

Exo4 partie 1

Démarche adoptée

On a débuté par chargé les données depuis un fichier CSV en utilisant `spark.read.csv`, en prenant soin d'activer les options `header` pour interpréter correctement la première ligne comme les noms de colonne, et `inferSchema` pour que Spark déduise automatiquement les types de données des différentes colonnes.

Maintenant voici la démarche que nous avons suivi pour comparer la performance entre une UDF Python et une UDF Scala effectuer cette comparaison de manière précise :

1. Préparation des Données

Les deux scripts utilisent exactement le même jeu de données pour garantir une comparaison équitable. La taille du jeu de données est suffisamment grande pour que les différences de performances soient perceptibles.

2. Mesure du Temps d'Exécution

Pour chaque script (Python UDF et Scala UDF), nous voulons mesurer le temps d'exécution de la partie critique du code, c'est-à-dire, l'application de l'UDF et toute opération de traitement de données subséquente jusqu'à la génération du résultat final.

Méthodes de Mesure :

- Utilisation de `.show()` : Nous n'avons vu aucune différence d'affichage entre les deux scripts car `show()` sur un petit nombre de lignes (par défaut 20) ne force pas Spark à traiter tout le DataFrame. Ce n'est donc pas une bonne mesure de performance.

```
(base) jobordeau@LAPTOP-GT5DD50V:~/spark/spark-handson/src/fr/hymaia/exo4$ poetry run scala_udf
The currently activated Python version 3.11.4 is not supported by the project (~3.10.12).
Trying to find and use a compatible version.
Using python3.10 (3.10.12)
24/03/05 21:52:56 WARN Utils: Your hostname, LAPTOP-GT5DD50V resolves to a loopback address: 127.0.1.1; using 172.30.165.220 instead (on interface eth0)
24/03/05 21:52:56 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
24/03/05 21:52:58 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).
```

id	date	category	price	category_name
2500000	2020-07-06	7	23.0	furniture
2500001	2015-06-10	6	91.0	furniture
2500002	2020-04-17	13	53.0	furniture
2500003	2021-04-03	4	60.0	food
2500004	2018-02-16	13	36.0	furniture
2500005	2019-07-27	14	92.0	furniture
2500006	2020-07-01	4	81.0	food
2500007	2021-05-06	11	79.0	furniture
2500008	2018-02-16	18	0.0	furniture
2500009	2019-10-02	1	17.0	food
2500010	2017-12-27	8	52.0	furniture
2500011	2014-12-08	0	44.0	food
2500012	2013-09-27	16	69.0	furniture
2500013	2015-08-25	19	77.0	furniture
2500014	2017-07-20	0	44.0	food
2500015	2019-12-27	12	63.0	furniture
2500016	2019-07-21	7	27.0	furniture
2500017	2013-03-02	9	5.0	furniture
2500018	2012-03-15	1	37.0	food
2500019	2018-07-21	1	78.0	food

only showing top 20 rows

- Utilisation de `.collect()` : Bien que `.collect()` force Spark à traiter toutes les données, on obtient une erreur qui indique un dépassement de la mémoire à cause du trop grand jeu de données. La mémoire en ram est insuffisante.

```
(base) jobordeau@LAPTOP-GT5DD50V:~/spark/spark-handson/src/fr/hymaia/exo4$ poetry run python_udf
The currently activated Python version 3.11.4 is not supported by the project (~3.10.12).
Trying to find and use a compatible version.
Using python3.10 (3.10.12)
24/03/05 20:53:56 WARN Utils: Your hostname, LAPTOP-GT5DD50V resolves to a loopback address: 127.0.1.1; using 172.30.165.220 instead (on interface eth0)
24/03/05 20:53:56 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
```

```
24/03/05 20:53:57 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
24/03/05 20:56:37 ERROR Executor: Exception in task 15.0 in stage 3.0 (TID 90)0]
java.lang.OutOfMemoryError: Java heap space
    at java.nio.HeapByteBuffer.<init>(HeapByteBuffer.java:57)
    at java.nio.ByteBuffer.allocate(ByteBuffer.java:335)
    at org.apache.spark.serializer.SerializerHelper$.anonfun$serializeToChunkedBuffer$1(SerializerHelper.scala:40)
    at org.apache.spark.serializer.SerializerHelper$.anonfun$serializeToChunkedBuffer$1$adapted(SerializerHelper.scala:40)
    at org.apache.spark.serializer.SerializerHelper$$Lambda$1759/1262342586.apply(Unknown Source)
```

- Utilisation de `count()` après un `groupBy()` : Une solution qui fonctionne consiste à appliquer un ``groupBy()`` suivi d'une opération d'agrégation comme `count()` sur le résultat de l'UDF. Cela force Spark à traiter toutes les données sans risque de dépassement de mémoire.

- Écriture dans un fichier: écrire le résultat dans un fichier par exemple, en utilisant `write()` est une excellente manière de mesurer le temps d'exécution, car cela inclut le temps de traitement complet de toutes les données.

Nous avons opté pour utiliser `write` car il implique une évaluation complète et réelle du DataFrame donc cela semble plus efficace pour mesurer le temps d'exécution. Mais le

désavantage est que nous aurons des résultats différents en fonction de la vitesse d'écriture de nos ordinateurs respectifs.

3. Isoler la Mesure du Temps d'UDF

Pour isoler le temps d'exécution de l'UDF de tout autre traitement, on mesure le temps avant et après l'application de l'UDF et l'écriture dans le fichier.

Implémentation :

```
start_time = time.time()

df_with_category_name = df.withColumn('category_name', category_name_udf(df['category']))
df_with_category_name.write.mode('overwrite').csv('/tmp/python_udf_output')

end_time = time.time()
print(f"Execution time with Python UDF: {end_time - start_time} seconds")
```

- Avant d'appliquer l'UDF : Enregistrez le temps de début.
- Appliquez l'UDF et forcez l'évaluation avec write.
- Immédiatement après l'évaluation: Enregistrez le temps de fin.
- Calculez la différence entre le temps de fin et le temps de début pour obtenir le temps d'exécution.

4. Comparaison des Temps d'Exécution

Nous pouvons maintenant comparer les temps d'exécution obtenus pour l'UDF Python, l'UDF Scala et Sans UDF. Cette comparaison devrait révéler laquelle des deux approches est la plus performante dans votre cas d'utilisation spécifique.

Execution time without UDF: 66.68806886672974 seconds

Execution time with Scala UDF: 72.47169733047485 seconds

Execution time with Python UDF: 93.72714614868164 seconds

72.47 - 66.8 = 5,7 sec

93.72 - 66.8 = 24,7 sec

Utiliser UDF sera toujours plus long que sans mais on observe que le temps d'exécution de scala UDF est à peu près quatre fois plus rapide que le python UDF.

Exo4 partie 2

Pour la partie window function

Après le chargement des données, on a ajouté une colonne `category_name` en utilisant la fonction `when` de Spark SQL. Cela nous a permis de classifier chaque enregistrement dans des catégories plus larges ('food' ou 'furniture'), facilitant ainsi l'analyse par catégorie.

Ensuite nous avons calculé la somme des prix par catégorie d'article et par jour. Pour cela, on a défini une spécification de fenêtre avec `Window.partitionBy('date', 'category_name')`, qui nous a permis de regrouper les données par date et par nom de catégorie. On a ensuite utilisé la fonction `sum('price').over(windowSpecDay)` pour calculer la somme désirée au sein de chaque groupe.

Dans notre premier essai, on a observé que les résultats n'étaient pas conformes à nos attentes. On a identifié que le problème venait de la manière dont on avait partitionné les données. Le résultat affiché montrait un total global plutôt qu'un total quotidien par catégorie, ce qui suggère que soit la spécification de fenêtre n'est pas appliquée correctement, soit il y a une confusion dans l'affichage des résultats

Après révision, on a ajusté la spécification de la fenêtre pour s'assurer que le calcul de la somme se faisait correctement par date et par catégorie.

```
df = spark.read.csv('../resources/exo4/sell.csv', header=True, inferSchema=True)

df_with_category_name = df.withColumn('category_name',
                                      when(col('category') < 6, 'food').otherwise('furniture'))

windowSpecDay = Window.partitionBy('date', 'category_name')

df_with_total_daily = df_with_category_name.withColumn('total_price_per_category_per_day',
                                                         sum('price').over(windowSpecDay))
```

Test unitaire :

Nous avons aussi réalisé des tests unitaires et d'intégration pour tester toutes les fonctions dans les fichiers `no_udf`, `scala_udf` et `python_udf`

Exo 5 :

1 :

On a commencé par créer un utilisateur IAM , puis nous lui avons attribué les autorisations afin de pouvoir créer des Jobs glue, de pouvoir modifier des autorisations IAM , et de pouvoir ajouter/supprimer des fichiers sur le service S3.

Nous avons également permis à l'utilisateur d'accéder à AWS Cli , et nous avons récupéré les clés permettant de s'identifier sur AWS Cli.

<input type="checkbox"/>	Nom de la politique ?	Type	Attaché via ?
<input type="checkbox"/>	Glue_access	Client en ligne	En ligne
<input type="checkbox"/>	IAM_Access	Client en ligne	En ligne
<input type="checkbox"/>	S3TotalAccess	Client en ligne	En ligne

2 :

On a ensuite créé un bucket S3 qu'on a nommé bucketsparkexam vi l'interface Web.

bucketsparkexam [Info](#)

Objets

Propriétés

Autorisations

Métriques

Gestion

Points d'accès

Objets (4) [Info](#)

Copier l'URI S3

Copier l'URL

Télécharger

Ouvrir [?](#)

Supprimer

Actions ▼

Créer un dossier

Charger

Les objets sont les entités fondamentales stockées dans Amazon S3. Vous pouvez utiliser [l'inventaire Amazon S3](#) pour obtenir une liste de tous les objets de votre compartiment. Pour que d'autres personnes puissent accéder à vos objets, vous devez leur accorder explicitement des autorisations. [En savoir plus](#)

Rechercher des objets en fonction du préfixe

< 1 > [?](#)

<input type="checkbox"/>	Nom	Type	Dernière modification	Taille	Classe de stockage
<input type="checkbox"/>	Data/	Dossier	-	-	-
<input type="checkbox"/>	Job/	Dossier	-	-	-
<input type="checkbox"/>	Output/	Dossier	-	-	-
<input type="checkbox"/>	wheel/	Dossier	-	-	-

3 :

Nous avons créé un dossier Infra, et initialisé le répertoire et nous nous sommes aussi loggés en ligne de commande avec aws configure.

```
PS C:\Users\erwan\Desktop\Rendu_Spark\terraform_1.7.4_windows_amd64\Infra> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.39.1...
- Installed hashicorp/aws v5.39.1 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.



Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

4 :

Nous avons utilisé la commande poetry build afin de récupérer le fichier .whl

 spark_handson-0.1.0.tar.gz	06/03/2024 00:21	Dossier d'archive c...	4 Ko
 spark_handson-0.1.0-py3-none-any.whl	06/03/2024 00:21	Fichier WHL	7 Ko

5 :

Nous avons upload les fichiers dont nous aurons besoin pour la configuration des fichiers tf

```
PS C:\Users\erwan\Desktop\Rendu_Spark\terraform_1.7.4_windows_amd64\Infra> aws s3 cp C:\Users\erwan\Desktop\Rendu_Spark\terraform_1.7.4_windows_amd64\spark_handson-0.1.0-py3-none-any.whl s3://bucketsparkexam/wheel/spark_handson-0.1.0-py3-none-any.whl
upload: ..\spark_handson-0.1.0-py3-none-any.whl to s3://bucketsparkexam/wheel/spark_handson-0.1.0-py3-none-any.whl
PS C:\Users\erwan\Desktop\Rendu_Spark\terraform_1.7.4_windows_amd64\Infra> aws s3 cp C:\Users\erwan\Desktop\Rendu_Spark\terraform_1.7.4_windows_amd64\city_zipcode.csv s3://bucketsparkexam/Data/city_zipcode.csv
upload: ..\city_zipcode.csv to s3://bucketsparkexam/Data/city_zipcode.csv
PS C:\Users\erwan\Desktop\Rendu_Spark\terraform_1.7.4_windows_amd64\Infra> aws s3 cp C:\Users\erwan\Desktop\Rendu_Spark\terraform_1.7.4_windows_amd64\clients_bdd.csv s3://bucketsparkexam/Data/clients_bdd.csv
upload: ..\clients_bdd.csv to s3://bucketsparkexam/Data/clients_bdd.csv
PS C:\Users\erwan\Desktop\Rendu_Spark\terraform_1.7.4_windows_amd64\Infra> aws s3 cp C:\Users\erwan\Desktop\Rendu_Spark\terraform_1.7.4_windows_amd64\exo2_glue_job.py s3://bucketsparkexam/Job/exo2_glue_job.py
upload: ..\exo2_glue_job.py to s3://bucketsparkexam/Job/exo2_glue_job.py
PS C:\Users\erwan\Desktop\Rendu_Spark\terraform_1.7.4_windows_amd64\Infra> terraform apply
```

6 :

Etant donné que nous avons créé le bucket à l'aide de l'interface web d'Aws, on a ajusté le fichier bucket.tf :

```
bucket.tf
1 data "aws_s3_bucket" "bucket" {
2   bucket = "bucketsparkexam"
3 }
```

Voici le fichier iam.tf

```
iam.tf
1 data "aws_iam_policy_document" "glue_assume_role_policy" {
2   statement {
3     actions = ["sts:AssumeRole"]
4   }
5   principals {
6     type       = "Service"
7     identifiers = ["glue.amazonaws.com"]
8   }
9 }
10
11 data "aws_iam_policy_document" "glue_role_policy" {
12   statement {
13     actions = [
14       "s3:*",
15       "kms:*"
16     ]
17     effect = "Allow"
18     resources = ["*"]
19   }
20 }
21
22 resource "aws_iam_policy" "glue_role_policy" {
23   name = "glue_policy"
24   path = "/"
25   policy = data.aws_iam_policy_document.glue_role_policy.json
26 }
27
28 resource "aws_iam_role" "glue" {
29   name = "glue_role"
30   assume_role_policy = data.aws_iam_policy_document.glue_assume_role_policy.json
31   managed_policy_arns = [aws_iam_policy.glue_role_policy.arn]
32 }
33 }
```

Ainsi que glue_job.tf

```

1 resource "aws_glue_job" "example" {
2   name = "example"
3   role_arn = aws_iam_role.glue.arn
4
5   command {
6     script_location = "s3://bucketsparkexam/Job/exo2_glue_job.py"
7   }
8
9   glue_version = "3.0"
10  number_of_workers = 2
11  worker_type = "Standard"
12
13  default_arguments = {
14    "--additional-python-modules" = "s3://bucketsparkexam/wheel/spark_handson-0.1.0-py3-none-any.whl",
15    "--python-modules-installer-option" = "--upgrade",
16    "--job-language" = "python",
17    "--PARAM_1" = "VALUE_1",
18    "--PARAM_2" = "VALUE_2"
19  }
20  tags = local.tags
21 }
22

```

Voilà ce que ça donne :

```

Terraform will perform the following actions:

# aws_glue_job.example will be created
+ resource "aws_glue_job" "example" {
+   arn = (known after apply)
+   default_arguments = {
+     "--PARAM_1" = "VALUE_1"
+     "--PARAM_2" = "VALUE_2"
+     "--additional-python-modules" = "s3://bucketsparkexam/wheel/spark_handson-0.1.0-py3-none-any.whl"
+     "--job-language" = "python"
+     "--python-modules-installer-option" = "--upgrade"
+   }
+   glue_version = "3.0"
+   id = (known after apply)
+   max_capacity = (known after apply)
+   name = "example"
+   number_of_workers = 2
+   role_arn = (known after apply)
+   tags = {
+     "Environment" = "Development"
+     "Project" = "Spark Job"
+   }
+   tags_all = {
+     "Environment" = "Development"
+     "Project" = "Spark Job"
+   }
+   timeout = (known after apply)
+   worker_type = "Standard"
+   command {
+     name = "gluectl"
+     python_version = (known after apply)
+     runtime = (known after apply)
+     script_location = "s3://bucketsparkexam/Job/exo2_glue_job.py"
+   }
+ }

# aws_iam_policy.glue_role_policy will be created
+ resource "aws_iam_policy" "glue_role_policy" {
+   arn = (known after apply)
+   id = (known after apply)
+   name = "glue_policy"
+   name_prefix = (known after apply)
+   path = "/"
+   policy = jsonencode(
+     {
+       Statement = [
+         {
+           Action = [
+             "s3:*"
+           ]
+           Effect = "Allow"
+           Resource = "*"
+         }
+       ],
+       Version = "2012-10-17"
+     }
+   )
+   policy_id = (known after apply)
+   tags_all = (known after apply)
+ }

# aws_iam_role.glue will be created
+ resource "aws_iam_role" "glue" {
+   arn = (known after apply)

```



```

# aws_iam_policy.glue_role_policy will be created
+ resource "aws_iam_policy" "glue_role_policy" {
+   arn          = (known after apply)
+   id           = (known after apply)
+   name         = "glue_policy"
+   name_prefix  = (known after apply)
+   path         = "/"
+   policy       = jsonencode(
+     {
+       + Statement = [
+         + {
+           + Action = [
+             + "s3:*",
+             + "kms:*",
+           ]
+           + Effect = "Allow"
+           + Resource = "*"
+         },
+       ]
+     }
+   )
+   Version = "2012-10-17"
+ }
+ policy_id = (known after apply)
+ tags_all  = (known after apply)
}

# aws_iam_role.glue will be created
+ resource "aws_iam_role" "glue" {
+   arn              = (known after apply)
+   assume_role_policy = jsonencode(
+     {
+       + Statement = [
+         + {
+           + Action = "sts:AssumeRole"
+           + Effect = "Allow"
+           + Principal = {
+             + Service = "glue.amazonaws.com"
+           }
+         },
+       ]
+     }
+   )
+   Version = "2012-10-17"
+ }
+ create_date      = (known after apply)
+ force_detach_policies = false
+ id               = (known after apply)
+ managed_policy_arns = (known after apply)
+ max_session_duration = 3600
+ name             = "glue_role"
+ name_prefix      = (known after apply)
+ path             = "/"
+ tags_all         = (known after apply)
+ unique_id        = (known after apply)
}

```

7 :

Tout est prêt, le job a été créé :

The screenshot shows the AWS Glue Studio console. On the left is a navigation menu with options like 'Getting started', 'ETL jobs', 'Data Catalog tables', 'Data connections', 'Workflows (orchestration)', 'Data Catalog', 'Data Integration and ETL', and 'Legacy pages'. The main area is titled 'AWS Glue Studio' and includes a 'Create job' section with three options: 'Visual ETL' (highlighted in orange), 'Notebook', and 'Script editor'. Below this is a section for 'Your jobs (1)' which contains a table with one job listed.

<input type="checkbox"/>	Job name	Type	Last modified	AWS Glue version
<input type="checkbox"/>	example	Glue ETL	05/03/2024 23:00:25	3.0

Basic properties [Info](#)

Name

example

Description - optional

Descriptions can be up to 2048 characters long.

IAM Role

Role assumed by the job with permission to access your data stores. Ensure that this role has permission to your Amazon S3 sources, targets, temporary directory, scripts, and any libraries used by the job.

glue_role

Type

The type of ETL job. This is set automatically based on the types of data sources you have selected.

Spark

Glue version [Info](#)

Glue 3.0 - Supports spark 3.1, Scala 2, Python 3

▼ Input arguments (5)

Key	Value
--PARAM_1	VALUE_1
--job-language	python
--PARAM_2	VALUE_2
--additional-python-modules	s3://bucketsparkexam/wheel/spark_handson-0.1.0-py3-none-any.whl
--python-modules-installer-option	--upgrade

▼ Advanced properties

Script filename

exo2_glue_job.py

Script path

S3 location of the script. Path must be in the form s3://bucket/prefix/path/. It must end with a slash (/) and not include any files.

Q s3://bucketsparkexam/Job/

×

View [↗](#)

Browse S3

Job metrics [Info](#)

8 :

Nous avons ajusté le fichier exo2_glue_job.py avant de le réupload :

```

import sys
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions
from src.fr.hymaia.exo2.spark_aggregate_job import calculate_population_by_department
from src.fr.hymaia.exo2.spark_clean_job import read_data, filter_adults, join_clients_cities, add_department

if __name__ == '__main__':
    spark = SparkSession.builder.getOrCreate()
    glueContext = GlueContext(spark.sparkContext)
    job = Job(glueContext)
    args = getResolvedOptions(sys.argv, ["JOB_NAME"])
    job.init(args['JOB_NAME'], args)

    input_clients_path = "s3://bucketsparkexam/Data/city_zipcode.csv"
    input_cities_path = "s3://bucketsparkexam/Data/clients_bdd.csv"
    output_clean_path = "s3://bucketsparkexam/Output/clean/"
    output_aggregate_path = "s3://bucketsparkexam/Output/aggregate/"

    df_clients = read_data(spark, input_clients_path)
    df_cities = read_data(spark, input_cities_path)
    df_clients_adults = filter_adults(df_clients)
    df_joined = join_clients_cities(df_clients_adults, df_cities)
    df_with_department = add_department_column(df_joined)
    df_with_department.write.parquet(output_clean_path)

```

```

df_clients = read_data(spark, input_clients_path)
df_cities = read_data(spark, input_cities_path)
df_clients_adults = filter_adults(df_clients)
df_joined = join_clients_cities(df_clients_adults, df_cities)
df_with_department = add_department_column(df_joined)
df_with_department.write.parquet(output_clean_path)

df_clean = spark.read.parquet(output_clean_path)
df_population = calculate_population_by_department(df_clean)
df_population.coalesce(1).write.option("header", "true").csv(output_aggregate_path)

job.commit()

```

```

PS C:\Users\erwan\Desktop\Rendu_Spark\terraform_1.7.4_windows_amd64\Infra> aws s3 cp C:\Users\erwan\Desktop\Rendu_Spark\terraform_1.7.4_windows_amd64\exo2_glue_job.py s3://bucketsparkexam/Job/exo2_glue_job.py
upload: ../exo2_glue_job.py to s3://bucketsparkexam/Job/exo2_glue_job.py

```

Nous avons rencontré cette erreur :

AWS Glue > Monitoring > Job run

Job Run - jr_bd832fd8f4efd8ded1de56d1aa7504435566e288c51dfea8237a05723858813f

Run details [Info](#) Rewind job bookmark ↺

jr_bd832fd8f4efd8ded1de56d1aa7504435566e288c51dfea8237a05723858813f

❌ ModuleNotFoundError: No module named 'src'

Job name	Id	Run status	Glue version
example	jr_bd832fd8f4efd8ded1de56d1aa7504435566e288c51dfea8237a05723858813f	❌ Failed	3.0
Retry attempt number	Start time (UTC)	End time (UTC)	Start-up time
Initial run	2024/03/05 22:03:32	2024/03/05 22:04:27	21 seconds
Execution time	Last modified on (UTC)	Trigger name	Security configuration

Nous avons regénéré le fichier .whl , vérifier son contenu en l’extrayant et sa structure était tout à fait normale.

Après plusieurs tentatives vaines , on a voulu se convaincre que le job fonctionnait et qu’il s’agissait d’une erreur de dépendances , ducoup nous avons upload un nouveau script :

Script [Job details](#) [Runs](#) [Data quality - updated](#) [Schedules](#) [Version Control](#)

Script [Info](#)

```
1 import sys
2 from aws glue.context import GlueContext
3 from aws glue.job import Job
4 from aws glue.utils import getResolvedOptions
5 from pyspark.sql import SparkSession
6 from pyspark.sql.functions import col
7
8 def process_clients_data(input_path, output_path):
9
10     spark = SparkSession.builder.getOrCreate()
11
12     # Lecture du dataset
13     df = spark.read.csv(input_path, header=True)
14
15     df_filtered_sorted = df.filter(col("age") > 30).orderBy(col("age"))
16
17     df_filtered_sorted.coalesce(1).write.option("header", "true").csv(output_path)
18
19     spark.stop()
20
```

```
16
17     df_filtered_sorted.coalesce(1).write.option("header", "true").csv(output_path)
18
19     spark.stop()
20
21 ▼ if __name__ == '__main__':
22
23     spark = SparkSession.builder.getOrCreate()
24     glueContext = GlueContext(spark.sparkContext)
25     job = Job(glueContext)
26     args = getResolvedOptions(sys.argv, ["JOB_NAME"])
27     job.init(args['JOB_NAME'], args)
28
29     input_path = "s3://bucketsparkexam/Data/clients_bdd.csv"
30     output_path = "s3://bucketsparkexam/Output/clients_bdd_output"
31
32     process_clients_data(input_path, output_path)
33
34     job.commit()
35
```

Et là tout fonctionne très bien :

<input type="checkbox"/>	Nom ▲	Type ▼	Dernière modification ▼	Taille ▼	Classe de stockage ▼
<input type="checkbox"/>	Data/	Dossier	-	-	-
<input type="checkbox"/>	Job/	Dossier	-	-	-
<input type="checkbox"/>	Output/	Dossier	-	-	-
<input type="checkbox"/>	wheel/	Dossier	-	-	-

<input type="checkbox"/>	Nom ▲	Type ▼	Dernière modification ▼	Taille ▼	Classe de stockage ▼
<input type="checkbox"/>	part-00000-4d9e8c6e-2389-42d7-b915-30543f2747d3-c000.csv	csv	06 Mar 2024 01:07:43 AM CET	149.3 Ko	Standard

Job runs (1/9) Info							
Last updated (UTC) March 6, 2024 at 24:52:38			<button>View details</button>	<button>Stop job run</button>	<button>Table View</button>	<button>Card View</button>	
<input type="text" value="Filter job runs by property"/>				<div>< 1 > </div>			
	Run status ▼	Retries ▼	Start time (UTC) ▼	End time (UTC) ▼	Duration ▼	Capacit... ▼	
	✔ Succeeded	0	2024/03/06 00:05:38	2024/03/06 00:07:53	1 m 13 s	2 DPU	
<input type="radio"/>	✘ Failed	0	2024/03/05 23:50:59	2024/03/05 23:51:42	37 s	2 DPU	
<input type="radio"/>	✘ Failed	0	2024/03/05 23:23:50	2024/03/05 23:24:28	31 s	2 DPU	
<input type="radio"/>	✘ Failed	0	2024/03/05 23:06:40	2024/03/05 23:07:23	35 s	2 DPU	
<input type="radio"/>	✘ Failed	0	2024/03/05 22:46:30	2024/03/05 22:47:15	38 s	2 DPU	
<input type="radio"/>	✘ Failed	0	2024/03/05 22:28:00	2024/03/05 22:28:42	35 s	2 DPU	
<input type="radio"/>	✘ Failed	0	2024/03/05 22:08:56	2024/03/05 22:09:51	48 s	2 DPU	
<input type="radio"/>	✘ Failed	0	2024/03/05 22:05:22	2024/03/05 22:06:02	33 s	2 DPU	
<input type="radio"/>	✘ Failed	0	2024/03/05 22:03:32	2024/03/05 22:04:27	33 s	2 DPU	

Conclusion :

Ces exercices nous ont beaucoup aidé à comprendre Spark, la différence entre Python et Scala et la gestion de pipelines de données. On a pu voir comment tout se connecte, de Spark à AWS Glue, en passant par les dépendances. La pratique nous a rendu tout ça bien plus clair et accessible. C'était super enrichissant. Merci !