Tesina di
Big Data Management
Corso di Laurea Magistrale in
Ingegneria Informatica e Robotica
curriculum data science
A.A. 2022/23
docente
Fabrizio Montecchiani

Kafka-speed car project

350395 **Filippo Francisci** email: filippo.francisci@studenti.unipg.it

Sommario

1. Introduzione 1

2. *Dataflow* e tecnologie utilizzate 4

2.1 *Dataset* e formato input 6

2.2 Tecnologie utilizzate 6

2.3 *Dataset* e formato *input* 6

3. Casi d'uso 4

4. Limiti e possibili estensioni 4

1. Introduzione

L'obiettivo di questo progetto è quello di recuperare informazioni da vari sensori in grado di rilevare la velocità mantenuta da un'automobile in corsa ed effettuare varie misurazioni su di esse. Per fare ciò si è utilizzato il modello di calcolo distribuito ***Spark***, integrandolo con la tecnologia ***Kafka***, particolarmente utile nella gestione di dati *real-time*. In seguito le misurazioni effettuate vengono restituite in output direttamente su console. Si è scelto di usare linguaggio di programmazione *Python* per semplicità di implementazione.

2. *Dataflow* e tecnologie utilizzate

2.1 *Dataset* e formato di *input*

Il *dataset* impiegato è stato realizzato tramite uno *script* apposito (***car-data.py***) che permette di generare informazioni in modo randomico. I dati vengono prodotti direttamente in formato ***JSON***. Ogni singolo dato viene definito come un oggetto ***JSON*** caratterizzato dai seguenti campi:

- ***id-sensor***: identificativo univoco per il sensore di cattura;
- ***destination***: identificativo per la destinazione dei dati;
- ***road_type***: tipologia di strada percorsa dall'automobile (strada urbana, strada extraurbana secondaria, strada extraurbana principale, autostrada);
- ***speed_limit***: limite di velocità imposto sulle corrispondenti strade;
- ***model***: modello di auto;
- ***plate***: targa dell'automobile;
- ***time***: data e ora in cui il dato è stato raccolto;
- ***speed***: velocità raggiunta dall'auto in corsa.

Prima di poter operare su questo file ***JSON***, le informazioni vengono pubblicate sul *broker* ***Kafka*** per poi essere consumate.

2.2 Tecnologie utilizzate

Le tecnologie utilizzate sono le seguenti:

- ***Spark***
- ***Kafka***
- ***Zookeeper*** (fornisce un servizio di sincronizzazione per Kafka)

- ***Python*** (linguaggio di programmazione usato)
- ***Pyspark*** (Interfaccia per Apache Spark in Python)

2.3 Architettura e *dataflow*

Innanzitutto è necessario creare i dati randomici, che simulano una raccolta dati *real-time* da dover gestire. All'inizio vengono avviati i servizi forniti da ***Zookeeper*** e ***Kafka*** con l'apposito *script* ***start-all.bash***. A questo punto basta eseguire lo *script* ***run-all.bash*** che permetterà di operare sui dati ***JSON*** prodotti su *broker Kafka*. In seguito si usa Structured Streaming integrato con Kafka per recuperare queste informazioni: tramite metodo *readStream* si sottoscrive per il *topic* di interesse, si eseguono poi le operazioni necessarie e infine con *writeStream* si restituisce il risultato in *output*.
L'operazione che si è deciso di realizzare è quella di recuperare semplicemente informazioni relative alle sole auto che hanno superato i limiti di velocità imposti sui tipi di strada dove i dati sono stati raccolti.
I risultati vengono mano a mano stampati su console in batch consecutive in base a quanti dati si decide di inserire.

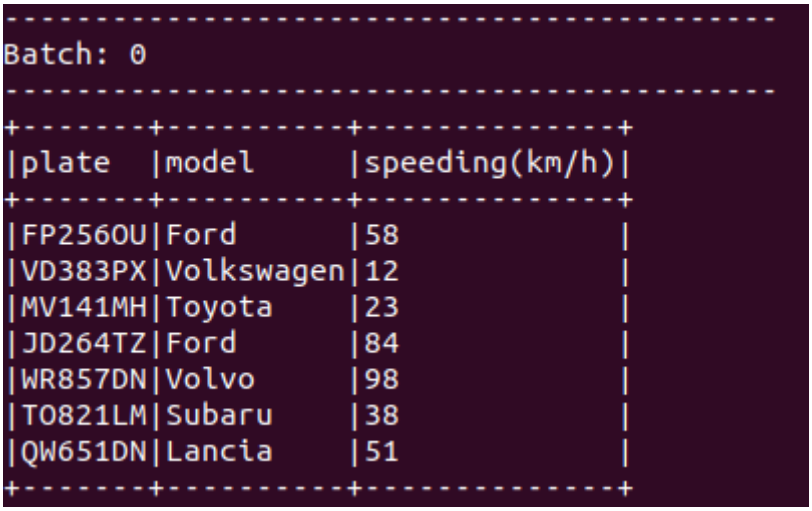
3. Casi d'uso

Per poter utilizzare l'applicazione è necessario innanzitutto scaricare ed installare ***Kafka***. Fatto ciò, per avere un *deployment* più semplice, è utile estrarre la cartella Kafka_2.12-3.3.2 sulla *root* della cartella *project*. Per eseguire il progetto sono stati sviluppati degli script *.bash* che automatizzano il processo descritto in precedenza.
Per prima cosa si esegue lo script *start-all.bash* che in successione avvia varie componenti di interesse: *Zookeeper* e *Kafka broker*. Poi viene creato sul broker il *topic testtopic* in cui si andranno a pubblicare le informazioni ottenute. Se si riscontra qualche errore di esecuzione, è probabile che ci sia ancora un processo *Zookeeper* in esecuzione che va chiuso usando eventualmente il seguente comando su *shell*:

```
sudo ./kafka_2.12-3.3.2/bin/zookeeper-server-stop.sh kafka_2.12-
3.3.2/config/zookeeper.properties
```

Per avviare l'applicazione è necessario richiamare lo script *run-all.bash* che funziona in questo modo: chiede all'utente di inserire un numero di dati che verranno quindi prodotti casualmente e assegnati al *topic* sopra descritto; dopodiché su un ulteriore terminale viene avviato lo script ***kafka-spark-car.py*** che raccoglie i dati passati, restituendo i risultati richiesti. Nel frattempo all'utente viene richiesto se ha intenzione di aggiungere nuovi dati:

se sì allora vengono aggiunti e la computazione continua anche su questi, altrimenti all'utente viene data la possibilità di uscire dall'applicazione.
Un possibile output viene riportato di seguito:



Una volta terminata l'esecuzione, lo *script stop-all.bash* permette di chiudere tutti i processi attivi.

4. Limiti e possibili estensioni

Il progetto che si è realizzato è una semplice simulazione di analisi di dati *real-time*. Una prima limitazione è rappresentata dal fatto che i dati considerati sono dati fittizi e creati in modo randomico per scopi didattici. L'impiego di dati reali permetterebbe ovviamente osservazioni più veritiere. Inoltre molti sono i miglioramenti che potrebbero essere apportati. Ad esempio si potrebbe pensare di realizzare anche altre diverse computazioni da restituire in *output*. Il progetto realizzato potrebbe essere sviluppato implementando un *database* in cui dopo aver computato adeguatamente i dati, si possono andare a conservare le informazioni che sono di maggiore interesse, per un possibile utilizzo futuro.