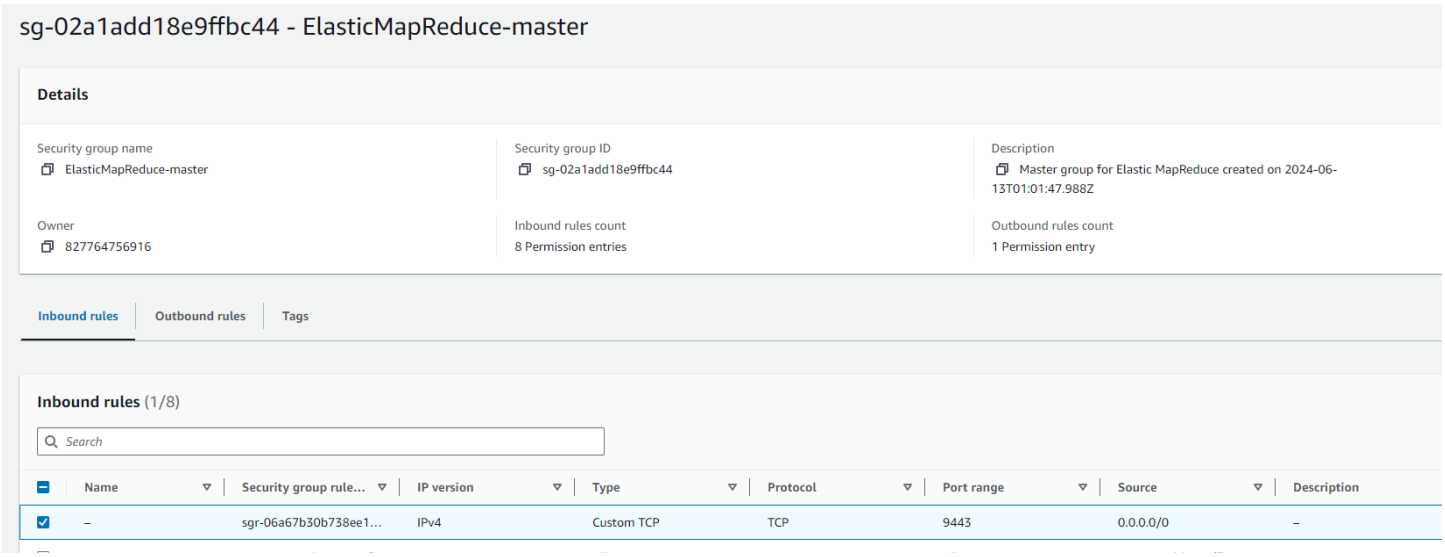
Grupo 1 – Lab5

Health Department inspection results – Jupyter: Pyspark

Script:

health_violations.py

1. Antes de dar início a atividade, precisamos fazer alterações nas configurações do EC2 para conseguir acessar nosso cluster (EC2) liberando os acessos vindo pela porta = '9443', Protocolo e Tipo = TCP Personalizado, Versão de IP = IPv4. Dessa forma, o Jupyter conseguirá acessar no cluster e realizar o processamento de dados pelo Spark.



2. Ao finalizar a configuração de rede, retornando as propriedades do cluster, acessando a tab 'aplicativos' encontramos a URL da JupyterHub. Podemos ver que no final da URL, consta o número da porta que foi liberado o acesso e garantindo que conseguimos acessar o app.
3. Ao abrir o JyputerHub, criamos um Pyspark script (em anexo no início da atividade) para calcular os seguintes:
    a. Análise de top 10 estabelecimentos com violação tipo RED
    b. Análise de top 10 cidades com violação tipo RED

Descrição do script:
- Realiza *import* do *SparkSession*
- Função: calculate_red_violations
    - Essa função recebe o valor de *data_source, output_uri*
    - Inicia o *SparkSession* e verifica se *data_source* não é vazio. Se verdadeiro, carrega os arquivos CSV.
    - Cria DataFrame na mémoria para realizar consulta. DataFrames *restaurants_df* e *city_df*
    - Cria dois novos DataFrames a partir de consultas utilizando 'spark.sql', calculando os 10 restaurantes/cidades com mais violações do tipo RED.
    - Salva os resultados das consultas nos paths:
        -- output_uri = 's3://emr-s3-bucket2024/saida/restaurant_violations_results'
        -- output_uri_city = 's3://emr-s3-bucket2024/saida/city_violations_results'

jupyterhub health_violations Last Checkpoint: 25 minutes ago (unsaved changes)

Logout | Control Panel

File Edit View Insert Cell Kernel Widgets Help

Trusted | PySpark

Run | Code

In [1]:
```
import argparse
from pyspark.sql import SparkSession

def calculate_red_violations(data_source, output_uri):
    """
    Processes sample food establishment inspection data and queries the data to find the top 10 establishments
    with the most Red violations from 2006 to 2020.

    :param data_source: The URI of your food establishment data CSV, such as 's3://DOC-EXAMPLE-BUCKET/food-establishment-data.csv
    :param output_uri: The URI where output is written, such as 's3://DOC-EXAMPLE-BUCKET/restaurant_violation_results'.
    """
    with SparkSession.builder.appName("Calculate Red Health Violations").getOrCreate() as spark:
        # Load the restaurant violation CSV data
        if data_source is not None:
            restaurants_df = spark.read.option("header", "true").csv(data_source)

        # Create an in-memory DataFrame to query
        restaurants_df.createOrReplaceTempView("restaurant_violations")

        # Create a DataFrame of the top 10 restaurants with the most Red violations
        top_red_violation_restaurants = spark.sql("""SELECT name, count(*) AS total_red_violations
          FROM restaurant_violations
          WHERE `Violation Type` = 'RED'
          GROUP BY name
          ORDER BY total_red_violations DESC LIMIT 10""")

        # Write the results to the specified output URI
        top_red_violation_restaurants.write.option("header", "true").mode("overwrite").csv(output_uri)

data_source = 's3://emr-s3-bucket2024/Food_Establishment_Inspection_Data_20240612.csv'
output_uri = 's3://emr-s3-bucket2024/saida/restaurant_violations_results'

# if __name__ == "__main__":
#     parser = argparse.ArgumentParser()
#     parser.add_argument(
#         '--data_source', help="The URI for you CSV restaurant data, like an S3 bucket location.")
#     parser.add_argument(
#         '--output_uri', help="The URI where output is saved, like an S3 bucket location.")
#     args = parser.parse_args()

calculate_red_violations(data_source,output_uri)
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	User	Current session?
3	application_1718332962171_0004	pyspark	idle	Link	Link	None	✓

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

jupyterhub health_violations Last Checkpoint: 32 minutes ago (unsaved changes)

Logout | Control Panel

File Edit View Insert Cell Kernel Widgets Help

Trusted | PySpark

Run | Code

```
from pyspark.sql import SparkSession

def calculate_red_violations(data_source, output_uri):
    """
    Processes sample food establishment inspection data and queries the data to find the top 10 establishments
    with the most Red violations from 2006 to 2020.

    :param data_source: The URI of your food establishment data CSV, such as 's3://DOC-EXAMPLE-BUCKET/food-establishment-data.csv
    :param output_uri: The URI where output is written, such as 's3://DOC-EXAMPLE-BUCKET/restaurant_violation_results'.
    """
    with SparkSession.builder.appName("Calculate Red Health Violations").getOrCreate() as spark:
        # Load the restaurant violation CSV data
        if data_source is not None:
            city_df = spark.read.option("header", "true").csv(data_source)

        # Create an in-memory DataFrame to query
        city_df.createOrReplaceTempView("city_violations")

        # Create a DataFrame of the top 10 restaurants with the most Red violations
        top_red_violation_city = spark.sql("""SELECT City, count(*) AS total_red_violations
          FROM city_violations
          WHERE `Violation Type` = 'RED'
          GROUP BY City
          ORDER BY total_red_violations DESC LIMIT 10""")

        # Write the results to the specified output URI
        top_red_violation_city.write.option("header", "true").mode("overwrite").csv(output_uri)

data_source = 's3://emr-s3-bucket2024/Food_Establishment_Inspection_Data_20240612.csv'
output_uri = 's3://emr-s3-bucket2024/saida/city_violations_results'

# if __name__ == "__main__":
#     parser = argparse.ArgumentParser()
#     parser.add_argument(
#         '--data_source', help="The URI for you CSV restaurant data, like an S3 bucket location.")
#     parser.add_argument(
#         '--output_uri', help="The URI where output is saved, like an S3 bucket location.")
#     args = parser.parse_args()

calculate_red_violations(data_source,output_uri)
```

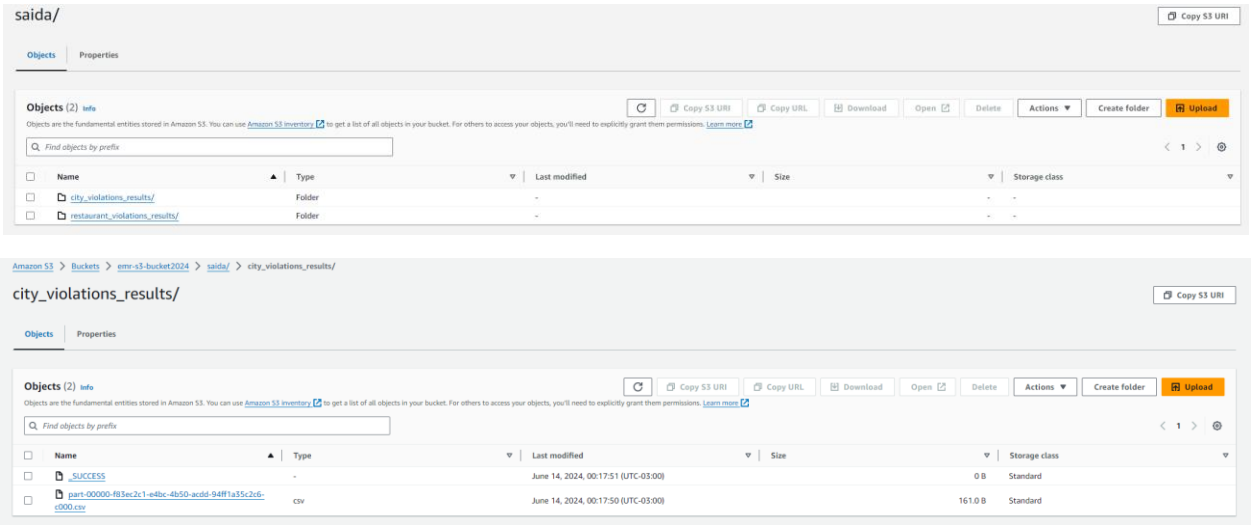
```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	User	Current session?
4	application_1718332962171_0005	pyspark	idle	Link	Link	None	✓

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

In [ ]:

4. Ao rodar script, é criado um ID e YARN App ID. Abaixo vemos a confirmação da criação que os diretórios e arquivos CSVs foram criados com os resultados das consultas.



5. Resultado de **Restaurantes** com maiores violações do tipo RED:

	A	B	C
1	name	total_red_violations	
2	T-MOBILE PARK	416	
3	SUBWAY	327	
4	WHOLE FOODS MARKET	314	
5	PCC COMMUNITY MARKETS	272	
6	TACO TIME	196	
7	SAFEWAY INC #1508	159	
8	TAQUERIA EL RINCONSITO	154	
9	PAGLIACCI PIZZA INC	145	
10	UWAJIMAYA	139	
11	SAFEWAY STORES INC #1550	136	
12			

6. Resultado de **Cidades** com maiores violações do tipo RED:

	A	B
1	City	total_red_violations
2	SEATTLE	24481
3	Seattle	13060
4	BELLEVUE	5197
5	KENT	3074
6	FEDERAL WAY	2425
7	RENTON	2364
8	REDMOND	2292
9	KIRKLAND	2264
10	Bellevue	1702
11	TUKWILA	1481
12		

CSV Files Results:

part-00000-60e42d92-e7b5-4ec7-8d8d-13be4bc-4b50-acdd-94ff

part-00000-f83ec2c1-