Streaming de Dados em Tempo Real: Aula 4

Prof. Felipe Timbó

UNI7

Ementa (dia 4)

- Cont. Spark Dataframes
- Resolução de problemas com Spark Streaming

Antes de começar...

Iniciar o Spark:

- `start-master.sh`
- `tart-worker.sh spark://posgrad-VirtualBox:7077`

Mais Exemplos utilizando Dataframes

Em outro terminal, baixar o conjunto de dados de voos nos anos de 2014 e 2015:

- `wget https://raw.githubusercontent.com/felipetimbo/streaming-data-course/main/voos-2014.csv`
- `wget https://raw.githubusercontent.com/felipetimbo/streaming-data-course/main/voos-2015.csv`

Lendo os Dados

Criar um dataframe de pessoas no terminal Pyspark:

- `voos14 = spark.read.csv("file:///home/posgrad/voos-2014.csv")`
- `voos15 = spark.read.csv("file:///home/posgrad/voos-2015.csv")`
- `voos14.show()`
- `voos15.show()`

Ler os dados incluindo o cabeçalho e esquema:

- `voos14 = spark.read.option("inferSchema", "true").option("header",True).csv("file:///home/posgrad/voos-2014.csv")`
- `voos15 = spark.read.option("inferSchema", "true").option("header",True).csv("file:///home/posgrad/voos-2015.csv")`

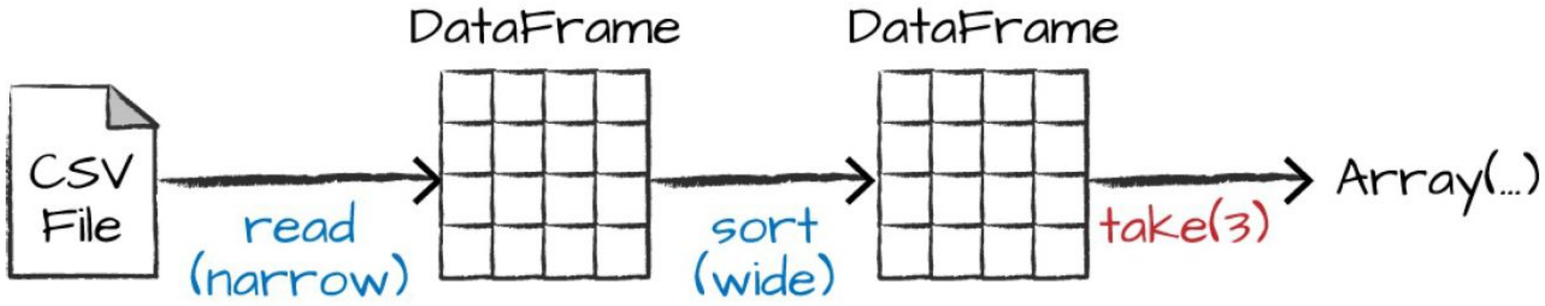
Obtendo Dataframes

Obter os 3 primeiros registros:

➢ `voos15.take(3)`

Ordenar e obter os 3 primeiros registros:

➢ `voos15.sort("count").take(2)`



Obtendo Dataframes

Obter o maior número de voos entre dois países

- `from pyspark.sql.functions import max`
- `voos15.select(max("count")).take(1)`

Obter o voo mais frequente em 2015

- `from pyspark.sql.functions import desc`
- `voos15.sort(desc("count")).take(1)`

Obter os 5 voos mais frequentes em 2015

- `voos15.sort(desc("count")).limit(5).show()`

Renomear uma coluna

- `voos15.sort(desc("count")).withColumnRenamed("DEST_COUNTRY_NAME", "destino").limit(5).show()`

Obtendo Dataframes

Obter a quantidade total de voos em 2015

- `from pyspark.sql.functions import sum`
- `voos15.select(sum("count")).take(1)`

Obter a média de voos diários em 2015

- `voos15.select(sum("count")/365).take(1)`

Outras operações

União

- `voos_concat = voos14.union(voos15)`

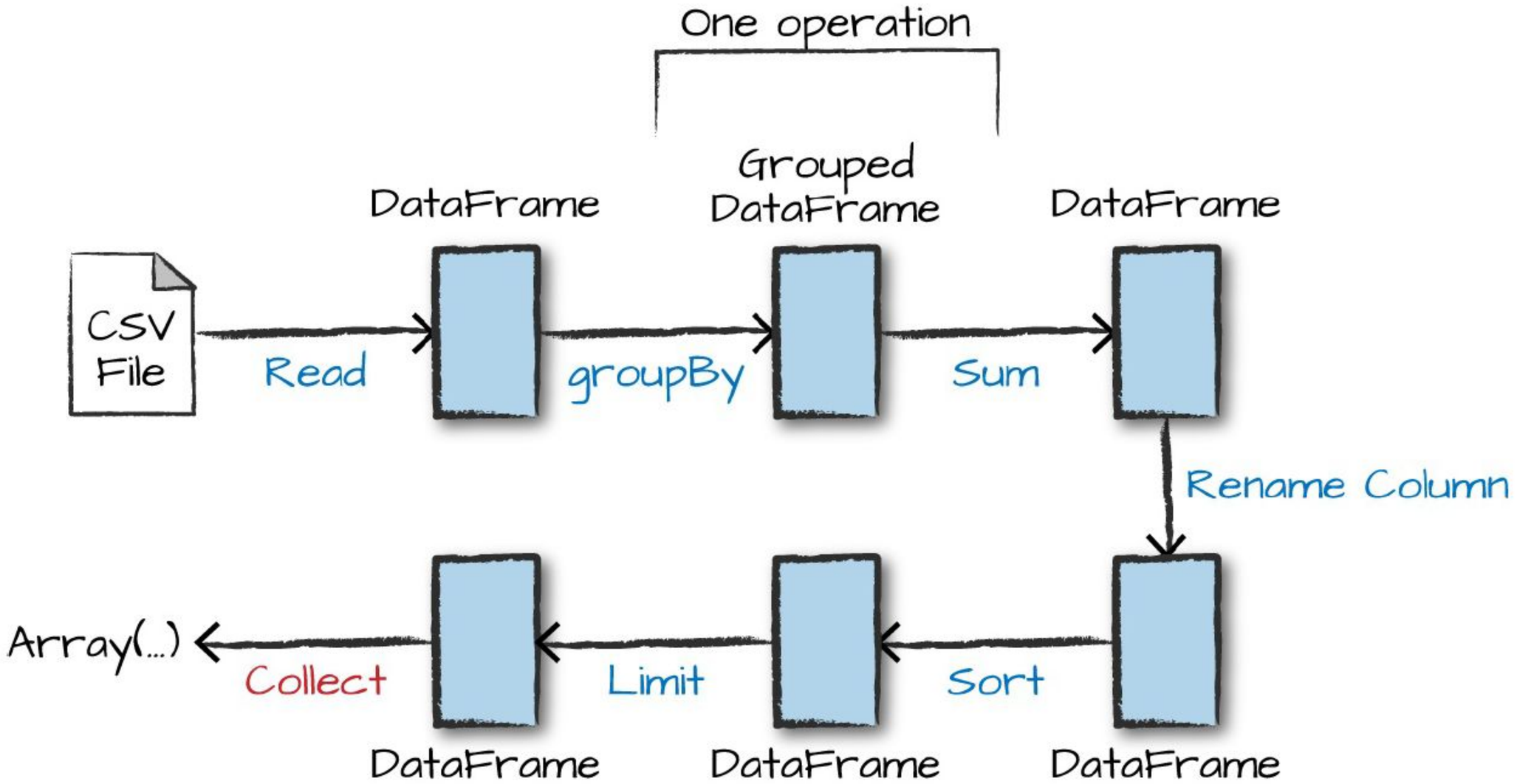
Interseção

- `voos_int = voos14.intersect(voos15)`

Exercício

Quais os 5 países destino que mais receberam voos em 2015?

Pipeline das operações



Exercício

Quais os 5 países destino que mais receberam voos em 2015 utilizando spark SQL?

Em Script Python

Criar um arquivo Python no VSCode chamado voos.py

Escrever as seguintes linhas:

```
from pyspark.sql import SparkSession

if __name__ == "__main__":
    spark = SparkSession.builder.appName("App").getOrCreate()
    voos15 = spark.read.csv("file:///home/posgrad/voos-2015.csv")
    voos15.show()
    spark.stop()
```

Para rodar o script:

```
/opt/spark/bin/spark-submit voos.py
```

Mais Exercícios

1. Qual a diferença no número total de voos entre 2014 e 2015?
2. Qual o voo mais frequente em 2014?
3. Qual a quantidade de voos total em 2014 entre os destinos "Bolivia" e "United States" ?
4. Qual a quantidade de voos total em 2014 e em 2015 entre os destinos "Germany" e "United States" ?
5. Qual a média de voos dentro dos EUA por dia considerando os anos de 2014 e 2015?

Mais Exercícios

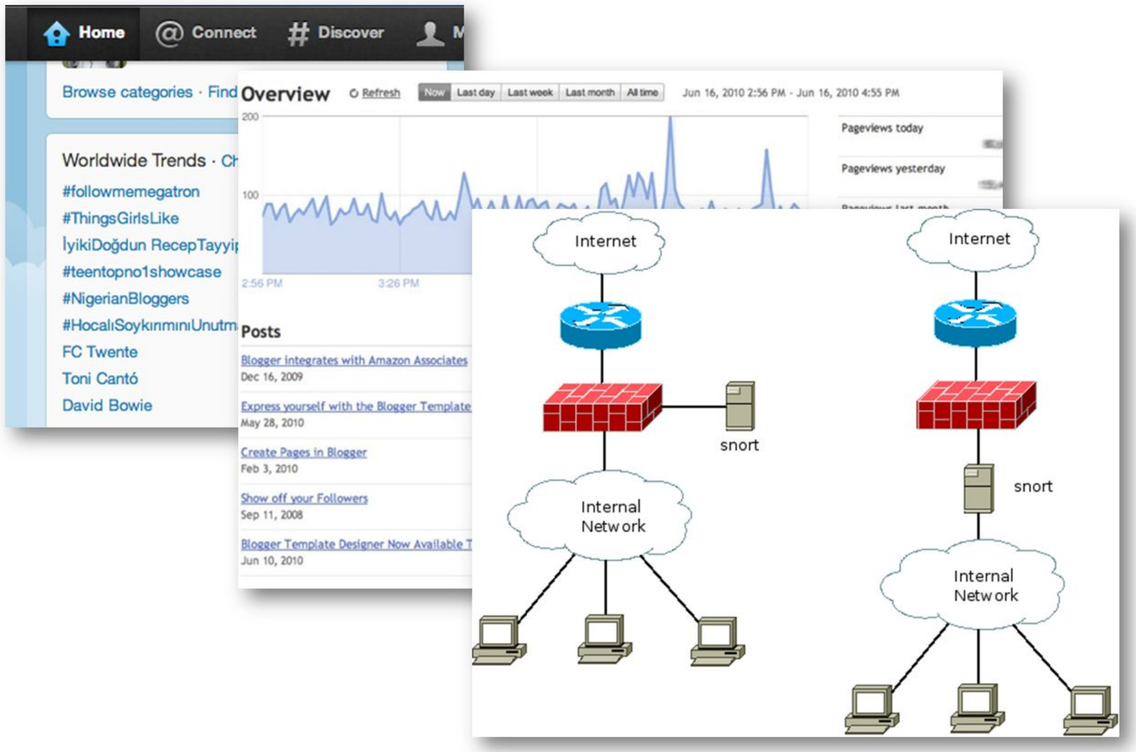
Escreva um script Python para:

1. Saber se existe algum registro de voos entre Brasil e EUA e em caso positivo, qual a quantidade no ano de 2014?
2. Saber se existe algum registro de voos que não parte ou não chega nos EUA. Em caso positivo, quais esses voos?
3. Sumarizar os dados de 2014 e 2015 em um só Dataframe (Certifique-se de que não contenha registros repetidos).

Spark Streaming

Motivação

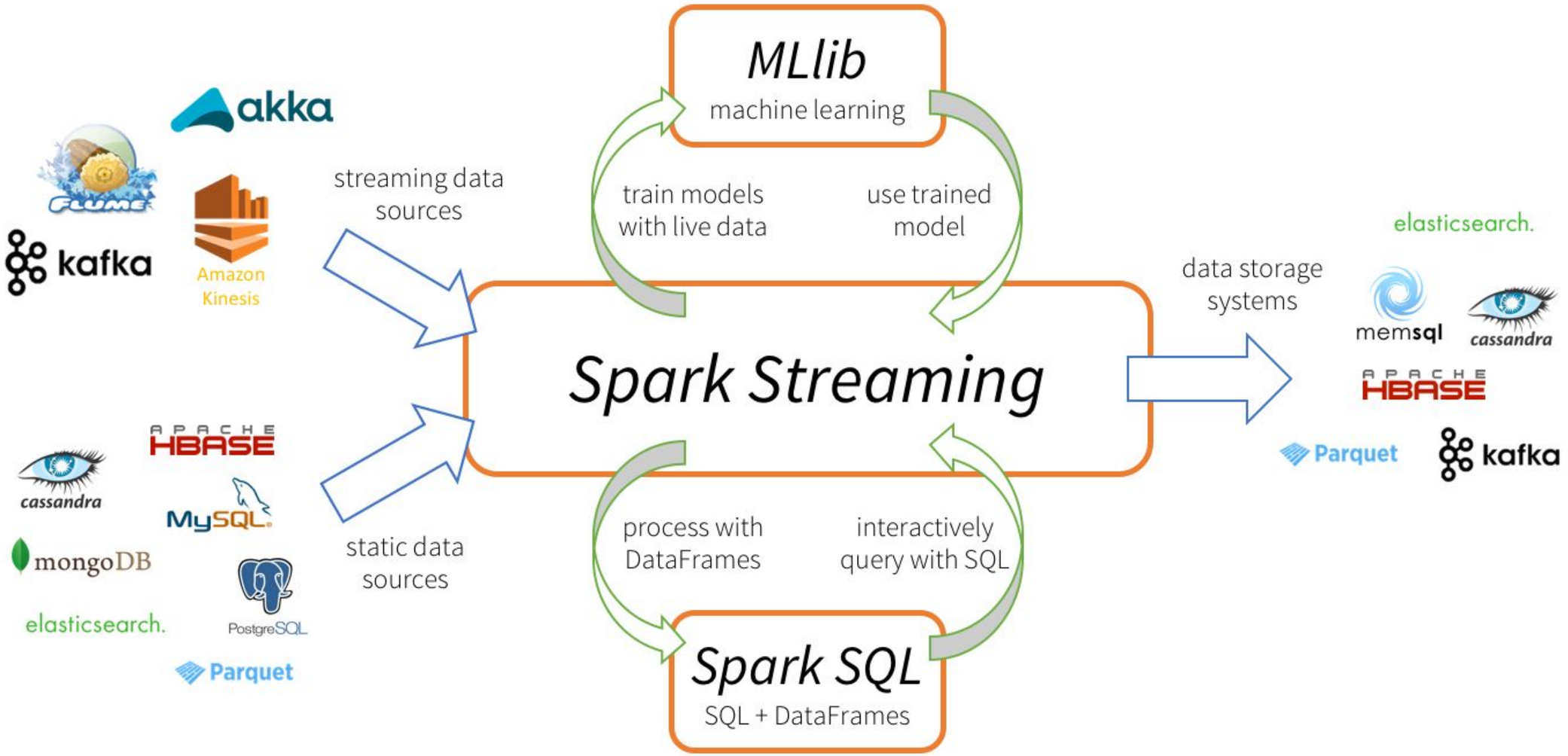
- Processamentos de streaming de dados em tempo real.
  - Tendências em redes sociais
  - Estatísticas web
  - Detecção de intrusão de sistemas
  - etc.



Spark Streaming

- Extensão da API do Spark para processar dados em tempo real
- Características:
  - Recuperação rápida de falhas e atrasos
  - Melhor balanceamento de carga e uso de recursos
  - Combinação de streaming de dados com conjuntos de dados estáticos e consultas interativas
  - Integração nativa com bibliotecas de processamento avançado (SQL, aprendizado de máquina, Grafos, etc.)

Spark Streaming

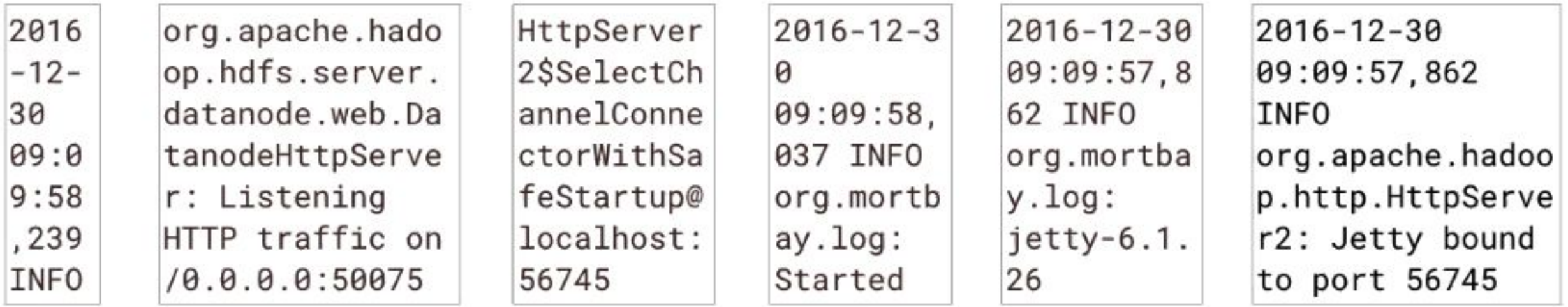


Spark Streaming APIs

- **DStream API**
  - Baseada em RDDs
  - Não possui Engine SQL
  - Suporte a semântica de janelas temporais
- **Structured Streaming API**
  - Baseada em Dataframes
  - Possui Engine SQL
  - Suporte a semântica de janelas e eventos temporais

DStream API

DStream - Discretized Stream

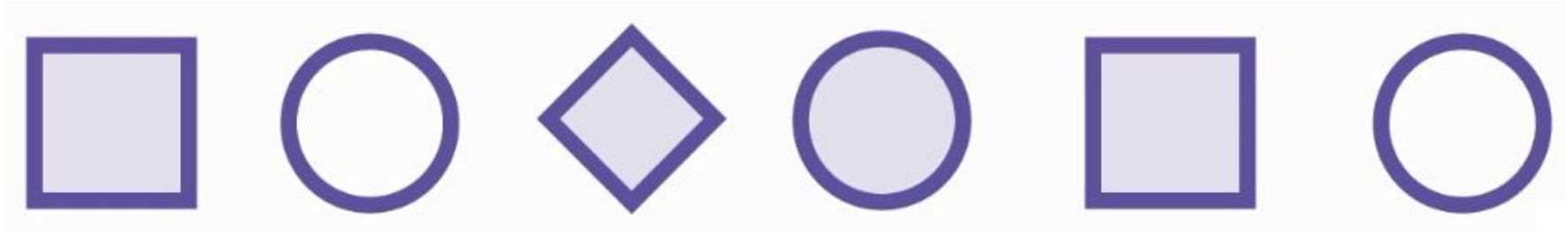


Cada mensagem é uma **entidade** no streaming.

Spark trabalha com streaming de dados usando a mesma abstração do RDD.

DStream - Discretized Stream

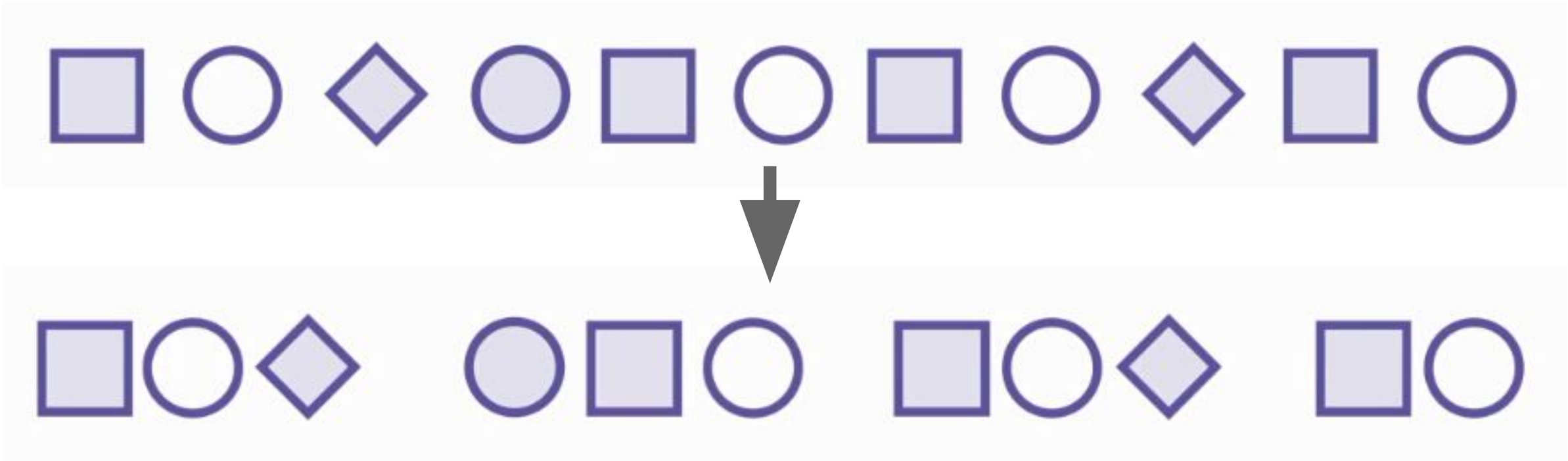
2016-12-30 09:09:58,239 INFO	org.apache.hadoop.hdfs.server.datanode.web.DatanodeHttpServer: Listening HTTP traffic on /0.0.0.0:50075	HttpServer2$SelectChannelConnectorWithSafeStartup@localhost:56745	2016-12-30 09:09:58,037 INFO org.mortbay.log: Started	2016-12-30 09:09:57,862 INFO org.mortbay.log: jetty-6.1.26	2016-12-30 09:09:57,862 INFO org.apache.hadoop.http.HttpServer2: Jetty bound to port 56745



Stream “discretizado” = DStream = Sequência de RDDs

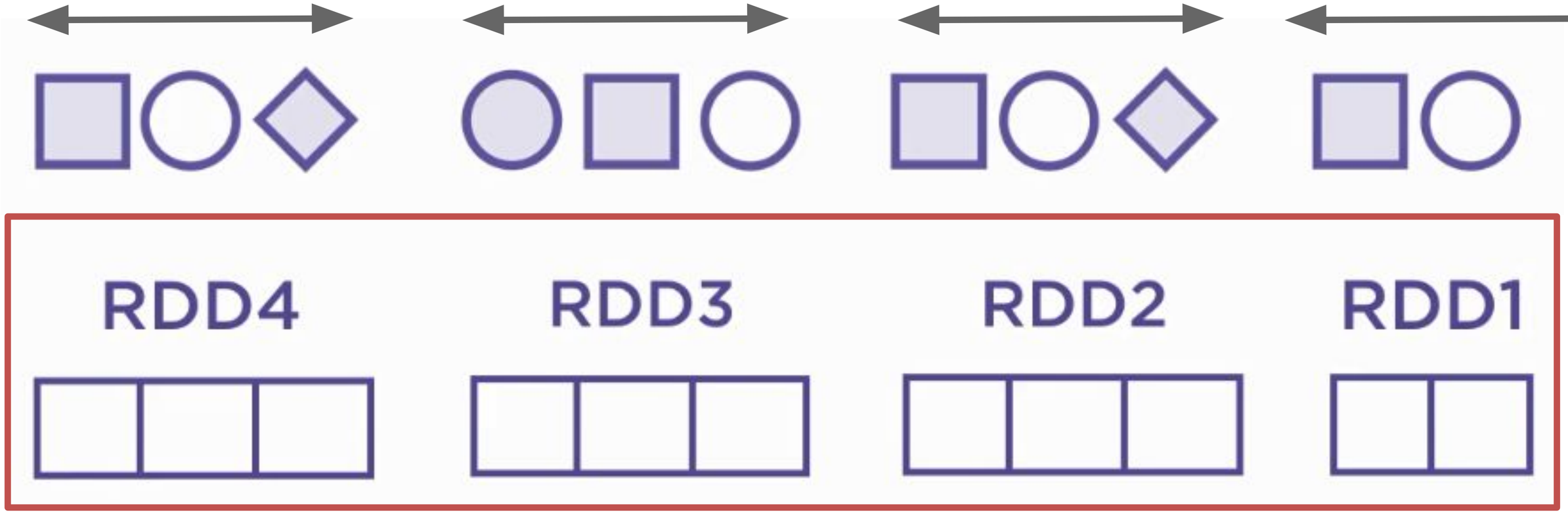
DStream - Discretized Stream

- Entidades são agrupadas em **batches.** Cada batch é um **RDD**.



DStream - Discretized Stream

- **Batches** são formados com base em um intervalo de tempo.



Antes de começar...

Criar uma nova pasta e abrir no VSCode

- `mkdir dstream`
- `cd dstream`
- `code .`

Filtragem

```
2016-12-30 09:09:57,862 INFO
org.apache.hadoop.http.HttpServer2: Jetty bound to port
56745
2016-12-30 09:09:57,862 INFO org.mortbay.log: jetty-6.1.26
2016-12-30 09:09:58,037 INFO org.mortbay.log: Started
HttpServer2$SelectChannelConnectorWithSafeStartup@localhost:
56745
2016-12-30 09:09:58,124 INFO
org.apache.hadoop.hdfs.server.datanode.web.DatanodeHttpServe
r: Listening HTTP traffic on /0.0.0.0:50075
2016-12-30 09:09:58,239 INFO
```

Nesse log, aparece a string **WARN**?

Filtragem

Terminal 1

1. Baixar e examinar o código no VS Code
   - ➢ `wget https://raw.githubusercontent.com/felipetimbo/streaming-data-course/main/sparkStreaming.py`
2. Acessar e editar propriedades de log do arquivo log4j
   - ➢ `cd /opt/spark/conf`
   - ➢ `cp log4j.properties.template log4j.properties`
   - ➢ `gedit log4j.properties`
3. Alterar a linha:

`log4j.rootCategory=INFO -> log4j.rootCategory=ERROR`

Filtragem

Terminal 2

4. Rodar o comando netcat

Netcat é uma ferramenta versátil para testes de rede o qual permite ler e escrever dados através das conexões, usando o protocolo TCP/IP.

- `nc -lk 9999`

Filtragem

VS Code

5. Executar sparkStreaming.py

➢ `/opt/spark/bin/spark-submit sparkStreaming.py localhost 9999`

Exercício

1. Alterar o intervalo de tempo para 10 segundos
2. Alterar a palavra buscada para "ERROR"
3. Filtrar apenas por palavras que começam com a letra A (maiúsculo).
4. Filtrar apenas por palavras que terminam com um número qualquer. ex. qwe4, des11, cvb0

Sumário de Contagens

Acumular a contagem de palavras ao longo do tempo, ou seja, à medida que os dados de streaming vão chegando.

`updateStateByKey`

Sumário de Contagens

1. Baixar e examinar o código

Terminal 1

```
wget
https://raw.githubusercontent.com/felipetimbo/strea
ming-data-course/main/sparkStreaming2.py
```

2. Rodar o comando netcat

Terminal 2

```
nc -lk 9999
```

3. Executar sparkStreaming2.py

Terminal 1

```
/opt/spark/bin/spark-submit sparkStreaming2.py
localhost 9999
```

Exercício

- Sumarizar não mais a palavra e quantas vezes ela aparece, mas sim, a quantidade de palavras que existem com um determinado número de letras.

  Exemplo:

  Entrada: pao casa show cao loja disco

  Saída esperada:

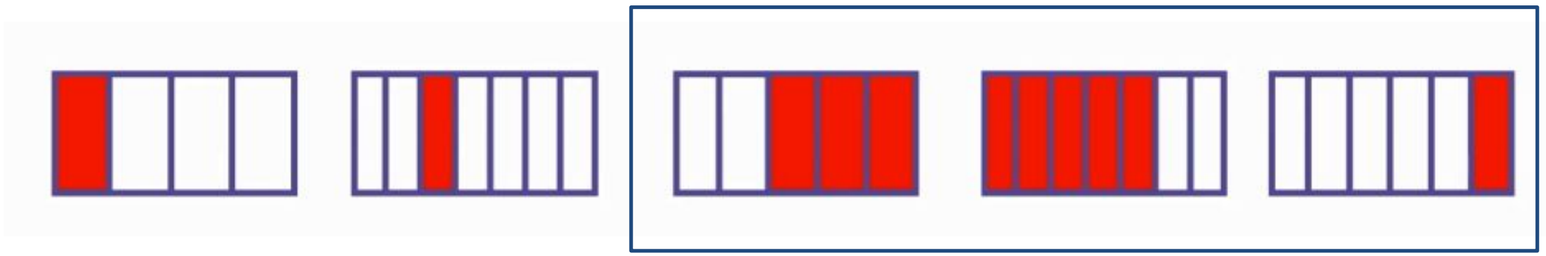
  3: 2 // (2 palavras com 3 letras)

  4: 3 // (3 palavras com 4 letras)

  5: 1 // (1 palavra com 5 letras)

Janelas

```
2016-12-30 09:09:57,862 INFO
org.apache.hadoop.http.HttpServer2: Jetty bound to port
56745
2016-12-30 09:09:57,862 INFO org.mortbay.log: jetty-6.1.26
2016-12-30 09:09:58,037 INFO org.mortbay.log: Started
HttpServer2$SelectChannelConnectorWithSafeStartup@localhost:
56745
2016-12-30 09:09:58,124 INFO
org.apache.hadoop.hdfs.server.datanode.web.DatanodeHttpServe
r: Listening HTTP traffic on /0.0.0.0:50075
2016-12-30 09:09:58,239 INFO
```



E se eu quiser o count de strings apenas dos últimos 10 minutos?

Janelas

1. Baixar e examinar o código

Terminal 1

```
wget
https://raw.githubusercontent.com/felipetimbo/strea
ming-data-course/main/sparkStreaming3.py
```

2. Rodar o comando netcat

Terminal 2

```
nc -lk 9999
```

3. Executar sparkStreaming3.py

Terminal 1

```
/opt/spark/bin/spark-submit sparkStreaming3.py
localhost 9999
```

Exercício

1. Alterar o tamanho da janela para 15 segundos
   e o intervalo de print dos dados para 3seg.
2. Contar na janela apenas palavras que possuem "a" (em qualquer posição)
3. Utilizando janela, contar o número de vezes
   que a palavra ERROR aparece nos últimos 20seg.

reduceByWindow

1. Baixar e examinar o código

Terminal 1

```
wget
https://raw.githubusercontent.com/felipetimbo/strea
ming-data-course/main/sparkStreaming4.py
```

2. Rodar o comando netcat

Terminal 2

```
nc -lk 9999
```

3. Executar sparkStreaming4.py

Terminal 1

```
/opt/spark/bin/spark-submit sparkStreaming4.py
localhost 9999
```

Exercício

- Utilizando reduceByWindow, calcular a média dos números que chegam em uma janela de 15seg.
- Utilizando reduceByWindow, calcular a soma dos caracteres de uma frase inserida via netcat, de forma cumulativa (ao invés da soma de números), no intervalo de 20 segundos

reduceByKeyAndWindow

1. Baixar e examinar o código

Terminal 1

```
wget
https://raw.githubusercontent.com/felipetimbo/strea
ming-data-course/main/sparkStreaming5.py
```

2. Rodar o comando netcat

Terminal 2

```
nc -lk 9999
```

3. Executar sparkStreaming5.py

Terminal 1

```
/opt/spark/bin/spark-submit sparkStreaming5.py
localhost 9999
```

Exercício

- Conte os registros das palavras WARN, INFO e ERROR, provenientes de streaming (netcat) acumulando-os nos últimos 20 segundos.

Exercício

Calcule a **soma** dos números que estão chegando via streaming (sem janela).

Structured Streaming API

Antes de começar...

Criar uma nova pasta e abrir no VSCode

- `cd ~`
- `mkdir structured`
- `cd structured`
- `code .`

Contagem de palavras via Struct. API

Criar um arquivo chamado wordCountStructured.py

Copiar código padrão abaixo:

```
from pyspark.sql import SparkSession

if __name__ == "__main__":

    spark = SparkSession \
        .builder \
        .appName("wordCountStructured") \
        .getOrCreate()
```

Contagem de palavras via Struct. API

Passo 1: Ler de uma fonte

```
lines_df = spark.readStream \
    .format("socket") \
    .option("host", "localhost") \
    .option("port", "9999") \
    .load()

lines_df.printSchema()  # pode comentar essa linha depois
```

Rodar e observar o esquema:

```
|-- value: string (nullable = true)
```

Contagem de palavras via Struct. API

Passo 2: Processar os dados

- 2.1: Importar funções sql do Spark

```
from pyspark.sql.functions import *
```

- 2.2: Split do valor por espaços em branco

```
words_df = lines_df.select(expr("explode(split(value,' ')) as word"))
```

*explode: transforma um array de palavras em linhas (como flatmap)
*expr: como explode e split são expressões, necessitamos usar 'expr'

- 2.3: Agrupar as palavras e contar

```
counts_df = words_df.groupBy("word").count()
```

Contagem de palavras via Struct. API

Passo 3: Escrever na saída (sink)

```
word_count_query = counts_df.writeStream \
    .format("console") \
    .outputMode("complete") \
    .start()

word_count_query.awaitTermination()
```

Produzir dados via netcat

➢ `nc -lk 9999`

Rodar o programa

➢ `/opt/spark/bin/spark-submit wordCountStructured.py`

Exercício

Utilizando streaming estruturado, contar quantas mensagens chegam contendo somente as strings '#bbb' e '#afazenda' e ordenar por quantidade de ocorrências.

Manipulando dados de atividade física

1. Baixar os dados de streaming **no terminal**:

   ➢ `wget www.lia.ufc.br/~timbo/streaming/activity-data.zip`

2. Descompactar a pasta

   ➢ `unzip activity-data.zip`

3. Criar novo arquivo no VSCode chamado activity.py

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("MyApp").getOrCreate()
```

Dados de streaming estruturados

Carregar os dados de streaming:

```
static = spark.read.json("activity-data/")
```

Mostrar o esquema:

```
dataSchema = static.schema
```

```
root
|-- Arrival_Time: long (nullable = true)
|-- Creation_Time: long (nullable = true)
|-- Device: string (nullable = true)
|-- Index: long (nullable = true)
|-- Model: string (nullable = true)
|-- User: string (nullable = true)
|-- _corrupt_record: string (nullable = true)
|-- gt: string (nullable = true)
|-- x: double (nullable = true)
|-- y: double (nullable = true)
|-- z: double (nullable = true)
```

Dados de streaming estruturados

Exemplos dos dados:

```
+-------------+-------------------+--------+-----+------+----+--------+-----+-----
| Arrival_Time| Creation_Time     | Device |Index| Model|User|_c...ord|. gt | x
|1424696634224|1424696632226236 85|nexus4_1| 62  |nexus4| a  |    null|stand|-0...
|1424696660715|1424696658723817 26|nexus4_1| 2342|nexus4| a  |    null|stand|-0...
+-------------+-------------------+--------+-----+------+----+--------+-----+-----
```

*gt: que atividade o usuário estava fazendo naquele instante

Dados de streaming estruturados

Lendo o streaming:

- ```
  streaming =
  spark.readStream.schema(dataSchema).option("maxFilesPer
  Trigger", 1).json("activity-data")
  ```

*maxFilesPerTrigger: controlar a velocidade com que o Spark irá ler todos os arquivos da pasta (não muito usual na prática).
  ```

Dados de streaming estruturados

Manipulando o streaming:

- `activityCounts = streaming.groupBy("gt").count()`

Evitando a criação de muitas partições:

- `spark.conf.set("spark.sql.shuffle.partitions", 5)`

Dados de streaming estruturados

Definir como será o output do dado:

```
activityQuery =
activityCounts.writeStream.queryName("activity_counts").format("mem
ory").outputMode("complete").start()

from time import sleep
for x in range(20):
    spark.sql("SELECT * FROM activity_counts").show()
    sleep(5)

activityQuery.awaitTermination()
```

*memory: irá guardar todo o dado em memória (por simplicidade, neste exemplo)
*complete: reescreve todas as chaves e suas contagens a cada trigger

Rodar o script

Seleções e Filtros

```
from pyspark.sql.functions import expr

simpleTransform = streaming.withColumn("stairs",
expr("gt like '%stairs%'"))\
.where("stairs")\
.where("gt is not null")\
.select("gt", "model", "arrival_time", "creation_time")\
.writeStream\
.queryName("simple_transform")\
.format("memory")\
.outputMode("append")\
.start()
```

*append: novos resultados são adicionados à resposta

Exercício

1. Filtrar apenas pelo gt = bike
2. Filtrar pelo gt = walk ou gt = stand
3. Mostrar apenas as colunas gt, index, model e user

Spark Streaming
& KAFKA

Integração com o KAFKA

1. Subir o KAFKA (na pasta kafka)
   - ➢ `bin/zookeeper-server-start.sh config/zookeeper.properties`
   - ➢ `bin/kafka-server-start.sh config/server.properties`

2. Criar um tópico chamado *wctopic*
   - ➢ `bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --topic wctopic`

3. Criar um produtor
   - ➢ `bin/kafka-console-producer.sh --broker-list localhost:9092 --topic wctopic`

Integração com o KAFKA

4. Baixar o arquivo sparkStreamingKafka

- `cd ~/structured`
- `wget https://raw.githubusercontent.com/felipetimbo/streaming-data-course/main/sparkStreamingKafka.py`

5. Rodar o script

- `spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.0 sparkStreamingKafka.py`

6. Enviar mensagens pelo broker

Dúvidas