Netflix Movie Recommendation Web application

Cloud Computing Final Project Fall 2020

Ruinan Zhang rz1109
Jiahui Wang jw6273
Wei Si ws1623

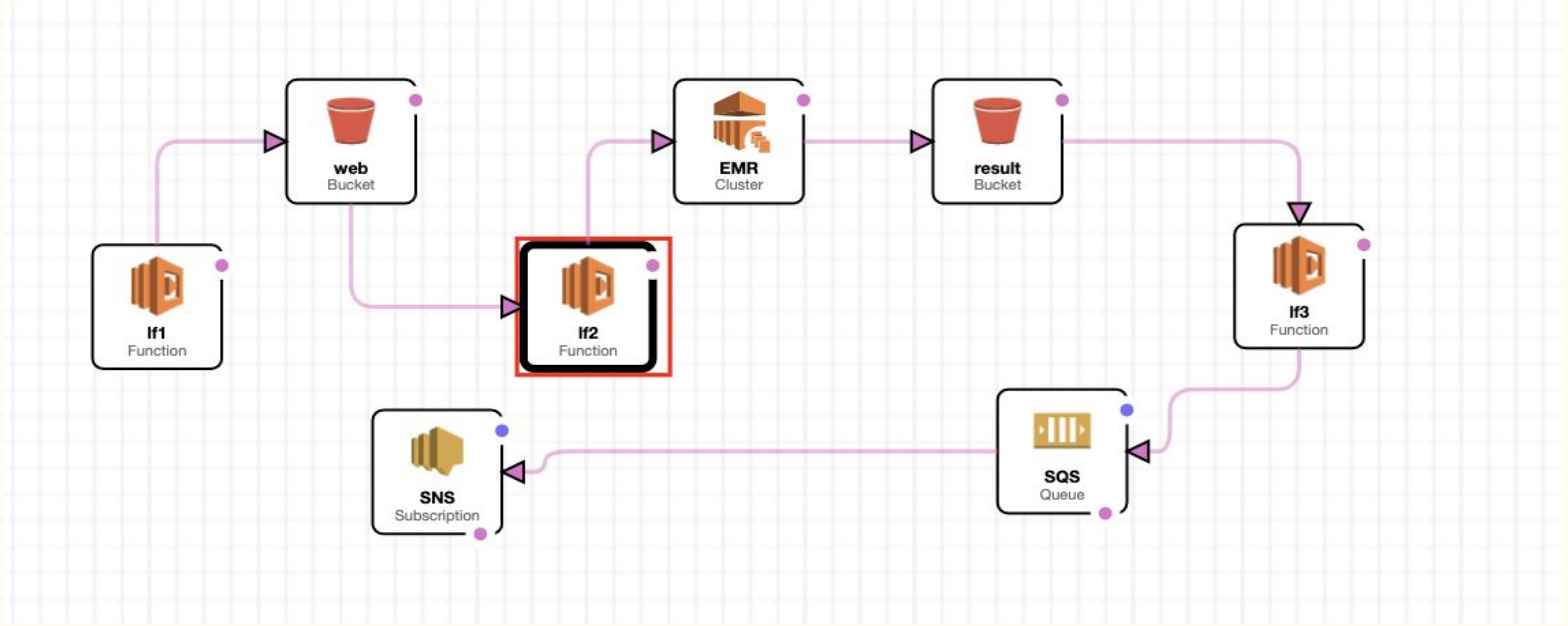
Overview

The project will provide Netflix movie recommendations to the user based on their personalized ratings for movies. Users will be able to select movies they like from the front-end website and then the system will provide customized recommendations for the users using Alternating Least Squares algorithms with the help of PySpark ML. The result of recommendation will be sent to users via text message after the computation is done.

Project Architecture

- Amazon S3 for storage of data and program for cluster and host front-end website
- Amazon EC2 instance for submitting Apache Spark jobs
- Amazon EMR cluster(3 nodes cluster) with Spark 2.4.7 for running Spark jobs
- Alternating Least Squares algorithm (implemented in Apache Spark ML that built for a larges-scale collaborative filtering problems) for recommendation
- DynamoDB for movie details storage and user info storage
- API gateway for connecting front-end to the back-end
- Lambda functions to get query and for each query submit different spark jobs and retrieve data and send them via SNS
- Amazon SNS service for delivering results to the users

Project Structure



Dataset- Netflix Prize data

(https://www.kaggle.com/netflix-inc/netflix-prize-data)

The original movie rating files contain over 100 million ratings from 480 thousand randomly-chosen, anonymous Netflix customers over 17 thousand movie titles. The data were collected between October, 1998 and December, 2005 and reflect the distribution of all ratings received during this period. The ratings are on a scale from 1 to 5 (integral) stars.

Number of Users: 750

Number of Movies: 1,000

Number of Ratings: 420,000

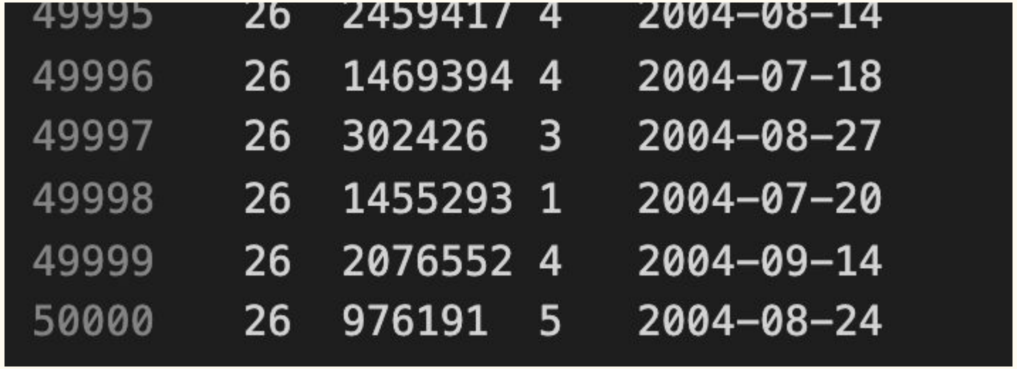
Input Ratings Data File contains (Before Cleaning): movie_id, user_id, ratings, date_of_rating

Input Movie Title file cotains: movie_id, year_of_release, movie_title

Note:* Due to the time and budget limit, we will be using 50,000 ratings instead of 4,20,000 in this demo project.**

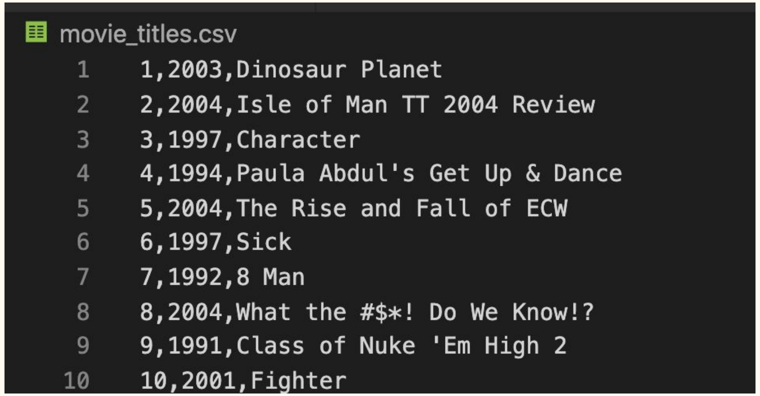
Sample data (after cleaning)

Movie rating data:
Movie_id, user_id, rating, rating date



```
49995      26   2459417  4    2004-08-14
49996      26   1469394  4    2004-07-18
49997      26   302426   3    2004-08-27
49998      26   1455293  1    2004-07-20
49999      26   2076552  4    2004-09-14
50000      26   976191   5    2004-08-24
```

Movie title csv:
Movie_id, year, movie_title



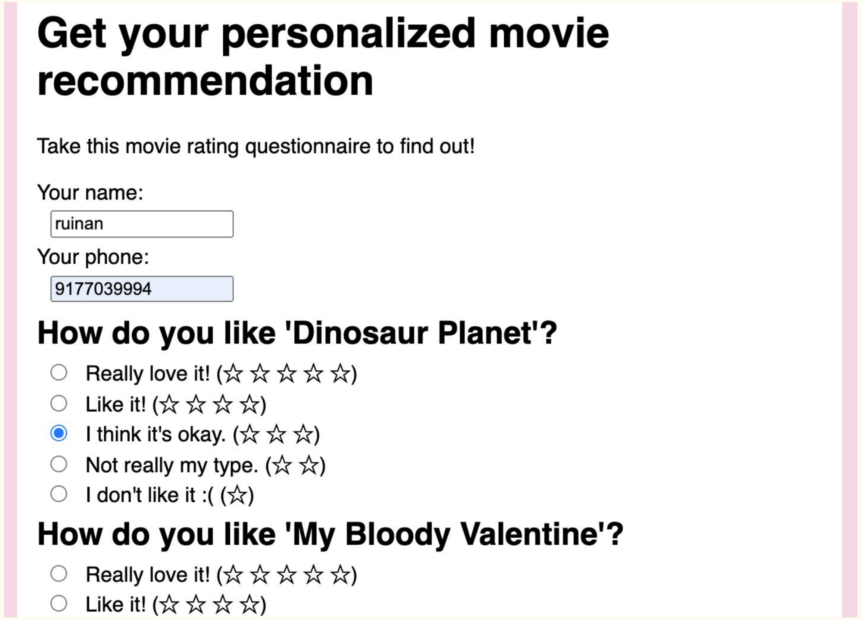
movie_titles.csv

```
1,2003,Dinosaur Planet
2,2004,Isle of Man TT 2004 Review
3,1997,Character
4,1994,Paula Abdul's Get Up & Dance
5,2004,The Rise and Fall of ECW
6,1997,Sick
7,1992,8 Man
8,2004,What the #$*! Do We Know!?
9,1991,Class of Nuke 'Em High 2
10,2001,Fighter
```

Front-end website:

http://cc-final-proj-web.s3-website-us-west-2.amazonaws.com/

- Use questionnaire to collect user's rating on 5 movies. (We planned to use more data but then computation would take too much time, so we decided to do the computation based on 5 ratings, using 26 movies in total)
- Collect user name and phone number for sending out recommendation results
- The data collected from the front-end would be sent to Lambda Function 'proj_insert_rating_py2' via API 'movie-user-input'



Get your personalized movie recommendation

Take this movie rating questionnaire to find out!

Your name: ruinan

Your phone: 9177039994

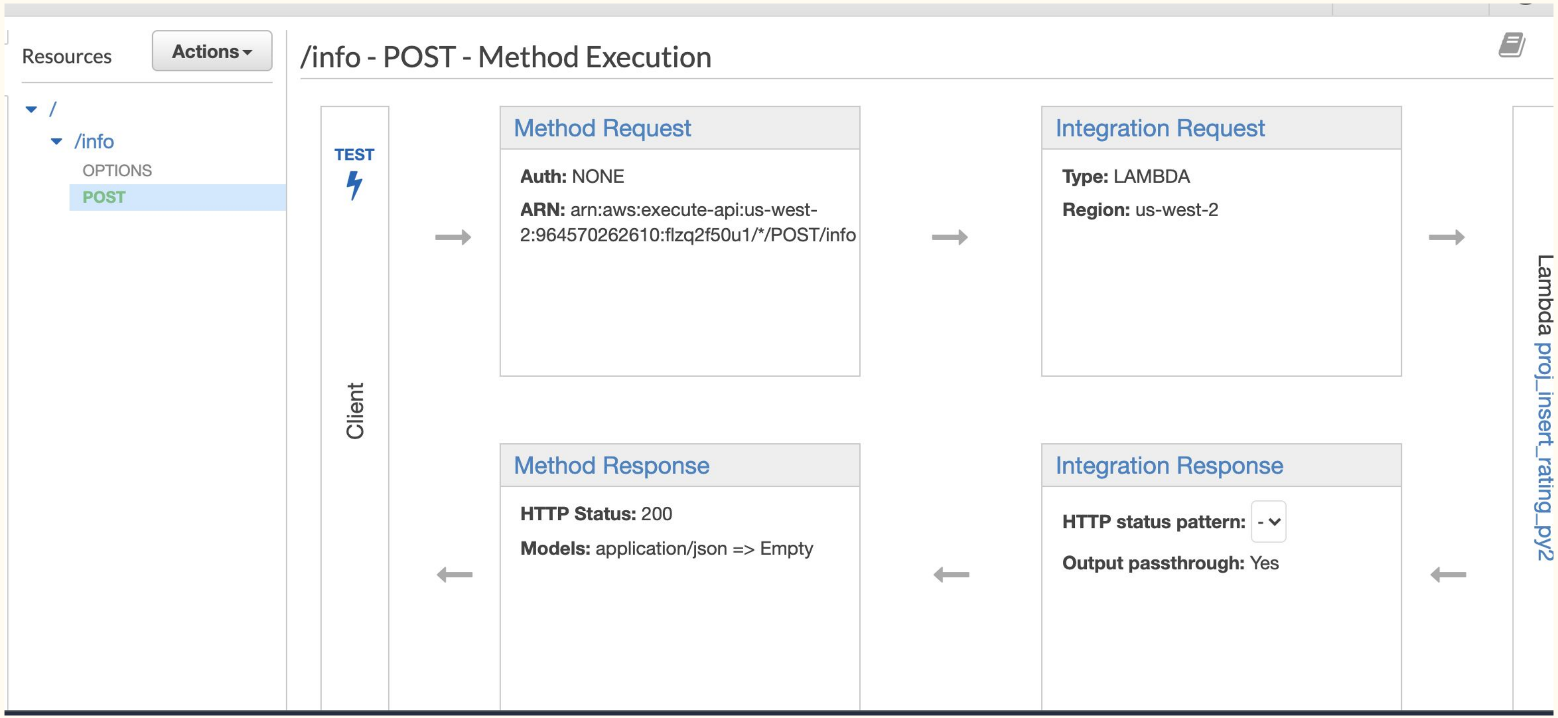
How do you like 'Dinosaur Planet'?

- Really love it! (☆☆☆☆☆)
- Like it! (☆☆☆☆)
- [✓ I think it's okay. (☆☆☆)]
- Not really my type. (☆☆)
- I don't like it :((☆)

How do you like 'My Bloody Valentine'?

- Really love it! (☆☆☆☆☆)
- Like it! (☆☆☆☆)

API ‘movie-user-input’



/info - POST - Method Execution

Method Request

Auth: NONE

ARN: arn:aws:execute-api:us-west-2:964570262610:flzq2f50u1/*/POST/info

Integration Request

Type: LAMBDA

Region: us-west-2

Lambda proj_insert_rating_py2

Method Response

HTTP Status: 200

Models: application/json => Empty

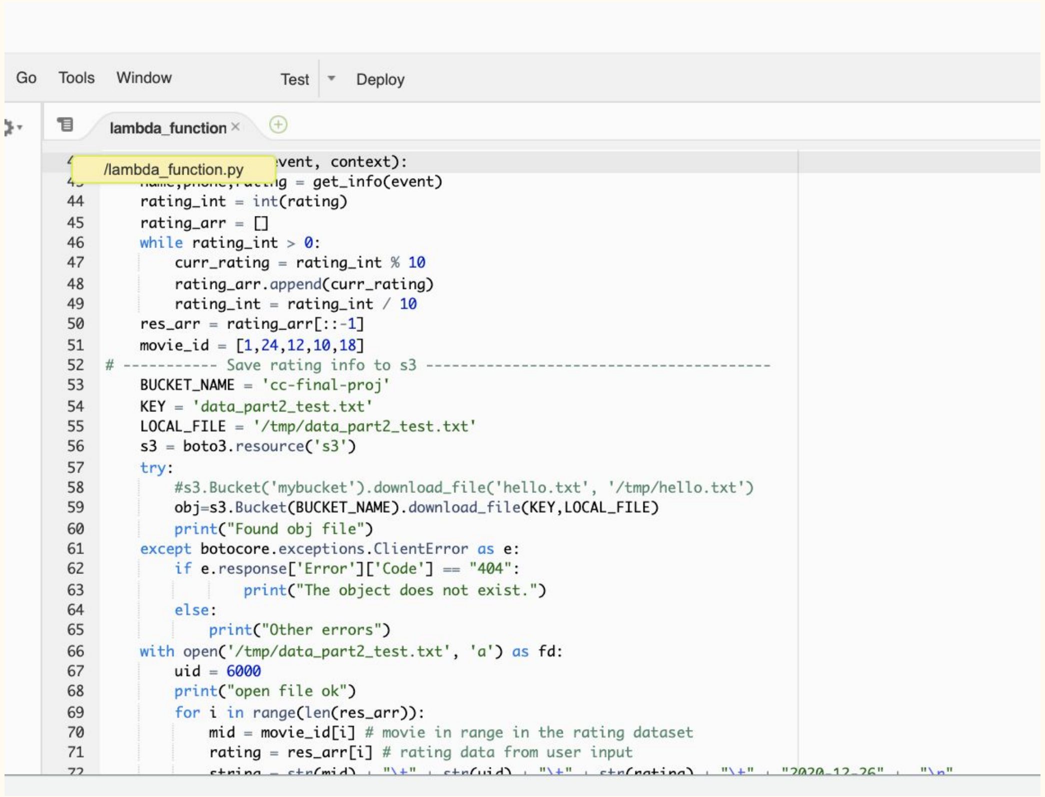
Integration Response

HTTP status pattern: -

Output passthrough: Yes

Lambda Function 1 - ‘proj_insert_rating_py2’

The LF1 will be triggered by API Gateway: movie-user-input and then save the newly collected user rating info to existing using rating data set. At the same time, it will save user’s info and phone number to DynamoDB for the further usage after getting the recommendation results.



```
                vent, context):
    g = get_info(event)
    rating_int = int(rating)
    rating_arr = []
    while rating_int > 0:
        curr_rating = rating_int % 10
        rating_arr.append(curr_rating)
        rating_int = rating_int / 10
    res_arr = rating_arr[::-1]
    movie_id = [1,24,12,10,18]
# ---------- Save rating info to s3 --------------------------------------
    BUCKET_NAME = 'cc-final-proj'
    KEY = 'data_part2_test.txt'
    LOCAL_FILE = '/tmp/data_part2_test.txt'
    s3 = boto3.resource('s3')
    try:
        #s3.Bucket('mybucket').download_file('hello.txt', '/tmp/hello.txt')
        obj=s3.Bucket(BUCKET_NAME).download_file(KEY,LOCAL_FILE)
        print("Found obj file")
    except botocore.exceptions.ClientError as e:
        if e.response['Error']['Code'] == "404":
                print("The object does not exist.")
        else:
            print("Other errors")
    with open('/tmp/data_part2_test.txt', 'a') as fd:
        uid = 6000
        print("open file ok")
        for i in range(len(res_arr)):
            mid = movie_id[i] # movie in range in the rating dataset
            rating = res_arr[i] # rating data from user input
            string = str(mid) + "\t" + str(uid) + "\t" + str(rating) + "\t" + "2020-12-26" + "\n"
```

Lambda Function 2- ‘submit-pyspark-job’

The LF2 will be triggered by S3 bucket: cc-final-proj.

Procedure:

1, User submits a rating data from front end to S3 bucket

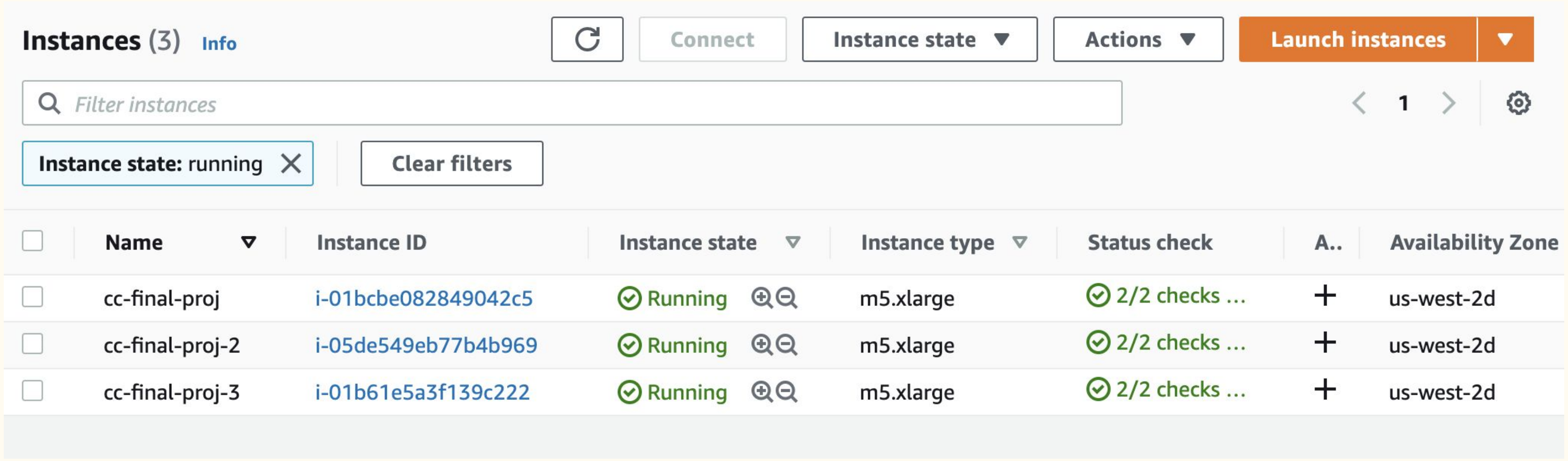
2, Trigger the LF2

3, LF2 submits a pyspark job to the AWS EMR cluster

AWS EMR (Elastic Map Reducer)

The goal we use Amazon EMR is to simplify the big data structure. The EMR cluster we build here includes three ec2 instances:

1 Master node and 2 Core nodes.



Name	Instance ID	Instance state	Instance type	Status check	A..	Availability Zone
cc-final-proj	i-01bcbe082849042c5	Running	m5.xlarge	2/2 checks ...	+	us-west-2d
cc-final-proj-2	i-05de549eb77b4b969	Running	m5.xlarge	2/2 checks ...	+	us-west-2d
cc-final-proj-3	i-01b61e5a3f139c222	Running	m5.xlarge	2/2 checks ...	+	us-west-2d

Recommendation Function: ALSForUid.py

This function runs in the EMR cluster. It takes the user id and rating data as input. Then use the ALS algorithm to get the most related ten movies and save the result into the S3 bucket.

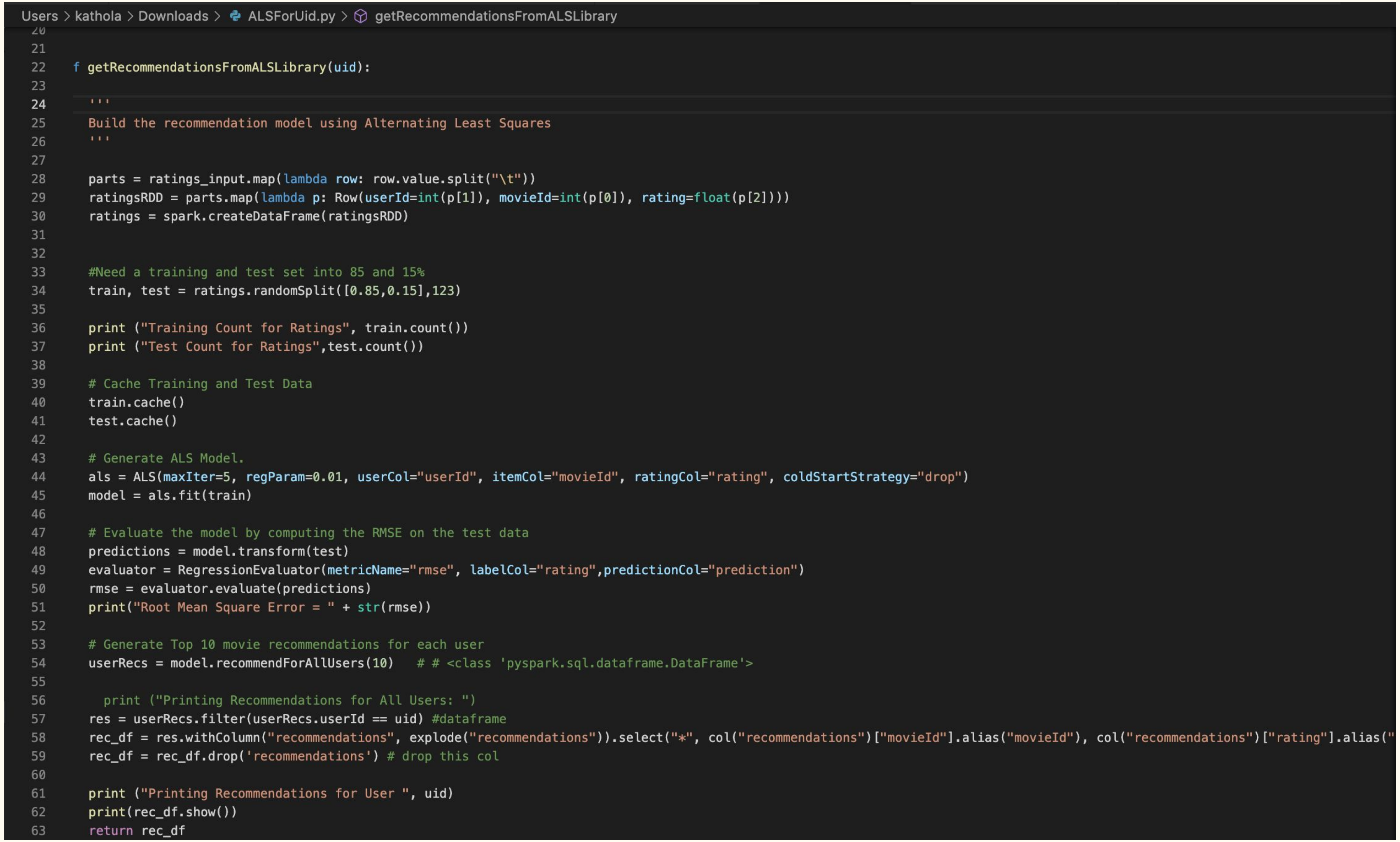
Procedure:

1, Input the user's uid and rating

2, Train the data using ALS model

3, Output the result into S3 bucket: cc-final-result

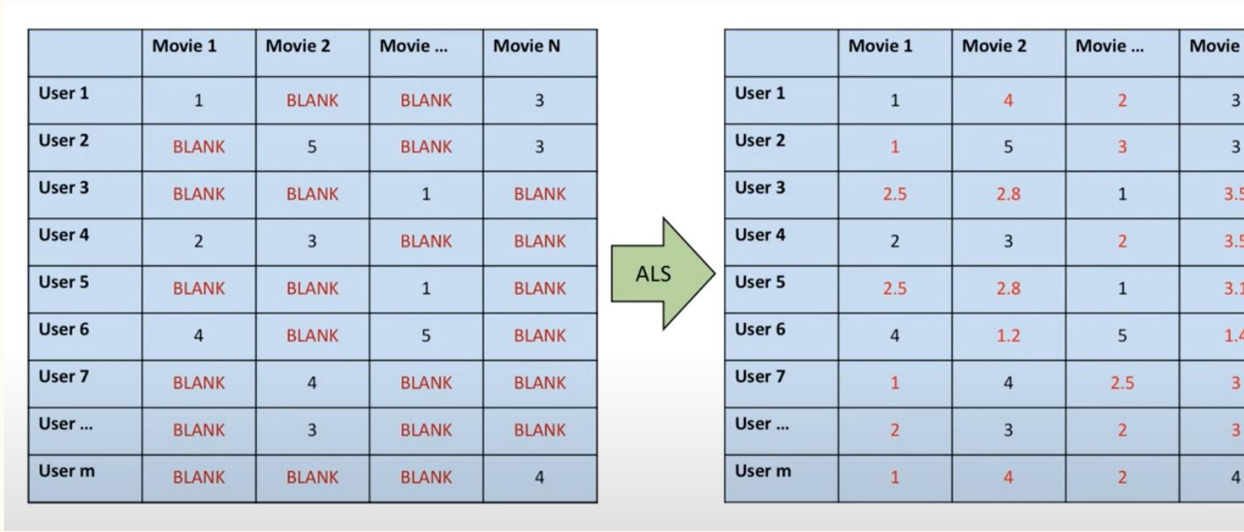
Reference: https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html


Users > kathola > Downloads > ALSForUid.py > getRecommendationsFromALSLibrary

```
 f getRecommendationsFromALSLibrary(uid):

    '''
    Build the recommendation model using Alternating Least Squares
    '''

    parts = ratings_input.map(lambda row: row.value.split("\t"))
    ratingsRDD = parts.map(lambda p: Row(userId=int(p[1]), movieId=int(p[0]), rating=float(p[2])))
    ratings = spark.createDataFrame(ratingsRDD)


    #Need a training and test set into 85 and 15%
    train, test = ratings.randomSplit([0.85,0.15],123)

    print ("Training Count for Ratings", train.count())
    print ("Test Count for Ratings",test.count())

    # Cache Training and Test Data
    train.cache()
    test.cache()

    # Generate ALS Model.
    als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating", coldStartStrategy="drop")
    model = als.fit(train)

    # Evaluate the model by computing the RMSE on the test data
    predictions = model.transform(test)
    evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",predictionCol="prediction")
    rmse = evaluator.evaluate(predictions)
    print("Root Mean Square Error = " + str(rmse))

    # Generate Top 10 movie recommendations for each user
    userRecs = model.recommendForAllUsers(10)   # # <class 'pyspark.sql.dataframe.DataFrame'>

      print ("Printing Recommendations for All Users: ")
    res = userRecs.filter(userRecs.userId == uid) #dataframe
    rec_df = res.withColumn("recommendations", explode("recommendations")).select("*", col("recommendations")["movieId"].alias("movieId"), col("recommendations")["rating"].alias("
    rec_df = rec_df.drop('recommendations') # drop this col

    print ("Printing Recommendations for User ", uid)
    print(rec_df.show())
    return rec_df
```

ALS Algorithm

ALS: Alternating Least Squares



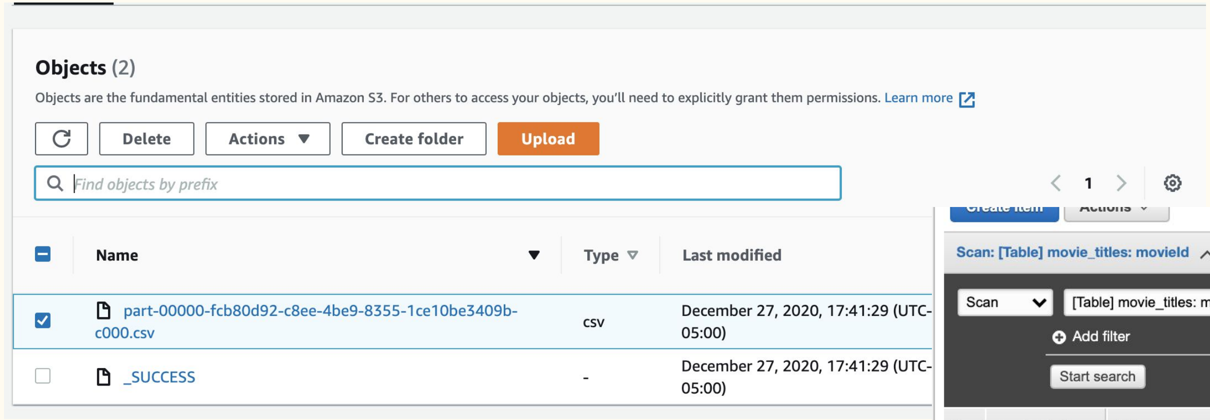
	Movie 1	Movie 2	Movie ...	Movie N
User 1	1	BLANK	BLANK	3
User 2	BLANK	5	BLANK	3
User 3	BLANK	BLANK	1	BLANK
User 4	2	3	BLANK	BLANK
User 5	BLANK	BLANK	1	BLANK
User 6	4	BLANK	5	BLANK
User 7	BLANK	4	BLANK	BLANK
User ...	BLANK	3	BLANK	BLANK
User m	BLANK	BLANK	BLANK	4

	Movie 1	Movie 2	Movie ...	Movie
User 1	1	4	2	3
User 2	1	5	3	3
User 3	2.5	2.8	1	3.5
User 4	2	3	2	3.5
User 5	2.5	2.8	1	3.1
User 6	4	1.2	5	1.4
User 7	1	4	2.5	3
User ...	2	3	2	3
User m	1	4	2	4

$$P_u := P_u - \alpha \cdot \frac{\partial L}{\partial P_u} = P_u - \alpha \cdot [\, \textstyle\sum_i 2(P_u^T Q_i - R_{ui}) Q_i + 2\lambda P_u]$$

$$Q_i := Q_i - \alpha \cdot \frac{\partial L}{\partial Q_i} = Q_i - \alpha \cdot [\, \textstyle\sum_u 2(P_u^T Q_i - R_{ui}) P_u + 2\lambda Q_i]$$

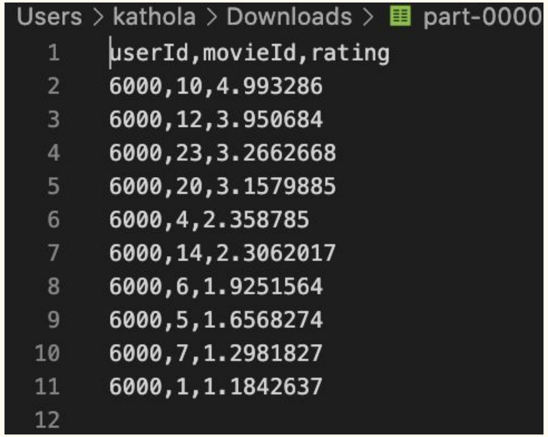
Spark Job Results



Objects (2)

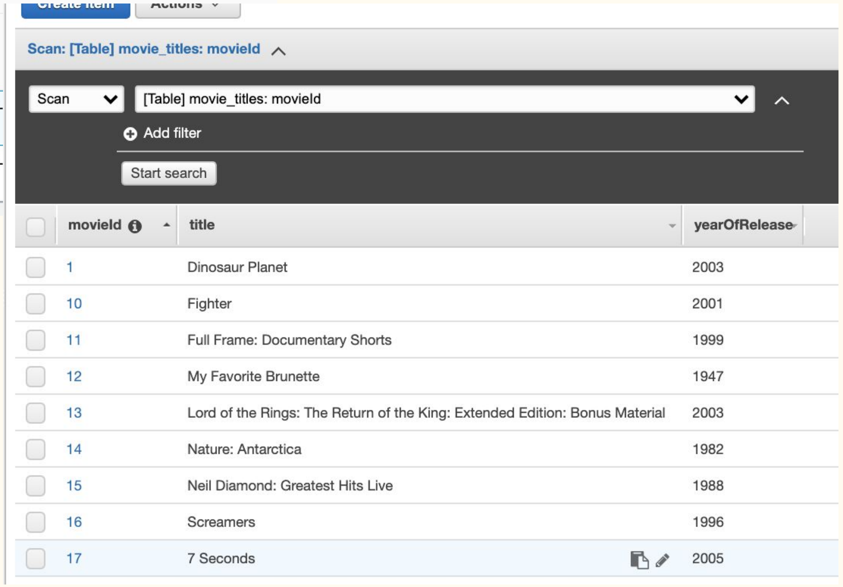
Objects are the fundamental entities stored in Amazon S3. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified
part-00000-fcb80d92-c8ee-4be9-8355-1ce10be3409b-c000.csv	csv	December 27, 2020, 17:41:29 (UTC-05:00)
_SUCCESS	-	December 27, 2020, 17:41:29 (UTC-05:00)



Users > kathola > Downloads > part-0000

```
userId,movieId,rating
6000,10,4.993286
6000,12,3.950684
6000,23,3.2662668
6000,20,3.1579885
6000,4,2.358785
6000,14,2.3062017
6000,6,1.9251564
6000,5,1.6568274
6000,7,1.2981827
6000,1,1.1842637
```



Scan: [Table] movie_titles: movieId

movieId	title	yearOfRelease
1	Dinosaur Planet	2003
10	Fighter	2001
11	Full Frame: Documentary Shorts	1999
12	My Favorite Brunette	1947
13	Lord of the Rings: The Return of the King: Extended Edition: Bonus Material	2003
14	Nature: Antarctica	1982
15	Neil Diamond: Greatest Hits Live	1988
16	Screamers	1996
17	7 Seconds	2005

Spark job's results will return recommendation for the user, and then we use the moiveID to locate more detailed info about the movie in DynamoDB

Lambda Function 3- ‘proj_send_rec_py2’

This function is triggered by the S3 bucket cc-final-result.

Procedure:

1, S3 bucket cc-final-result receives the updating recommendation result

2, Trigger the LF3

3, LF3 submit the sending message request to AWS SQS.

4, SQS triggers the AWS SNS service and sends text message to user.

Individual Contributions

Ruinan Zhang: Front-end website / APIGateway / LF1

Wei Si: Spark job Submission / LF2 / EMR cluster creation

Jiahui Wang: ALS algorithm development / LF3 / SNS service