# Project 2 - Analyzing IMDB Datasets

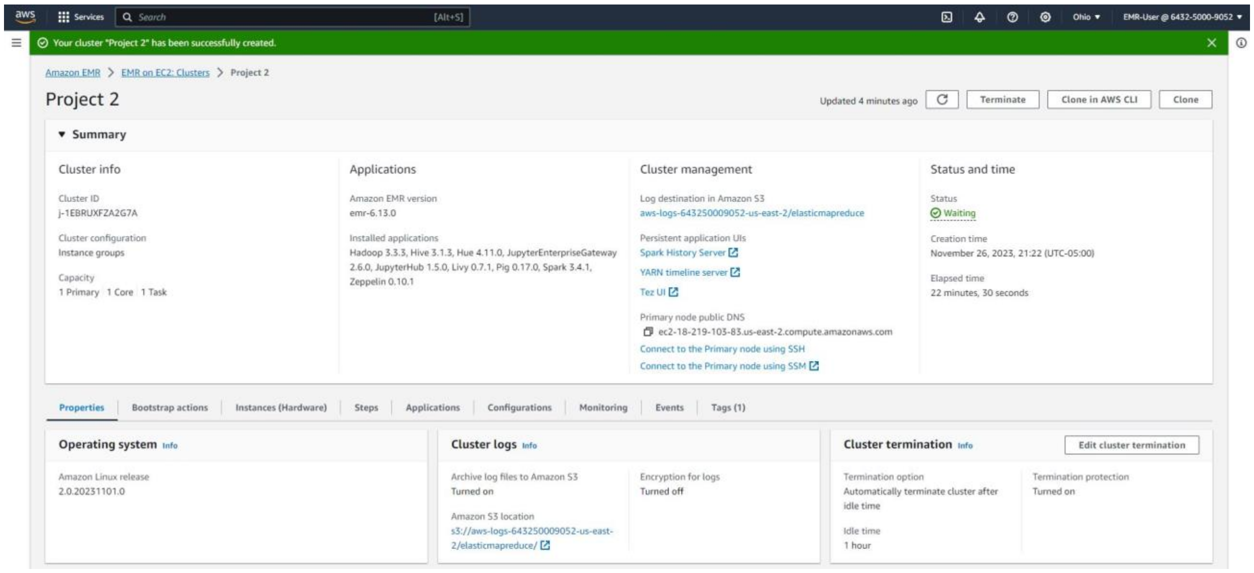
*Alessandro Sciorilli*

## 1. AWS Setup

For this project, I set up a Spark cluster on AWS EMR to load and analyze datasets from IMDB, obtained from Kaggle. I conduct my analysis using a Jupyter Notebook. As a first step, since root users are not authorized to use PySpark or other kernels, I create an IAM user named "Project 2". During this process, I grant the user access to the AWS Management Console by setting up a username and password. Additionally, I attach the necessary policies to this user (policies are:

- *AdministratorAccess*
- *AmazonElasticFileSystemReadOnlyAccess*
- *AmazonEMRFullAccessPolicy_v2*
- *AmazonEMRReadOnlyAccessPolicy_v2*
- *AmazonEMRServicePolicy_v2).*

After signing in to the Console, I create an EMR cluster with the following characteristics:



Your cluster "Project 2" has been successfully created.

Amazon EMR > EMR on EC2: Clusters > Project 2

Project 2

Updated 4 minutes ago

Summary

Cluster info

Cluster ID
j-1EBRUXFZA2G7A

Cluster configuration
Instance groups

Capacity
1 Primary | 1 Core | 1 Task

Applications

Amazon EMR version
emr-6.13.0

Installed applications
Hadoop 3.3.3, Hive 3.1.3, Hue 4.11.0, JupyterEnterpriseGateway 2.6.0, JupyterHub 1.5.0, Livy 0.7.1, Pig 0.17.0, Spark 3.4.1, Zeppelin 0.10.1

Cluster management

Log destination in Amazon S3
aws-logs-643250009052-us-east-2/elasticmapreduce

Persistent application UIs
Spark History Server
YARN timeline server
Tez UI

Primary node public DNS
ec2-18-219-103-83.us-east-2.compute.amazonaws.com
Connect to the Primary node using SSH
Connect to the Primary node using SSM

Status and time

Status
Waiting

Creation time
November 26, 2023, 21:22 (UTC-05:00)

Elapsed time
22 minutes, 30 seconds

Operating system Info

Amazon Linux release
2.0.20231101.0

Cluster logs Info

Archive log files to Amazon S3
Turned on

Amazon S3 location
s3://aws-logs-643250009052-us-east-2/elasticmapreduce/

Encryption for logs
Turned off

Cluster termination Info

Termination option
Automatically terminate cluster after idle time

Idle time
1 hour

Termination protection
Turned on

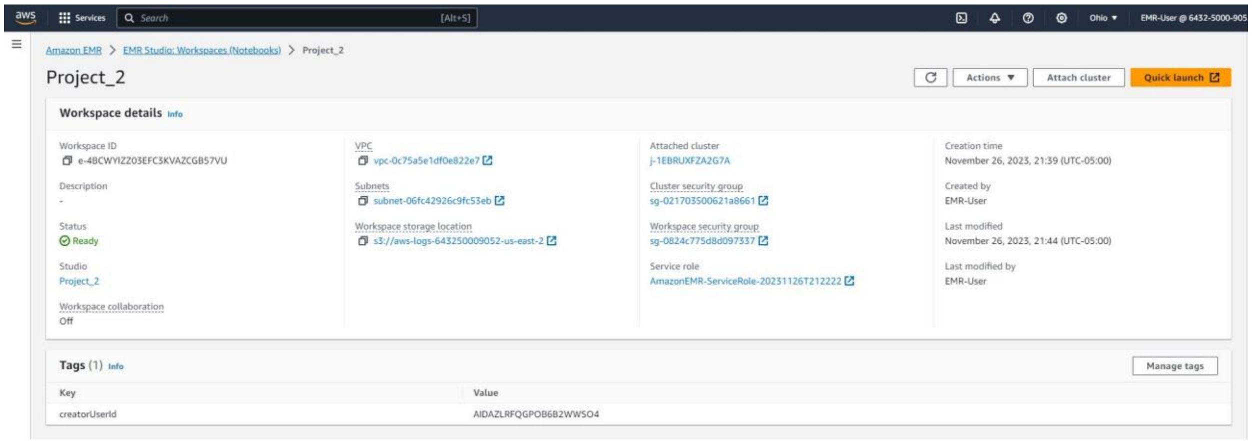
Moreover, I attach the following permission polices:

- *AmazonS3FullAccess*
- *AmazonElasticMapReduceRole*
- *AmazonElasticMapReduceEditorsRole*

As the next step, I set up an EMR Studio: I input the VPC and Subnet of my EMR cluster and select the previously created AWS IAM service role named "Project 2". Then, for "Workspace storage," I choose the S3 bucket that was automatically created when setting up the EMR cluster. After the EMR Studio is created, I create and launch a workspace named "Project 2" within my EMR Studio. Inside this workspace, I attach my EMR cluster. The EMR cluster provides the necessary computational power for data processing and analysis.

Attaching the cluster to the EMR Studio workspace is an essential step since it enables my Notebook to directly utilize the computational power provided by the EMR Cluster, allowing to run large-scale data processing jobs with PySpark.

Below is the screenshot showing the configuration of my notebook:



Project_2

Workspace details Info

| | |
|---|---|
| Workspace ID | e-4BCWYIZZ03EFC3KVAZCGB57VU |
| Description | - |
| Status | Ready |
| Studio | Project_2 |
| Workspace collaboration | Off |
| VPC | vpc-0c75a5e1df0e822e7 |
| Subnets | subnet-06fc42926c9fc53eb |
| Workspace storage location | s3://aws-logs-643250009052-us-east-2 |
| Attached cluster | j-1EBRUXFZA2G7A |
| Cluster security group | sg-021703500621a8661 |
| Workspace security group | sg-0824c775d8d097337 |
| Service role | AmazonEMR-ServiceRole-20231126T212222 |
| Creation time | November 26, 2023, 21:39 (UTC-05:00) |
| Created by | EMR-User |
| Last modified | November 26, 2023, 21:44 (UTC-05:00) |
| Last modified by | EMR-User |

Tags (1) Info

| Key | Value |
|---|---|
| creatorUserId | AIDAZLRFQGPOB6B2WWSO4 |

Finally, I select PySpark and launch a new Jupyter Notebook. To ensure everything is working as expected, I execute **%%info**. I install the **pandas** and **matplotlib** libraries. I import **numpy** for performing calculations**,** **matplotlib.pyplot** for creating charts, and **pyspark.sql.functions** and **types** for manipulating and analyzing my database.

## 2. Analysis of the IMDB Dataset

### Loading Data

I load the datasets into Spark DataFrames from publicly available S3 buckets. I name the dataframes: actors, genres, movie_actors and movie_ratings.

### Data Exploration and Cleaning

The initial step involves exploring the schemas and some records of each DataFrame to understand the data structure. This exploration is crucial for identifying the right columns for joining tables and performing analyses.

### Analyzing Genres

The goal is to identify the number of unique genres and the most popular ones. This required transforming the genre data from a delimited format into a more analysis-friendly form using the explode and split functions. This transformation facilitates the calculation of the number of unique genres and the aggregation of data to find the most popular genres by average ratings.

### Job Categories Analysis

The analysis involves counting distinct job categories and understanding the most common roles in movie production. The findings provide a quantitative view of various job roles and their prevalence in the dataset.

## Query

1) **Find all the "movies" featuring "Johnny Depp" and "Helena Bonham Carter".**
   Three datasets – actors, genres, and movie actors – are joined together after renaming overlapping columns to avoid confusion. The combined dataset is then filtered separately for each actor, focusing on their movie appearances. Finally, these filtered datasets are joined to pinpoint movies where both Depp and Bonham Carter appeared, and the resulting list of movie titles is displayed.

2) **Find all the "movies" featuring "Brad Pitt" after 2010.**
   For this question I utilize Spark's SQL package. I register the previously joined DataFrame (joined_df) as a temporary SQL table. Then I execute a SQL query against this table to select movie titles (primaryTitle) and their release years (startYear). The query specifically targets entries classified as 'movies', featuring 'Brad Pitt', and with a startYear beyond 2010. The results are ordered in descending order of the year.

3) **What is the number of "movies" "acted" by "Zendaya" per year?**
   I use again Spark's SQL to query joined_df. The query selects the year of release (startYear) and counts the number of movies (COUNT(*) as count) in which Zendaya has acted, grouping the results by year. It filters for entries where the titleType is 'movie', the actor's name includes 'Zendaya', and the startYear is not null. The results are ordered in descending order by startYear.

4) **What are the "movies" by average rating greater than "9.7" and released in "2019"?**
   I begin by renaming columns in the movie_ratings DataFrame to avoid duplicate column names during joins. Next, I join genres with movie_ratings and convert the averageRating column to a float type for accurate numerical comparisons. I then execute a SQL query on this joined data, filtering to select movies released in 2019 with an average rating above 9.7. Results are ordered by their ratings in descending order.
5) **Which "actors" have acted in the greatest variety of "genres"?**
   I select the primaryName and count the distinct number of genres for each actor from the joined_df DataFrame. My focus is on entries where the title type is 'movie' and the category is 'actor'. After grouping the results by the actor's name, I order them in descending order based on the count of distinct genres. Finally, I display the results to see which actors have the broadest range of genre appearances, highlighting the versatility of their careers.

6) **Who are the "directors" with the highest "average ratings" among all the "movies" produced? To be considered in the ranking, directors must have produced at least 5 movies.**
   To accomplish this, I first join the joined_df DataFrame with the movie_ratings_renamed DataFrame on the movie identifier column, creating a dataframe called all_joined where all the four datasets are combined. After storing all_joined as a temporary table in my Spark session, I execute a SQL query on it. This query is designed to select the director's name (primaryName) and calculate the average rating (AVG(averageRating)) - average of the average - for each director. I format the average rating to two decimal places. The query filters for entries where the title type is 'movie' and the category is 'director', and it groups the results by the director's name. A key part of the query is the HAVING clause, where I ensure that only directors with more than five distinct movies are considered. Finally, I order the results by the calculated average rating in descending order.

7) **Which "actor-director" pairs have worked together on the most "movies" from 1980?**

I start by filtering the joined_df DataFrame to separate out movies directed by directors and acted in by actors, focusing only on those released after 1980. In this process, I rename the primaryName column to directorName in the director's dataset and to actorName in the actor's dataset for clarity. Next, I further process the actors' movies DataFrame by renaming the tconst column to avoid duplicate column names during the joining process. Then, I join the director and actor DataFrames on the movie identifier (tconst), effectively creating a dataset of collaborations. The key of this analysis involves grouping this joined dataset by both directorName and actorName and then counting the number of collaborations for each pair. I sort these pairs by their collaboration count in descending order to identify the most frequent collaborations**.**