Big Data Application Final Project - FIFA World Cup

Github Repository link:
https://github.com/FarFlyField/Big_Data_Application_Architecture_Final_Project
Link to dataset:
https://www.kaggle.com/datasets/tadhgfitzgerald/fifa-international-soccer-mens-ranking-1993now
Link that explains FIFA ranking rules:
https://en.wikipedia.org/wiki/FIFA_Men%27s_World_Ranking

Introduction: Find the data

I like soccer a lot, so I decided to work on something related to soccer. Since the world cup is coming next year, I want to build something related to soccer country teams. After searching around online, I saw most datasets are related to players, not teams. However, after searching for an hour and half, I found a csv file that records the data about each world cup team since 1993. One small issue with this dataset is that some columns only have zeros before 8/24/2011, because there has been some changes since that year as quoted below:

Wikipedia:
“In the 2010s, teams realised the ranking system could be 'gamed', specifically by avoiding playing non-competitive matches, particularly against weaker opponents.[32] This was because the low weighting of friendlies meant that even victories could reduce a team's average score: in other words, a team could win a match and lose points.”

Therefore, I will keep that in mind and be careful with how I run aggregations. The file is 5MB (16 columns * 57793 rows), which is not large because the world cup only plays every 4 years, but I will try to extract useful information from the dataset using our cluster. Although world cup is held every 4 years, country teams play with other teams in each confederations and hold friendly matches sometimes, so the rank of countries actually has some updates every year.

The dataset has important information like rank, total points of each counter, the rank change, etc. These data can help us see which teams are strong and strong consistently. We can search countries and see how they perform since 1993. We can also do things like “Which team is consistently strong?” For the speed layer, I will see if I can let users submit new results (fake results, assumed by users) on teams so that they can get new information based on their

alternative knowledge. In fact, people can submit new results from games in 2026 for submissions.

The following steps explain how the app is built and everything tried. Not all the tables and materials are used for the final video demo, but all were useful in helping me understand the dataset and constructing the right architecture. I recorded my video on 12/6/2025, it's just 1 minute that you can download to watch.

Step 1: Put the data onto the cluster

Download the csv file on my own laptop first.

Now let's first create a directory on cluster:
mkdir -p ~/wuh_fifa_project

Move the csv file to the cluster:
scp /where_it_is_on_my_laptop/fifa_ranking.csv
hadoop@ec2-34-230-47-10.compute-1.amazonaws.com:~/wuh_fifa_project/

Next, we want to move it to HDFS.

We first create an HDFS directory for the raw CSV:
hdfs dfs -mkdir -p /inputs/wuh_fifa

Move it from cluster to HDFS:
hdfs dfs -put ~/wuh_fifa_project/fifa_ranking.csv /inputs/wuh_fifa/

Step 2: Create a Hive table

Now I want to create a table in Hive so that I can write further queries and put things to HBase. We learnt Hive from lecture 3, so I mainly referenced files from there.

My data has total of 16 columns:
rank
country_full
country_abrv
total_points
previous_points
rank_change
cur_year_avg
cur_year_avg_weighted
last_year_avg
last_year_avg_weighted
two_year_ago_avg

two_year_ago_weighted
three_year_ago_avg
three_year_ago_weighted
confederation
rank_date

The columns that are marked in bold have only zeros before 8/24/2011. But we will create our tables with them in the first place, we need to keep them in mind during aggregation, filtering them out in aggregation.

Again, they are:
total_points
cur_year_avg
cur_year_avg_weighted
last_year_avg
last_year_avg_weighted
two_year_ago_avg
two_year_ago_weighted
three_year_ago_avg
Three_year_ago_weighted

Create an external table for my data lake. I named it wuh_fifa_rankings_csv.

Open hive: hive

Drop the table since the csv file if the table is wrong:
hive> DROP TABLE IF EXISTS wuh_fifa_rankings_csv;

Create the table:
hive>
```
CREATE EXTERNAL TABLE IF NOT EXISTS wuh_fifa_rankings_csv (
    rank INT,
    country_full STRING,
    country_abrv STRING,
    total_points DOUBLE,
    previous_points DOUBLE,
    rank_change INT,
    cur_year_avg DOUBLE,
    cur_year_avg_weighted DOUBLE,
    last_year_avg DOUBLE,
    last_year_avg_weighted DOUBLE,
    two_year_ago_avg DOUBLE,
    two_year_ago_weighted DOUBLE,
    three_year_ago_avg DOUBLE,
```

```
    three_year_ago_weighted DOUBLE,
    confederation STRING,
    rank_date STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/inputs/wuh_fifa'
TBLPROPERTIES ("skip.header.line.count"="1");
```

We must jump over the first row because the header of the csv file is the first row.

Check if the table is there (correct):
```
hive> SELECT * FROM wuh_fifa_rankings_csv LIMIT 10;
```

Check the header (correct):
```
hive> DESCRIBE wuh_fifa_rankings_csv;
```

All checked, now we have moved the csv to a hive table on our cluster.

Step 2: Convert to an ORC format
If we still have the world cup in a thousand years from now, we will have more data. Therefore, we make it ORC to gain efficiency for reading columns.

Let's create: wuh_fifa_rankings_orc
```
CREATE TABLE wuh_fifa_rankings_orc
STORED AS ORC
AS
SELECT * FROM wuh_fifa_rankings_csv;
```

With the ORC format, it would take me less time to create my aggregation tables if the dataset is much larger. The goal here is to keep the solution scalable.

Step 3: Aggregation Tables

We want to have some meaningful tables that can help us get the most out of the dataset. We need to keep notice of the following things:
1. Our work is across the entire 1993–2018 dataset.
2. Handle pre-2011 zeros cleanly.
3. Meaningful results that can be queried by year/team throughout time.

Not all the tables will be used in the final batch view, but I would like to build them and query them briefly to see which table will be most meaningful.

I therefore come up with 3 tables first, and only 1 of them will be made to HBase:

1. wuh_fifa_yearly_team_stats
2. wuh_fifa_team_batch_stats
3. wuh_fifa_confed_yearly_stats

With wuh_fifa_yearly_stats, we then will be able to know in each year, how did the teams perform:

```
CREATE TABLE wuh_fifa_yearly_team_stats AS
SELECT
   country_full AS team,
   confederation,
   year(to_date(rank_date)) AS year,
   MIN(rank) AS best_rank_this_year,
   MAX(rank) AS worst_rank_this_year,
   AVG(rank) AS avg_rank_this_year,
   AVG(ABS(rank_change)) AS avg_rank_volatility_this_year
FROM wuh_fifa_rankings_orc
GROUP BY
   country_full,
   confederation,
   year(to_date(rank_date));
```

wuh_fifa_yearly_team_stats can help me with questions like:

- “Which team was strongest in 2015?”
- “Which confederation dominated in 2000?”
- “Which teams improved from 2005 → 2010?”

With wuh_fifa_team_batch_stats, we get to get a very aggregated result that shows performance throughout the time:

```
CREATE TABLE wuh_fifa_team_batch_stats AS
WITH confed AS (
   SELECT
      confederation,
      AVG(CASE WHEN total_points > 0 THEN total_points END) AS confed_avg_points
   FROM wuh_fifa_rankings_orc
   GROUP BY confederation
)
SELECT
   r.country_full AS team,
   r.confederation,
   MIN(r.rank) AS best_rank,
   MAX(r.rank) AS worst_rank,
   AVG(ABS(r.rank_change)) AS avg_abs_rank_change,
   MIN(year(to_date(r.rank_date))) AS first_year_active,
```

```
    MAX(year(to_date(r.rank_date))) AS last_year_active,
    COUNT(DISTINCT year(to_date(r.rank_date))) AS total_years_ranked,
    c.confed_avg_points
FROM wuh_fifa_rankings_orc r
LEFT JOIN confed c
    ON r.confederation = c.confederation
GROUP BY
    r.country_full, r.confederation, c.confed_avg_points;
```

It has the following features:

- Zero values before 2011 don't harm aggregates.
- Team-level stability questions become easy.
- Confederation comparisons work.

There are 6 confederations in FIFA world cup (AFC (Asia), CAF (Africa), CONCACAF (North, Central America, and the Caribbean), CONMEBOL (South America), OFC (Oceania), and UEFA (Europe). We then have a table named wuh_fifa_confed_yearly_stats:

```
CREATE TABLE wuh_fifa_confed_yearly_stats AS
SELECT
    confederation,
    year(to_date(rank_date)) AS year,
    AVG(rank) AS avg_rank_confed,
    COUNT(*) AS num_teams_ranked
FROM wuh_fifa_rankings_orc
GROUP BY confederation, year(to_date(rank_date));
```

This allows me to quickly get results like:

- "Which confederation was strongest in 2002?"
- "How did UEFA's strength evolve over time?"

Up until now, we have 2 raw tables:

- wuh_fifa_rankings_csv
- Wuh_fifa_rankings_orc

We then generated 3 more tables:

- wuh_fifa_yearly_team_stats
- wuh_fifa_team_batch_stats
- wuh_fifa_confed_yearly_stats

Step 3: Move to HBase

Let's get the serving layer ready with what we had in lecture 4 so that my Node.js app will query HBase's REST API. After checking the tables, I think wuh_fifa_team_batch_stats is not only the most meaningful but also the one that resembles what we had in our flight app. The HBase batch view has one row per team and row key using team name.

We do this: wuh_fifa_team_batch_stats (Hive) -> wuh_fifa_team_stats (HBase)

By lecture 4 we create it as following:
hbase shell
create 'wuh_fifa_team_stats', 'stats'
Under wuh_fifa_project on cluster:
nano wuh_write_to_hbase.hql

The file wuh_write_to_hbase.hql is in repo, quite similar to what we had in lec_4_files's write_to_hbase.hql.

Run:
hive -f ~/wuh_fifa_project/wuh_write_to_hbase.hql

Check the HBase (correct):
base:001:0> scan 'wuh_fifa_team_stats', {LIMIT => 10}

Step 4: Work on the app that connects to HBase

I have modified the original app to connect with what I have.
The key files are:
app.js, team_result.mustache, public/index.html, submit-event.html.

One issue I found is that if you search China, it will display an HBase Error because China is named as China PR in the database. To resolve this issue I use "hclient.table('wuh_fifa_team_stats').scan({ maxVersions: 1 }" so that I can get all the countries' names and put them on a separate page.

So there are 2 ways to see the results.

1. Go to "View all Available Countries" and browse for the country you are looking for there.
2. Know the country name like how we knew the airport name and entered it.

To verify the correctness, I search Kosovo and press enter, and I will see its first ranking is in 2016:

Stats for Kosovo

Statistic	Value
Best Rank	141
Worst Rank	190
First Ranking Year	2016
Last Ranking Year	2018
Total Years Ranked	3
Avg Absolute Rank Change	4.208333333333333
Confederation Avg Points	651.9654763536685

Back

This is because Kosovo joined FIFA in 2016. This verifies the correctness of my HBase table.

The submit-event.html will let users enter rank and points. For now, it will tell you it's submitted but nothing actually happens. After we get to the speed layer, we will have it working and displayed:

1. Create a new country and see if we can search it.
2. Update the original data with a large number and see if anything changes.

Step 4: Build Speed Layer

HBase table for speed layer. Spark streaming will populate it.
In HBase shell:
create 'wuh_fifa_events_log', 'e'

The wuh_fifa_events_log is just for log. Our spark task will also update wuh_fifa_team_stats.

We name the kafka topic: wuh_fifa_events
Due to some version issue, I was only able to create my kafka topic with a python file named create-topic.py which I put in my repo.
The original way of starting the kafka topic which didn't work for me:

```
kafka-topics.sh --create \
  --bootstrap-server
boot-publicbyg.mpcs53014kafka.2siu49.c2.kafka.us-east-1.amazonaws.com:9196 \
  --command-config ~/kafka.client.properties \
  --replication-factor 3 \
  --partitions 3 \
  --topic wuh_fifa_events
```

app.js updates:
We need kafkajs to do this. The app.js is like the app we have in lec_8_files, meaning we need to submit it to our kafka topic.

For streaming, we take speed-layer-weather-archetype and replace the scala file with StreamFifaEvents.scala and deploy it. Remember to not leave it running. I submitted the uber jar to wuh_fifa_project. I had it named untitled because I would have to change ti many times and rename it every time is wasting my time. I have the StreamFifaEvents.scala file in the repo which one can check. I also updated the pom.xml file's spark dependency to match with what we have on cluster on 12/6/2025 as I saw some errors reported about dependencies.

Checked Kafka as I submitted Brazil as my event:

```
[hadoop@ip-172-31-91-77 bin]$ kafka-console-consumer.sh \
  --bootstrap-server
boot-public-byg.mpcs53014kafka.2siu49.c2.kafka.us-east-1.amazonaws.com:9196 \
  --consumer.config ~/kafka.client.properties \
  --topic wuh_fifa_events \
  --from-beginning
{"country":"Brazil","new_rank":100,"points":200,"ts":1765063390411}
```

My speed-layer handles errors and successfully updates what's in kafka topic to HBase.
In /home/hadoop, referenced from lecture 8, run:

```
spark-submit \
  --driver-java-options "-Dlog4j.configuration=file:///home/hadoop/log4j.properties" \
  --master local[2] \
  --class StreamFifaEvents \
  --files /etc/hbase/conf/hbase-site.xml \
  wuh_fifa_project/target/uber-untitled-1.0-SNAPSHOT.jar \
  b-3-public.mpcs53014kafka.2siu49.c2.kafka.us-east-1.amazonaws.com:9196
```

To show and to kill spark:

```
yarn application -list
yarn application -kill application_id
```

Check logs of spark (Updated is my log in StreamFifaEvents.scala:

```
yarn logs -applicationId application_1763001378295_1138 | grep Updated
```

I used the following commands to check my updates to HBase and found successful results:

hbase:008:0> get 'wuh_fifa_team_stats', 'Brazil'

```
COLUMN                          CELL
 stats:avg_abs_rank_change          timestamp=2025-12-05T06:37:26.535, value=0.5
 stats:best_rank                   timestamp=2025-12-05T06:37:26.535, value=1
 stats:confed_avg_points             timestamp=2025-12-05T06:37:26.535,
value=929.0054096385551
 stats:first_year_active            timestamp=2025-12-05T06:37:26.535, value=1993
 stats:last_year_active              timestamp=2025-12-05T06:37:26.535, value=2018
```

```
 stats:latest_live_points              timestamp=2025-12-07T05:45:18.566,
value=@\x81X\x00\x00\x00\x00\x00
 stats:latest_live_rank                timestamp=2025-12-07T05:45:18.566, value=\x00\x00\x00o
 stats:latest_live_ts                  timestamp=2025-12-07T05:45:18.566,
value=\x00\x00\x01\x9A\xF6C5\xB7
 stats:total_years_ranked              timestamp=2025-12-05T06:37:26.535, value=26
 stats:worst_rank                      timestamp=2025-12-05T06:37:26.535, value=22
1 row(s)
Took 0.0072 seconds
```

hbase:010:0> scan 'wuh_fifa_events_log'

```
ROW                                   COLUMN+CELL
 Brazil#1765063390411                 column=e:points, timestamp=2025-12-07T05:45:18.561,
value=@i\x00\x00\x00\x00\x00\x00
 Brazil#1765063390411                 column=e:rank, timestamp=2025-12-07T05:45:18.561,
value=\x00\x00\x00d
 Brazil#1765063390411                 column=e:ts, timestamp=2025-12-07T05:45:18.561,
value=\x00\x00\x01\x9A\xF5\xF9\xF4\xCB
 Brazil#1765064093954                 column=e:points, timestamp=2025-12-07T05:45:18.410,
value=@r\xC0\x00\x00\x00\x00\x00\x00
 Brazil#1765064093954                 column=e:rank, timestamp=2025-12-07T05:45:18.410,
value=\x00\x00\x00\x96
 Brazil#1765064093954                 column=e:ts, timestamp=2025-12-07T05:45:18.410,
value=\x00\x00\x01\x9A\xF6\x04\xB1\x02
 Brazil#1765066744671                 column=e:points, timestamp=2025-12-07T05:45:18.561,
value=@t\xD0\x00\x00\x00\x00\x00\x00
 Brazil#1765066744671                 column=e:rank, timestamp=2025-12-07T05:45:18.561,
value=\x00\x00\x00\xDE
 Brazil#1765066744671                 column=e:ts, timestamp=2025-12-07T05:45:18.561,
value=\x00\x00\x01\x9A\xF6-#_
 Brazil#1765068191159                 column=e:points, timestamp=2025-12-07T05:45:18.565,
value=@\x81X\x00\x00\x00\x00\x00\x00
 Brazil#1765068191159                 column=e:rank, timestamp=2025-12-07T05:45:18.565,
value=\x00\x00\x00o
 Brazil#1765068191159                 column=e:ts, timestamp=2025-12-07T05:45:18.565,
value=\x00\x00\x01\x9A\xF6C5\xB7
```

Step 5: Deploy the web app

Use what we have in lecture 8 to copy the app to the server.

My port is 3001:

pm2 start app.js --name wuh_app -- 3001

http://ec2-34-230-47-10.compute-1.amazonaws.com:8070/

I have successfully put it on port 3001 and can open at:
http://ec2-52-20-203-80.compute-1.amazonaws.com:3001/

To remove it:
pm2 delete wuh_app

To check logs:
pm2 logs wuh_app

The app running on the webserver shows the result for the speed layer for Kosovo and Brazil which were countries I chose to submit the event for. The demo of speed layer is in the video.