

Текстовая расшифровка видео:

НАЧАЛО РАБОТЫ

План:

- Самодостаточная система;
- Запуск приложений;
- С точки зрения кода;
- Отличия SparkContext и SparkSession;
- Источники, приемники, форматы.

Самодостаточная система

Spark можно [скачать с сайта](#) (аналогично с Hadoop). Однако есть нюанс: если вы хотите писать под Spark на Python, то **скачанный с сайта Spark поддерживает только конкретные версии Python**.

Рассмотрим на примере.

Это скачанный Spark (версия 2.4.8), который мы распаковали. Python, который стоит – 3.11:

```
meow-nofer@clefairy /opt/spark/bin$ cd ..
meow-nofer@clefairy /opt/spark$ ls
bin  data  jars  LICENSE  NOTICE  R  RELEASE  yarn
conf  examples  kubernetes  licenses  python  README.md  sbin
meow-nofer@clefairy /opt/spark$ cd bin
meow-nofer@clefairy /opt/spark/bin$ python
Python 3.11.3 (main, Jun 5 2023, 09:32:32) [GCC 13.1.1 20230429] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



▶ Запустить стенд

⌚ Дедлайн 07 июля, 23:59 Мск



```
meow-nofer@clefairy /opt/spark/bin /pyspark
Python 3.11.3 (main, Jun 5 2023, 09:32:32) [GCC 13.1.1 20230429] on Linux
Type "help", "copyright", "credits" or "license" for more information.
Traceback (most recent call last):
  File "/opt/spark/python/pyspark/shell.py", line 31, in <module>
    from pyspark import SparkConf
  File "/opt/spark/python/pyspark/__init__.py", line 51, in <module>
    from pyspark.context import SparkContext
  File "/opt/spark/python/pyspark/context.py", line 31, in <module>
    from pyspark import accumulators
  File "/opt/spark/python/pyspark/accumulators.py", line 97, in <module>
    from pyspark.serializers import read_int, PickleSerializer
  File "/opt/spark/python/pyspark/serializers.py", line 72, in <module>
    from pyspark import cloudpickle
  File "/opt/spark/python/pyspark/cloudpickle.py", line 145, in <module>
    _cell_set_template_code = _make_cell_set_template_code()
  File "/opt/spark/python/pyspark/cloudpickle.py", line 126, in _make_cell_set_template_code
    return types.CodeType(
TypeError: code expected at least 16 arguments, got 15
>>>
```

Подобный подход можно использовать, но важно подобрать подходящую версию Python.

Еще один вариант – **поставить клиент для Spark вместе с пакетом PySpark**. Он доступен из pip'a:

```
~$ pip install pyspark
```

Для практики лучше воспользоваться самым простым вариантом, а именно взять подсобранный PySpark в контейнере с уже готовым Jupyter'ом:

```
~$ docker pull jupyter/pyspark-notebook
```

```
~$ docker run --name pyspark -p 8888:8888 -p 4040:4040 jupyter/pyspark-notebook
```

Нужно скачать соответствующий образ докера с Docker Hub и запустить его, прокинув порты. Так как у нас Jupyter – 8888. 4040 – отладочный UI Spark'a.

Перейдем в каталог и скажем «docker run»:

```
meow-nofer@clefairy /opt/spark/bin cd ~/Experiments/slurm
meow-nofer@clefairy ~/Experiments/slurm docker run --rm --name pyspark -v $(pwd):/home
-p 4040:4040 jupyter/pyspark-notebook
```

Добавим «-v» в качестве домашнего каталога пользователя внутри контейнера, так как в текущем каталоге есть дополнительные файлы.

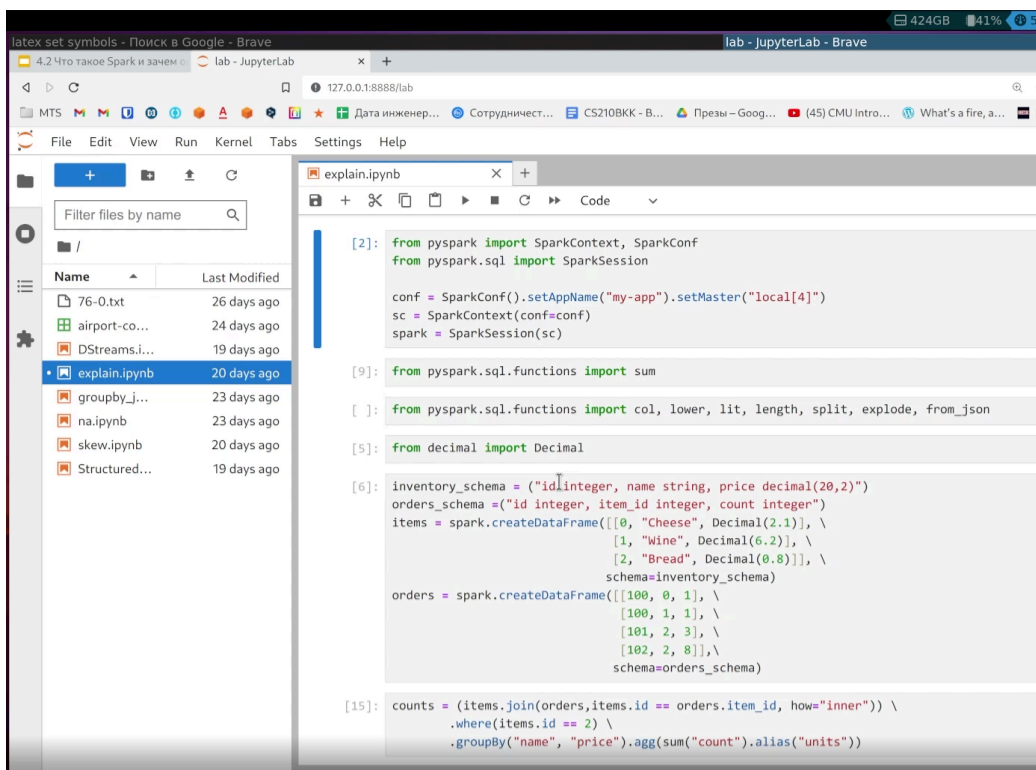
Запустили:

```
meow-nofer@clefairy /opt/spark/bin cd ~/Experiments/slurm
meow-nofer@clefairy ~/Experiments/slurm docker run --rm --name pyspark -v $(pwd):/home
-p 4040:4040 jupyter/pyspark-notebook
Entered start.sh with args: jupyter lab
/usr/local/bin/start.sh: running hooks in /usr/local/bin/before-notebook.d as uid / gid:
/usr/local/bin/start.sh: running script /usr/local/bin/before-notebook.d/spark-config.sh
/usr/local/bin/start.sh: done running hooks in /usr/local/bin/before-notebook.d
Executing the command: jupyter lab
```

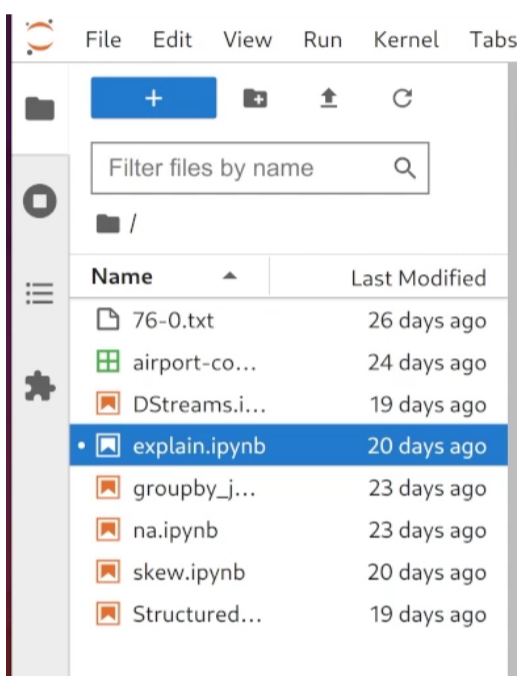
Если вы уже пробовали работать с Jupyter'ом, то знаете, что он «выплевывает» адрес, куда нужно подключаться с токеном:

```
To access the server, open this file in a browser:
file:///home/jovyan/.local/share/jupyter/runtime/jpserver-7-open.html
Or copy and paste one of these URLs:
http://d6b85a7160d7:8888/lab?token=245cccb1419edfb9cfc0fb222d96b375c7f9d91de8f52f38
http://127.0.0.1:8888/lab?token=245cccb1419edfb9cfc0fb222d96b375c7f9d91de8f52f38
[I 2023-07-20 11:15:37.892 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-language
-server-nodejs, javascript-typescript-langserver, jedi-language-server, julia-language-server, pyright, python-l
anguage-server, python-lsp-server, r-languageserver, sql-language-server, texlab, typescript-language-server, un
ified-language-server, vscode-css-languageserver-bin, vscode-html-languageserver-bin, vscode-json-languageserver
-bin, yaml-language-server
```

Мы можем зайти в браузер, скопировать целиком URL; у нас откроется интерфейс Jupyter'a, где можно



Обратите внимание: поскольку мы добавили локальный каталог, у нас появились несколько разных демонстрационных notebooks:



Запуск приложений

Запуск приложений на кластере Spark происходит несколькими способами:

- **Использовать shell.** Можно использовать Spark-shell, если используете язык программирования Scala, а PySpark, если пишете на Python.
- **Использовать отдельную утилиту – Spark-submit.** Мы пишем код, работающий с драйвером Spark'a. С помощью Spark-submit мы задаем дополнительный параметр того, как все сконфигурировать, и передаем код для запуска на кластере.

Дополнительно можно ознакомиться с UI Spark'a, перейдя по ссылке: <http://127.0.0.1:4040/jobs/>

Важно: чтобы ссылка открылась, и у Spark'a запустился UI нужно проинициализировать драйвер, создать контекст.

С точки зрения кода

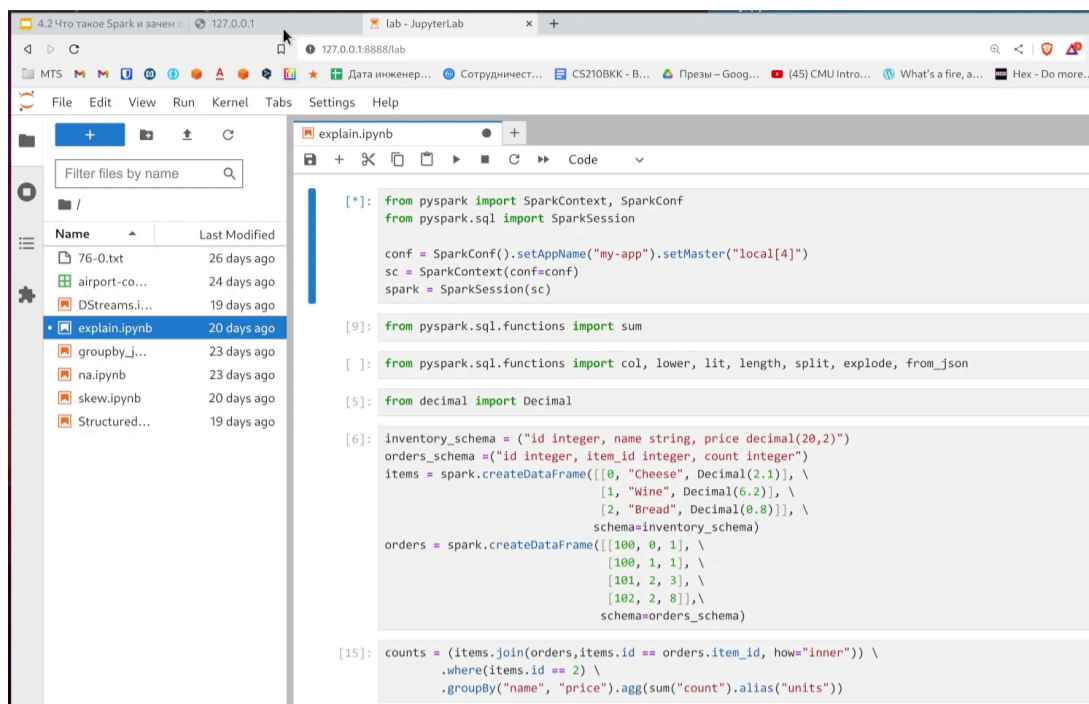
Рассмотрим пример:

```
1. from pyspark import SparkContext, SparkConf
2.
3. conf = SparkConf().setAppName("my-app").setMaster("local[4]")
4. sc = SparkContext(conf=conf)
```

Мы импортируем SparkContext для получения доступа к кластеру (также можно взять SparkConf, чтобы сконфигурировать кластер так, как мы хотим).

В целом требования минималистичные: мы можем задать базовое имя для приложения, а также задать то, какую конфигурацию хотим использовать. Поскольку в данном примере мы запускаем Spark на локальной машине, то говорим «запусти конфигурацию с четырьмя воркерами», все будет считаться на них. Далее, мы создаем объект SparkContext'a, указывая соответствующий конфигурационный файл.

В примерах с notebooks код более развесистый:



```
[*]: from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession

conf = SparkConf().setAppName("my-app").setMaster("local[4]")
sc = SparkContext(conf=conf)
spark = SparkSession(sc)

[9]: from pyspark.sql.functions import sum

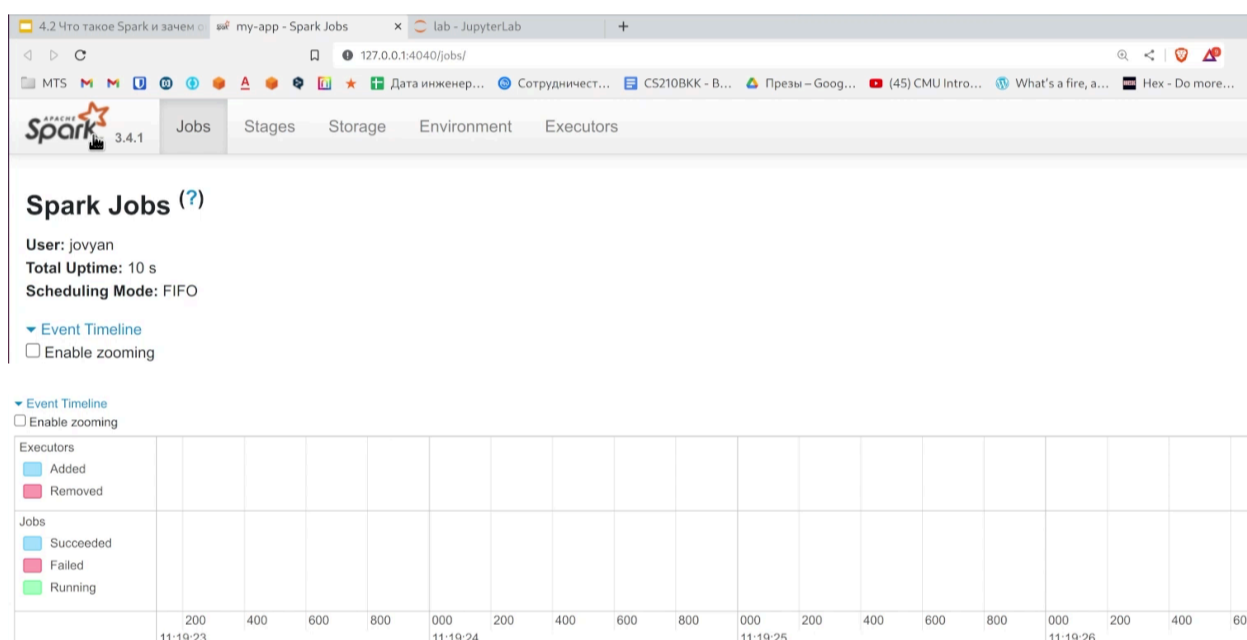
[]: from pyspark.sql.functions import col, lower, lit, length, split, explode, from_json

[5]: from decimal import Decimal

[6]: inventory_schema = ("id integer, name string, price decimal(20,2)")
orders_schema = ("id integer, item_id integer, count integer")
items = spark.createDataFrame([[0, "Cheese", Decimal(2.1)], \
                               [1, "Wine", Decimal(6.2)], \
                               [2, "Bread", Decimal(0.8)]], \
                              schema=inventory_schema)
orders = spark.createDataFrame([[100, 0, 1], \
                                [100, 1, 1], \
                                [101, 2, 3], \
                                [102, 2, 8]], \
                               schema=orders_schema)

[15]: counts = (items.join(orders, items.id == orders.item_id, how="inner")) \
            .where(items.id == 2) \
            .groupBy("name", "price").agg(sum("count").alias("units"))
```

Для начала запустим ячейку с инициализатором. По порту 4040 должен подняться UI самого Spark'a:



Здесь пока ничего особенного не будет, однако, если потребуется запускать сложные нагрузки и отлаживать их, то мы получим большую поддержку от данного UI.

Отличия SparkContext и SparkSession

SparkContext:

- Появился в Spark 1.x;
- Исторически использовался для создания RDD и «размазывания» данных по кластеру;
- Доступен в spark-shell, как sc.

SparkSession:

- Появился в Spark 2.x;
- Фактически заменил собой большую часть SparkContext;
- Включает в себя обертки для Hive, streaming'a и прочего;
- Позволяет работать как с RDD, так и с Dataframe/Dataset;
- Доступен в spark-shell, как spark.

Источники, приемники, форматы

Так как мы запускаем все на своей локальной машине, мы можем работать с локальной файловой системой. Однако, если говорить о распределенном кластере, там необходимо хранилище.

Очевидно, поддерживаются HDFS, S3, а также коннекторы для других баз данных: Redis, Elasticsearch, MangoDB.

Для большинства реляционных баз и баз, которые поддерживают работу через SQL-запросы, есть реализация Java – Database connector (JDBC). Мы можем подключаться к Postgres SQL, My SQL и т.д.

Форматы, с которыми Spark может работать:

- CSV;
- JSON;
- XML;
- RCFile;
- Parquet;
- ORC;
- Avro.

Как вам урок?



Изучил, далее >

Слёрм ©

[+7 \(495\) 248-05-80](tel:+74952480580)

[Лицензия №ДЛ-1368 от 22.08.2019](#)

[Политика конфиденциальности](#)

[Публичная оферта](#)