Code Logic: Online Advertising

Problem statement:

With increasing digitization, the online advertising sector is experiencing a major boom as more and more companies pay large amounts of money to reach their customers through online platforms
In this project, we will build an online advertising platform to facilitate the connection between businesses and their target audiences in an efficient, targeted, and measurable manner.

Solution Approach:

We will be creating a database and an interface for campaign managers to manage ad campaigns and another interface for clients to display ads and report user activity to the advertising platform.

Step 1:

Creating database and exporting data:
Download this csv file in local of emr cluster
Wget https://de capstone-project1.s3.amazonaws.com/users_500k.csv
To connect to MySQL-RDS:
mysql -h database-1.cxueuenwsllg.us-east-1.rds.amazonaws.com -u admin -p

Built the "served_ads," "ads," and "users" tables in the Advertising database.
Users -- We will use the user table to store the user data. The data to be loaded into this table is taken from https://de capstone-project1.s3.amazonaws.com/users_500k.csv
Ads -- The ad information will be in the ads table. The information in the table, along with certain derived attributes like campaign type, status, CPM, and current slot budget, will be retrieved from the Kafka queue using ad_manager script.
served_ads -- served_ads will hold the auction of Ads for displaying the Ad to a user device

Step2:

Ad Manager using PyKafka:

- Ad campaign events are read from Upgrad's Kafka central repository, and the primary job of the ad_manager is to process all kinds of events, including New, Update, and Stop campaign events.
- We will be using MYSQL connector to connect to the database and write the data in the ads table.
- Used pykafka to read the data from the kafka queue.
- The script additionally generates additional parameters such as Status, Slot Budget for every campaign, and CPM

To Run ad_manager:

```
sudo su

sudo yum update
export SPARK_KAFKA_VERSION=0.10

wget https://ds-spark-sql-kafka-jar.s3.amazonaws.com/spark-sql-kafka-0-10_2.11-2.3.0.jar

pip install mysql-connector-python==8.0.12 --user

pip install pykafka==2.8.0 --user
pip install Flask --user
pip install urllib3==1.26.6 --user
Pip install requests  --user
```

Command syntax to run **ad_manager** script

```
ad_manager.py <kafka_bootstrap_server> <kafka_topic> <database_host>
<database_username> <database_password> <database_name>"
```

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 ad_manager.py
18.211.252.152:9092 de-capstone1 database-1.cxueuenwsllg.us-east-1.rds.amazonaws.com
admin 12345678 advertisment_db
```

Step3

Ad_server

- Ad_server transmits the request and user data while the user is online and engaged. The ad_server will obtain the list of advertisements upon receiving the request before starting the auction.
- Here we are using the Flask library to create a web server and serve APIs in Python.
- Next, the **user simulator script** will make similar requests to the Ad Server's hosted APIs as if it were a real user. It will first make a request to the Ads API, and then it will make a request to the Feedback API using the request identifier obtained from the previous API call

To run ad_server:

```
python ad_server.py database-1.cxueuenwsllg.us-east-1.rds.amazonaws.com admin 12345678
advertisment_db
```

To run user_simulator:

```
python user_simulator.py database-1.cxueuenwsllg.us-east-1.rds.amazonaws.com admin 12345678
advertisment_db http 0.0.0.0 5000 0.0.0.0 8000
```