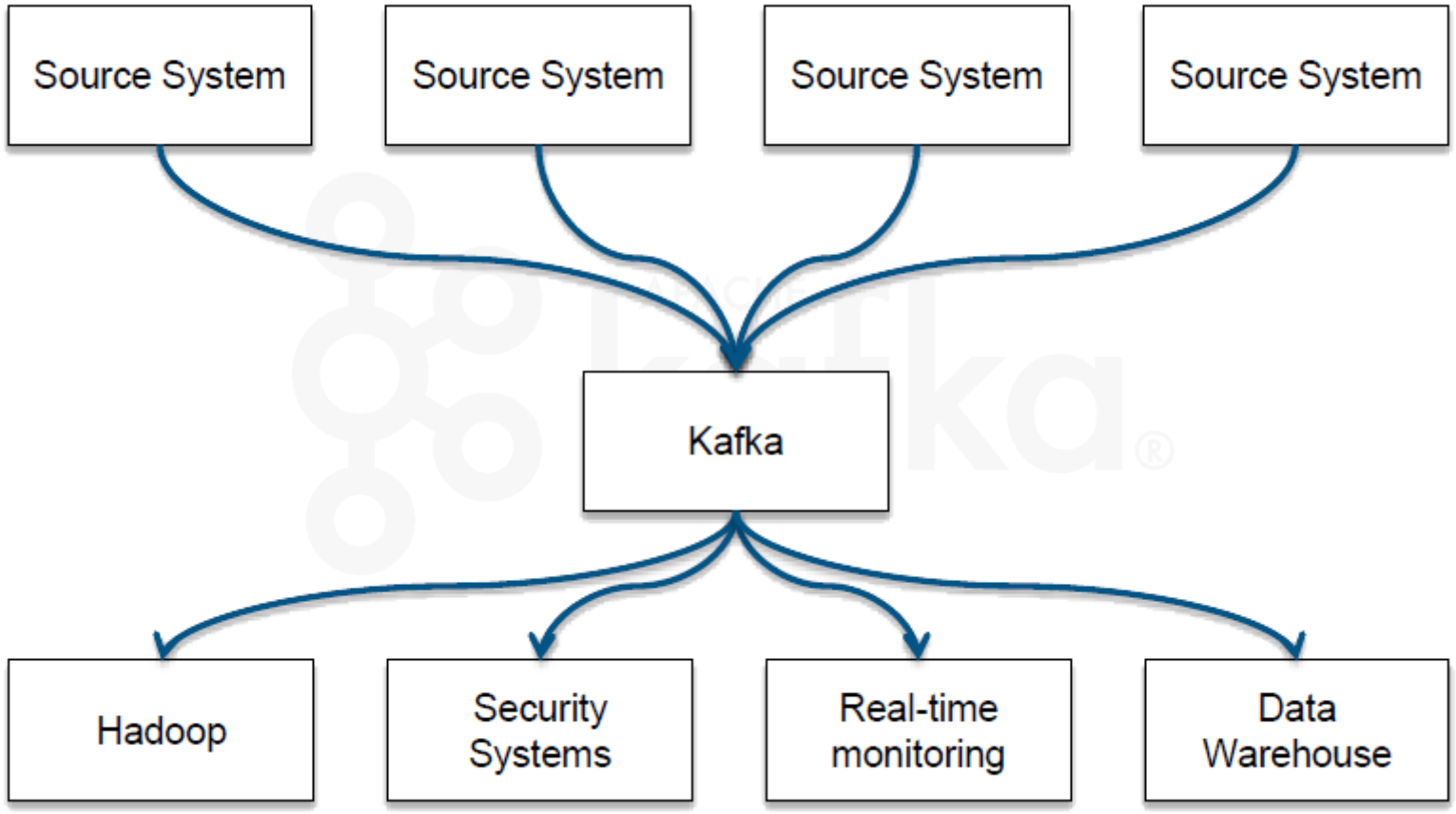
Il problema di Linkedin

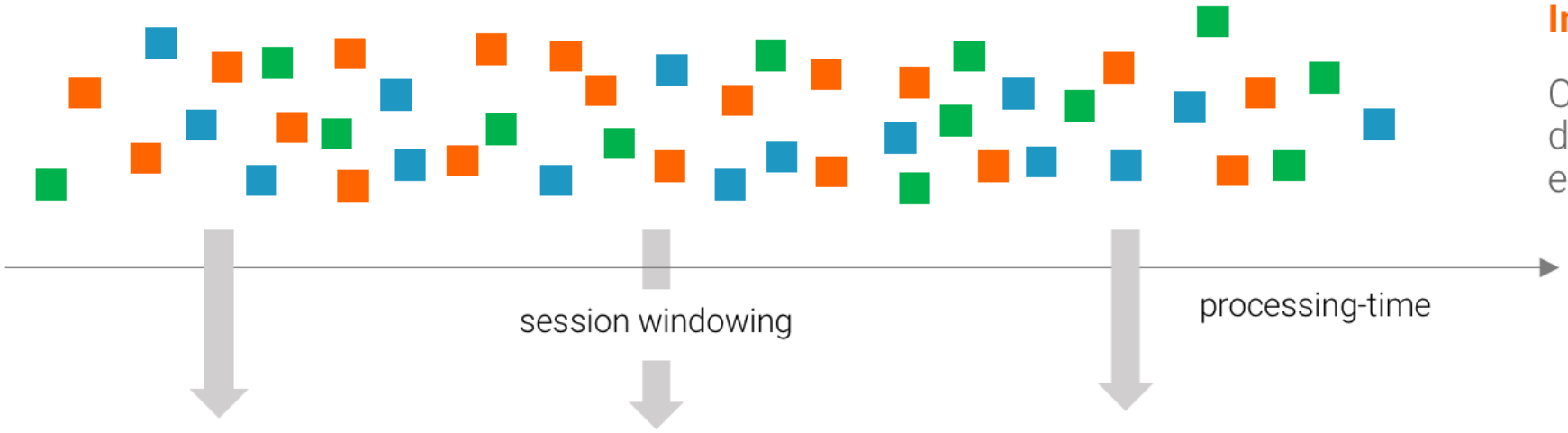


La soluzione di Linkedin



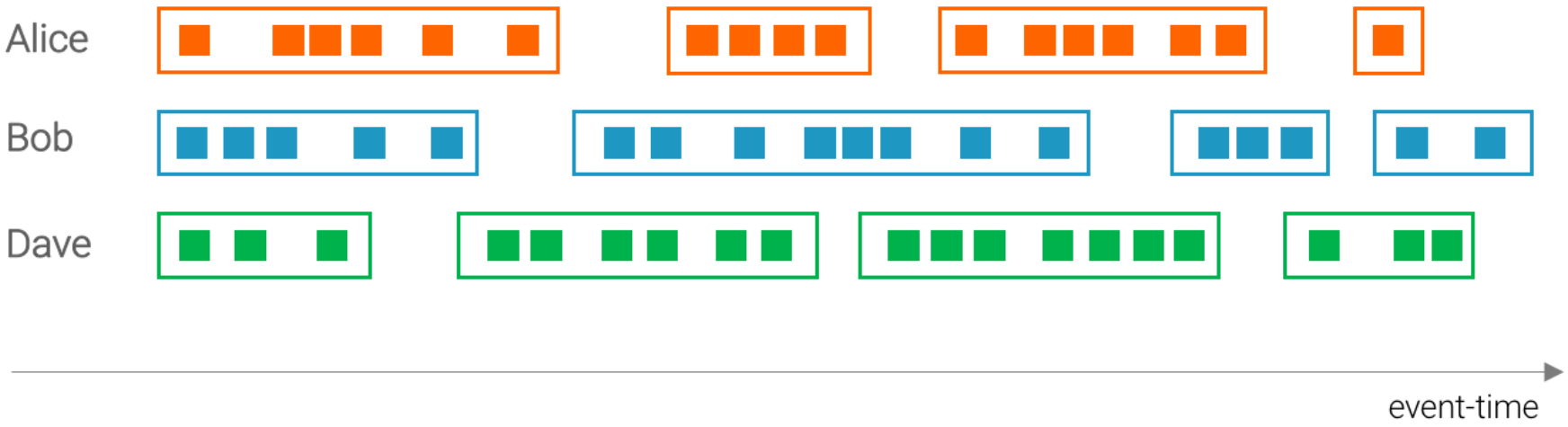
Kafka Enterprise Architecture





Input data

Colors represent different users event



Results

User sessions, grouped by event-time session windows

L’importanza dello streaming

- Per i creatori di Kafka (che hanno poi fondato una loro società, Confluent) è la **gestione dello streaming dei dati** il punto chiave per il futuro della piattaforma.

- L’idea è che gran parte delle imprese si stia muovendo verso ambienti con streaming di dati in tempo reale e questo permetta a Kafka, con le sue varie evoluzioni, di diventare un asse portante per tutto lo sviluppo applicativo di queste realtà.

- Anche la transizione ai microservizi viene considerata un fattore di accelerazione per Kafka (non in tutte le realtà, ma laddove i microservizi vengono usati per la gestione di eventi asincroni certamente sì).

L’importanza dello streaming in ecosistemi a microservizi

- Avere servizi asincroni introduce la possibilità che eventi o informazioni si perdano, una piattaforma nata proprio per garantire la certezza della consegna dei messaggi risolve il problema (o perlomeno intende farlo).

- Nell’evoluzione di Kafka, Confluent ha aggiunto il passo in più (in termini di componenti e soluzioni) che ne sta agevolando la diffusione.

- Confluent propone, inoltre, «Kafka as a Service» attraverso un servizio cloud, togliendo alle aziende utenti il peso dell’implementazione e portandole direttamente alla fase di sviluppo.

UNIT

Kafka

Apache Software Foundation

- L'Apache Software Foundation (**ASF**) è una fondazione no-profit, costituita nel giugno 1999
- La storia di ASF è collegata al suo web server HTTP Apache, il cui sviluppo è iniziato nel 1994
- In pratica un gruppo di otto sviluppatori (in seguito definito Apache Group) iniziò a lavorare al miglioramento del demone HTTPd dell'NCSA

- È una community distribuita di sviluppatori che lavorano su progetti software open source.

- I progetti sono caratterizzati da un processo di sviluppo distribuito, collaborativo e basato sul consenso molto simile al progetto wikipedia; ciascun progetto è gestito da un team di volontari che sono i contributori attivi al progetto stesso.

Licenza Apache

- La Licenza Apache (d'ora in poi definita semplicemente con **AL** = Apache License) è una licenza di software libero non copyleft scritta dalla **ASF** che obbliga gli utenti a preservare l'informativa di diritto d'autore e d'esclusione di responsabilità nelle versioni modificate.

- Tutti i software realizzati dalla ASF sono licenziati secondo i termini della LA.

- Qualsiasi azienda può rilasciare software adottando la LA come modello.

Licenza Apache

- Consente agli utenti di:
  - Usare il software per ogni scopo
  - Distribuirlo
  - Modificarlo
  - Distribuire versioni modificate dello stesso

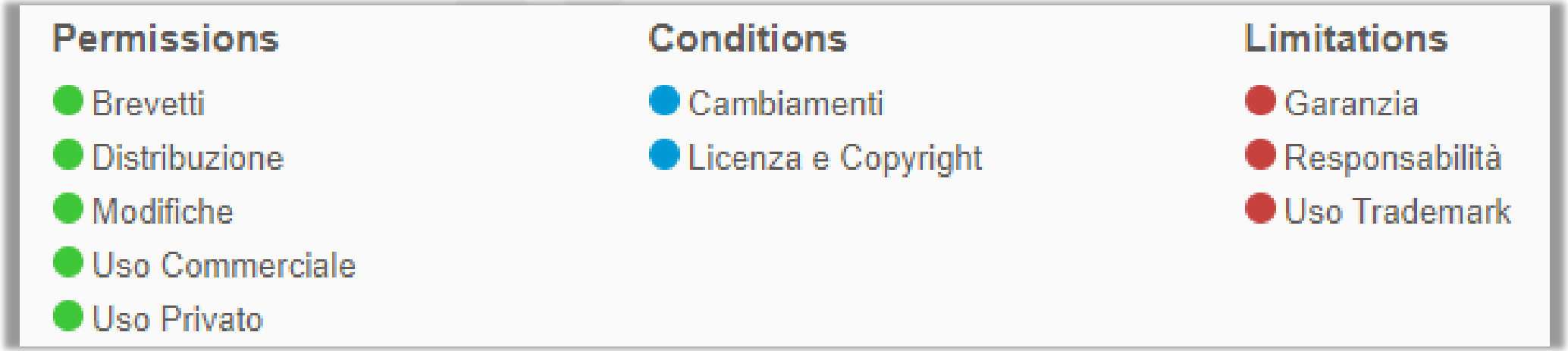
- Non richiede che versioni modificate del software siano distribuite secondo i termini della stessa licenza o come software libero ma solo che si includa un'informativa del fatto che si è utilizzato software licenziato secondo i termini della AL.

Licenza Apache

- E’ una licenza permissiva la cui principale condizione è quella di preservare le note sul copyright e la licenza stessa.

- Ai contributori è fornita una esplicita concessione sui brevetti.

- L'opera, le sue modifiche, o lavori più estesi basati su di essa **possono essere distribuiti con una licenza diversa**, anche proprietaria e anche senza codice sorgente.



Licenza Apache

- Comparazione licenze: https://www.linux.it/scegli-una-licenza/licenses/

- **Nota**: L'Apache Software Foundation e la Free Software Foundation (FSF) hanno concordato che la Licenza Apache 2.0 è una licenza di software libero compatibile con la versione 3 della GNU General Public License (GPL), il che significa che il codice sotto licenza GPL versione 3 e Apache License 2.0 possono essere combinati, ed il codice risultante è sotto licenza GPL versione 3.

- **Apache Kafka** è «un sistema di **messaggistica di pubblicazione-sottoscrizione** ad alta produttività (high throughput) implementato come servizio di registro di commit distribuito, partizionato e replicato».

- In altri termini è un sistema open source di **messaggistica istantanea** che consente la gestione di un elevato numero di operazioni in tempo reale da migliaia di client, sia in lettura che in scrittura.

- La piattaforma si dimostra ideale per la progettazione di applicazioni di alta fascia.

- Apache Kafka è una piattaforma di stream-processing distribuita scritta in Scala e in Java, inizialmente sviluppata da LinkedIn e divenuta open source a inizio 2011

- Tra i punti di forza di Kafka vi sono l'incremento della produttività e l'affidabilità, fattori che hanno consentito al sistema di sostituire famosi broker di messaggistica come JMS e AMQP.

- Per un rapido apprendimento di Kafka è preferibile possedere una conoscenza approfondita di

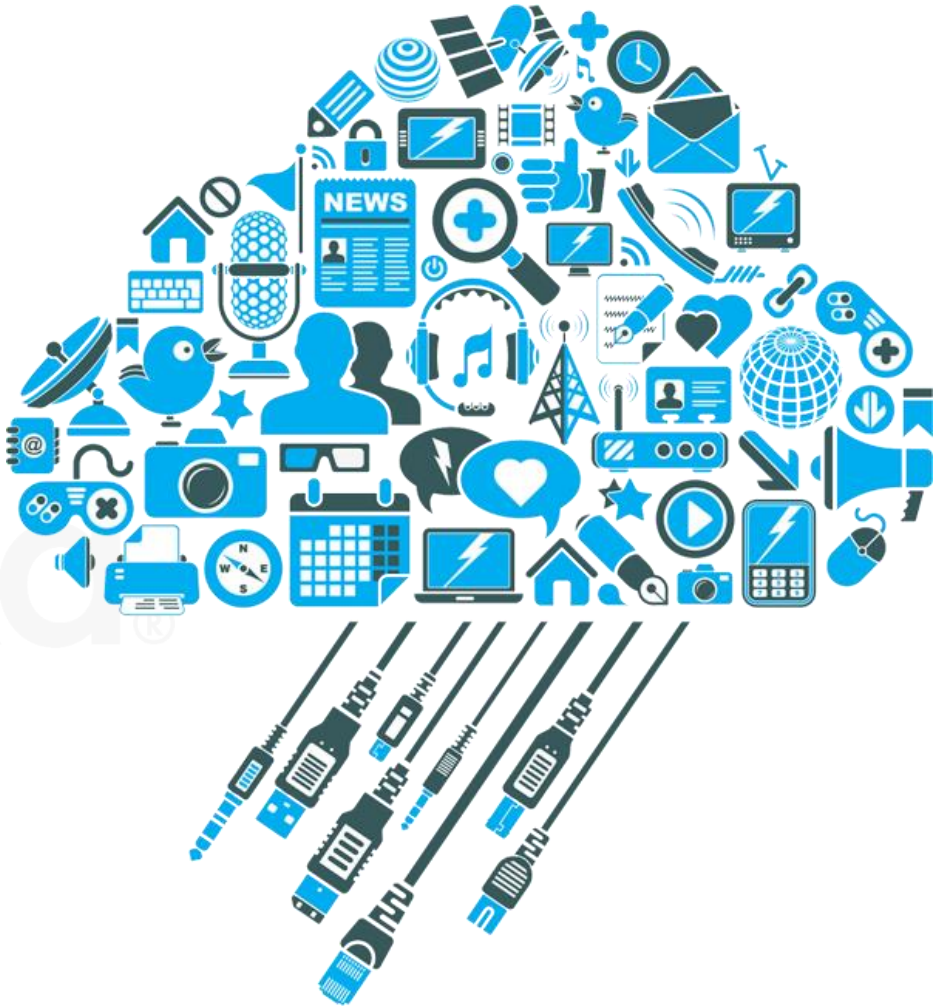
  - Java
  - Scala
  - Linux.

Scalable Language

Caratteristiche salienti di Kafka (Tratte dal sito ufficiale)

- **Veloce**: Un singolo broker Kafka può gestire centinaia di megabyte di letture e scritture al secondo da migliaia di client.

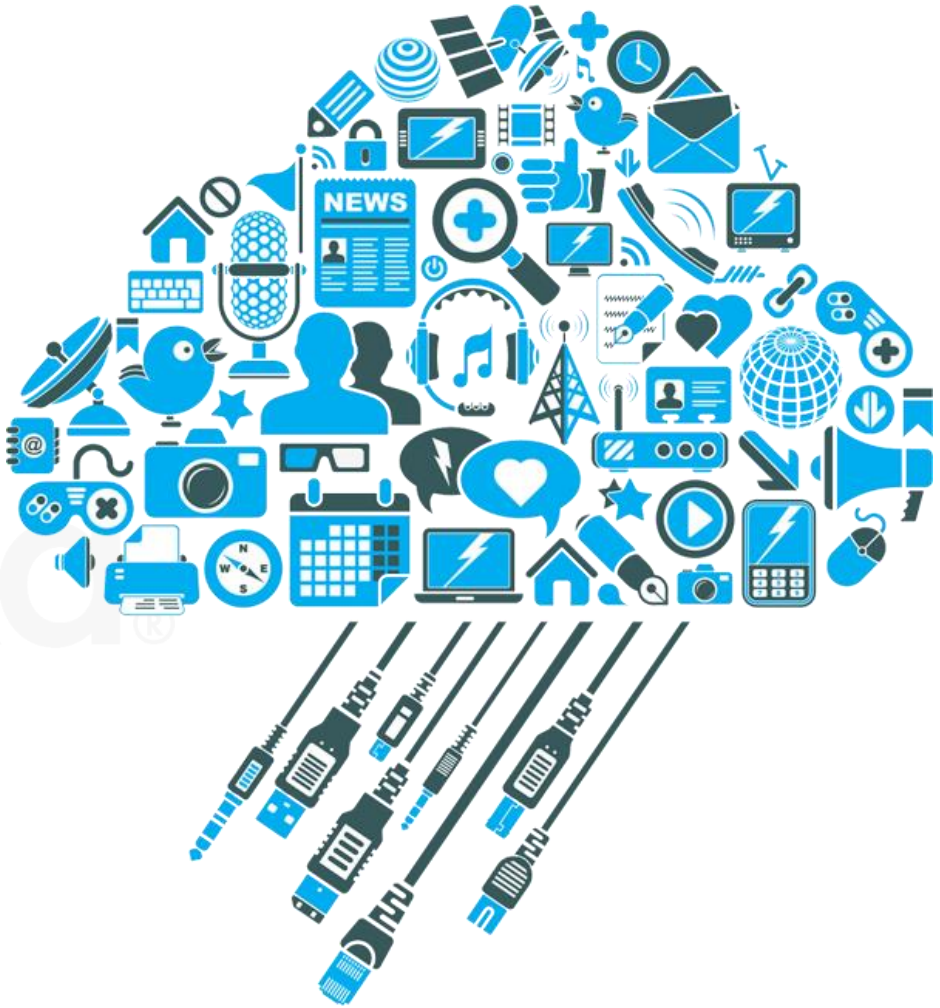
- **Scalabile**: Kafka è progettato per consentire a un singolo cluster di fungere da backbone centrale dei dati per una grande organizzazione. Può essere espanso elasticamente e in modo trasparente senza tempi di fermo (alta scalabilità). I flussi di dati sono partizionati e distribuiti su un cluster di macchine per consentire la «trasmissione» di grandi quantità di dati e per consentire cluster di consumer coordinati.



Caratteristiche salienti di Kafka (Tratte dal sito ufficiale)

- **Durevole**: I messaggi sono persistenti su disco e replicati all'interno del cluster per prevenire la perdita di dati. Ogni broker può gestire terabyte di messaggi senza impatto sulle prestazioni.

- **Distribuito**: Kafka ha un design moderno incentrato sul cluster che offre una lunga durata e garanzie di tolleranza agli errori.



Cosa offre Kafka

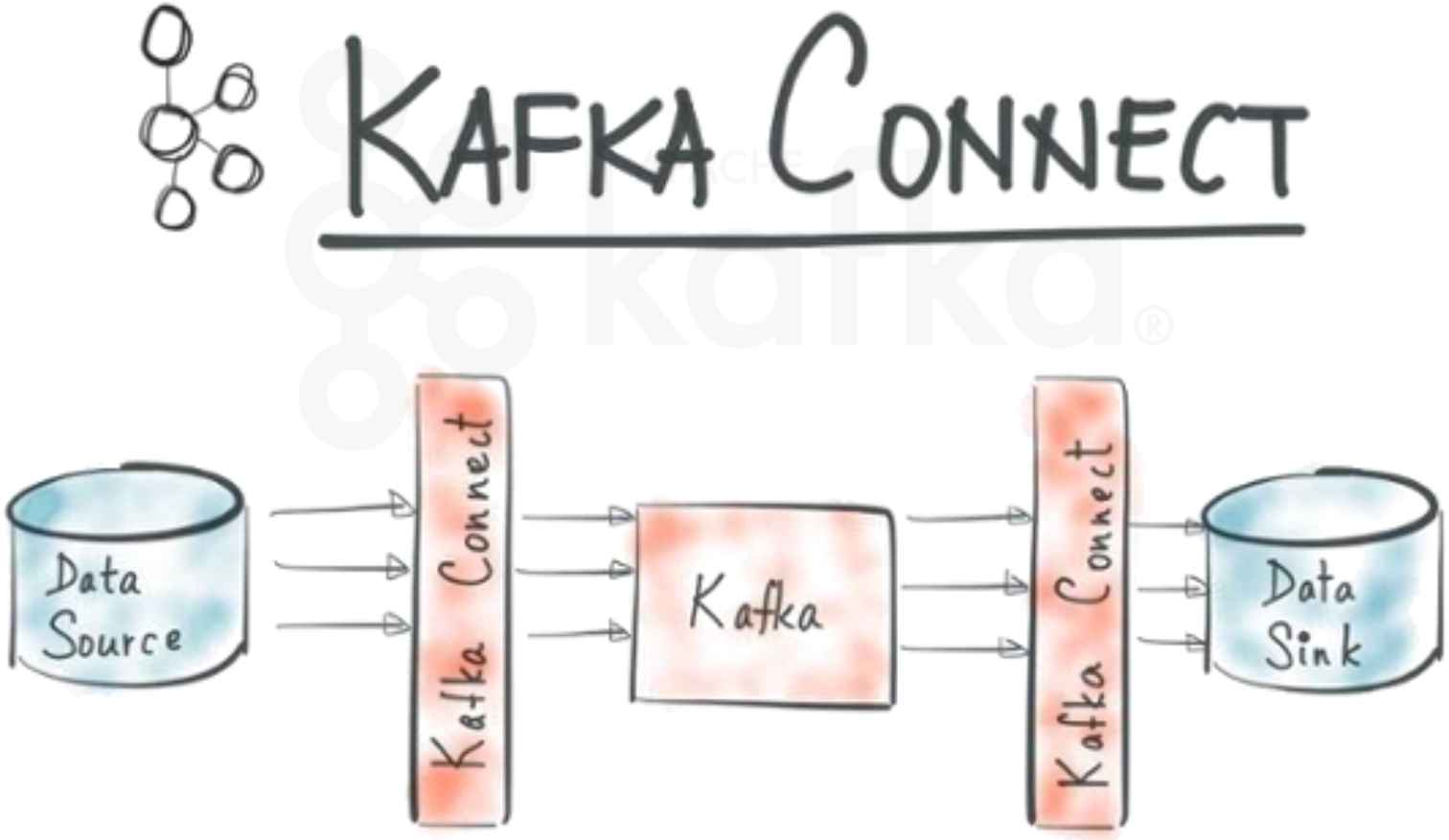
- La storia di Kafka inizia nel 2010, quando il celebre social LinkedIn sentì l'esigenza di affrontare le problematiche delle basse latenze inerenti alla gestione di grandi quantità di dati sul web.

- Considerando che in quel momento non vi erano soluzioni all'altezza, si decise di dare vita a una nuova piattaforma; in realtà avevano già delle tecnologie per l'elaborazione in batch, **ma non consentivano l'elaborazione dei dati in tempo reale**.

- Nello specifico Kafka è stato creato per risolvere il problema della pipeline di dati su LinkedIn.

- È stato progettato per fornire un sistema di messaggistica ad alte prestazioni in grado di gestire molti tipi di dati e fornire dati chiari e strutturati sull'attività degli utenti e le metriche di sistema in tempo reale.

- Tra le sfide più importanti per un software di messaggistica vi è quella relativa all'**elaborazione di grandi volumi di dati**. Un efficiente sistema deve essere in grado di rendere immediatamente disponibili i dati agli utenti: ed è proprio questo che Kafka riesce a fare.



I Vantaggi

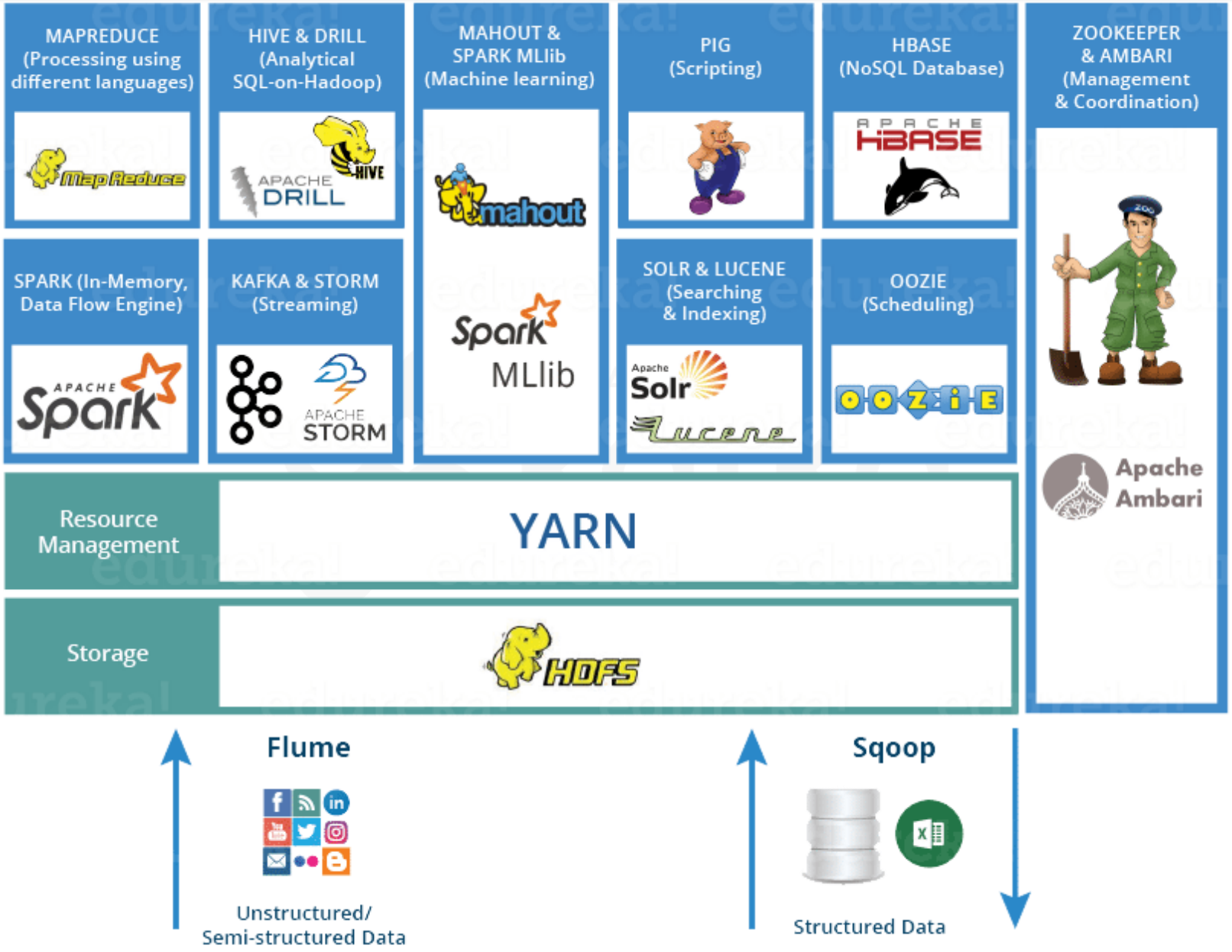
- Tra i vantaggi offerti dalla piattaforma vi sono:
  - Il monitoraggio delle attività sul web memorizzando e inviando i flussi di dati in tempo reale
  - La memorizzazione dei messaggi su hard disk per prevenire la perdita dei dati, con la facoltà di stabilire un preciso tempo di conservazione degli stessi
  - Il rilevamento delle minacce

I Vantaggi

- Kafka è generalmente integrato in ecosistemi software di tipo enterprise con Apache Storm, Apache HBase e Apache Spark.

- Il fine ovviamente è quello di **elaborare i dati in tempo reale e in streaming** con supporto ai flussi di messaggi al cluster di Hadoop (a prescindere dalla classe di utilizzo).

Kafka & Hadoop Software Ecosystem



Cluster, Topic e Record

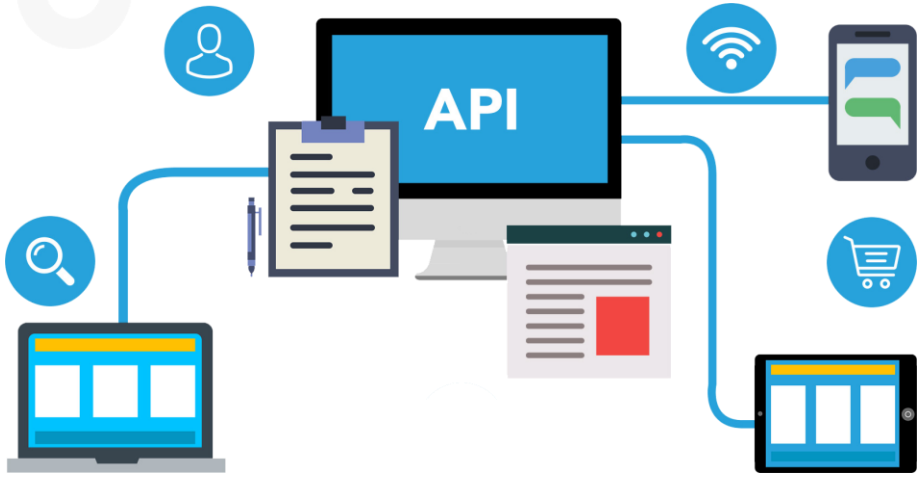
- Per quanto concerne il funzionamento, la piattaforma viene distribuita come **cluster** da installare su uno o più server.

- Il cluster ha la capacità di memorizzare vari topic, ovvero dei flussi di record.

- Ogni record contiene al suo interno tre dettagli:

  - una chiave
  - un valore
  - un timestamp.

- L'architettura è basata su quattro API principali:
  - Producer API: permette alle applicazioni di pubblicare flussi in uno o più topic.
  - Consumer API: consente l'elaborazione dei topic e del flusso prodotto dai record.
  - Stream API: riceve un determinato input dai topic e produce degli output, convertendo i flussi.
  - Connector API: gestisce il collegamento tra varie applicazioni.



Concetti fondamentali

- TOPIC: è una sorta di categoria utilizzata per raggruppare i messaggi;
- PARTITION: ciascuna delle sottosezioni in cui è diviso un topic;
- RECORD: è il messaggio vero e proprio, costituito da una chiave, un valore e un timestamp;
- OFFSET: indice che identifica univocamente un record all'interno di una partizione;
- PRODUCER: entità che invia i messaggi a Kafka;
- CONSUMER: entità che riceve i messaggi da Kafka;

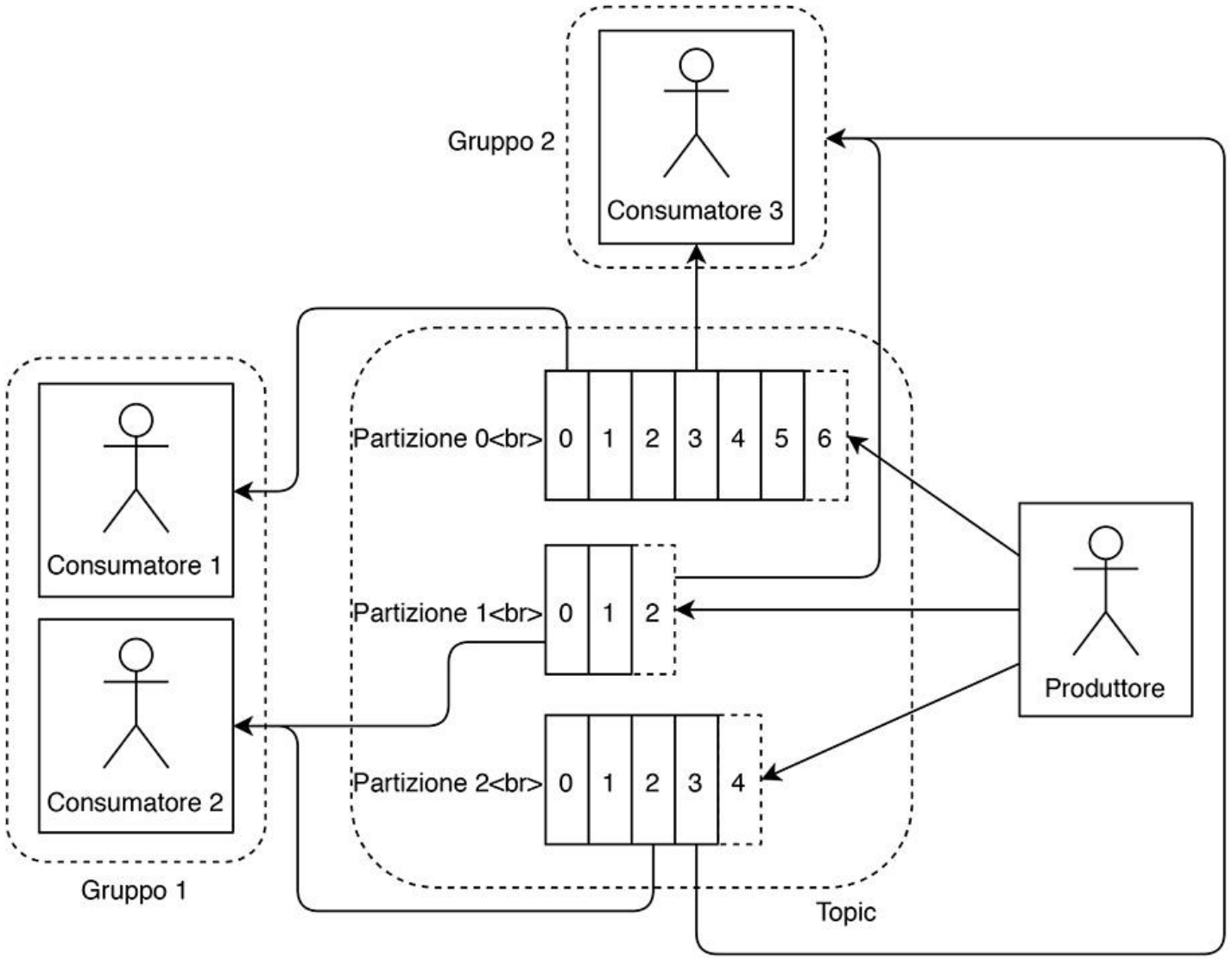
Concetti fondamentali

- GRUPPO: è un’etichetta utilizzata per distinguere insiemi di consumatori sottoscritti a un topic;

- BROKER: è un processo che si occupa di gestire la ricezione e il salvataggio dei messaggi e i relativi offset;

- CLUSTER: insieme di più broker utile per replicare e distribuire le partizioni.

Esempio di un Topic con tre Partitions



Topic e Partitions

- Ogni partizione contiene un numero variabile di record, ciascuno con il suo offset.
- La partizione a cui viene assegnato un messaggio è:
  - quella scelta dal produttore, se specificata al momento dell'invio;
  - quella ottenuta elaborando l'hash della chiave del record, se presente;
  - una qualsiasi secondo una logica di tipo round-robin, se il produttore non ha indicato né una partizione né una chiave per il messaggio.

- Se un gruppo è formato da più consumatori, le partizioni di un topic e i relativi record vengono distribuiti tra di essi.

- In particolare, detto C il numero di consumatori di un gruppo e P il numero di partizioni:

  - se C > P allora:
    - a P consumatori verrà assegnata una partizione
    - mentre C – P consumatori resteranno senza partizioni;

  - se C = P ciascun consumatore avrà una e una sola partizione;

  - se C < P le varie partizioni saranno distribuite in maniera uniforme tra i vari consumatori.

- Il bilanciamento delle partizioni è dinamico, ossia esse vengono redistribuite al momento della connessione o disconnessione dei consumatori.

- Come si può facilmente intuire le partizioni e i gruppi di consumatori sono gli elementi che permettono di scalare orizzontalmente un sistema.

- Nell'esempio mostrato precedentemente possiamo notare che:

  - il consumatore 1 riceve messaggi solo dalla partizione 0;
  - il consumatore 2 riceve messaggi dalle partizioni 1 e 2;
  - il consumatore 3 riceve messaggi da tutte le partizioni essendo l'unico membro del suo gruppo.

- È importante sottolineare che un consumatore non riceve mai automaticamente i messaggi, bensì deve richiederli esplicitamente quando è pronto a elaborarli.

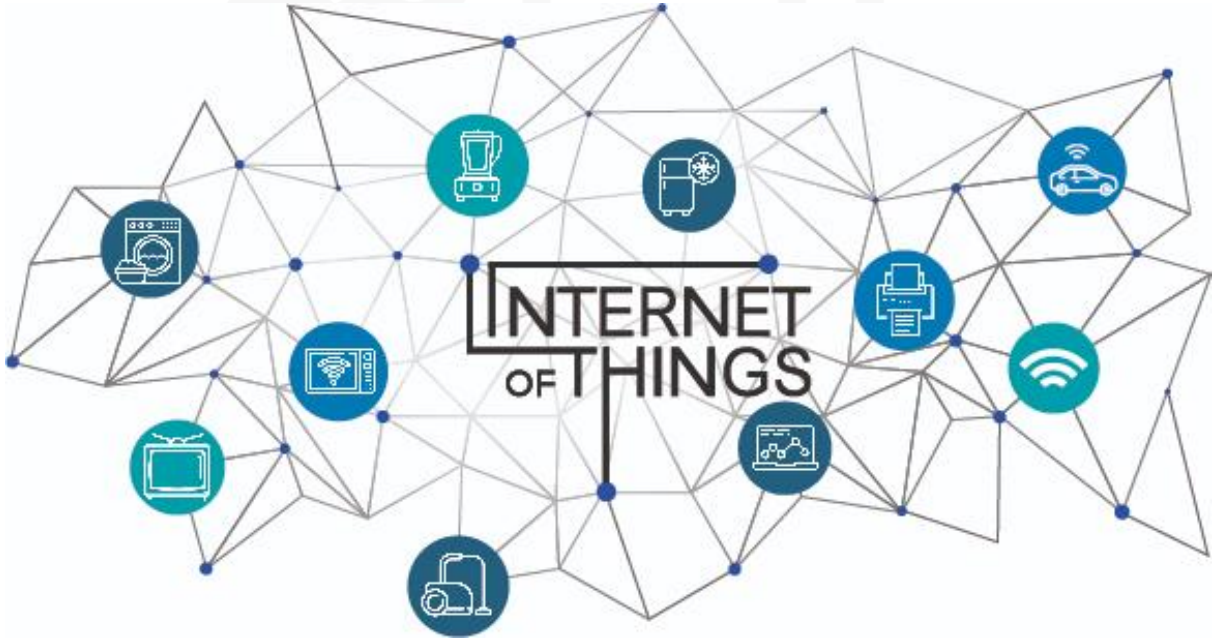
UNIT

Casi D’Uso

Casi D'Uso - IoT, sensor data analysis

- l'utilizzo di Kafka si colloca in un contesto in cui l'uso di sensori applicati a macchine industriali, autovetture e oggetti di uso comune sta crescendo enormemente.

- Offre infatti la possibilità di gestire le enormi quantità di dati generate dai sensori, con bassissima latenza.



- Attraverso lo stream processing è possibile operare modifiche o trasformazioni dei dati durante il flusso.

- Non è quindi necessario affidarsi a un servizio esterno per gestire attività di storage e processing.

Batch Processing

The collection and storage of data, for processing at a scheduled time when a sufficient amount of data has been accumulated

Examples:
- Cheque Clearing
- Generation of Bills
- Credit Card Transactions

Advantages / Disadvantages:
- Many transactions are completed at one time in a single process
- Data takes time to be processed

Real-Time Processing

The immediate processing of data after the transaction occurs, with the database being updated at the time of the event

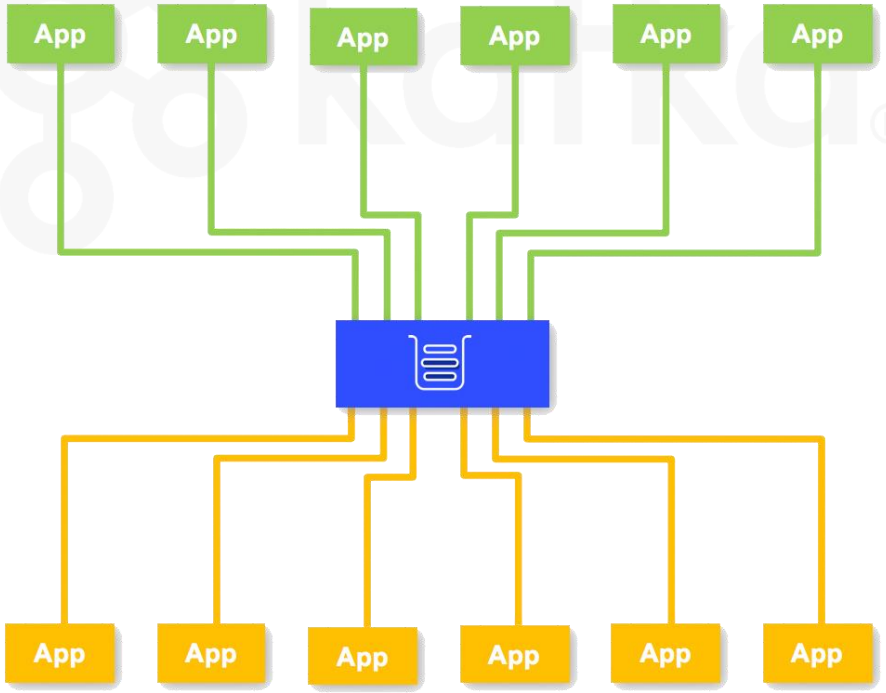
Examples:
- Reservation Systems
- Point of Sales Terminals (POS)

Advantages / Disadvantages :
- Data is processed immediately
- The act of processing data is repetitive

Casi D'Uso - Scalabilità e Fault-tolerance

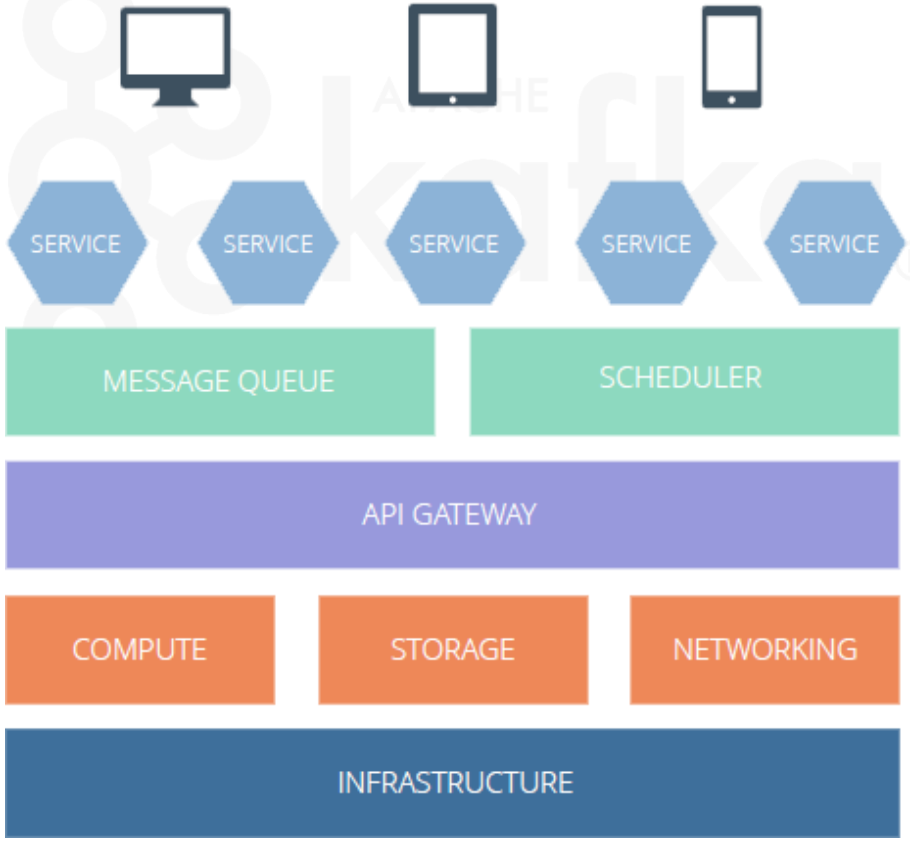
- La velocità di elaborazione e il volume dei dati in entrata possono essere variabili, quindi è necessario poter manipolare la capacità di uno strumento di messaging per modulare le variabili di precisione o performance.

- Kafka è in grado di gestire tutto questo, e grazie alla persistenza dei dati all'interno dei cluster (anche questa gestibile dall'utente) garantisce anche un alto livello di fault tolerance.



- Kafka può essere utilizzato come strumento per lo scambio di messaggi asincroni in un ecosistema di microservizi.

- Velocizza le comunicazioni tra le applicazioni e riduce al minimo il rischio di perdita di informazioni



UNIT

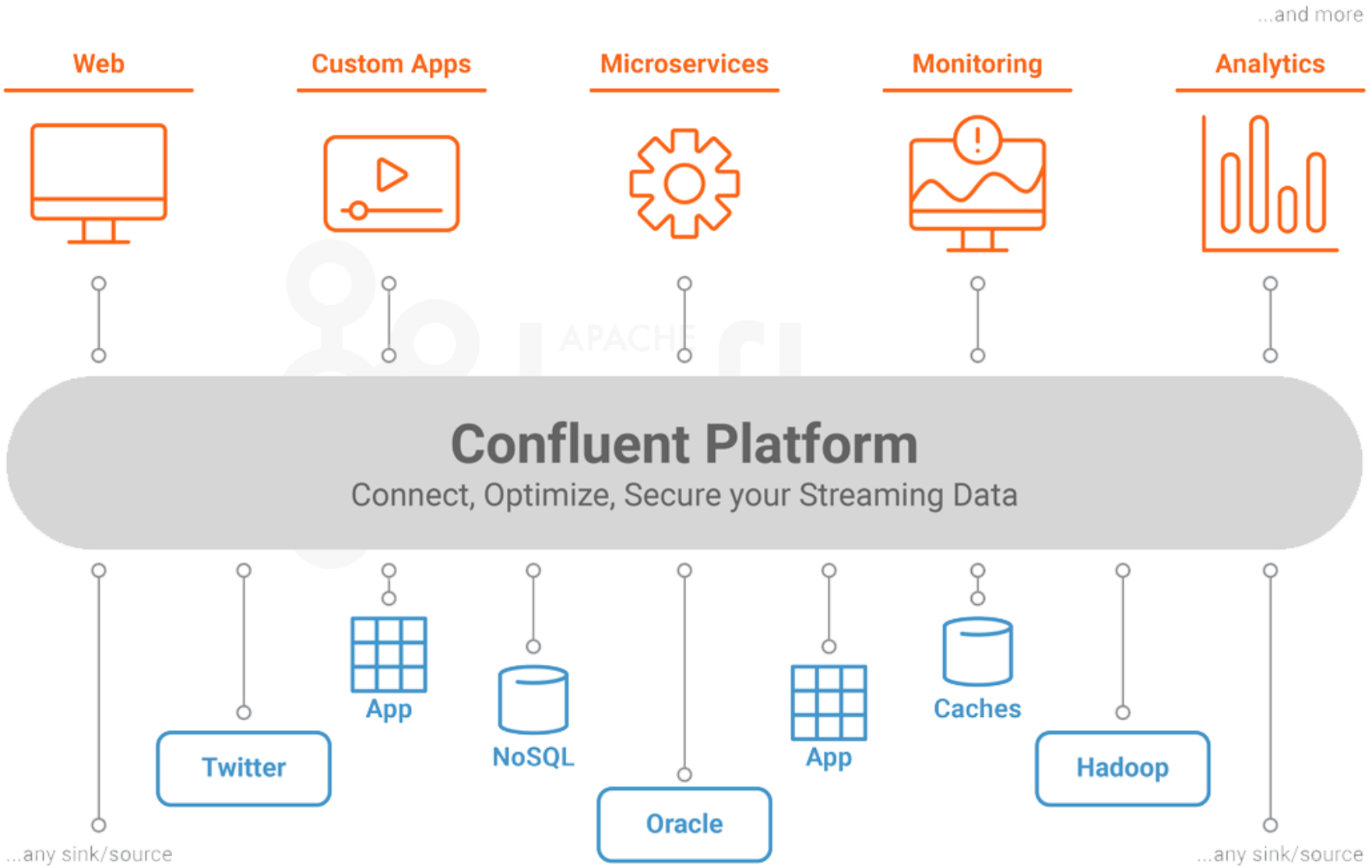
Confluent

- Confluent è la soluzione opensource basata su Apache Kafka™, per costruire funzioni avanzate di Stream Data Processing.

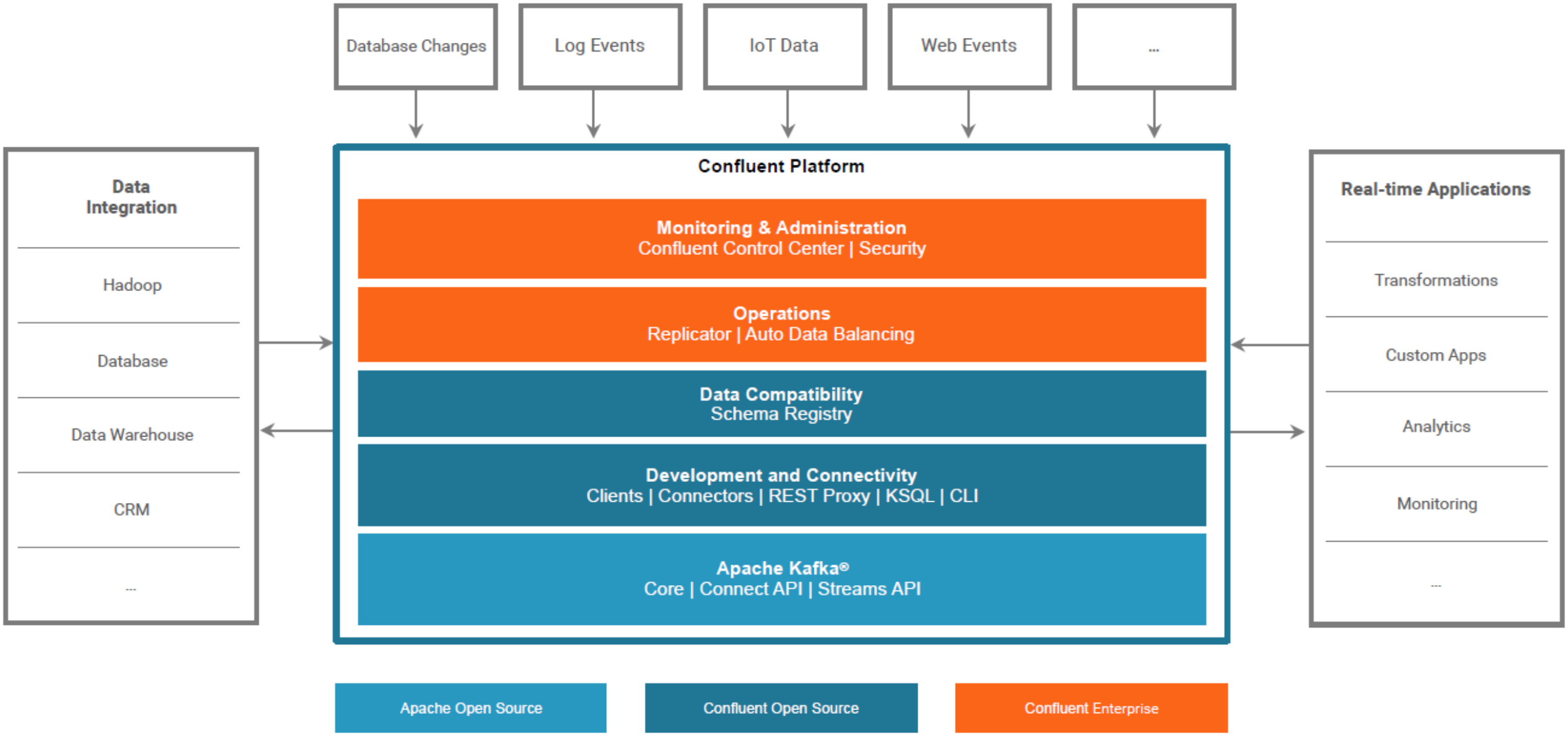
- Ideata dai creatori di Kafka, la piattaforma migliora l'applicazione permettendogli di espandere le sue capacità di integrazione, in modo da aggiungere strumenti per ottimizzare, gestire i cluster e garantire la sicurezza dei flussi di dati.

- Confluent rende Kafka più facile da configurare e più facile da usare, ed è disponibile in licenza Open Source ed Enterprise, tramite un abbonamento dedicato.

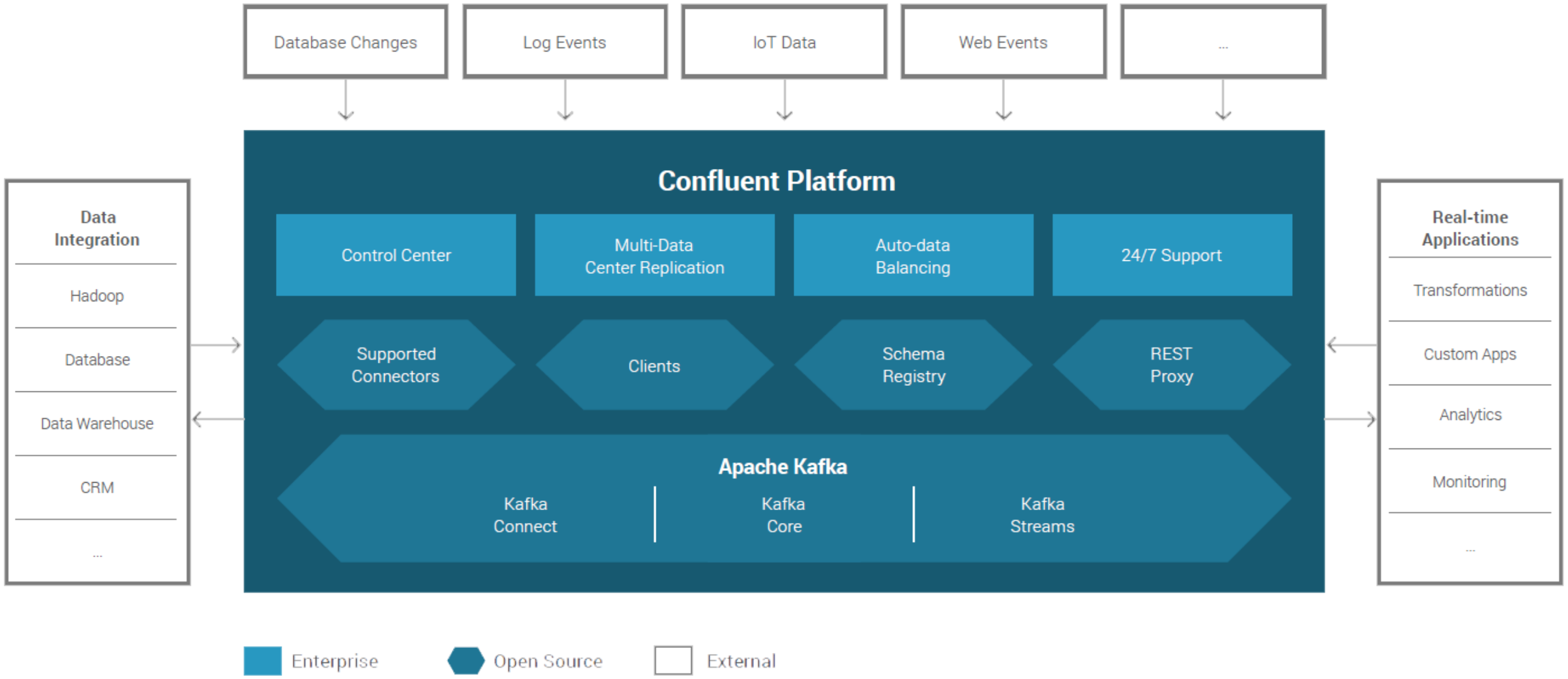
Confluent Platform



Confluent Platform



Confluent Platform



UNIT

Osservazioni

Perché scegliere Kafka

- Kafka è una piattaforma in costante crescita, sviluppata per velocizzare e per massimizzare la sicurezza circa i flussi di dati prodotti dalla messaggistica.

- Il programma è passato dalla gestione di un miliardo di messaggi agli oltre tre trilioni di oggi, dimostrandosi sempre più gettonato dalle grandi aziende, tra cui Linkedin, Twitter, Netflix.

- Si tratta della scelta giusta per coloro che si occupano della gestione di grandi volumi di dati, per gli sviluppatori di applicazioni e per gli sviluppatori su piattaforma Hadoop.

Quando usare Apache Kafka

- Apache Kafka è integrato nei flussi di data streaming che consentono la condivisione di dati tra sistemi e/o applicazioni, ma anche nei sistemi e nelle applicazioni che usano quei dati.

- Può essere utilizzato in tutti quei casi in cui velocità e scalabilità elevate sono fondamentali.

- La sua capacità di ridurre al minimo la necessità di integrazioni point-to-point per la condivisione di dati in determinate applicazioni, riduce infatti la latenza a millisecondi.

- Gli utenti possono quindi usufruire dei dati più velocemente, aspetto utile quando i dati devono essere disponibili in tempo reale.

Quando usare Apache Kafka

- Riuscendo a gestire milioni di punti dati al secondo, Apache Kafka è la soluzione ideale alle problematiche legate ai Big Data.

- Kafka è uno strumento utile anche per le aziende che non si trovano ad affrontare difficoltà di questo tipo ovvero può essere introdotto in un normale contesto applicativo ove, ad esempio, è necessario redistribuire dati in ambienti ad alta deframmentazione del software.

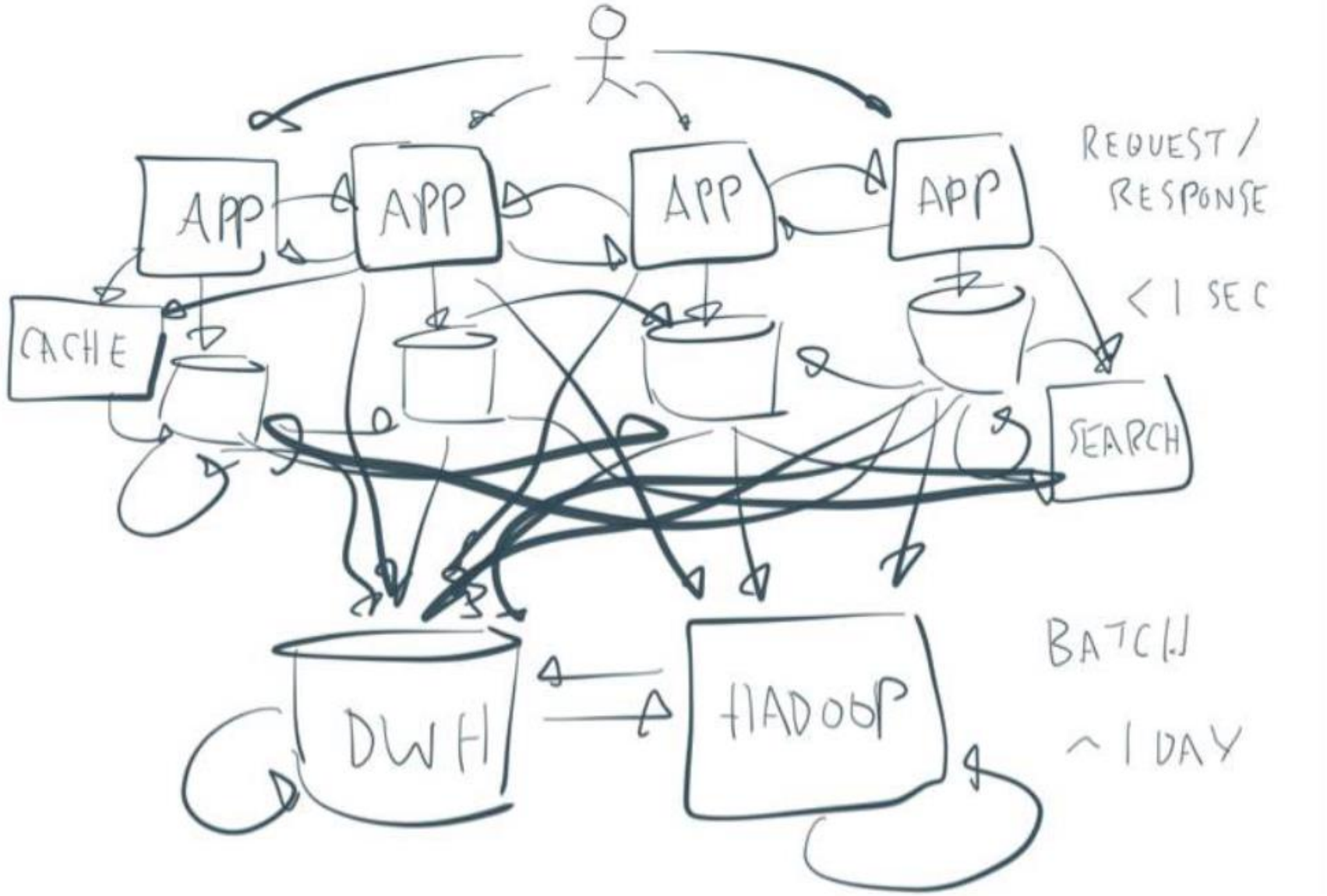
- In molti esempi di utilizzo che implicano l'elaborazione di dati, come l'Internet of Things (IoT) e i social media, dove il volume di dati stessi cresce a una velocità tale da poter compromettere le applicazioni esistenti.

- In generale, quando occorre prevedere una certa scalabilità nell'elaborazione, che consenta di rispondere alla crescente proliferazione dei dati.

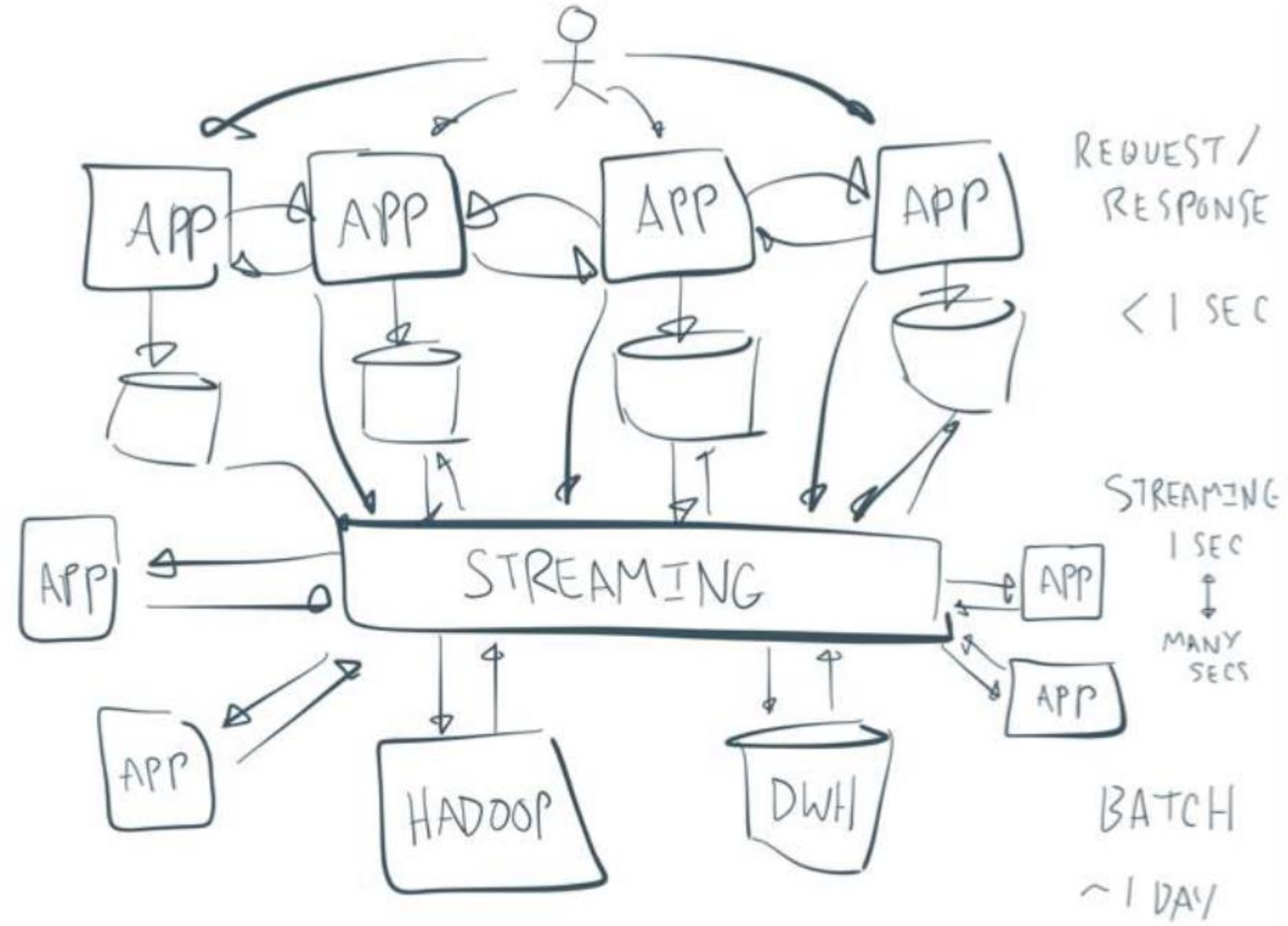
Riepilogo

- Kafka si occupa di elaborare grandi moli di dati in tempo reale, permettendo di creare sistemi scalabili a elevato throughput e bassa latenza.
- Gestisce la persistenza dei dati stessi sia sul server principale che sulle repliche, garantendo così la cosiddetta fault-tolerance.
- Kafka può essere usato come strumento per lo scambio di messaggi asincroni in un ecosistema di microservizi
- …e molto altro…

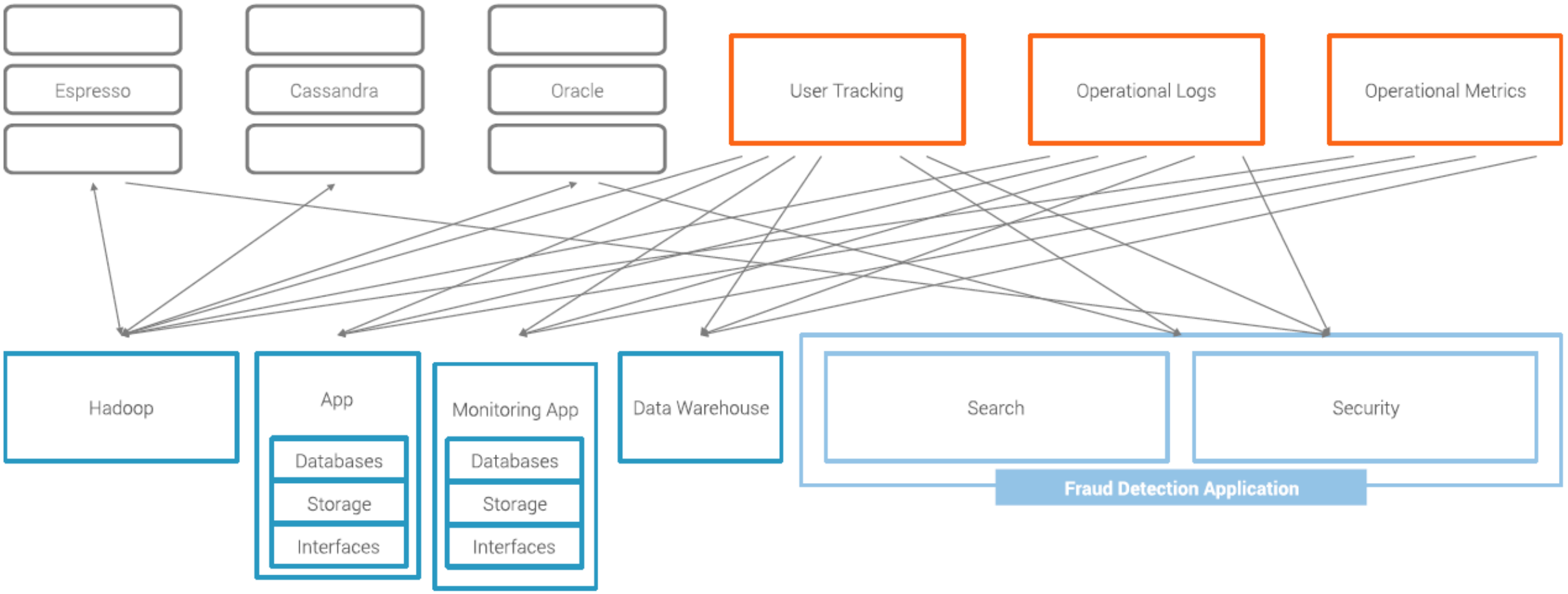
Prima Di Kafka



Dopo Kafka

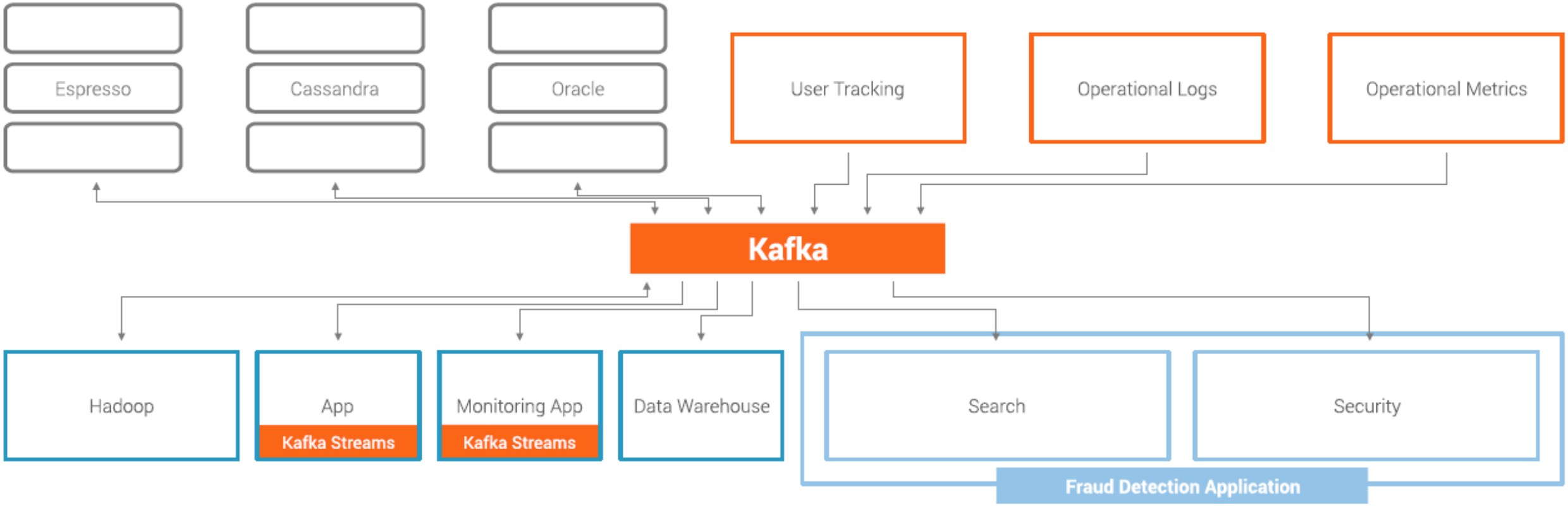


Prima Di Kafka



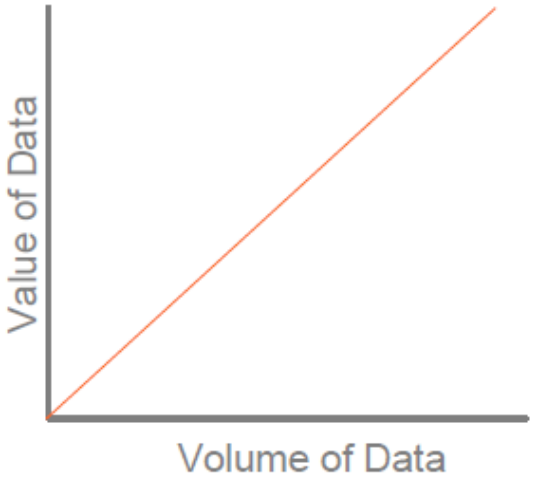
Dopo Kafka

✓ Processes Streams ✓ Distributed ✓ Fault Tolerant ✓ Stores Messages

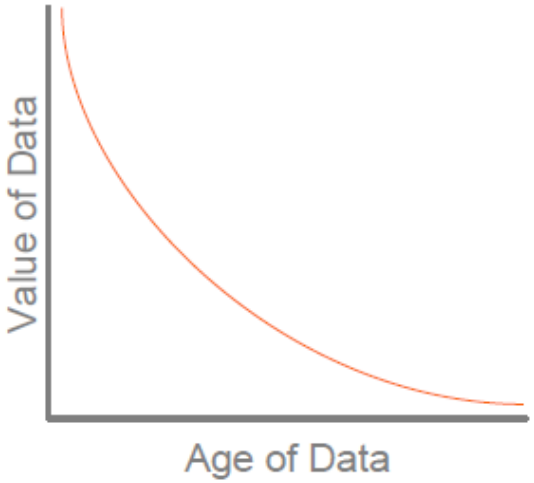


Kafka promette il passaggio «From Big Data to Fast Data»

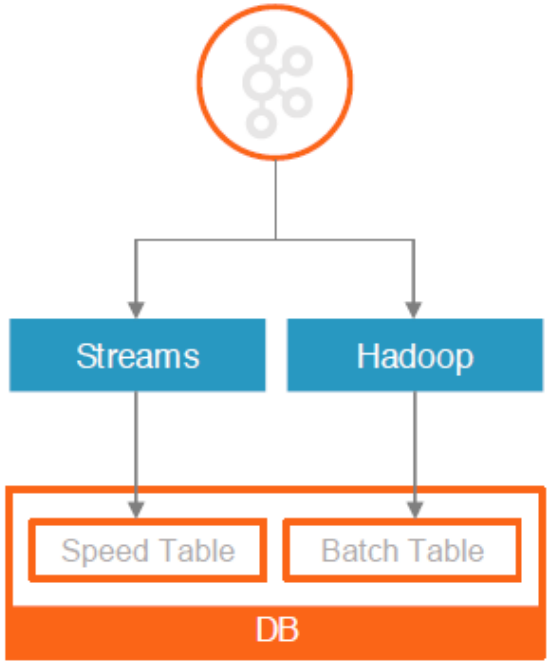
Big data was
The more the better



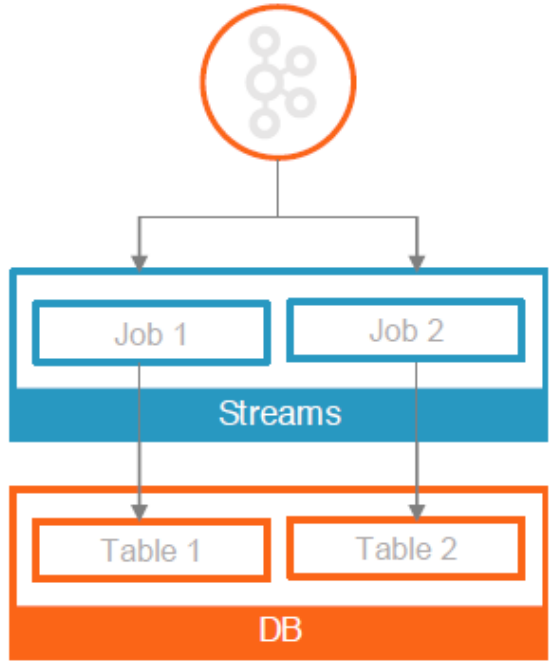
Stream data is
The faster the better



Stream data can be
big or fast (Lambda)

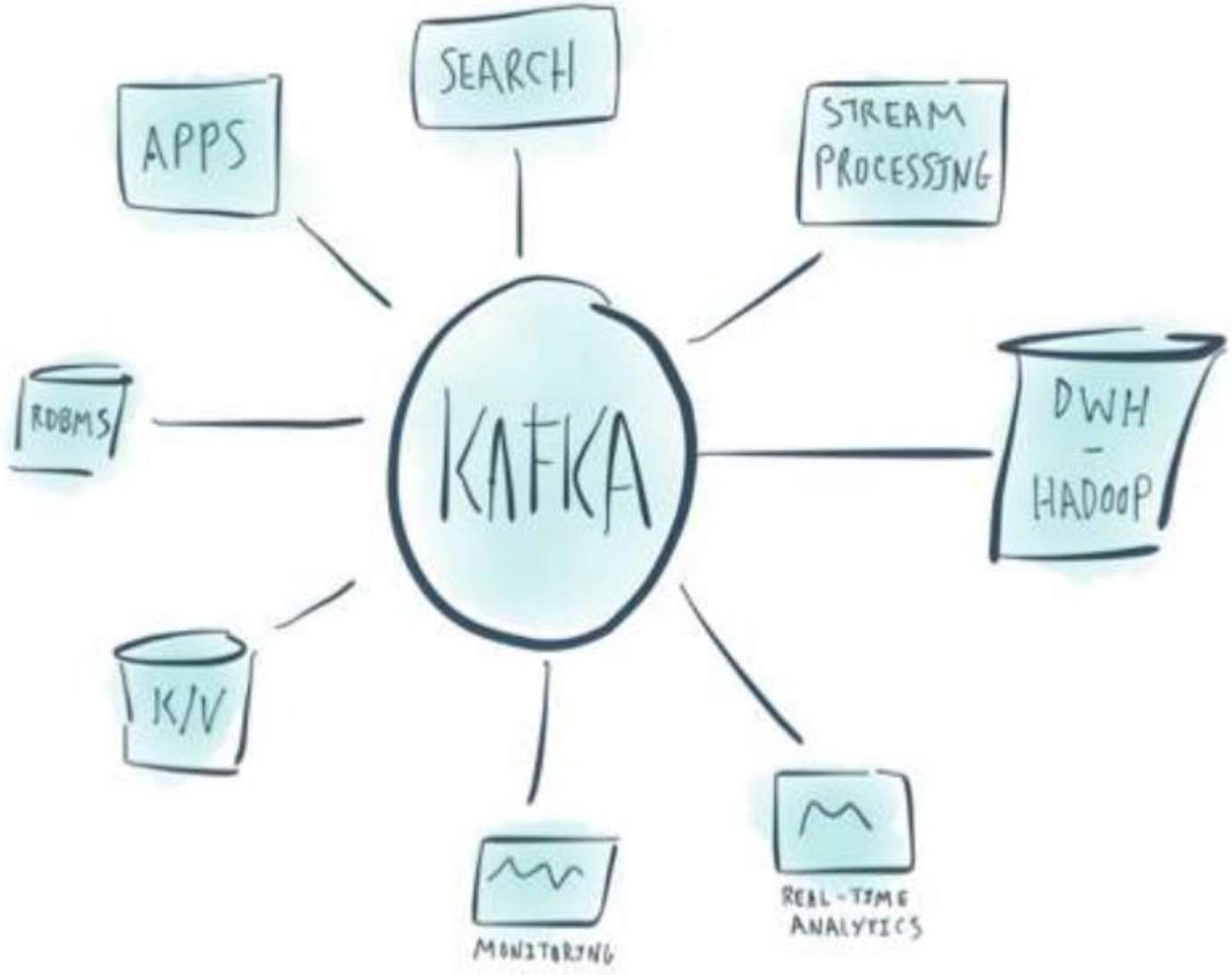


Stream data will be
big AND fast (Kappa)

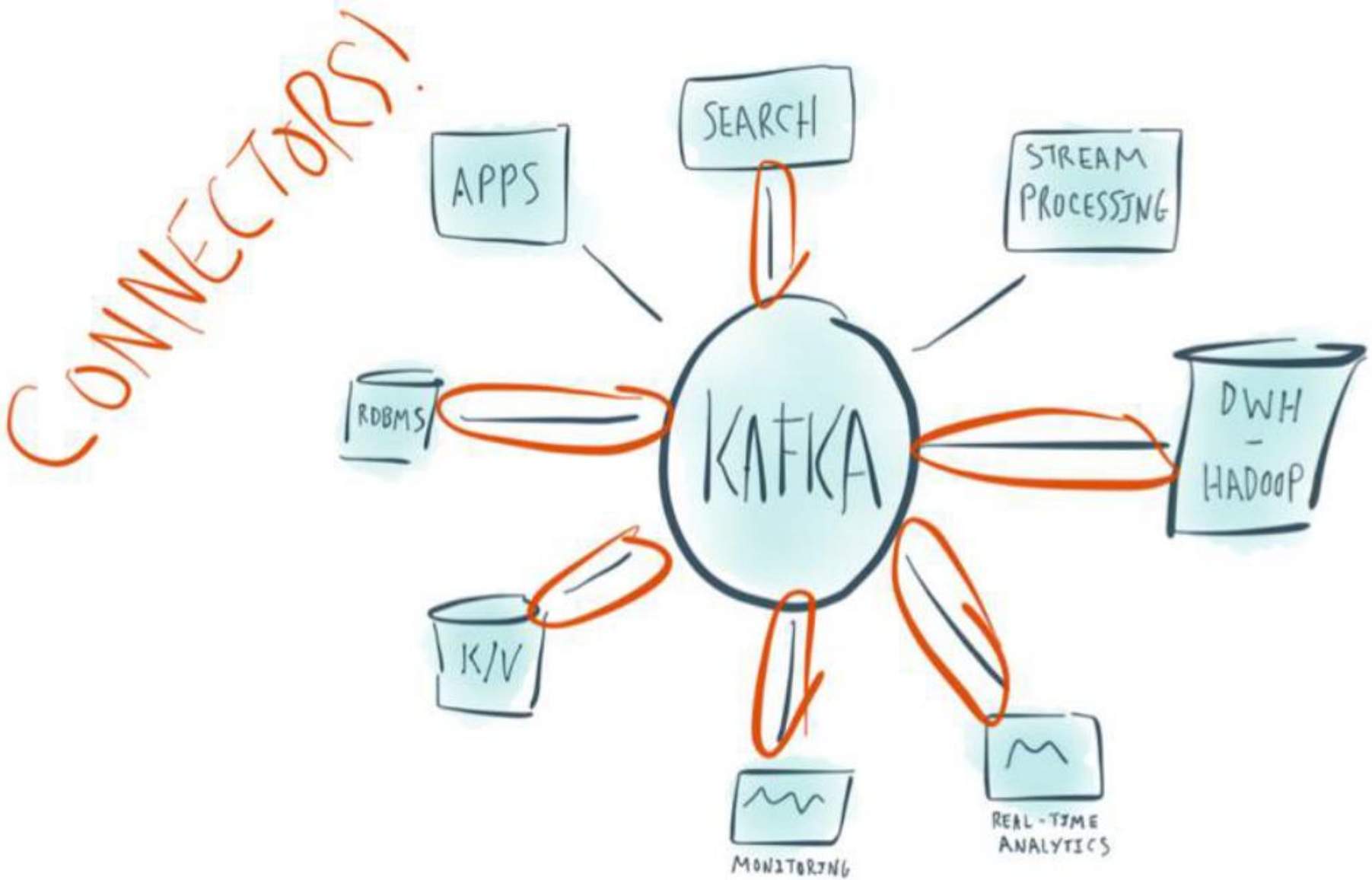


Apache Kafka is the enabling technology of this transition

Kafka – Streaming Platform

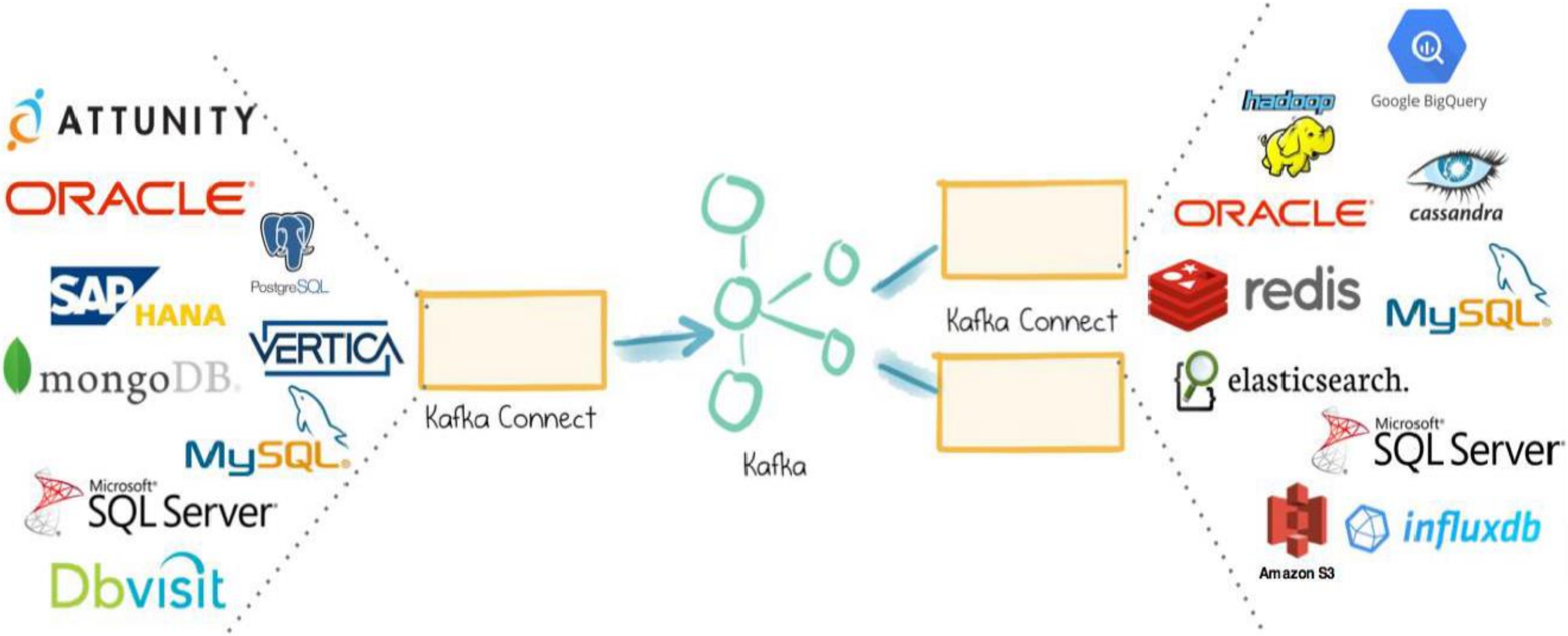


Kafka – Connectors

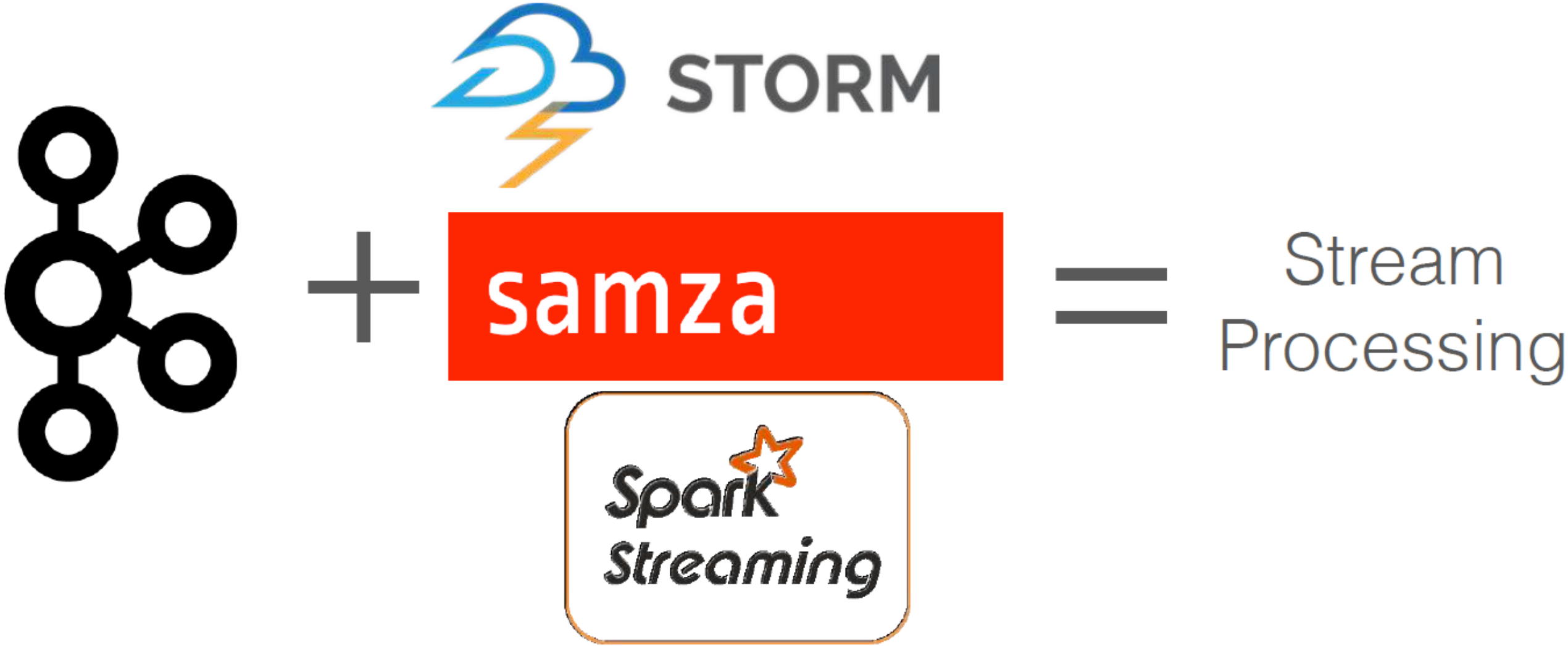


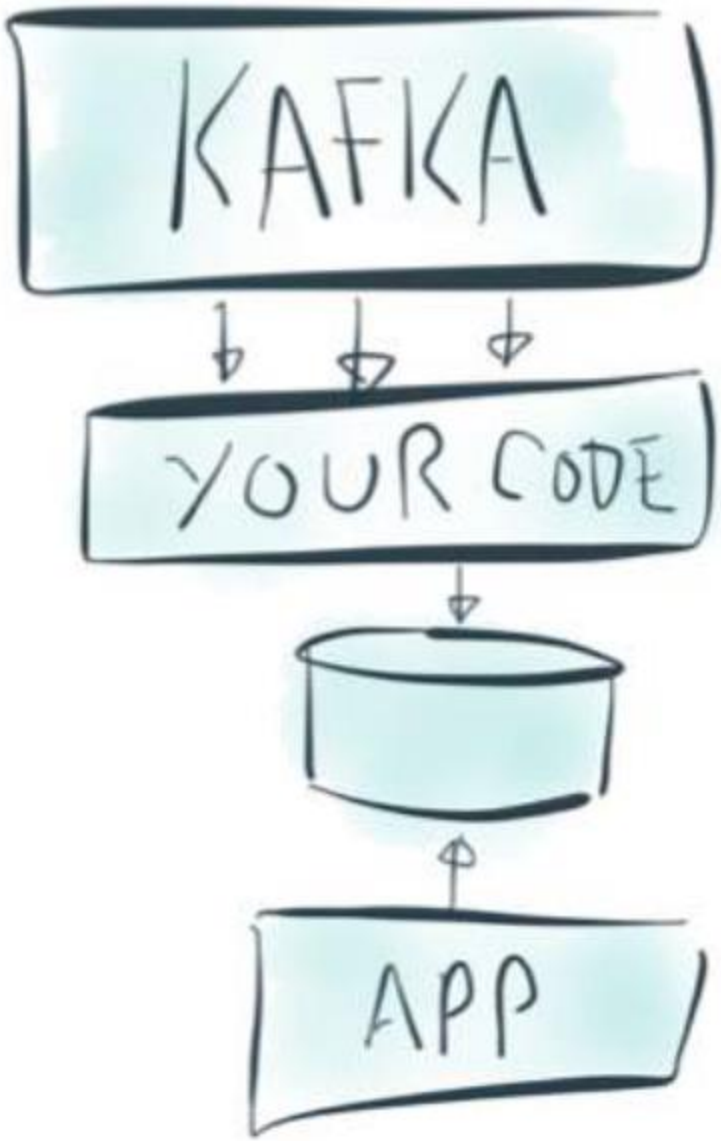
Kafka – CONNECT

Kafka – CONNECT

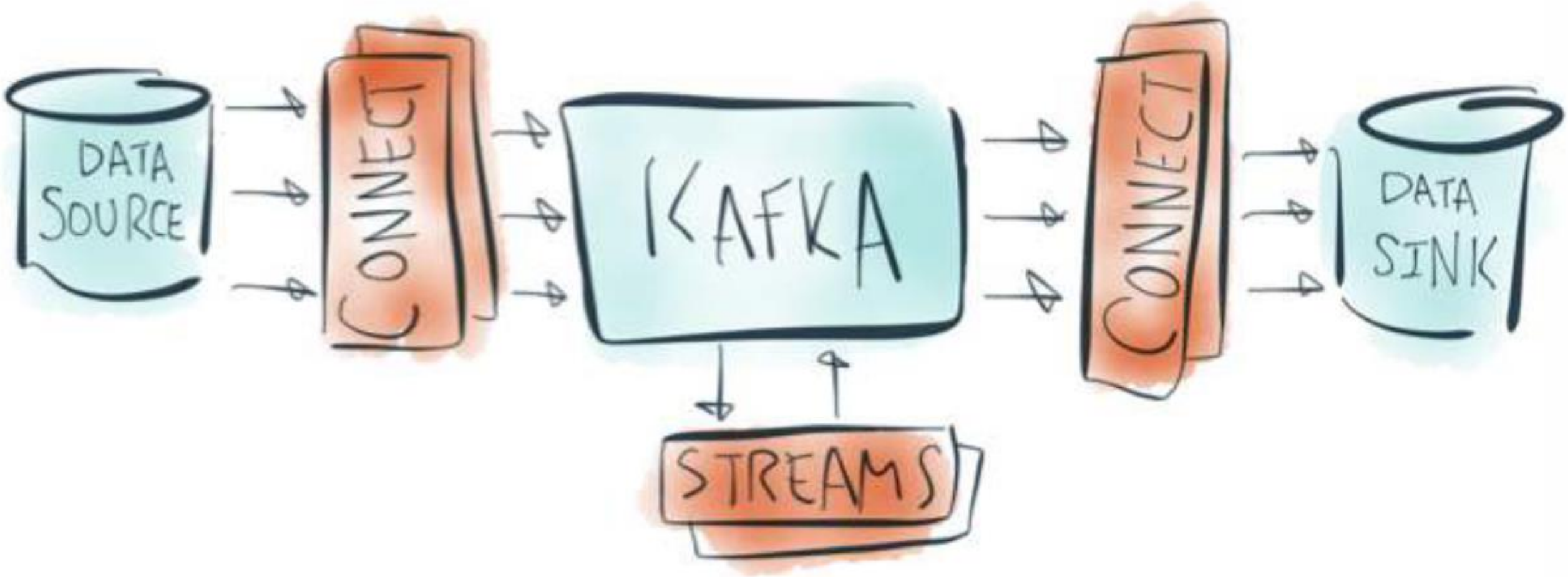


Kafka – CONNECT

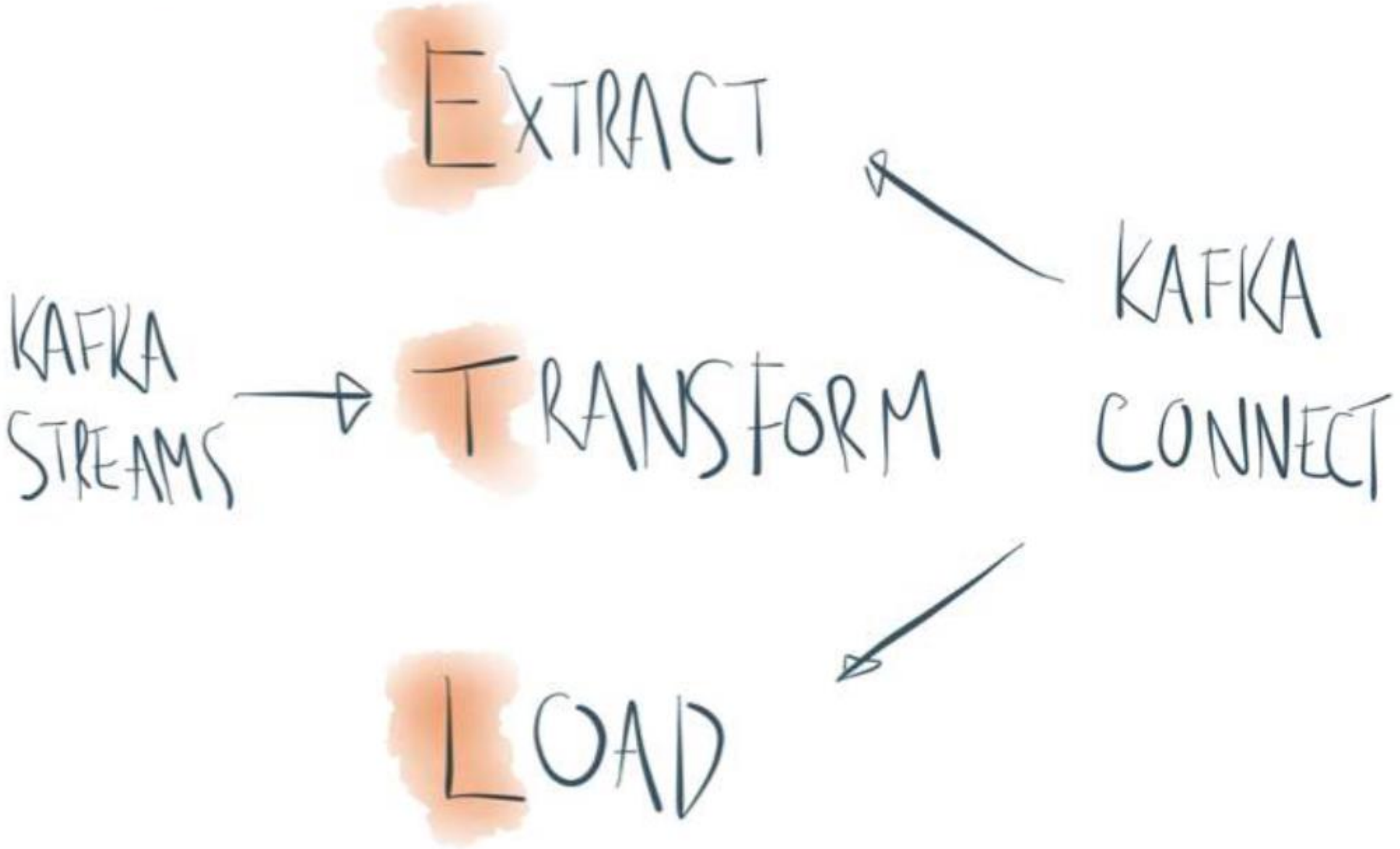




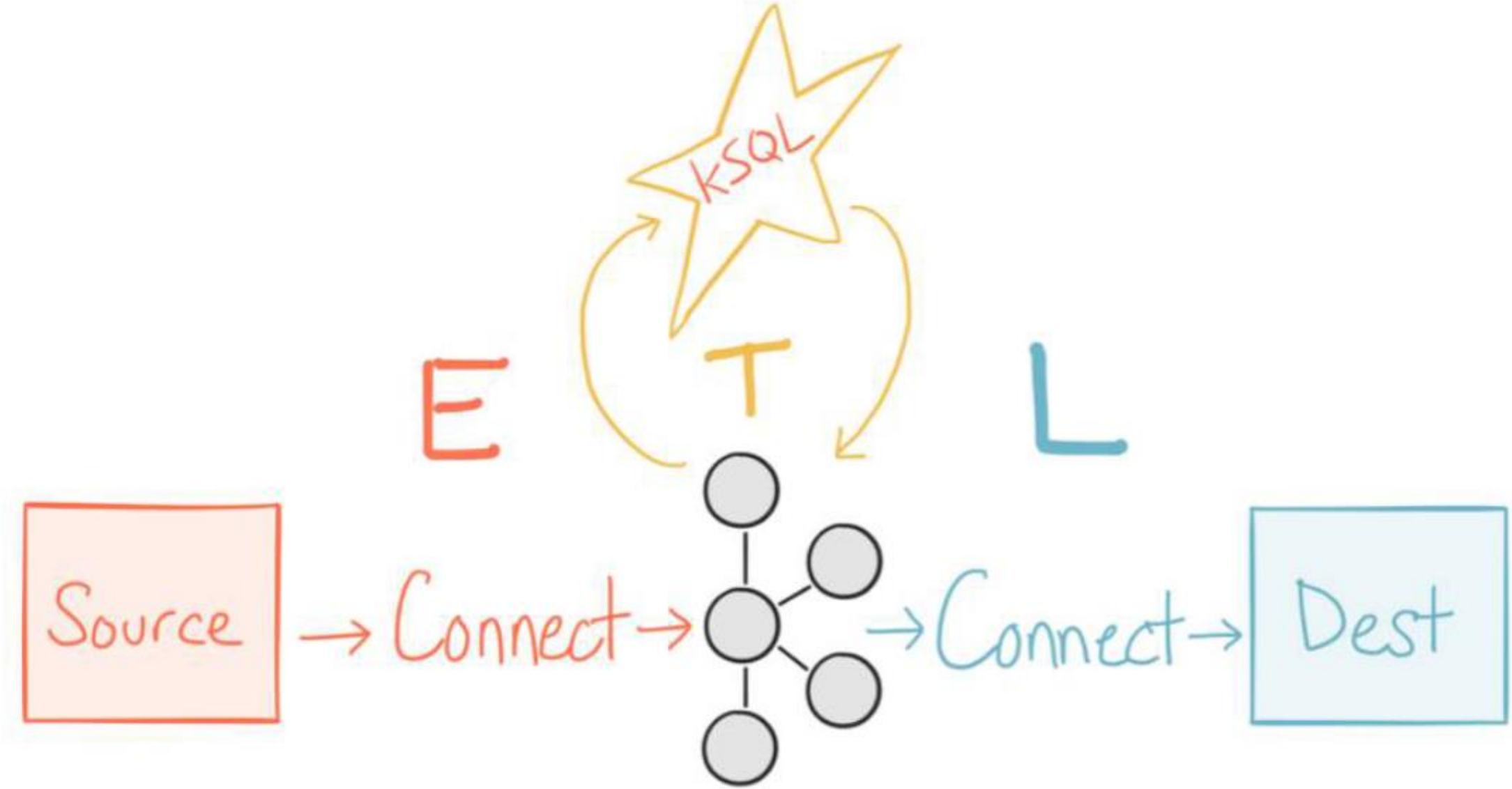
Kafka – Connect + Streams



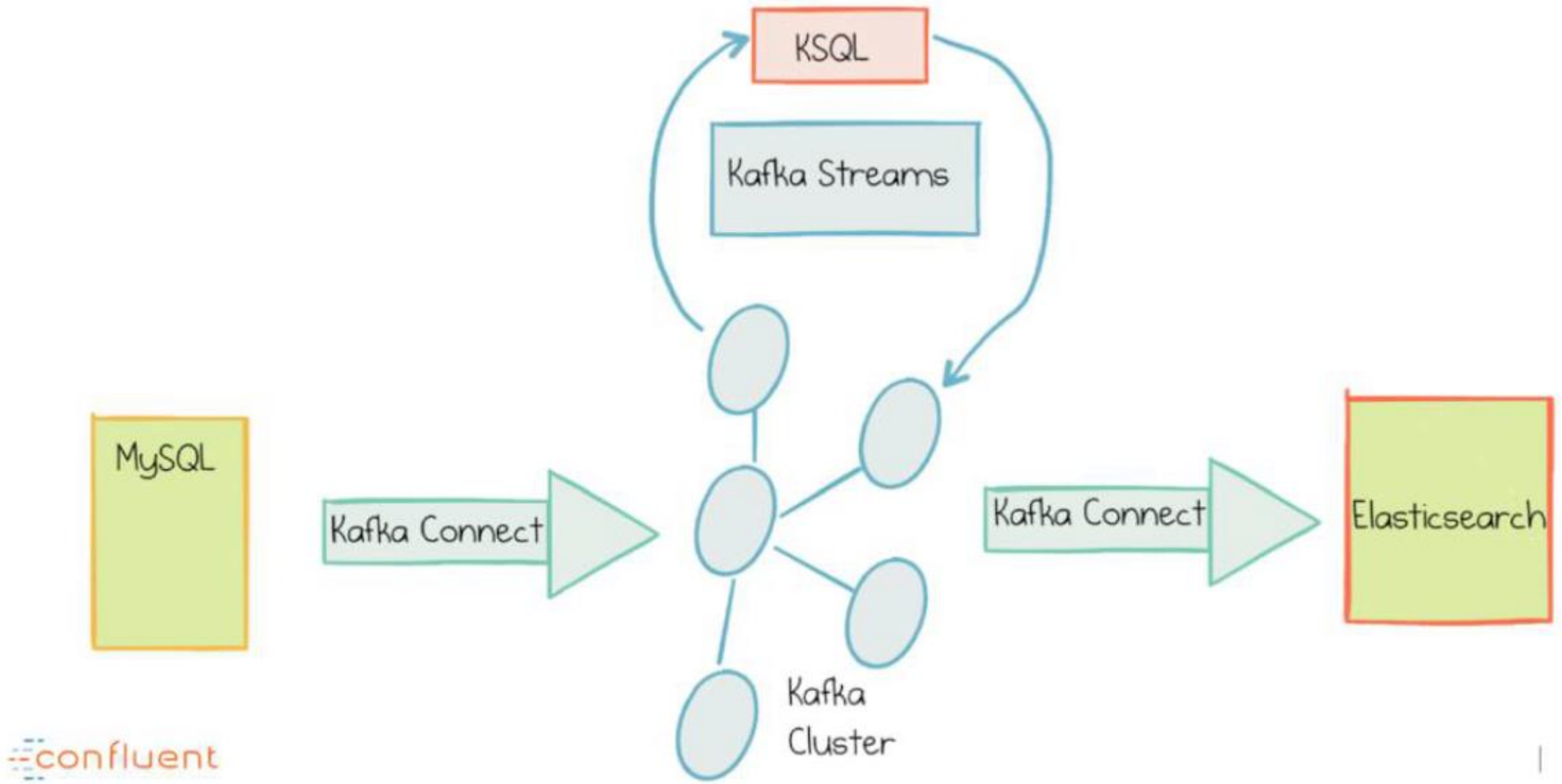
Kafka – Connect + Streams = ETL



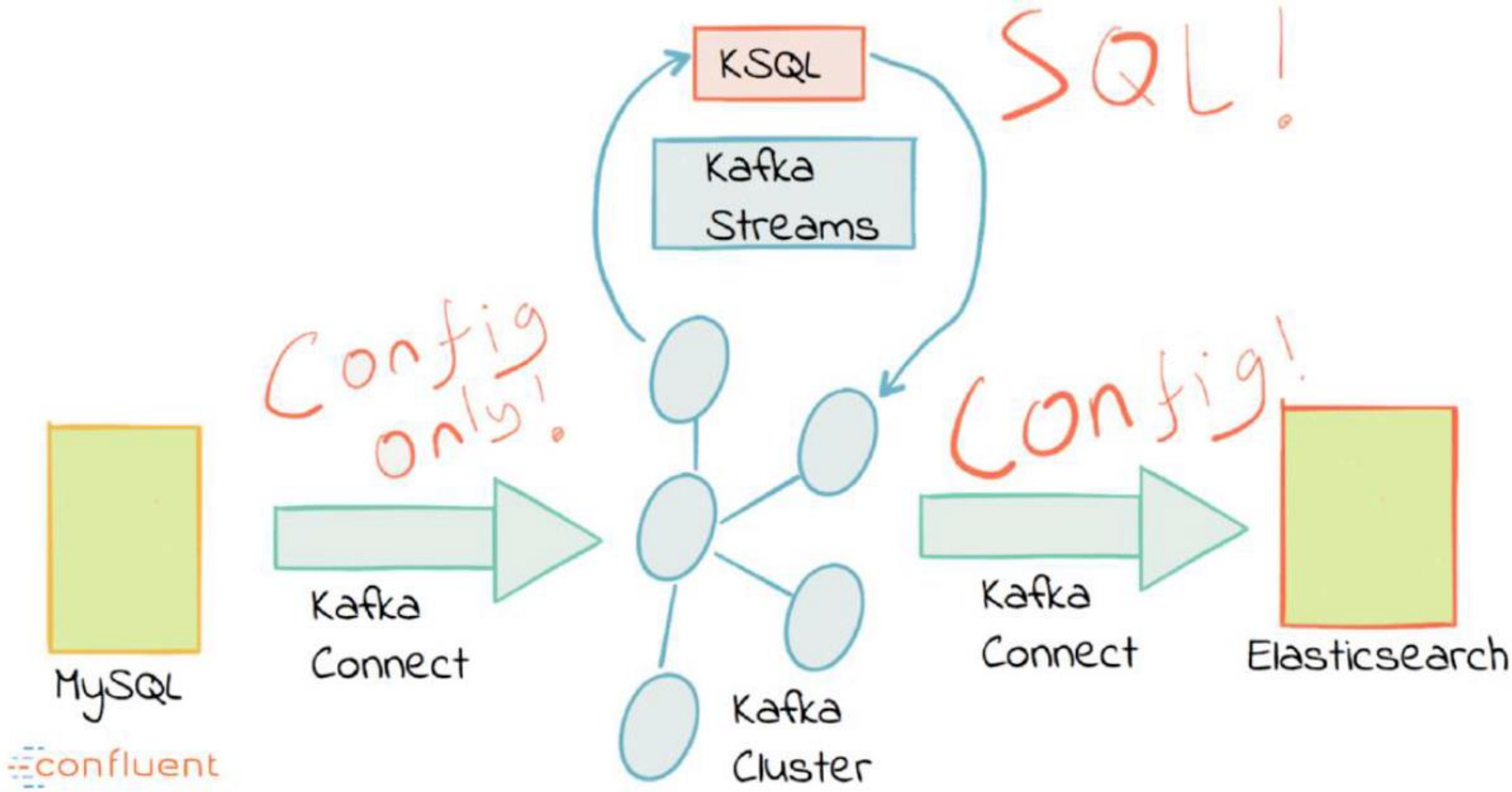
Kafka – Connect + Streams = ETL



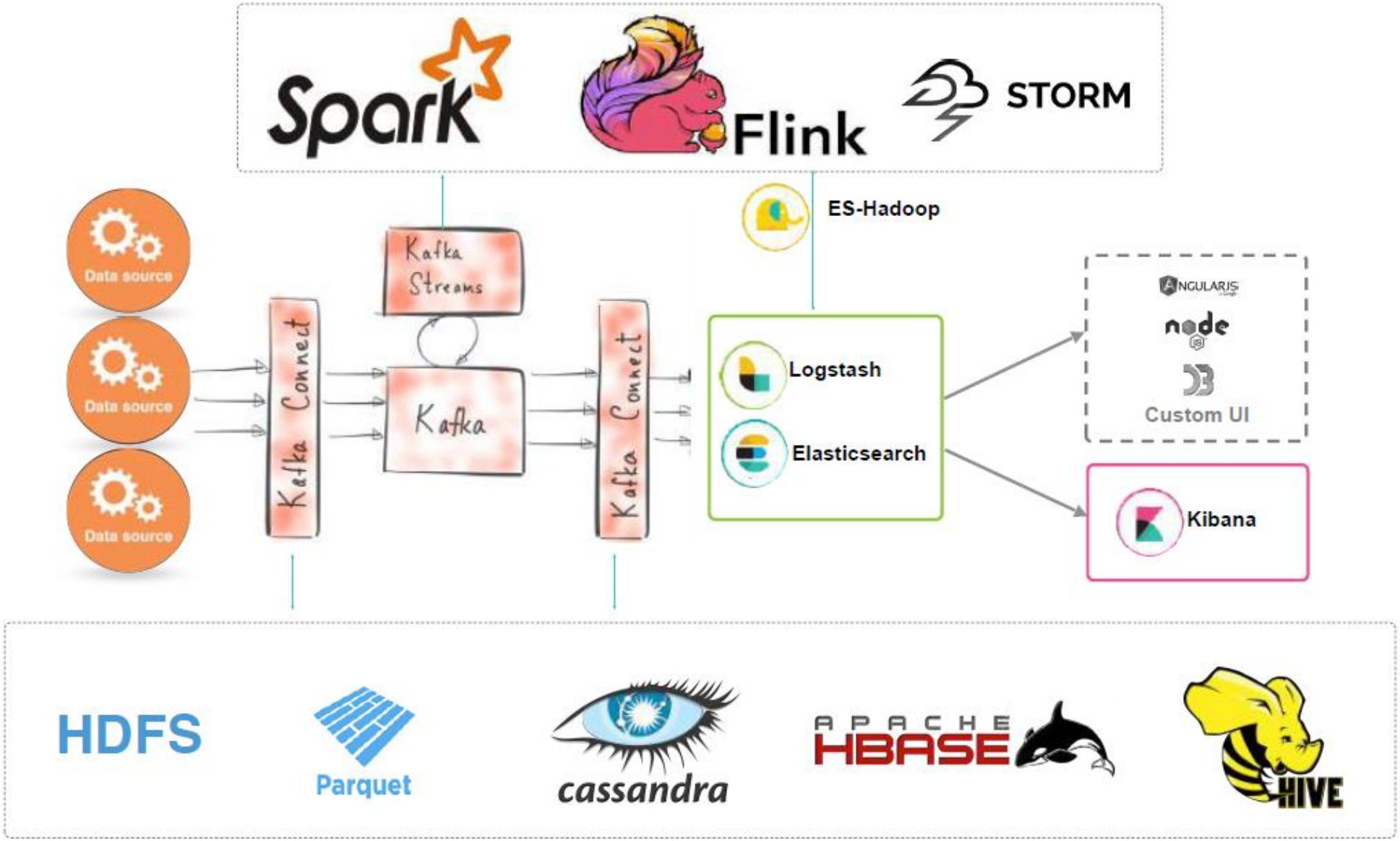
Kafka – No Code Stream Analytics



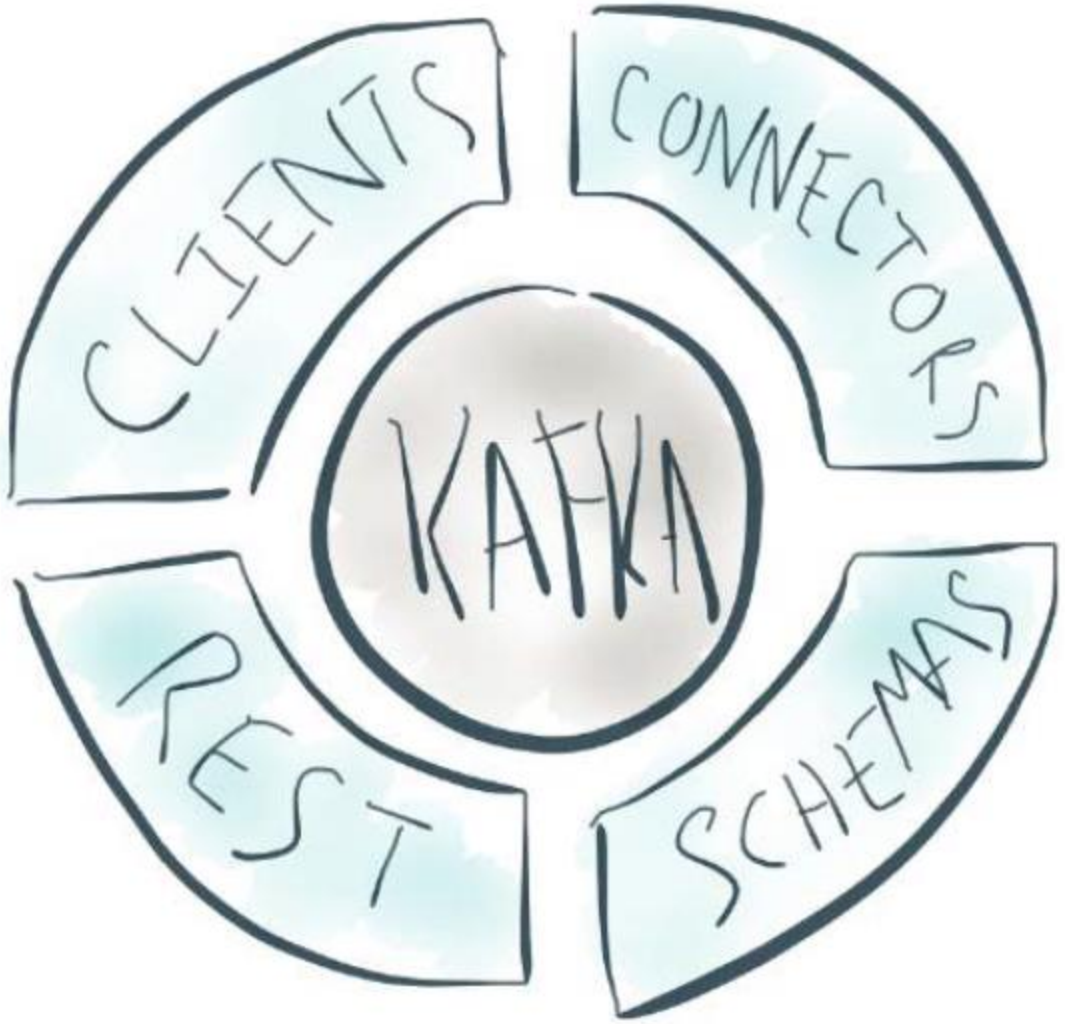
Kafka – No Code Stream Analytics



Kafka – ESEMPIO di un'architettura completa



Kafka – Confluent Open Source



Kafka – Confluent Open Source