Instituto Federal de Educação, Ciência e Tecnologia

Câmpus Campinas

Professores: Bianca Pedrosa
Samuel Martins

Projeto Interdisciplinar

1. Especificação

O objetivo do projeto interdisciplinar é a aplicação das técnicas de Ciência de Dados aprendidas nas diferentes disciplinas do semestre, a partir de um problema apoiado por um conjunto de dados (dataset). O projeto requer o desenvolvimento de uma solução de aprendizado de máquina cujo treinamento e processamento deverão ser realizados em ambiente de nuvem (AWS ou Google), com processamento distribuído (cluster de computadores), explorando o uso de paralelismo (spark/mapreduce).

Aluno: Omar Hajime Fidelis

Data: 05/12/2021

2. Regras Gerais

- O tema do projeto é de livre escolha e deve ser levantado pelos os alunos integrantes do grupo.
- O projeto deverá ser feito em grupos de até 3 (três) pessoas.
- Ao final do semestre, os alunos devem elaborar uma apresentação do projeto, mostrando os resultados das respectivas atividades requeridas por cada uma das disciplinas cursadas.
- Todos os grupos devem apresentar o trabalho. A entrega pelo Moodle é só uma formalidade e não garante avaliação. Os alunos que não participarem da apresentação, a princípio, terão nota 0.
- Cada professor exigirá atividades relativas a conteúdos específicos de sua disciplina.
- As notas das disciplinas serão obtidas separadamente.

Orientações para elaboração do Trabalho Interdisciplinar Big Data + ML

2º semestre 2021 – Prof. Bianca

Detalhamento da parte do trabalho referente à Big Data

Processamento de algoritmos de ML em ambiente distribuído, formado por um cluster de máquinas rodando hadoop map reduce/spark. Estes ambientes podem ser encontrados na AWS e Google, entre outros.

1. **Organização do conteúdo do trabalho**

Os trabalhos devem ter os seguintes tópicos:

a) Introdução, que deve ser uma breve contextualização.
b) Descrição dos dados (fonte, período, volume, formato, amostras dos dados)
c) Workflow/pipeline de arquitetura
d) Infraestrutura (Configuração do cluster como nro de nós, tipo de máquinas, permisões)
e) Análise do Processamento (análise do desempenho das operações ETL, segundo critérios como tempo de processamento, formas intermediárias de armazenaneto dos dados (partições), comparação do desempenho com diferentes configurações de cluster)
f) Funções definidas para processamento distribuído (map, reduce, outras)
g) Conclusões (limitações e vantagens)
h) Referências

1. Introdução

1.1. Objetivo

Este projeto tem como objetivo a aplicação das técnicas de Ciência de Dados no conjunto de dados (dataset) de "Wine Quality". A escolha do dataset está relacionado a continuidade e comparação do processo de desenvolvimento do projeto do semestre anterior.

O foco principal deste projeto é o desenvolvimento de uma solução de aprendizado de máquina cujo treinamento e processamento deverão ser realizados em ambiente de nuvem (Amazon Web Service - AWS), com processamento distribuído (cluster de computadores), explorando o uso de paralelismo (spark/mapreduce).

Além do ambiente distribuído, aplicou-se técnicas de machine learning (normalização, pipeline, cross validation), e alguns algoritmos de classificação para avaliação de performance e resultados.

1.2. Descrição Dataset

A base de dados consolidada é composta por 1599 amostra de vinho tinto e 4898 amostra de vinho branco, totalizando 6497 amostra e descritas pela coluna "style". O dataset foi criado Paulo Cortez (Univ. Minho), António Cerdeira, Fernando Almeida, Telmo Matos and José Reis em 2009. Cada linha da base de dados é uma amostra de vinho com suas medições fisico-químicas, tipo do vinho (vermelho ou branco) e a nota/avaliação de especialistas.
Neste projeto, consideramos a análise para a classificação do vinho, podendo ser "tinto" ou "branco".

1.3. Detalhamento dos atributos

fixed_acidity = Acidez Fixa: ácido não volátil encontrado no vinho, que é tartárico, málico, cítrico e succínico. Todos esses ácidos são originários das uvas, com exceção do ácido succínico, que é produzido pela levedura durante o processo de fermentação

volatile_acidity = Acidez volátil: os elementos ácidos de um vinho que são gasosos, em vez de líquidos e, portanto, podem ser sentidos como um cheiro, mostrando um aroma, em vez de encontrados no palato

citric_acid = Ácido cítrico: um ácido orgânico fraco, frequentemente usado como conservante natural ou aditivo para alimentos ou bebidas para adicionar um sabor azedo e frescor aos alimentos

residual_sugar = Açúcar residual: a quantidade de açúcar remanescente após a parada da fermentação, é raro encontrar vinhos com menos de 1 grama / litro e vinhos com mais de 45 gramas / litro são considerados doces

chlorides = Cloretos: a quantidade de sal no vinho

free_sulfur_dioxide = Dióxido de Enxofre Livre (SO2): o SO2 é usado em todas as fases do processo de vinificação para prevenir a oxidação e o crescimento microbiano. Quantidades excessivas de SO2 podem inibir a fermentação e causar efeitos sensoriais indesejáveis

total_sulfur_dioxide = Dióxido de Enxofre Total: quantidade das formas livre e ligada de S02S02; em baixas concentrações, SO2SO2 é principalmente indetectável no vinho, mas em concentrações de SO2SO2 livre acima de 50 ppm, SO2SO2 torna-se evidente no nariz e no sabor do vinho

density = Densidade: a densidade é próxima à da água, dependendo da porcentagem de álcool e açúcar

pH: produtores de vinho usam o pH como forma de medir a maturação em relação à acidez. Vinhos de pH baixo terão gosto ácido e crocante, enquanto vinhos de pH mais alto são mais suscetíveis ao crescimento bacteriano. A maioria do pH do vinho está em torno de 3 ou 4

sulphates = Sulfatos: aditivo do vinho que pode contribuir para os níveis de gás de dióxido de enxofre (S02S02), que atua como um antimicrobiano e antioxidante

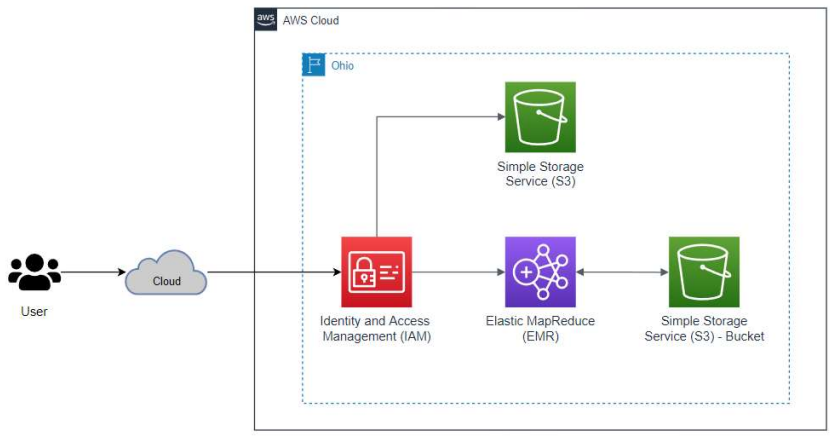
alcohol = Álcool: o teor de álcool do vinho em percentual

quality = Qualidade: notas/avaliações realizadas por no mínimo 3 especialistas em vinhos

style = Tipo: Tipo do Vino (Vermelho ou Branco)

2. Infraestrutura

2.1. Infraestrutura AWS



2.2. AWS Applications

2.2.1. Apache Spark

Apache Spark is a distributed processing framework and programming model that helps you do machine learning, stream processing, or graph analytics using Amazon EMR clusters. Similar to Apache Hadoop, Spark is an open-source, distributed processing system commonly used for big data workloads. However, Spark has several notable differences from Hadoop MapReduce. Spark has an optimized directed acyclic graph (DAG) execution engine and actively caches data in-memory, which can boost performance, especially for certain algorithms and interactive queries.

Spark version information for emr-5.33.0

Amazon EMR Release Label	Spark Version	Components Installed With Spark
emr-5.33.0	Spark 2.4.7	aws-sagemaker-spark-sdk, emrfs, emr-goodies, emr-ddb, emr-s3-select, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hadoop-yarn-timeline-server, hudi, hudi-spark, livy-server, nginx, r, spark-client, spark-history-server, spark-on-yarn, spark-yarn-slave

2.2.2. Apache Hive

Hive is an open-source, data warehouse, and analytic package that runs on top of a Hadoop cluster. Hive scripts use an SQL-like language called Hive QL (query language) that abstracts programming models and supports typical data warehouse interactions. Hive enables you to avoid the complexities of writing Tez jobs based on directed acyclic graphs (DAGs) or MapReduce programs in a lower level computer language, such as Java.

Hive version information for emr-6.4.0

Amazon EMR Release Label	Hive Version	Components Installed With Hive
emr-6.4.0	Hive 3.1.2	emrfs, emr-ddb, emr-goodies, emr-kinesis, emr-s3-dist-cp, emr-s3-select, hadoop-client, hadoop-mapred, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-httpfs-server, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hadoop-yarn-timeline-server, hive-client, hive-hbase, hcatalog-server, hive-server2, hudi, mariadb-server, tez-on-yarn, zookeeper-client, zookeeper-server

2.2.3. Apache Livy

Livy enables interaction over a REST interface with an EMR cluster running Spark. You can use the REST interface or an RPC client library to submit Spark jobs or snippets of Spark code, retrieve results synchronously or asynchronously, and manage Spark Context. For more information, see the Apache Livy website. Livy is included in Amazon EMR release version 5.9.0 and later.

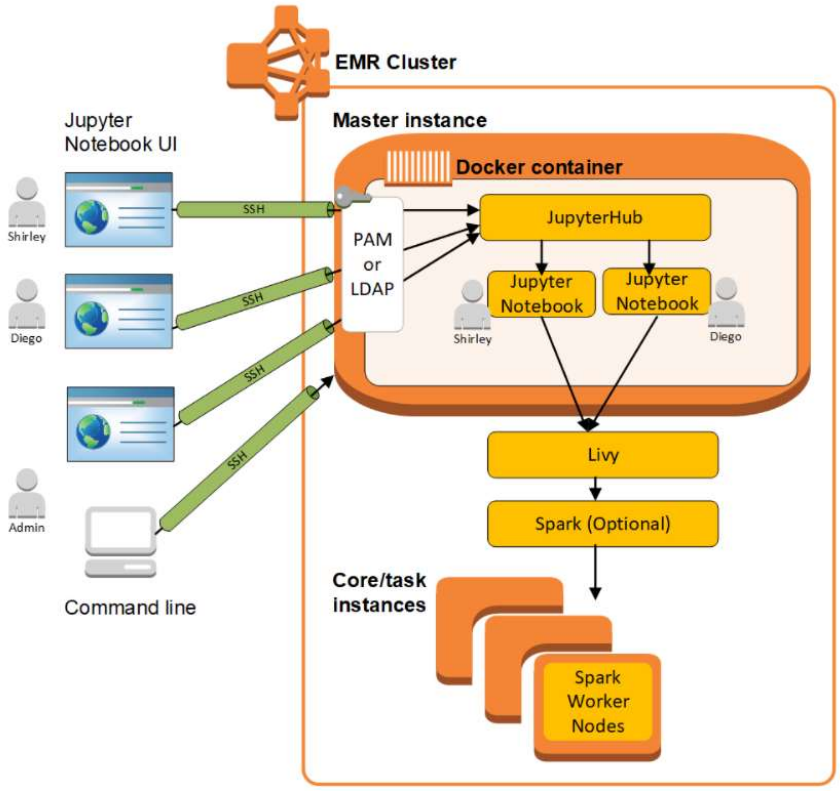
Livy version information for emr-6.4.0

Amazon EMR Release Label	Livy Version	Components Installed With Livy
emr-6.4.0	Livy 0.7.1	aws-sagemaker-spark-sdk, emrfs, emr-goodies, emr-ddb, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hadoop-yarn-timeline-server, r, spark-client, spark-history-server, spark-on-yarn, spark-yarn-slave, livy-server, nginx

2.2.4. Amazon EMR Notebook based on Jupyter Notebook

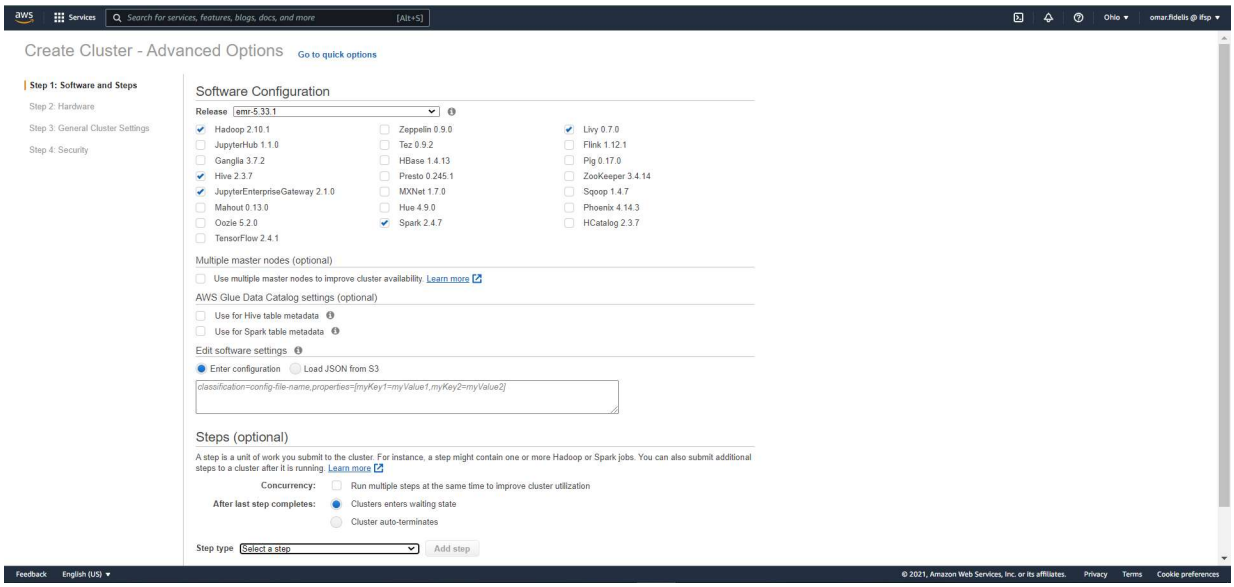
EMR Notebooks is a Jupyter Notebook environment built in to the Amazon EMR console that allows you to quickly create Jupyter notebooks, attach them to Spark clusters, and then open the Jupyter Notebook editor in the console to remotely run queries and code. An EMR notebook is saved in Amazon S3 independently from clusters for durable storage, quick access, and flexibility. You can have multiple notebooks open, attach multiple notebooks to a single cluster, and re-use a notebook on different clusters.

2.2.4. Amazon EMR Notebook Arquitetura

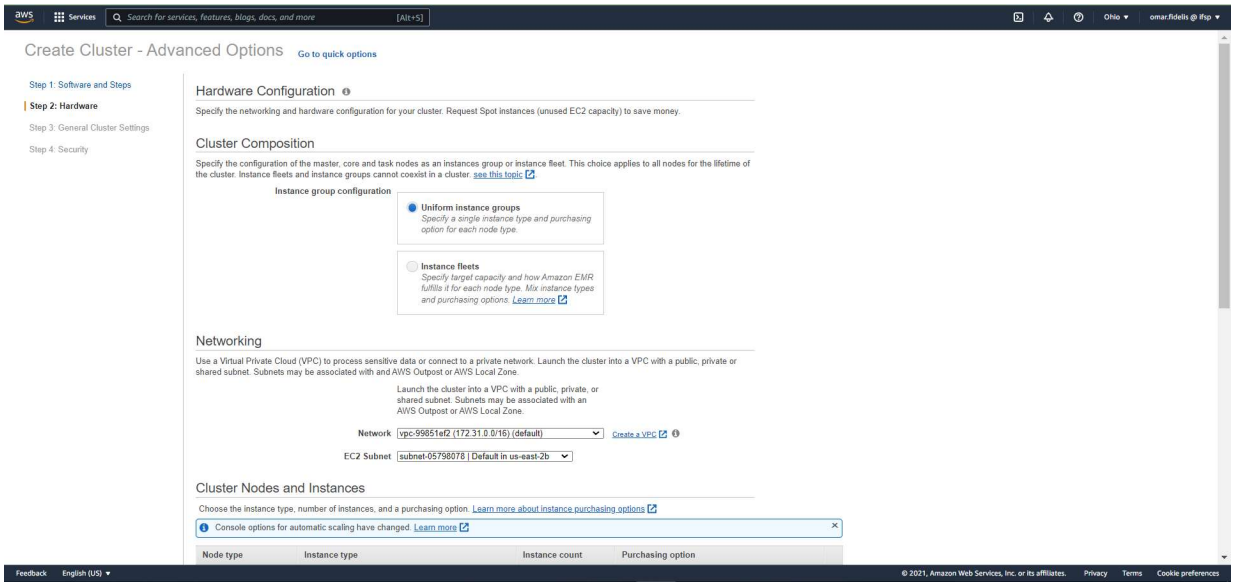


2.3. Create EMR Cluster

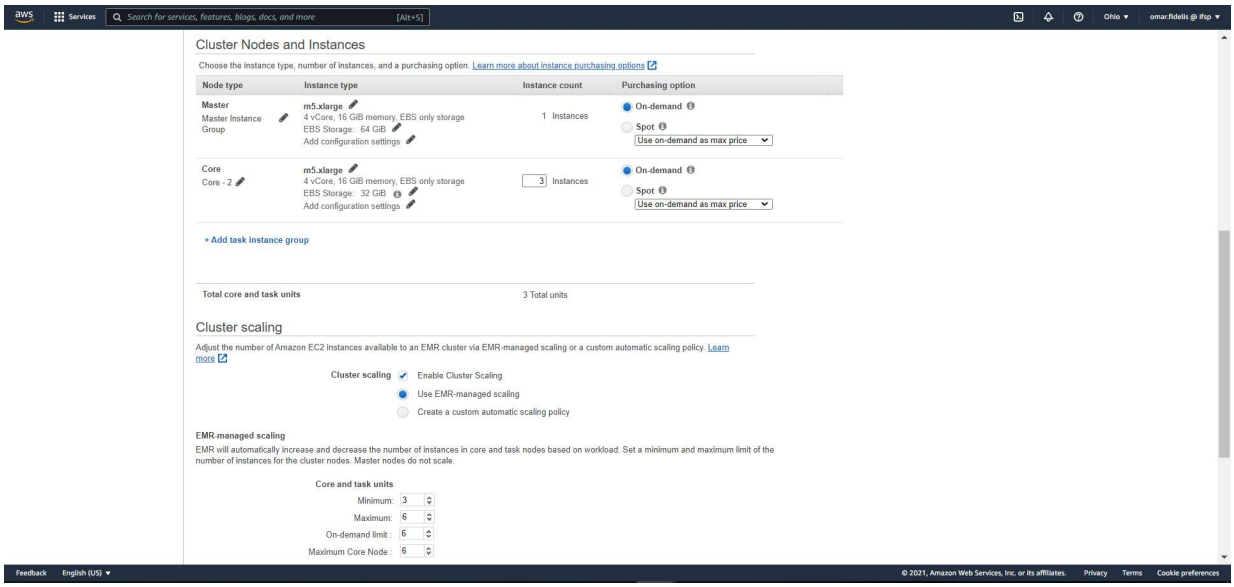
Lista de aplicações do Cluster



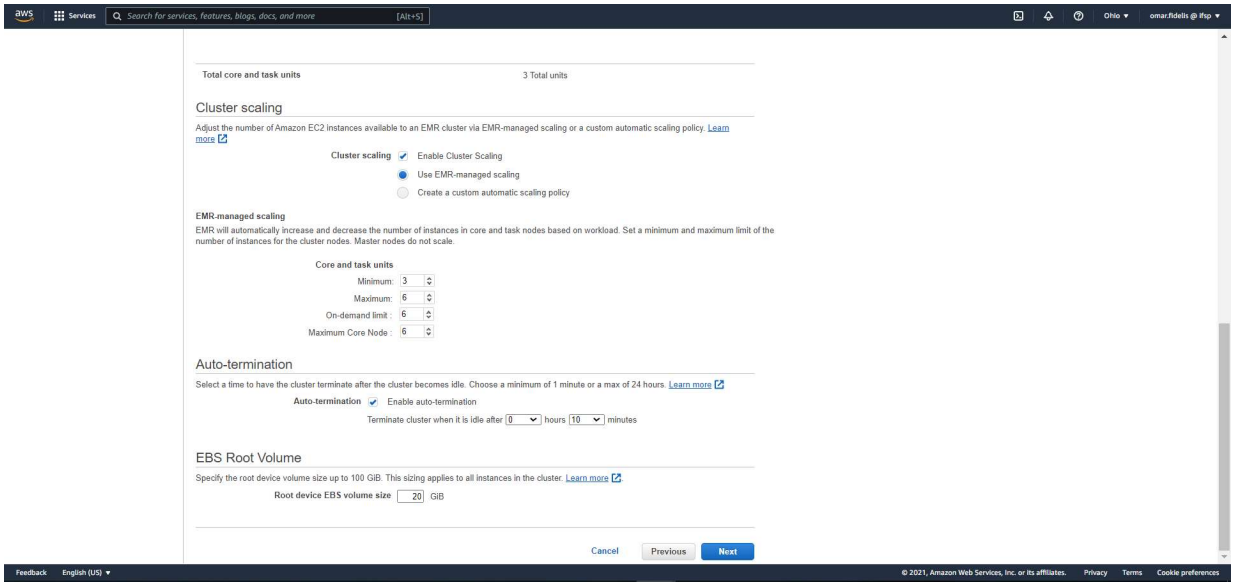
Hardware Configuration



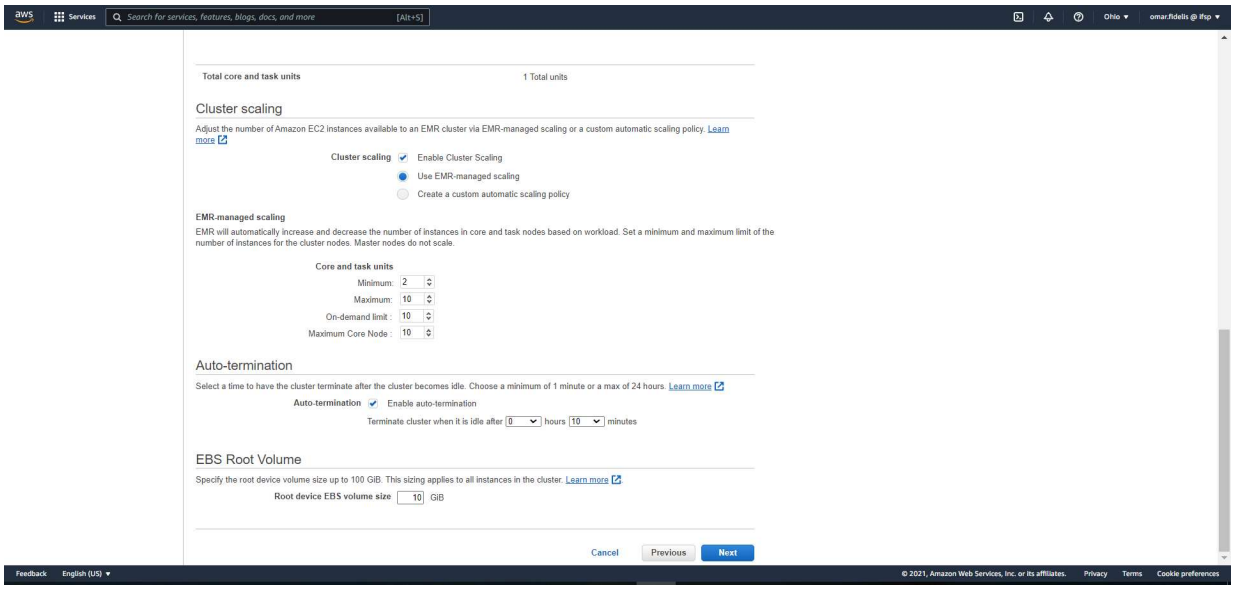
Hardware Description



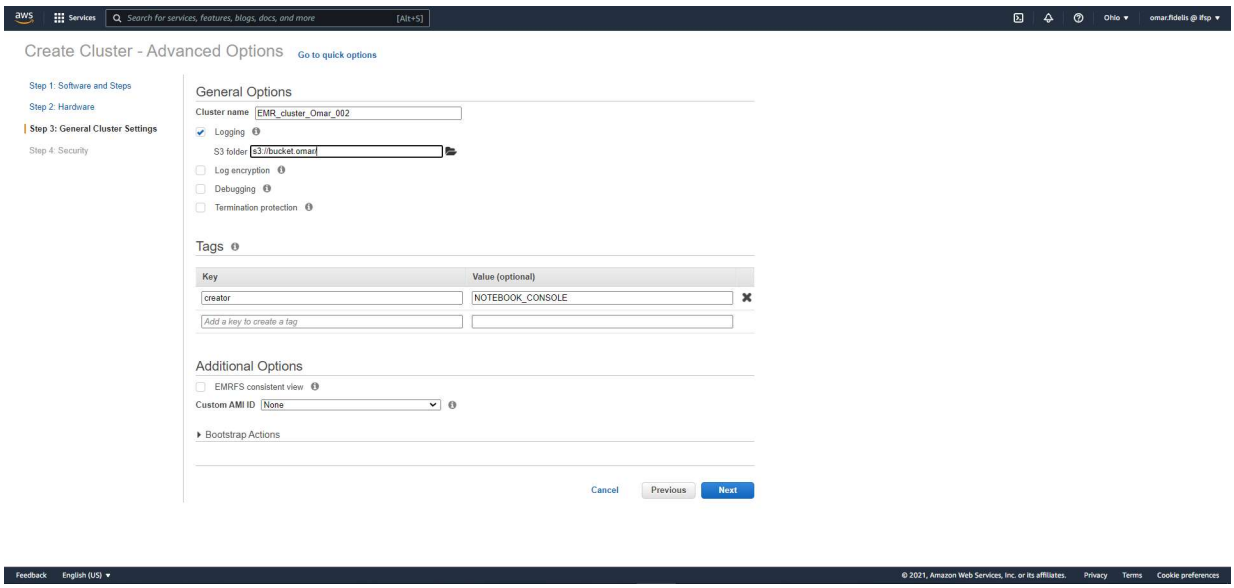
Habilitando Cluster Scaling



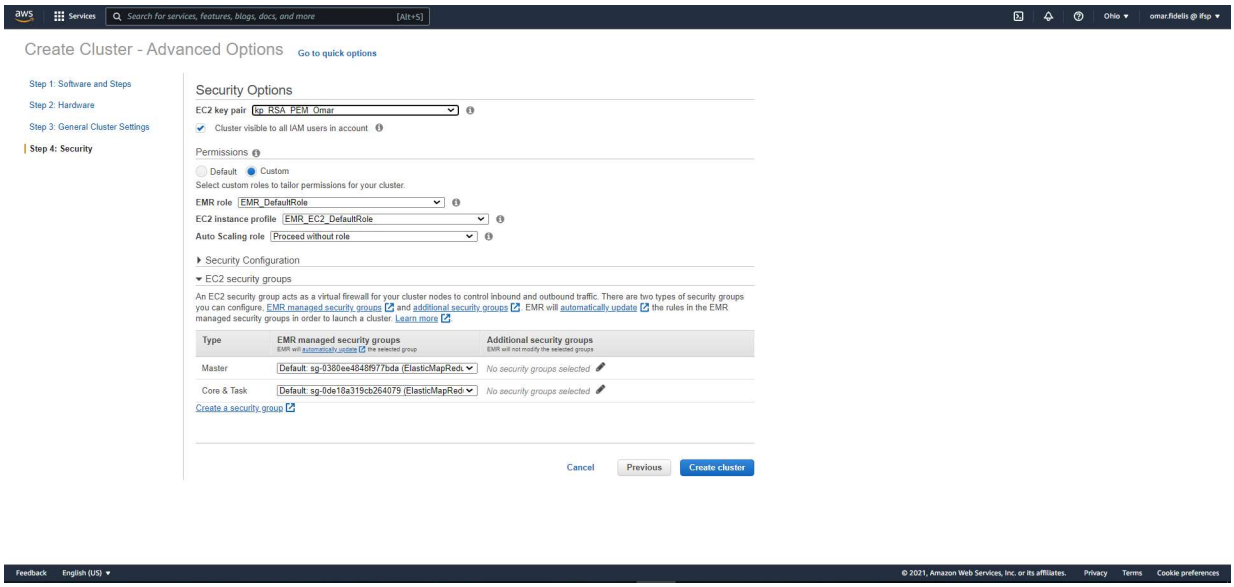
Aumento hardware configuration



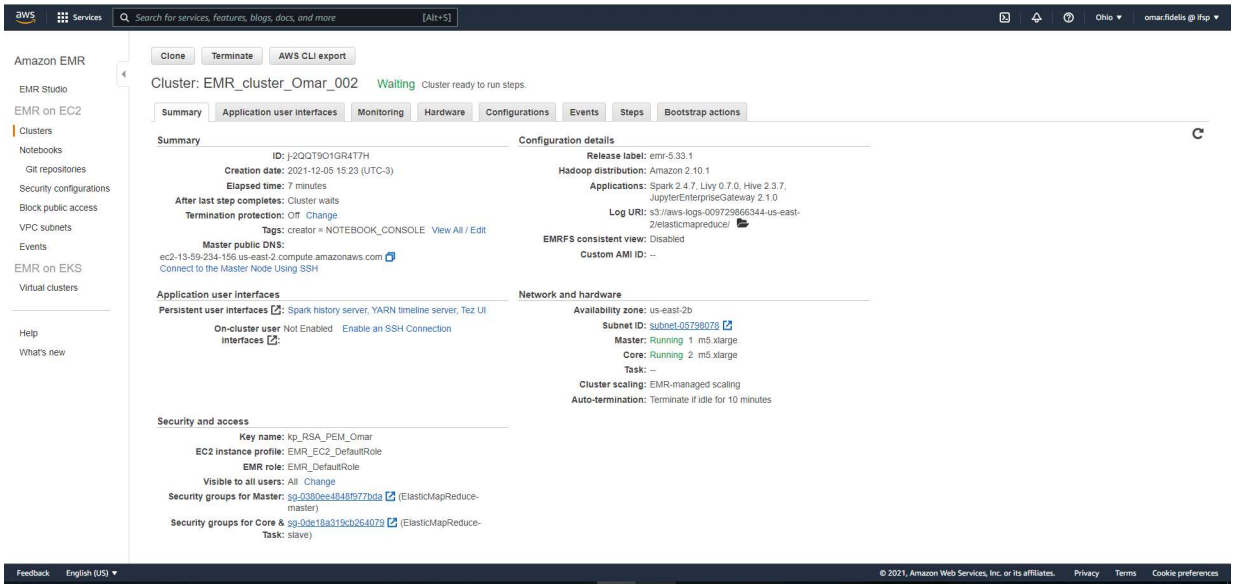
Definição de Bucket de LOG



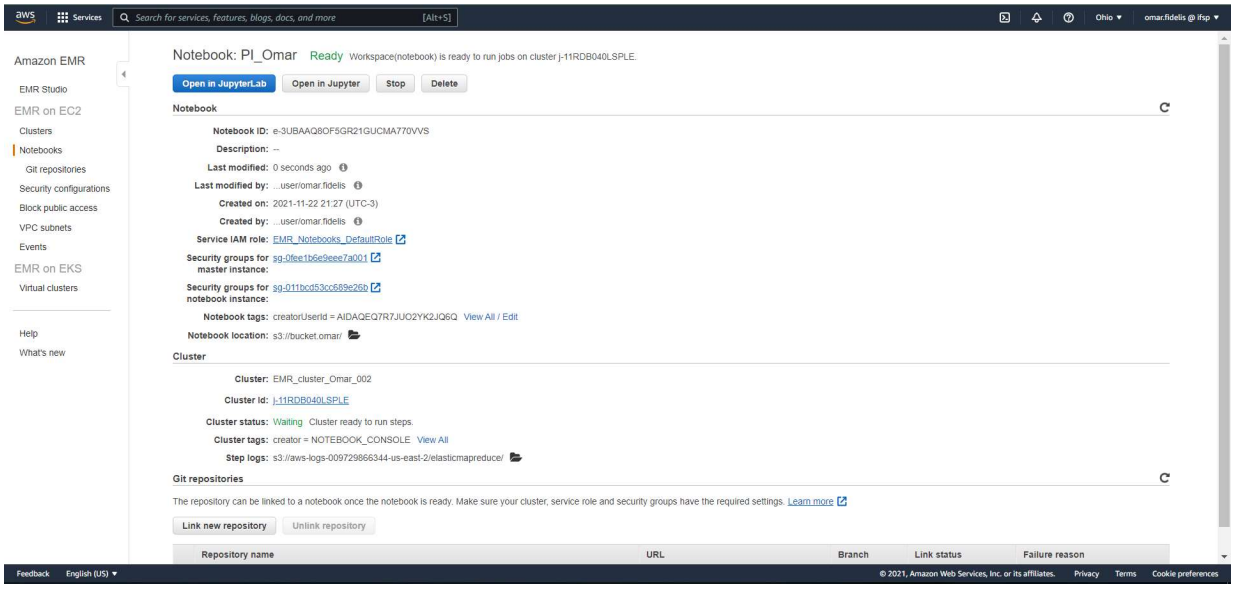
Configuração padrão (key, role, profile, etc.)



Running server

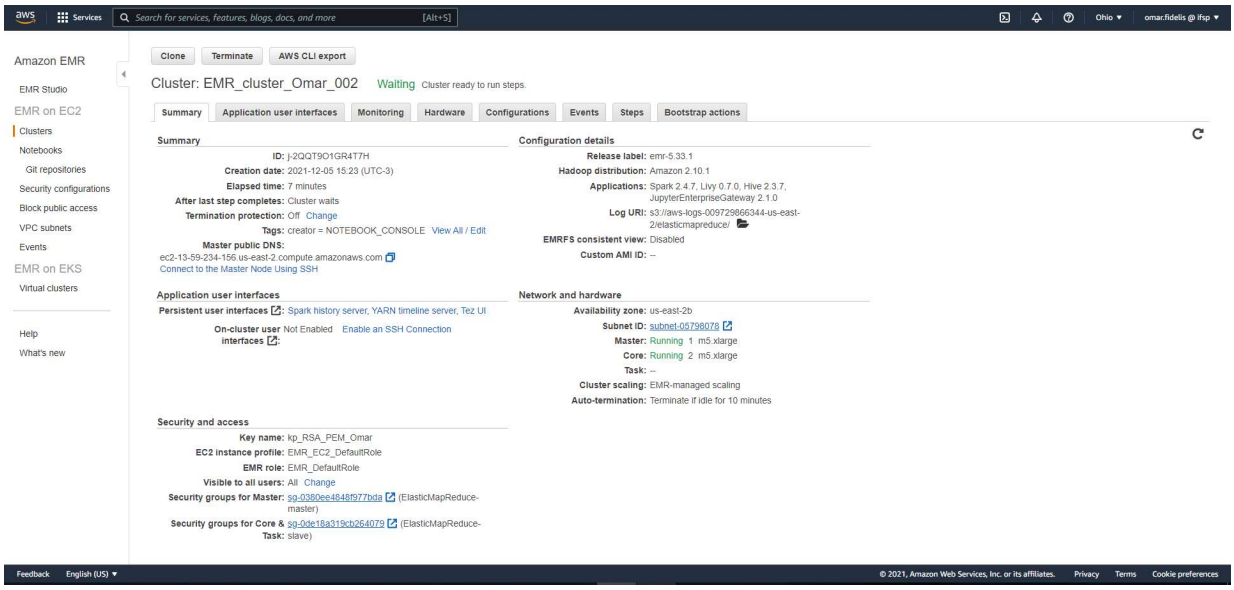


Running Jupyter Notebook

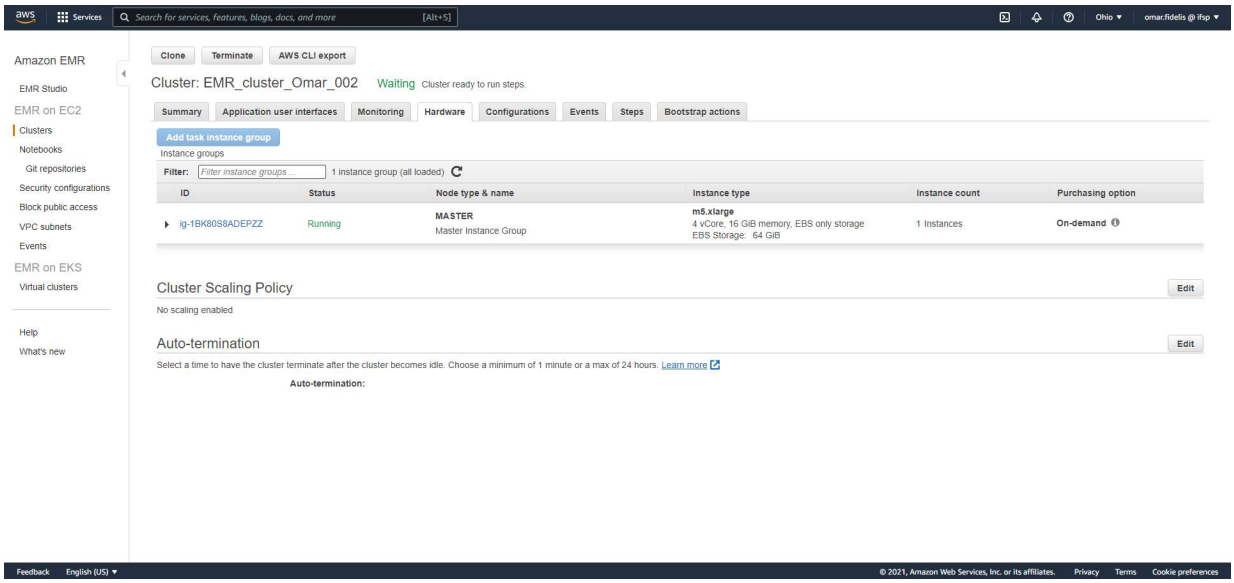


2.4. Hardware Configuration

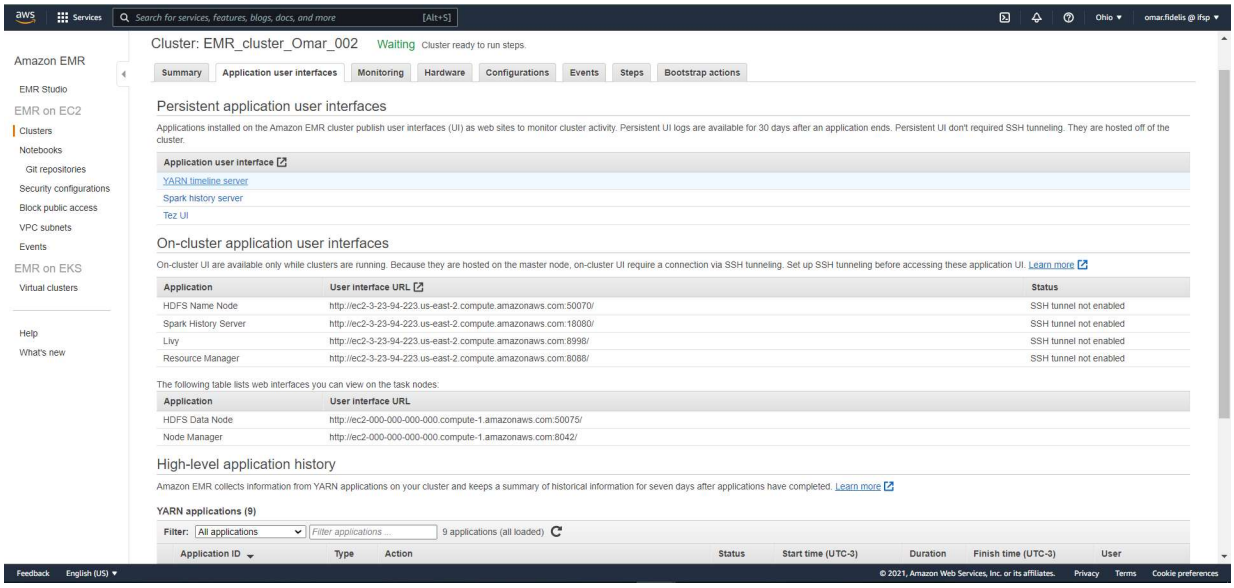
Summary Configuration



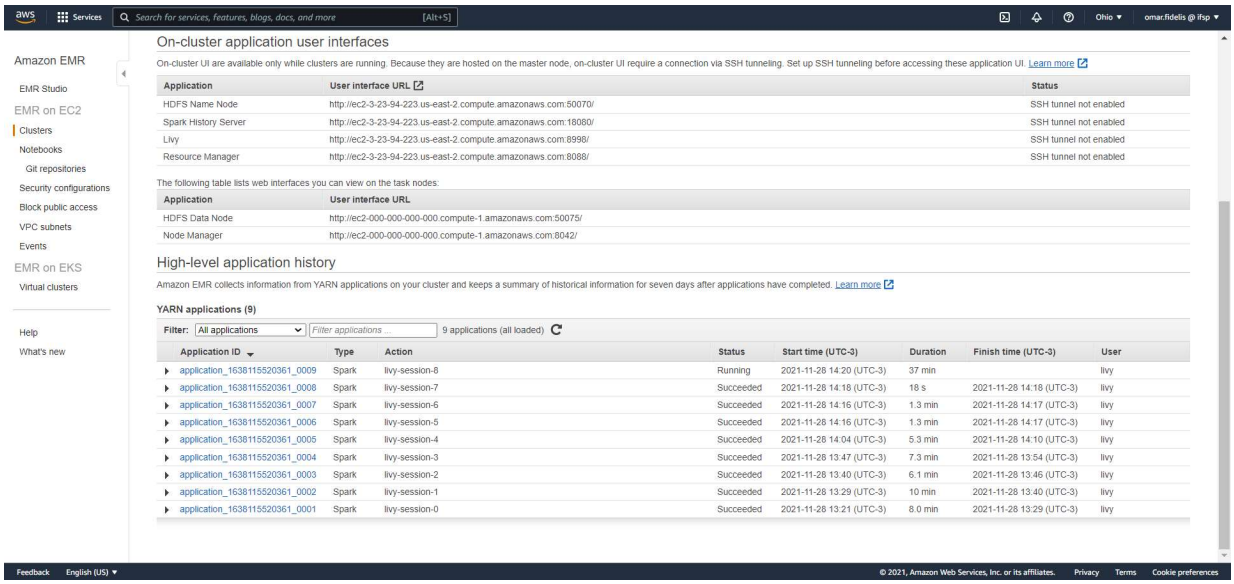
Hardware Configuration



Application Configuration



Spark Sessions



3. Preparação do ambiente

In [1]:
```
# Import Library
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
```

▸ Spark Job Progress

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	
1	application_1638734260039_0002	pyspark	idle	Link (http://ip-172-31-28-2.compute.internal:20888/proxy/application_163873426

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

In [2]:
```
# Start Spark Session
spark = SparkSession \
            .builder \
            .getOrCreate()
```

In [3]:
```
# Data Strucuture
schema = StructType([ \
                    StructField("Acidez_Fixa", FloatType(), True), \
                    StructField("Acidez_Volatil", FloatType(), True), \
                    StructField("Acido_Citrico", FloatType(), True), \
                    StructField("Acucar_Residual", FloatType(), True), \
                    StructField("Cloreto_Sodio", FloatType(), True), \
                    StructField("Dioxido_Enxofre_Livre", FloatType(), True), \
                    StructField("Dioxido_Enxofre_Total", FloatType(), True), \
                    StructField("Densidade", FloatType(), True), \
                    StructField("pH", FloatType(), True), \
                    StructField("Sulfato", FloatType(), True), \
                    StructField("Alcool", FloatType(), True), \
                    StructField("Qualidade", FloatType(), True), \
                    StructField("Tipo", StringType(), True)])
```

3.1. Preparação dataset

In [4]:
```
# Import Dataset from S3 Bucket
df = spark.read.schema(schema).option("header","true").csv(r"s3n://bucket.omar/da
```

In [5]:
```
# Overview
df.show(5)
```

▸ Spark Job Progress

```
+-----------+-------------+-------------+--------------+-------------+-------
-------------+--------------------+---------+----+-------+------+---------+--
--+
|Acidez_Fixa|Acidez_Volatil|Acido_Citrico|Acucar_Residual|Cloreto_Sodio|Dioxido
_Enxofre_Livre|Dioxido_Enxofre_Total|Densidade|  pH|Sulfato|Alcool|Qualidade|Ti
po|
+-----------+-------------+-------------+--------------+-------------+-------
-------------+--------------------+---------+----+-------+------+---------+--
--+
|        7.4|          0.7|          0.0|           1.9|        0.076|
11.0|                34.0|   0.9978|3.51|   0.56|   9.4|      5.0| red|
|        7.8|         0.88|          0.0|           2.6|        0.098|
25.0|                67.0|   0.9968| 3.2|   0.68|   9.8|      5.0| red|
|        7.8|         0.76|         0.04|           2.3|        0.092|
15.0|                54.0|    0.997|3.26|   0.65|   9.8|      5.0| red|
|       11.2|         0.28|         0.56|           1.9|        0.075|
17.0|                60.0|    0.998|3.16|   0.58|   9.8|      6.0| red|
|        7.4|          0.7|          0.0|           1.9|        0.076|
11.0|                34.0|   0.9978|3.51|   0.56|   9.4|      5.0| red|
+-----------+-------------+-------------+--------------+-------------+-------
-------------+--------------------+---------+----+-------+------+---------+--
--+
only showing top 5 rows
```

In [6]:
```
# Staset Schema
df.printSchema()
```

```
root
 |-- Acidez_Fixa: float (nullable = true)
 |-- Acidez_Volatil: float (nullable = true)
 |-- Acido_Citrico: float (nullable = true)
 |-- Acucar_Residual: float (nullable = true)
 |-- Cloreto_Sodio: float (nullable = true)
 |-- Dioxido_Enxofre_Livre: float (nullable = true)
 |-- Dioxido_Enxofre_Total: float (nullable = true)
 |-- Densidade: float (nullable = true)
 |-- pH: float (nullable = true)
 |-- Sulfato: float (nullable = true)
 |-- Alcool: float (nullable = true)
 |-- Qualidade: float (nullable = true)
 |-- Tipo: string (nullable = true)
```

In [7]:
```
# No missing values
df.select(df.columns).describe().show()
```

▸ Spark Job Progress

```
+-------+------------------+------------------+------------------+---------
--------+------------------+--------------------+--------------------+----
---------------+------------------+-----------------+------------------+--
---------------+-----+
|summary|       Acidez_Fixa|     Acidez_Volatil|      Acido_Citrico|  Acucar_
Residual|      Cloreto_Sodio|Dioxido_Enxofre_Livre|Dioxido_Enxofre_Total|
Densidade|                pH|           Sulfato|            Alcool|
Qualidade| Tipo|
+-------+------------------+------------------+------------------+---------
--------+------------------+--------------------+--------------------+----
---------------+------------------+-----------------+------------------+--
---------------+-----+
|  count|              6497|              6497|              6497|
6497|              6497|                6497|                6497|
6497|              6497|              6497|              6497|
6497| 6497|
|   mean|7.2153070684174345| 0.3396659997464309|0.31863321590700183|5.4432353
35688381|0.05603386190293733|   30.525319378174544|    115.7445744189626|  0.
9946966336675106|  3.218500847887659|0.5312682763074607|10.491800813027597|
5.818377712790519| null|
| stddev|1.2964337581432526|0.16463647361833078|0.14531786519512516|4.7578037
43705875|0.03503360134013471|    17.74939977200251|    56.52185452263024|0.00
2998673927059...|0.16078720153691367|0.1488058737693289|1.1927117488301993|0.
8732552715311255| null|
|    min|               3.8|              0.08|               0.0|
0.6|             0.009|                1.0|                6.0|
0.98711|              2.72|              0.22|               8.0|
3.0|  red|
|    max|              15.9|              1.58|              1.66|
65.8|             0.611|               289.0|               440.0|
1.03898|              4.01|               2.0|             14.9|
9.0|white|
+-------+------------------+------------------+------------------+---------
--------+------------------+--------------------+--------------------+----
---------------+------------------+-----------------+------------------+--
---------------+-----+
```

In [8]:
```
# Create VIEW for dataset
df.createOrReplaceTempView("table_wine")
```

In [9]:
```
# Quantidede de amostrar por tipo de vinho
df2 = spark.sql('SELECT count(*) as Total, Tipo from table_wine group by Tipo')
df2.collect()
```

▸ Spark Job Progress

```
[Row(Total=4898, Tipo='white'), Row(Total=1599, Tipo='red')]
```

In [10]:
```
# Valor máximo de cada atributo
df2 = spark.sql('SELECT max(Acidez_Fixa),\
                        max(Acidez_Volatil),\
                        max(Acido_Citrico),\
                        max(Acucar_Residual),\
                        max(Cloreto_Sodio),\
                        max(Dioxido_Enxofre_Livre),\
                        max(Dioxido_Enxofre_Total),\
                        max(Densidade),\
                        max(pH),\
                        max(Sulfato),\
                        max(Alcool),\
                        max(Qualidade) from table_wine')
df2.collect()
```

▸ Spark Job Progress

```
[Row(max(Acidez_Fixa)=15.899999618530273, max(Acidez_Volatil)=1.5800000429153442, max(Acido_Citrico)=1.659999966621399, max(Acucar_Residual)=65.80000305175781, max(Cloreto_Sodio)=0.6110000014305115, max(Dioxido_Enxofre_Livre)=289.0, max(Dioxido_Enxofre_Total)=440.0, max(Densidade)=1.0389800071716309, max(pH)=4.010000228881836, max(Sulfato)=2.0, max(Alcool)=14.899999618530273, max(Qualidade)=9.0)]
```

In [11]:
```
# Valor mínimo de cada atributo
df2 = spark.sql('SELECT min(Acidez_Fixa),\
                        min(Acidez_Volatil),\
                        min(Acido_Citrico),\
                        min(Acucar_Residual),\
                        min(Cloreto_Sodio),\
                        min(Dioxido_Enxofre_Livre),\
                        min(Dioxido_Enxofre_Total),\
                        min(Densidade),\
                        min(pH),\
                        min(Sulfato),\
                        min(Alcool),\
                        min(Qualidade) from table_wine')
df2.collect()
```

▸ Spark Job Progress

```
[Row(min(Acidez_Fixa)=3.799999952316284, min(Acidez_Volatil)=0.07999999821186066, min(Acido_Citrico)=0.0, min(Acucar_Residual)=0.6000000238418579, min(Cloreto_Sodio)=0.008999999612569809, min(Dioxido_Enxofre_Livre)=1.0, min(Dioxido_Enxofre_Total)=6.0, min(Densidade)=0.9871100187301636, min(pH)=2.7200000286102295, min(Sulfato)=0.2199999988079071, min(Alcool)=8.0, min(Qualidade)=3.0)]
```

In [12]:
```
# Valor médio de cada atributo
df2 = spark.sql('SELECT avg(Acidez_Fixa),\
                        avg(Acidez_Volatil),\
                        avg(Acido_Citrico),\
                        avg(Acucar_Residual),\
                        avg(Cloreto_Sodio),\
                        avg(Dioxido_Enxofre_Livre),\
                        avg(Dioxido_Enxofre_Total),\
                        avg(Densidade),\
                        avg(pH),\
                        avg(Sulfato),\
                        avg(Alcool),\
                        avg(Qualidade) from table_wine')
df2.collect()
```

▸ Spark Job Progress

```
[Row(avg(Acidez_Fixa)=7.2153070684174345, avg(Acidez_Volatil)=0.3396659997464309, avg(Acido_Citrico)=0.31863321590700183, avg(Acucar_Residual)=5.443235335688381, avg(Cloreto_Sodio)=0.05603386190293733, avg(Dioxido_Enxofre_Livre)=30.525319378174544, avg(Dioxido_Enxofre_Total)=115.7445744189626, avg(Densidade)=0.9946966336675106, avg(pH)=3.218500847887659, avg(Sulfato)=0.5312682763074607, avg(Alcool)=10.491800813027597, avg(Qualidade)=5.818377712790519)]
```

In [13]:
```
# 10 amostras melhor avaliadas
df2 = spark.sql('SELECT Qualidade, Tipo from table_wine order by Qualidade desc l
df2.collect()
```

▸ Spark Job Progress

```
[Row(Qualidade=9.0, Tipo='white'), Row(Qualidade=9.0, Tipo='white'), Row(Qualid
ade=9.0, Tipo='white'), Row(Qualidade=9.0, Tipo='white'), Row(Qualidade=9.0, Ti
po='white'), Row(Qualidade=8.0, Tipo='red'), Row(Qualidade=8.0, Tipo='red'), Ro
w(Qualidade=8.0, Tipo='red'), Row(Qualidade=8.0, Tipo='red'), Row(Qualidade=8.
0, Tipo='red')]
```

In [14]:
```
# Quantidade de amostras por avaliação e label
df2 = spark.sql('SELECT count(*) as Total, Qualidade, Tipo\
                    from table_wine\
                    group by Qualidade, Tipo\
                    order by Qualidade desc, Tipo')
df2.collect()
```

▸ Spark Job Progress

```
[Row(Total=5, Qualidade=9.0, Tipo='white'), Row(Total=18, Qualidade=8.0, Tipo
='red'), Row(Total=175, Qualidade=8.0, Tipo='white'), Row(Total=199, Qualidade=
7.0, Tipo='red'), Row(Total=880, Qualidade=7.0, Tipo='white'), Row(Total=638, Q
ualidade=6.0, Tipo='red'), Row(Total=2198, Qualidade=6.0, Tipo='white'), Row(To
tal=681, Qualidade=5.0, Tipo='red'), Row(Total=1457, Qualidade=5.0, Tipo='whit
e'), Row(Total=53, Qualidade=4.0, Tipo='red'), Row(Total=163, Qualidade=4.0, Ti
po='white'), Row(Total=10, Qualidade=3.0, Tipo='red'), Row(Total=20, Qualidade=
3.0, Tipo='white')]
```

In [15]:
```
# Transform categorical atrribute ("Tipo") to bynary attribute
df = df.withColumn('label', col('Tipo') == 'red')
df = df.drop('Tipo')
df.show(5)
```

▸ Spark Job Progress

```
+-----------+-------------+------------+--------------+------------+-----
---------------+--------------------+---------+----+-------+------+--------
-+-----+
|Acidez_Fixa|Acidez_Volatil|Acido_Citrico|Acucar_Residual|Cloreto_Sodio|Dioxi
do_Enxofre_Livre|Dioxido_Enxofre_Total|Densidade|  pH|Sulfato|Alcool|Qualidad
e|label|
+-----------+-------------+------------+--------------+------------+-----
---------------+--------------------+---------+----+-------+------+--------
-+-----+
|        7.4|          0.7|         0.0|           1.9|       0.076|
11.0|                34.0|   0.9978|3.51|   0.56|   9.4|     5.0| true|
|        7.8|         0.88|         0.0|           2.6|       0.098|
25.0|                67.0|   0.9968| 3.2|   0.68|   9.8|     5.0| true|
|        7.8|         0.76|        0.04|           2.3|       0.092|
15.0|                54.0|    0.997|3.26|   0.65|   9.8|     5.0| true|
|       11.2|         0.28|        0.56|           1.9|       0.075|
17.0|                60.0|    0.998|3.16|   0.58|   9.8|     6.0| true|
|        7.4|          0.7|         0.0|           1.9|       0.076|
11.0|                34.0|   0.9978|3.51|   0.56|   9.4|     5.0| true|
+-----------+-------------+------------+--------------+------------+-----
---------------+--------------------+---------+----+-------+------+--------
-+-----+
only showing top 5 rows
```

In [16]:
```
# Cast bynary attribute to numerical attribute named "label" (standard)
from pyspark.sql.types import IntegerType
df = df.withColumn("label",col("label").cast(IntegerType()))
df.printSchema()
```

```
root
 |-- Acidez_Fixa: float (nullable = true)
 |-- Acidez_Volatil: float (nullable = true)
 |-- Acido_Citrico: float (nullable = true)
 |-- Acucar_Residual: float (nullable = true)
 |-- Cloreto_Sodio: float (nullable = true)
 |-- Dioxido_Enxofre_Livre: float (nullable = true)
 |-- Dioxido_Enxofre_Total: float (nullable = true)
 |-- Densidade: float (nullable = true)
 |-- pH: float (nullable = true)
 |-- Sulfato: float (nullable = true)
 |-- Alcool: float (nullable = true)
 |-- Qualidade: float (nullable = true)
 |-- label: integer (nullable = true)
```

4. Processamento (Machine Learning)

In [17]:
```
# Import Libraies - Machine Learning
from pyspark.ml.feature import VectorAssembler, Normalizer, StandardScaler
from pyspark.ml.classification import LogisticRegression, NaiveBayes, LinearSVC,
from pyspark.ml.evaluation import MulticlassClassificationEvaluator, BinaryClassi
from pyspark.ml import Pipeline
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

In [18]:
```
# Split dataset (training 80% and test 20%)
training, test = df.randomSplit([0.8, 0.2], seed=10)
```

In [19]:
```
# Create Assembler
assembler = VectorAssembler(inputCols=['Acidez_Fixa','Acidez_Volatil','Acido_Citr
                                       'Cloreto_Sodio','Dioxido_Enxofre_Livre','D
                                       'Densidade','pH','Sulfato','Alcool','Quali
                            outputCol="inputFeatures")
```

In [20]:
```
# Standard Scaler (Normalize)
#scaler = Normalizer(inputCol = "inputFeatures", outputCol = "features")
scaler = StandardScaler(inputCol = "inputFeatures", outputCol = "features", withS
```

4.1. Logistic Regression

In [21]:
```
# Model Creation
lr = LogisticRegression()
```

In [22]:
```
# Creating Pipeline
#pipeline_lr = Pipeline(stages = [assembler, lr])
pipeline_lr = Pipeline(stages = [assembler, scaler, lr])
```

In [23]:
```
# Parameter
paramgrid_lr = ParamGridBuilder().addGrid(lr.regParam, [0.0, 0,1]).addGrid(lr.max
```

In [24]:
```
# Evaluator
evaluator_lr = MulticlassClassificationEvaluator(metricName = "accuracy")
```

In [25]:
```
# Cross Validation
crossval_lr = CrossValidator(estimator = pipeline_lr, estimatorParamMaps = paramg
                             evaluator = evaluator_lr, numFolds = 3, seed=10)
```

In [26]:
```
# Training and evaluation
cvModel_lr = crossval_lr.fit(training)
evaluator_lr.evaluate(cvModel_lr.transform(test))
```

▸ Spark Job Progress

```
0.7574568288854003
```

In [27]:
```
# Evaluator
evaluator_lr = MulticlassClassificationEvaluator(metricName = "f1")
```

In [28]:
```
# Cross Validation
crossval_lr = CrossValidator(estimator = pipeline_lr, estimatorParamMaps = paramg
                             evaluator = evaluator_lr, numFolds = 3, seed=10)
```

In [29]:
```
# Training and evaluation
cvModel_lr = crossval_lr.fit(training)
evaluator_lr.evaluate(cvModel_lr.transform(test))
```

▸ Spark Job Progress

```
0.653716541955618
```

4.2. Naive Bayes

In [30]:
```
# Model Creation
nb = NaiveBayes()
```

In [31]:
```
# Creating Pipeline
#pipeline_nb = Pipeline(stages = [assembler, nb])
pipeline_nb = Pipeline(stages = [assembler, scaler, nb])
```

In [32]:
```
# Parameter
paramgrid_nb = ParamGridBuilder().build()
```

In [33]:
```
# Evaluator
evaluator_nb = MulticlassClassificationEvaluator(metricName = "accuracy")
```

In [34]:
```
# Cross Validation
crossval_nb = CrossValidator(estimator = pipeline_nb, estimatorParamMaps = paramg
                             evaluator = evaluator_nb, numFolds = 3, seed=10)
```

In [35]:
```
# Training and evaluation
cvModel_nb = crossval_nb.fit(training)
evaluator_nb.evaluate(cvModel_nb.transform(test))
```

▸ Spark Job Progress

```
0.957613814756672
```

In [36]:
```
# Evaluator
evaluator_nb = MulticlassClassificationEvaluator(metricName = "f1")
```

In [37]:
```
# Cross Validation
crossval_nb = CrossValidator(estimator = pipeline_nb, estimatorParamMaps = paramg
                             evaluator = evaluator_nb, numFolds = 3, seed=10)
```

In [38]:
```
# Training and evaluation
cvModel_nb = crossval_nb.fit(training)
evaluator_nb.evaluate(cvModel_nb.transform(test))
```

▸ Spark Job Progress

```
0.9567514419209051
```

4.3. Linear Support Vector Machine

In [39]:
```
# Model Creation
lsvc = LinearSVC()
```

In [40]:
```
# Creating Pipeline
#pipeline_lsvc = Pipeline(stages = [assembler, lsvc])
pipeline_lsvc = Pipeline(stages = [assembler, scaler, lsvc])
```

In [41]:
```
# Parameter
paramgrid_lsvc = ParamGridBuilder().build()
```

In [42]:
```
# Evaluator
evaluator_lsvc = MulticlassClassificationEvaluator(metricName = "accuracy")
```

In [43]:
```
# Cross Validation
crossval_lsvc = CrossValidator(estimator = pipeline_lsvc, estimatorParamMaps = pa
                               evaluator = evaluator_lsvc, numFolds = 3, seed=10)
```

In [44]:
```
# Training and evaluation
cvModel_lsvc = crossval_lsvc.fit(training)
evaluator_lsvc.evaluate(cvModel_lsvc.transform(test))
```

▸ Spark Job Progress

```
Exception in thread cell_monitor-44:
Traceback (most recent call last):
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 926, in _bootstrap_
inner
    self.run()
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "/mnt/notebook-env/lib/python3.7/site-packages/awseditorssparkmonitoring
widget-1.0-py3.7.egg/awseditorssparkmonitoringwidget/cellmonitor.py", line 178,
in cell_monitor
    job_binned_stages[job_id][stage_id] = all_stages[stage_id]
KeyError: 1299


0.978021978021978
```

In [45]:
```
# Evaluator
evaluator_lsvc = MulticlassClassificationEvaluator(metricName = "f1")
```

In [46]:
```
# Cross Validation
crossval_lsvc = CrossValidator(estimator = pipeline_lsvc, estimatorParamMaps = pa
                               evaluator = evaluator_lsvc, numFolds = 3, seed=10)
```

In [47]:
```
# Training and evaluation
cvModel_lsvc = crossval_lsvc.fit(training)
evaluator_lsvc.evaluate(cvModel_lsvc.transform(test))
```

▸ Spark Job Progress

```
Exception in thread cell_monitor-47:
Traceback (most recent call last):
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 926, in _bootstrap_
inner
    self.run()
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "/mnt/notebook-env/lib/python3.7/site-packages/awseditorssparkmonitoring
widget-1.0-py3.7.egg/awseditorssparkmonitoringwidget/cellmonitor.py", line 178,
in cell_monitor
    job_binned_stages[job_id][stage_id] = all_stages[stage_id]
KeyError: 2699


0.9778985331439274
```

4.4. Random Forest Classifier

In [48]:
```
# Model Creation
rf = RandomForestClassifier()
```

In [49]:
```
# Creating Pipeline
#pipeline_rf = Pipeline(stages = [assembler, rf])
pipeline_rf = Pipeline(stages = [assembler, scaler, rf])
```

In [50]:
```
# Parameter
paramgrid_rf = ParamGridBuilder().build()
```

In [51]:
```
# Evaluator
evaluator_rf = MulticlassClassificationEvaluator(metricName = "accuracy")
```

In [52]:
```
# Cross Validation
crossval_rf = CrossValidator(estimator = pipeline_rf, estimatorParamMaps = paramg
                             evaluator = evaluator_rf, numFolds = 3, seed=10)
```

In [53]:
```
# Training and evaluation
cvModel_rf = crossval_rf.fit(training)
evaluator_rf.evaluate(cvModel_rf.transform(test))
```

▸ Spark Job Progress

```
0.9945054945054945
```

In [54]:
```
# Evaluator
evaluator_rf = MulticlassClassificationEvaluator(metricName = "f1")
```

In [55]:
```
# Cross Validation
crossval_rf = CrossValidator(estimator = pipeline_rf, estimatorParamMaps = paramg
                             evaluator = evaluator_rf, numFolds = 3, seed=10)
```

In [56]:
```
# Training and evaluation
cvModel_rf = crossval_rf.fit(training)
evaluator_rf.evaluate(cvModel_rf.transform(test))
```

▸ Spark Job Progress

```
Exception in thread cell_monitor-53:
Traceback (most recent call last):
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 926, in _bootstrap_
inner
    self.run()
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "/mnt/notebook-env/lib/python3.7/site-packages/awseditorssparkmonitoring
widget-1.0-py3.7.egg/awseditorssparkmonitoringwidget/cellmonitor.py", line 178,
in cell_monitor
    job_binned_stages[job_id][stage_id] = all_stages[stage_id]
KeyError: 3793


0.9944902653065121
```

4.5. Gradient Boosted Tree Classifier

In [57]:
```
# Model Creation
gbt = GBTClassifier()
```

In [58]:
```
# Creating Pipeline
#pipeline_gbt = Pipeline(stages = [assembler, gbt])
pipeline_gbt = Pipeline(stages = [assembler, scaler, gbt])
```

In [59]:
```
# Parameter
paramgrid_gbt = ParamGridBuilder().build()
```

In [60]:
```
# Evaluator
evaluator_gbt = MulticlassClassificationEvaluator(metricName = "accuracy")
```

In [61]:
```
# Cross Validation
crossval_gbt = CrossValidator(estimator = pipeline_gbt, estimatorParamMaps = para
                              evaluator = evaluator_gbt, numFolds = 3, seed=10)
```

In [62]:
```
# Training and evaluation
cvModel_gbt = crossval_gbt.fit(training)
evaluator_gbt.evaluate(cvModel_gbt.transform(test))
```

▸ Spark Job Progress

```
Exception in thread cell_monitor-56:
Traceback (most recent call last):
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 926, in _bootstrap_
inner
    self.run()
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "/mnt/notebook-env/lib/python3.7/site-packages/awseditorssparkmonitoring
widget-1.0-py3.7.egg/awseditorssparkmonitoringwidget/cellmonitor.py", line 178,
in cell_monitor
    job_binned_stages[job_id][stage_id] = all_stages[stage_id]
KeyError: 3937


0.9905808477237049
```

In [63]:
```
# Evaluator
evaluator_gbt = MulticlassClassificationEvaluator(metricName = "f1")
```

In [64]:
```
# Cross Validation
crossval_gbt = CrossValidator(estimator = pipeline_gbt, estimatorParamMaps = para
                              evaluator = evaluator_gbt, numFolds = 3, seed=10)
```

In [65]:
```
# Training and evaluation
cvModel_gbt = crossval_gbt.fit(training)
evaluator_gbt.evaluate(cvModel_gbt.transform(test))
```

▸ Spark Job Progress

```
0.9905808477237049
```

In [66]:
```
# spark.sparkContext.stop()
spark.stop()
```

▸ Spark Job Progress

4.6. Tempo de Execução

Tempo de execução	Servers		Dif.
	01 node (*)	05 nodes (*)	
Preparation	01:25,2	00:40,9	00:44,3
Logistic Regression	00:45,3	00:37,2	00:08,1
Naive Bayes	00:28,0	00:17,1	00:10,9
Linear Support Vector Machine	02:32,2	02:13,4	00:18,8
Random Forest Classifier	00:38,7	00:35,3	00:03,4
Gradient Boosted Tree Classifier	01:42,8	01:32,7	00:10,1

(*) node = m5.xlarge (4 vCore, 16 GiB memory)

4.7. Consolidado dos Resultados

			Normalizer		Standard Scaler	
	Accuracy	F1	Accuracy	F1	Accuracy	F1
Logistic Regression	0,7575	0,6537	0,9788	0,9787	0,7575	0,6537
Naive Bayes	0,9184	0,9185	0,7763	0,6955	0,9576	0,9568
Linear Support Vector Machine	0,9780	0,9779	0,9804	0,9803	0,9780	0,9779
Random Forest Classifier	0,9945	0,9945	0,9780	0,9780	0,9945	0,9945
Gradient Boosted Tree Classifier	0,9906	0,9906	0,9843	0,9844	0,9906	0,9906

(*) Normalizer is a Transformer which transforms a dataset of Vector rows, normalizing each Vector to have unit norm. It takes parameter p, which specifies the p-norm used for normalization. (p=2 by default.) This normalization can help standardize your input data and improve the behavior of learning algorithms.

(*) StandardScaler transforms a dataset of Vector rows, normalizing each feature to have unit standard deviation and/or zero mean. It takes parameters:

withStd: True by default. Scales the data to unit standard deviation.
withMean: False by default. Centers the data with mean before scaling. It will build a dense output, so take care when applying to sparse input.

5. Referência

Página do Professor Paulo Cortez
http://www3.dsi.uminho.pt/pcortez/Home.html

Página de download da base de dados (dataset winequality.zip)
http://www3.dsi.uminho.pt/pcortez/wine/

Spark Apache
https://spark.apache.org/

Apache Spark
https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark.html

Create a cluster with Spark
https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark-launch.html

Apache Hive
https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hive.html

Apache Livy
https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-livy.html

Amazon EMR Notebook based on Jupyter Notebook
https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-jupyter-emr-managed-notebooks.html

Spark 3.2.0 - ML Pipelines
https://spark.apache.org/docs/latest/ml-guide.html

Machine Learning with PySpark and Amazon EMR
https://towardsdatascience.com/machine-learning-with-pyspark-and-amazon-emr-3149dbc847ae

3. Apresentação

- Cada grupo terá no máximo 15 minutos para apresentar o trabalho.
- Todos os membros do grupo deverão estar presentes na data de sua apresentação.
- Todos os grupos deverão estar presentes nos dias de apresentação.
- A banca de avaliação, formada pelos professores das disciplinas, poderá arguir o grupo ou diretamente qualquer membro do grupo.

- A banca é soberana e responsável por resolver os casos previstos nestas orientações e definir os casos omissos.
- As datas das apresentações serão definidas durante o semestre juntamente com professores e alunos de todas as disciplinas.

4. Entregáveis

O que será entregue:

- Relatório (pode ser o notebook ou um github bem detalhado e organizado)
- Apresentação (slides)
- Todos os demais artefatos produzidos (*scripts, architecture workflows, jobs*)

5. Critérios de Avaliação

Por ser um projeto interdisciplinar, a banca definirá, em comum acordo, a nota obedecendo-se a proporção de 40% referente à nota geral do projeto, valorada para cada aluno (20% apresentação oral, 10% slides e 10% arguição) e 60% a cargo de cada professor em sua disciplina.