Lab 2 - First steps with Spark

Outline

1. Launching a Spark cluster on AWS
2. First steps with Spark

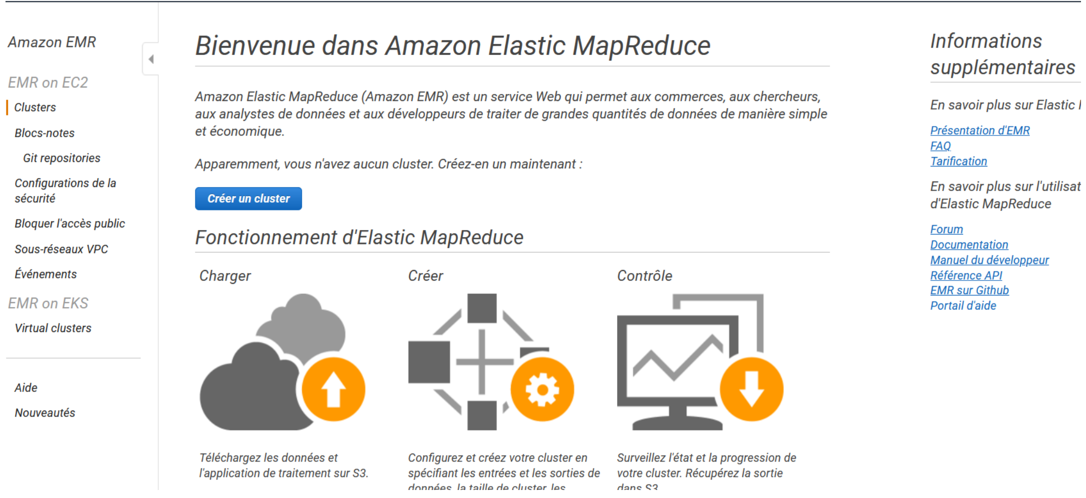
Lab 2 - First steps with Spark
- Outline
- ⛅ Spark cluster creation in AWS
- 💾 First steps with Spark - Data importation
  - 📝 Hands-on 1 - Data importation
- 🥉 Data frame basic manipulations
  - 😴 Lazy evaluation
  - 📝 Hands-on 2 - Data frame basic manipulations
- 🥈 Basic DataFrame column manipulation
  - 📝 Hands-on 3 - Basic DataFrame column manipulation
- 🥇 Advance DataFrame column manipulation
  - 🥽 Array manipulation
    - 📝 Hands-on 4 - Array manipulation
  - 🥼 User defined function
    - 📝 Hands-on 5 - User defined function
- 🗜 Aggregation functions
  - 📝 Hands-on 6 - Aggregation functions
- 🧲 Grouping functions
  - 📝 Hands-on 7 - Grouping functions
- 🔌 Spark SQL
  - 📝 Hands-on 8 - Spark SQL
  - Turn off your cluster

⛅ Spark cluster creation in AWS

First: **DO NOT FORGET TO TURN YOUR CLUSTER OFF A THE END OF THIS TUTORIAL!**

- [ ] Once connected to the management console, search for "EMR" (Elastic Map Reduce). It a platform as a service made to manage Hadoop cluster in AWS. You just have to choose the configuration of your cluster (how many machines ? How many CPU/Ram ? Which release for Spark ?) and AWS will create your cluster. Doing this all by yourself is time consuming and not a pleasant task. That's why cloud providers provide service like EMR.

- [ ] You should land on a page like this



Amazon EMR

EMR on EC2

Clusters

Blocs-notes

Git repositories

Configurations de la sécurité

Bloquer l'accès public

Sous-réseaux VPC

Événements

EMR on EKS

Virtual clusters

Aide

Nouveautés

Bienvenue dans Amazon Elastic MapReduce

Amazon Elastic MapReduce (Amazon EMR) est un service Web qui permet aux commerces, aux chercheurs, aux analystes de données et aux développeurs de traiter de grandes quantités de données de manière simple et économique.

Apparemment, vous n'avez aucun cluster. Créez-en un maintenant :

Créer un cluster

Fonctionnement d'Elastic MapReduce

Charger — Téléchargez les données et l'application de traitement sur S3.

Créer — Configurez et créez votre cluster en spécifiant les entrées et les sorties de données, la taille de cluster, les

Contrôle — Surveillez l'état et la progression de votre cluster. Récupérez la sortie dans S3

Informations supplémentaires

En savoir plus sur Elastic

Présentation d'EMR

FAQ

Tarification

En savoir plus sur l'utilisat d'Elastic MapReduce

Forum

Documentation

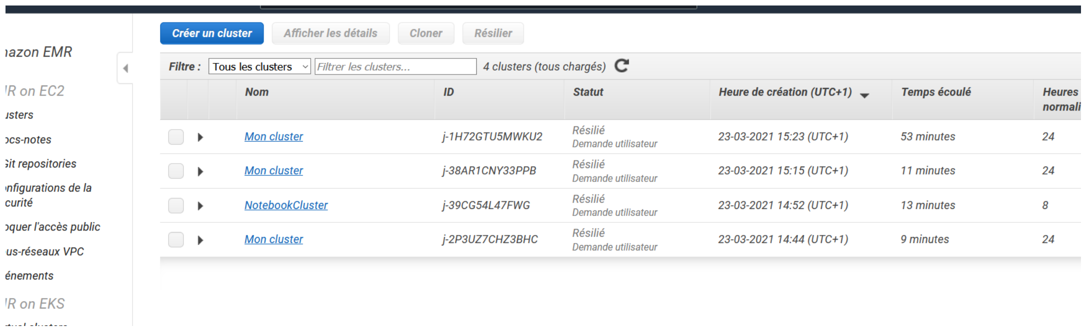
Manuel du développeur

Référence API

EMR sur Github

Portail d'aide

Next time it should be this one.



azon EMR

R on EC2

usters

ocs-notes

Git repositories

nfigurations de la curité

oquer l'accès public

us-réseaux VPC

énements

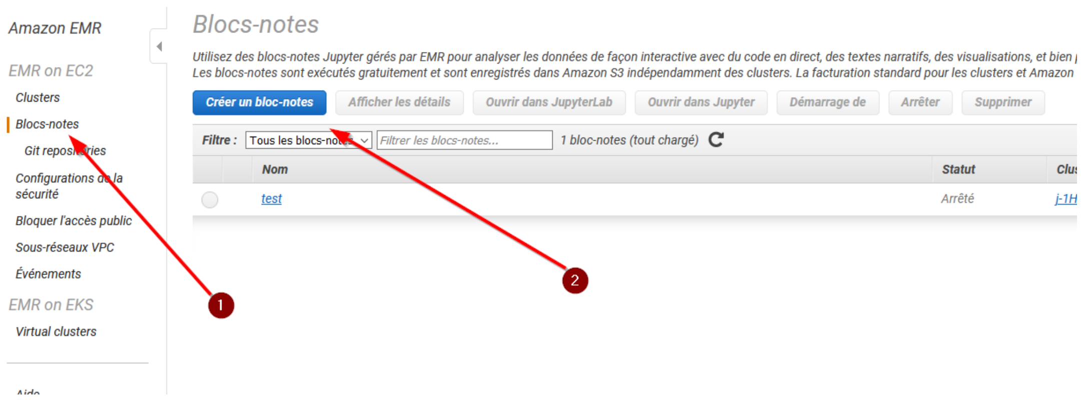
R on EKS

Créer un cluster · Afficher les détails · Cloner · Résilier

Filtre : Tous les clusters · Filtrer les clusters... · 4 clusters (tous chargés)

Nom	ID	Statut	Heure de création (UTC+1)	Temps écoulé	Heures normali
Mon cluster	j-1H72GTU5MWKU2	Résilié Demande utilisateur	23-03-2021 15:23 (UTC+1)	53 minutes	24
Mon cluster	j-38AR1CNY33PPB	Résilié Demande utilisateur	23-03-2021 15:15 (UTC+1)	11 minutes	24
NotebookCluster	j-39CG54L47FWG	Résilié Demande utilisateur	23-03-2021 14:52 (UTC+1)	13 minutes	8
Mon cluster	j-2P3UZ7CHZ3BHC	Résilié Demande utilisateur	23-03-2021 14:44 (UTC+1)	9 minutes	24

In every cases click on `Bloc-notes` then `Créer un bloc-notes`



Amazon EMR

EMR on EC2

Clusters

Blocs-notes

Git repositories

Configurations de la sécurité

Bloquer l'accès public

Sous-réseaux VPC

Événements

EMR on EKS

Virtual clusters

Blocs-notes

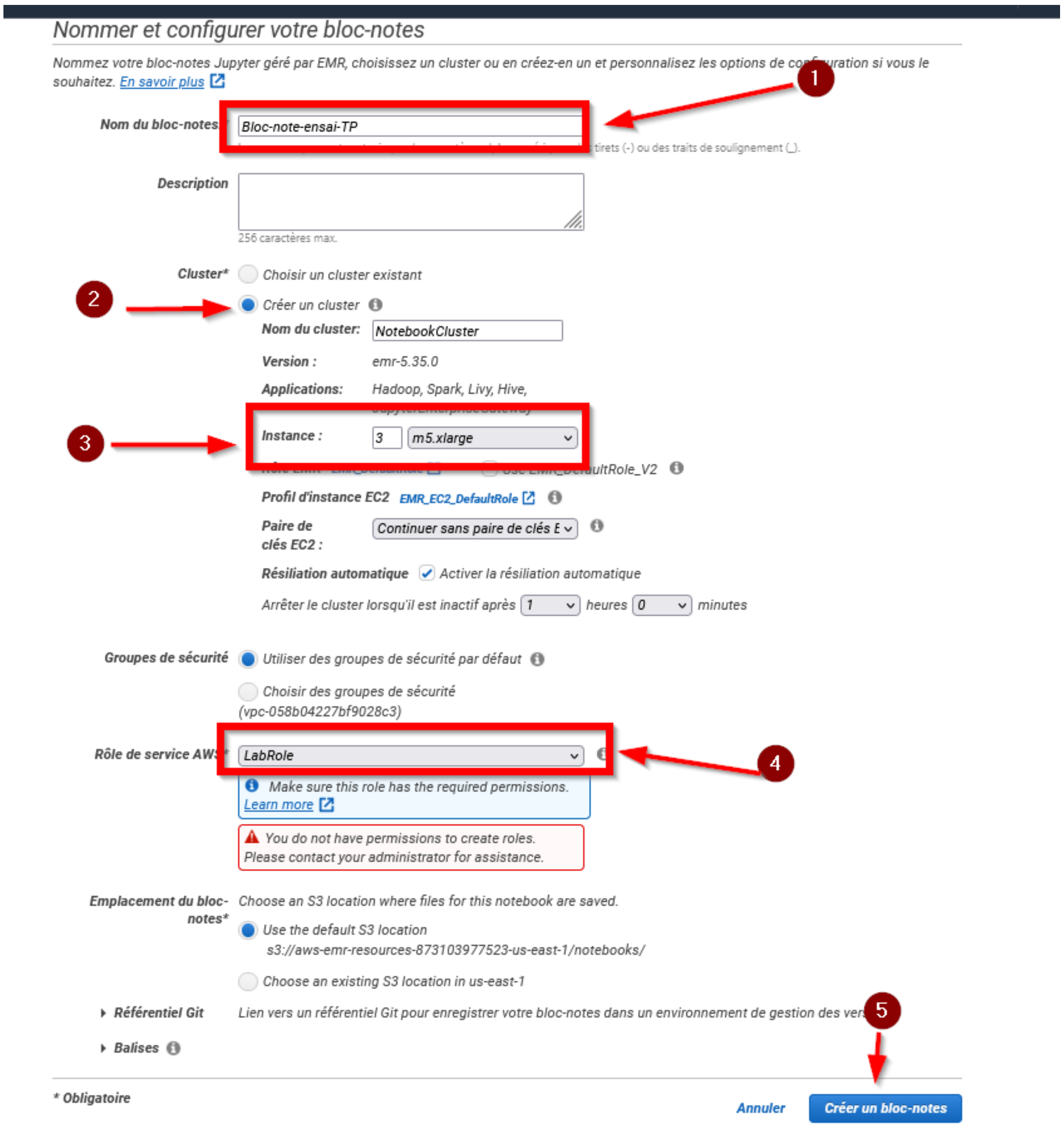
Utilisez des blocs-notes Jupyter gérés par EMR pour analyser les données de façon interactive avec du code en direct, des textes narratifs, des visualisations, et bien
Les blocs-notes sont exécutés gratuitement et sont enregistrés dans Amazon S3 indépendamment des clusters. La facturation standard pour les clusters et Amazon

Créer un bloc-notes · Afficher les détails · Ouvrir dans JupyterLab · Ouvrir dans Jupyter · Démarrage de · Arrêter · Supprimer

Filtre : Tous les blocs-notes · Filtrer les blocs-notes... · 1 bloc-notes (tout chargé)

Nom	Statut	Clu
test	Arrêté	j-1H

- [ ] You notebook configuration should be
    - [ ] `Nom du bloc-note` : a simple name like "Bloc-note-ensai-TP"
    - [ ] Choose `Créer un cluster`
    - [ ] `Instance` : between 3 and 5, and for the type use this wheel https://pickerwheel.com/pw?id=Bkz8Q. Those instances are the ones you can access with a academy account.
    - [ ] `Rôle de service AWS` choose `LabRole`
    - [ ] Then click on `Créer un bloc note`


Nommer et configurer votre bloc-notes

Nommez votre bloc-notes Jupyter géré par EMR, choisissez un cluster ou en créez-en un et personnalisez les options de configuration si vous le souhaitez. En savoir plus

Nom du bloc-notes: Bloc-note-ensai-TP

Description:

256 caractères max.

Cluster*: ( ) Choisir un cluster existant [✓ Créer un cluster]

Nom du cluster: NotebookCluster

Version : emr-5.35.0

Applications: Hadoop, Spark, Livy, Hive,

Instance : 3 m5.xlarge

Profil d'instance EC2: EMR_EC2_DefaultRole

Paire de clés EC2 : Continuer sans paire de clés E

Résiliation automatique: ✓ Activer la résiliation automatique

Arrêter le cluster lorsqu'il est inactif après 1 heures 0 minutes

Groupes de sécurité: [✓ Utiliser des groupes de sécurité par défaut] ( ) Choisir des groupes de sécurité (vpc-058b04227bf9028c3)

Rôle de service AWS*: LabRole

Make sure this role has the required permissions. Learn more

You do not have permissions to create roles. Please contact your administrator for assistance.

Emplacement du bloc-notes*: Choose an S3 location where files for this notebook are saved.
[✓ Use the default S3 location] s3://aws-emr-resources-873103977523-us-east-1/notebooks/
( ) Choose an existing S3 location in us-east-1

Référentiel Git: Lien vers un référentiel Git pour enregistrer votre bloc-notes dans un environnement de gestion des ver

Balises

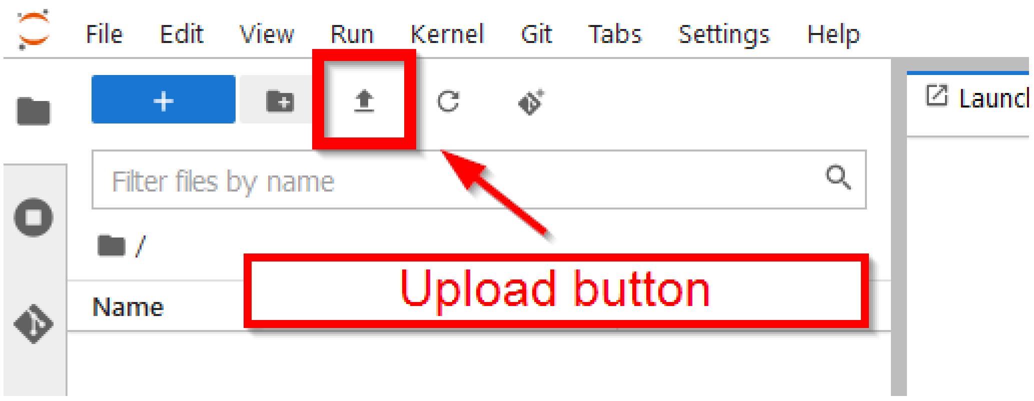
* Obligatoire

- [ ] ⏳The cluster creation takes time (between 5 and 10min), please wait and read this tutorial.

Here is a table with the hourly price of some instances just to give you an idea of the cost of an EMR cluster (hourly instance price*cluster size)

Instance	Hourly price per instance
m4.xlarge	0.24 $
m5.xlarge	0.23$
c4.xlarge	0.25$
c5.xlarge	0.22$
r4.xlarge	0.30$
c5.24xlarge	5,3$

Once your notebook is ready click on `Ouvrir dans JupyterLab`. This will open JupyterLab. Download the notebook available on moodle and upload it

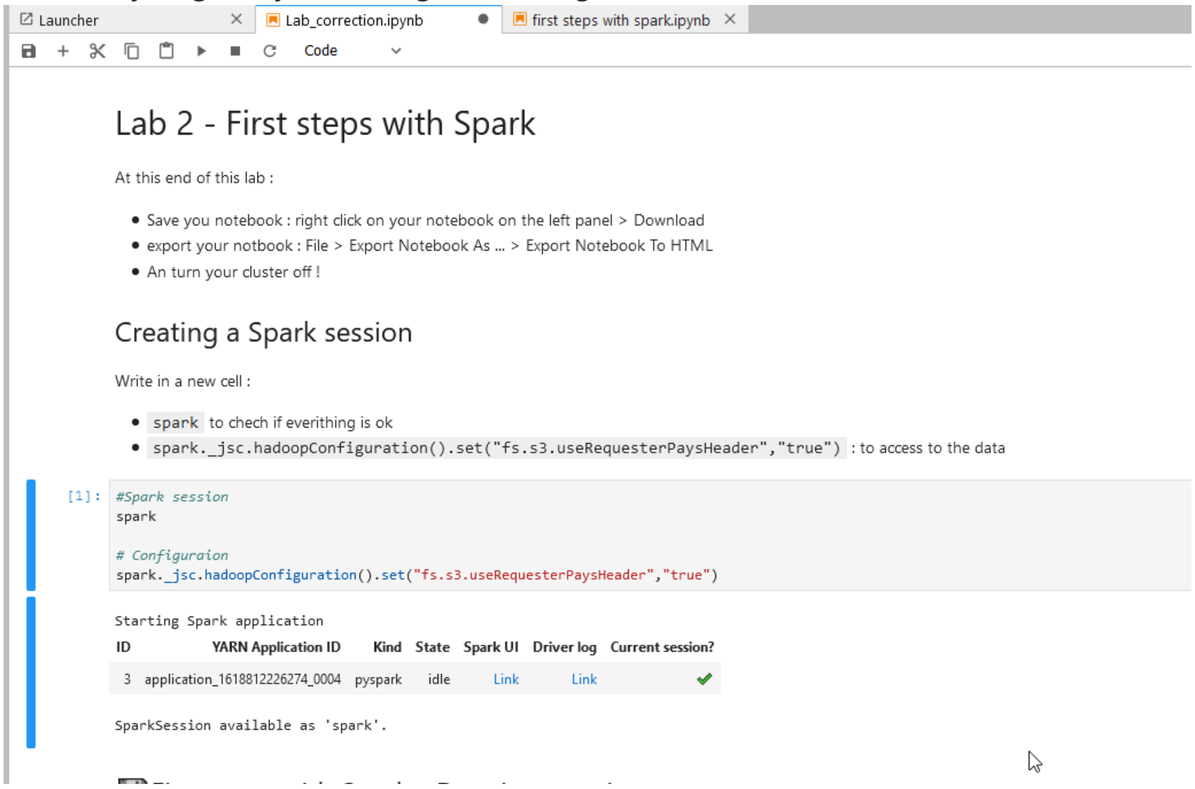


Then on the first call input :

```
#Spark session
spark

# Configuraion
spark._jsc.hadoopConfiguration().set("fs.s3.useRequesterPaysHeader","true")
```

If everything is ok you should get something like this after one or two minutes.



Lab 2 - First steps with Spark

At this end of this lab :

- Save you notebook : right click on your notebook on the left panel > Download
- export your notbook : File > Export Notebook As ... > Export Notebook To HTML
- An turn your cluster off !

Creating a Spark session

Write in a new cell :

- `spark` to chech if everithing is ok
- `spark._jsc.hadoopConfiguration().set("fs.s3.useRequesterPaysHeader","true")` : to access to the data

```
[1]: #Spark session
     spark

     # Configuraion
     spark._jsc.hadoopConfiguration().set("fs.s3.useRequesterPaysHeader","true")
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
3	application_1618812226274_0004	pyspark	idle	Link	Link	✓

```
SparkSession available as 'spark'.
```

If not, just check if your cluster is `En attente`. If not, just wait, if so, ask for help.

DO NOT FORGET TO TURN YOUR CLUSTER OFF A THE END OF THIS TUTORIAL!

💾First steps with Spark - Data importation

Spark's main object class is the **DataFrame**, which is a distributed table. It is analogous to R's or Python (Pandas)'s data frames: one row represents an observation, one column represents a variable. But contrary to R or Python, Spark's DataFrames can be distributed over hundreds of nodes.

Spark support multiple data formats, and multiple ways to load them.

- data format : csv, json, parquet (an open source column oriented format)

- can read archive files
- schema detection or user defined schema. For static data, like a json file, schema detection can be use with good results.

Spark has multiple syntaxes to import data. Some are simple with no customisation, others are more complexes but you can specify options.

The simplest syntaxes to load a json or a csv file are :

```
# JSON
json_df = spark.read.json([location of the file])
# csv
csv_df = spark.read.csv([location of the file])

```

In the future, you may consult the Data Source documentation to have the complete description of Spark's reading abilities.

The data you will use in this lab are real data from the twitter sampled stream API and filtered stream API. The tweets folder contains more than 20 files and more than 2 million tweets (more than 2Go of raw data). The tweets was collected between the 22/03/2021 and the 18/04/2021. The total collection time was less than 10 hours.

✍️Hands-on 1 - Data importation

- Load the json file store here : `s3://spark-lab-input-data-ensai20212022/tweets/tweets20220324-155940.jsonl.gz` and name you DataFrame `df_tweet`

  ⚙️ This file is an a `JSONL` (JSON-line) format, which means that each line of it is a JSON object. A JSON object is just a Python dictionary or a JavaScript object and looks like this: `{ key1: value1, key2: ["array", "of", "many values]}`). This file has been compressed into a `GZ` archive, hence the `.jsonl.gz` ending. Also this file is not magically appearing in your S3 storage. It is hosted on one of your teacher's bucket and has been made public, so that you can access it.

- It's possible to load multiple file in a unique DataFrame. It's useful when you have daily files and want to process them all. It's the same syntax as the previous one, just specify a folder. Like `s3n://spark-lab-input-data-ensai20212022/tweets/`. Name you DataFrame `df_tweet_big`

Now you have two DataFrames 🎉.

Remember that **Spark is lazy**, in the sense that it will avoid at all cost to perform unnecessary operations and wait to the last moment for performing only the duly requested computations. (Maybe you remember that R is lazy in that sense, but Spark is one degree more lazy than R.)

- Knowing that, do you think that when you run `spark.read.json()`, the data is actually migrated from S3 to the cluster ? If you want some data to be actually loaded, you can use the `show(n)` method (omitting `n` defaults to 20).
- Each time you will transform this DataFrame, the data will be transferred to your cluster. To avoid that, you need to cache your two DataFrame with the `cache()` method.

Sparks has very loose constraints on what you can actually store in a DataFrame column. The objects we just imported are actually quite messy.

- Use the `printSchema()` method to see the structure of one object.

Spark's DataFrames are immutable: there is no method to alter one specific value once one is created. This on purpose: mutations are famously hard to track, and Spark want to track them in order to avoid unnecessary computations. Suppressing mutations is actually the the best way to track changes.

Also, **DataFrames are distributed over the cluster**: they are split into blocks, ill-named **partitions**[1], that are stored separately in the memory of the workers nodes. Since Spark is lazy evaluation, all reading and intermediary computation is only kept in memory as your data are being processed.

🥉 Data frame basic manipulations

If DataFrames are immutable, they can however be ***transformed*** in other DataFrames, in the sense that a modified copy is returned. Such **transformations** include: filtering, sampling, dropping columns, selecting columns, adding new columns...

First, you can get information about the columns with:

```
df.columns       # get the column names
df.schema        # get the column names and their respective type
df.printSchema() # same, but human-readable
```

You can select columns with the `select()` method. It takes as argument a list of column name. For example :

```
df_with_less_columns = df\
  .select("variable3","variable_four","variable-6")

# Yes, you do need the ugly \ at the end of the line,
# if you want to chain methods between lines in Python
```

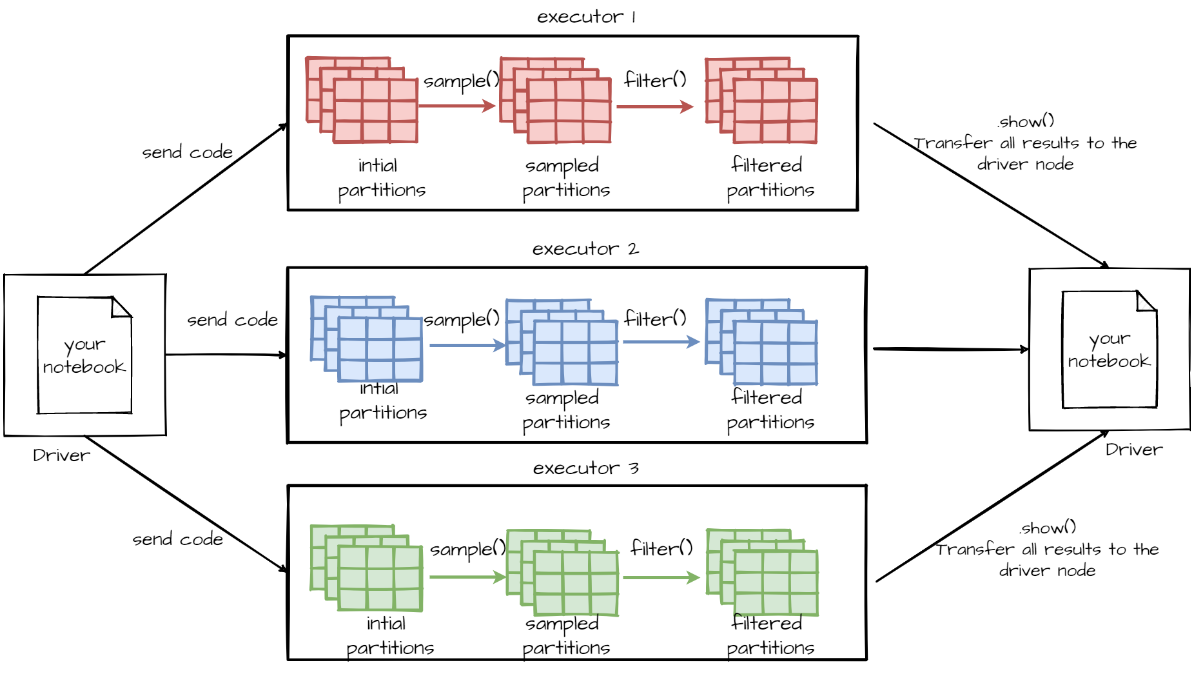
You can get nested columns easily with :

```
df.select("parentField.nestedField")
```

To filter data you could use the `filter()` method. It take as input an expression that gets evaluated for each observation and should return a boolean. Sampling is performed with the `sample()` method. For example :

```
df_with_less_rows = df\
  .sample(fraction=0.001)\
  .filter(df.variable1=="value")\
  .show(10)
```

As said before your data are distributed over multiple nodes (executors) and data inside a node are split into partitions. Then each transformations will be run in parallel. They are called *narrow transformation* For example, to sample a DataFrame, Spark sample every partitions in parallel because sample all partition produce the sample DataFrame. For some transformations, like `groupBy()` it's impossible, and it's cannot be run in parallel.



All partition can be processed in parallel

😴Lazy evaluation

This is because Spark has what is known as **lazy evaluation**, in the sense that it will wait as much as it can before performing the actual computation. Said otherwise, when you run an instruction such as:

```
tweet_author_hashtags = df_tweet_big.select("auteur","hashtags")
```

... you are not executing anything! Rather, you are building an **execution plan**, to be realised later.

Spark is quite extreme in its laziness, since only a handful of methods called **actions**, by opposition to **transformations**, will trigger an execution. The most notable are:

1. `collect()`, explicitly asking Spark to fetch the resulting rows instead of to lazily wait for more instructions,
2. `take(n)`, asking for `n` first rows
3. `first()`, an alias for `take(1)`
4. `show()` and `show(n)`, human-friendly alternatives [2]
5. `count()`, asking for the numbers of rows
6. all the "write" methods (write on file, write to database), see here for the list

This has advantages: on huge data, you don't want to accidently perform a computation that is not needed. Also, Spark can optimize each **stage** of the execution in regard to what comes next. For instance, filters will be executed as early as possible, since it diminishes the number of rows on which to perform later operations. On the contrary, joins are very computation-intense and will be executed as late as possible. The resulting **execution plan** consists in a **directed acyclic graph** (DAG) that contains the tree of all required actions for a specific computation, ordered in the most effective fashion.

This has also drawbacks. Since the computation is optimized for the end result, the intermediate stages are discarded by default. So if you need a DataFrame multiple times, you have to cache it in memory because if you don't Spark will recompute it every single time.

✍️Hands-on 2 - Data frame basic manipulations

- How many rows have your two DataFrame ?
- Sample `df_tweet_big` and keep only 10% of it. Create a new DataFrame named `df_tweet_sampled`. If computations take too long on the full DataFrame, use this one instead or add a sample transformation in your expression.
- Define a DataFrame `tweet_author_hashtags` with only the `auteur` and `hashtags` columns
- Print (few lines of) a DataFrame with only the `auteur`, `mentions`, and `urls` columns. (`mentions` and `urls` are both nested columns in `entities`.)
- Filter your first DataFrame and keep only tweets with more than 1 like. Give a name for this new, transformed DataFrame. Print (few lines of) it.

🥈Basic DataFrame column manipulation

You can add/update/rename column of a DataFrame with spark :

- Drop : `df.drop(columnName : str )`
- Rename : `df.withColumnRenamed(oldName : str, newName : str)`
- Add/update : `df.withColumn(columnName : str, columnExpression)`

For example

```
tweet_df_with_like_rt_ratio = tweet_df\
  .withColumn(        # computes new variable
    "like_rt_ratio", # like_rt_ratio "OVERCONFIDENCE"
    (tweet_df.like_count /tweet_df.retweet_count
   )

```

See here for the list of all functions available in an expression.

✍️Hands-on 3 - Basic DataFrame column manipulation

- Define a DataFrame with a column names `interaction_count`. This column is the sum of `like_count`, `reply_count` and `retweet_count`.
- Update the DataFrame you imported at the beginning of this lab and drop the `other` column

🥇Advance DataFrame column manipulation

🥽Array manipulation

Some columns often contain arrays (lists) of values instead of just one value. This may seem surprising but this actually quite natural. For instance, you may create an array of words from a text, or generate a list of random numbers for each observation, etc.

You may **create array of values** with:

- `split(text : string, delimiter : string)`, turning a text into an array of strings

You may **use array of values** with:

- `size(array : Array)`, getting the number of elements
- `array_contains(inputArray : Array, value : any)`, checking if some value appears
- `explode(array : Array)`, unnesting an array and duplicating other values. For instance it if use `explode()` over the hashtags value of this DataFrame:

Auteur	Contenu	Hashtags
Bob	I love #Spark and #bigdata	[Spark, bigdata]
Alice	Just finished #MHrise, best MH ever	[MHrise]

  I will get :

Auteur	Contenu	Hashtags	Hashtag
Bob	I love #Spark and #bigdata	[Spark, bigdata]	Spark
Bob	I love #Spark and #bigdata	[Spark, bigdata]	bigdata
Alice	Just finished #MHrise, best MH ever	[MHrise]	MHrise

All this function must be imported first :

```
from pyspark.sql.functions import split, explode, size, array_contains
```

Do not forget, to create a new column, you should use `withColumn()`. For example :

```
df.withColumn("new column", explode("array"))
```

✍Hands-on 4 - Array manipulation

- Keep all the tweets with hashtags and for each remaining line, split the hashtag text into an array of hashtags
- Create a new column with the number of words of the `contenu` column. (Use `split()` + `size()`)
- Count how many tweet contain the `Ukrraine` hashtag (use the `count()` action)

🥼User defined function

For more very specific column manipulation you will need Spark's `udf()` function (*User Defined Function*). It can be useful if you Spark does not provide a feature you want. But Spark is a popular and active project, so before coding an udf, go check the documentation. For instance for natural language processing, Spark already has some [functions](). Last things, python udf can lead to performance issues (see https://stackoverflow.com/a/38297050) and learning a little bit of scala or java can be a good idea.

For example :

```python
# !!!! DOES NOT WORK !!!!
def to_lower_case(string):
    return string.lower()

df.withColumn("tweet_lower_case", to_lower_case(df.contenu))
```

will just crash. Keep in mind that Spark is a distributed system, and that Python is only installed on the central node, as a convenience to let you execute instructions on the executor nodes. But by default, pure Python functions can only be executed where Python is installed! We need `udf()` to enable Spark to send Python instructions to the worker nodes.

Let us see how it is done :

```python
# imports
from pyspark.sql.functions import udf
from pyspark.sql.functions import explode
from pyspark.sql.types import StringType

# pure python functions
def to_lower_case(string):
    return string.lower()

# user definid function
to_lower_case_udf = udf(
    lambda x: to_lower_case(x), StringType()
) #we use a lambda function to create the udf.

# df manipulation
df_tweet\
  .select("auteur","hashtags")\
  .filter("size(hashtags)!=0")\
  .withColumn("hashtag", explode("hashtags"))\
  .withColumn("hashtag", to_lower_case_udf("hashtag")).show(10)
```

✍️Hands-on 5 - User defined function

- Create an user defined function that counts how many words a tweet contains. (your function will return an `IntegerType` and not a `StringType`)

🔩Aggregation functions

Spark offer a variety of aggregation functions :

- `count(column : string)` will count every not null value of the specify column. You cant use `count(1)` of `count("*")` to count every line (even row with only null values)
- `countDisctinct(column : string)` and `approx_count_distinct(column : string, percent_error: float)`. If the exact number is irrelevant, `approx_count_distinct()` should be preferred.

  Counting distinct elements cannot be done in parallel, and need a lot data transfer. But if you only need an approximation, there is a algorithm, named hyper-log-log (more info here) that can be parallelized.

```python
from pyspark.sql.functions import count, countDistinct, approx_count_distinct

df.select(count("col1")).show()
df.select(countDistinct("col1")).show()
df.select(approx_count_distinct("col1"), 0.1).show()
```

- You have access to all other common functions `min()`, `max()`, `first()`, `last()`, `sum()`, `sumDistinct()`, `avg()` etc (you should import them first `from pyspark.sql.functions import min, max, avg, first, last, sum, sumDistinct`)

✍Hands-on 6 - Aggregation functions

- What are the min, max, average of `interaction_count`
- How many tweets have hashtags ? Distinct hashtags ? Try the approximative count with 0.1 and 0.01as maximum estimation error allowed.

🧲Grouping functions

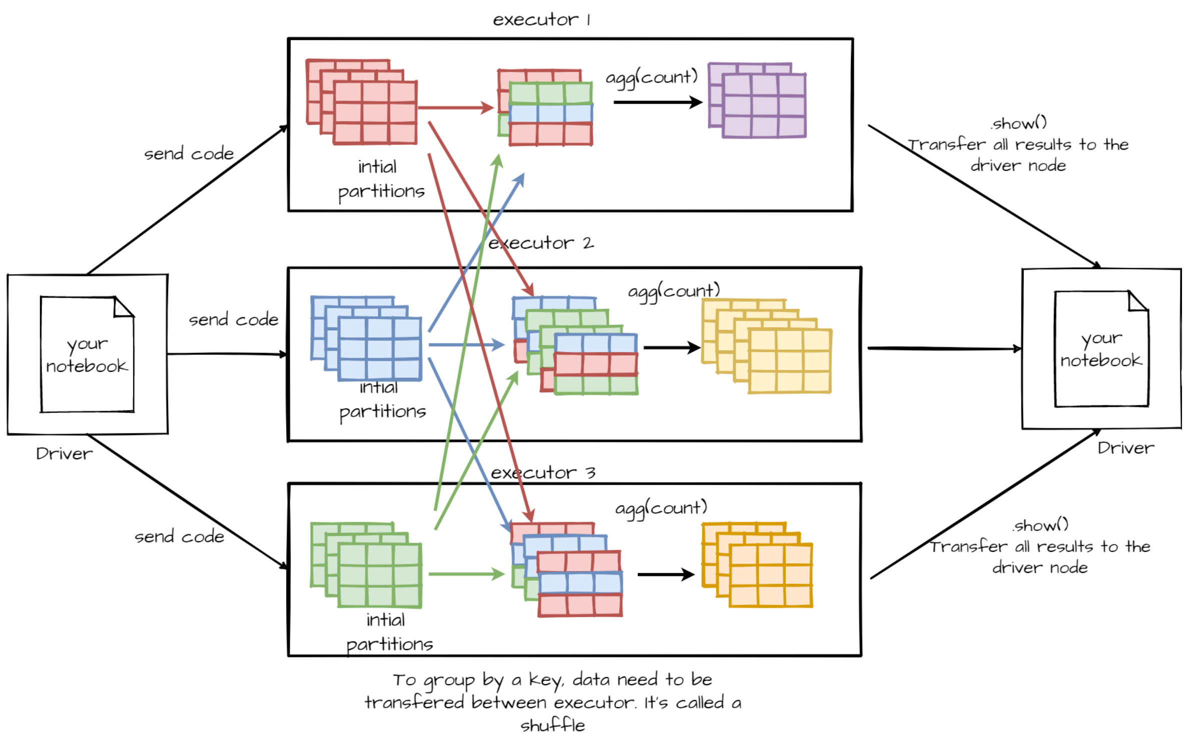
Like SQL you can group row by a criteria with Spark. Just use the `groupBy(column : string)` method. Then you can compute some aggregation over those groups.

```python
df.groupBy("col1").agg(
  count("col2").alias("quantity") # alias is use to specify the name of the new column
).show()
```

The `agg()` method can take multiples argument to compute multiple aggregation at once.

```python
df.groupBy("col1").agg(
    count("col2").alias("quantity"), min("col2").alias("min"), avg("col3").alias("avg3") ).show()
```

Aggregation and grouping transformations work differently than the previous method like `filter()`, `select()`, `withColumn()` etc. Those transformations cannot be run over each partitions in parallel, and need to transfer data between partitions and executors. They are called "wide transformations"



✍️Hands-on 7 - Grouping functions

- Compute a daframe with the min, max and average retweet of each `auteur`. Then order it by the max number of retweet in descending order by . To do that you can use the following syntax

```
from pyspark.sql.functions import desc
df.orderBy(desc("col"))
```

🔌Spark SQL

Spark understand SQL statement. It's not a hack nor a workaround to use SQL in Spark, it's one a the more powerful feature in Spark. To use SQL in you need :

1. Register a view pointing to your DataFrame. In SQL statement you will refer to your DataFrame with its view name

```
my_df.createOrReplaceTempView(viewName : str)
```

2. Use the sql function

```
spark.sql("""
You sql statment
""")
```

You could manipulate every registered DataFrame by their view name with plain SQL.

For instance

```
df_tweet.createOrReplaceTempView("small_tweet_df")
spark.sql("""
SELECT *
FROM small_tweet_df
""")
```

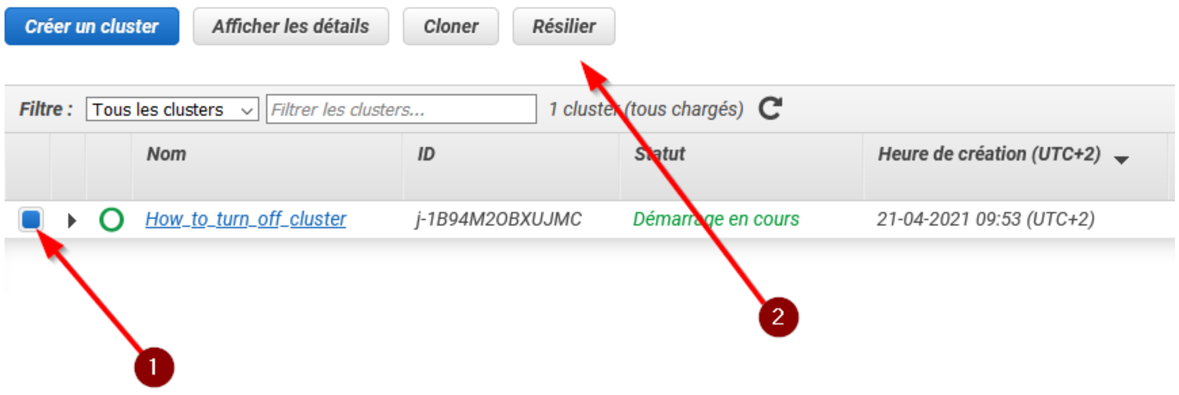
In fact you can do most of this tutorial without any knowledge in PySpark nor Spark. Lot of things can be done in Sparkk only by only knowing SQL and how to use it in Spark.

✍Hands-on 8 - Spark SQL

- How many tweets have hashtags ? Distinct hashtags ?
- Compute a DataFrame with the min, max and average retweet of each `auteur` using Spark SQL

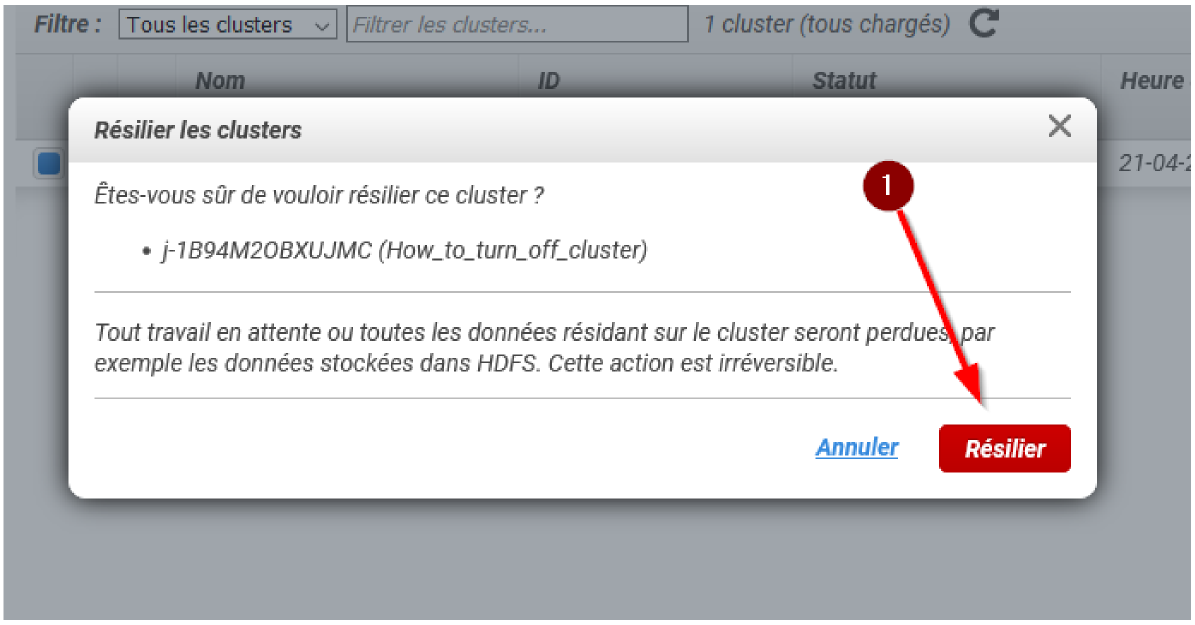
Turn off your cluster

To turn off your cluster go to the EMR page. Then select your cluster and click on `Résilier`



			Nom	ID	Statut	Heure de création (UTC+2)
			How_to_turn_off_cluster	j-1B94M2OBXUJMC	Démarrage en cours	21-04-2021 09:53 (UTC+2)

A dialog box will pop up and select one again `Résilier`



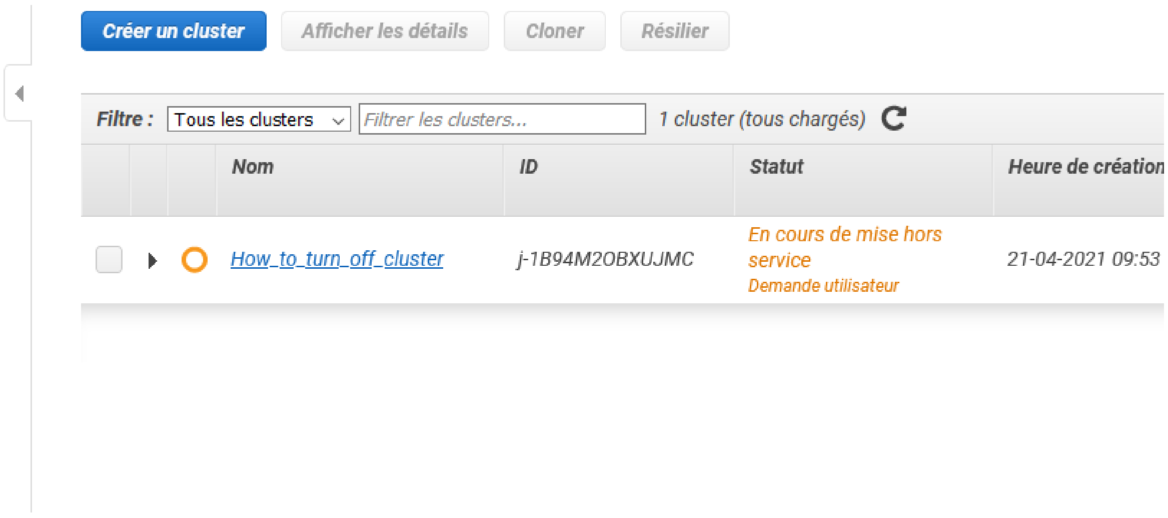
Résilier les clusters

Êtes-vous sûr de vouloir résilier ce cluster ?

- j-1B94M2OBXUJMC (How_to_turn_off_cluster)

Tout travail en attente ou toutes les données résidant sur le cluster seront perdues, par exemple les données stockées dans HDFS. Cette action est irréversible.

Then your cluster will be shutting down.



Créer un cluster | Afficher les détails | Cloner | Résilier

Filtre : Tous les clusters | Filtrer les clusters... | 1 cluster (tous chargés)

			Nom	ID	Statut	Heure de création
			How_to_turn_off_cluster	j-1B94M2OBXUJMC	En cours de mise hors service Demande utilisateur	21-04-2021 09:53

DO NOT FORGET TO TURN YOUR CLUSTER OFF!

---

1. In mathematics and data science, the "partition" of set E is usually any collection of subsets whose union makes E and whose 2-by-2 intersections are empty. But in Spark a "partition" refers to **one** block, not the set of blocks. And even if we consider the set, when replication is enforced, intersections between blocks are not necessarily empty. However, the union of all the blocks do produce the full original set. ↩

2. `first()` is exactly `take(1)` (ref) and show prints the result instead of returning it as a list of rows (ref) ↩