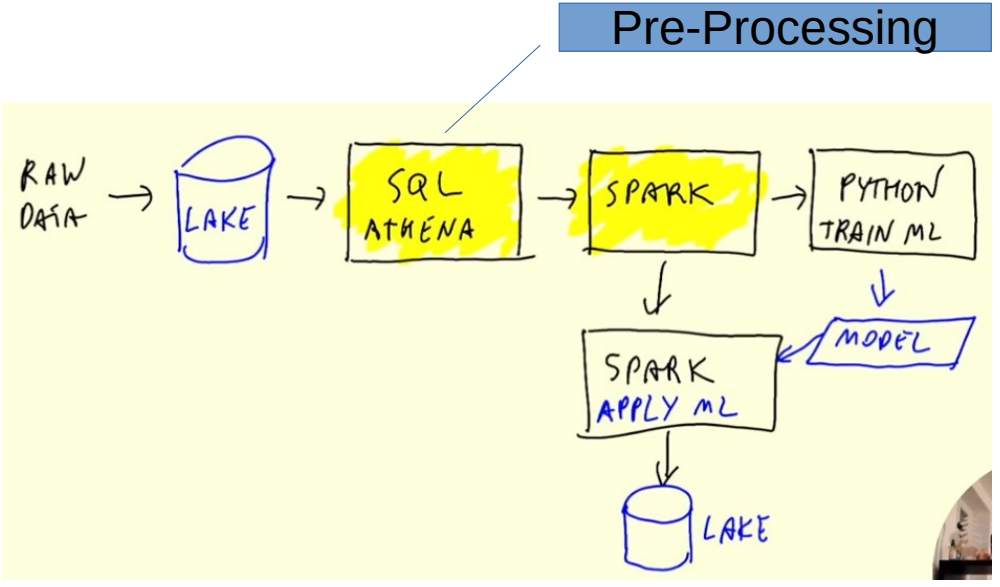
Batch Processing

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:

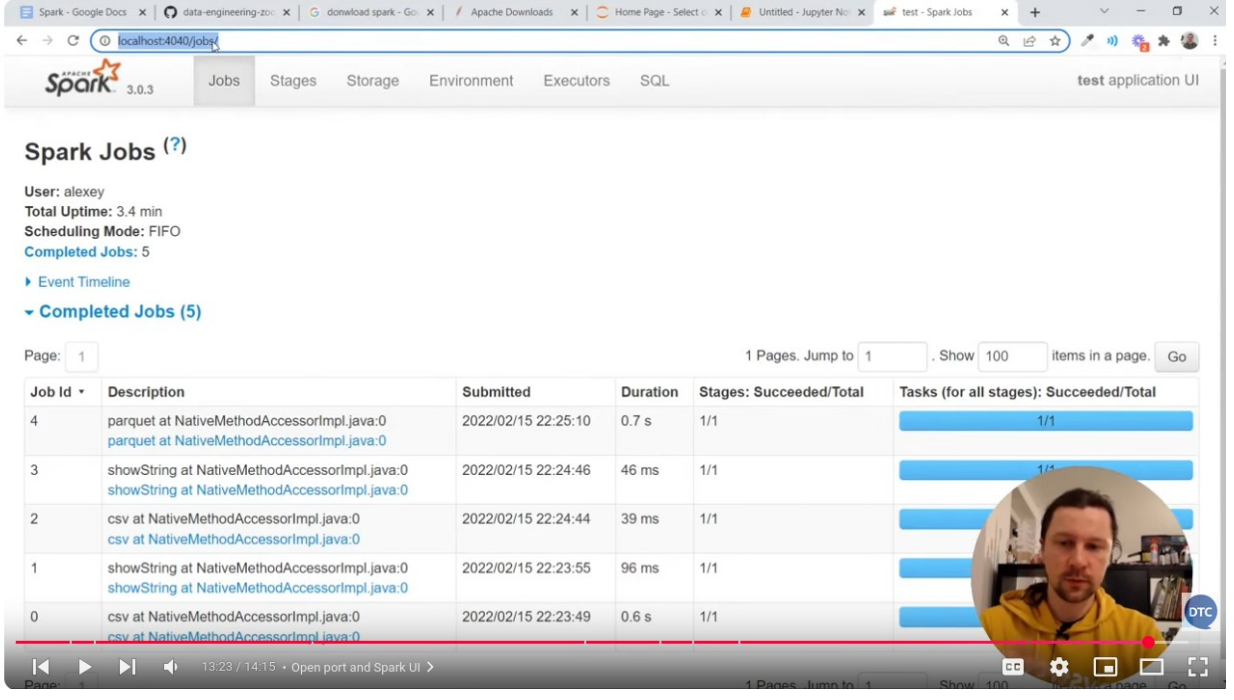
```
import pyspark
from pyspark.sql import SparkSession

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')
```

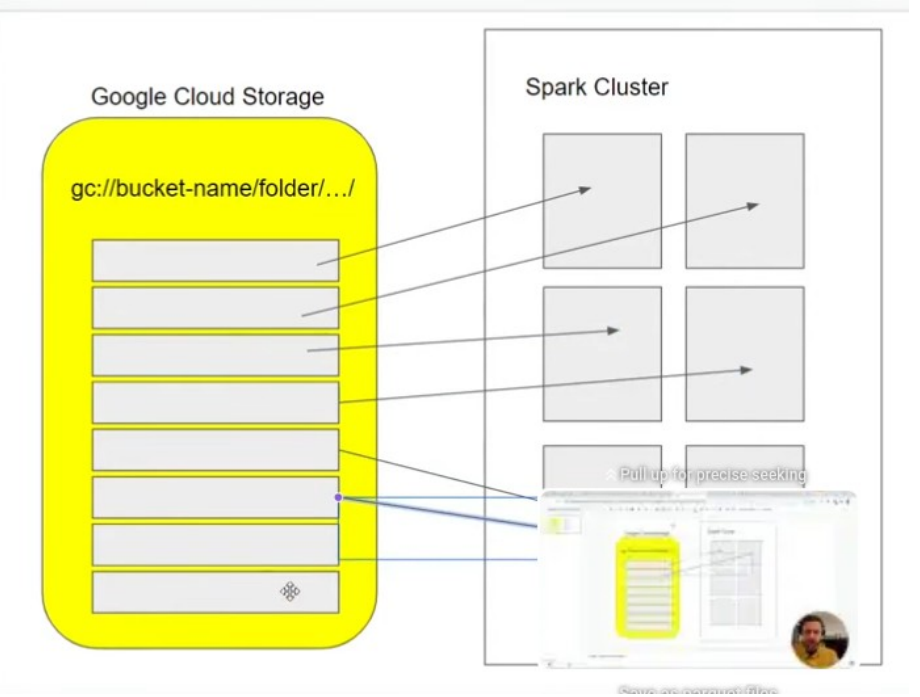
Access Spark jobs

http://localhost:4040/jobs/



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}'
```

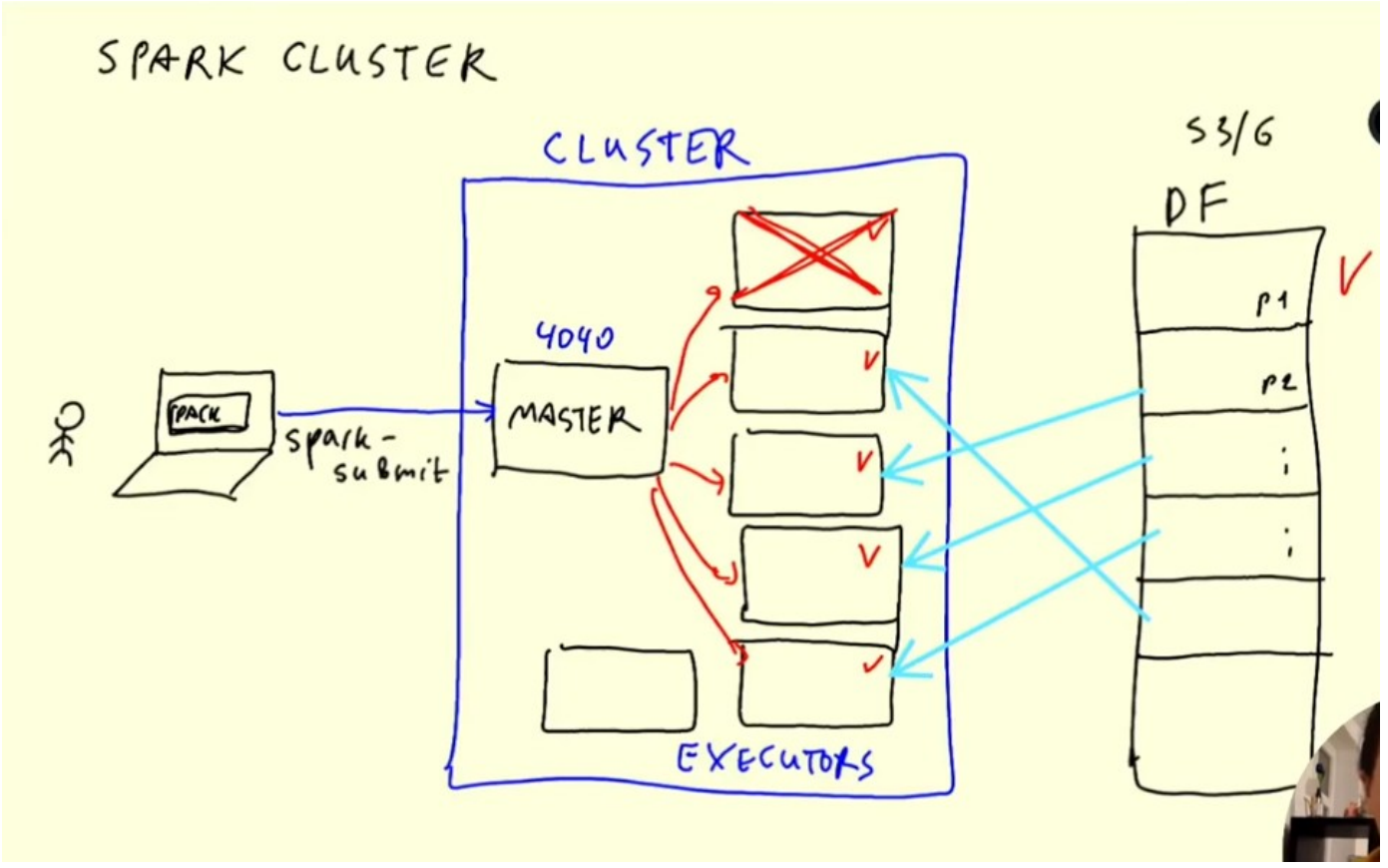
27]:
```
crazy_stuff('B02884')
```

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

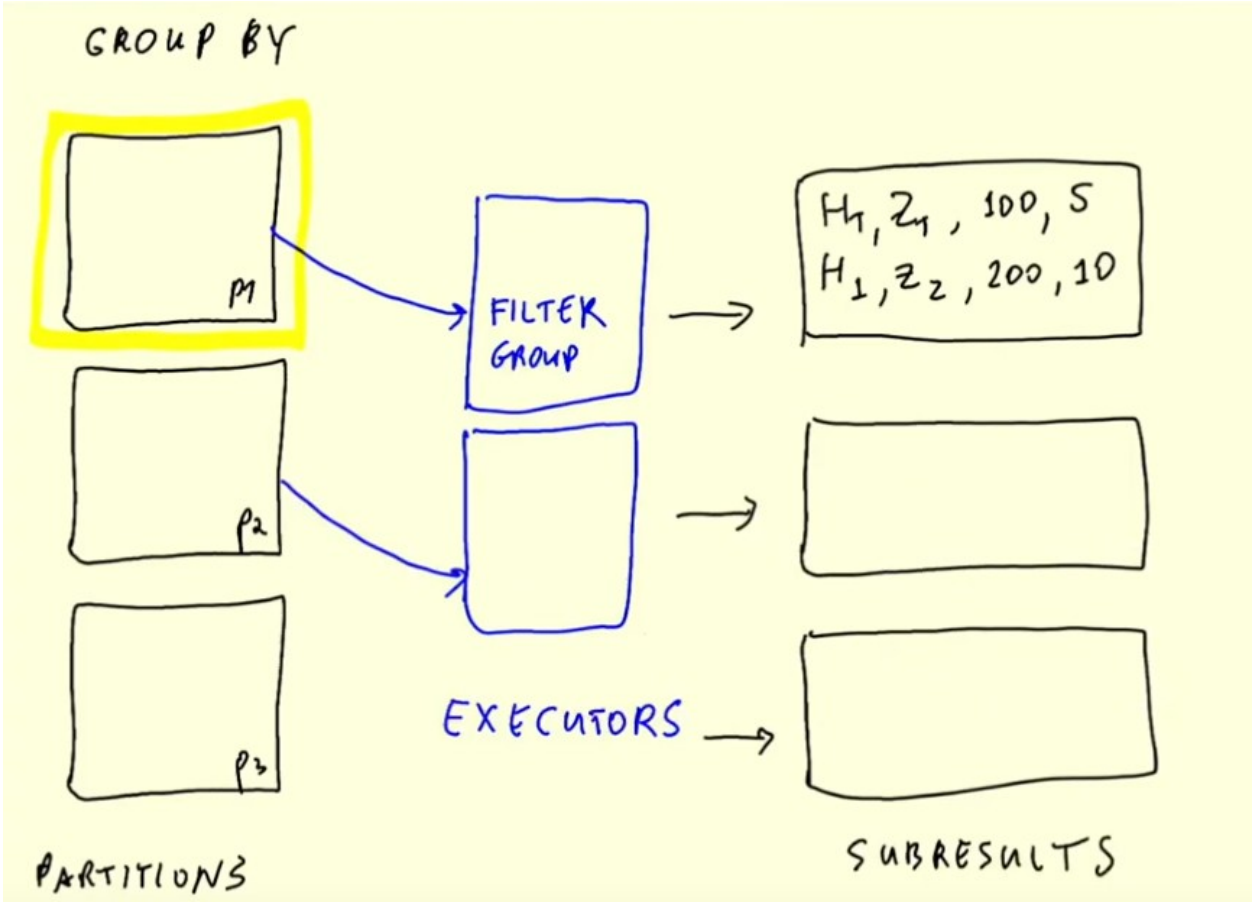
28]:
```
crazy_stuff_udf = F.udf(crazy_stuff, returnType=types.StringType())
```

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()
```

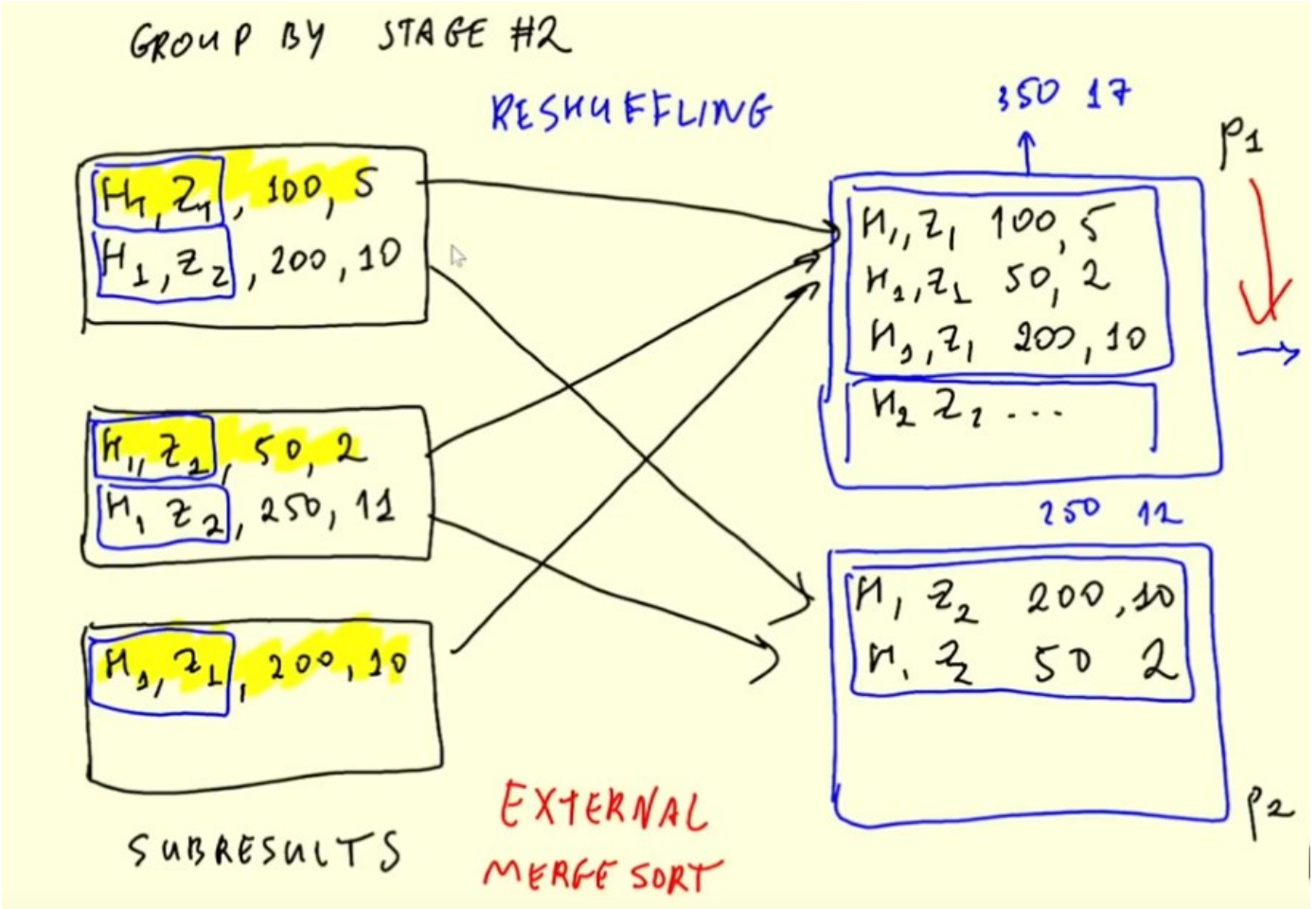
Anatomy of a Spark Cluster



Spark GroupBy



Spark GroupBy



Spark Joins

Spark in GCP

Use all available cores

Recursive (because it contains subfolders)

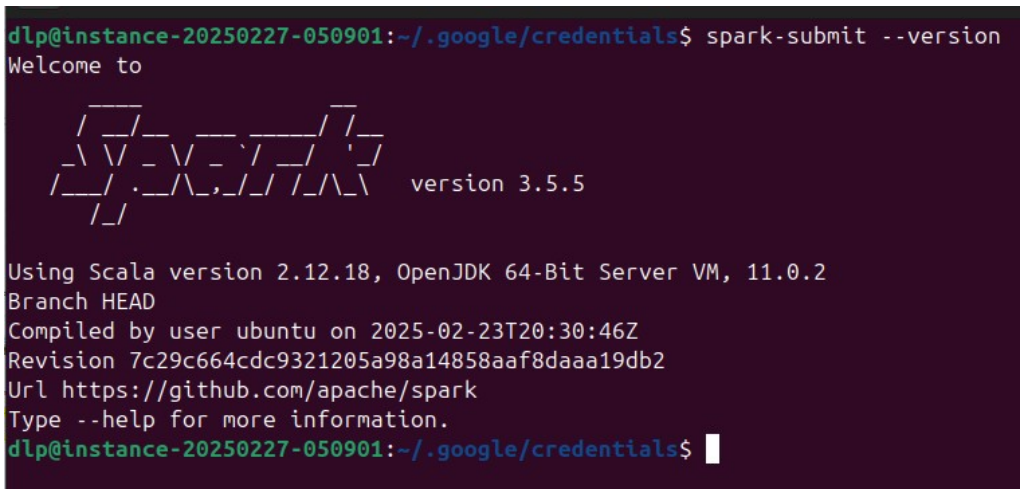
- First, the data must be uploaded to a bucket.
- Create a bucket and use this command:

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

- 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:




```
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

gcp connector hadoop

Settings Help

Test_PySpark_Install.ipynb | 04_PySpark_dlp.ipynb | 05_PySpark_dlp.ipynb | 06_spark_sql_dlp.ipynb | 08_rdds_dlp.ipynb | 09_spark_gcs_dlp.ipynb

Code

[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/*/*')
```

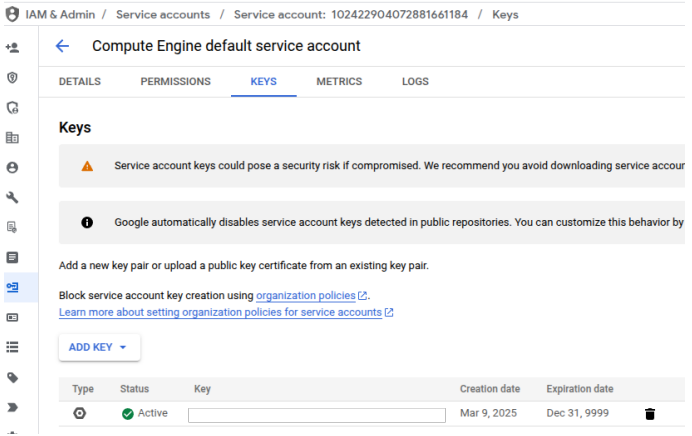
[6]:
```
df_green.count()
```

[6]:
```
2304517
```

Distribution of directories in VM

```
dlp@instance-20250227-050901: ~
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



Creating a Local Spark Cluster

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

`./sbin/start-master.sh`

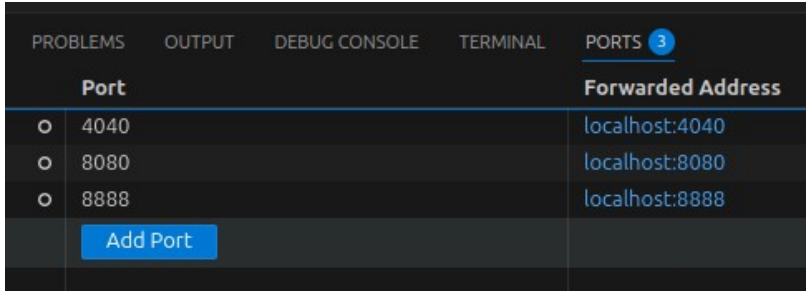
- Go to the directory saved as SPARK_HOME

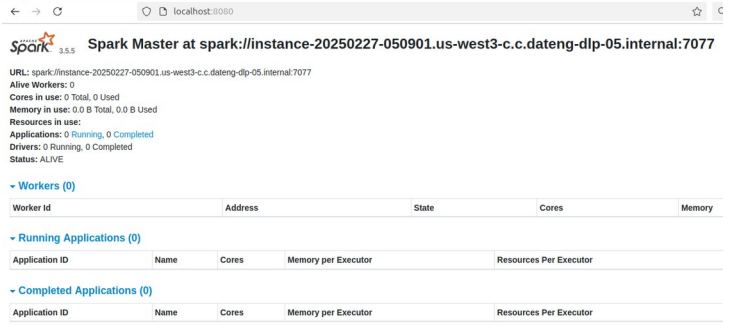
```
dlp@instance-20250227-050901:~/spark$ echo $SPARK_HOME
/home/dlp/spark/spark-3.5.5-bin-hadoop3
```

- `./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$
```

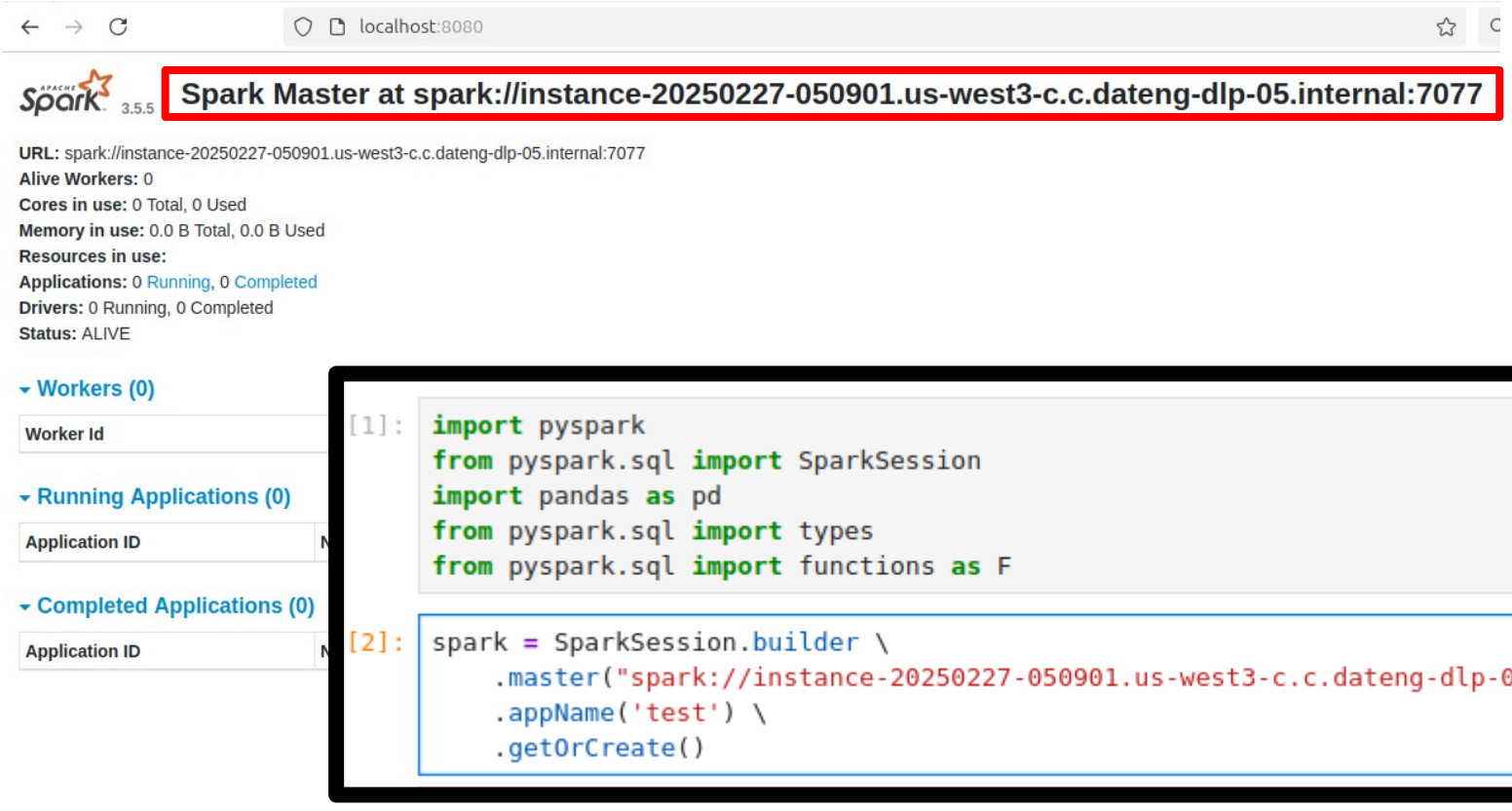
- This local cluster can be accessed on Port 8080, so map this port (VM Instance → local host) and check the UI in a browser





Creating a Local Spark Cluster

- Now change the master from local to the created cluster



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

notebooks > 10_spark_sql_local_cluster_dlp.py

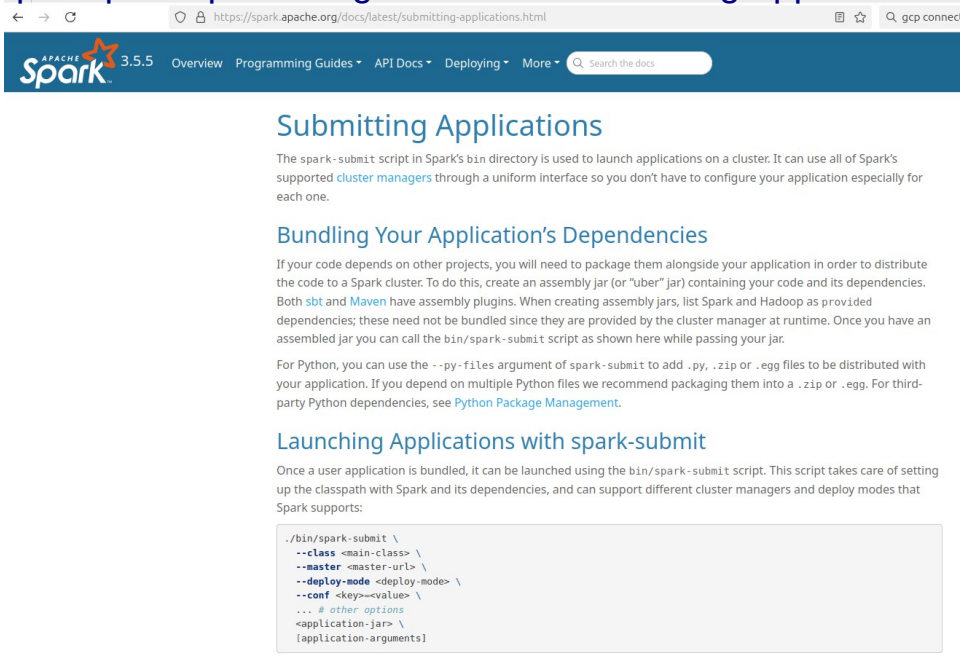
```python
import argparse
import pyspark
from pyspark.sql import SparkSession
import pandas as pd
from pyspark.sql import types
from pyspark.sql import functions as F

parser = argparse.ArgumentParser()

parser.add_argument('--input_green', required=True)
parser.add_argument('--input_yellow', required=True)
parser.add_argument('--output', required=True)

args = parser.parse_args()

input_green = args.input_green
input_yellow = args.input_yellow
output = args.output


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


# Read green taxi data
df_green = spark.read.parquet(input_green)
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 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