IST 769 Final Project

Sana Khan

IST 769 Spring 2023

Summary

The purpose of this project was to demonstrate skills in database management by using Spark to store time-series data in a Cassandra Keyspace and query it via Drill. Two different data sources were used to demonstrate proficiency in pulling data from an API. The first data source was NYC 311 complaints, which saved the data in a JSON. The second source was Weather data for Syracuse New York and the data was saved as a CSV. The steps and code are provided below.

Steps:

1. Create a Keyspace in Cassandra
2. Pull data from the API and store it as a JSON/CSV
3. Read the JSON/CSV and store the data in a variable that's transferred to a Data Frame.
4. Create a table in Jyputer that is transcribed to a Cassandra Keyspace using Spark
5. Write the data in the data frame to the table created in step 4 in the keyspace
6. Enable the Cassandra plugin in Drill
7. Query the newly created tables in Drill

NYC 311 complaints

Create a Cassandra Keyspace called project_311

```
CREATE KEYSPACE IF NOT EXISTS project311
WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
```

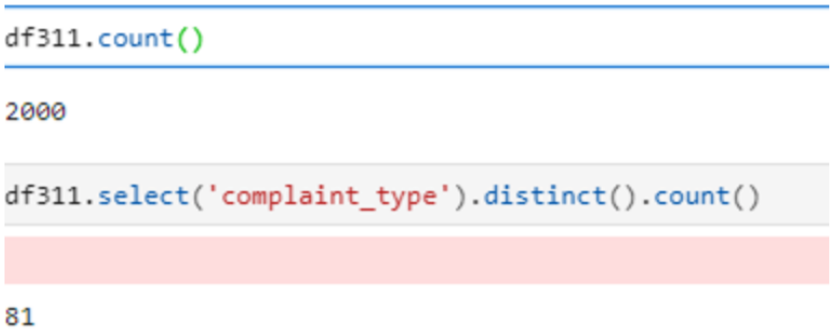
Pull the data from the API and save it as JSON/CSV

```
!pip install pandas
!pip install sodapy

import pandas as pd
from sodapy import Socrata

data_url='data.cityofnewyork.us'
data_set='erm2-nwe9'
app_token='T2Jeqm4J7zvAiJruefHcYvb7y'
selectc = "unique_key, created_date, agency, complaint_type, descriptor, location_type, intersection_street_1, i
client = Socrata(data_url,app_token)
client.timeout = 360

```

```
results = client.get(data_set,select=selectc,  limit=2000)

df = pd.DataFrame.from_records(results)

df.to_json("my_311_data.json", orient="records")
df.to_csv("my_311_data.csv")
```

Read the JSON file and save it as a variable

```
file_path = 'file:///home/jovyan/datasets/my_311_data.json'
df311= spark.read.json(file_path)
df311.printSchema()

df311.count()
df311.select('complaint_type').distinct().count()

```



Save the table in Cassandra

```
create_table_sql = '''
CREATE TABLE project311.data
(

 address_type text,
 agency text,
 borough text,
 city text,
 community_board text,
 complaint_type text,
 created_date timestamp,
 descriptor text,
 incident_address text,
 incident_zip text,
 intersection_street_1 text,
 intersection_street_2 text,
 latitude float,
 location_type text,
 longitude float,
 open_data_channel_type text,
 resolution_action_updated_date timestamp,
 status text,
 unique_key text,
 PRIMARY KEY (unique_key)
```

```
25
26   );
27   '''
28 from cassandra.cluster import Cluster
29 with Cluster([cassandra_host]) as cluster:
30     session = cluster.connect()
31 session.execute(create_table_sql)
32 print(create_table_sql)
33
```

```
CREATE TABLE project311.data
(

 address_type text,
 agency text,
 borough text,
 city text,
 community_board text,
 complaint_type text,
 created_date timestamp,
 descriptor text,
 incident_address text,
 incident_zip text,
 intersection_street_1 text,
 intersection_street_2 text,
 latitude float,
 location_type text,
 longitude float,
 open_data_channel_type text,
 resolution_action_updated_date timestamp,
 status text,
 unique_key text,
 PRIMARY KEY (unique_key)

  );
```

Save the data from the JSON into the table `data`

```
1  df2 = df311.toDF(
2
3  "address_type",
4  "agency",
5  "borough",
6  "city",
7  "community_board",
8  "complaint_type",
9  "created_date",
10 "descriptor",
11 "incident_address",
12 "incident_zip",
13 "intersection_street_1",
```

```
"intersection_street_2",
"latitude",
"location_type",
"longitude",
"open_data_channel_type",
"resolution_action_updated_date",
"status",
"unique_key")

df2.write.format("org.apache.spark.sql.cassandra")\
    .mode("Append")\
    .option("table", "data")\
    .option("keyspace", "project311")\
    .save()

```

View data in Cassandra to confirm it has been added

```
select * from project311.data311;
```

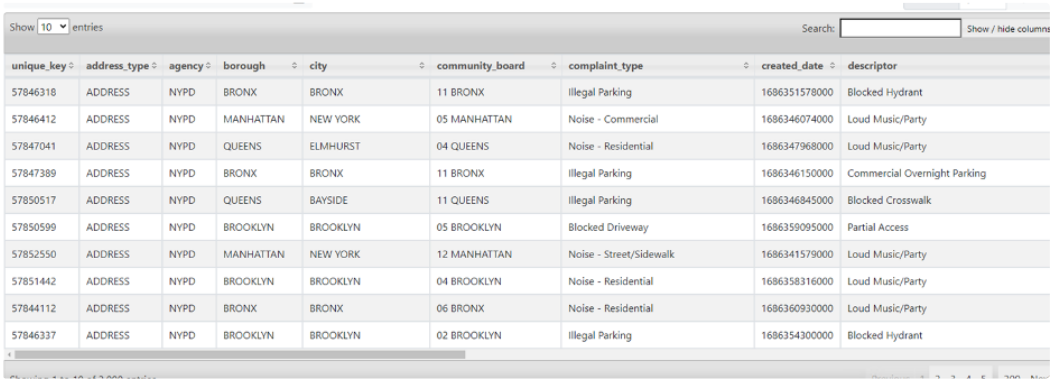
Query the data using Spark

```
df3 =spark.read.format("org.apache.spark.sql.cassandra")\

    .option("table", "data")\

    .option("keyspace","project311")\

    .load()

df3.count()
```

```
5]: 2000

6]: df3.printSchema()

root
 |-- unique_key: string (nullable = false)
 |-- address_type: string (nullable = true)
 |-- agency: string (nullable = true)
 |-- borough: string (nullable = true)
 |-- city: string (nullable = true)
 |-- community_board: string (nullable = true)
 |-- complaint_type: string (nullable = true)
 |-- created_date: timestamp (nullable = true)
 |-- descriptor: string (nullable = true)
 |-- incident_address: string (nullable = true)
 |-- incident_zip: string (nullable = true)
 |-- intersection_street_1: string (nullable = true)
 |-- intersection_street_2: string (nullable = true)
 |-- latitude: float (nullable = true)
 |-- location_type: string (nullable = true)
 |-- longitude: float (nullable = true)
 |-- open_data_channel_type: string (nullable = true)
 |-- resolution_action_updated_date: timestamp (nullable = true)
 |-- status: string (nullable = true)
```

Query the data using Drill

```
SELECT * from Cassandra.project311.`data`
```



Show 10 entries Search:

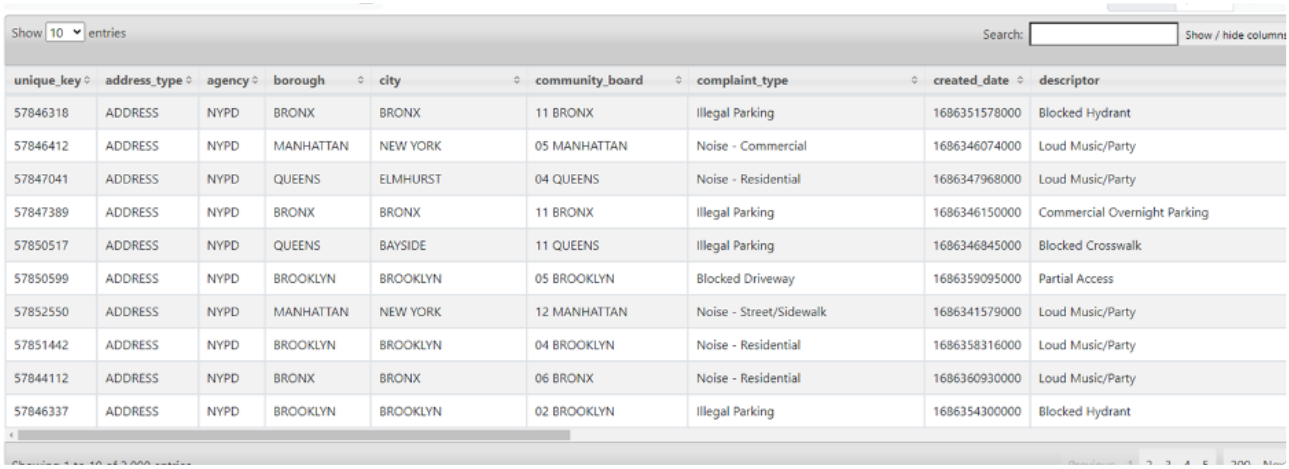
unique_key	address_type	agency	borough	city	community_board	complaint_type	created_date	descriptor
57846318	ADDRESS	NYPD	BRONX	BRONX	11 BRONX	Illegal Parking	1686351578000	Blocked Hydrant
57846412	ADDRESS	NYPD	MANHATTAN	NEW YORK	05 MANHATTAN	Noise - Commercial	1686346074000	Loud Music/Party
57847041	ADDRESS	NYPD	QUEENS	ELMHURST	04 QUEENS	Noise - Residential	1686347968000	Loud Music/Party
57847389	ADDRESS	NYPD	BRONX	BRONX	11 BRONX	Illegal Parking	1686346150000	Commercial Overnight Parking
57850517	ADDRESS	NYPD	QUEENS	BAYSIDE	11 QUEENS	Illegal Parking	1686346845000	Blocked Crosswalk
57850599	ADDRESS	NYPD	BROOKLYN	BROOKLYN	05 BROOKLYN	Blocked Driveway	1686359095000	Partial Access
57852550	ADDRESS	NYPD	MANHATTAN	NEW YORK	12 MANHATTAN	Noise - Street/Sidewalk	1686341579000	Loud Music/Party
57851442	ADDRESS	NYPD	BROOKLYN	BROOKLYN	04 BROOKLYN	Noise - Residential	1686358316000	Loud Music/Party
57844112	ADDRESS	NYPD	BRONX	BRONX	06 BRONX	Noise - Residential	1686360930000	Loud Music/Party
57846337	ADDRESS	NYPD	BROOKLYN	BROOKLYN	02 BROOKLYN	Illegal Parking	1686354300000	Blocked Hydrant

```
SELECT count(unique_key) as Num_Complaints,
borough
from Cassandra.project311.`data`
group by 2
order by 1 DESC

```

Num_Complaints	borough
575	BROOKLYN
558	QUEENS
413	BRONX
402	MANHATTAN
51	STATEN ISLAND
1	Unspecified

```
SELECT count(unique_key) as Num_Complaints,
complaint_type,
borough
from Cassandra.project311.`data`
```

```
group by 2,3
order by 3

```



Show 10 entries Search: Show / hide columns

unique_key	address_type	agency	borough	city	community_board	complaint_type	created_date	descriptor
57846318	ADDRESS	NYPD	BRONX	BRONX	11 BRONX	Illegal Parking	1686351578000	Blocked Hydrant
57846412	ADDRESS	NYPD	MANHATTAN	NEW YORK	05 MANHATTAN	Noise - Commercial	1686346074000	Loud Music/Party
57847041	ADDRESS	NYPD	QUEENS	ELMHURST	04 QUEENS	Noise - Residential	1686347968000	Loud Music/Party
57847389	ADDRESS	NYPD	BRONX	BRONX	11 BRONX	Illegal Parking	1686346150000	Commercial Overnight Parking
57850517	ADDRESS	NYPD	QUEENS	BAYSIDE	11 QUEENS	Illegal Parking	1686346845000	Blocked Crosswalk
57850599	ADDRESS	NYPD	BROOKLYN	BROOKLYN	05 BROOKLYN	Blocked Driveway	1686359095000	Partial Access
57852550	ADDRESS	NYPD	MANHATTAN	NEW YORK	12 MANHATTAN	Noise - Street/Sidewalk	1686341579000	Loud Music/Party
57851442	ADDRESS	NYPD	BROOKLYN	BROOKLYN	04 BROOKLYN	Noise - Residential	1686358316000	Loud Music/Party
57844112	ADDRESS	NYPD	BRONX	BRONX	06 BRONX	Noise - Residential	1686360930000	Loud Music/Party
57846337	ADDRESS	NYPD	BROOKLYN	BROOKLYN	02 BROOKLYN	Illegal Parking	1686354300000	Blocked Hydrant

Open Meteo Weather Data

Create a Cassandra Keyspace called `project_weather_data`

```
CREATE KEYSPACE IF NOT EXISTS project_weather_data
WITH REPLICATION = {
  'class': 'SimpleStrategy',
  'replication_factor': 1
};
```

Code to pull data from the API:

```
import requests
response = requests.get('https://archive-api.open-meteo.com/v1/archive?latitude=43.0481&longitude=76.1474&start_
data = response.json()
```

Code to save data as CSV

```
from pyspark.sql.functions import col,explode
import pandas as pd
df = pd.json_normalize(data)
df.to_json("project_weather_data_meteo.json2", orient="records")
df.to_csv("project_weather_data_meteo.csv2",index=False,header=True)
```

Code to open the CSV and save it as a data frame:

```

df_weather= pd.read_csv(file_weather)
```

```
df_weather = df_weather.rename(columns={'Time': 'hour'})

df_spark = spark.createDataFrame(df_weather)

```

```
df_spark.printSchema()
```

```
root
 |-- Date: string (nullable = true)
 |-- hour: string (nullable = true)
 |-- time: string (nullable = true)
 |-- temperature_2m: double (nullable = true)
 |-- dewpoint_2m: double (nullable = true)
 |-- precipitation: double (nullable = true)
 |-- rain: double (nullable = true)
 |-- snowfall: double (nullable = true)
 |-- weathercode: long (nullable = true)
 |-- cloudcover: long (nullable = true)
 |-- cloudcover_low: long (nullable = true)
 |-- windspeed_10m: double (nullable = true)
 |-- windspeed_100m: double (nullable = true)
 |-- winddirection_10m: long (nullable = true)
 |-- winddirection_100m: long (nullable = true)
 |-- windgusts_10m: double (nullable = true)
```

Code to create a table in Jupyter notebook under `project_weather_data` keyspace in Cassandra and save the data to table called `weather_data_table`

```
create_table_sql =  '''
CREATE TABLE IF NOT EXISTS project_weather_data.weather_data_table (
    date date,
    hour time,
    time timestamp,
    temperature_2m float,
    dewpoint_2m float,
    precipitation float,
    rain float,
    snowfall float,
    weathercode float,
    cloudcover float,
    cloudcover_low float,
    windspeed_10m float,
    windspeed_100m float,
    winddirection_10m float,
    winddirection_100m float,
    windgusts_10m float,
    PRIMARY KEY (time)
  );
  '''
```

```
from cassandra.cluster import Cluster
with Cluster([cassandra_host]) as cluster:
    session = cluster.connect()
    session.execute(create_table_sql)
print(create_table_sql)

```

Code to write the data back to Cassandra

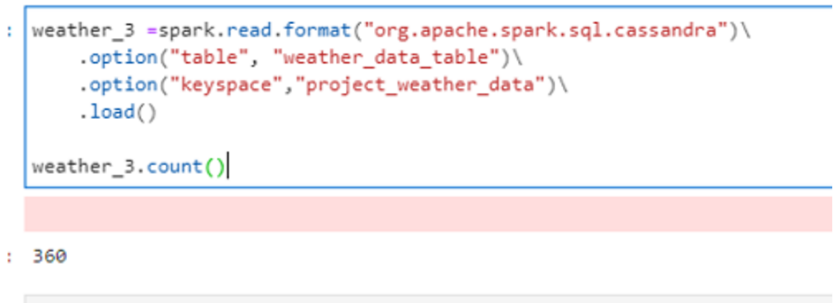
```
weather_2 = df_spark.toDF("date",
"hour",
"time",
"temperature_2m",
"dewpoint_2m",
"precipitation",
"rain",
"snowfall",
"weathercode",
"cloudcover",
"cloudcover_low",
"windspeed_10m",
"windspeed_100m",
"winddirection_10m",
"winddirection_100m",
"windgusts_10m");

weather_2.write.format("org.apache.spark.sql.cassandra")\
    .mode("Append")\
    .option("table", "weather_data_table")\
    .option("keyspace", "project_weather_data")\
    .save()
```

Code to query the table in cql

```
project_weather_data> select * from project_weather_data.weather_data_table;
```

Code to query in Spark

```
weather_3 =spark.read.format("org.apache.spark.sql.cassandra")\
    .option("table", "weather_data_table")\
    .option("keyspace","project_weather_data")\
    .load()

weather_3.count()
```



```
weather_3.createOrReplaceTempView("weather")
query =  '''
 SELECT time, temperature_2m, precipitation, rain, snowfall, weathercode FROM weather_data_table;
 '''
spark.sql(query).explain()

```

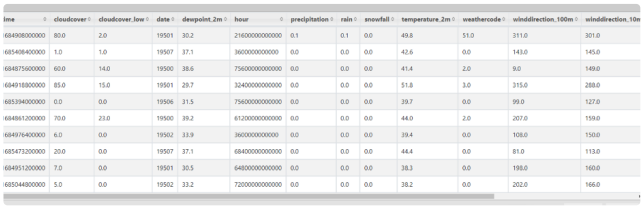
```
15]: weather_3.createOrReplaceTempView("weather")
query =  '''
 SELECT time, temperature_2m, precipitation, rain, snowfall, weathercode FROM weather_data_table;
 '''
spark.sql(query).explain()

== Physical Plan ==
*(1) Project [time#123, temperature_2m#132, precipitation#129, rain#130, snowfall#131, weathercode#133]
+- BatchScan[time#123, precipitation#129, rain#130, snowfall#131, temperature_2m#132, weathercode#133] Cassandra Scan: project_weather_data.weathe
r_data_table
 - Cassandra Filters: []
 - Requested Columns: [time,precipitation,rain,snowfall,temperature_2m,weathercode]
```

```
query =  '''
 SELECT time, temperature_2m, precipitation, rain, snowfall, weathercode from weather_data_table  WHERE temperatu
 '''
spark.sql(query).explain()

```

```
query =  '''
 SELECT time, temperature_2m, precipitation, rain, snowfall, weathercode from weather_data_table  WHERE temperature_2m  > '60'
 '''
spark.sql(query).explain()

== Physical Plan ==
*(1) Project [time#123, temperature_2m#132, precipitation#129, rain#130, snowfall#131, weathercode#133]
+- *(1) Filter (temperature_2m#132 > 60.0)
   +- BatchScan[time#123, precipitation#129, rain#130, snowfall#131, temperature_2m#132, weathercode#133] Cassandra Scan: project_weather_data.wea
ther_data_table
 - Cassandra Filters: []
 - Requested Columns: [time,precipitation,rain,snowfall,temperature_2m,weathercode]
```

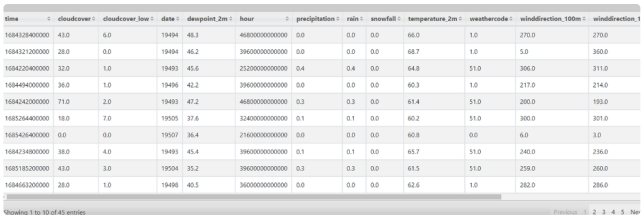
Query in Drill

```
SELECT * from Cassandra.project_weather_data.`weather_data_table`
```



time	cloudcover	cloudcover_low	date	dewpoint_2m	hour	precipitation	rain	snowfall	temperature_2m	weathercode	winddirection_100m	winddirection_10m
1684908000000	80.0	2.0	19501	30.2	21600000000000	0.1	0.1	0.0	49.8	51.0	311.0	301.0
1685406400000	1.0	1.0	19507	37.1	3600000000000	0.0	0.0	0.0	42.6	0.0	143.0	145.0
1684875600000	60.0	14.0	19500	38.6	75600000000000	0.0	0.0	0.0	41.4	2.0	9.0	149.0
1684918800000	85.0	15.0	19501	29.7	32400000000000	0.0	0.0	0.0	51.8	3.0	315.0	288.0
1685394000000	0.0	0.0	19506	31.5	75600000000000	0.0	0.0	0.0	39.7	0.0	99.0	127.0
1684861200000	70.0	23.0	19500	39.2	61200000000000	0.0	0.0	0.0	44.0	2.0	207.0	159.0
1684976400000	6.0	0.0	19502	33.9	3600000000000	0.0	0.0	0.0	39.4	0.0	108.0	150.0
1685473200000	20.0	0.0	19507	37.1	68400000000000	0.0	0.0	0.0	44.4	0.0	81.0	113.0
1684951200000	7.0	0.0	19501	30.5	64800000000000	0.0	0.0	0.0	38.3	0.0	198.0	160.0
1685044800000	5.0	0.0	19502	33.2	72000000000000	0.0	0.0	0.0	38.2	0.0	202.0	166.0

```

SELECT * from cassandra.project_weather_data.`weather_data_table` where temperature_2m  > '60'

```



time	cloudcover	cloudcover_low	date	dewpoint_2m	hour	precipitation	rain	snowfall	temperature_2m	weathercode	winddirection_100m	winddirection_1
1684328400000	43.0	6.0	19494	48.3	46800000000000	0.0	0.0	0.0	66.0	1.0	270.0	270.0
1684321200000	28.0	0.0	19494	46.2	39600000000000	0.0	0.0	0.0	68.7	1.0	5.0	360.0
1684220400000	32.0	1.0	19493	45.6	25200000000000	0.4	0.4	0.0	64.8	51.0	306.0	311.0
1684494000000	36.0	1.0	19496	42.2	39600000000000	0.0	0.0	0.0	60.3	1.0	217.0	214.0
1684242000000	71.0	2.0	19493	47.2	46800000000000	0.3	0.3	0.0	61.4	51.0	200.0	193.0
1685264400000	18.0	7.0	19505	37.6	32400000000000	0.1	0.1	0.0	60.2	51.0	300.0	301.0
1685426400000	0.0	0.0	19507	36.4	21600000000000	0.0	0.0	0.0	60.8	0.0	6.0	3.0
1684234800000	38.0	4.0	19493	45.4	39600000000000	0.1	0.1	0.0	65.7	51.0	240.0	236.0
1685185200000	43.0	3.0	19504	35.2	39600000000000	0.3	0.3	0.0	61.5	51.0	259.0	260.0
1684663200000	28.0	1.0	19498	40.5	36000000000000	0.0	0.0	0.0	62.6	1.0	282.0	286.0

Showing 1 to 10 of 45 entries Previous 1 2 3 4 5 Ne

Conclusion

Cassandra is well suited for time series data given that it can handle writes with high reliability. Since the weather data has hourly data, Cassandra can handle the amount of data with ease. The 311 data also had a large volume of data, but the scope was limited to 2000 rows. The next iteration of this project would be to use the 311 data and show the complaints on a map using Elastic. I attempted to do this but the latitude/longitude was being stored in Elastic as a text variable even after transformations, which is incompatible with being able to be used on a map.