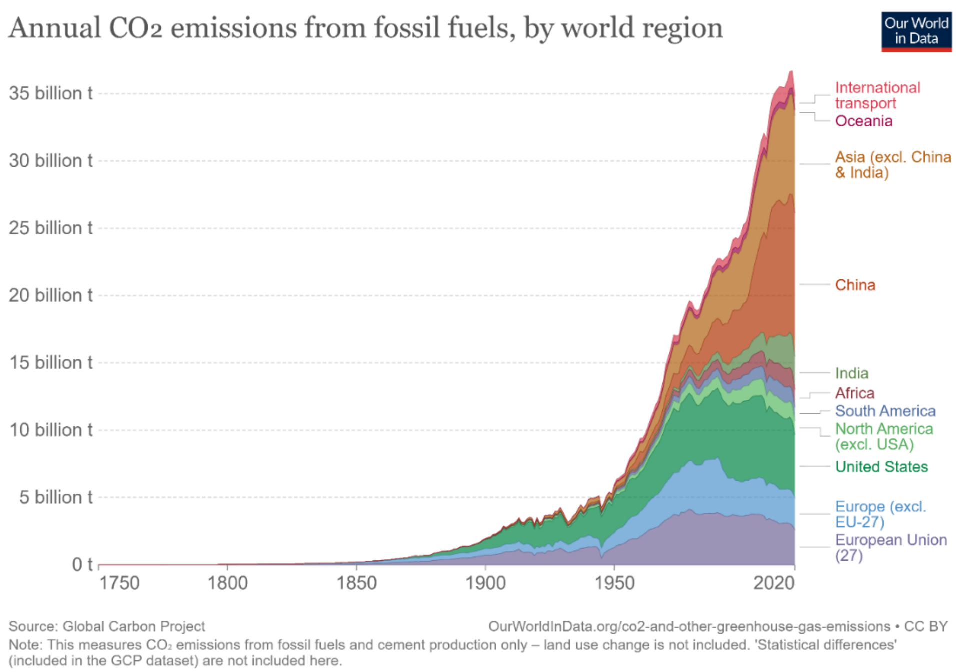
Big Data Application Architecture

Analyzing the impact of Greenhouse gas emissions on weather

Thanushri Rajmohan

Steps:

1. Describing the datasets used:
   Link to the greenhouse emissions dataset: [CO_2 and Greenhouse Gas Emissions - Our World in Data]



The greenhouse gas emissions in each country or region is represented in tonnes and the data is collected from 1971-2020 for each country. It has four columns – Country, three-letter country code, year, greenhouse gas emissions for that year. The header is removed from the dataset and uploaded to the Hadoop local file system using WinScp with "Hadoop" as the user and the private key putty file (.ppk). I had to choose WinScp because I am a windows user. Otherwise I could have tried uploading my data using the scp command from my local system to the Hadoop's local file system.

FTP link to the weather dataset: ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/by_year/
The weather dataset that we used in the class was "Global summaries of the day" dataset for the states in the US from years 2000 to 2021. However, the dataset that I used is "Global summaries of the year" dataset for all countries/stations in the world from the years 1750 to 2021. Since the dataset is different and has different formats compared to the ones that we used in class (csv.gz whereas the ones used in class where tar zipped), I had to extract the weather dataset again from the ftp link using modified shell scripts (*getWeather.sh*) and put it on the local Hadoop file system.

2. Getting the weather data on HDFS:
   The weather data is serialized and ingested into HDFS. If you unzip, "weatherData_thanushrir_ingest.zip", you can find "thanushrir" folder which has the java code to serialize the inputs using Thrift. The uber-jar is generated in the "/home/hadoop/thanushrir/src/target" directory of Hadoop.
   The yarn command to run the uber jar is: yarn jar <uber_jar> <className> <local Hadoop directory> <HDFS output directory>

   yarn jar uber-thanushrir-1.0-SNAPSHOT.jar
   edu.uchicago.mpcs53013.SerializeWeatherSummary
   /home/hadoop/thanushrir/project/weatherData_thanushrir /inputs/thanushrir_weatherData

   Running the uber jar took 1.5 hours and the data was finally on HDFS in the "/inputs/thanushrir_weatherData" directory.

3. Getting the greenhouse emissions data on HDFS:
   I uploaded the greenhouse emissions data in "thanushrir/project/greenhouse_emissions/" directory. You might not find the file there because I removed it due to storage issues on the local file system as mentioned by Prof. Mike. So, I used a hdfs -put command to put the file on HDFS.

   hdfs dfs -put thanushrir_greenhouse_emissions.csv /inputs/greenhouse_emissions

4. Create tables in Hive:
   Run the commands in "load_weatherData.hql" to load the weather data in hive using the beeline command.
   From the next image, you can see that the table "thanushrir_project_weathersummary" has the following columns – Country Code, Station, Year, Month, Day, Value Type and Mean Temperature.
   - Country code – FIPS two letter code
   - Station – Station ID
   - Value Type – TMAX for Maximum Temperature, TMIN for Minimum Temperature, PRCP for precipitation and TAVG for Average Temperature. In our case, we need only the average temperature, so we later filter out and take into consideration the TAVG values only.
   - Mean Temperature – This column has the mean temperature of that station over the entire year.

```
0: jdbc:hive2://localhost:10000/default> SELECT * FROM thanushrir_project_WeatherSummary LIMIT 5;
INFO  : Compiling command(queryId=hive_20221111223915_cce0eb1f-dd0a-424f-b408-fd45fb1570fa): SELECT
* FROM thanushrir_project_WeatherSummary LIMIT 5
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:thanushrir_project_weathersumma
ry.countrycode, type:string, comment:null), FieldSchema(name:thanushrir_project_weathersummary.stati
on, type:string, comment:null), FieldSchema(name:thanushrir_project_weathersummary.year, type:smalli
nt, comment:null), FieldSchema(name:thanushrir_project_weathersummary.month, type:tinyint, comment:n
ull), FieldSchema(name:thanushrir_project_weathersummary.day, type:tinyint, comment:null), FieldSche
ma(name:thanushrir_project_weathersummary.valuetype, type:string, comment:null), FieldSchema(name:th
anushrir_project_weathersummary.meantemperature, type:double, comment:null)], properties:null)
INFO  : EXPLAIN output for queryid hive_20221111223915_cce0eb1f-dd0a-424f-b408-fd45fb1570fa : STAGE
DEPENDENCIES:
  Stage-0 is a root stage [FETCH]

STAGE PLANS:
  Stage: Stage-0
    Fetch Operator
      limit: 5
      Processor Tree:
        TableScan
          alias: thanushrir_project_weathersummary
          GatherStats: false
          Select Operator
            expressions: countrycode (type: string), station (type: string), year (type: smallint),
month (type: tinyint), day (type: tinyint), valuetype (type: string), meantemperature (type: double)
            outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5, _col6
            Limit
              Number of rows: 5
              ListSink


INFO  : Completed compiling command(queryId=hive_20221111223915_cce0eb1f-dd0a-424f-b408-fd45fb1570fa
); Time taken: 0.027 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=hive_20221111223915_cce0eb1f-dd0a-424f-b408-fd45fb1570fa): SELECT
* FROM thanushrir_project_WeatherSummary LIMIT 5
INFO  : Completed executing command(queryId=hive_20221111223915_cce0eb1f-dd0a-424f-b408-fd45fb1570fa
); Time taken: 0.001 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
+-------------------------------------------------+---------------------------------------------+-----
-----------------------------------+------------------------------------------+--------------------
--------------------+----------------------------------------------+-------------------------------
--------------------+
| thanushrir_project_weathersummary.countrycode  | thanushrir_project_weathersummary.station  | than
ushrir_project_weathersummary.year  | thanushrir_project_weathersummary.month  | thanushrir_project_
weathersummary.day  | thanushrir_project_weathersummary.valuetype  | thanushrir_project_weathersumma
ry.meantemperature  |
+-------------------------------------------------+---------------------------------------------+-----
-----------------------------------+------------------------------------------+--------------------
--------------------+----------------------------------------------+-------------------------------
--------------------+
| AG                                             | AG000060390                                | 1971
                                    | 1                                        | 1
                    | TMAX                                         | 96.0
                    |
| AG                                             | AG000060390                                | 1971
                                    | 1                                        | 1
                    | TMIN                                         | 51.0
                    |
| AG                                             | AG000060390                                | 1971
                                    | 1                                        | 1
                    | PRCP                                         | 442.0
                    |
| AG                                             | AG000060590                                | 1971
                                    | 1                                        | 1
                    | TMAX                                         | 193.0
                    |
| AG                                             | AG000060590                                | 1971
                                    | 1                                        | 1
                    | TMIN                                         | 7.0
                    |
+-------------------------------------------------+---------------------------------------------+-----
-----------------------------------+------------------------------------------+--------------------
--------------------+----------------------------------------------+-------------------------------
--------------------+
5 rows selected (0.251 seconds)
```

Run the commands in “load_greenhouse_emissions.hql” to load greenhouse emissions data in hive using the beeline command. The table “thanushrir_greenhouse_emissions” stored as ORC form has the greenhouse emissions data.

5. Adding a separate country table to Hive:
The weather data has only the FIPS code with station ID. We need to join the weather data to the greenhouse emissions data using the FIPS code from weather data and the country name in the greenhouse emissions data. We get rid of the three letter country code in the greenhouse emissions data as it is not necessary. “ghcnm-countries.csv” is the csv file that maps FIPS country code to the country names. This is loaded to HDFS in the same way by loading to local HDFS using WinScp and using hdfs put command to put it on HDFS. This file is in the “/inputs/thanushrir_country” on HDFS. Run the commands in “load_countryData.hql” to create a “thanushrir_country_data” table in Hive.

6. Joining Hive tables to create one large final table to store it in HBase:
Join the greenhouse emissions data with the countries data based on the FIPS code using hive commands in “join_greenhouse_country.hql”. The table created in Hive is “thanushrir_project_greenhouse”.
Now, the weather data is grouped by FIPS country code and year by running the commands in “group_weatherData.hql”.
Now this resultant table is in turn joined with the “thanushrir_project_greenhouse” hive table based on the FIPS code and the year. The commands are present in “join_weather_greenhouse.hql”. The Hive table that is created in this step “thanushrir_project_weather_greenhouse” is the final Hive table which we will be using for processing everywhere.

7. Generating HBase table for this Hive table:
Create table “thanushrir_project_hbase_HiveStyle” in hbase shell with column key as “values”.

```
hbase:001:0> create 'thanushrir_project_hbase_HiveStyle','values';
hbase:002:0> scan 'thanushrir_project_hbase_HiveStyle';
ROW                  COLUMN+CELL
0 row(s)
Took 0.0881 seconds
```

Now, run the commands in “hbase_storage_hivestyle.hql” in Hive to store the hive table.
Run the following commands in HBase to see if the tables values are successfully loaded.

```
hbase:005:0> get 'thanushrir_project_hbase_HiveStyle','US2006';
    COLUMN                CELL
    values:avg_temp       timestamp=2022-11-18T00:12:45.585, value=78.70165
    values:country        timestamp=2022-11-18T00:12:45.585, value=United States
    values:emissions      timestamp=2022-11-18T00:12:45.585, value=6057163300
    1 row(s)
Took 0.0043 seconds
  hbase:007:0> get 'thanushrir_project_hbase_HiveStyle','UK2000';
```

COLUMN CELL
values:avg_temp timestamp=2022-11-18T00:12:45.569, value=77.47375
values:country timestamp=2022-11-18T00:12:45.569, value=United Kingdom
values:emissions timestamp=2022-11-18T00:12:45.569, value=569033660
1 row(s)
Took 0.0037 seconds

8. Generating Hive style HBase tables from Spark Scala:
   Execute all the commands in "Spark_HiveStyle_tables.scala". Create a HBase table using hbase shell with the following command:
   hbase:001:0> create 'thanushrir_project_spark_hivestyle','values';
   Created table thanushrir_project_spark_hivestyle
   Took 0.9098 seconds
   Export the Spark Hive Style table to HBase by running the commands in "Spark_export_HiveStyle_table_to_hbase.hql" in hive using beeline command. If you type "get 'thanushrir_project_spark_hivestyle','US2006';" in hbase shell, it should return the values stored in the hbase tables.

9. Generating Object style HBase tables from Spark Scala:

   Execute all the commands in "Spark_ObjectStyle_tables.scala". Create a HBase table using hbase shell with the following command:

   hbase:001:0> create 'thanushrir_project_spark_objectstyle','values';

   Created table thanushrir_project_spark_objectstyle

   Took 0.9098 seconds

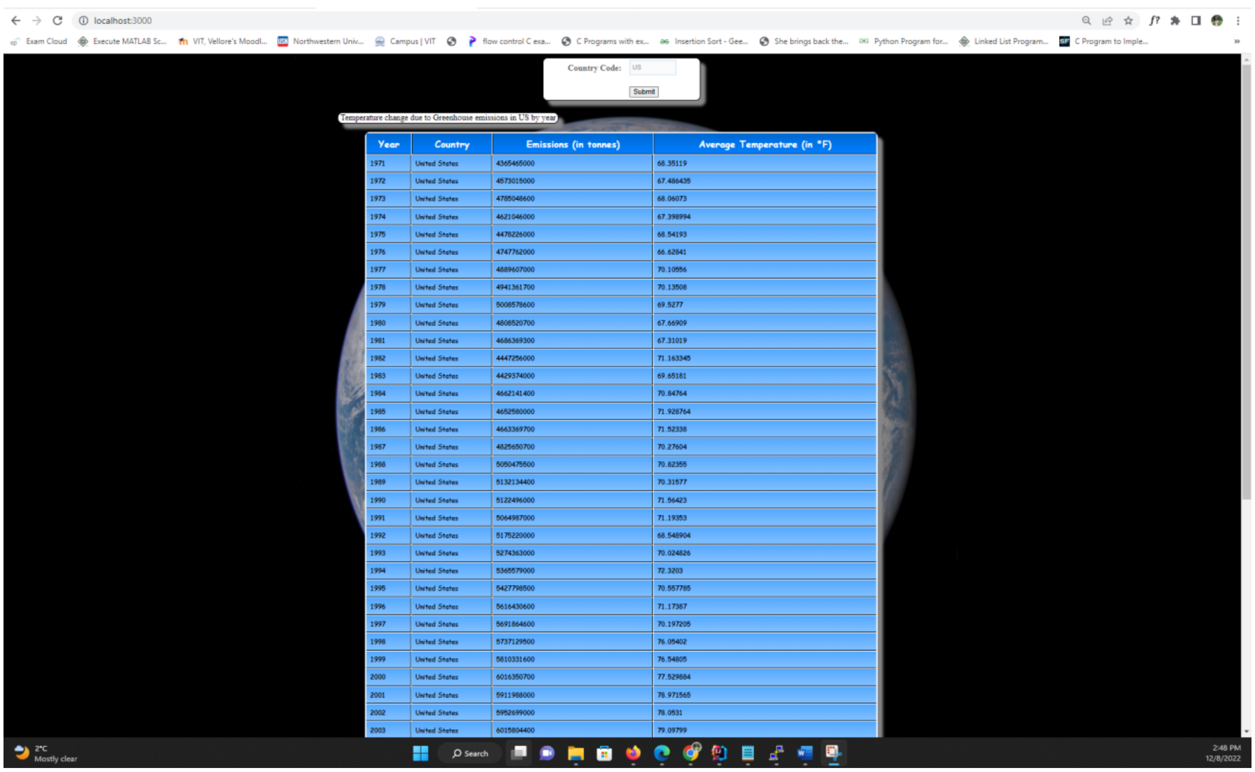
   Export the Spark Object Style table to HBase by running the commands in "Spark_export_ObjectStyle_table_to_hbase.hql" in hive using beeline command. If you type "get 'thanushrir_project_spark_objectstyle','US2006';" in hbase shell, it should return the values stored in the hbase tables.

10. Running the applications on local machines:
    There are three different ways that these apps use the tables.
    - "Thanushri_Rajmohan_Project_HBase_Hive" uses the HBase table "thanushrir_project_hbase_HiveStyle" stored initially using plain hive commands.
    - "Thanushri_Rajmohan_Project_Spark_HiveStyleTable" uses the spark hive style HBase table ""
    - "Thanushri_Rajmohan_Project_Spark_ObjectStyleTable" uses the spark object style HBase table ""

    All these applications have the same code. Just the tables that have been used in each of the applications are different.

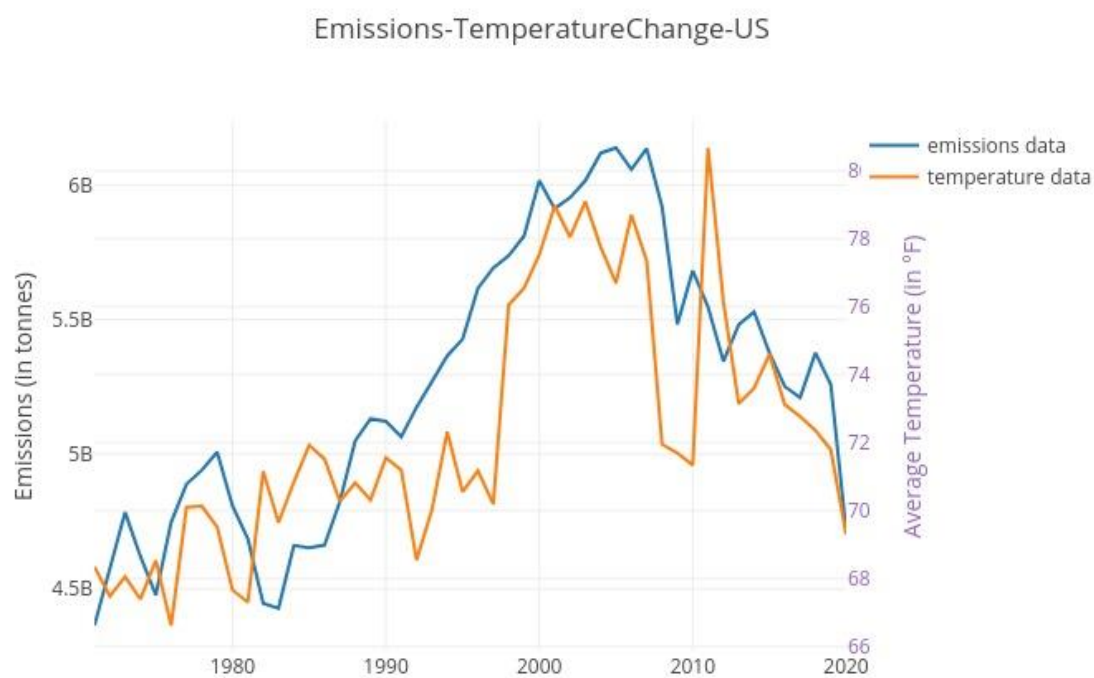
One additional functionality that I incorporated in this was to autogenerate a graph for each country code that is submitted to view the greenhouse gas emissions and the weather changes in a single graph as a better form of representation. I used the plotly library which can be installed using npm install plotly. Plotly requires a api token which I generated and the api key along with the code to autogenerate the graph using plotly is in app.js file.

Once each of the zip files of codes are unzipped, each of the application with the app.js code must be built and the node.js configurations called "RunApp" have to be set up to run it locally on port 3000 with "3000 localhost 8070". Then run the project with localhost:3000/ and submit FIPS code of any country (UK for United Kingdom, US for United States, CA for Canada, IN for India, etc.) and you should get your results in a table format below. In addition to that, if you go to your IntelliJ console, you should see a json object with an url to plotly (something similar to this - https://chart-studio.plotly.com/~thanushrir/8). Click on this url and then you can see the graph that is generated for the country code that you entered as input. You can save it for use at a later point of time.



Year	Country	Emissions (in tonnes)	Average Temperature (in °F)
1971	United States	4365465000	68.35119
1972	United States	4573015000	67.486435
1973	United States	4785048600	68.06073
1974	United States	4621046000	67.398994
1975	United States	4478226000	68.54193
1976	United States	4747762000	66.62841
1977	United States	4889607000	70.10556
1978	United States	4941361700	70.13508
1979	United States	5008578600	69.5277
1980	United States	4808520700	67.66909
1981	United States	4686369300	67.31019
1982	United States	4447256000	71.163345
1983	United States	4429374000	69.65181
1984	United States	4662141400	70.84764
1985	United States	4652580000	71.928764
1986	United States	4663369700	71.52338
1987	United States	4825650700	70.27604
1988	United States	5050475500	70.82355
1989	United States	5132134400	70.31577
1990	United States	5122496000	71.56423
1991	United States	5064987000	71.19353
1992	United States	5175220000	68.548904
1993	United States	5274363000	70.024826
1994	United States	5365579000	72.3203
1995	United States	5427798500	70.557785
1996	United States	5616430600	71.17387
1997	United States	5691864600	70.197205
1998	United States	5737129500	76.05402
1999	United States	5810331600	76.54805
2000	United States	6016350700	77.529884
2001	United States	5911988000	78.971565
2002	United States	5952699000	78.0531
2003	United States	6015804400	79.09799

```
{
  streamstatus: undefined,
  url: 'https://chart-studio.plotly.com/~thanushrir/8',
  message: '',
  warning: '',
  filename: 'Emissions-TemperatureChange-US',
  error: ''
}
```



11. <u>Web app which sends real-time data to kafka consumer:</u>
    I first created a kafka topic.
    kafka-topics.sh --create --zookeeper z-2.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:2181,z-1.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:2181,z-3.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:2181 --replication-factor 1 --partitions 1 --topic thanushrir_project_greenhouseWeather

When you unzip "Thanushri_Rajmohan_Project_app_with_weather__greenhouse_form.zip", you should find the project that takes as input weather and greenhouse emissions data for a year from the user using the "submit-weather-emissions.html" form. The app.js file takes this input and puts it in the kafka consumer.
View the kafka-consumer using the following command.

kafka-console-consumer.sh --bootstrap-server b-2.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092,b-3.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092,b-1.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092 --topic thanushrir_project_greenhouseWeather --from-beginning

Every app from now on uses "thanushrir_project_spark_hivestyle" hbase table to run.
To run this project, once you unzip, set SFTP with name Webserver to ec2-server in the deployment options. Exclude node modules and set mappings deployment path to "/thanushrir/weather_greenhouse_form_app". Change the RunApp configuration of the project to "3000 localhost 8070 localhost:9092". Upload your entire project directly to "/home/ec2-user/thanushrir/weather_greenhouse_form_app" by right clicking on the project directory -> Deployment -> Upload to Webserver. Go to the above directory in the ssh shell and run the following. (3004 is my assigned port)

npm install
npm install kafka-node --no-optional
node app.js 3004 ec2-54-166-56-39.compute-1.amazonaws.com 8070 b-2.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092,b-3.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092,b-1.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092

Because port 3004 is occupied by the load balanced servers currently, running this command might give an error.

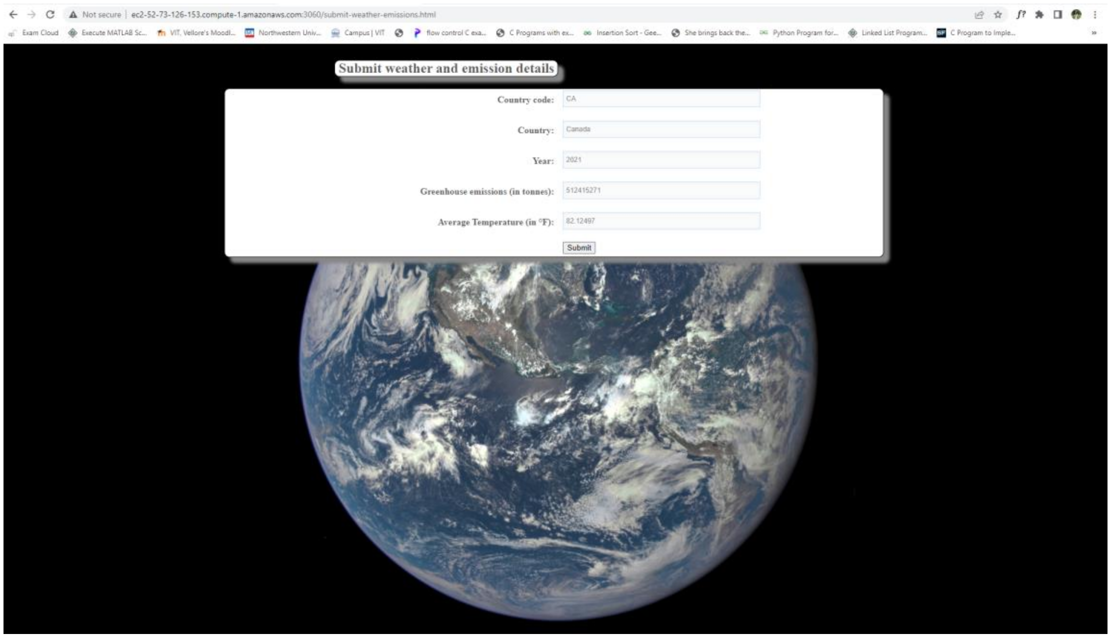
Instead run this command with a random port 3060 to check if the app is working. If this also doesn't work and gives port in use error, try another port below 3100.

node app.js 3060 ec2-54-166-56-39.compute-1.amazonaws.com 8070 b-2.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092,b-3.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092,b-1.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092

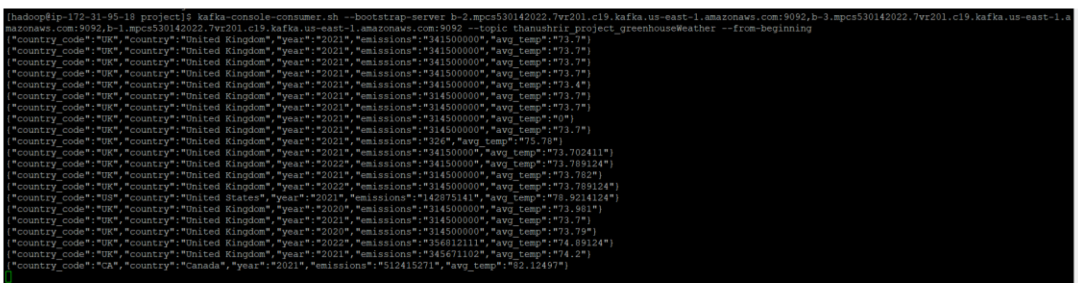
Now if you go to the following link, you should see your app running.

http://ec2-52-73-126-153.compute-1.amazonaws.com:3060/

Go to this link to enter input to your app.
http://ec2-52-73-126-153.compute-1.amazonaws.com:3060/submit-weather-emissions.html



Type input and enter Submit. Once you do that, your kafka consumer should have your input values as a JSON object report like below. You can see Canada is in the Kafka Consumer as the last value.



```
[hadoop@ip-172-31-95-18 project]$ kafka-console-consumer.sh --bootstrap-server b-2.mpcs530142022.7vr201.c19.kafka.us-east-1.amazonaws.com:9092,b-3.mpcs530142022.7vr201.c19.kafka.us-east-1.amazonaws.com:9092,b-1.mpcs530142022.7vr201.c19.kafka.us-east-1.amazonaws.com:9092 --topic thanushrir_project_greenhouseWeather --from-beginning
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"341500000","avg_temp":"73.7"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"341500000","avg_temp":"73.7"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"341500000","avg_temp":"73.7"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"341500000","avg_temp":"73.7"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"341500000","avg_temp":"73.4"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"314500000","avg_temp":"73.7"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"314500000","avg_temp":"73.7"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"314500000","avg_temp":"0"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"314500000","avg_temp":"73.7"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"326","avg_temp":"75.78"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"34150000","avg_temp":"73.702411"}
{"country_code":"UK","country":"United Kingdom","year":"2022","emissions":"34150000","avg_temp":"73.789124"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"314500000","avg_temp":"73.782"}
{"country_code":"UK","country":"United Kingdom","year":"2022","emissions":"314500000","avg_temp":"73.789124"}
{"country_code":"US","country":"United States","year":"2021","emissions":"142875141","avg_temp":"78.9214124"}
{"country_code":"UK","country":"United Kingdom","year":"2020","emissions":"314500000","avg_temp":"73.981"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"314500000","avg_temp":"73.7"}
{"country_code":"UK","country":"United Kingdom","year":"2020","emissions":"314500000","avg_temp":"73.79"}
{"country_code":"UK","country":"United Kingdom","year":"2022","emissions":"356812111","avg_temp":"74.89124"}
{"country_code":"UK","country":"United Kingdom","year":"2021","emissions":"345671102","avg_temp":"74.2"}
{"country_code":"CA","country":"Canada","year":"2021","emissions":"512415271","avg_temp":"82.12497"}
```

But if you go to your home page and try to view the newly entered value for that country, the new input won't be there. This is because the values from kafka consumer are not updated in the HBase table. So the next step is going to be how to put the values in kafka consumer into HBase table.

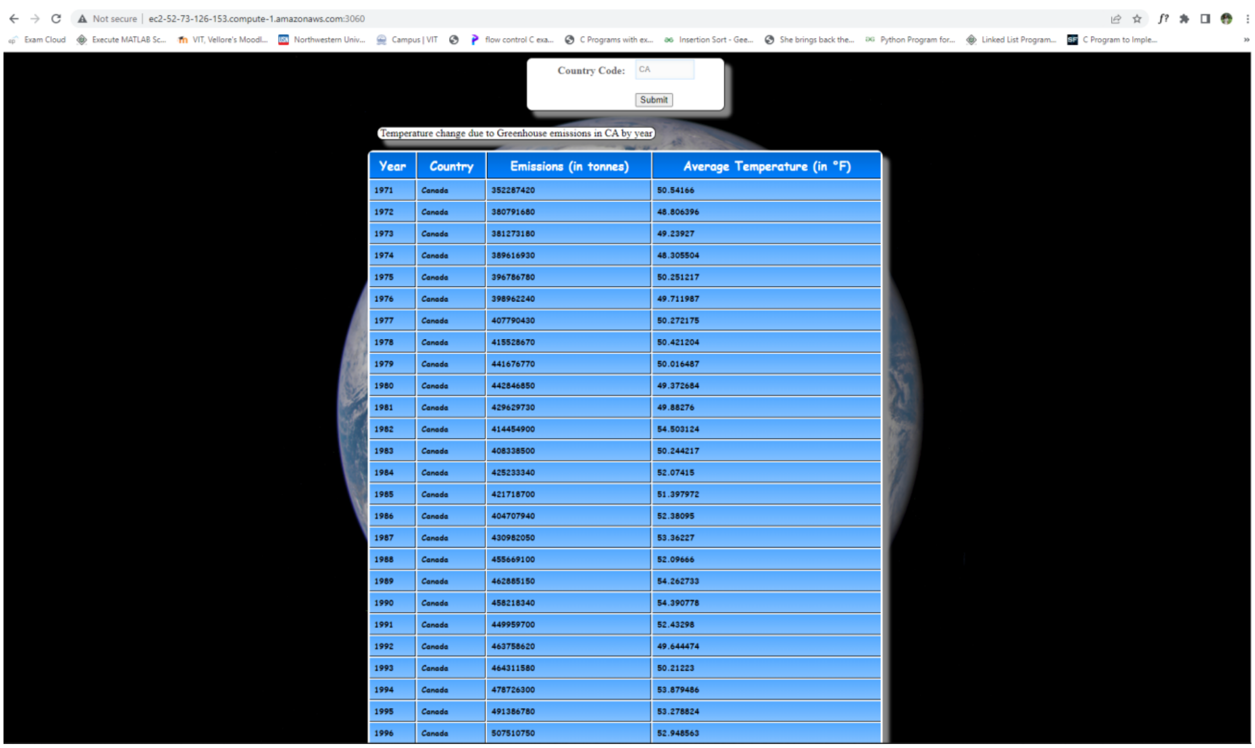
12. <u>Final streaming web app where input values are updated in the HBase table:</u>

My streaming application writes to the HBase table directly from the speed layer data. This was the best way to insert data into my application that takes as input float, string values. This is because HBase Increment is not the best way because increment is allowed only for "long" datatype values and I have string and float values for my application. So I used the HBase put command to directly update my HBase table.

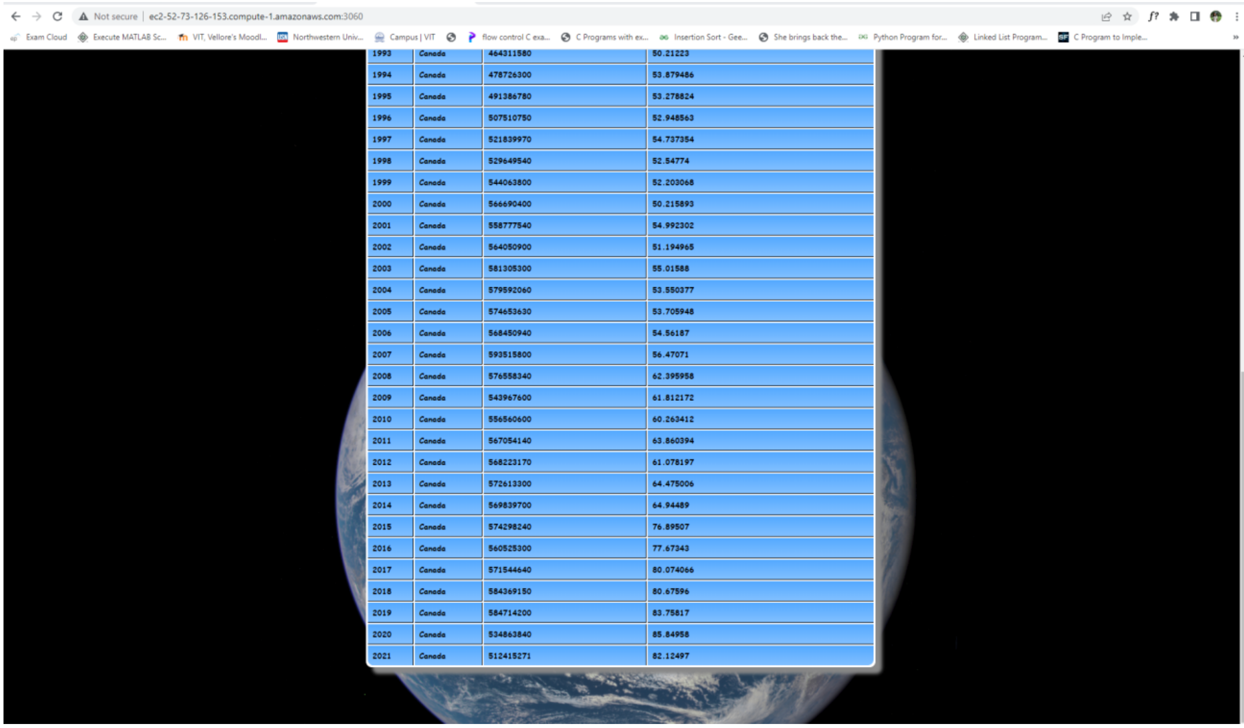
Unzip “Thanushri_Rajmohan_Project_app_speed_layer.zip” and build the project. Your uber jar files should be generated (in my case its already there in the Hadoop file system). Cd to “/home/hadoop/thanushrir/src/target” after starting a Hadoop user ssh session. Run the uber jar using the following spark-submit command.

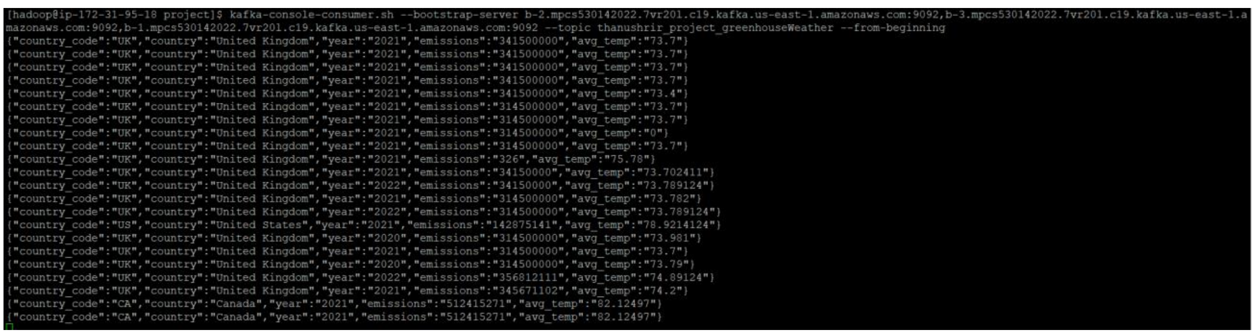
```
spark-submit --master local[2] --driver-java-options "-Dlog4j.configuration=file:///home/hadoop/ss.log4j.properties" --class StreamWeather uber-Thanushri_Rajmohan_Project_app_speed_layer-1.0-SNAPSHOT.jar b-2.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092,b-3.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092,b-1.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092
```

Keep the application in the 11th step running and go to the same link to submit a new value. Once you submit the result and go to the home page and enter “CA” as input, you should see the new value for 2021 in your result table.



Year	Country	Emissions (in tonnes)	Average Temperature (in °F)
1971	Canada	352287420	50.54166
1972	Canada	380791680	48.806396
1973	Canada	381273180	49.23927
1974	Canada	389616930	48.305504
1975	Canada	396786780	50.251217
1976	Canada	398962240	49.711987
1977	Canada	407790430	50.272175
1978	Canada	415528670	50.421204
1979	Canada	441676770	50.016487
1980	Canada	442846850	49.372684
1981	Canada	429629730	49.88276
1982	Canada	414454900	54.503124
1983	Canada	408338500	50.244217
1984	Canada	425233340	52.07415
1985	Canada	421718700	51.397972
1986	Canada	404707940	52.38095
1987	Canada	430982050	53.36227
1988	Canada	455669100	52.09666
1989	Canada	462885150	54.262733
1990	Canada	458218340	54.390778
1991	Canada	449959700	52.43298
1992	Canada	463758620	49.644474
1993	Canada	464311580	50.21223
1994	Canada	478726300	53.879486
1995	Canada	491386780	53.278824
1996	Canada	507510750	52.948563

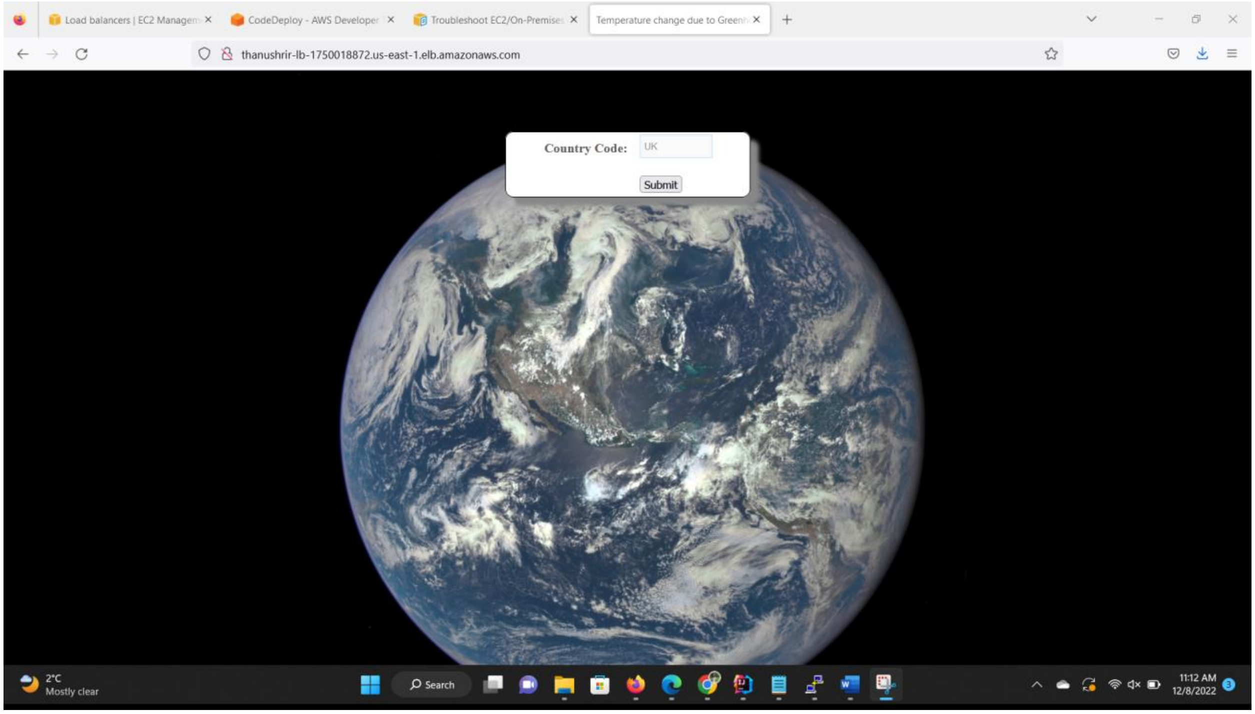


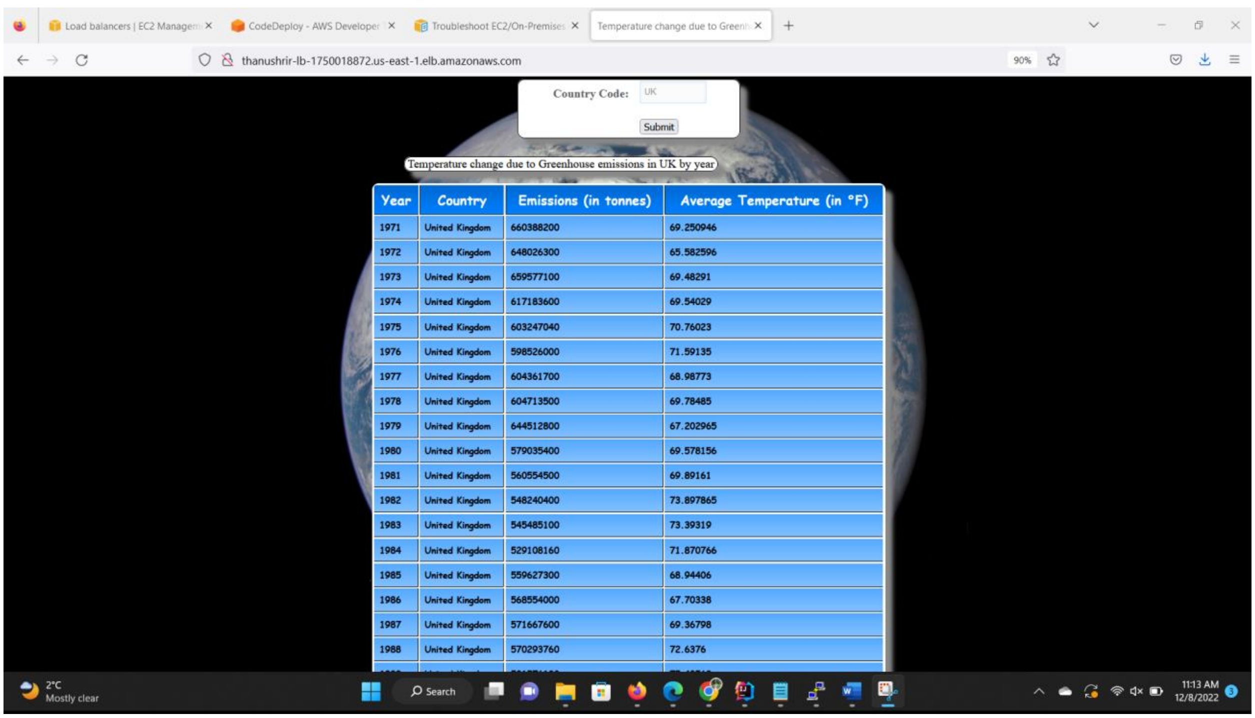


You can see the new Canada value that you entered in step 12 in your Kafka Console Consumer.

13. Deploying the application:
The project is deployed to CodeDeploy and the following link can be used to run the CodeDeploy application.
http://thanushrir-lb-1750018872.us-east-1.elb.amazonaws.com/

When you enter an input in http://thanushrir-lb-1750018872.us-east-1.elb.amazonaws.com/submit-weather-emissions.html , the application sends the values to the kafka consumer. Kafka consumer gets messages from the code deploy application but these new values are not updated in hbase because streaming app is not uploaded on codedeploy. Following are some screenshots of the app using loadbalancer DNS name.





Year	Country	Emissions (in tonnes)	Average Temperature (in °F)
1971	United Kingdom	660388200	69.250946
1972	United Kingdom	648026300	65.582596
1973	United Kingdom	659577100	69.48291
1974	United Kingdom	617183600	69.54029
1975	United Kingdom	603247040	70.76023
1976	United Kingdom	598526000	71.59135
1977	United Kingdom	604361700	68.98773
1978	United Kingdom	604713500	69.78485
1979	United Kingdom	644512800	67.202965
1980	United Kingdom	579035400	69.578156
1981	United Kingdom	560554500	69.89161
1982	United Kingdom	548240400	73.897865
1983	United Kingdom	545485100	73.39319
1984	United Kingdom	529108160	71.870766
1985	United Kingdom	559627300	68.94406
1986	United Kingdom	568554000	67.70338
1987	United Kingdom	571667600	69.36798
1988	United Kingdom	570293760	72.6376