Praktikum 4 Apache Spark – Kafka
Data Frames und Structured Streaming

Ziel des Praktikums ist das Verständnis für die gemeinsame Programmierschnittstelle, die **Apache Spark** für seine Komponenten *Spark SQL und Data Frames* sowie *Spark Structured Streaming* seit der Version 2.0 zur Verfügung stellt.

Apache Kafka kommt hierbei als Messaging System zur Anwendung. Es kann sowohl als *Producer* als auch *Consumer* und *Transformer* von Datenströmen eingesetzt werden und stellt *Connectoren* zur Verfügung zu zahlreichen anderen Systemen.

Vorbereitung

Informieren Sie sich im **Kafka Quickstart-Tutorial** zum Konzept der **Kafka Topics**: https://kafka.apache.org/quickstart ab Schritt 3.

Beachten Sie die beiden Folien zur **Praktikum 4 Kafka Infrastruktur** im gleichnamigen pdf.

Machen Sie sich vertraut mit dem Schema und den Daten der **New Yorker Taxidaten des Jahres 2015**. Diese kommen in Aufgabe 2 für einen **Join zwischen Streaming- und Static-Data** zur Anwendung. Analysieren Sie die Daten hinsichtlich ihrer Qualität.

Hinweis: Die zur Verfügung gestellten Daten wurden nicht(!) bereinigt.

New Yorker Taxidaten (*Yellow Cab*): Schema und Hive-Tabelle
Das **Schema** der New Yorker Taxidaten aus dem Jahr 2015 finden Sie auf der letzten Seite dieses Aufgabenblattes.
Die **Daten** der New Yorker Taxifahrten des gesamten Jahres 2015 sind auf dem Cluster in der folgenden Tabelle abgelegt:

Tabelle im Parquet File Format – *column based*
yellow_tripdata – alle Taxidaten des Jahres 2015 von Januar bis einschließlich Dezember
Datenvolumen (im csv-Format): **22 GB**

Datenquelle: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

---

Durchführung des Praktikums

Loggen Sie sich auf Zeppelin ein und kopieren Sie das Notebook praktikum4/_istaccount.

Aufgabe 1 – Nutzung von Kafka als Producer und Consumer von Data Streams

Für diese Aufgabe wurden die folgenden beiden **Kafka-Topics** angelegt:

bda-gruppe1-topic für die Gruppe Do5y, also am 23.05.2019,
bda-gruppe2-topic für die Gruppe Mi6x, also am 29.05.2019, und
bda-gruppe3-topic für die Gruppe Mi6y, also am 05.06.2019.

In den beiden folgenden Sourcecode-Beispielen für *Aufgabe 1* wird immer das Topic der *Gruppe 1* verwendet – für *Gruppe 2* beziehungsweise *Gruppe 3* bitte entsprechend ändern!

Producer *(auf jeder Maschine Broker mit 9092) - Publish*

Auf jedem der 6 Nodes kann wie folgt ein Producer gestartet werden, der sich bei einem Kafka-Broker registriert:

```
/opt/kafka/bin/kafka-console-producer.sh --broker-list
<Node-IP>:9092 --topic bda-gruppe1-topic
```

Consumer - *Subscribe*

Als Consumer können die Daten des Producers über jeden der 6 Nodes als Stream empfangen werden:

```
/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server
<Node-IP>:9092 --topic bda-gruppe1-topic --from-beginning
```

---

a) Öffnen Sie zwei SSH-Sessions und starten Sie in einer der beiden Sessions einen Producer für das Kafka-Topic Ihrer Gruppe und in der anderen einen Consumer.
   1. Starten Sie den Consumer zunächst **ohne** die Option `--from-beginning`.
   2. Schreiben Sie dann über Ihre Producer einen beliebigen Text und beobachten Sie, was passiert.
   3. Starten Sie nun einen Consumer **mit** der Option `--from-beginning` und beobachten Sie erneut was passiert.

b) Lesen Sie nun in einer PySpark-Session in Ihrem Zeppelin-Notebook erneut aus Ihrem Kafka Topic aus a) als `Consumer ("subscribe")` den entsprechenden Datenstrom mit folgendem `format`()- und `option`()-Parametern:

```
format: "kafka"
option: "kafka.bootstrap.servers", "141.100.62.88:9092"
option: "subscribe", " bda-gruppe1-topic"
option: "startingOffsets", "earliest"
option: "kafkaConsumer.pollTimeoutMs", "8192"
```

Das Schema des DataFrame, der diesen Stream repräsentiert, können Sie sich anschauen mit *<myReadStream>*`.printSchema()`. Machen Sie sich die Bedeutung der einzelnen Spalten klar.

Binary codierte Spaltenwerte können Sie wie folgt in einen String casten, z.B. die Spalte `'value'`: *<myReadStream>=*

<myReadStream>`.withColumn('value', `*<myReadStream>.value.cast(*`'string'))`

Wenden Sie nun die **wordCount-Query** aus Praktikum 3 auf den Message-Datenstrom (`value`) Ihres Topics an.

WICHTIG: Stoppen Sie im Folgenden Ihre laufende Streaming Query immer, bevor Sie eine neue starten!

Ausarbeitung

Dokumentieren Sie das beobachtete Verhalten bei Aufgabe 1a) mit und ohne die Option `--from-beginning`.

Aufgabe 2 – Nutzung von Kafka als Datenquelle für Spark mit den NY Taxidaten des Monats *Dezember 2015* als *„real time“* Datenstrom

Ziel dieser Aufgabe ist es, Daten eines real time-Datenstroms zu aggregieren (Dezember 2015) und real time einen Join mit historischen, bereist aggregierten Daten zu generieren (Juli 2015). Die Unterschiede der aggregierten Werte sollen in Echtzeit visualisiert werden.

Führen sie alle Teilaufgaben in den vorgesehenen Paragraphen Ihres Zeppelin Notebooks aus.

Datenvorbereitung *Data at Rest*

a) Als historische Daten sollen die NY-Taxidaten des Monats **Juli 2015** auf Tagesebene aggregiert werden. Hierzu soll zunächst ein count(*) für die **Anzahl der Fahrten pro Tag** durchgeführt werden:

   Um Aggregationen auf Tagesebene durchzuführen und die Daten später auf Tagesbasis mit den Daten des Monats Dezember 2015 zu joinen, muss zunächst aus dem Timestamp `tpep_pickup_datetime` der Wert des Tages extrahiert und einer neuen (Join-)Spalte zugewiesen werden. Über diese neue Spalte können Sie alle gewünschten Aggregate pro Tag durchführen.

   Hinweis 1: Für die Extraktion des Tages bzw. des Monats als int-Werte aus einem Attribut vom Typ Timestamp können Sie die Funktionen `dayofmonth(·)` bzw. `month(·)` verwenden, Sie können aber auch mit dem `date`-Anteil des Timestamps arbeiten.

   Hinweis 2: Der 1. Juli 2015 ist ein Mittwoch, der 1. Dezember 2015 ein Dienstag. Da es für einen Vergleich von aggregierten Werten pro Tag sicherlich sinnvoll ist, die Wochentage zu berücksichtigen, sollten Sie die Wochentage der Julidatei für den Join um einen Tag „shiften“, also Join zwischen Mittwoch, 1. Juli und Mittwoch, 2. Dezember etc.

   → Speichern Sie die aggregierten Daten für den Monat Juli in einer Hive-Tabelle. Stellen Sie beim Namen Ihren ist-Account voran.

b) Lesen Sie die Daten der Hive-Tabelle aus a) in einen DataFrame ein und lassen Sie sich die Tabelleninhalte in Zeppelin ausgeben.

Datentransformation *Data in Motion*

Die Taxidaten des Monats *Dezember 2015* sind als Topic unter dem Topic-Namen *"yellow_tripdata_2015_12"* erreichbar.

→ Sie können zunächst die Dezemberdaten als Consumer in einer SSH-Session an der Konsole anschauen – vgl. Aufgabe 1a).

c) Abonnieren Sie den *"yellow_tripdata_2015_12"*-Topic in Analogie zu 1b) (`"subscribe"`). Ergänzen Sie die Option `.option("maxOffsetsPerTrigger",1000000)`, mit der Sie die maximale Anzahl an Messages per Batch definieren. Den Fortschritt der einzelnen Batches können Sie auf der 141.100.62.85:8080 beobachten.

   Definieren Sie alle benötigten Spalten des Schemas für die nachfolgende(n) Aggregation(en). Leiten Sie insbesondere ein neues Attribut ab, das den aktuellen Tag des Timestamps `tpep_pickup_datetime` enthält in Analogie zur Aggregations- und Join-Spalte aus Aufgabe 2a).

   Generieren Sie nun einen write-Stream, der die Daten über ein Sliding Window mit *tpep_pickup_datetime* als *time_column, window_size ”24 hours”* (ohne *sliding_interval*) gruppiert bzgl. des abgeleiteten Attributes mit dem Tag des Kalenderdatums.

Starten Sie die entsprechende Query und lassen Sie die Daten in einer Tabelle ausgeben. Stoppen Sie die Query anschließend wieder.

d) Lesen Sie erneut die Daten der Tabelle mit den aggregierten Juli-Daten in ein DataFrame ein und definieren Sie einen Join zwischen diesen Daten und dem erweiterten Schema der Streaming-Daten, um anschließend wie in 2c) auf diesen Daten das *groupBy* entsprechend der gewünschten Spalten über einem Sliding Window von 24 hours durchzuführen:

```
<decData>.join(<DF historische Julidaten>, <"join-Spalte">).groupBy(…).…
```

Generieren Sie in Analogie zu 2c) einen write-Stream, starten Sie die query und visualisieren Sie die aggregierten Daten aus den historischen Juli-Daten und den real time-Dezember-Daten in einer geeigneten Grafik im Vergleich.

Ausarbeitung

An welchen Tagen können Sie sehr starke Unterschiede in der Anzahl der Taxifahrten zwischen den Juli- und den Dezember-Daten erkennen, und wie lassen sich diese erklären?

e) Implementieren Sie eine weitere Aggregation Ihrer Wahl zum Vergleich der historischen Juli-Daten mit den real time Dezember-Daten, z.B. durchschnittliche Distanz pro Tag, durchschnittlicher Total-Amount pro Tag, etc. Beachten Sie hierbei, dass die Daten nicht bereinigt sind und führen Sie zuvor eine Analyse durch, um geeignete Filter für die verrauschten Daten zu verwenden.

Optional können Sie auch „untertägig“ mit geänderter Windowsize Analysen durchführen.

Ausarbeitung

Dokumentieren Sie, welche Datenwerte Sie im Hinblick auf Ihre Aggregation in den Juli-Daten als Rauschen interpretieren und wie Sie den Filter wählen, um verrauschte Daten von der Aggregation auszuschließen.

WICHTIG: Vergessen Sie nicht, am Ende des Praktikums Ihre laufende Streaming Query zu schließen.

Testat: Protokollieren Sie spätestens 1 Woche nach dem Praktikumstermin alle Anweisungen aus den Aufgaben 1-2 in Ihrem Notebook und geben Sie dieses der Gruppe „Dozent“ frei. Laden Sie weiterhin ein Dokument (pdf) in Moodle hoch, das die beiden genannten Ausarbeitungs-Tasks enthält.

New Yorker Taxidaten (*Yellow Cab*), 2015 – Schema und Bedeutung der Merkmale

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. **1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.**
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
trip_distance	The elapsed trip distance in miles reported by the taxi-meter.
pickup_longitude	Longitude where the meter was engaged.
pickup_latitude	Latitude where the meter was engaged.
RateCodeID	The final rate code in effect at the end of the trip. **1= Standard rate** **2=JFK** **3=Newark** **4=Nassau or Westchester** **5=Negotiated fare** **6=Group ride**
store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server. **Y= store and forward trip** **N= not a store and forward trip**
dropoff_longitude	Longitude where the meter was disengaged.
dropoff_ latitude	Latitude where the meter was disengaged.
payment_type	A numeric code signifying how the passenger paid for the trip. **1= Credit card** **2= Cash** **3= No charge** **4= Dispute** **5= Unknown** **6= Voided trip**
fare_amount	The time-and-distance fare calculated by the meter.
extra	Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
mta_tax	$0.50 MTA tax that is automatically triggered based on the metered rate in use.
tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
tolls_amount	Total amount of all tolls paid in trip.
improvement_surcharge	$0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
total_amount	The total amount charged to passengers. Does not include cash tips.