Big Data Technologies

Agenda

- Spark SQL
- Streaming -- Introduction
- Spark Streaming
- Databricks Cloud
- Kafka -- Introduction

Spark SQL Setup (on Linux) with Derby Metastore

- Build Spark single-node cluster.
    - Download spark and extract it.
    - In ~/.bashrc, set SPARK_HOME and PATH.
    - Setup single-node cluster settings spark-defaults.conf and spark-env.sh
        - spark-defaults.conf
            - spark.master spark://localhost:7077
            - spark.sql.warehouse.dir file:///home/nilesh/spark-warehouse
- Copy hive-site.xml in $SPARK_HOME/conf.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:derby:;databaseName=/home/nilesh/metastore_db;create=true</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>org.apache.derby.jdbc.EmbeddedDriver</value>
```

```
    </property>
    <property>
        <name>javax.jdo.PersistenceManagerFactoryClass</name>
        <value>org.datanucleus.api.jdo.JDOPersistenceManagerFactory</value>
    </property>
    <property>
        <name>spark.sql.warehouse.dir</name>
        <value>file:///home/nilesh/spark-warehouse</value>
    </property>
</configuration>
```

- Start Master and Workers.
  - terminal> start-master.sh
  - terminal> start-workers.sh
- Start ThriftServer.
  - terminal> start-thriftserver.sh
  - terminal> netstat -tln | grep "10000"
  - Internally creates spark-warehouse directory and spark metastore_db (in Hive metastore format).
- Start beeline.
  - terminal> beeline -u jdbc:hive2://localhost:10000 -n $USER

Spark SQL Setup (on Linux) with MySQL Metastore

- Build Spark single-node cluster.
  - Download spark and extract it.
  - In ~/.bashrc, set SPARK_HOME and PATH.
  - Setup single-node cluster settings spark-defaults.conf and spark-env.sh
    - spark-defaults.conf
      - spark.master spark://localhost:7077
      - spark.sql.warehouse.dir file:///home/nilesh/spark-warehouse
- Copy hive-site.xml in $SPARK_HOME/conf.

```
```

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:mysql://localhost:3306/metastore_db</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.cj.jdbc.Driver</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>hive</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>hive</value>
    </property>
    <property>
        <name>javax.jdo.PersistenceManagerFactoryClass</name>
        <value>org.datanucleus.api.jdo.JDOPersistenceManagerFactory</value>
    </property>
    <property>
        <name>spark.sql.warehouse.dir</name>
        <value>file:///home/nilesh/spark-warehouse</value>
    </property>
</configuration>
```

- Create metastore schema on MySQL.
    - Download on your machine. https://raw.githubusercontent.com/apache/hive/master/standalone-metastore/metastore-server/src/main/sql/mysql/hive-schema-3.1.0.mysql.sql
    - terminal> sudo mysql
    - mysql> CREATE DATABASE metastore_db;

    - mysql> CREATE USER 'hive'@'%' IDENTIFIED BY 'hive';
    - mysql> GRANT ALL ON metastore_db.* TO 'hive'@'%';
    - mysql> FLUSH PRIVILEGES;
    - mysql> USE metastore_db;
    - mysql> SOURCE /path/of/hive-schema-3.1.0.mysql.sql
    - mysql> EXIT;
- Copy mysql driver jar into $SPARK_HOME/jars.
- Start Master and Workers.
    - terminal> start-master.sh
    - terminal> start-workers.sh
- Start ThriftServer.
    - terminal> start-thriftserver.sh
    - terminal> netstat -tln | grep "10000"
    - Internally creates spark-warehouse directory and spark metastore_db (in Hive metastore format).
- Start beeline.
    - terminal> beeline -u jdbc:hive2://localhost:10000 -n $USER

```
SHOW DATABASES;

CREATE DATABASE test;

SHOW DATABASES;

USE test;

SHOW TABLES;

-- create table using hive serde
CREATE TABLE hbooks(id INT, name STRING, author STRING, subject STRING, price DOUBLE)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

LOAD DATA LOCAL
```

```
INPATH 'file:///home/nilesh/sep22/dbda/bigdata/data/books.csv'
INTO TABLE hbooks;

SELECT * FROM hbooks;

SELECT subject, SUM(price) FROM hbooks
GROUP BY subject;

EXPLAIN
SELECT subject, SUM(price) FROM hbooks
GROUP BY subject;

SET spark.sql.shuffle.partitions=2;

SELECT subject, SUM(price) FROM hbooks
GROUP BY subject;

EXPLAIN
SELECT subject, SUM(price) FROM hbooks
GROUP BY subject;
```

```
-- create table using spark data formats
CREATE TABLE sbooks(id INT, name STRING, author STRING, subject STRING, price DOUBLE)
USING csv;

DESCRIBE sbooks;

INSERT INTO sbooks
SELECT * FROM hbooks;

SELECT * FROM sbooks;

SELECT subject, SUM(price) FROM sbooks
GROUP BY subject;
```

```
EXPLAIN
SELECT subject, SUM(price) FROM sbooks
GROUP BY subject;

DROP TABLE sbooks;
```

```
-- create external table using spark data formats
CREATE EXTERNAL TABLE movies(id INT, title STRING, genres STRING)
USING csv
OPTIONS(
header true,
path 'file:///tmp/movies'
);

SELECT * FROM movies
LIMIT 20;

CREATE VIEW mg AS
SELECT id, title, EXPLODE(SPLIT(genres,'[|]')) genre FROM movies;

DESCRIBE mg;

SELECT * FROM mg
LIMIT 20;

SHOW TABLES;

SHOW VIEWS;

SELECT genre, COUNT(id) cnt FROM mg
GROUP BY genre;

DROP VIEW mg;
```

```
SHOW TABLES;
```

Spark DStream

- Need StreamingContext.
- DStream programming
    - Stream source
    - String processing/operations
    - Stream sink
- This process is repeated periodically (as per batch duration).

Spark Structured Streaming

- Spark Dataframe is wrapper on Spark RDD.
- Spark Structured Streaming is NOT wrapper on Spark DStreams.
- Spark Structured Streaming a new framework developed from scratch.
- Spark Structured Streaming data is considered as infinite Dataframe i.e. new data gets appended at the dataframe. Hence most of dataframe operations are applicable on structured streaming.

Output Modes

- append --- only if no aggregations in the proceswsing.
    - result is processed and appended to the sink.
    - mainly used for data cleaning/filtering.
    - e.g. twitter tweets -- sentiment analysis i.e. tweet --> score.
- complete -- works with aggregation operations
    - the full aggregate result is displayed each time.
    - e.g. live poll with 4 options -- live counting -- final result is not too big.
- update -- works with or without aggregation operations
    - only updated/modified aggregate result is displayed each time.
    - e.g. monitor cabs (GPS) movement -- only modified results to be displayed.

Assignment

- Implement trending tweets program in Databricks cloud.