Projekt - konspekt

Patryk Rakus
Łukasz Tomaszewski
Michał Tomczyk

Spis treści

1 Cel projektu

Projekt ma na celu utworzenie systemu Big Data, który pozwoli na przetwarzanie oraz analizę wpływu pogody oraz ruchu drogowego (głównie korków) na jakość powietrza w wybranych przez nas stolicach państw. W ostatnich latach wiele badań donosi o tym jak zanieczyszczenie powietrza zwiększa ryzyko wystąpienia nowotworów. Zanieczyszczenia te w dużej części pochodzą ze spalin samochodowych. Dzięki naszemu systemowi będziemy mogli wskazać jak niebezpieczne dla ludzkiego zdrowia potrafią być kombinacje złej pogody oraz nadmiernego ruchu drogowego. Pomoże to również poinformować obywateli, kiedy warto unikać zbędnego pobytu na świeżym powietrzu.

2 Opis danych

Dane, wykorzystywane w projekcie pochodzą z 3 różnych źródeł. We wszystkich przypadkach pobierane są poprzez API. Wszystkie dane pochodzą zatem z dynamicznych źródeł. W projekcie rozważanych jest 14 stolic:

- Warszawa
- Berlin
- Paryż
- Londyn
- Wiedeń
- Rzym
- Bruksela
- Luksemburg
- Berno
- Zagrzeb
- Waszyngton
- Brasilia
- Nowe Delhi
- Kair

2.1 Dane dotyczące utrudnień w ruchu drogowym

Dane te pochodzą ze strony https://developer.tomtom.com/products/traffic-api. Przedstawiają one aktualne utrudnienia w ruchu drogowym, takie jak:

- korki uliczne
- wypadki drogowe
- zamknięcie dróg
- niebezpieczne warunki drogowe
- i wiele innych

Istnieje możliwość pobrania wielu informacji dotyczących każdego utrudnienia, jednak na rzecz projektu niezbędne są jedynie informacje dotyczące rodzaju utrudnienia, jego współrzędne, moment rozpoczęcia oraz identyfikator. Dodatkowo pobieramy informacje o momencie zakończenia oraz wielkości opóźnienia spowodowanego przez utrudnienie, jeśli są one podane.

2.2 Dane pogodowe

Dane te pochodzą ze strony https://open-meteo.com/. Przedstawiają one zarazem historyczne jak i aktualne dane pogodowe z całego świata. Na rzecz projektu pobierane są następujące informacje:

- Temperatura (2m nad ziemią) w C°
- Opady deszczu w mm
- Opady śniegu w cm
- Widoczność w m
- Prędkość wiatru (10m nad ziemią) w km/h
- Wilgotność gleby (na głębokości 0-1cm) w m^3/m^3

2.3 Dane o zanieczyszczeniu powietrza

Pochodzą ze strony https://openweathermap.org/api/air-pollution. Strona ta pozwala na uzyskanie aktualnych, historycznych oraz przewidywanych stężeń różnych szkodliwych substancji w powietrzu na całym świecie, bazując na współrzędnych geograficznych. Dla danej lokalizacji, w danym momencie czasowym, zwracane są następujące informacje:

- indeks jakości powietrza (Air Quality Index), czyli stopień ogólnego zanieczyszczenia powietrza w skali od 1 do 5
- stężenie czadu (CO)
- stężenie tlenku azotu (NO)
- stężenie dwutlenku azotu (NO_2)
- stężenie ozonu (O_3)
- stężenie dwutlenku siarki (SO_2)
- stężenie pyłu zawieszonego ($PM_{2,5}$)
- stężenie pyłu zawieszonego (PM_{10})
- stężenie amoniaku (NH_3)

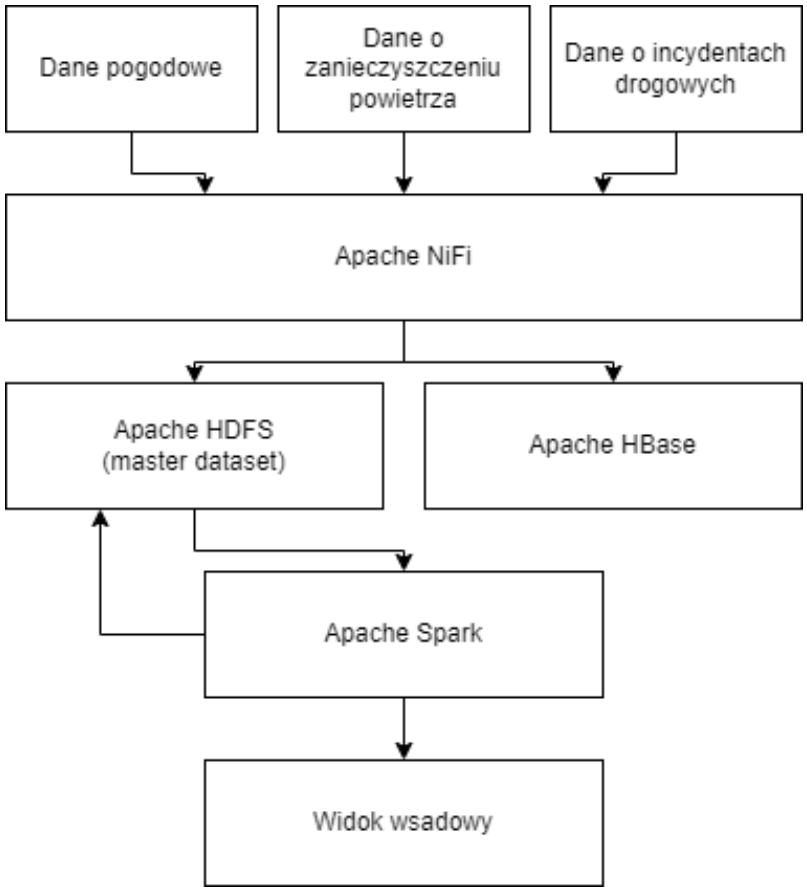
Wszystkie wartości stężenia podane są w $\frac{\mu g}{m^3}$. Zapytania do API będą kierowane dla konkretnych współrzędnych geograficznych. Takie zapytanie zwróci też czas, dla którego dane są pobrane, w postaci czasu uniksowego ($UTC \pm 0$).

3 Zastosowany stos architektoniczny

Stos architektoniczny został zaprojektowany zgodnie z architekturą lambda. Dane trafiają do dwóch warstw składowania: warstwy speed (Apache HBase) oraz batch (Apache HDFS), zawierającej master dataset z pełnymi danymi. Dane przekazywane do warstwy serving (Apache Spark) przekazywane są z HDFS, ponieważ dynamika ich napływu nie jest duża. Cały przepływ danych wygląda następująco:

1. Dane zbierane są przez Apache NiFi przy użyciu API. Dane o utrudnieniach, pogodzie oraz zanieczyszczeniu powietrza pobierane są co godzinę.
2. Dane trafiają do miejsca ich składowania. Wstępnie przetworzone w NiFi dane trafiają do Apache HDFS. Dane te trafiają także do Apache HBase, gdzie składowane są przez 24 godziny.
3. Dane w HDFS są czyszczone, transformowane i agregowane zgodnie z harmonogramem przy pomocy Apache Spark

4. Dane są analizowane i generowane są widoki wsadowe przy użyciu Spark



Rysunek 1: Diagram planowanego stosu architektonicznego

4 Implementacja

Poniżej opisana jest nasza implementacja projektu. Projekt wymaga pobrania projektu z repozytorium do folderu `/home/vagrant`, co powinno skutkować utworzeniem tam folderu `Big-Data-Project2023`.

4.1 Pliki konfiguracyjne

W poniższej sekcji znajduje się opis dodatkowych plików, wymaganych do uruchomienia i modyfikacji działania projektu.

4.1.1 Utworzenie niezbędnych katalogów

W głównym folderze projektu znajduje się plik `setup.sh`. Należy go uruchomić przed korzystaniem z projektu w celu uworzenia katalogów do składowania danych w Apache HDFS. Utworzy on rekursywnie foldery `/user/traffic/nifi/incidents_raw`, `/user/traffic/nifi/weather_raw` oraz `/user/traffic/nifi/pollution_raw` w Apache HDFS.

4.1.2 Pobieranie danych

Pliki niezbędne do pobrania danych poprzez API znajdują się w `Big-Data-Project2023/http_info/` Są to 2 pliki JSON zawierające listy koordynatów.

Pierwszy z plików to `incident_coords.json` zawierający informacje potrzebne do pobrania danych o ruchu drogowym. Pola "minLat", "minLon", "maxLat" i "maxLon" oznaczają odpowiednio szerokość i długość geograficzną 2 przeciwnych rogów prostokąta. Prostokąt ten obejmuje obszar miasta podanego w polu "city". Poniżej znajduje się przykładowa zawartość takiego pliku dla 2 miast.

```
[
    {"minLat": 52.117891, "minLon": 20.888559, "maxLat": 52.326527, "maxLon": 21.149659,
        "city": "Warsaw"},
    {"minLat": 52.398984, "minLon": 13.216745, "maxLat": 52.657144, "maxLon": 13.639092,
        "city": "Berlin"}
]
```

Drugi plik to `weather_coords.json` zaawierający informacje potrzebne do pobrania danych o zanieczyszczeniu powietrza oraz pogodzie. Pola "lat" i "lon" oznaczają odpowiednio szerokość i długość geograficzną centrum miasta podanego w polu "city". Poniżej znajduje się przykładowa zawartość takiego pliku dla 2 miast.

```
[
    {"lat": 52.229675, "lon": 21.01223, "city": "Warsaw"},
    {"lat": 52.520008, "lon": 13.404954, "city": "Berlin"}
]
```

Powyższa struktura pozwala w miarę potrzeby na proste dodawanie kolejnych obiektów.

4.2 Pobieranie, przetwarzanie i składowanie danych w Apache NiFi

W przypadku wszystkich danych sam początek jest bardzo podobny:

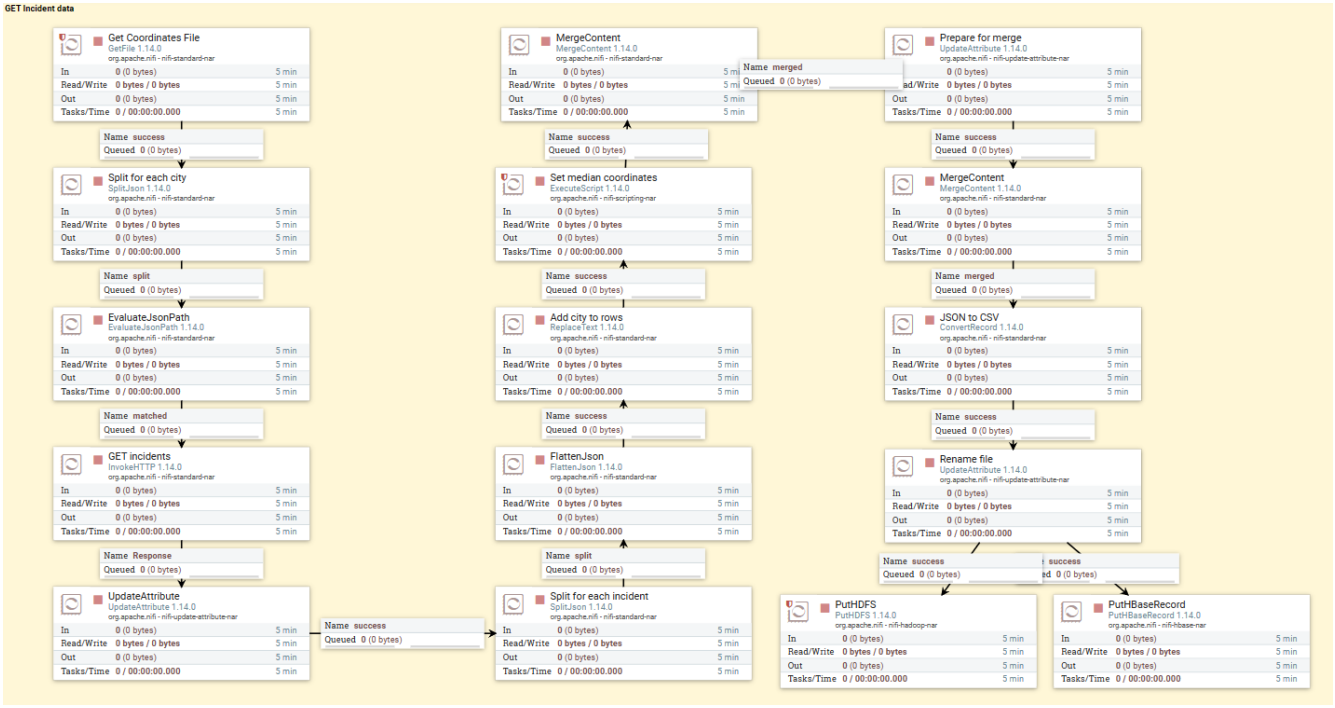
1. Pobierany z folderu `http_info` jest odpowiedni plik JSON.
2. Następnie następuje podział pliku na każde miasto.
3. Z każdego pliku wyciągane są informacje o koordynatach i nazwie miasta.
4. Z wykorzystaniem wyciągniętych danych wysyłane jest zapytanie GET na odpowiedni adres.

4.2.1 Dane o ruchu drogowym

W przypadku danych o ruchu drogowym w następnych krokach tworzony jest kolejny podział.

1. Zapamiętywane są informacje o starym podziale (fragment.index, fragment.count, fragment.identifier)
2. Pliki są dzielone na każdy incydent drogowy.
3. Każdy plik JSON zostaje spłaszczony.
4. Do każdego pliku dodana zostaje informacja w jakim mieście wystąpił.
5. Dla utrudnień, które dotyczą dłuższego fragmentu drogi, jako pozycja wybierana jest mediana koordynatów za pomocą skryptu pythonowego "incident-script.py". Przy okazji upraszczane zostają nazwy pól.
6. Następnie wypadki dla każdego miasta są ponownie łączone w jeden plik
7. Aby móc bezproblemowo połączyć dane dla każdego miasta, wczytywane są dane o starym podziale z punktu 1. i następuje połączenie danych.
8. Połączone dane zostają zamienione w format .csv, nadawana jest im nazwa z daną datą: "incidents-[yyyy]-[MM]-[dd]-[HH].csv"
9. W ostatnim kroku przepływ danych rozdziela się:
   (a) Gotowy plik .csv zostaje umieszczony w Apache HDFS w `/user/traffic/nifi/incidents_raw/`

(b) Dane trafiają do HBase, gdzie rodziną kolumn jest *incident*, a nazwą wiersza jest zawartość kolumny *id*.

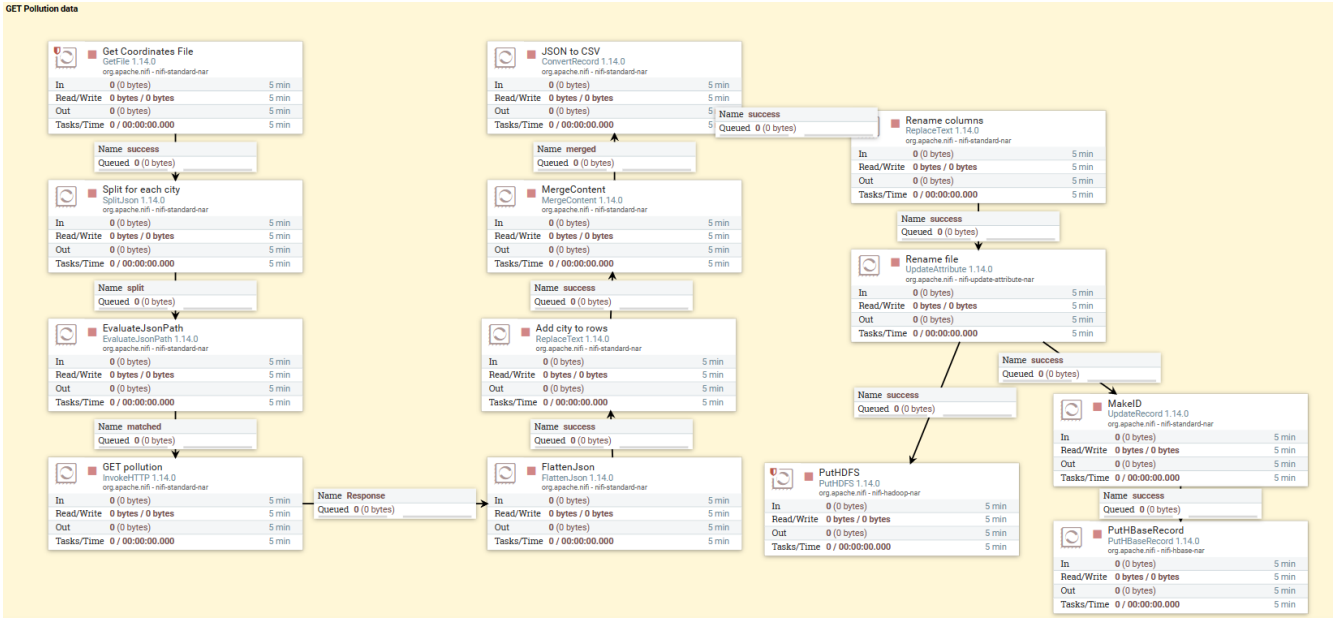


Rysunek 2: Przepływy Apache NiFi dla danych o ruchu drogowym

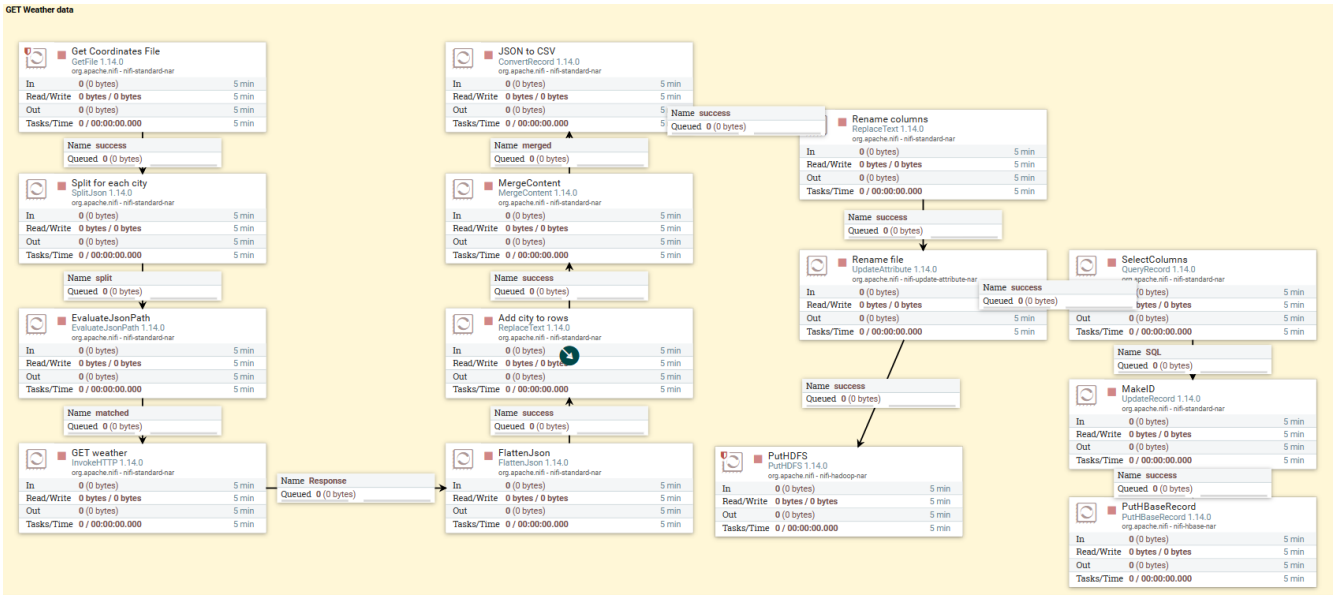
4.2.2 Dane pogodowe oraz o zanieczyszczeniu powietrza

Sposób pobierania i przetwarzania danych dla danych pogodowych i o zanieczyszczeniu powietrza jest bardzo podobny, w skutek czego podany opis adekwatnie opisuje oba przypadki.

1. Odebrany plik JSON zostaje spłaszczony oraz do zawartości dodane zostaje miasto, którego dotyczy.
2. Pliki dotyczące danych dla miast są ponownie łączone i zamieniane w format .csv.
3. Następuje uproszczenie nazw kolumn
4. nadawana jest im nazwa z daną datą: "weather-[yyyy]-[MM]-[dd]-[HH].csv" w przypadku danych pogodowych lub "pollution-[yyyy]-[MM]-[dd]-[HH].csv" w przypadku danych o zanieczyszczeniu powietrza.
5. Dane trafiają do miejsca składowania:
   (a) Gotowy plik .csv zostaje umieszczony w Apache HDFS w `/user/traffic/nifi/weather_raw/` w przypadku danych pogodowych lub `/user/traffic/nifi/pollution_raw/` w przypadku danych o zanieczyszczeniu powietrza.
   (b) Przed trafieniem do HBase, w obydwu przypadkach tworzona jest kolumna *id* składająca się z nazwy miejscowości oraz stempla czasowego. Kolumna ta posłuży jako nazwa wiersza w HBase. Dane pogodowe trafiają do rodziny kolumn *weather*, a dane o zanieczyszczeniu powietrza do rodziny kolumn *pollution*. Dodatkowo z danych pogodowych usuwane są zbędne kolumny, zawierające opis jednostek miary.



Rysunek 3: Przepływy Apache NiFi dla danych o zanieczyszczeniu powietrza



Rysunek 4: Przepływy Apache NiFi dla danych pogodowych

4.3 Składowanie danych w HBase

W Apache Hbase stworzona została tabela *traffic*, w której przechowywane są pobrane dane. W tabeli występują trzy rodziny kolumn: *incident*, *weather* oraz *pollution*, do których trafiają odpowiednio dane o utrudnieniach drogowych, dane pogodowe oraz dane o zanieczyszczeniu powietrza. Dodatkowo, każda z rodzin ma ustawiony TTL o wartości 86400. Oznacza to, że dane, które trafiły do tabeli zostaną z niej automatycznie usunięte po 24 godzinach. Dzięki temu w HBase przechowywać będziemy tylko najświeższe dane.

4.4 Transformowanie i agregacja w Apache Spark

Tworzenie widoków wsadowych zostało zaimplementowane w Apache Spark, korzystając z języka python (a więc z PySpark). Oprócz PySpark, wykorzystywane są też 3 dodatkowe biblioteki pythona: sys, logging oraz

datetime. Dane w Spark są ładowane z HDFS. Dla danego typu danych (zanieczyszczenie powietrza, pogoda oraz zdarzenia drogowe) za pomocą funkcji `merge_tables_from_hdfs` tworzona jest jedna ramka danych, na podstawie wszystkich plików .csv w danym katalogu w Apache HDFS. Oprócz tej funkcji pomocniczej, istnieją także dwie kolejne:

- `round_to_half_hour`: funkcja przyjmuje czas uniksowy i zwraca obiekt datetime z połówką godziny najbliższą przekazanemu czasowi (połówka godziny to godzina X:30, w takich terminach dane są pobierane z API i składowane w HDFS)
- `map_categories`: funkcja mapuje id danej kategorii zdarzenia (w ramkach z danymi o zdarzeniach drogowych) na tekst opisujący to zdarzenie

Generowane są 4 widoki wsadowe. Dla każdego z nich można wybrać miasta, dla których dane mają być prezentowane (poprzez przekazanie do funkcji generującej dany widok argumentu cities z listą z nazwami wybranych miast), a także zakres dat, dla których dane mają być brane pod uwagę (przekazanie do argumentów start_time oraz end_time obiektów datetime). Jeśli te argumenty są przekazane, ramki danych są filtrowane tak aby prezentowane dane dotyczyły odpowiednich miast w odpowiednich terminach. Generowane są następujące widoki wsadowe:

- WeatherAndPollutionData: połączone dane o pogodzie i zanieczyszczeniu powietrza. Dane są ładowane z HDFS, usuwane są z nich niepotrzebne kolumny, a czas danych o zanieczyszczeniu jest transformowany z czasu uniksowego i zaokrąglany do połówki godziny (API o danych o zanieczyszczeniu nie zawsze zwraca dane z wartością timestamp dokładnie odpowiadającej czasowi wysłania zapytania, można jednak przyjąć, że zanieczyszczenie powietrza w wystarczająco wolnym tempie się zmienia żeby móc wykorzystywać dane przesunięte o maksymalnie pół godziny). Następnie wykorzystywana jest operacja join ze względu na przekształcony czas. Dane z tego widoku mogą bardziej służyć do dalszej analizy zależności pogody i zanieczyszczenia powietrza niż służyć jako gotowy widok.
- SumOfIncidentsForCategories: suma zdarzeń drogowych w danych miastach, z podziałem na kategorię zdarzenia. Dane o zdarzeniach są ładowane z HDFS, usuwane są z nich kolumny 'lon' oraz 'lat', na id danej kategorii (kolumna 'iconCat') są mapowane nazwy kategorii. Następnie, jeśli odpowiednie argumenty są przekazane, wyselekcjonowane są zdarzenia, które zaczęły się w zapewnionym terminie (patrzymy tu tylko na to, kiedy zdarzenie się rozpoczęło, bez względu na to czy i kiedy się skończyło) orz wydarzyły się w danych miastach. Potem usuwane są wiersze, których kolumna 'id' jest duplikatem, tak aby dane zdarzenie było zliczane tylko raz. Dalej następuje grupowanie ze względu na miasto oraz kategorię zdarzenia i agregacja z użyciem count().
- DailyWeatherData: zagregowane względem dnia i miasta dane o pogodzie, dla danego typu agregacji oraz danych kolumn (funkcja generująca przyjmuje zatem 2 dodatkowe argumenty w porównaniu z poprzednimi: columns, zawierający listę kolumn, na których przeprowadzana będzie agregacja oraz type, opisujący typ agregacji - 'mean', 'max' albo 'sum'). Dane są pobierane z HDFS, usuwane są z nich niepotrzebne kolumny, tworzona jest także kolumna 'day' zawierająca datę dla danego wiersza (zamiast datetime z datą oraz godziną). Następnie wykonywana jest filtracja ze względu na miasta oraz zakres czasowy, dane są grupowane ze względu na miasto oraz datę dzienną i agregowane w odpowiedni sposób
- DailyPollutionData: zagregowane względem dnia i miasta dane o zanieczyszczeniu powietrza, dla danego typu agregacji oraz danych kolumn. Funkcja generująca widok działa analogicznie jak ta generująca DailyWeatherData, tylko na danych o zanieczyszczeniu powietrza.

5 Testy

5.1 Utworzenie niezbędnych katalogów w HDFS

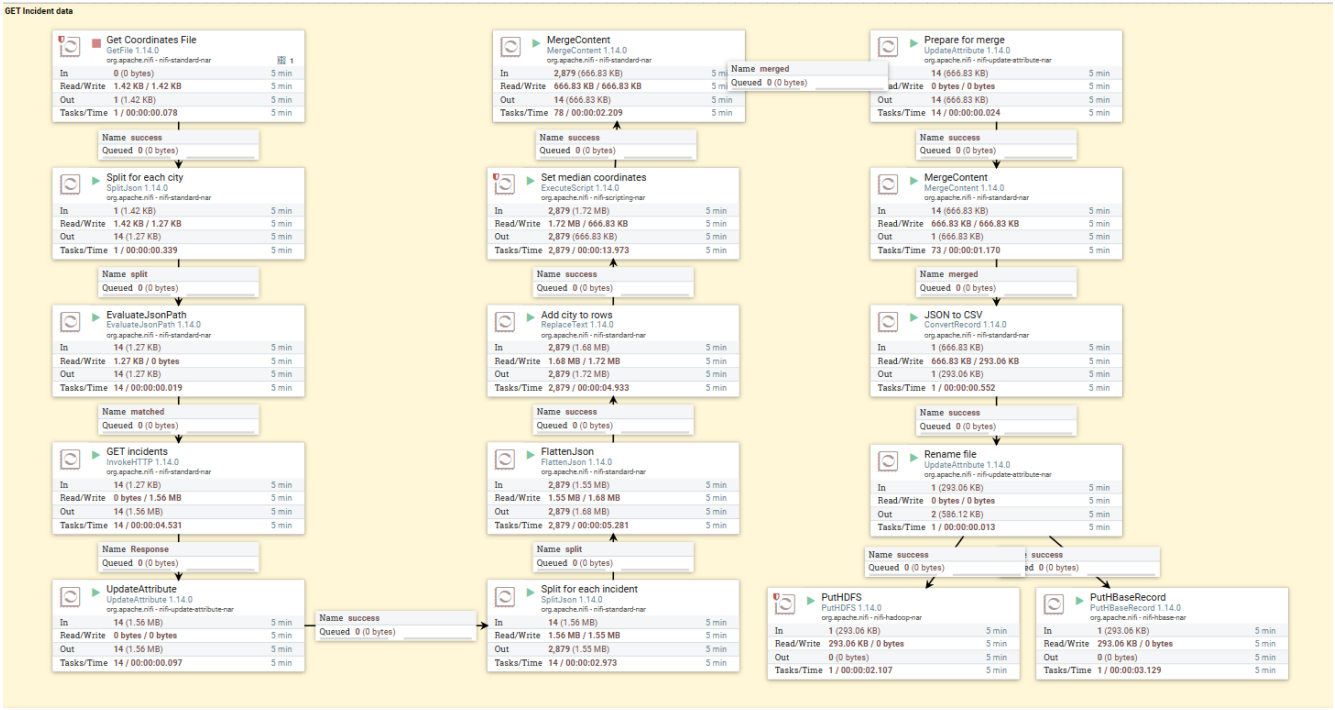
Testujemy działanie skryptu `setup.sh`. Po wykonaniu skryptu, w HDFS powinny pojawić się 3 katalogi: `/user/traffic/nifi/incidents_raw`, `/user/traffic/nifi/weather_raw` oraz `/user/traffic/nifi/pollution_raw`. W konsoli powinny zostać wypisane stworzone katalogi.

```
vagrant@node1:~$ bash Big-Data-Project2023/setup.sh
Making nifi dirs
There should be 3 dirs listed now
Found 3 items
drwxr-xr-x   - vagrant supergroup          0 2024-01-10 18:47 /user/traffic/nifi/
    incidents_raw
drwxr-xr-x   - vagrant supergroup          0 2024-01-10 18:47 /user/traffic/nifi/
    pollution_raw
drwxr-xr-x   - vagrant supergroup          0 2024-01-10 18:47 /user/traffic/nifi/weather_raw
```

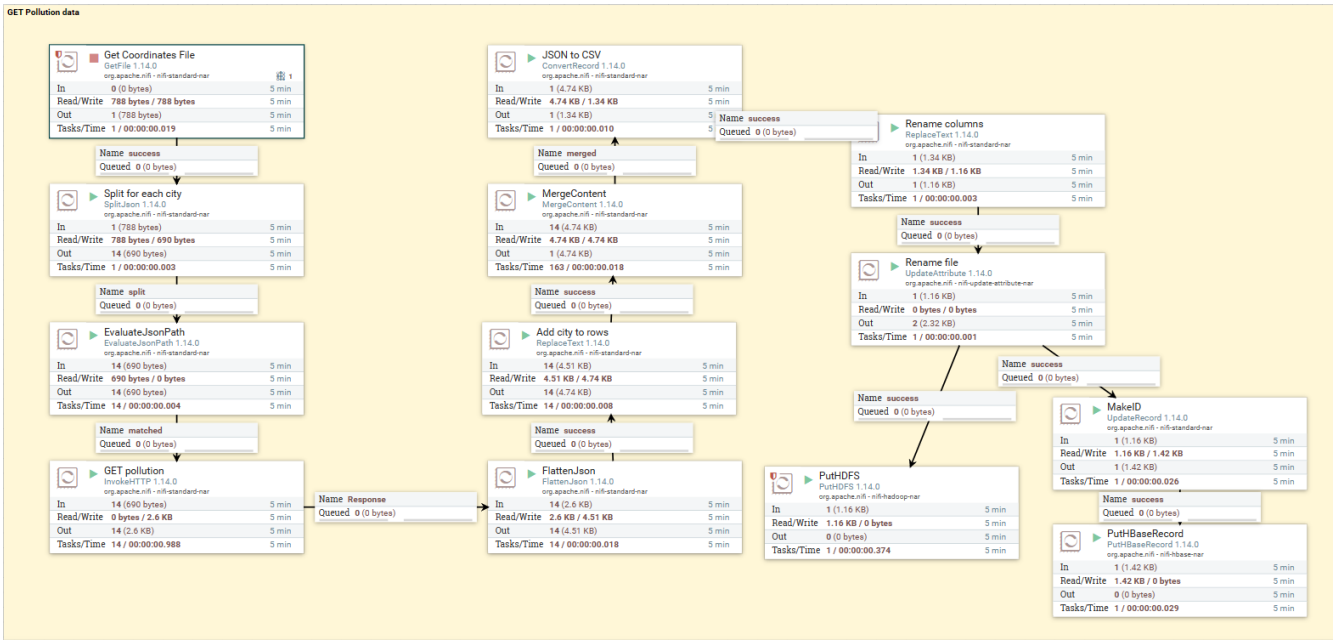
Skrypt zadziałał zgodnie z założeniami i w HDFS zostały utworzone odpowiednie katalogi.

5.2 Przepływy w Apache NiFi

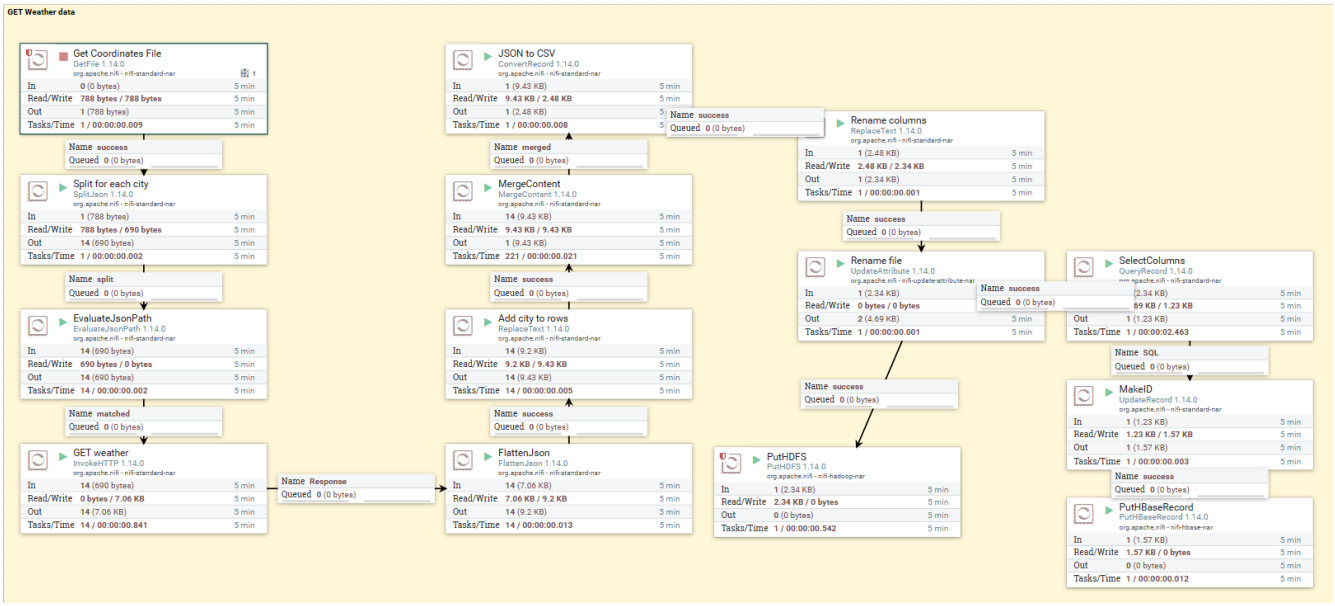
Testujemy działanie przepływów w Apache Nifi. Uruchamiamy wszystkie procesory i sprawdzamy, czy następuje prawidłowy przepływ danych, który nie zwraca żadnych błędów. Procesory, które są pierwsze w przepływie uruchamiane są z opcją *Run Once*.



Rysunek 5: Test przepływu danych o utrudnieniach drogowych



Rysunek 6: Test przepływu danych o zanieczyszczeniu powietrza



Rysunek 7: Test przepływu danych pogodowych

Na powyższych screenach możemy zobaczyć, że przepływy pobierają dane dla 14 miast. Działają one w sposób prawidłowy i nie zwracają błędów.

5.3 Załadowanie danych do HDFS

Sprawdzamy, czy pliki .csv są ładowane do HDFS oraz czy mają odpowiednią strukturę.

```
vagrant@node1:~$ hadoop fs -ls /user/traffic/nifi/incidents_raw
Found 1 items
-rw-r--r--   1 root supergroup     300096 2024-01-10 19:02 /user/traffic/nifi/incidents_raw/
    incidents-2024-01-10-19.csv
```

```
vagrant@node1:~$ hadoop fs -ls /user/traffic/nifi/pollution_raw
Found 1 items
-rw-r--r--   1 root supergroup       1190 2024-01-10 19:04 /user/traffic/nifi/pollution_raw/
    pollution-2024-01-10-19.csv

vagrant@node1:~$ hadoop fs -ls /user/traffic/nifi/weather_raw
Found 1 items
-rw-r--r--   1 root supergroup       2399 2024-01-10 19:05 /user/traffic/nifi/weather_raw/
    weather-2024-01-10-19.csv
```

Z powyższych logów wynika, że dane są ładowane do HDFS.

```
city,start,lon,magnitudeOfDelay,iconCat,end,id,lat
Warsaw,2023-07-30T15:51:30Z,20.8895761894,4,8,,08fb285613515ba9ddc64008543174bd,52.2274643485
Warsaw,2023-07-30T17:07:30Z,20.8895761894,4,8,,36de8a843f6fc71a970bd48142087ea9,52.2274643485
Warsaw,2024-01-05T07:11:30Z,20.8922503518,4,8,,0e801406aae14ce96602aa547d66e8aa,52.2336441541
Warsaw,2024-01-04T15:59:30Z,20.8923750745,4,8,,bd5ab0258380ea9785dac6ed2f96095b,52.2339365025
Warsaw,2024-01-10T14:04:26Z,20.8973009514,4,8,2024-01-12T22:59:00Z,a8e4f8c7f23a5505d4b428f864c1b2a6,52.1980767008
Warsaw,2023-10-02T18:10:30Z,20.8992737161,4,8,,f224f87b65cbef4b4ad9302fdb6b783c,52.2397756732
Warsaw,2023-10-01T19:01:30Z,20.9157290684,4,8,,a0b79b83666eec2b78a73a29f2784909,52.2070017837
Warsaw,2023-02-19T21:00:00Z,20.9158551323,4,8,2024-03-29T22:00:00Z,8026ea09fcef8e945f8a88c7418f12f3,52.2067590312
Warsaw,2023-10-27T10:31:00Z,20.916659795,4,8,,132306ad121c295d3d8e8bdf8e11dfe5,52.2070393316
```

Rysunek 8: Dane o utrudnieniach drogowych

```
city,lon,lat,aqi,co,no,no2,o3,so2,pm2_5,pm10,nh3,timestamp
Warsaw,21.0122,52.2297,2,340.46,0.01,11.82,46.49,12.99,13.39,17.31,3.93,1704912955
Berlin,13.405,52.52,2,660.9,37.55,54.84,0.01,22.65,23.22,26.31,12.16,1704913101
Paris,2.347,48.859,2,540.73,1.08,47.3,14.13,10.13,24.55,27.64,7.41,1704912992
London,-0.1257,51.5085,1,270.37,0.01,18.34,56.51,5.72,1.63,2.85,0.72,1704913257
Vienna,16.3738,48.2082,2,514.03,0.04,33.93,27.54,9.54,15.15,18.2,5.45,1704913490
Rome,12.4964,41.9028,2,607.49,5.2,24.68,2.62,1.27,13.54,16.23,1.95,1704913490
Brussels,4.3517,50.8503,3,494,7.49,71.97,0.18,17.17,17.47,21.08,16.47,1704913490
Luxembourg,6.1296,49.61,2,353.81,0.03,31.19,33.26,7.21,12.22,13.94,6.02,1704913490
Bern,7.455,46.9534,3,614.17,12.41,43.87,0.01,7.57,28.36,30.84,7.03,1704913459
Zagreb,15.9819,45.815,2,407.22,0.18,24.33,15.02,3.13,18.26,25.42,5.26,1704913490
Washington,-77.0369,38.9072,1,300.41,0.48,4.97,52.21,0.65,0.5,0.77,0.31,1704913490
Brasilia,-47.9218,-15.8267,1,273.71,0.23,5.1,39.7,1.24,0.56,1.44,0.93,1704913491
New Delhi,77.209,28.6139,5,1308.44,0,21.08,76.53,12.28,255.93,277.54,3.14,1704913491
Cairo,31.2357,30.0444,3,293.73,0,12.51,55.79,11.44,16.18,66.18,4.56,1704913156

```

Rysunek 9: Dane o zanieczyszczeniu powietrza

Rysunek 10: Dane pogodowe

Z powyższych screenów wynika, że do HDFS ładowane są odpowiednie dane. Ładowane są dane z wybranych 14 miast z godziny, w której zostały one pobrane.

5.4 Załadowanie danych do HBase

Testujemy ładowanie danych do HBase. W tym celu wykonamy przykładowe zapytania oraz sprawdzimy czy w tabeli *traffic* znajduje się tyle samo wierszy co w danych zapisanych do HDFS (2907).

```
hbase(main):004:0> scan 'traffic', {'LIMIT' => 1}
ROW                                             COLUMN+CELL
 0003bdb28344e744d83ac973e892a667               column=incident:city, timestamp=2024-01-10T
    19:02:39.192, value=Berlin
 0003bdb28344e744d83ac973e892a667               column=incident:iconCat, timestamp=2024-01-
    10T19:02:39.192, value=8
 0003bdb28344e744d83ac973e892a667               column=incident:lat, timestamp=2024-01-10T1
    9:02:39.192, value=52.4616815527
 0003bdb28344e744d83ac973e892a667               column=incident:lon, timestamp=2024-01-10T1
    9:02:39.192, value=13.4382868915
 0003bdb28344e744d83ac973e892a667               column=incident:magnitudeOfDelay, timestamp
    =2024-01-10T19:02:39.192, value=4
 0003bdb28344e744d83ac973e892a667               column=incident:start, timestamp=2024-01-10
    T19:02:39.192, value=2023-10-27T06:31:06Z
1 row(s)
Took 0.0150 seconds
```

```
hbase(main):004:0> scan 'traffic', {'LIMIT' => 1, COLUMNS => 'weather'}
ROW                                             COLUMN+CELL
 Berlin 2024-01-10T19:00                        column=weather:city, timestamp=2024-01-10T1
    9:05:22.431, value=Berlin
 Berlin 2024-01-10T19:00                        column=weather:elevation, timestamp=2024-01
    -10T19:05:22.431, value=38.0
 Berlin 2024-01-10T19:00                        column=weather:lat, timestamp=2024-01-10T19
    :05:22.431, value=52.52
 Berlin 2024-01-10T19:00                        column=weather:lon, timestamp=2024-01-10T19
    :05:22.431, value=13.4
 Berlin 2024-01-10T19:00                        column=weather:rain, timestamp=2024-01-10T1
    9:05:22.431, value=0.0
 Berlin 2024-01-10T19:00                        column=weather:snowfall, timestamp=2024-01-
    10T19:05:22.431, value=0.0
 Berlin 2024-01-10T19:00                        column=weather:soil_moisture_0_1cm,
    timestamp=2024-01-10T19:05:22.431, value=0.3
 Berlin 2024-01-10T19:00                        column=weather:temperature_2m, timestamp=20
    24-01-10T19:05:22.431, value=-5.8
```

```
 Berlin 2024-01-10T19:00                          column=weather:time, timestamp=2024-01-10T19:05:22.431, value=2024-01-10T19:00
 Berlin 2024-01-10T19:00                          column=weather:timezone, timestamp=2024-01-10T19:05:22.431, value=GMT
 Berlin 2024-01-10T19:00                          column=weather:utc_offset_seconds, timestamp=2024-01-10T19:05:22.431, value=0
 Berlin 2024-01-10T19:00                          column=weather:visibility, timestamp=2024-01-10T19:05:22.431, value=24140.0
 Berlin 2024-01-10T19:00                          column=weather:windspeed_10m, timestamp=2024-01-10T19:05:22.431, value=3.3
1 row(s)
Took 0.0216 seconds
```

```
hbase(main):022:0> count 'traffic'
Current count: 1000, row: 5a1b7fe7b40eef74f003bec3d5db91e1
Current count: 2000, row: af8c6009506b55143c8116283d6d3870
2907 row(s)
Took 1.0076 seconds
=> 2907
```

Z powyższych logów wynika, że przykładowe zapytania zwracają pożądane wyniki. Liczba wierszy w HBase jest zgodna z liczbą wierszy danych zapisanych w HDFS.

5.5 Automatyczne usuwanie danych z HBase

Testujemy automatyczne usuwanie danych z HBase. Po 24 godzinach od dodania danych, w tabeli nie powinno być żadnych wierszy.

```
vagrant@node1:~$ date
Thu Jan 11 19:24:26 UTC 2024
vagrant@node1:~$ /usr/local/hbase/bin/hbase shell
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.3.5, rfd3fdc08d1cd43eb3432a1a70d31c3aece6ecabe, Thu Mar 25 20:50:15 UTC 2021
Took 0.0023 seconds
hbase(main):001:0> scan 'traffic'
ROW                                              COLUMN+CELL
0 row(s)
Took 0.5218 seconds
```

Na powyższych logach możemy zobaczyć, że tabela *traffic* jest pusta.

5.6 Generowanie widoków w Apache Spark

Testujemy, czy funkcje z pliku `spark_processing_batch.py` poprawnie generują widoki wsadowe. W tym celu tworzymy testowy skrypt w języku python `test.py`, który importuje plik `spark_processing_batch.py`, przywołuje funkcje tworzące widoki oraz sprawdza zawartość utworzonych widoków za pomocą zapytania SQL. Dla każdego typu widoków wykonywany jest jeden test. Testy wykonywane są za pomocą wypisania komendy `spark-submit -master local[2] test.py`, będąc w katalogu zawierającym pliki projektowe.
Dla WeatherAndPollutionData przekazujemy do argumentu funkcji miasto Warszawę oraz zakres czasowy od 29.12.2023 18:30 do 21:12:2023 00:00. W zapytaniu SQL select przeprowadzanym na widoku dodatkowo wybieramy kolumny 'city', 'time', 'temperature_2m', 'rain', 'pm2_5', 'co', 'o3'. Dodatkowo ustawiamy warunek aby wybrać tylko dane wcześniejsze niż 30.12.2023 11:30.

```
[2024-01-10 21:38:37,670]INFO @ line 43: Merging weather and pollution data...
[2024-01-10 21:38:58,116]INFO @ line 66: Done
+------+-------------------+-------------+----+-----+------+-----+
|  city|               time|temperature_2m|rain|pm2_5|    co|   o3|
+------+-------------------+-------------+----+-----+------+-----+
|Warsaw|2023-12-29 18:30:00|          7.8| 0.0| 3.33|277.04|61.51|
|Warsaw|2023-12-29 19:30:00|          7.5| 0.0|  3.4|277.04|61.51|
|Warsaw|2023-12-29 20:30:00|          7.6| 0.0| 3.57|277.04| 60.8|
|Warsaw|2023-12-29 21:30:00|          7.8| 0.0|  3.4|273.71|59.37|
|Warsaw|2023-12-29 22:30:00|          7.5| 0.0| 3.08|270.37|58.65|
|Warsaw|2023-12-29 23:30:00|          7.0| 0.0| 2.94|267.03|57.22|
+------+-------------------+-------------+----+-----+------+-----+
```

Rysunek 11: Widok WeatherAndPollutionData

Z powyższych logów możemy zauważyć, że pokazane komendą show() dane zgadzają się z oczekiwaniami. Nie są to pełne dane, gdyż maszyna wirtualna nie była uruchomiona przez cały przekazany w argumentach okres czasowy, stąd nie wszystkie dane zostały pobrane poprzez Apache NiFi.
Dla SumOfIncidentsForCategories do argumentu przekazujemy tylko liczbę miast, zawierającą 'Warsaw' 'London' oraz 'Luxembourg', a w zapytaniu SQL wybieramy wszystkie kolumny bez dodatkowych restrykcji.

```
[2024-01-10 21:39:07,767]INFO @ line 72: Getting incidents in a period data...
[2024-01-10 21:39:15,443]INFO @ line 87: Done
+----------+--------------------+-----+
|      city|            category|count|
+----------+--------------------+-----+
|    London| Broken Down Vehicle|    3|
|    London|Dangerous Conditions|    4|
|    London|                 Jam| 5102|
|    London|         Road Closed|  925|
|    London|          Road Works|  634|
|    London|             unknown|    4|
|Luxembourg| Broken Down Vehicle|    3|
|Luxembourg|                 Jam|   63|
|Luxembourg|         Road Closed|    7|
|Luxembourg|          Road Works|   10|
|    Warsaw|                 Jam|  568|
|    Warsaw|         Road Closed|  244|
|    Warsaw|          Road Works|   78|
+----------+--------------------+-----+
```

Rysunek 12: Widok SumOfIncidentsForCategories

Te dane również zgadzają się z oczekiwaniami.
Dla DailyWeatherData do agregacji wybierane są kolumny 'temperature_2m' oraz 'soil_moisture_0_1cm', czas początkowy jest ustawiany na 28.12.2023 8:30, końcowy na 31.12.2023 00:00, a typ agregacji na 'mean'. W zapytaniu SQL wybierane są miasta 'Warsaw', 'Brasilia' oraz 'London'.

```
[2024-01-10 21:39:32,008]INFO @ line 93: Getting daily weather data...
[2024-01-10 21:39:34,026]INFO @ line 117: Done
+--------+----------+--------------------+-----------------------+
|    city|       day|avg(temperature_2m)|avg(soil_moisture_0_1cm)|
+--------+----------+--------------------+-----------------------+
|Brasilia|2023-12-29|              25.187|                  0.295|
|Brasilia|2023-12-30|                22.4|                  0.325|
|  London|2023-12-29|               9.361|                  0.358|
|  London|2023-12-30|               6.738|                  0.354|
|  Warsaw|2023-12-29|               6.278|                  0.353|
|  Warsaw|2023-12-30|                 7.0|                  0.359|
+--------+----------+--------------------+-----------------------+
```

Rysunek 13: Widok DailyWeatherData

Tutaj także wygląd loga świadczy o powodzeniu testu.
Dla DailyPollutionData do agregacji wybierane są kolumny ’pm10’, ’pm2_5’, ’co’ i ’aqi’, miasta ’Warsaw’, ’Bern’ i ’New Delhi’, przedział czasowy od 28.12.2023 8:30 do 31.12.2023 0:00. W zapytaniu wybieramy wszystkie kolumny bez restrykcji.

```
[2024-01-10 21:39:39,528]INFO @ line 123: Getting daily pollution data...
[2024-01-10 21:39:41,542]INFO @ line 146: Done
+---------+----------+---------+----------+--------+--------+
|     city|       day|avg(pm10)|avg(pm2_5)| avg(co)|avg(aqi)|
+---------+----------+---------+----------+--------+--------+
|     Bern|2023-12-29|    3.706|     3.059|   267.9|   1.739|
|     Bern|2023-12-30|    4.964|     4.377| 281.216|     1.0|
|New Delhi|2023-12-29|  704.055|   597.498|3567.739|     5.0|
|New Delhi|2023-12-30|  708.319|   642.564|3558.158|     5.0|
|   Warsaw|2023-12-29|   10.127|      7.88|  301.86|   1.826|
|   Warsaw|2023-12-30|    3.394|     2.638| 270.784|   1.125|
+---------+----------+---------+----------+--------+--------+
```

Rysunek 14: Widok DailyPollutionData

W tym wypadku ponownie logi zwracają poprawny wygląd danych.
Wszystkie testy wygenerowały widoki, dla których komenda show() pokazuje poprawnie wyglądające ramki danych, zawierające dane zgodne z wymaganiami (odpowiednie miasta, czy zakresy dat). W testach filtracja danych odbywała się zarówno na poziomie przekazywania argumentów do funkcji generujących widoki, jak i za pomocą zapytań SQL na tworzonych widokach. Pokazuje to, że generowanie widoków w Apache Spark działa prawidłowo i zgodnie z oczekiwaniami.

6 Podsumowanie

Projekt, który wykonaliśmy pokazał jak można w praktyce wykorzystywać dane pogodowe, o zanieczyszczeniu powietrza i o zdarzeniach drogowych. Umożliwiliśmy dalszą analizę tych danych poprzez automatyzację ich przepływu i składowania, a także generację wartościowych z analitycznego punktu widzenia widoków wsadowych. W testach dowiedliśmy, że udało nam się zgodnie z oczekiwaniami zaimplementować założenia teoretyczne. Projekt wykazał także użyteczność narzędzi Big Data, dzięki którym łatwo można pobierać, składować, przetwarzać i analizować dane różnego typu i o różnej tematyce.

7 Podział pracy

- Przetwarzanie, agregowanie danych oraz tworzenie widoków wsadowych w Apache Spark - Michał Tomczyk
- Przetwarzanie, ładowanie oraz składowanie danych w HBase, wykonanie testów - Łukasz Tomaszewski
- Mechanizm pobierania, przetwarzania i składowania danych w Apache HDFS z wykorzystaniem Apache NiFi - Patryk Rakus