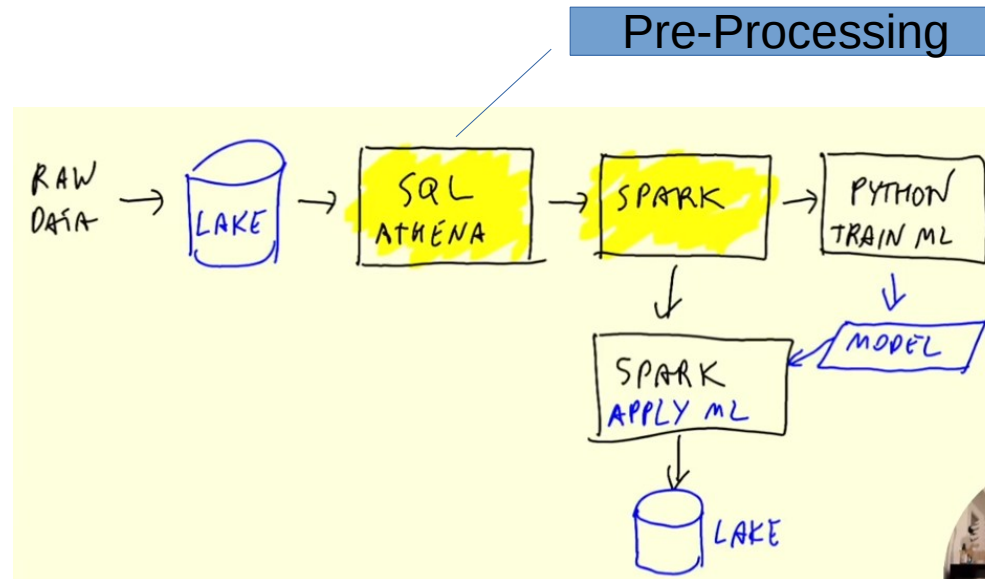


# Batch Processing

# Apache Spark

- Data Processing Engine
- Can be used for Batch and Streaming
- Used when data is in a data lake
- Usual Workflow:



# Setup Spark

## Linux

Here we'll show you how to install Spark 3.3.2 for Linux. We tested it on Ubuntu 20.04 (also WSL), but it should work for other Linux distros as well

### Installing Java

Download OpenJDK 11 or Oracle JDK 11 (It's important that the version is 11 - spark requires 8 or 11)

We'll use [OpenJDK](#)

Download it (e.g. to `~/spark`):

```
wget https://download.java.net/java/GA/jdk11/9/GPL/openjdk-11.0.2_linux-x64_bin.tar.gz
```

Unpack it:

```
tar xzfv openjdk-11.0.2_linux-x64_bin.tar.gz
```

define `JAVA_HOME` and add it to `PATH`:

```
export JAVA_HOME="${HOME}/spark/jdk-11.0.2"
export PATH="${JAVA_HOME}/bin:${PATH}"
```

check that it works:

```
java --version
```

Output:

```
openjdk 11.0.2 2019-01-15
OpenJDK Runtime Environment 18.9 (build 11.0.2+9)
OpenJDK 64-Bit Server VM 18.9 (build 11.0.2+9, mixed mode)
```

# Setup Spark

Remove the archive:

```
rm openjdk-11.0.2_linux-x64_bin.tar.gz
```

## Installing Spark

Download Spark. Use 3.3.2 version:

```
wget https://archive.apache.org/dist/spark/spark-3.3.2/spark-3.3.2-bin-hadoop3.tgz
```

Unpack:

```
tar xzfv spark-3.3.2-bin-hadoop3.tgz
```

Remove the archive:

```
rm spark-3.3.2-bin-hadoop3.tgz
```

Add it to `PATH` :

```
export SPARK_HOME="${HOME}/spark/spark-3.3.2-bin-hadoop3"
export PATH="${SPARK_HOME}/bin:${PATH}"
```

## Testing Spark

Execute `spark-shell` and run the following:

```
val data = 1 to 10000
val distData = sc.parallelize(data)
distData.filter(_ < 10).collect()
```

# Setup PySpark

This document assumes you already have python.

To run PySpark, we first need to add it to `PYTHONPATH` :

```
export PYTHONPATH="${SPARK_HOME}/python/:$PYTHONPATH"
export PYTHONPATH="${SPARK_HOME}/python/lib/py4j-0.10.9.5-src.zip:$PYTHONPATH"
```



Make sure that the version under `${SPARK_HOME}/python/lib/` matches the filename of py4j or you will encounter `ModuleNotFoundError: No module named 'py4j'` while executing `import pyspark`.

For example, if the file under `${SPARK_HOME}/python/lib/` is `py4j-0.10.9.3-src.zip`, then the `export PYTHONPATH` statement above should be changed to

```
export PYTHONPATH="${SPARK_HOME}/python/lib/py4j-0.10.9.3-src.zip:$PYTHONPATH"
```



# Setup PySpark

Now you can run Jupyter or IPython to test if things work. Go to some other directory, e.g. `~/tmp`.

Download a CSV file that we'll use for testing:

```
wget https://d37ci6vzurychx.cloudfront.net/misc/taxi_zone_lookup.csv
```



Now let's run `ipython` (or `jupyter notebook`) and execute:

```
import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .master("local[*]") \
    .appName('test') \
    .getOrCreate()

df = spark.read \
    .option("header", "true") \
    .csv('taxi_zone_lookup.csv')

df.show()
```



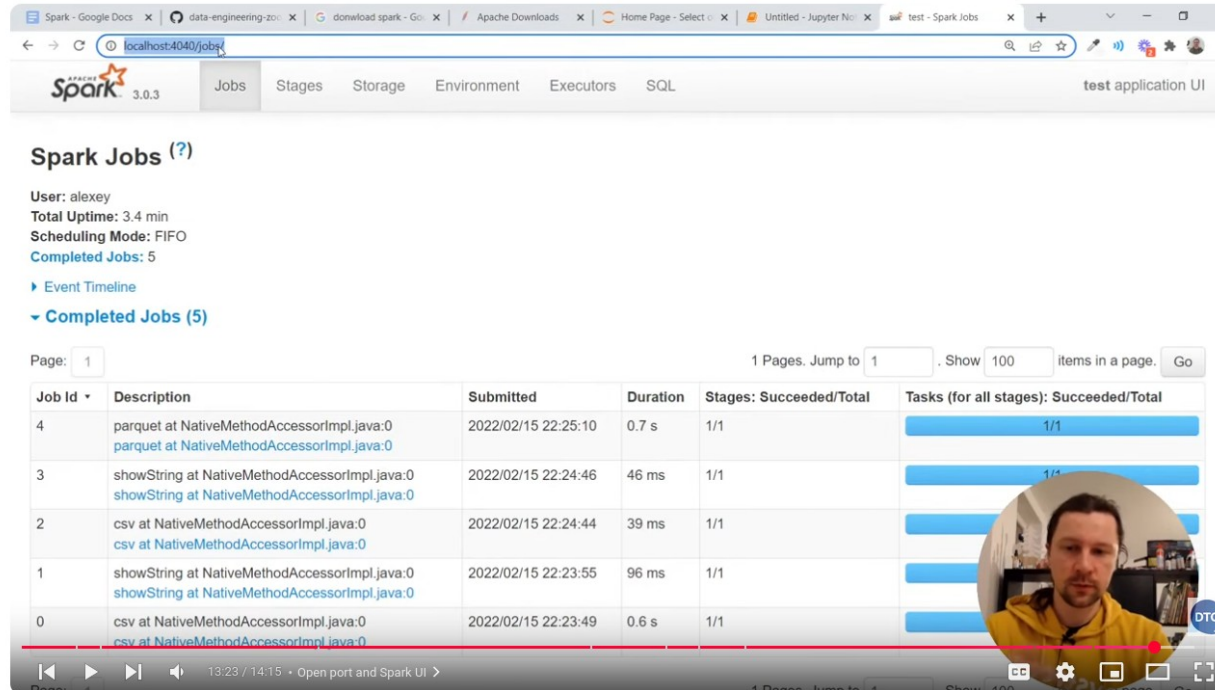
Test that writing works as well:

```
df.write.parquet('zones')
```



# Access Spark jobs

<http://localhost:4040/jobs/>

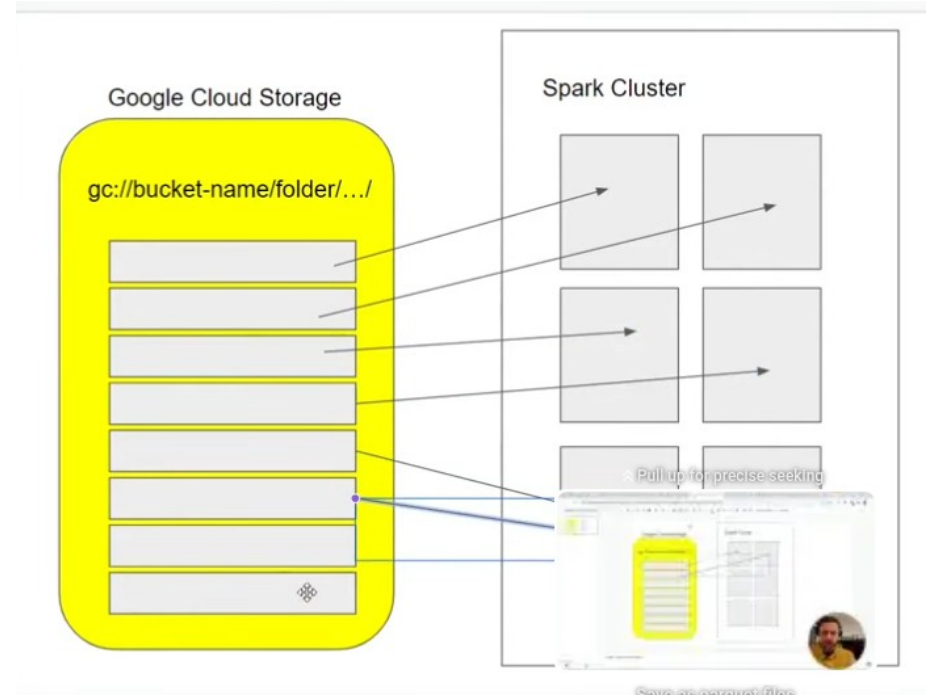


The screenshot shows the Apache Spark 3.0.3 Jobs UI. The browser address bar displays `localhost:4040/jobs/`. The page title is "Spark Jobs (?)". Below the title, it shows "User: alexey", "Total Uptime: 3.4 min", "Scheduling Mode: FIFO", and "Completed Jobs: 5". A link for "Event Timeline" is present. A section titled "Completed Jobs (5)" shows a table of jobs. The table has columns: Job Id, Description, Submitted, Duration, Stages: Succeeded/Total, and Tasks (for all stages): Succeeded/Total. All jobs are completed. A video player is overlaid on the bottom right of the screenshot, showing a person in a yellow shirt.

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	parquet at NativeMethodAccessorImpl.java:0 <a href="#">parquet at NativeMethodAccessorImpl.java:0</a>	2022/02/15 22:25:10	0.7 s	1/1	1/1
3	showString at NativeMethodAccessorImpl.java:0 <a href="#">showString at NativeMethodAccessorImpl.java:0</a>	2022/02/15 22:24:46	46 ms	1/1	1/1
2	csv at NativeMethodAccessorImpl.java:0 <a href="#">csv at NativeMethodAccessorImpl.java:0</a>	2022/02/15 22:24:44	39 ms	1/1	1/1
1	showString at NativeMethodAccessorImpl.java:0 <a href="#">showString at NativeMethodAccessorImpl.java:0</a>	2022/02/15 22:23:55	96 ms	1/1	1/1
0	csv at NativeMethodAccessorImpl.java:0 <a href="#">csv at NativeMethodAccessorImpl.java:0</a>	2022/02/15 22:23:49	0.6 s	1/1	1/1

# Spark Clusters

- Spark can create a cluster of executors
- To take advantage of this, the data should be partitioned in several files (instead of just one huge file which can only be processed by one executor)





## Spark differentiates between actions and transformations

- Transformations are not executed immediately (lazy), Spark creates a sequence of transformations.
- Actions are executed immediately (eager), they trigger the execution of transformations in the pipeline.

## Why use Spark instead of SQL to transform data?

- Spark can be written as python code, thus making the code testable
- Spark provides user defined functions, which can modularized certain transformations
- Spark also offers some built-in functions

## Add custom functions

```
26]: def crazy_stuff(base_num):  
      num = int(base_num[1:])  
      if num % 7 == 0:  
          return f's/{num:03x}'  
      elif num % 3 == 0:  
          return f'a/{num:03x}'  
      else:  
          return f'e/{num:03x}'
```

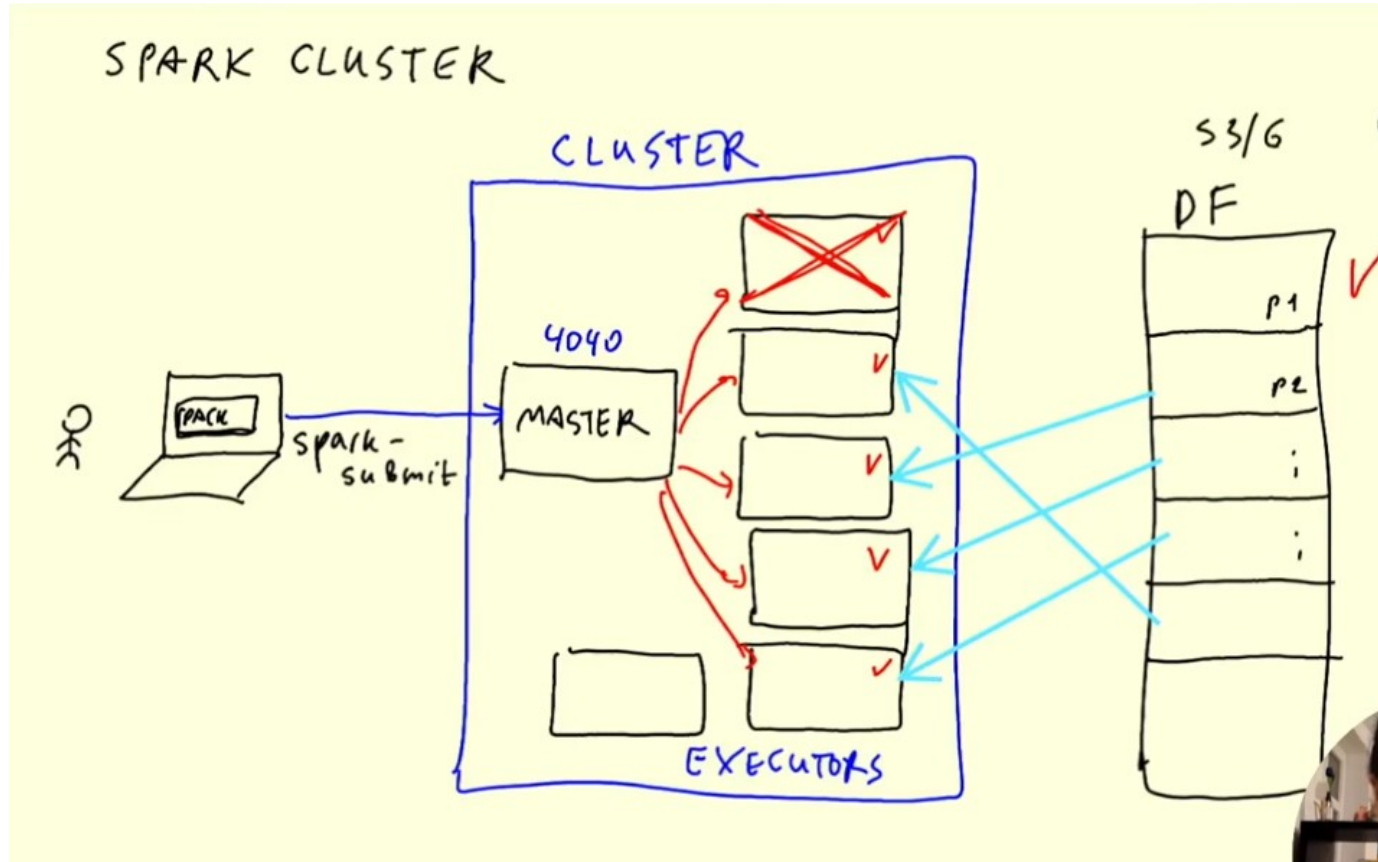
```
27]: crazy_stuff('B02884')
```

```
27]: 's/b44'
```

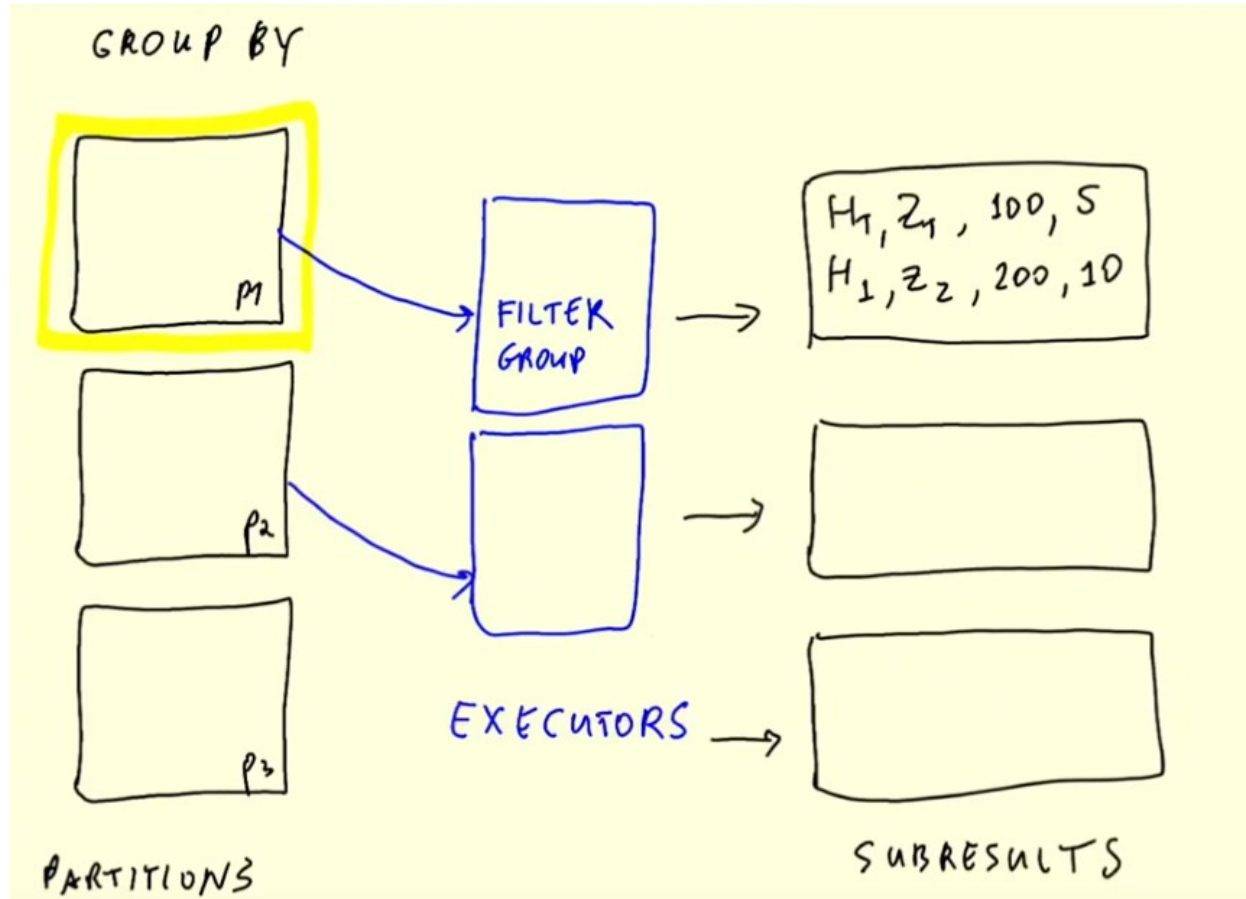
```
28]: crazy_stuff_udf = F.udf(crazy_stuff, returnType=types.StringType())
```

```
29]: df \  
      .withColumn('pickup_date', F.to_date(df.pickup_datetime)) \  
      .withColumn('dropoff_date', F.to_date(df.dropoff_datetime)) \  
      .withColumn('base_id', crazy_stuff_udf(df.dispatching_base_num)) \  
      .select('base_id', 'pickup_date', 'dropoff_date', 'PULocationID', 'DOLocationID') \  
      .show()
```

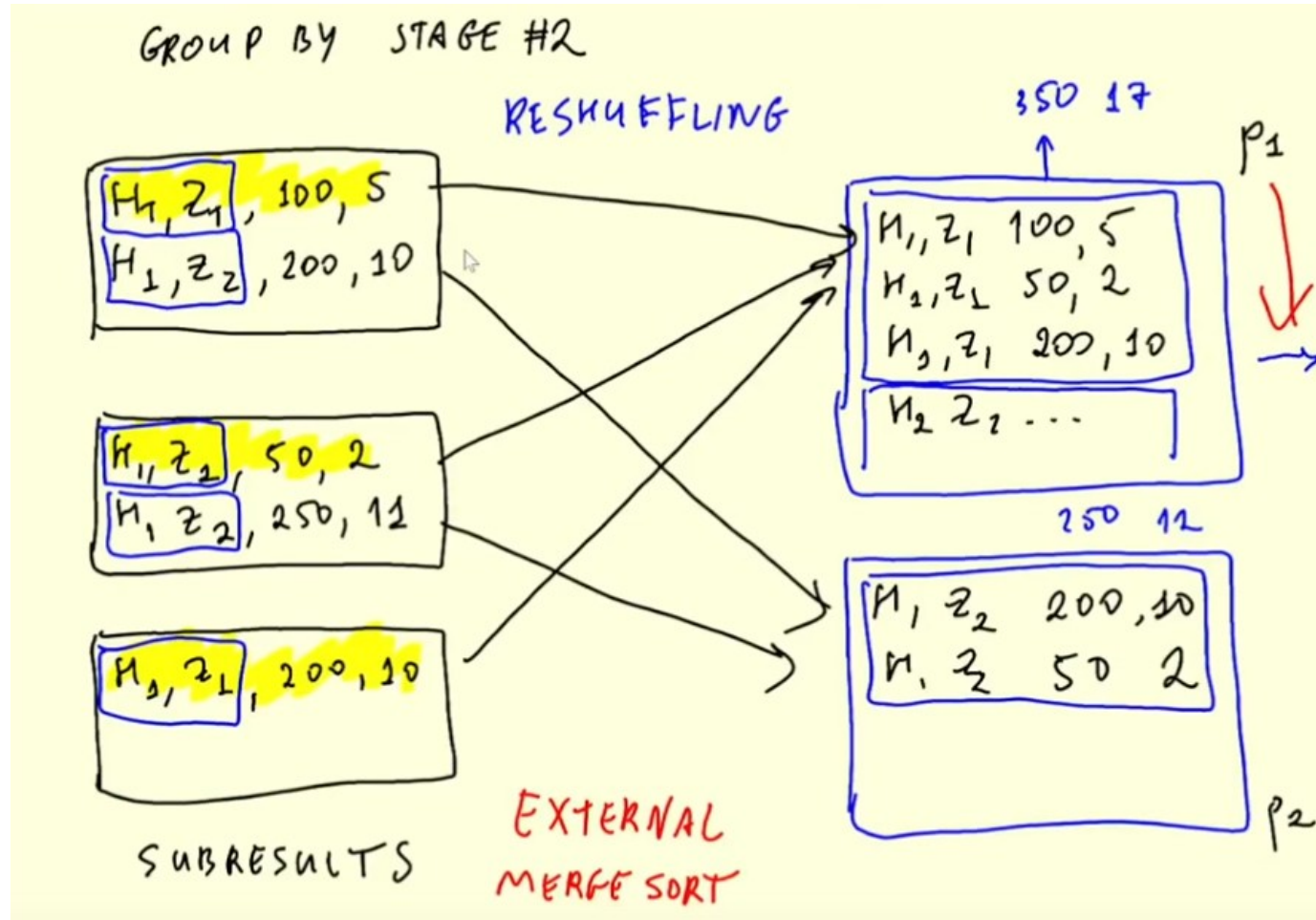
# Anatomy of a Spark Cluster



# Spark GroupBy



# Spark GroupBy



# Spark Joins

# Spark in GCP

Use all available cores

- First, the data must be uploaded to a bucket.
- Create a bucket and use this command:

```
gsutil -m cp -r pq/ gs://dlp_2025_dateng_week_05/pq
```

Recursive (because it contains subfolders)

- If the access is denied, try this:
  - 1) Stop VM
  - 2) goto --> VM instance details.
  - 3) in "Cloud API access scopes" select "Allow full access to all Cloud APIs" then Click "save".
  - 4) restart VM and Delete ~/.gsutil .

# Spark in GCP

- We also need a hadoop connector to GCS, so that Spark can connect to the GCS bucket.
- The connector has to be compatible with the installed Spark version:

```
dlp@instance-20250227-050901:~/.google/credentials$ spark-submit --version
Welcome to

  ____      _
 / ___ \    / \
/ /   \ \  / _ \
\ \___) \ / ___ \
 \___) \/_/   \_\
  version 3.5.5

Using Scala version 2.12.18, OpenJDK 64-Bit Server VM, 11.0.2
Branch HEAD
Compiled by user ubuntu on 2025-02-23T20:30:46Z
Revision 7c29c664cdc9321205a98a14858aaf8daaa19db2
Url https://github.com/apache/spark
Type --help for more information.
dlp@instance-20250227-050901:~/.google/credentials$
```

```
gsutil cp gs://hadoop-lib/gcs/gcs-connector-hadoop3-2.2.5.jar ~/lib/
```

# Spark in GCP

```
localhost:8888/lab/tree/notebooks/09_spark_gcs_dlp.ipynb
[1]: import pyspark
    from pyspark.sql import SparkSession
    from pyspark.conf import SparkConf
    from pyspark.context import SparkContext

[2]: credentials_location = '/home/dlp/.google/credentials/google_credentials.json'

    conf = SparkConf() \
        .setMaster('local[*]') \
        .setAppName('test') \
        .set("spark.jars", "../lib/gcs-connector-hadoop3-2.2.5.jar") \
        .set("spark.hadoop.google.cloud.auth.service.account.enable", "true") \
        .set("spark.hadoop.google.cloud.auth.service.account.json.keyfile", credentials_location)

[3]: sc = SparkContext(conf=conf)

    hadoop_conf = sc._jsc.hadoopConfiguration()

    hadoop_conf.set("fs.AbstractFileSystem.gs.impl", "com.google.cloud.hadoop.fs.gcs.GoogleHadoopFS")
    hadoop_conf.set("fs.gs.impl", "com.google.cloud.hadoop.fs.gcs.GoogleHadoopFileSystem")
    hadoop_conf.set("fs.gs.auth.service.account.json.keyfile", credentials_location)
    hadoop_conf.set("fs.gs.auth.service.account.enable", "true")

25/03/10 03:41:03 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

[4]: spark = SparkSession.builder \
    .config(conf=sc.getConf()) \
    .getOrCreate()

[5]: df_green = spark.read.parquet('gs://dlp_2025_dateng_week_05/pq/green/**')

[6]: df_green.count()

[6]: 2304517
```

Distribution of directories in VM

```
dlp@instance-20250227-050901:~$ ls
Programs data de-zoomcamp download_data.sh lib notebooks snap spark tmp
dlp@instance-20250227-050901:~$
```

This credentials is the .json key for the VM instance service account

IAM & Admin / Service accounts / Service account: 102422904072881661184 / Keys

Compute Engine default service account

DETAILS PERMISSIONS KEYS METRICS LOGS

**Keys**

Service account keys could pose a security risk if compromised. We recommend you avoid downloading service account keys.

Google automatically disables service account keys detected in public repositories. You can customize this behavior by [configuring Google Cloud IAM](#).

Add a new key pair or upload a public key certificate from an existing key pair.

Block service account key creation using [organization policies](#) (2). [Learn more about setting organization policies for service accounts](#) (2)

**ADD KEY**

Type	Status	Key	Creation date	Expiration date	
Key	Active		Mar 9, 2025	Dec 31, 9999	



# Creating a Local Spark Cluster

<https://spark.apache.org/docs/latest/spark-standalone.html#installing-spark-standalone-to-a-cluster>

```
./sbin/start-master.sh
```

- Go to the directory saved as SPARK\_HOME

```
dlp@instance-20250227-050901:~/spark$ echo $SPARK_HOME
/home/dlp/spark/spark-3.5.5-bin-hadoop3
```

- ```
./sbin/start-master.sh
```

```
dlp@instance-20250227-050901:~/spark/spark-3.5.5-bin-hadoop3$ ./sbin/start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /home/dlp/spark/spark-3.5.5-bin-hadoop3/logs
Master-1-instance-20250227-050901.out
dlp@instance-20250227-050901:~/spark/spark-3.5.5-bin-hadoop3$
```

- This local cluster can be accessed on Port 8080, so map this port (VM Instance → local host) and check the UI in a browser

| PROBLEMS              |      | OUTPUT            |  | DEBUG CONSOLE |  | TERMINAL |  | PORTS <span>3</span> |  |
|-----------------------|------|-------------------|--|---------------|--|----------|--|----------------------|--|
| Port                  |      | Forwarded Address |  |               |  |          |  |                      |  |
| <input type="radio"/> | 4040 | localhost:4040    |  |               |  |          |  |                      |  |
| <input type="radio"/> | 8080 | localhost:8080    |  |               |  |          |  |                      |  |
| <input type="radio"/> | 8888 | localhost:8888    |  |               |  |          |  |                      |  |
| <div>Add Port</div>   |      |                   |  |               |  |          |  |                      |  |

Spark 3.5.5 Spark Master at spark://instance-20250227-050901.us-west3-c.c.dateng-dlp-05.internal:7077

URL: spark://instance-20250227-050901.us-west3-c.c.dateng-dlp-05.internal:7077

Alive Workers: 0  
Cores in use: 0 Total, 0 Used  
Memory in use: 0.0 B Total, 0.0 B Used

Resources in use:  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

Workers (0)

| Worker Id | Address | State | Cores | Memory |
|-----------|---------|-------|-------|--------|
|-----------|---------|-------|-------|--------|

Running Applications (0)

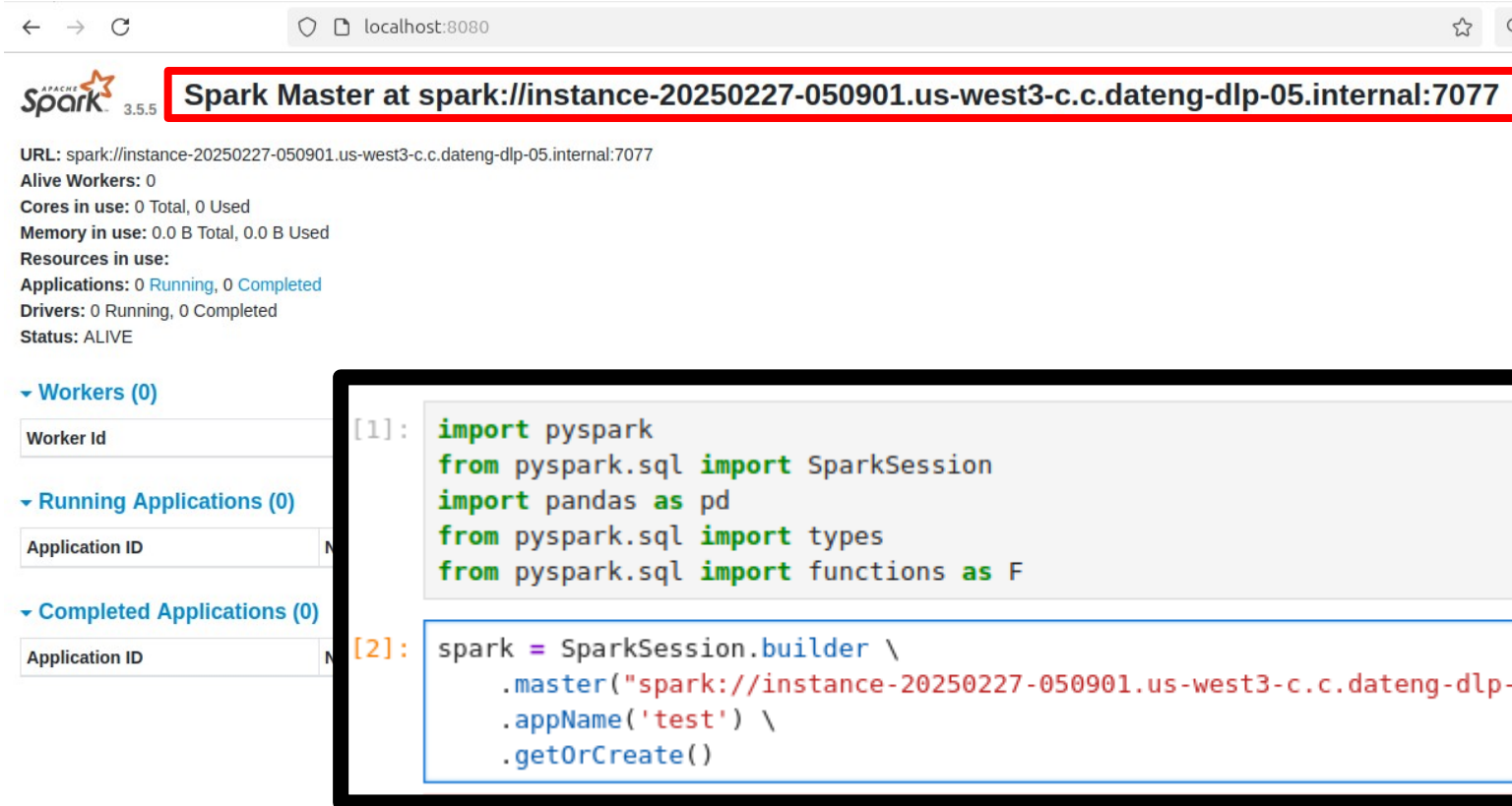
| Application ID | Name | Cores | Memory per Executor | Resources Per Executor |
|----------------|------|-------|---------------------|------------------------|
|----------------|------|-------|---------------------|------------------------|

Completed Applications (0)

| Application ID | Name | Cores | Memory per Executor | Resources Per Executor |
|----------------|------|-------|---------------------|------------------------|
|----------------|------|-------|---------------------|------------------------|

# Creating a Local Spark Cluster

- Now change the master from local to the created cluster



The screenshot shows the Databricks Spark UI interface. The browser address bar displays 'localhost:8080'. The main header shows the Apache Spark logo and version '3.5.5'. A red box highlights the 'Spark Master' URL: 'spark://instance-20250227-050901.us-west3-c.c.dateng-dlp-05.internal:7077'. Below this, the URL is repeated. The status section shows 'Alive Workers: 0', 'Cores in use: 0 Total, 0 Used', 'Memory in use: 0.0 B Total, 0.0 B Used', 'Resources in use:', 'Applications: 0 Running, 0 Completed', 'Drivers: 0 Running, 0 Completed', and 'Status: ALIVE'. The left sidebar has sections for 'Workers (0)', 'Running Applications (0)', and 'Completed Applications (0)'. A code block on the right shows the following Python code:

```
[1]: import pyspark
      from pyspark.sql import SparkSession
      import pandas as pd
      from pyspark.sql import types
      from pyspark.sql import functions as F

[2]: spark = SparkSession.builder \
      .master("spark://instance-20250227-050901.us-west3-c.c.dateng-dlp-05.internal:7077") \
      .appName('test') \
      .getOrCreate()
```

# Creating a Local Spark Cluster

- A Spark session can now be created but we need to manually create workers in order for the cluster to be able to do anything.
- Now the script to obtain the revenue can be run (with input parameters)

```
./sbin/start-worker.sh <master-spark-URL>
```

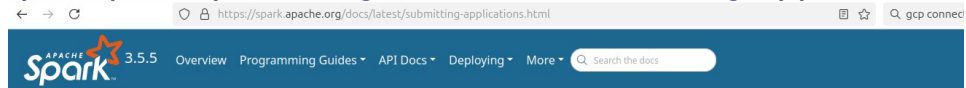
```
10_spark_sql_local_cluster_dlp.py X
notebooks > 10_spark_sql_local_cluster_dlp.py
1  import argparse
2  import pyspark
3  from pyspark.sql import SparkSession
4  import pandas as pd
5  from pyspark.sql import types
6  from pyspark.sql import functions as F
7
8  parser = argparse.ArgumentParser()
9
10 parser.add_argument('--input_green', required=True)
11 parser.add_argument('--input_yellow', required=True)
12 parser.add_argument('--output', required=True)
13
14 args = parser.parse_args()
15
16 input_green = args.input_green
17 input_yellow = args.input_yellow
18 output = args.output
19
20
21 spark = SparkSession.builder \
22     .master("spark://instance-20250227-050901.us-west3-c.c.dateng-dlp-05.internal:7077") \
23     .appName('test') \
24     .getOrCreate()
25
26
27 # Read green taxi data
28 df_green = spark.read.parquet(input_green)
29 df_green.printSchema()
```

# Creating a Local Spark Cluster

- However, we don't want to hard code the Spark master, we want to be able to define the number of executors and other settings.
- We use Spark Submit for this!

```
21 # spark = SparkSession.builder \  
22 #   .master("spark://instance-20250227-050901.us-west3-c.c.dateng-dlp-05.internal:7077") \  
23 #   .appName('test') \  
24 #   .getOrCreate() \  
25 \  
26 spark = SparkSession.builder \  
27   .appName('test') \  
28   .getOrCreate()
```

<https://spark.apache.org/docs/latest/submitting-applications.html>



## Submitting Applications

The `spark-submit` script in Spark's bin directory is used to launch applications on a cluster. It can use all of Spark's supported [cluster managers](#) through a uniform interface so you don't have to configure your application especially for each one.

## Bundling Your Application's Dependencies

If your code depends on other projects, you will need to package them alongside your application in order to distribute the code to a Spark cluster. To do this, create an assembly jar (or "uber" jar) containing your code and its dependencies. Both [sbt](#) and [Maven](#) have assembly plugins. When creating assembly jars, list Spark and Hadoop as provided dependencies; these need not be bundled since they are provided by the cluster manager at runtime. Once you have an assembled jar you can call the `bin/spark-submit` script as shown here while passing your jar.

For Python, you can use the `--py-files` argument of `spark-submit` to add `.py`, `.zip` or `.egg` files to be distributed with your application. If you depend on multiple Python files we recommend packaging them into a `.zip` or `.egg`. For third-party Python dependencies, see [Python Package Management](#).

## Launching Applications with spark-submit

Once a user application is bundled, it can be launched using the `bin/spark-submit` script. This script takes care of setting up the classpath with Spark and its dependencies, and can support different cluster managers and deploy modes that Spark supports:

```
./bin/spark-submit \  
--class <main-class> \  
--master <master-url> \  
--deploy-mode <deploy-mode> \  
--conf <key>=<value> \  
... # other options  
<application-jar> \  
[application-arguments]
```

# Submitting jobs with spark-submit

```
URL="spark://instance-20250227-050901.us-west3-c.c.dateng-dlp-05.internal:7077"
```

```
spark-submit \
```

```
--master="${URL}" \
10_spark_sql_local_cluster_dlp.py \
--input_green='../data/pq/green/2021/*' \
--input_yellow='../data/pq/yellow/2021/*' \
--output='../data/report-2021'
```

```
(de-zoomcamp-py3.12) dlp@instance-20250227-050901:~/notebooks$ URL="spark://instance-20250227-050901.us-west3-c.c.dateng-dlp-05.internal:7077"
(de-zoomcamp-py3.12) dlp@instance-20250227-050901:~/notebooks$
(de-zoomcamp-py3.12) dlp@instance-20250227-050901:~/notebooks$ spark-submit \
> --master="${URL}" \
> 10_spark_sql_local_cluster_dlp.py \
> --input_green='../data/pq/green/2021/*' \
> --input_yellow='../data/pq/yellow/2021/*' \
> --output='../data/report-2021'
```

# Shutting down the Spark Cluster

- Once the Spark job is done, both the workers/executors and the master/cluster must be closed.
- Navigate to the folder where Spark was installed

```
dlp@instance-20250227-050901:~/spark/spark-3.5.5-bin-hadoop3$ ./sbin/stop-worker.sh
no org.apache.spark.deploy.worker.Worker to stop
dlp@instance-20250227-050901:~/spark/spark-3.5.5-bin-hadoop3$ ./sbin/stop-master.sh
no org.apache.spark.deploy.master.Master to stop
dlp@instance-20250227-050901:~/spark/spark-3.5.5-bin-hadoop3$
```