

Projekt Big Data - sprawozdanie

Wybrane Algorytmy i Systemy Przetwarzania Danych

Mikołaj Jakubowski, Wojciech Szczypek
283745, 298843

20.01.2022

1 Wstęp

Celem projektu było stworzenie architektury systemu do rozwiązywania problemu Big Data. Nasze rozwiązanie bazuje na doskonale znanej *Lambda architecture*. W ten sposób zapewniamy, że nasze rozwiązanie jest łatwo skalowalne, a także wykonane zgodnie z dobrymi praktykami. W trakcie trwania projektu zbierane i analizowane były dane dotyczące zarówno pogody, jak i zanieczyszczenia powietrza.

2 Dane

Wszystkie dane wykorzystane w projekcie, pochodzą z zapytań dwóch API różnych stacji pogodowych. Z [weatherapi](#) zostały pobrane wszelkie pomiary dotyczące pogody. Za pomocą [openweathermap](#) za to, pozyskiwane są pomiary dotyczące zanieczyszczenia powietrza. Oba endpointy dostarczają dane w formacie *JSON*.

Dane z obu źródeł napływają z częstotliwością rekord na godzinę, dla każdego odpytywanego miasta (w celu pokazania działającego produktu zostało odpytane tylko 5 miast, jednak istnieje możliwość skalowania do wszystkich miast na świecie).

Źródło [weatherapi](#) wymaga podania nazwy miasta w zapytaniu, [openweathermap](#) współrzędnych geograficznych.

2.1 Przykładowa odpowiedź API dot. danych pogodowych

```
1 {
2   "location":{
3     "name":"London",
4     "region":"City of London, Greater London",
5     "country":"United Kingdom",
6     "lat":51.52,
7     "lon":-0.11,
8     "tz_id":"Europe/London",
9     "localtime_epoch":1642703574,
10    "localtime":"2022-01-20 18:32"
11  },
12  "current":{
13    "last_updated_epoch":1642702500,
14    "last_updated":"2022-01-20 18:15",
15    "temp_c":4.0,
16    "temp_f":39.2,
17    "is_day":0,
18    "condition":{
```

```

19     "text": "Overcast",
20     "icon": "//cdn.weatherapi.com/weather/64x64/night/122.png",
21     "code": 1009
22 },
23     "wind_mph": 3.8,
24     "wind_kph": 6.1,
25     "wind_degree": 330,
26     "wind_dir": "NNW",
27     "pressure_mb": 1038.0,
28     "pressure_in": 30.65,
29     "precip_mm": 0.0,
30     "precip_in": 0.0,
31     "humidity": 60,
32     "cloud": 100,
33     "feelslike_c": 0.7,
34     "feelslike_f": 33.2,
35     "vis_km": 10.0,
36     "vis_miles": 6.0,
37     "uv": 1.0,
38     "gust_mph": 10.5,
39     "gust_kph": 16.9
40 }
41 }

```

2.2 Przykładowa odpowiedź API dot. danych o zanieczyszczeniu powietrza

```

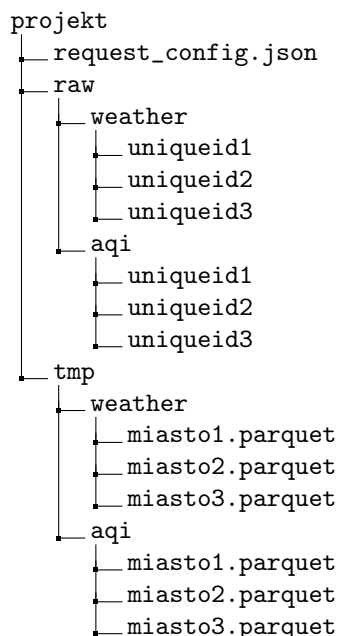
1 {
2     "coord": {
3         "lon": 50,
4         "lat": 50
5     },
6     "list": [
7         {
8             "main": {
9                 "aqi": 1
10            },
11            "components": {
12                "co": 277.04,
13                "no": 0,
14                "no2": 0.42,
15                "o3": 60.08,
16                "so2": 0.32,
17                "pm2_5": 0.55,
18                "pm10": 0.58,
19                "nh3": 0.19
20            },
21            "dt": 1642705200
22        }
23    ]
24 }

```

3 Architektura

Podstawą architektury naszego projektu jest NiFi. To tam znajdują się wszystkie procesory zarządzające flow całego projektu. Za pośrednictwem NiFi zaciągany jest config do odpytywania API, wysyłane są odpowiednie requesty, zapisywane dane, uruchamiane skrypty sparkowe, a także dokonywany processing danych. W poniższej sekcji znajdują się opisy poszczególnych części odpowiadających za warstwę pozyskiwania danych, batch layer, serving layer, speed layer. Całość diagramu, można zobaczyć na zdjęciu, natomiast każda z w.w. części jest dokładniej opisana poniżej. Cały system jest skoordynowany za pomocą programu do harmonogramowania CRON.

3.1 Struktura plików zapisywanych na HDFS



3.2 Uruchamianie workflow na podstawie pliku konfiguracyjnego

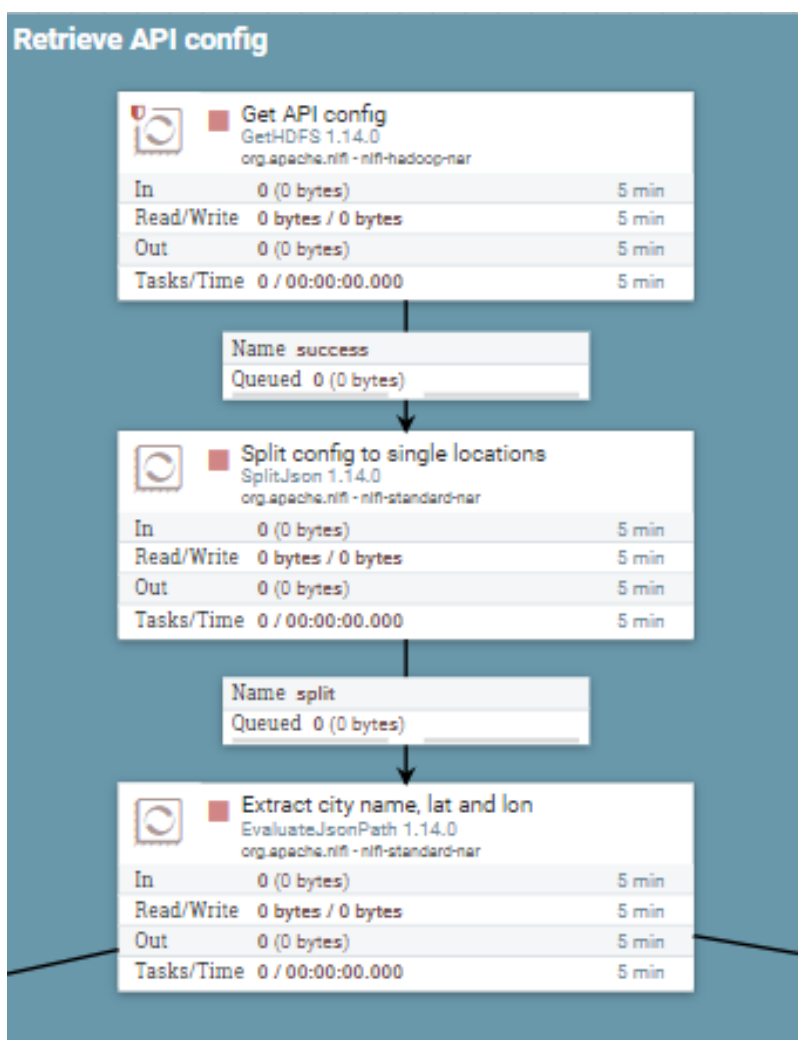
W celu zapewnienia skalowalności rozwiązania został przygotowany plik konfiguracyjny, który zawiera dane, które są używane jako headeary w requestach wysyłanych do API. Definiują one, jakie miejsca są odpytywane - nazwa miasta oraz jego długość i szerokość geograficzna, ponieważ każde z API potrzebuje nieco innych danych wsadowych.

Przykładowy plik konfiguracyjny znajduje się poniżej:

```
1 {
2   "requests": [
3     {
4       "city": "Radom",
5       "lat": "50",
6       "lon": "20"
7     },
8     {
9       "city": "London",
10      "lat": "50",
11      "lon": "50"
12    }
13  ]
14 }
```

Część NiFi odpowiadająca za przetwarzanie danych konfiguracyjnych i przekazanie ich do zapytań API znajduje się na rysunku 1. Składa się z następujących procesorów:

- **Get API config** - odpowiedzialny za zaciągnięcie configu z pliku na hdfsie (zakładamy, że to może być bardzo duży plik)
- **Split config to single locations** - rozdzielenie jednego pliku w formacie json, na jsony zawierające pojedyncze lokacje wraz z koordynantami
- **Extract city name, lat, long** - wyciągnięcie z plików koordynantów oraz nazwy miejsca i dodanie ich do atrybutów FlowFiles, tak aby można się było do nich odnieść w zapytaniach kierowanych do API



Rysunek 1: Część w NiFi odpowiadająca za przetwarzanie configu

3.3 Transformacje danych pogodowych

Biorąc pod fakt, że dane pogodowe znajdują się w dość zagnieżdżonym pliku json, wymagają one spłaszczenia, a także odpowiedniego processingu zanim zostaną zapisane jako pliki RAW. Część NiFi odpowiadająca za pobieranie i przetwarzanie danych pogodowych znajduje się na rysunku 2. Składa się z następujących procesorów:

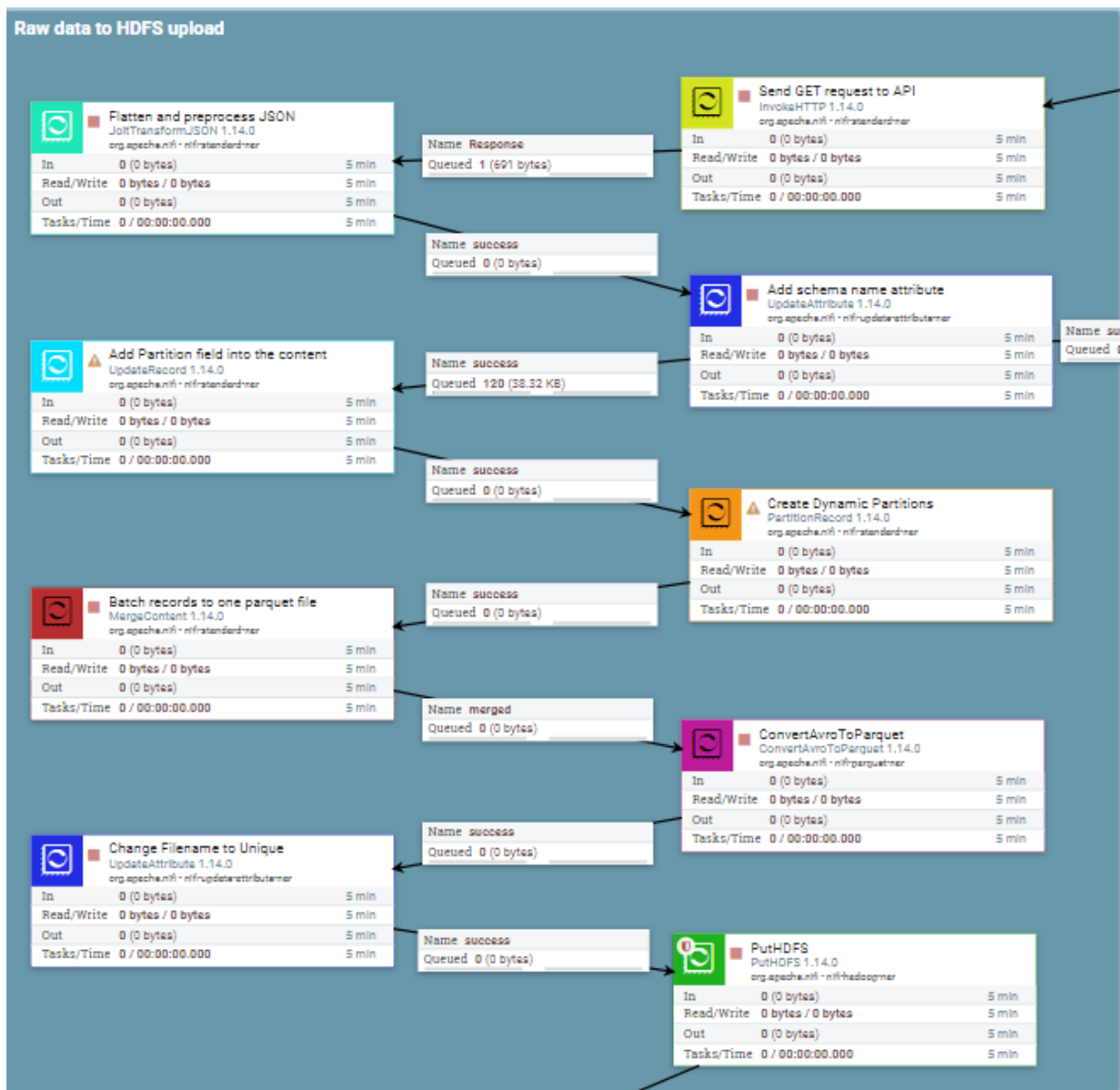
- **Send GET request to API** - odpowiedzialny za wysłanie i odczytanie zapytania z API, na podstawie podanego przez config miasta
- **Flatten and preprocess JSON** - przekształcenie zagnieżdżonego pliku JSON do płaskiej postaci, oraz rozszerzenie nazw pól do pełnych słów za pomocą transformacji JOLT
- **Add schema name attribute** - dodanie schematu Avro do atrybutu FlowFile
- **Add Partition field into the content** - walidacja, czy we wchodzących danych jest zachowany wcześniej zdefiniowany format Avro. W ten sposób sprawdzamy, czy format zwracanych danych przez API nie uległ przypadkiem zmianie.
- **Create Dynamic Partitions** - Stworzenie partycji, pod którymi zapisywane będą dane na HDFSie. Obecnie jest są to partycje po dacie w formacie "yyyy-MM-dd"
- **Batch records to one parquet file** - Batchowanie rekordów do jednego pliku, tak aby nie zapisywać każdego rekordu w osobnym pliku
- **ConvertAvroToParquet** - Zamiana pliku w formacie Avro na Parqueta
- **Change Filename to Unique** - Nadanie unikalnej nazwy plikowi - atrybut UUID FlowFile
- **PutHDFS** - Zapisanie pliku na HDFSie w zdefiniowanych wcześniej partycjach w folderze

3.4 Transformacje danych o zanieczyszczeniach

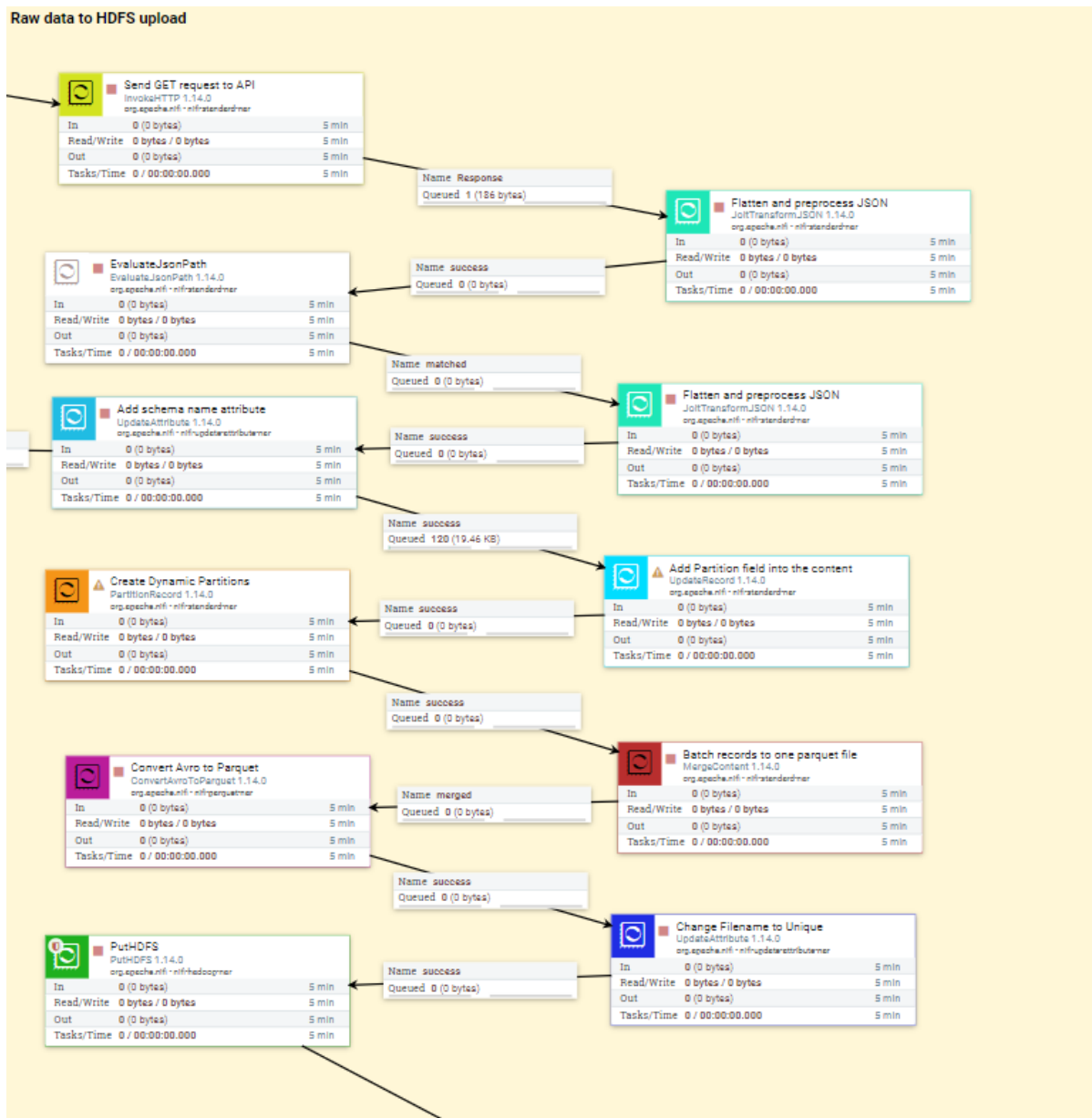
Dane o zanieczyszczeniu również są pobierane w specyficznym formacie

Część NiFI odpowiadająca za za pobieranie i przetwarzanie danych dot. zanieczyszczenia powietrza znajduje się na rysunku 3. Składa się z następujących procesorów:

- **Send GET request to API** - odpowiedzialny za wysłanie i odczytanie zapytania z API, na podstawie podanych przez config długości i szerokości geograficznej
- **Flatten and preprocess JSON** - przekształcenie zagnieżdżonego pliku JSON do płaskiej postaci, oraz rozszerzenie nazw pól do pełnych słów za pomocą transformacji JOLT
- **EvaluateJsonPath** - dodanie do atrybutu pliku wartości mówiącej o czasie pomiaru w formacie unix timestamp
- **Flatten and preprocess JSON** - przekształcenie formatu daty do formatu "yyyy-MM-dd"tak, aby była spójna z datą w rekordach dot. pogody
- **Add schema name attribute** - dodanie schematu Avro do atrybutu FlowFile
- **Add Partition field into the content** - walidacja, czy we wchodzących danych jest zachowany wcześniej zdefiniowany format Avro. W ten sposób sprawdzamy, czy format zwracanych danych przez API nie uległ przypadkiem zmianie.
- **Create Dynamic Partitions** - Stworzenie partycji, pod którymi zapisywane będą dane na HDFSie. Obecnie jest są to partycje po dacie w formacie "yyyy-MM-dd"
- **Batch records to one parquet file** - Batchowanie rekordów do jednego pliku, tak aby nie zapisywać każdego rekordu w osobnym pliku
- **ConvertAvroToParquet** - Zamiana pliku w formacie Avro na Parqueta
- **Change Filename to Unique** - Nadanie unikalnej nazwy plikowi - atrybut UUID FlowFile
- **PutHDFS** - Zapisanie pliku na HDFSie w zdefiniowanych wcześniej partycjach w folderze



Rysunek 2: Część w NiFi odpowiadająca za pobieranie i przetwarzanie danych pogodowych



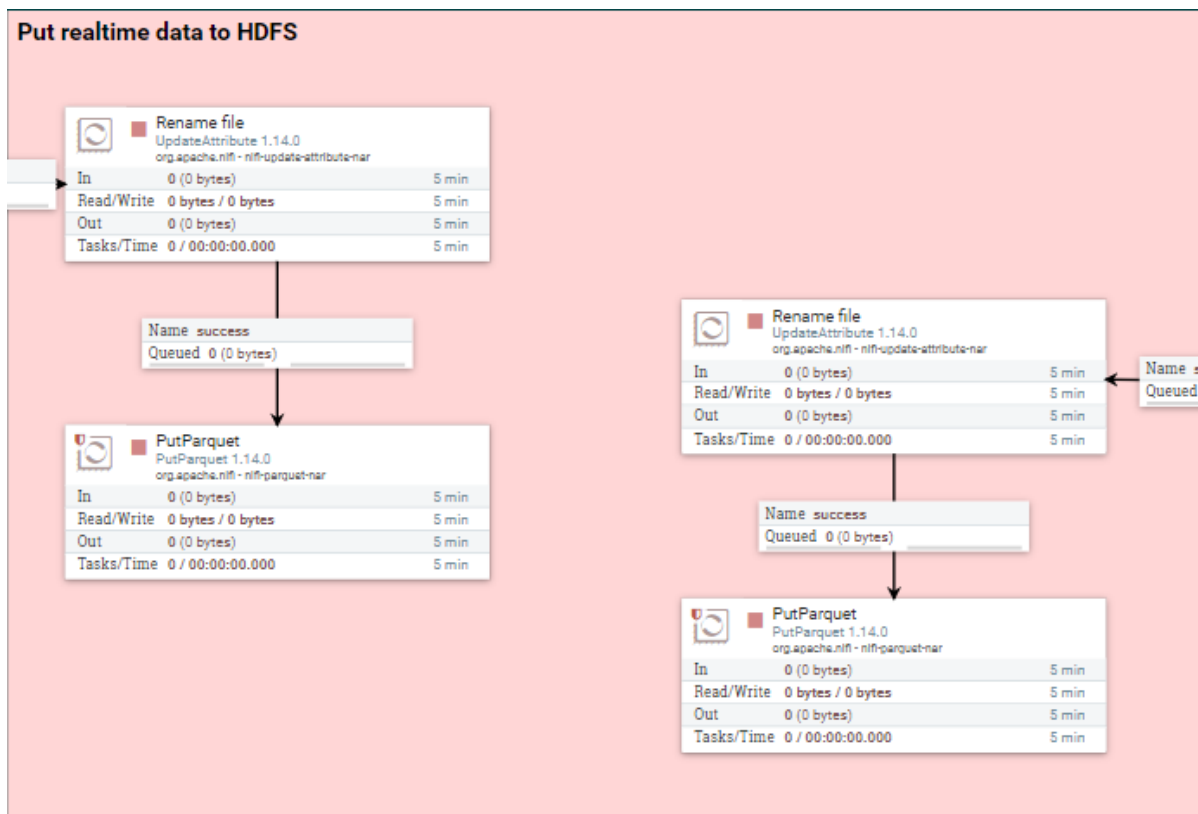
Rysunek 3: Część w NiFi odpowiadająca za pobieranie i przetwarzanie danych dot. zanieczyszczenia powietrza

3.5 Nadpisanie najnowszych danych dla każdego miasta

Aby przeprowadzić szybką analizę online najbardziej aktualnych danych, wszystkie nowo napływające rekordy dla każdego miasta, nadpisują ich poprzednie wersje w dedykowanych folderach:

Część NiFi odpowiadająca za te operacje znajduje się na rysunku 4. Składa się z następujących procesorów:

- **Rename file** - zmiana nazwa pliku zgodnie z formatem `<miasto>_<wysokość_geograficzna>_<szerokość_geograficzna>.parquet`
- **PutParquet** - Zapisanie plików w systemie HDFS zgodnie ze zdefiniowanym wcześniej *AvroSchema*



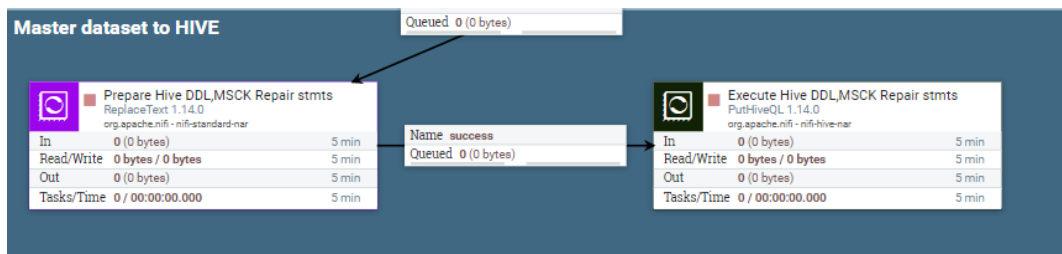
Rysunek 4: Część w NiFi odpowiadająca za nadpisywanie tylko bieżących pomiarów

3.6 Zapisywanie danych do HIVE

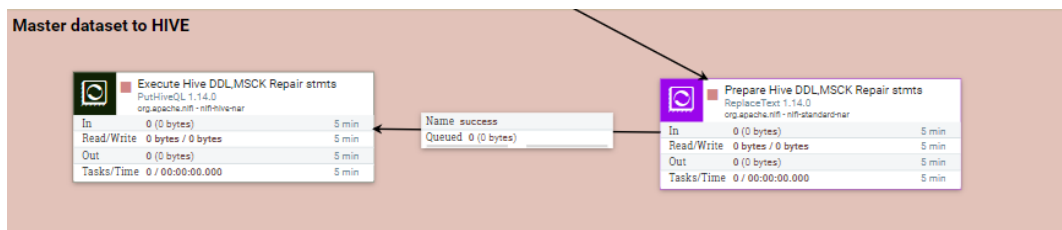
Zarówno dane dot. pogody, jak i te dot. zanieczyszczenia powietrza po zapisaniu na HDFSa są dodatkowo wrzucane do tabel hiveowych. Wynika to z ewentualnej potrzeby kierowania zapytań SQLowych bezpośrednio na surowych danych. W tym celu z poziomu NiFi jest tworzona tabela w Hive, osobna dla każdego z API. Procesory potrzebne do stworzenia tabel w Hive przedstawione są na rysunkach 5 oraz 6.

3.7 Analiza batchowa - batch layer

Ta warstwa służy do tworzenia widoków wygenerowanych na podstawie danych surowych zapisanych w postaci tzw. *master tables*. Wygenerowane w ten sposób widoki są następnie odkładane historycznie w HBase. Widoki są tworzone 4 razy dziennie w odstępach 6-godzinnych. Do stworzenia widoków używane są wszystkie dane, które dotychczas zostały odłożone w danych raw. To znaczy, że widok wygenerowany i zapisany w



Rysunek 5: Część w NiFi odpowiadająca za wrzucanie tabel dot. pogody z HDFSa do Hive'a



Rysunek 6: Część w NiFi odpowiadająca za wrzucanie tabel dot. zanieczyszczenia powietrza z HDFSa do Hive'a

HBase z id wiersza równym *2022-01-01-12* został stworzony na podstawie wszystkich danych, które był dostępny o godz. 12.00 pierwszego stycznia. Zapisywane są 3 typy widoków:

- **Zagregowane dane dot. pogody** - są to statystyki opisowe dla wszystkich kolumn numerycznych zawartych w danych pogodowych wyliczone i pogrupowane per miasto. W ten sposób otrzymujemy np. średnią z dotychczasowo odnotowanych temperatur w Londynie.
- **Zagregowane dane dot. zanieczyszczenia powietrza** - podobnie zagregowane dane co powyżej, natomiast dotyczą one danych o zanieczyszczeniu powietrza
- **Dane potrzebne do stworzenia mapy** - jest to połączenie danych o pogodzie i zanieczyszczeniu powietrza, które służy do wygenerowania wizualizacji (np. map), bądź pozwala na wyliczenie zależności pomiędzy pogodą, a zanieczyszczeniem (na uprzednio zagregowanych już danych)

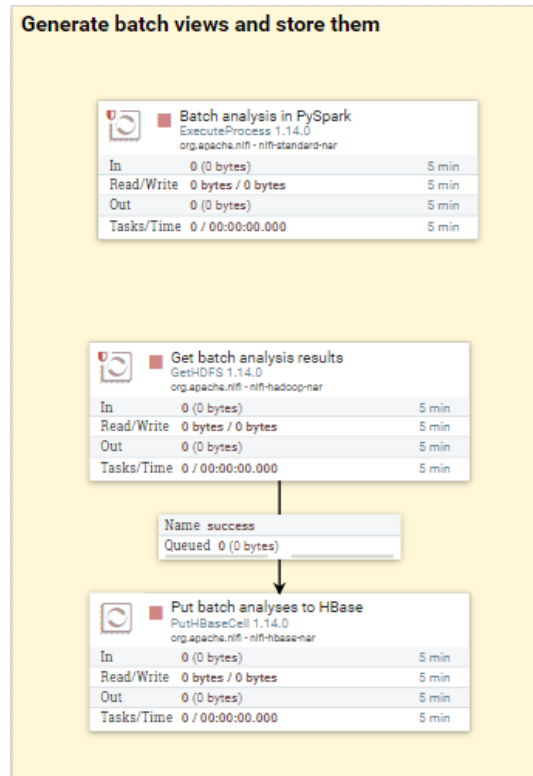
Tak stworzone analizy są zapisywane w formacie *JSON*, które są następnie wrzucane do HBase'a, gdzie identyfikatorem jest data zawierająca również godzinę, aby potem odróżnić widoki stworzone w różnych godzinach/dniach. Rodziną kolumn w tabeli znajdującej się w hbasie, jest kolumna *analysis*, natomiast każdy z widoków jest wrzucany z odpowiednią nazwą kolumny (inną dla każdego z w.w. widoków) w formacie JSONa. Schemat w NiFi przedstawiony jest na rysunku 7.

3.8 Analiza online - speed layer

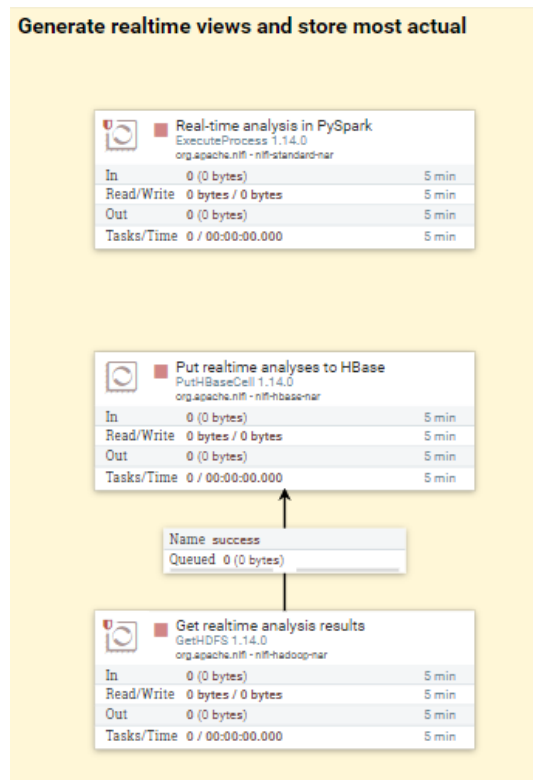
Warstwa szybkiej analizy służy tutaj do tworzenia interaktywnej mapy, zawierającej podstawowe pobrane statystyki dotyczące obsługiwanych miast. Najnowsza stworzona mapa zapisywana jest w formacie *html* w HBase(z nazwą kolumny *map*), zawsze nadpisując poprzednią mapę.

Część NiFi odpowiadająca za obsługę tej warstwy znajduje się na rysunku 8. Składa się z następujących procesorów:

- **Real-time analysis in PySpark** - włączenie skryptu PySpark, zbierającego aktualne dane z HDFS, przetwarzającego je w interaktywną mapę oraz zapis mapy na HDFS
- **Get realtime analysis results** - pobranie aktualnej analizy w postaci mapy z HDFS
- **Put realtime analysis to HBase** - nadpisanie aktualnej analizy w HBase



Rysunek 7: Część w NiFi odpowiadająca za warstwę batchową



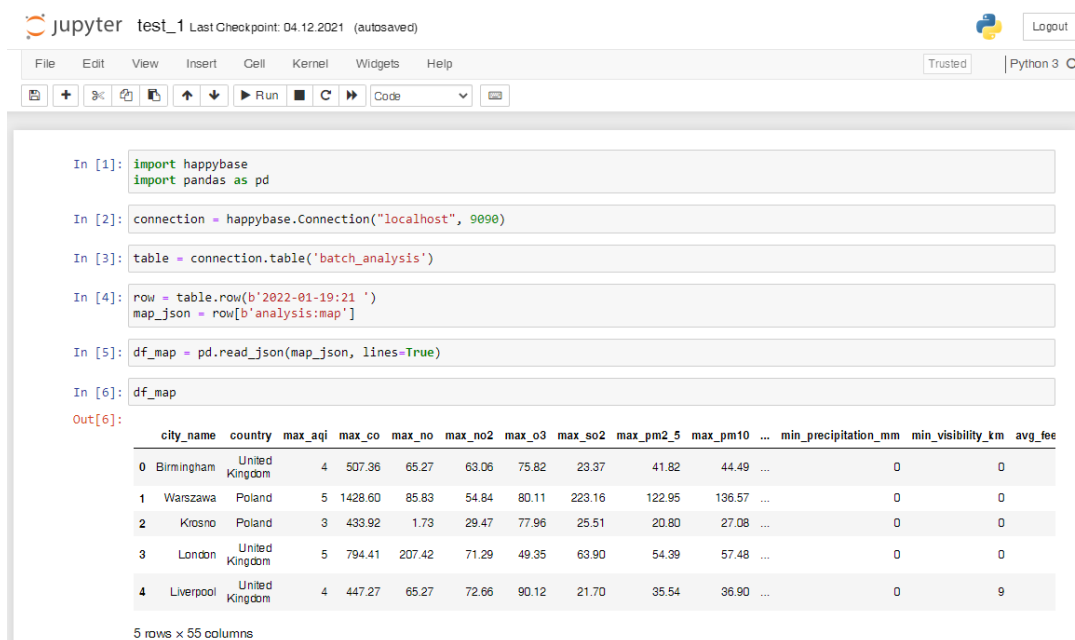
Rysunek 8: Część w NiFi odpowiadająca za szybką analizę danych w czasie rzeczywistym

4 Testy

4.1 Wyciągnięcie analiz batchowych z HBase

Cel	Sprawdzenie, czy stworzone widoki batchowe są odczytywalne z poziomu HBase'a za pomocą zewn. skryptu pythonowego z użyciem biblioteki Happy Base
Kroki	- zapisanie widoków batchowych w HBase za pomocą flow w NiFi - odczytanie widoków z HBase za pomocą biblioteki HappyBase
Oczekiwany wynik	Dane będzie się dało odczytać i stworzyć z nich wizualizację

Na rysunkach 9 oraz 10 widać sukces w odczytaniu danych z HBase'a.



```
In [1]: import happybase
import pandas as pd

In [2]: connection = happybase.Connection("localhost", 9090)

In [3]: table = connection.table('batch_analysis')

In [4]: row = table.row(b'2022-01-19:21 ')
map_json = row[b'analysis:map']

In [5]: df_map = pd.read_json(map_json, lines=True)

In [6]: df_map
```

Out[6]:

	city_name	country	max_aqi	max_co	max_no	max_no2	max_o3	max_so2	max_pm2_5	max_pm10	...	min_precipitation_mm	min_visibility_km	avg_fee
0	Birmingham	United Kingdom	4	507.36	65.27	63.06	75.82	23.37	41.82	44.49	...	0	0	
1	Warszawa	Poland	5	1428.60	85.83	54.84	80.11	223.16	122.95	136.57	...	0	0	
2	Krosno	Poland	3	433.92	1.73	29.47	77.96	25.51	20.80	27.08	...	0	0	
3	London	United Kingdom	5	794.41	207.42	71.29	49.35	63.90	54.39	57.48	...	0	0	
4	Liverpool	United Kingdom	4	447.27	65.27	72.66	90.12	21.70	35.54	36.90	...	0	9	

5 rows x 55 columns

Rysunek 9: Część pierwsza rezultatów testu I - ściągnięcie jasona i zamienienie na ramkę danych

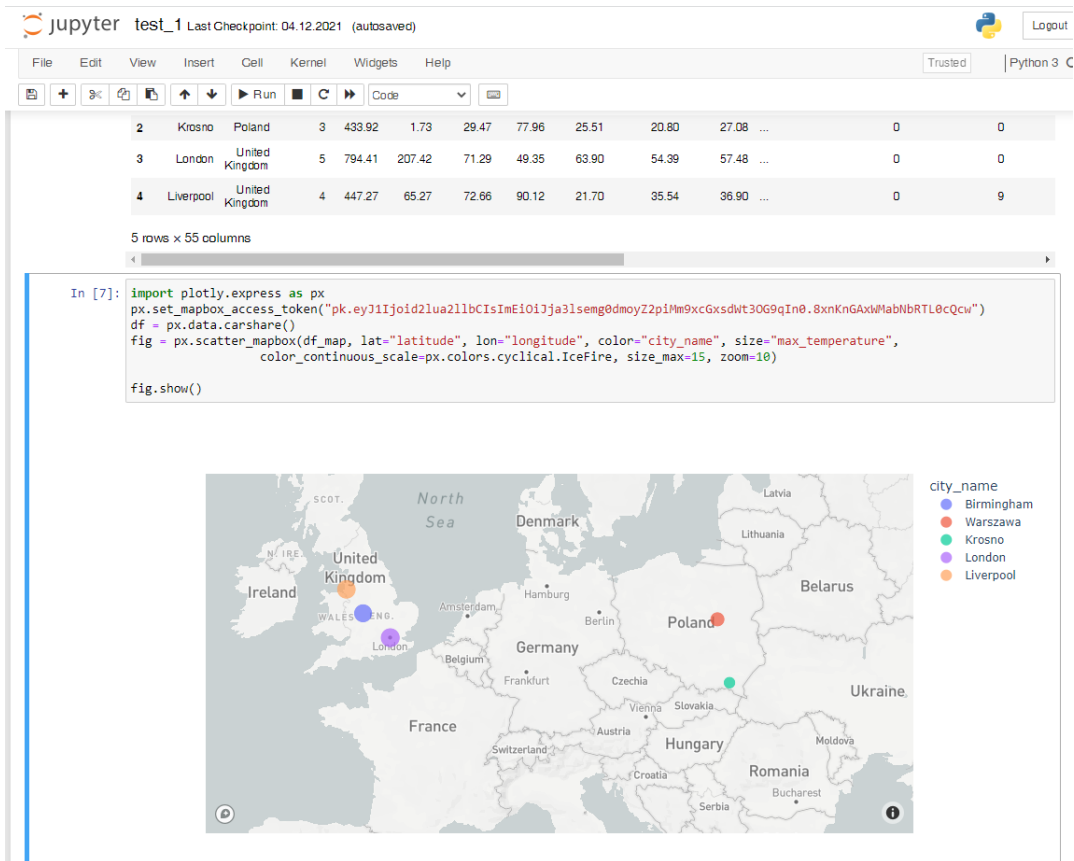
4.2 Wyciągnięcie analiz realtime z HBase

Cel	Sprawdzenie, czy widok realtime zapisywany w postaci pliku .html da się odczytać z HBase i zachowuje swoją strukturę i zawartość
Kroki	- zapisanie widoku realtime w HBase za pomocą flow w NiFi - odczytanie widoku z HBase za pomocą biblioteki HappyBase
Oczekiwany wynik	Pobrany plik .html będzie się dało otworzyć w przeglądarce

Na rysunkach 11 oraz 12 widać sukces w odczytaniu danych z HBase'a.

5 Podsumowanie

Jesteśmy zadowoleni ze stworzonej w ramach projektu architektury, ponieważ jest stworzona zgodnie z dobrymi praktykami, a także zapewniona jest wysoka skalowalność, aby znacznie zwiększyć liczbę rekordów, aby faktycznie mieć do czynienia z Big Data. Wierzymy, że zaproponowane rozwiązanie spełnia wymogi architektury lambda i pozwala na ciekawe wizualizacje danych dot. pogody oraz zanieczyszczonego powietrza. Świetny to był projekt, nie zapomnimy go nigdy.



Rysunek 10: Część druga rezultatów testu I - wykorzystanie ramki do narysowania mapy

```

In [1]: import happybase
import pandas as pd

In [2]: connection = happybase.Connection("localhost", 9090)

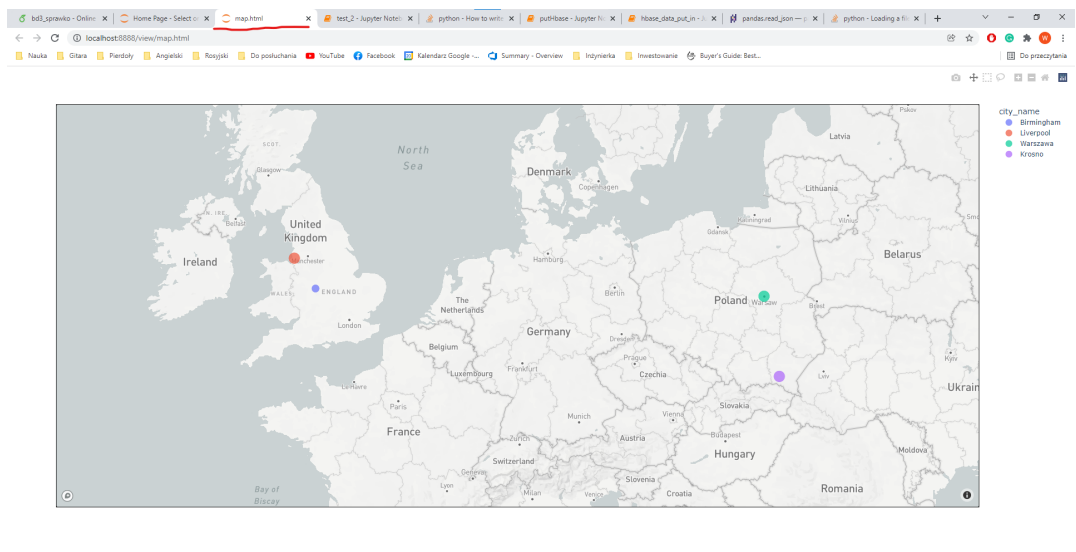
In [3]: table = connection.table('realtime_analysis')

In [4]: row = table.row(b'2022-01-19:23:12 ')
html_map = row[b'analysis:map']

In [5]: file= open("./map.html", "wb")
file.write(html_map)
file.close()

```

Rysunek 11: Część pierwsza rezultatów testu II - ściąganie htmla z HBase



Rysunek 12: Część druga rezultatów testu II - wyświetlenia htmla