Alma Mater Studiorum
Università di Bologna

Corso di Paradigmi di programmazione e sviluppo

# Report

Ravaglia Alex

# Indice

Il progetto presentato ha lo scopo di voler mostrare una possibile realizzazione di un sistema software realizzato con un'infrastruttura big data. Il sistema avrà in input un flusso di transazioni bancarie eseguite da diversi utenti. È richiesto che venga fornita una previsione sul fatto che la transazione in esame si tratti di un'operazione legittima o fraudolenta. Per la realizzazione sono state considerate diverse tecnologie per coprire le diverse funzionalità richieste. Nello specifico ci si è focalizzati nello studio di Spark, Kafka e Cassandra. Questo progetto è da intendersi come un prototipo più che un sistema reale vero e proprio. Per questo motivo alcune parti sono state realizzate in maniera semplice. Questo progetto può essere considerato come un buon punto di partenza e vuole per l'appunto essere una dimostrazione di come sia possibile integrare queste tecnologie per la realizzazione di sistemi che affrontino l'elaborazione real time di flussi di dati.

# 1 Spark

Apache Spark è un motore di elaborazione di big data open-source per eseguire computazioni su larga scala. Viene fornita un'interfaccia di programmazione che permette la realizzazione di applicativi studiati per essere eseguiti su un cluster di computer. Grazie a questo, i calcoli e i processi vengono eseguiti, implicitamente, in maniera parallela e distribuita, viene inoltre garantita tolleranza ai guasti ( entro certi limiti) . Spark può essere eseguito all'interno di diversi ambienti di esecuzione come Hadoop YARN, Mesos, EC2 o Kubernetes e accedere a diverse sorgenti di dati come HDFS, Apache Cassandra, Apache Hbase etc. Può essere considerato una valida alternativa al paradigma: map-reduce proposto in Hadoop. È possibile scrivere i programmi in diversi linguaggi di programmazione come: Java, Scala, Python, R e SQL. Tra le motivazioni che rendono Spark un candidato ideale per l'analisi di grandi quantità di dati, oltre alle performance, vi è uno stack di librerie che permette di eseguire compiti in diversi ambiti. Nell'ecosistema di Spark si trovano: Spark SQL, Spark Streaming, MLib e Graphx.

## 1.1 Spark SQL

È un modulo di Apache Spark che permette di lavorare con tipi di dato strutturato all'interno di un programma Spark. L'interrogazione può essere portata a termine tramite il linguaggio SQL oppure con un insieme di API che manipolano un tipo di dato *DataFrame*. Le sorgenti dati possono essere molteplici: Hive, Avro, Parquet, ORC, JSON e JDBC. Grazie alle informazioni relative alla struttura interna dei dati, rispetto le classiche RDD API di Spark, Spark SQL riesce a fare delle ottimizzazioni sui piani di esecuzione delle query, in questo modo vengono garantiti determinati livelli di performance.

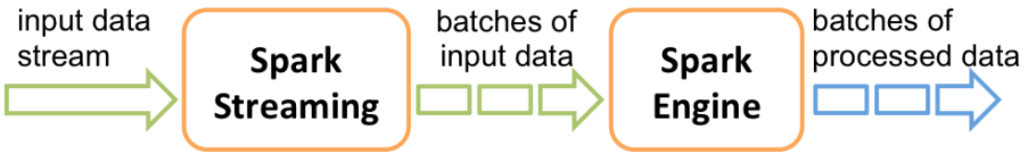
### 1.1.1 Modello dati: Dataset e DataFrame

Un Dataset è una collezione di dati distribuita. È un'interfaccia che fornisce i benefici degli *RDD*, tramite i quali è possibile sfruttare le potenzialità del lambda calcolo, con i benefici del motore di esecuzione ottimizzato di Spark sql. Un *Dataset* può essere costruito a partire da oggetti nella JVM e inseguito sarà possibile effettuare delle manipolazioni tramite trasformazioni funzionali ( map, filter etc. ). Un *DataFrame* è un *Dataset* organizzato secondo colonne. È equivalente a una tabella nel modello relazionale. Può essere costituito a partire da dati strutturati, tabelle Hive, database o RDD.

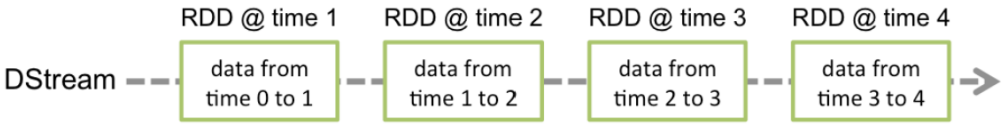
## 1.2 Spark Streaming

È un'estensione di Apache Spark che fornisce delle API per manipolare stream di dati real time. I dati possono entrare nel sistema da diverse sorgenti come: Apache Kafka, Kinesis o socket TCP. La manipolazione dei dati avviene esprimendo algoritmi complessi tramite funzioni ad alto livello come : Map,reduce, join e window. I dati processati possono infine essere memorizzati su filesystem o database.

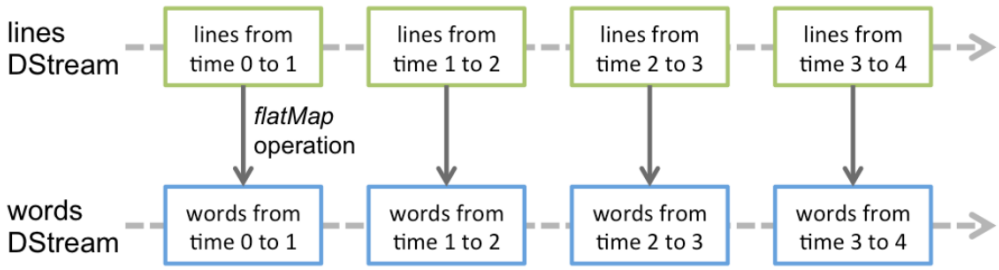

Spark Streaming ha come input uno stream di dati e partiziona questi dati in batch. Ogni batch viene processato da Spark e viene generato uno stream finale di risultati, frutto delle manipolazioni di ogni batch.




Spark Streaming rappresenta uno streaming di dati tramite l'astrazione: *DStream* sigla di: "discretized stream". Internamente un *Dstream* è rappresentato come una sequenza continua di *RDD*. Ogni *RDD* che compone un *Dstream* contiene i dati relativi a un certo intervallo.



Ogni operazione eseguita su un *Dstream* si riflette come un'operazione sui relativi *RDD*. Viene riportato un esempio in figura che mostra come avviene il processo di trasformazione da uno stream di righe di testo, a uno stream di parole tramite una trasformazione *flatMap*.

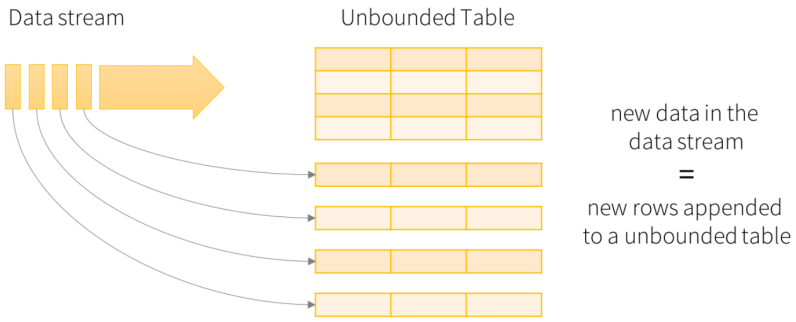



## 1.3 Structured Streaming

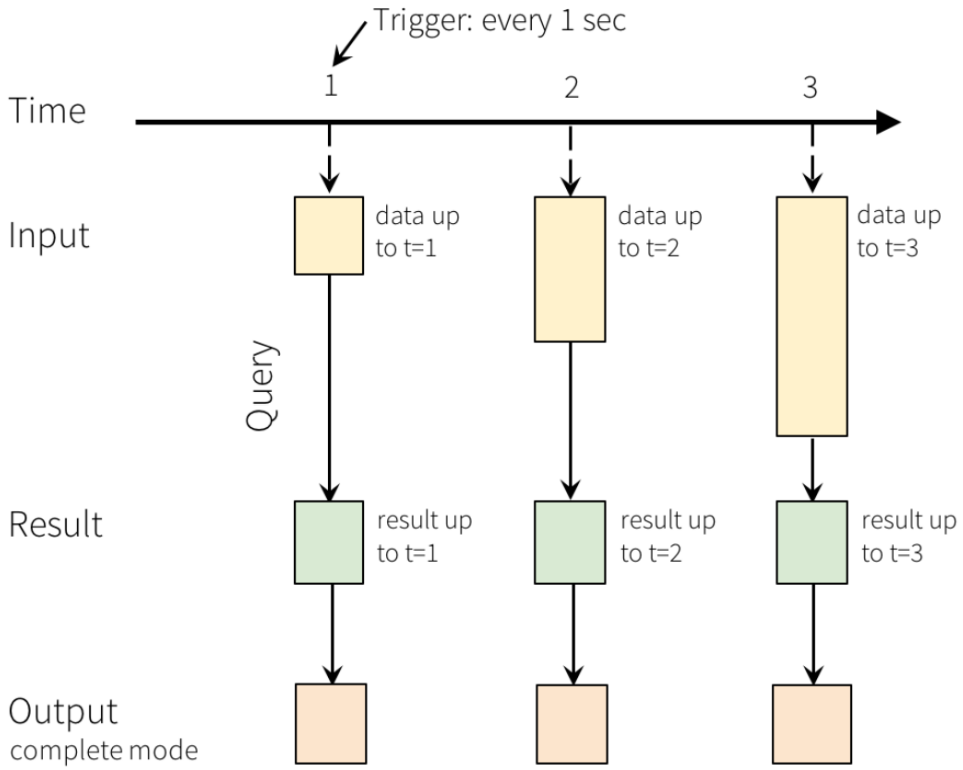
Structured streaming è costruito sopra il motore di esecuzione Spark SQL. Permette di esprimere computazioni su stream di dati come se fossero operazioni batch. Il motore di Spark SQL, si occupa di eseguire ripetitivamente le query e aggiornare il risultato nel mentre che nuovi dati continuano ad arrivare. Lo strumento fornito cerca di permettere al programmatore di astrarre la natura di un flusso di dati e fare si che si possano esprimere le computazioni come manipolazioni di *DataFrame* e *Dataset*. Internamente viene utilizzato un motore di esecuzione che si basa su micro-batch di dati. Gli stream vengono processati come una serie di piccoli job batch in modo tale da avere una bassa latenza ( 100 ms) e far si che sia garantita un semantica 'exactly-once'.

### 1.3.1 Tabelle "illimitate"

L'idea chiave in Spark Structured Streaming, è quella di avere delle tabelle alle quali vengono aggiunti i dati man mano che sono disponibili. La computazione può essere espressa come una query batch su una tabella statica. Spark si occupa di eseguire continuamente la query.




Una query sulla tabella in input genera una 'tabella risultato'. Ogni delta 't' di tempo, nuove righe vengono aggiunte alla tabella in input e questo permette che ci possano essere degli aggiornamenti sulla 'tabella risultato'. Ogni volta che vi è un aggiornamento nel risultato finale, questo cambiamento deve essere propagato su una eventuale sorgente dati esterna.




Considerata la natura 'illimitata' della tabella, questa non viene interamente materializzata. Vengono letti gli ultimi valori disponibili, dopo essere stati processati, viene aggiornato il risultato, successivamente i dati della tabella in input vengono cancellati. Sono mantenuti i dati che permettono di rappresentare uno stato intermedio dal quale partire per aggiornare il risultato con i nuovi dati.

## 1.4 Spark MLlib

Spark Machine Learning (ML) library, è una libreria il cui scopo è quello di voler portare il machine learning in maniera semplice e scalabile in un'infrastruttura cluster. Vengono forniti una serie di servizi, vi sono diverse tipologie di algoritmi per compiti quali: classificazione, regressione, clustering. Meccanismi per manipolare i dati come estrazione di feature, trasformazione di dati e riduzione della dimensionalità.

### 1.4.1 Modello dei dati

I modelli di riferimento per la rappresentazione dei dati in Spark sono i *DataFrame* e gli *RDD*. Le API e le funzionalità di libreria di SparkML sono disponibili per entrambi i modelli di dati. Anche se il supporto alle API basate su *RDD* è in'maintenance mode', continueranno a essere supportate, però non verranno aggiunte nuove estensioni e feature. SparkML consiglia di fare riferimento e/o preferibilmente migrare alle DataFrame-API. I motivi sono da ricercare nel fatto che viene fornita un'interfaccia più user-friendly, si ha un modello di API uniforme rispetto differenti algoritmi di Machine Learning e differenti linguaggi.

# 2 Kafka

Apache Kafka è una piattaforma streaming ad eventi. Lo scopo della piattaforma è quello di catturare i dati in ingresso in maniera real-time. Le sorgenti dei dati possono essere diverse: database, sensori, dispositivi mobili, servizi cloud o applicazioni software che pubblicano stream di eventi. Gli eventi devono essere intercettati e memorizzati per far si che possano essere gestiti, processati e vengano date delle risposte in tempo reale. Un'architettura a stream di eventi è un flusso continuo di eventi in ingresso nel sistema. Kafka può essere usato per diverse funzionalità in ambiente distribuito. Le principali sono:

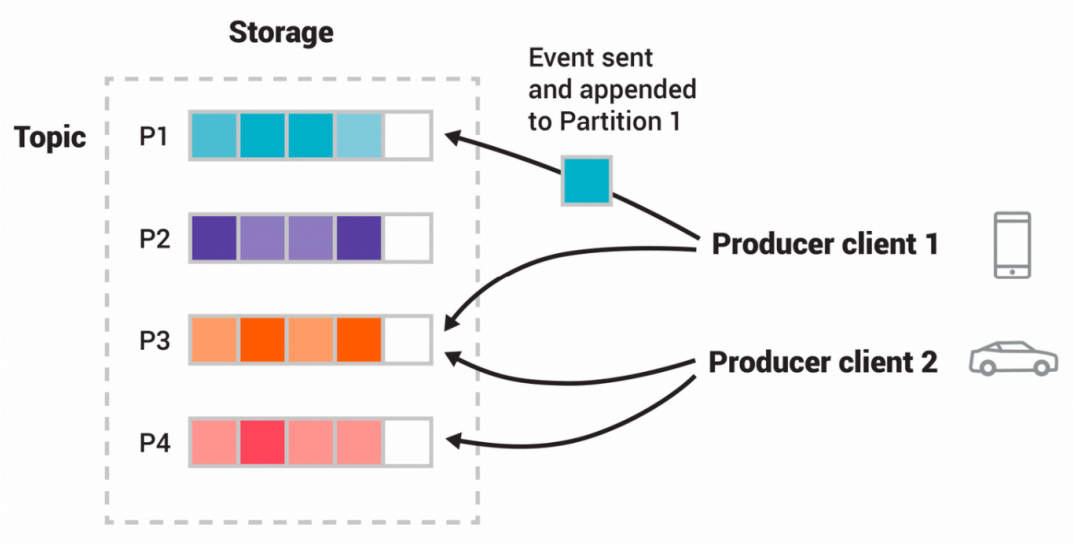
- Scrivere e leggere stream di eventi continui. Gli stream possono provenire o essere generati sia sullo stesso sistema che su altri.
- Memorizzare stream di eventi in modo tale che possano essere reperibili in qualsiasi momento.
- Processare stream di eventi.

## 2.1 Struttura

Kafka è un sistema distribuito che comprende server e client. Il server è da intendersi come un un cluster di uno più server. Alcuni di questi, formano dei 'Brokers',corrispondono al livello di memorizzazione dei dati. Altri server eseguono 'Kafka Connect', uno strumento per importare ed esportare stream di eventi. I client permettono di implementare applicazioni distribuite che effettuano operazioni in parallelo. Grazie a 'Kafka Streams' viene fornito un livello di API per trasformare gli stream di eventi. In alternativa possono essere utilizzate altre soluzioni tecnologiche come Spark, nelle versioni Spark Streaming o Structured Streaming.

## 2.2 Concetti principali

Un'entità Client che scrive dati in Kafka è un 'Producer', a differenza dei 'Consumer' che leggono gli eventi pubblicati. Uno dei principali elementi di design di Kafka è il fatto di avere Producer e Consumer completamente disaccoppiati. Gli eventi sono memorizzati in 'Topic', ovvero astrazioni che contengono gli eventi. I Topic in Kafka possono avere più producer e più consumer. Gli eventi non per forza devono essere cancellati dopo essere stati letti, si può definire per quanto tempo Kafka deve mantenere memorizzati questi dati.



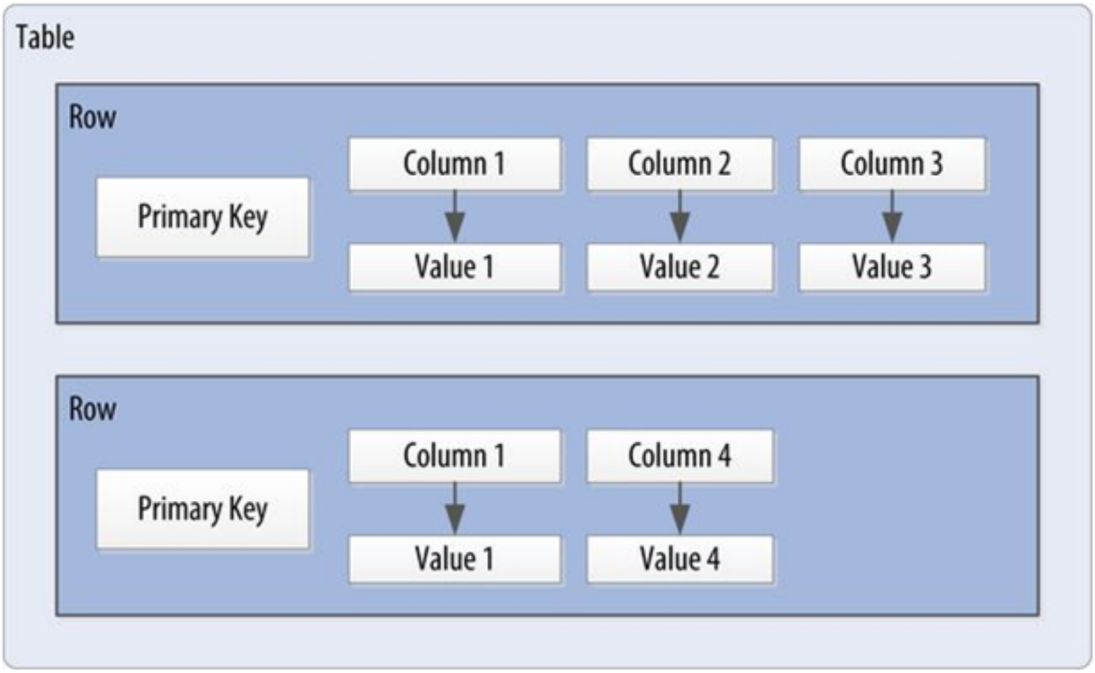

# 3 Cassandra

Apache Cassandra è una Database NoSQL di tipologia wide-column. Progettato per gestire grandi quantità di dati, fornisce un servizio con *high availability* e

decentralizzazione per evitare di avere single point of failure. Tra le principali feature di Cassandra vi è il fatto che è un database distribuito. I nodi del cluster sono replicati e i dati sono distribuiti tra i diversi nodi. Ogni nodo contiene dati differenti, essendo una soluzione decentralizzata non vi è un master e ogni richiesta può essere gestita da ogni nodo nella rete. La strategia e il numero di repliche possono essere opportunamente configurate, questo permette un certo grado di confidenza sulla tolleranza ai guasti. Come per altri database NoSQL, viene data priorità al partizionamento della rete e all' *availability* piuttosto che alla consistenza.

## 3.1 Modello dei dati

I dati sono memorizzati in tabelle composte da righe e colonne. Non è supportata la visione relazionale. Il modello dei dati è: "orientato alle query", è ottimizzato per le query che possono essere eseguite sui dati. I pattern di accesso e le interrogazioni ne determinano la struttura e l'organizzazione. Il modello dati è perciò ottimizzato su specifiche interrogazioni. Le query sono progettate per accedere ad una sola tabella, quindi tutti gli elementi di un'interrogazione devono essere al suo interno, così viene garantito un accesso in lettura molto veloce. Considerando che ad ogni query corrisponde una tabella, i dati vengono duplicati tra tabelle in un procedimento che prende il nome di: denormalizzazione. La scelta di una chiave primaria e una chiave di partizionamento sono importanti per la distribuzione dei dati all'interno di un cluster. La distribuzione in un cluster, avviene su nodi differenti, una chiave di partizionamento è usata per la ridistribuzione dei dati. Una chiave di partizionamento viene generata a partire dal primo campo della chiave primaria. Ogni riga è identificata da una chiave e ha più colonne, ognuna delle quali ha un nome, un valore e un timestamp. Differentemente dai modelli RDBMS, righe diverse all'interno della stessa tabella non devono condividere lo stesso set di colonne, una colonna può essere aggiunta a una o più righe in qualsiasi momento.



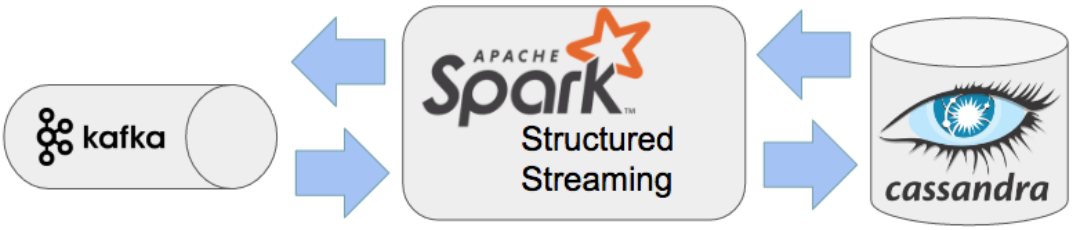

### 3.2 Cassandra Query Language - CQL

Il linguaggio fornito da Cassandra per eseguire delle interrogazioni è il "Cassandra Query Language - CQL". È simile a SQL, i dati sono messi in *tabelle* che contengono *righe* di *colonne*. Sono presenti diversi livelli di organizzazione: I "keyspace" definiscono come un dataset deve essere replicato, in quale datacenter e con quante copie. Una "Tabella" definisce il tipo di schema per una collezione di partizioni. Le "partizioni" definiscono la parte di chiave primaria che tutte le righe devono avere. Se le interrogazioni sono ottimizzate, il valore della chiave di partizionamento deve essere espresso nella query. Le "righe" contengono collezioni di colonne identificate da una chiave primaria. Infine la "colonna" è un singolo valore con un tipo specifico, appartiene ad una riga. I join non sono supportati.

## 4 Scelta delle tecnologie

La parte principale del sistema è stata realizzata in Spark. La scelta sul servizio per la manipolazione di stream di dati ricade su: Spark Streaming o Structured Streaming. Considerati entrambi i modelli supportati, da una parte si hanno gli *RDD* e dall'altra i *DataFrame*. Spark offre delle ottimizzazioni migliori sui piani di lavoro relativi a trasformazioni di *DataFrame*. È inoltre necessario considerare il modello predittivo da realizzarsi con MLlib. Considerato che supporta in maniera più completa i *DataFrame*, si è deciso di adottare la tecnologia: Spark Structured

Streaming. La persistenza dei dati viene eseguita da un Database NoSQL. Il motivo per il quale non si è optato per una soluzione RDBMS è il fatto che la priorità viene data alla bassa latenza e distribuzione del sistema. Negli scenari di lavoro che il sistema dovrà gestire, il fatto di avere consistenza è sicuramente un elemento di importanza, ha però la priorità la bassa latenza. Considerati diversi scenari di utilizzo, è più grave avere un sistema poco reattivo piuttosto che un sistema non sempre consistente. Tra le diverse soluzione NoSql, è stato scelto il database distribuito Cassandra. L'integrazione e la comunicazione con questo database viene fornita dalla libreria: Datastax. Il livello di trasmissione dei dati tra componenti viene realizzato tramite Apache Kafka.



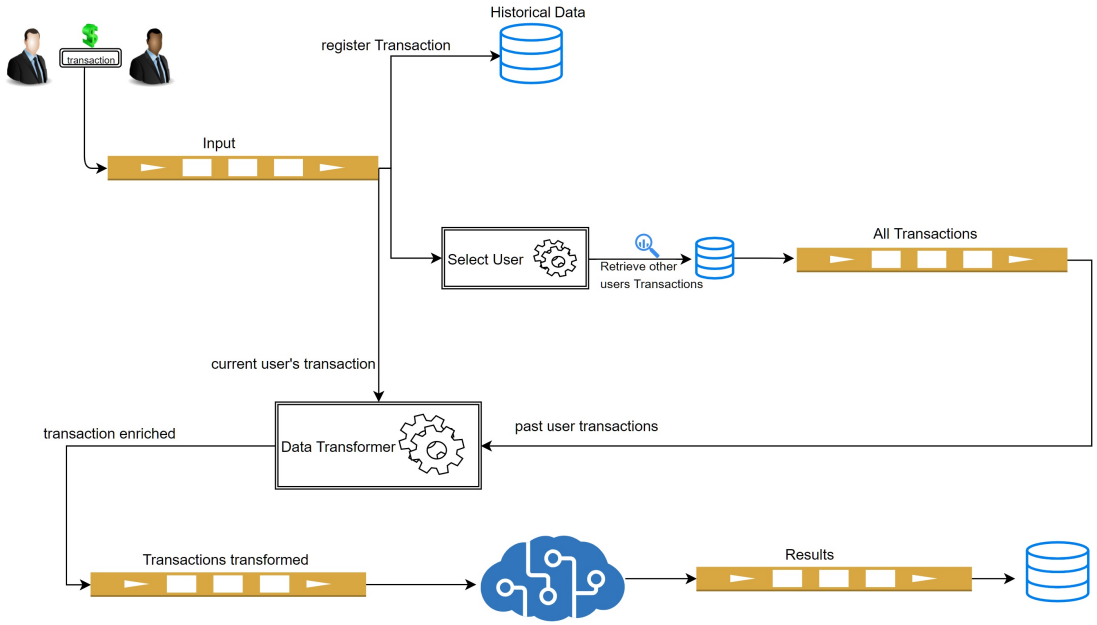

# 5 Architettura

Il sistema software progettato, deve essere una soluzione che possa permettere di gestire in input, flussi continui di dati. A seguire deve esservi una fase di manipolazione in modo tale da estrarre una serie di informazioni per la creazione di nuovi dati. È necessario che prima di trasformazioni complesse, i dati in ingresso debbano essere integrati con informazioni già memorizzate all'interno del sistema che compongono uno storico dati. Inseguito alla manipolazione e trasformazione, i nuovi dati andranno ad aggiornare lo storico del sistema. È infine richiesto al sistema di effettuare una classificazione binaria: *Frode* o *Non Frode*.

## 5.1 Architettura di sistema

L'architettura del sistema è composta principalmente da 3 entità. I topic si occupano di mettere i dati in ingresso e in transito nel sistema, in code ordinate. L'entità che si occupa della manipolazione dei dati è spark Structured Streaming. I dati sono prelevati dai topic e vengono elaborati mediante trasformazioni eseguite su *DataFrame*. Successivamente i dati trasformati sono scritti nuovamente su code

Kafka. L'ultima entità, identifica il livello di persistenza e gestione dello storico dati tramite Cassandra. Il database può essere sia un punto di memorizzazione dei dati, sia una sorgente di informazioni. Si possono eseguire delle interrogazioni e ricevere flussi finiti di dati pronti per l'elaborazione.




# 6 Design

Considerata la natura dinamica del sistema è opportuno identificare alcuni elementi principali e modellare il dominio di riferimento basandosi su questi. Sono stati identificati:

- Sorgenti dati.
- Flussi di dati.
- Trasformazioni.

Basandosi sulle 3 entità elencate è possibile avere una rappresentazione ad alto livello dell'intero sistema. Qualsiasi operazione può essere sintetizzata come una **trasformazione** che avviene su un **flusso continuo di dati** che vengono spostati da una **sorgente** ad un'altra. Per comprendere nello specifico il design di ogni entità è opportuno entrare più nello specifico nell'analisi di ogni componente.

## 6.1 Flussi di dati

Un flusso di dati è un'astrazione creata con lo scopo di voler catturare la natura dinamica degli stream di dati. Un flusso di dati legge i dati da una sorgente, ha un meccanismo interno per trasformare i dati e infine scrive i dati ottenuti in una sorgente differente. Possono essere identificati diversi flussi:

- **Input-stream** : Flusso che legge i dati in input al sistema. Partendo da questi, viene eseguita un'interrogazione a un database che detiene i dati storici. I dati vengono convogliati in un'altra sorgente e termina il primo flusso.

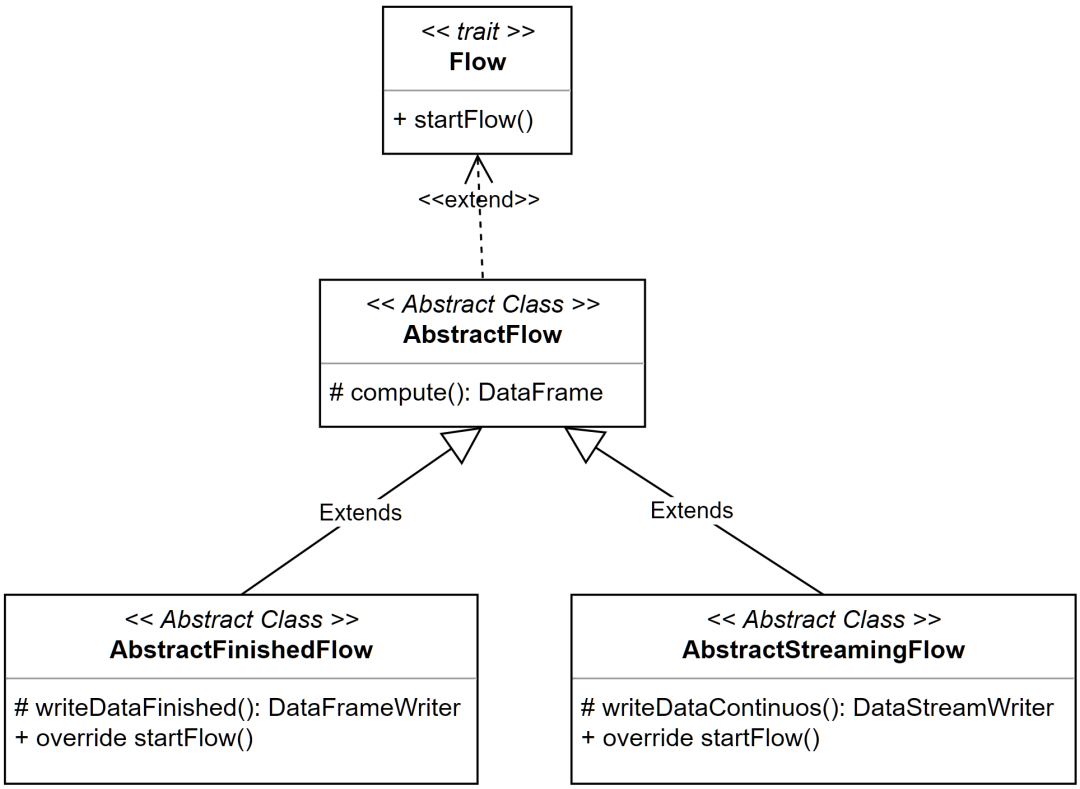
- **Register-Transaction** : Flusso che legge i dati in ingresso al sistema e li invia in una sorgente che rappresenta lo storico del sistema.

- **Data-Transformer** : Flusso che legge i dati da diverse sorgenti, effettua un 'merge' delle informazioni, applica una serie di trasformazioni che sono il frutto di un'elaborazione tra la transazione in arrivo e lo storico dati. Successivamente le transazioni trasformate e arricchite di nuove informazioni vengono inviate in un'altra sorgente dati.

- **Predict** : Flusso che legge i dati da una sorgente, effettua una previsione sulla base di un modello AI pre-addestrato e invia i risultati in una sorgente finale.

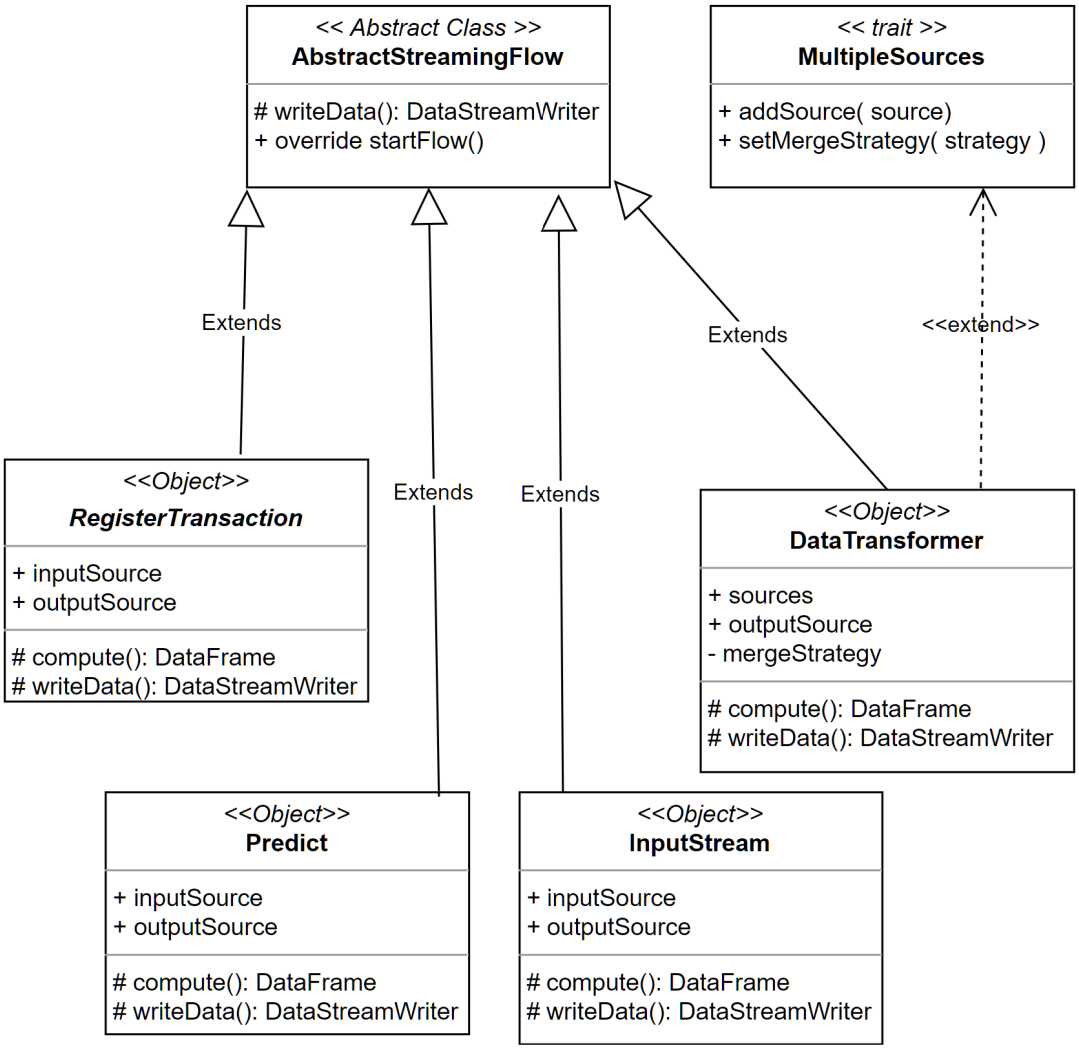
### 6.1.1 Astrazioni: *flussi di dati*

Sono presenti 2 tipi di flussi di dati, uno finito e uno potenzialmente infinito. Ne sono un esempio il flusso di dati ingresso al sistema che è progettato in modo da essere continuo, e il flusso dati relativo a un'interrogazione sql che sposta un quantitativo finito di informazione. A questo scopo sono state create 2 astrazioni: *AbstractFinishedFlow* e *AbstractStreamingFlow*. Entrambe le classi specializzano un'idea più astratta di flusso di dati: *AbstractFlow* e a loro volta vengono estese nelle implementazioni concrete dai diversi flussi che compongono il sistema.



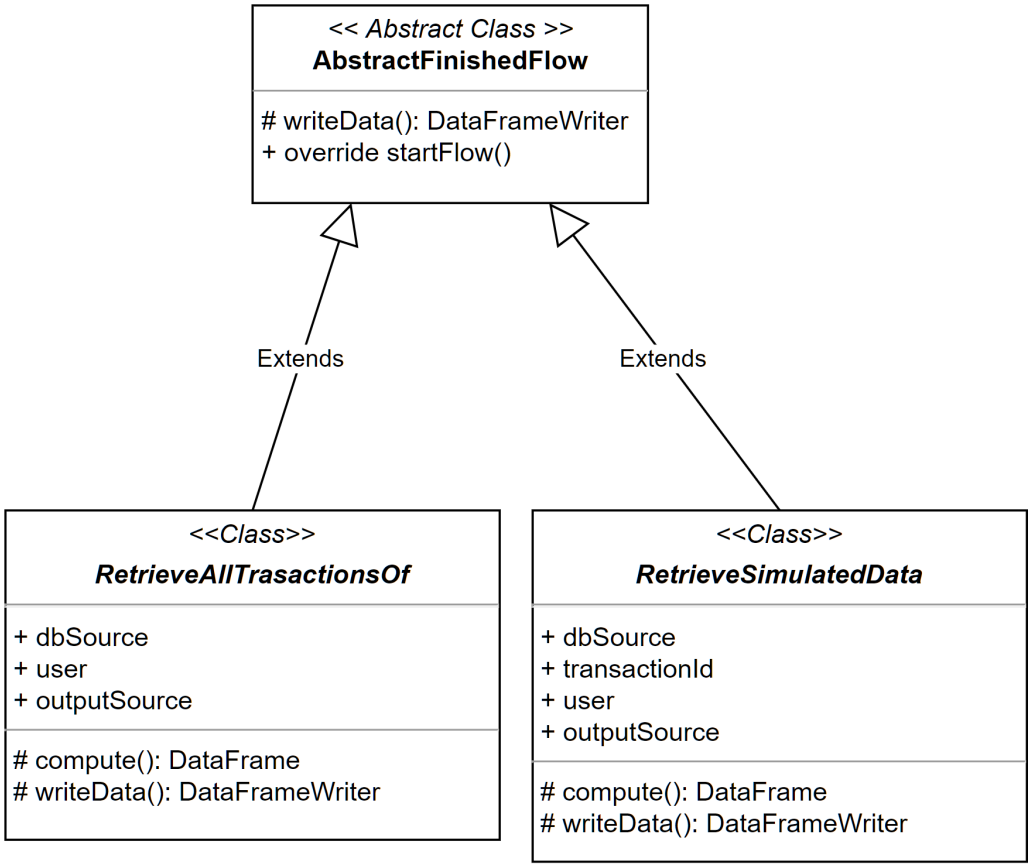

### 6.1.2 Implementazione concreta: *Flussi continui*

Sono rappresentate le implementazioni concrete dei flussi di dati continui. *Data-Transformer* ha la particolarità, che può attingere i dati da sorgenti diverse.



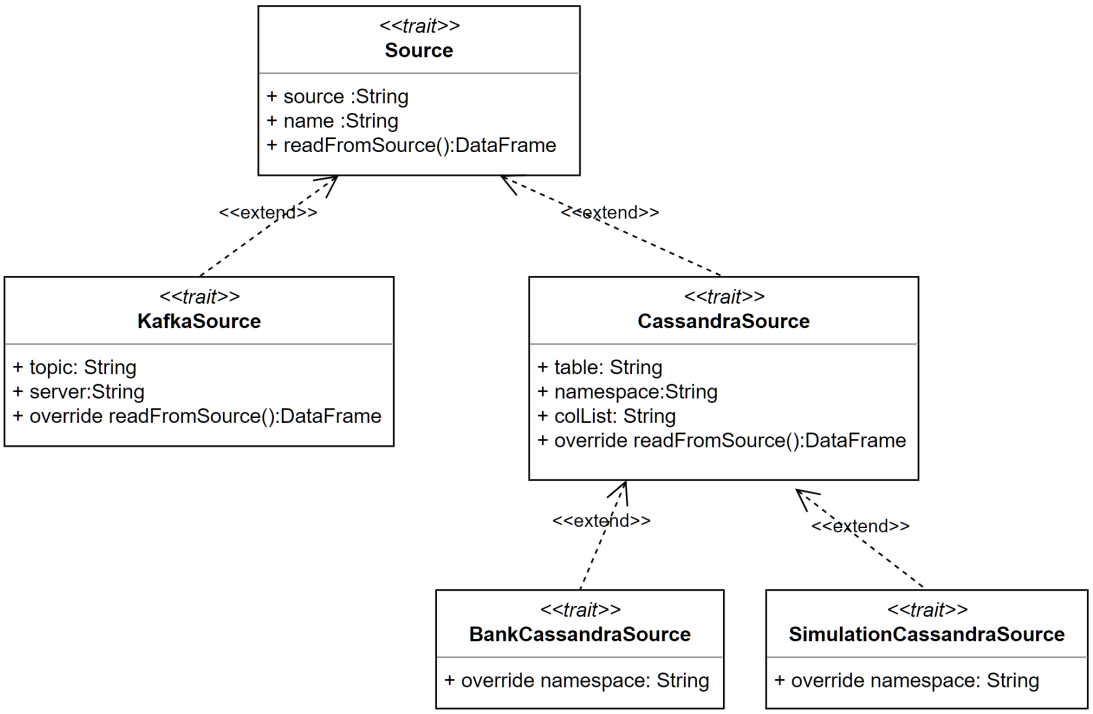

### 6.1.3 Implementazione concreta: *Flussi finiti*

Vi sono 2 tipi di implementazioni concrete dei flussi finiti. *RetrieveAllTransactionsOf* recupera tutti i dati relativi alle informazioni passate di ogni utente. *RetrieveSimulatedData* esegue un'interrogazione diversa e si occupa di recuperare dati 'simulati'.



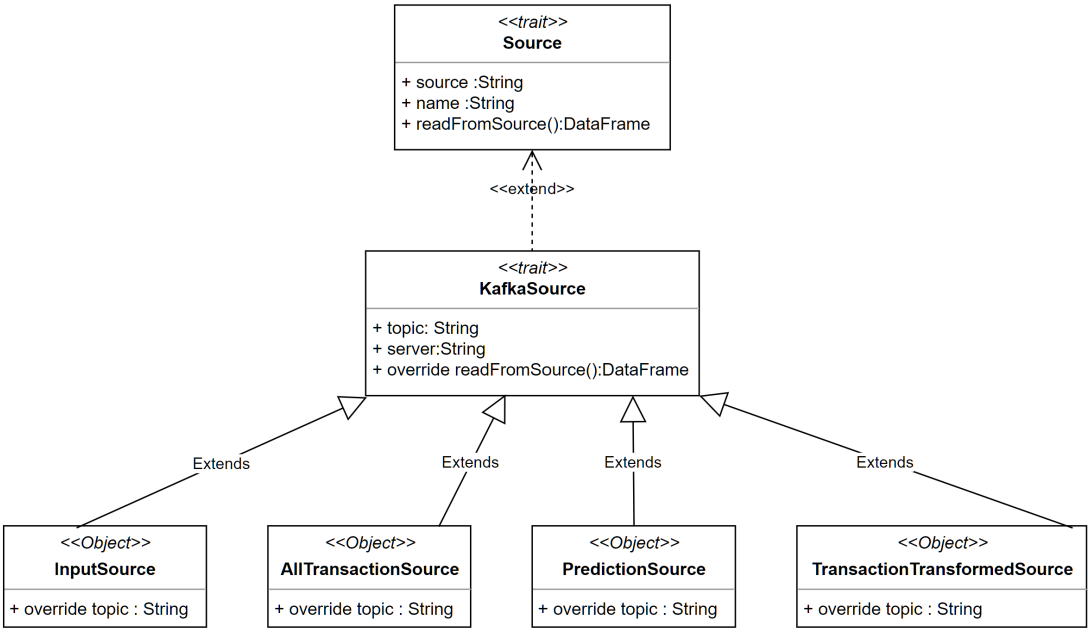

## 6.2 Sorgenti Dati

Una sorgente dati è un'astrazione creata con lo scopo di voler identificare un contenitore di uno stream di dati. È possibile interagire in 2 modi con una sorgente, si devono poter leggere i dati che custodisce e si deve poter scrivere su di essa. Ogni flusso di dati descritto nel paragrafo precedente ha inizio con l'operazione di lettura di uno stream da una sorgente e termina con un'operazione di scrittura su un'altra sorgente. La sorgente non è da considerarsi propriamente come una collezione di dati, bensì come elemento statico che permette l'interazione con una sorgente specifica. All'interno del sistema sono presenti 2 sorgenti: le code Kafka ( Kafka Topic ) e la base dati Cassandra.



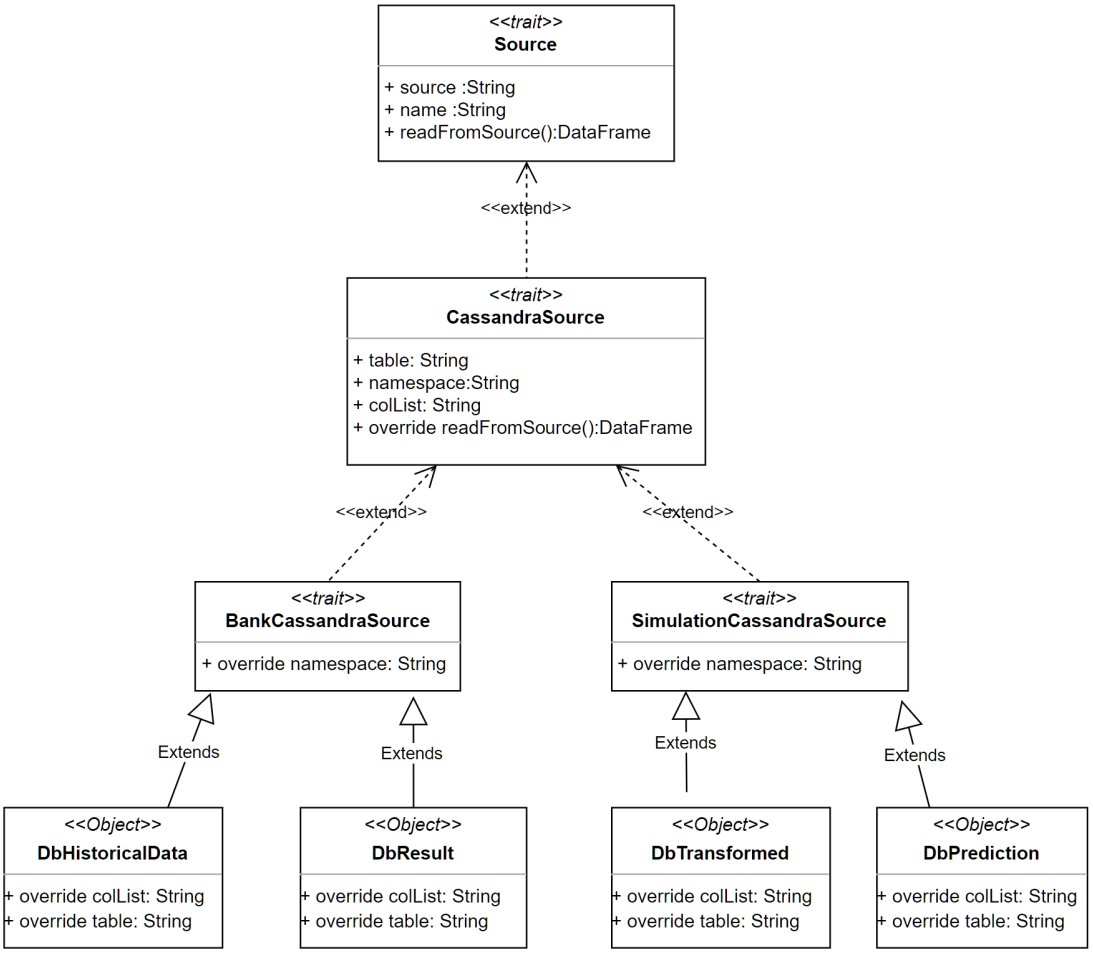

### 6.2.1 Kafka Topic

I Topic Kafka sono sorgenti che custodiscono i dati in transito tra flussi differenti. L'input del sistema è memorizzato in un topic, questi dati verranno consumati, elaborati e memorizzati in un'altra coda dove poi verranno consumati nuovamente da altri componenti. Anche i dati letti da un database possono essere registrati in un topic Kafka prima di essere manipolati e consumati. È il meccanismo più semplice per avere un punto di riferimento nello spostamento dei dati all'interno del sistema.



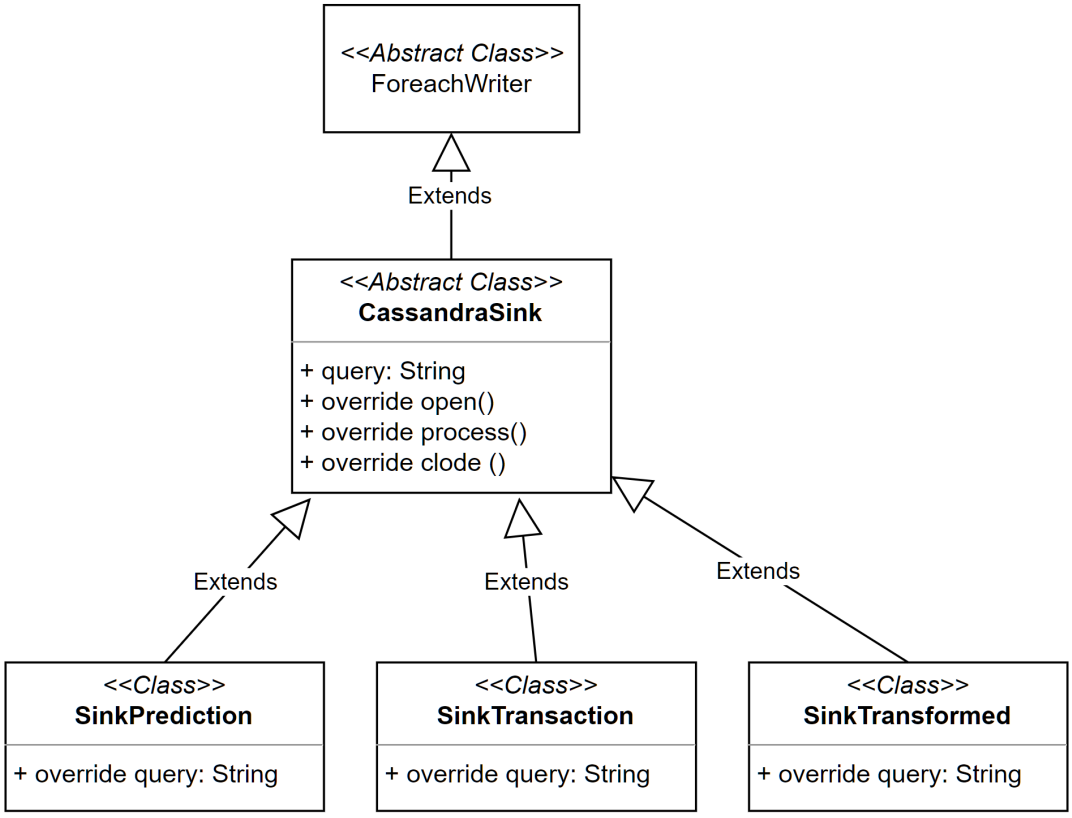

### 6.2.2 Database Cassandra

La base dati Cassandra svolge le duplici funzioni richieste a una sorgente: lettura e scrittura. Inseguito a una serie di trasformazioni e risultati ottenuti, il database funge come *layer* per la persistenza delle informazioni. Ogni nuova transazione in ingresso al sistema deve essere memorizzata in modo da essere reperibile per interrogazioni future. In maniera analoga può essere rilevante memorizzare i risultati finali ottenuti. La seconda funzionalità è la lettura dei dati. Il sistema può interrogare la base dati per richiedere una serie di informazioni, ogni richiesta genera una risposta finita e i dati potranno essere consumati direttamente dal sistema o pubblicati in una coda kafka e poi consumati successivamente interagendo con il Topic.




### 6.2.3 Operazione di scrittura

Un'interrogazione al database può essere indotta da ogni singolo input nel sistema. Si pensi per esempio che per ogni record letto, questo deve essere memorizzato nella base dati persistente. Per fare questo è richiesto un componente specifico che dia un'implementazione della query di inserimento di uno specifico tipo di dato. Viene quindi specificata la connessione con la base dati e l'esecuzione dell'interrogazione. Grazie al pattern 'Template Method' la definizione specifica della query viene fornita in sottoclassi specializzate.



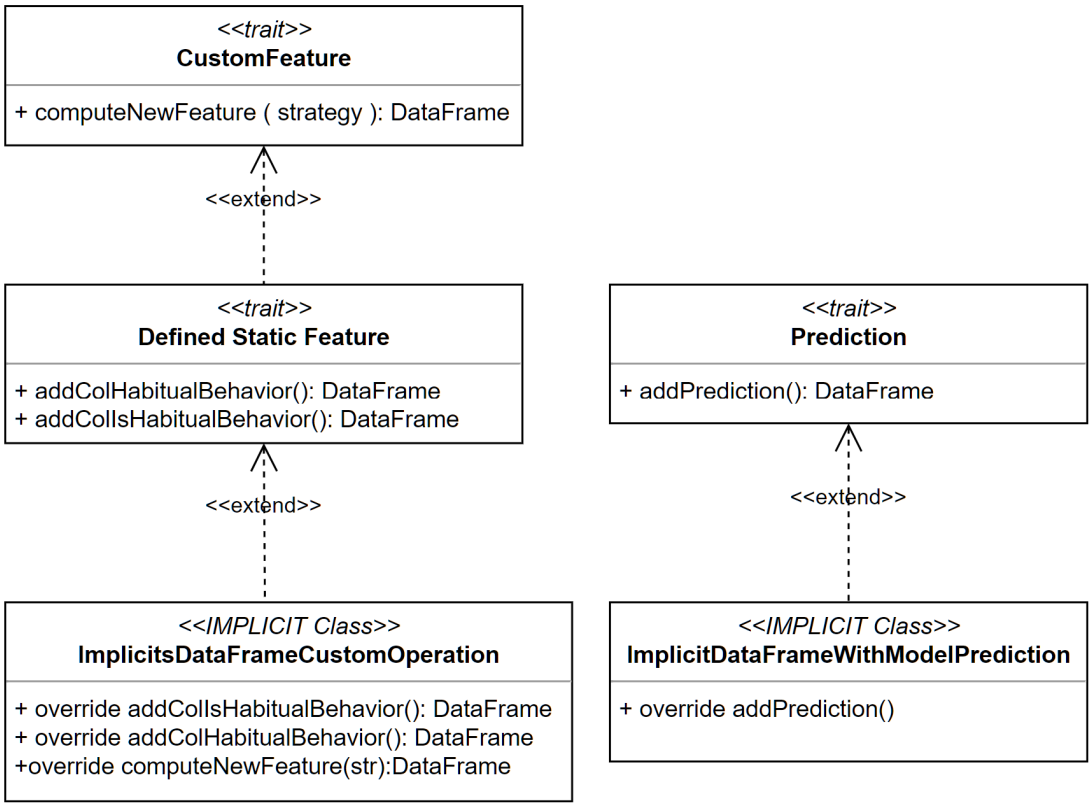

## 6.3 Trasformazioni

Le trasformazioni sono tutte le operazioni che vengono eseguite all'interno di un flusso di dati. Determinano tutte le manipolazioni che possono essere eseguite sui dati. Nel prototipo proposto, le operazioni di trasformazione e il modello di intelligenza artificiale sono stati realizzati fornendo delle soluzioni molto semplici. Sarà frutto di estensioni future andare ad estendere questa parte. È possibile principalmente definire 2 tipi di trasformazioni :

- La manipolazione di un *DataFrame* per aggiungere nuove feature.
- La fase predittiva che verrà eseguita da un modello di intelligenza artificiale pre-addestrato.

### 6.3.1 Design Trasformazioni - Impliciti

Per aggiungere una serie di funzionalità più complesse si è deciso di estendere le operazioni con le quali è possibile manipolare un *DataFrame*. Ciò è stato possibile grazie all'utilizzo degli *impliciti*. Definendo una classe implicita in modo da estendere i *DataFrame*. È possibile definire una serie di funzionalità 'custom' che andranno ad estendere l'espressività delle operazioni eseguibili su un *DataFrame*. Le trasformazioni principali prevedono di aggiungere ad ogni *Dataframe* un insieme di colonne frutto di combinazioni delle colonne già presenti. Questo procedimento viene definito come 'feature engineering'. Grazie alla nuova interfaccia realizzata sarà possibile definire un'operazione di trasformazione per ogni feature da aggiungere. Viene definita anche un'operazione di trasformazione 'custom' tramite la quale si prevede che la strategia di trasformazione sia passata in input. In modo analogo all'estensione per le operazioni di trasformazione, viene fatta anche un'estensione per gestire la fase predittiva.




## 6.4 merge strategy

Il principale processo di feature engineering dei dati, può dover trasformare i dati provenienti da diversi sorgenti differenti. Motivo per il quale prima della

trasformazione deve esserci una fase di join degli stream di dati. Il meccanismo prende il nome: ‘merge‘ e la strategia può essere definita in maniera ‘custom ‘ dal programmatore, definendo una funzione di trasformazione da passare in input a questa entità. In alternativa vi può essere la possibilità di usare delle funzionalità di fusione degli stream basandosi su delle strategie di ‘default‘ fornite tramite un’interfaccia.

# 7 Altri componenti

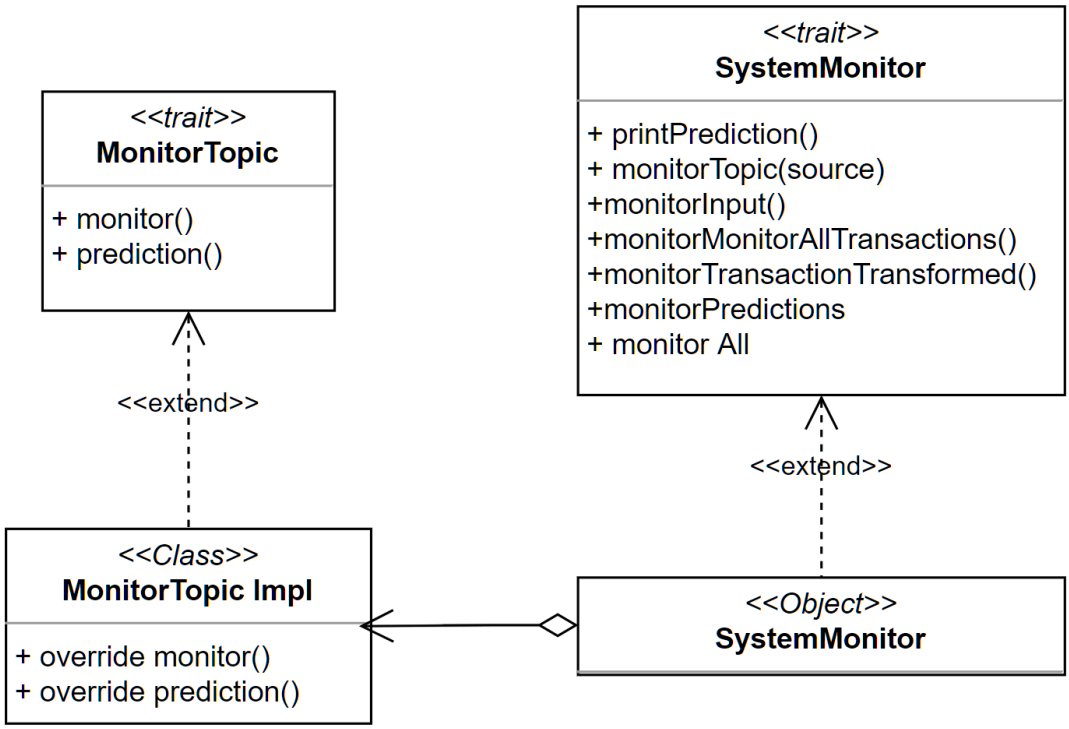
Il sistema è composto da altri componenti minori. Un sistema per monitorare lo stato delle code di dati e un meccanismo per inviare dati all’interno del sistema simulando diversi utenti che generano delle transazioni da analizzare.

## 7.1 Input simulato

Per simulare un caso d’uso reale si è deciso di leggere i dati relativi alle transazioni bancarie da un file ‘.csv‘ e inviarli al topic Kafka che cattura gli eventi in ingresso al sistema.

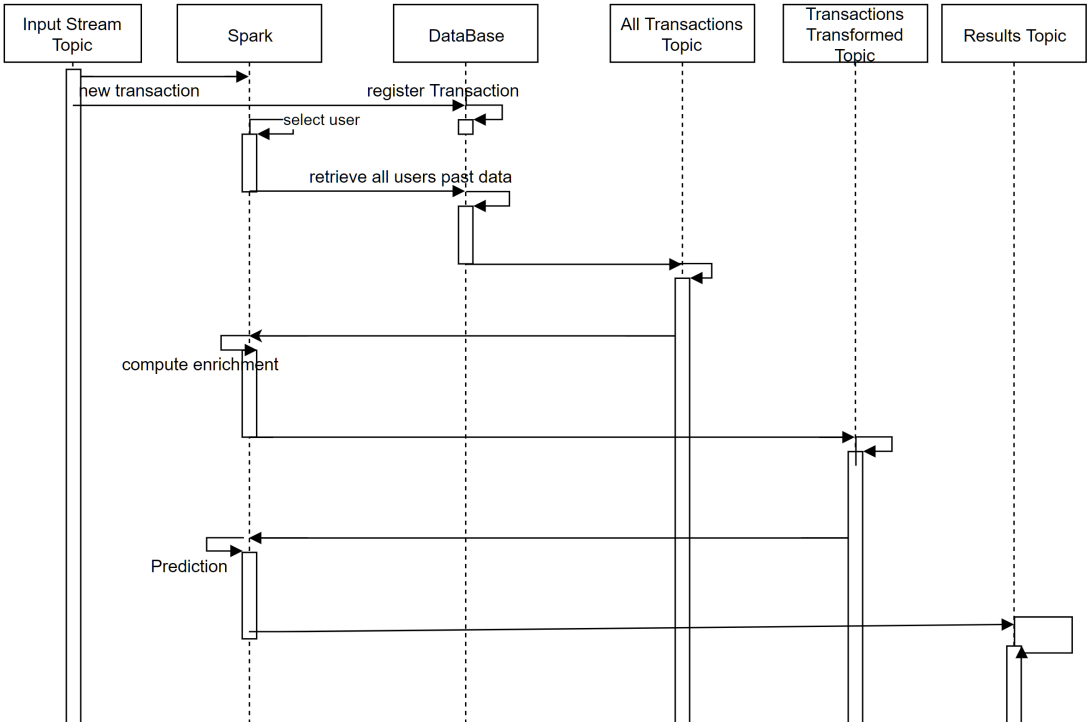
## 7.2 Sistema di monitoraggio

Tramite il componente *SystemMonitor* è possibile monitorare le code Kafka. È possibile in qualsiasi momento analizzare lo stato delle code stampandone il loro contenuto su *stdout* . Tramite questo componente, è inoltre possibile stampare il risultato delle previsioni.




# 8 Design delle interazioni

Viene descritto il normale flusso di esecuzione. L'input del sistema è costituito da transazioni bancarie eseguite da diversi utenti. Ogni transazione deve essere classificata come: *Transazione legittima* o *Transazione fraudolenta*. La transazione in ingresso viene inserita in una coda Kafka. L'informazione deve essere propagata a un database per arricchire la base dati che detiene lo storico dell'utente. È opportuno che vengano recuperate dal database tutte le transazioni dello specifico utente. In questo modo avendo la transazione corrente e lo storico delle transazioni passate sarà possibile creare una transazione 'arricchita' che abbia una serie di informazioni relative alla relazione dell'operazione corrente rispetto le altre operazioni passate. Queste informazioni aggiuntive sono di primaria importanza ai fini della classificazione che verrà eseguita da un modello di intelligenza artificiale ( attualmente non implementato nella soluzione corrente). Inseguito alla previsione il risultato può essere semplicemente stampato in output o memorizzato su un supporto che fornisca persistenza (es. database).




# 9 Validazione

Si è deciso di analizzare diversi meccanismi al fine di comprendere se il tutto funzionasse correttamente. Nello specifico, i controlli sono stati eseguiti analizzando:

- Se il sistema funziona senza incorrere in crash o malfunzionamenti improvvisi. Valutandone la *reliability*.
- Se i dati vengano correttamente spostati tra le diverse sorgenti dati.
- Se a fronte di un determinato input, vi è l'output atteso.

## 9.1 Valutazione *Reliability*

Si è deciso di dare una valutazione su quanto il sistema si potesse considerare *reliable*. È stato valutato avviando la simulazione e misurando il tempo durante il quale il sistema è rimasto disponibile. Avendo un flusso di transazioni da classificare, si è misurato quanto tempo intercorresse prima che si verificassero problemi, malfunzionamenti o crash del sistema. Sono stati consumati tutti i dati della simulazione ( 45 min circa) e non si sono verificati problemi. Il medesimo test è stato eseguito 5 volte. Considerati gli esiti positivi si ritiene che il sistema si possa considerare sufficientemente *reliable* e *available*.

## 9.2 Spostamento dati

Uno dei compiti principale del sistema è quello di spostare i dati tra diversi componenti. I dati entrano nel sistema e vengono intercettati da spark, in seguito a una prima elaborazione vengono spostati su una coda kafka e poi sul database. Successivamente vengono nuovamente letti da un altro flusso di dati e dopo eventuali manipolazioni, vengono memorizzati su altre sorgenti. Un aspetto importante che necessita un'analisi specifica è la valutazione se ogni passaggio di dati tra una sorgente e l'altra avviene nella maniera corretta. È un meccanismo abbastanza complesso da monitorare, per questo motivo è stato realizzato il componente: *SystemMonitor* descritto nel paragrafo precedente. Tramite questa entità è possibile in qualsiasi momento analizzare i dati in transito in qualsiasi sorgente dati. Si può infatti stampare su standard output il continuo aggiornamento di ogni coda di dati. In questo modo analizzando il contenuto di ogni topic si è valutato se gli spostamenti dei dati avvenissero in maniera legittima.

## 9.3 Output Atteso

Il sistema ha un unico obiettivo : classificare le transazioni bancarie in input come 'fraudolenti' o 'legittime'. È richiesto che a fronte di un input atteso: ovvero una transazione bancaria, in output si abbia la classificazione desiderata. Per analizzare questo aspetto si possono astrarre tutti i procedimenti e operazioni intermedie. Se il sistema nel complesso lavora in maniera corretta, a fronte di un input, si deve avere il corrispondente output. Avviando la simulazione vengono lette una serie di transazioni da file e il sistema effettua la classificazione del flusso continuo. In figura è rappresentato un frammento di file in input con alcune delle transazioni che verranno processate. Sempre in figura è possibile osservare evidenziati l'identificativo dell'utente e un identificativo più lungo corrispondente alla transazione specifica.

```
10 64,41,1,69,288,254,32.232,0,16,21,2,38,1,9,11,10,2018-05-27 18:53:35,3566497
11 35,51,0,17,274,97,85.0,0,149,17,2,1,0,7,41,25,2018-05-27 20:03:01,3566667
12 64,12,1,69,72,871,8.3,0,120,21,2,38,1,9,14,13,2018-05-27 20:33:46,3566750
13 35,31,0,32,274,254,100.0,1,188,17,4,40,0,7,11,10,2018-05-27 20:41:35,3095009
14 64,70,1,69,292,201,7.747000000000001,0,120,21,2,38,1,9,11,10,2018-05-27 20:46:09,3566787
15 64,41,1,69,218,214,29.879,0,45,4,2,38,1,9,14,13,2018-05-27 22:24:28,3567037
```

A seguire in figura il risultato delle transazioni dopo essere state classificate.

```
+----+-------------+---------------------------------+
|user|TransactionID|Prediction ( 0 - 1 ) : Ok - Fraud|
+----+-------------+---------------------------------+
| 871|      3566750|                                0|
+----+-------------+---------------------------------+

-------------------------------------------
Batch: 2590
-------------------------------------------
+----+-------------+---------------------------------+
|user|TransactionID|Prediction ( 0 - 1 ) : Ok - Fraud|
+----+-------------+---------------------------------+
| 254|      3095009|                                1|
+----+-------------+---------------------------------+

-------------------------------------------
Batch: 2591
-------------------------------------------
+----+-------------+---------------------------------+
|user|TransactionID|Prediction ( 0 - 1 ) : Ok - Fraud|
+----+-------------+---------------------------------+
| 201|      3566787|                                0|
+----+-------------+---------------------------------+

-------------------------------------------
Batch: 2592
-------------------------------------------
+----+-------------+---------------------------------+
|user|TransactionID|Prediction ( 0 - 1 ) : Ok - Fraud|
+----+-------------+---------------------------------+
| 214|      3567037|                                0|
+----+-------------+---------------------------------+
```

Analizzando la corrispondenza tra i dati in ingresso e i dati in output si è valutato che il comportamento fosse quello desiderato.

# 10 Conclusione

Tramite il sistema descritto si è deciso di presentare una possibile soluzione e architettura per gestire la casistica di manipolazione di flussi di dati all'interno di un sistema reale da gestire con tecnologie big data. Ci si è focalizzati sull'architettura generale e le interazioni tra componenti, realizzando un sistema funzionante. Tuttavia per avere a un sistema completo è opportuno fare delle modifiche. Soprattutto per quanto riguarda la fase di integrazione di un modello di intelligenza artificiale e operazioni di feature engineering.