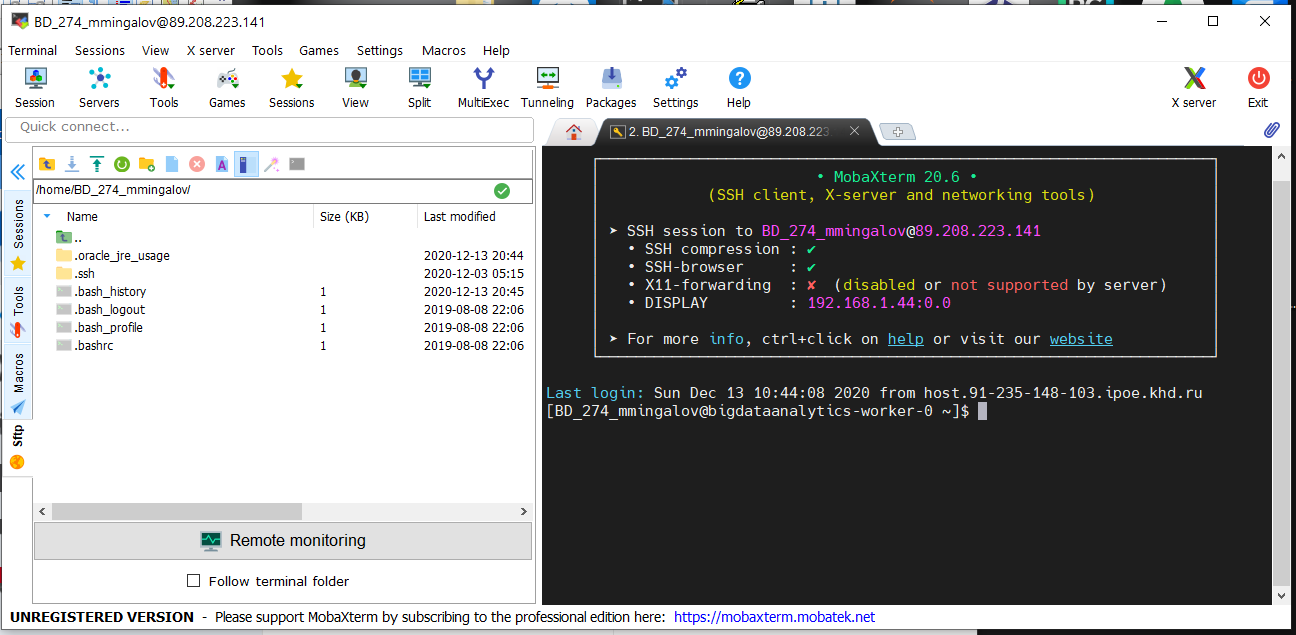
# **ЗАДАНИЕ 1: Повторить чтение файлов со своими файлами со своей схемой.**

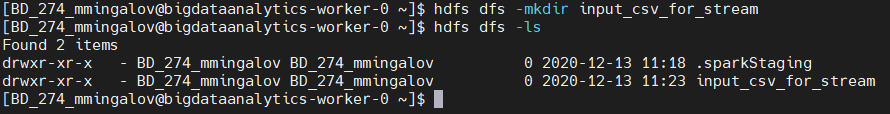
1.

Запускаем mobaexterm

Выбираем Putty-сессию на worker-2



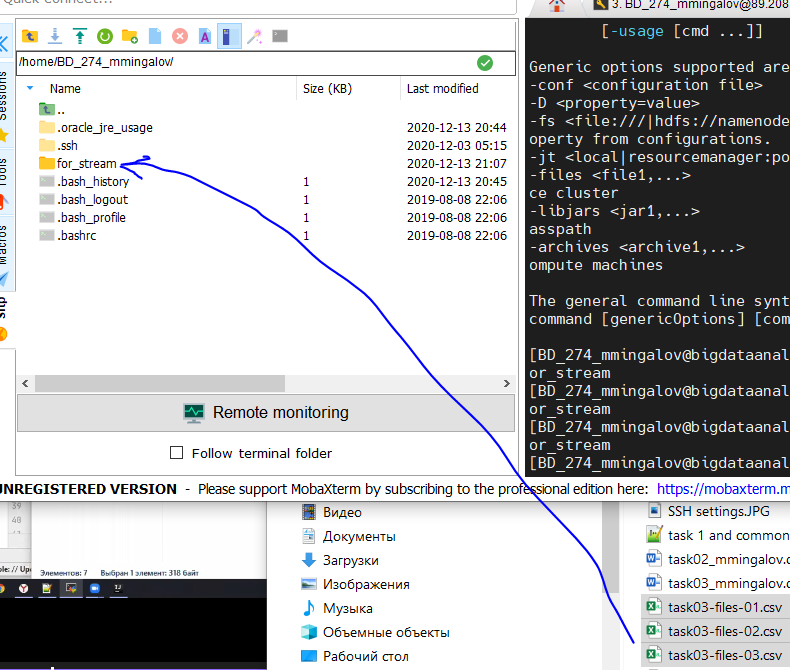
----  
создадим директорию в HDFS



-----

Создадим локальную директорию **for\_stream**/ командой **mkdir for\_stream**

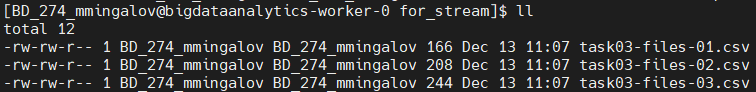
Подготовленные файлы методом drag&drop кидаем в локальную директорию for\_stream на worker-2



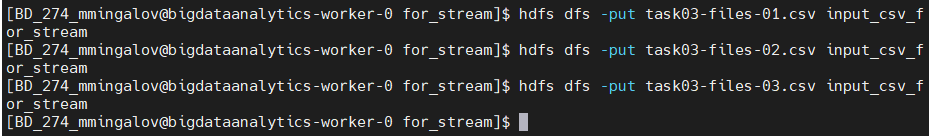
Переходим в эту директорию



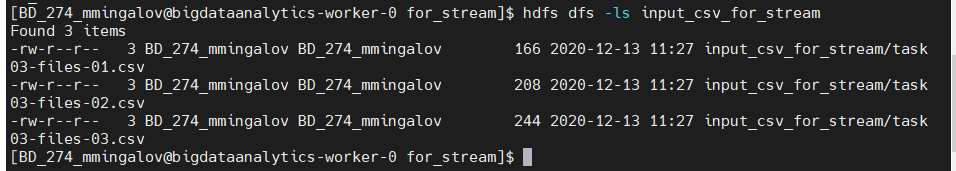
Смотрим файлы в ней такой командой:



----  
копируем файлы в HDFS директорию



Проверим



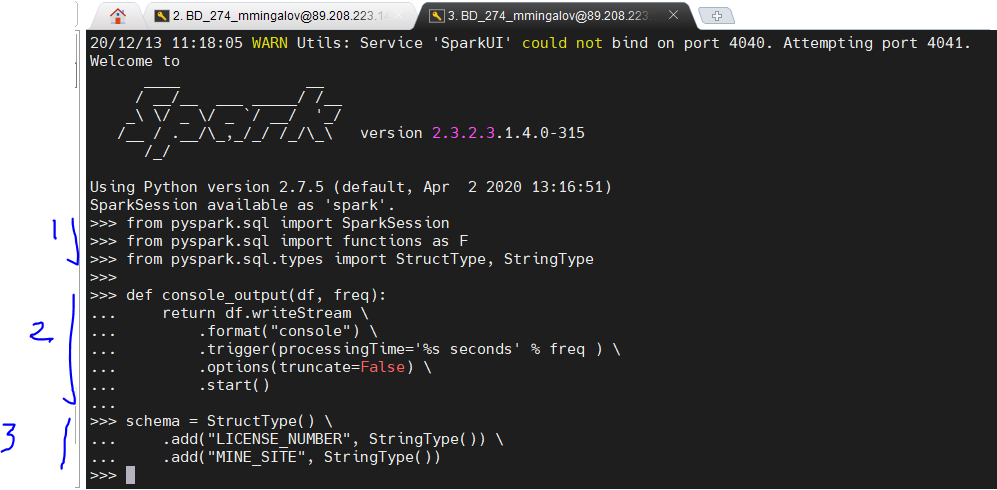
2.

ОТКРОЕМ ОТДЕЛЬНУЮ ВКЛАДКУ и выполним следующие сценарии

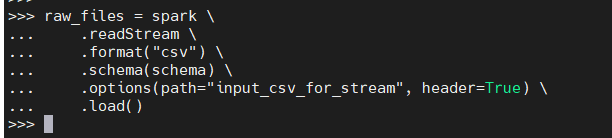
Для начала:

**pyspark**

далее выполняем:

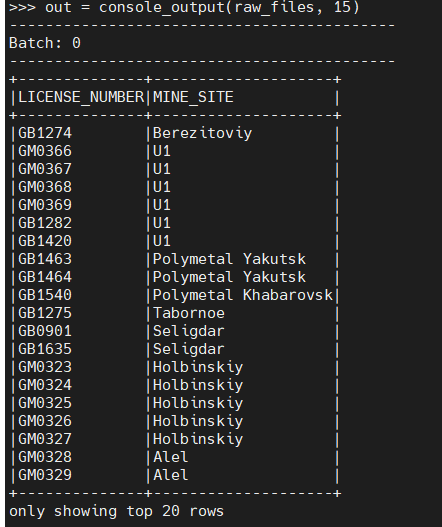


Далее



Запускаем процесс

**out = console\_output(raw\_files, 15)**



**Были обработаны все файлы за раз (отображено 20 строк только).**

**Важно:** если мы сейчас из директории input\_for\_csv все файлы удалим, а затем опять их туда поместим через -put, то наш stream ничего с ними не сделает. Он знает, что уже читал их

Stream постоянно проверяет что он уже сделал

# **ЗАДАНИЕ 2: Создать свой топик/топики, загрузить туда через консоль осмысленные данные с kaggle. Лучше в формате json. Много сообщений не нужно, достаточно штук 10-100.**

**Прочитать свой топик так же, как на уроке.**

**РЕШЕНИЕ 1**

Открываем новую сессию.

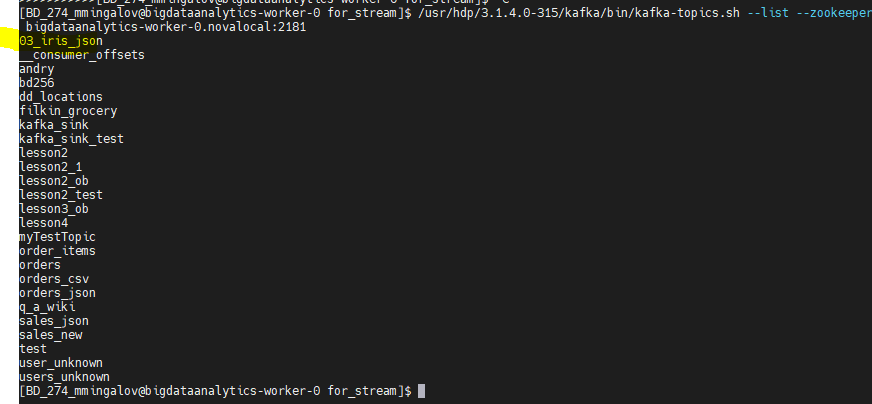
1.

Создаем топик **03\_iris\_json**

/usr/hdp/3.1.4.0-315/kafka/bin/kafka-topics.sh --create --topic 03\_iris\_json --zookeeper bigdataanalytics-worker-0.novalocal:2181 --partitions 3 --replication-factor 2 --config retention.ms=-1

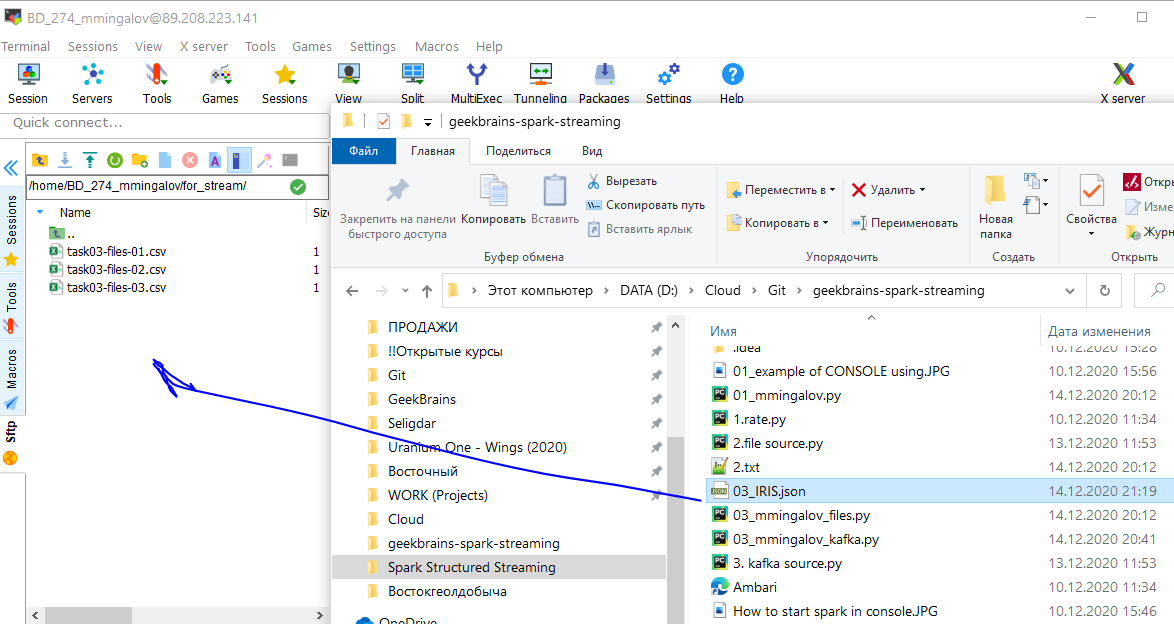
**Выведем список топиков:**

/usr/hdp/3.1.4.0-315/kafka/bin/kafka-topics.sh --list --zookeeper bigdataanalytics-worker-0.novalocal:2181

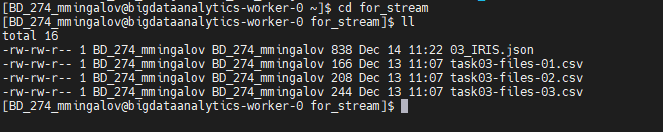


2.

Скачиваем json файл с ресурса Kaggle, помещаем его в нашу локальную директорию на хосте:



**Переходим в директорию, проверяем наличие файла:**



3.

**Читаем файл**

/usr/hdp/3.1.4.0-315/kafka/bin/kafka-console-producer.sh --broker-list bigdataanalytics-worker-0.novalocal:6667 --topic **03\_iris\_json < 03\_IRIS.json**

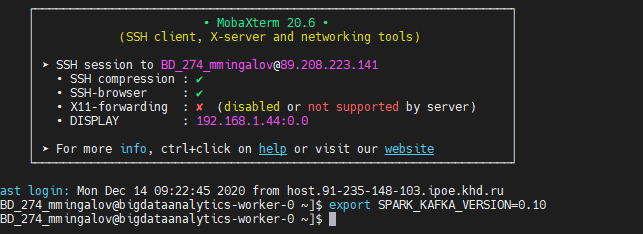


4.

Начинаем с того, что создаем новую сессию.

Даем команду:

*export SPARK\_KAFKA\_VERSION=0.10*



Далее запускаем **pyspark** с указанием нужного нам **packages.** Это нужно для того, чтобы Spark увидел новые функции, которые мы будем использовать

(использование packages означает, что он сам найдет его в интернете и скачает. Т.е. не нужно как в случае с jar предварительно качать их себе на локалку)

Итак, начинаем командой:

*pyspark --packages org.apache.spark:spark-sql-kafka-0-10\_2.11:2.3.2*

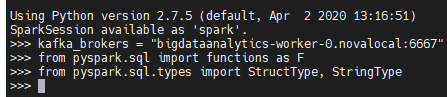
*Примечание: либо вот так, если глючит учебное окружение*

*pyspark --packages org.apache.spark:spark-sql-kafka-0-10\_2.11:2.3.2 -–master local[1]*

Добавляем kafka-брокер, импортируем типы объектов:

kafka\_brokers = **"bigdataanalytics-worker-0.novalocal:6667"**

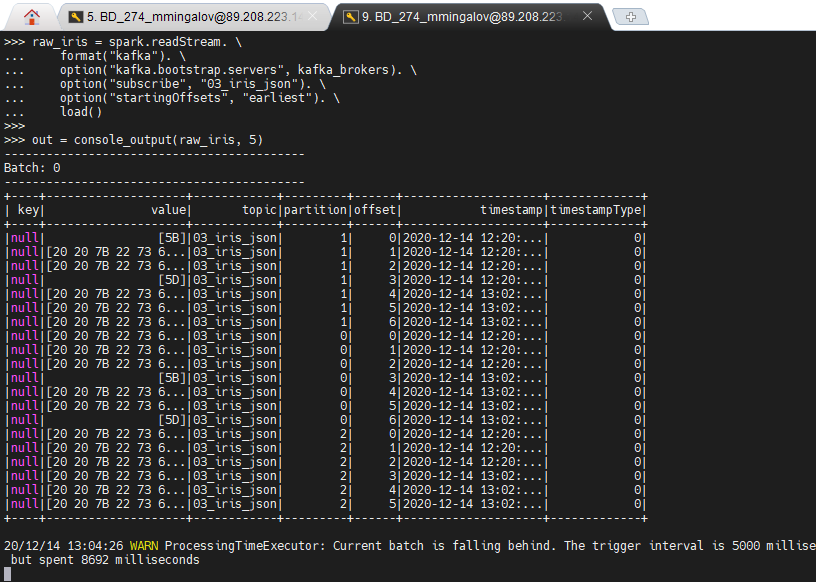
**from** pyspark.sql **import** functions **as** F  
**from** pyspark.sql.types **import** StructType, StringType



*#функция, чтобы выводить на консоль вместо show()***def** console\_output(df, freq):  
 **return** df.writeStream \  
 .format(**"console"**) \  
 .trigger(processingTime=**'%s seconds'** % freq ) \  
 .options(truncate=**True**) \  
 .start()  
  
raw\_iris = spark.readStream. \  
 format(**"kafka"**). \  
 option(**"kafka.bootstrap.servers"**, kafka\_brokers). \  
 option(**"subscribe"**, **"03\_iris\_json"**). \  
 option(**"startingOffsets"**, **"earliest"**). \  
 load()

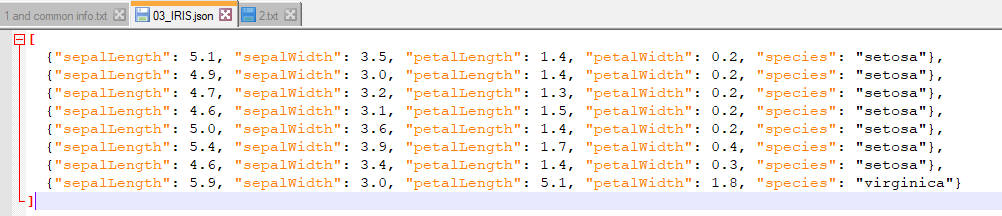
out = console\_output(raw\_iris, 5)

**ЧИТАЕМ STREAM:**



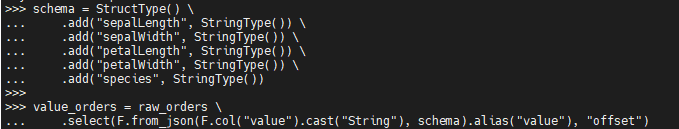
**РАЗБИРАЕМ VALUE:**

Помним, как выглядит на JSON:



Соответственно:

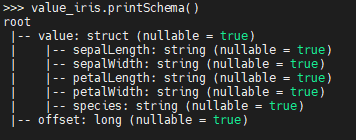
schema = StructType() \  
 .add(**"sepalLength"**, StringType()) \  
 .add(**"sepalWidth"**, StringType()) \  
 .add(**"petalLength"**, StringType()) \  
 .add(**"petalWidth"**, StringType()) \  
 .add(**"species"**, StringType())



далее:

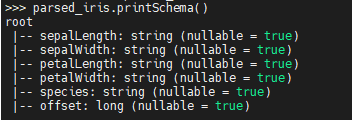
value\_iris = raw\_iris \  
 .select(F.from\_json(F.col(**"value"**).cast(**"String"**), schema).alias(**"value"**), **"offset"**)

value\_iris.printSchema()



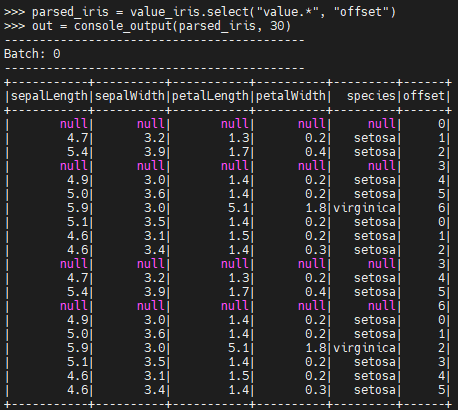
parsed\_iris = value\_iris.select(**"value.\*"**, **"offset"**)

parsed\_iris.printSchema()



**Кидаем в консоль:**

out = console\_output(parsed\_iris, 30)



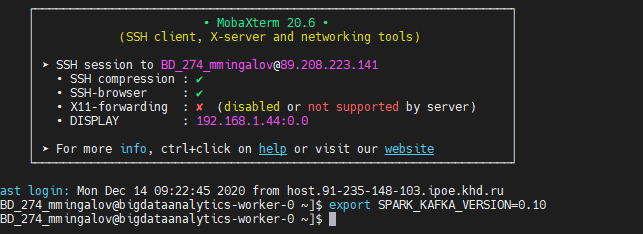
out.stop()

# **ДОП. ПРАКТИКА с KAFKA**

Начинаем с того, что создаем новую сессию.

Даем команду:

*export SPARK\_KAFKA\_VERSION=0.10*



Далее запускаем **pyspark** с указанием нужного нам **packages.** Это нужно для того, чтобы Spark увидел новые функции, которые мы будем использовать

(использование packages означает, что он сам найдет его в интернете и скачает. Т.е. не нужно как в случае с jar предварительно качать их себе на локалку)

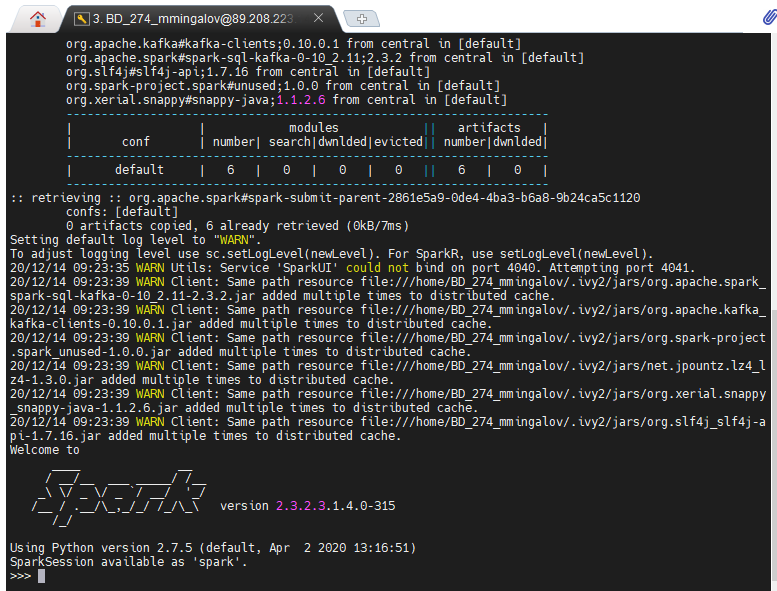
Итак, начинаем командой:

*pyspark --packages org.apache.spark:spark-sql-kafka-0-10\_2.11:2.3.2*

*Примечание: либо вот так, если глючит учебное окружение*

*pyspark --packages org.apache.spark:spark-sql-kafka-0-10\_2.11:2.3.2 –master local[1]*

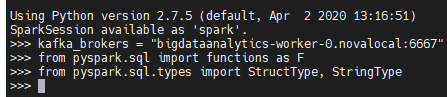
Packages успешно скачался:



Добавляем kafka-брокер, импортируем типы объектов:

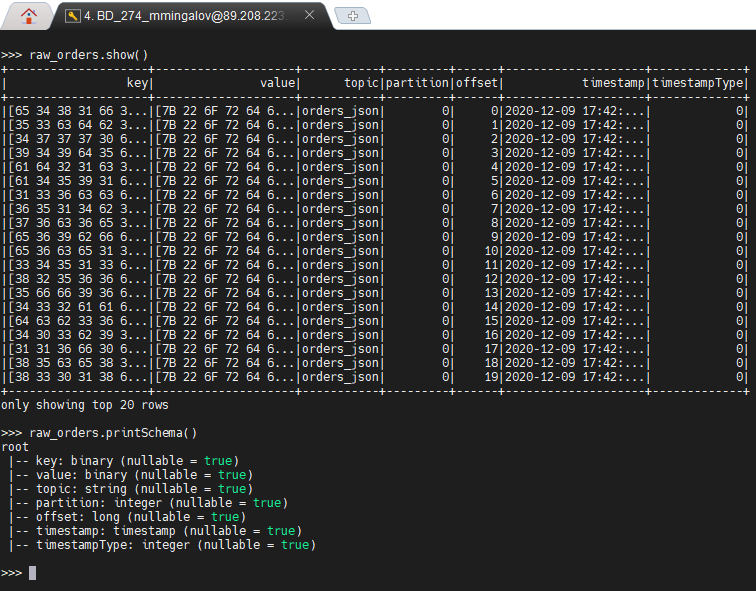
kafka\_brokers = **"bigdataanalytics-worker-0.novalocal:6667"**

**from** pyspark.sql **import** functions **as** F  
**from** pyspark.sql.types **import** StructType, StringType



*#функция, чтобы выводить на консоль вместо show()***def** console\_output(df, freq):  
 **return** df.writeStream \  
 .format(**"console"**) \  
 .trigger(processingTime=**'%s seconds'** % freq ) \  
 .options(truncate=**True**) \  
 .start()  
  
*#читаем без стрима*raw\_orders = spark.read. \  
 format(**"kafka"**). \  
 option(**"kafka.bootstrap.servers"**, kafka\_brokers). \  
 option(**"subscribe"**, **"orders\_json"**). \  
 option(**"startingOffsets"**, **"earliest"**). \  
 load()

Прочитаем командой .show(), посмотрим схему: :



**ПРАКТИКА**

Можно подсчитать кол-во записей

**raw\_order.count()**

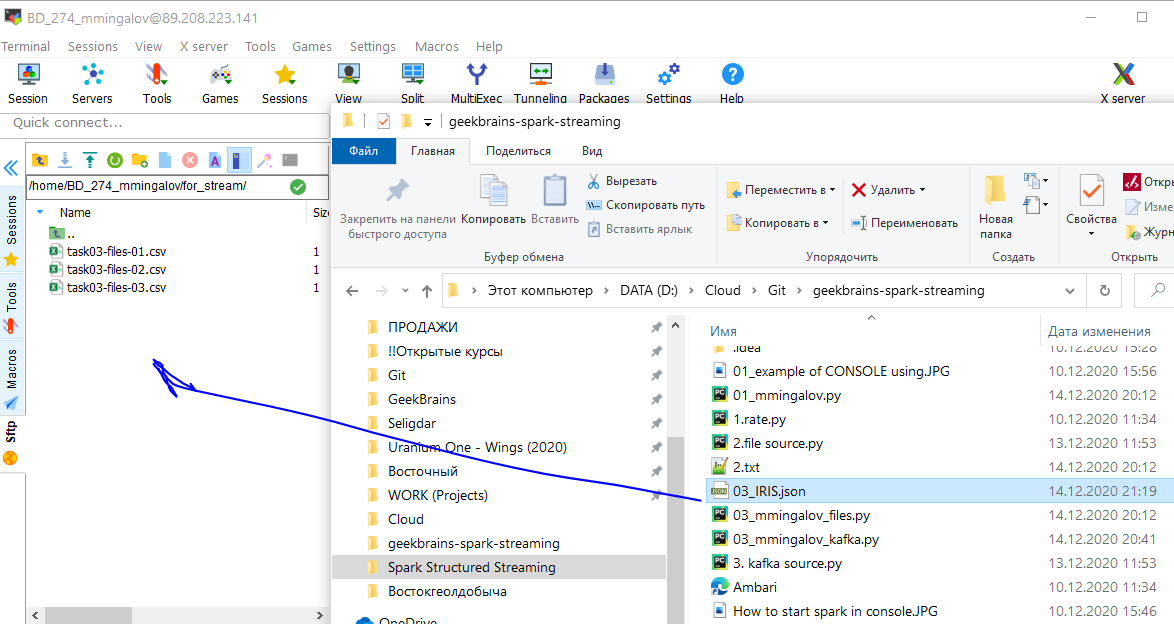
*#прочитали до 20го оффсета*raw\_orders = spark.read. \  
 format(**"kafka"**). \  
 option(**"kafka.bootstrap.servers"**, kafka\_brokers). \  
 option(**"subscribe"**, **"orders\_json"**). \  
 option(**"startingOffsets"**, **"earliest"**). \  
 option(**"endingOffsets"**, **"""{"orders\_json":{"0":20}}"""**). \  
 load()  
  
raw\_orders.show(100)

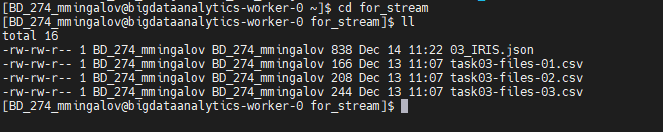
**Переходим на STREAM**

*# прочитали в стриме ВСЁ*raw\_orders = spark.readStream. \  
 format(**"kafka"**). \  
 option(**"kafka.bootstrap.servers"**, kafka\_brokers). \  
 option(**"subscribe"**, **"orders\_json"**). \  
 option(**"startingOffsets"**, **"earliest"**). \  
 load()  
  
out = console\_output(raw\_orders, 5)  
out.stop()  
  
*# прочитали потихоньку*raw\_orders = spark.readStream. \  
 format(**"kafka"**). \  
 option(**"kafka.bootstrap.servers"**, kafka\_brokers). \  
 option(**"subscribe"**, **"orders\_json"**). \  
 option(**"startingOffsets"**, **"earliest"**). \  
 option(**"maxOffsetsPerTrigger"**, **"5"**). \  
 load()  
  
out = console\_output(raw\_orders, 5)  
out.stop()  
  
  
*# прочитали один раз с конца*raw\_orders = spark.readStream. \  
 format(**"kafka"**). \  
 option(**"kafka.bootstrap.servers"**, kafka\_brokers). \  
 option(**"subscribe"**, **"orders\_json"**). \  
 option(**"maxOffsetsPerTrigger"**, **"5"**). \  
 option(**"startingOffsets"**, **"latest"**). \  
 load()  
  
out = console\_output(raw\_orders, 5)  
out.stop()  
  
  
*# прочитали с 10го оффсета*raw\_orders = spark.readStream. \  
 format(**"kafka"**). \  
 option(**"kafka.bootstrap.servers"**, kafka\_brokers). \  
 option(**"subscribe"**, **"orders\_json"**). \  
 option(**"startingOffsets"**, **"""{"orders\_json":{"0":10}}"""**). \  
 option(**"maxOffsetsPerTrigger"**, **"5"**). \  
 load()  
  
out = console\_output(raw\_orders, 5)  
out.stop()  
  
  
*##разбираем value*schema = StructType() \  
 .add(**"order\_id"**, StringType()) \  
 .add(**"customer\_id"**, StringType()) \  
 .add(**"order\_status"**, StringType()) \  
 .add(**"order\_purchase\_timestamp"**, StringType()) \  
 .add(**"order\_approved\_at"**, StringType()) \  
 .add(**"order\_delivered\_carrier\_date"**, StringType()) \  
 .add(**"order\_delivered\_customer\_date"**, StringType()) \  
 .add(**"order\_estimated\_delivery\_date"**, StringType())  
  
  
value\_orders = raw\_orders \  
 .select(F.from\_json(F.col(**"value"**).cast(**"String"**), schema).alias(**"value"**), **"offset"**)  
  
value\_orders.printSchema()  
  
parsed\_orders = value\_orders.select(**"value.\*"**, **"offset"**)  
  
parsed\_orders.printSchema()  
  
out = console\_output(parsed\_orders, 30)  
out.stop()  
  
*#добавляем чекпоинт***def** console\_output\_checkpointed(df, freq):  
 **return** df.writeStream \  
 .format(**"console"**) \  
 .trigger(processingTime=**'%s seconds'** % freq) \  
 .option(**"truncate"**,**False**) \  
 .option(**"checkpointLocation"**, **"orders\_console\_checkpoint"**) \  
 .start()  
  
out = console\_output\_checkpointed(parsed\_orders, 5)  
out.stop()

# **ДОП. ПРАКТИКА с чтением ФАЙЛА JSON**

Скачиваем json файл с ресурса Kaggle, помещаем его в нашу локальную директорию на хосте:



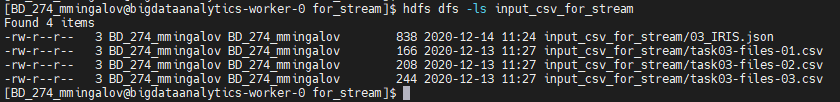


**Копируем в директорию HDFS**

hdfs dfs -put 03\_IRIS.json input\_csv\_for\_stream

**Проверяем что скопировался**

hdfs dfs -ls input\_csv\_for\_stream



Идем в **pyspark**, выполняем команды

**from** pyspark.sql **import** functions **as** F  
**from** pyspark.sql.types **import** StructType, StringType

**def** console\_output(df, freq):  
 **return** df.writeStream \  
 .format(**"console"**) \  
 .trigger(processingTime=**'%s seconds'** % freq ) \  
 .options(truncate=**False**) \  
 .start()

schema = StructType() \  
 .add(**"sepalLength"**, StringType()) \  
 .add(**"sepalWidth"**, StringType()) \  
 .add(**"petalLength"**, StringType()) \  
 .add(**"petalWidth"**, StringType()) \  
 .add(**"species"**, StringType())  
  
*#все разом*raw\_files = spark \  
 .readStream \  
 .format(**"json"**) \  
 .schema(schema) \  
 .options(path=**"input\_csv\_for\_stream"**, header=**True**) \  
 .load()

out = console\_output(raw\_files, 5)

