



**WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI**
POLITECHNIKI RZESZOWSKIEJ

Marek Jachyra

Data pipeline w Apache Airflow

Projekt z przedmiotu Usługi Sieciowe w Biznesie

Opiekun pracy:

dr inż. Mariusz Borkowski, prof. PRz

Rzeszów, 2022

Spis treści

Wykaz symboli, oznaczeń i skrótów	5
1. Wstęp	6
2. Tekst zasadniczy – I	7
2.1. Opis projektu	7
3. Tekst zasadniczy – II	9
3.1. Opis realizacji projektu	9
3.1.1. Opis utworzenia środowiska pracy	9
3.1.2. Opis zaprogramowania przepływu danych	9
3.1.3. Rezultaty	13
4. Podsumowanie i wnioski końcowe	15
Załączniki	16
Literatura	17

Wykaz symboli, oznaczeń i skrótów

- API - interfejs programistyczny aplikacji (ang. application programming interface).
- DAG - nieskierowany graf acykliczny (ang. directed acyclic graph).

1. Wstęp

Przetwarzanie potokowe (ang. data pipeline) jest jednym ze sposobów sekwencyjnego przetwarzania danych. Przetwarzanie danych odbywa się w oddzielnych blokach. Po przetworzeniu w jednym, dane przekazywane są do następnego i tak aż do przetworzenia przez wszystkie bloki lub wystąpienia błędu, w którymś z bloków.

Celem pracy było stworzenie systemu przetwarzania potokowego automatyzującego pobieranie danych dotyczących ostatnio słuchanych przez użytkownika utworów. Dane najpierw pobierane są z API Spotify, następnie wyodrębniane są z nich istotne informacje, a na koniec zapisywane są w bazie danych.

2. Tekst zasadniczy – I

2.1. Opis projektu

Tematem projektu było utworzenie systemu przetwarzania potokowego (ang. data pipeline). Praca potokowa to cykl połączonych ze sobą operacji, w których wyniki poprzedniej przekazywane są do następnej. W przypadku wystąpienia błędu w jakiegokolwiek operacji, kolejne nie wykonują się [1].

Język Python udostępnia szeroki wachlarz bibliotek pozwalających na pobieranie, przetwarzanie, zapisywanie czy wizualizację danych. Jednak często istnieje konieczność automatyzacji wykonywania utworzonych programów.

Do najpopularniejszych bibliotek wykorzystywanych przez firmy do automatyzacji zadań przetwarzania danych należą obecnie Apache Airflow stworzony przez Airbnb i przekazany Apache Software Foundation oraz Luigi stworzony przez Spotify.

Apache Airflow jest platformą pozwalającą na tworzenie zadań i ich harmonogramów oraz ich monitorowanie. Schemat przepływu zadań tworzony jest jako skierowany graf acykliczny (DAG) opisujący zależności pomiędzy zadaniami. Środowisko udostępnia bogaty interfejs do monitorowania i wizualizacji istniejących cykli pracy. Głównymi założeniami Apache Airflow to dynamiczność, rozszerzalność, elegancja i skalowalność [2].

Do realizacji projektu wykorzystano narzędzia takie jak:

- 1) Python - język programowania wysokiego poziomu ogólnego przeznaczenia, w tym biblioteki:
 - glom - biblioteka do przetwarzania złożonych struktur danych.
 - requests - biblioteka do wykonywania żądań HTTP.
 - pymongo - biblioteka do pracy z MongoDB.
 - json - biblioteka do pracy z danymi JSON.
 - pendulum - biblioteka do zarządzania czasem i datami.
 - base64 - biblioteka do enkodowania danych binarnych.
- 2) Apache Airflow - otwarty system zarządzania cyklami pracy.

- 3) Docker - otwarte oprogramowanie do wirtualizacji na poziomie systemu operacyjnego (konteneryzacji).
- 4) MongoDB - otwarty, nierelacyjny system zarządzania bazą danych.

Środowisko pracy zostało utworzone jako kontener składający się z modułów obsługujących środowisko Apache Airflow, bazy Postgres wykorzystywanej przez to środowisko, bazy MongoDB, oraz interfejsów webowych zapewniających dostęp do wyżej wymienionych. DAG opisujący przepływ danych oraz funkcje implementujące działania wykorzystywane w zadaniach opisywanych przez graf zostały zaimplementowane w języku Python.

3. Tekst zasadniczy – II

Ponad 50% objętości pracy – część autorska:

- a) założenia – dane,
- b) opis zastosowanej metody rozwiązania lub analizy,
- c) opis proponowanego rozwiązania, wyniki analizy teoretycznej, obliczenia, projekt konstrukcyjny, procesowy, technologiczny,
- d) wyniki badań analitycznych, symulacyjnych lub eksperymentalnych itp.

Przy stosowaniu podziału na rozdziały i podrozdziały zaleca się unikać podziału więcej niż trzystopniowego. Podział tekstu, szczególnie na rozdziały główne, wynikać powinien z zakresu i charakterystyki realizowanej pracy.

3.1. Opis realizacji projektu

3.1.1. Opis utworzenia środowiska pracy

Środowisko pracy zostało utworzone jako kontener Docker. W tym celu zmodyfikowano plik `docker-compose.yaml`, w którym dodano nowe serwisy takie jak `pgadmin`, `mongo` i `mongo-express` oraz zmienne środowiskowe `CLIENT_ID`, `CLIENT_SECRET` i `REFRESH_TOKEN`. Wartości zmiennych zapisane są w pliku `.env`. Są one konieczne do poprawnego łączenia się z API. Utworzono również plik `Dockerfile`, w którym instalowane są dodatkowe biblioteki Python [2][4].

3.1.2. Opis zaprogramowania przepływu danych

Utworzono funkcję `refresh_token`, Listing 1, znajdującą się w pliku `spotify_helpers.py`, służącą do autoryzacji użytkownika i uzyskiwania tokenu dostępu [5]. Korzysta ona ze zmiennych środowiskowych zdefiniowanych wcześniej. Zmienne `client_id` oraz `client_secret` są w odpowiedni sposób enkodowane i przekazywane w nagłówku żądania POST. Funkcja zwraca token dostępu.

```
1 def refresh_token():
2     import base64
3     from airflow.models import Variable
4     import requests
5     uri = 'https://accounts.spotify.com/api/token'
6     refresh_token = Variable.get('REFRESH_TOKEN')
7     client_id = Variable.get('CLIENT_ID')
8     client_secret = Variable.get('CLIENT_SECRET')
```

```

9     encoded_string = base64.urlsafe_b64encode((client_id + ':' +
10    client_secret).encode())
11    response = requests.post(
12        uri,
13        data={
14            "grant_type": "refresh_token",
15            "refresh_token": refresh_token
16        },
17        headers={
18            "Authorization": "Basic " + encoded_string.decode()
19        }
20    )
21    return response.json()['access_token']

```

Listing 1: Listing funkcji uzyskującej token dostępu

Utworzono funkcję `get_request`, Listing 2, znajdującą się w pliku `spotify_helpers.py`, służącą do wykonywania zapytań do API. Jako parametry przyjmuje endpoint API oraz słownik parametrów. Zwraca odpowiedź z API.

```

1 def get_request(uri: str, params: dict):
2     """Get recently played songs from spotify API"""
3     import requests
4
5     response = requests.get(
6         uri,
7         headers={
8             'Accept': 'application/json',
9             'Content-Type': 'application/json',
10            'Authorization': f'Bearer {refresh_token()}',
11        },
12        params=params,
13    )
14
15    return response.json()

```

Listing 2: Listing funkcji wysyłającej zapytanie do API

Utworzono funkcję `get_database`, Listing 3, znajdującą się w pliku `mongo_helpers.py`, służącą do łączenia z bazą danych MongoDB.

```

1 def get_database():
2     from pymongo import MongoClient
3     import pymongo
4
5     CONNECTION_STRING = 'mongodb://root:example@mongo:27017'
6
7     from pymongo import MongoClient
8     client = MongoClient(CONNECTION_STRING)
9
10    return client['spotify']

```

Listing 3: Listing funkcji łączącej z bazą danych

Utworzono funkcję `save_data`, Listing 4, znajdującą się w pliku `mongo_helpers.py`, służącą do zapisywania danych do bazy. Jako parametry przyjmuje ona dane oraz nazwę kontenera, do którego mają zostać zapisane dane.

```
1 def save_data(data, container_name: str):
2     from pymongo import MongoClient
3     import pymongo
4
5     db = get_database()
6     container = db[container_name]
7     if type(data) == list:
8         container.insert_many(data)
9     elif type(data) == dict:
10        container.insert_one(data)
```

Listing 4: Listing funkcji zapisującej dane w bazie

W pliku `spotify_recent_songs_dag.py` zdefiniowano graf oraz odpowiednie zadania. Zadania służą do pobierania danych z API, transformacji pobranych danych do bardziej czytelnej postaci i zapisywania ich w bazie [2].

Przykładowy task pobierający ostatnio odtwarzane utwory z API, Listing 5. Zdefiniowany task pobiera dane z wykorzystaniem funkcji `get_request`, Listing 2, a następnie zwraca je w celu dalszego przetwarzania przez kolejne.

```
1 @task()
2 def get_recent_songs():
3     """Get recently played songs from spotify API"""
4     from helpers.spotify_helpers import get_request
5
6     response = get_request('https://api.spotify.com/v1/me/player/
7                             recently-played', {})
8
9     return response
```

Listing 5: Przykładowy task do pobierania danych z API

Przykładowy task wydobywający z otrzymanych danych utwory, Listing 6. Przy pomocy biblioteki `glom` dane formatowane są do czytelniejszej postaci i zwracane do dalszego przetwarzania.

```
1 @task()
2 def extract_songs(data: dict):
3     """Extract song data from API response"""
4     from glom import glom
5
6     spec = ('items', [{
7         'id': ('track', 'id'),
8         'name': ('track', 'name'),
9         'artists': ('track.artists', ['id']),
10        'duration': ('track', 'duration_ms'),
```

```

11         'popularity': ('track', 'popularity'),
12     })
13
14     songs = glom(data, spec)
15     return songs

```

Listing 6: Przykładowy task do wydobycia interesujących danych z odpowiedzi od API

Przykładowy task do wyznaczenia dziennych statystyk, Listing 7.

```

1 @task()
2 def compute_stats(songs: list):
3     """Compute daily song stats"""
4     from glom import glom
5     song_count = len(songs)
6
7     spec = ['duration']
8     durations = glom(songs, spec)
9     total_duration = sum(durations)
10
11     stats = {
12         'date': str(pendulum.now('Europe/Warsaw'))[:10],
13         'song_count': song_count,
14         'total_duration': total_duration,
15     }
16
17     return stats

```

Listing 7: Przykładowy task do wyznaczania statystyk z danych

Przykładowy task zapisujący dane w bazie, Listing 8. Przy pomocy funkcji `save_data`, Listing 4 dane zapisywane są w bazie

```

1 @task()
2 def save_songs(songs: list):
3     """Save song data into Mongo"""
4     from helpers.mongo_helpers import save_data
5     save_data(songs, 'recent_songs')

```

Listing 8: Przykładowy task zapisujący dane w bazie

Na Listingu 9 znajduje się część kodu opisująca przepływ danych w grafie.

```

1 data = get_recent_songs()
2
3 songs = extract_songs(data)
4 save_songs(songs)
5
6 albums = extract_albums(data)
7 save_albums(albums)
8
9 artists = get_artists_data(data)
10 artists = extract_artists(artists)
11 save_artists(artists)
12
13 genres = extract_genres(artists)

```

```

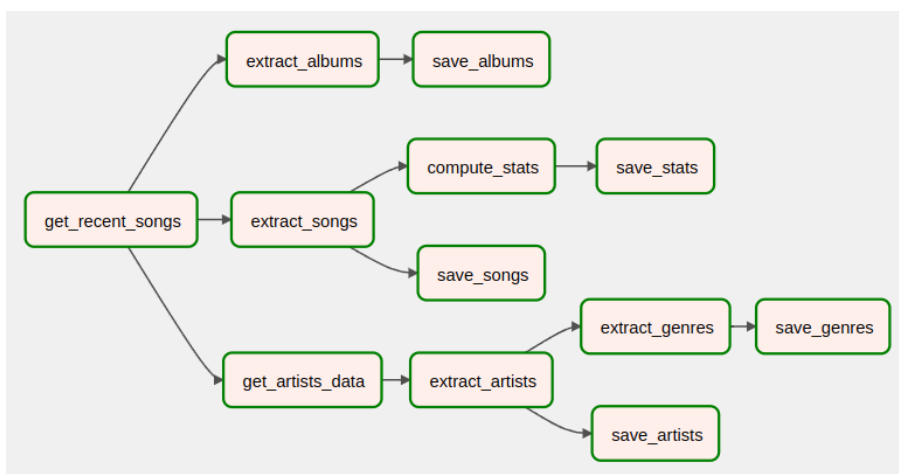
14     save_genres(genres)
15
16     stats = compute_stats(songs)
17     save_stats(stats)

```

Listing 9: Definicja przepływu danych

3.1.3. Rezultaty





Na Rysunku 3.1 znajduje się zdefiniowany w pliku `spotify_recent_song_dag.py` DAG. Cały pipeline zadziałał prawidłowo. Dane zostały pobrane z API, przetransformowane i zapisane w bazie. Na Rysunkach 3.2, 3.3, 3.4, 3.5, 3.6, znajdują się dane zapisane w bazie MongoDB po pojedynczym przepływie. Graf został zdefiniowany tak, aby wykonywał się raz dziennie. Dostęp do interfejsu Airflow znajduje się na adresie <http://localhost:8080/> zaś dostęp do bazy MongoDB na adresie <http://localhost:5000>.



Rysunek 3.1: Utworzony DAG

_id	id	name	artists	duration	popularity
629f8c572f24cf77da0ba274	3PuHwI5gUeNB9XKaCanLAv	Sorgen	5I4eibDkC7gg0hmczTZuND	184146	29
629f8c572f24cf77da0ba275	4MbfmcmXZ0mIWFY0L149A	The Night (Our Vinyl Sessions)	4eYE8Z6cfEHEdG22ITyucP4MhBe0d439ToDXCWT3wae	269664	40
629f8c572f24cf77da0ba276	7cGsNyhkoM55Lb3nOwPQlp	Honeymoon Phase / Wide Awake & Dreaming	4aT85lix0NSNB6w9Ozzksq44M8i4BCwuBbmcQWwMaOfH	396075	26
629f8c572f24cf77da0ba277	2lpfogHx3sIMFLsDgnifa1	Atlantic	2n2RSaZqBuUUukhBLlPnE6	293000	50






Rysunek 3.2: Utwory zapisane w MongoDB

_id	id	name	artists	release_date	total_tracks
 629f8c575e29c533d03a990d	4SWE8QpNwPf6CYylZhoiK	I solnedgången	5i4eibDkC7gg0hmczTZuND	2020-03-12	22
 629f8c575e29c533d03a990e	2wb1JmJvDusH0PsUlcR3o	The Night (Our Vinyl Sessions)	4eYE8Z6cEHedG22lTyucP,4MhBe0d439ToDXCWIT3wae	2020-05-14	1
 629f8c575e29c533d03a990f	6uMJVg0n7lQAT0X8XBLMEV	Honeymoon Phase / Wide Awake & Dreaming	4aT85lix0NSNB6w9Ozzksq,44M8i4BCwuBbmcQWwMaOfH	2020-11-20	3
 629f8c575e29c533d03a9910	53PFSATgN0YM7lslf2CDvs	This Place Will Become Your Tomb	2n2RSaZqBuUUukhblLpnE6	2021-09-24	12


Rysunek 3.3: Albumy zapisane w MongoDB

_id	id	name	genres	popularity
 629f8c57ad350f6665742dd5	5i4eibDkC7gg0hmczTZuND	Mando Diao	dalarna indie,swedish garage rock,swedish indie rock	54
 629f8c57ad350f6665742dd6	4eYE8Z6cEHedG22lTyucP	Morgan Wade	contemporary country	59
 629f8c57ad350f6665742dd7	4MhBe0d439ToDXCWIT3wae	Our Vinyl		57
 629f8c57ad350f6665742dd8	4aT85lix0NSNB6w9Ozzksq	OSKA	austrian indie	44

Rysunek 3.4: Artyści zapisane w MongoDB

_id	name
 629f8c593c6f5460eaffb0c9	dalarna indie,swedish garage rock,swedish indie rock
 629f8c593c6f5460eaffb0ca	contemporary country
 629f8c593c6f5460eaffb0cb	
 629f8c593c6f5460eaffb0cc	austrian indie
 629f8c593c6f5460eaffb0cd	indie anthem-folk,indie folk,indieacoustica

Rysunek 3.5: Gatunki zapisane w MongoDB

_id	date	song_count	total_duration
 629f8c58d2b380230f74828c	2022-06-07	20	5044426

Rysunek 3.6: Statystyki zapisane w MongoDB

4. Podsumowanie i wnioski końcowe

Całość kodu znajduje się na stronie podanej w załącznikach. W pliku RE-ADME.MD znajduje się instrukcja jego uruchomienia.

Apache Airflow jest narzędziem, który w prosty sposób pozwoliło na zautomatyzowanie pobierania i przetwarzania danych z API. W łatwy sposób pozwala na rozszerzenie grafu w przyszłości, poprzez dodawanie kolejnych zadań oraz tworzenie nowych grafów w oparciu o zapisane dane. Dzięki umieszczeniu aplikacji w kontenerze Docker, może zostać ona w łatwy sposób uruchomiona na innych maszynach czy też wyeksportowana do chmury aby zdefiniowane zadania wykonywały się codziennie.

Autor za własny wkład pracy uważa: zmodyfikowanie pliku `docker-compose.yaml` udostępnionego przez twórców Apache Airflow oraz zaprogramowanie nieskierowanego grafu acyklicznego opisującego przepływ danych i funkcji pomocniczych.

Załączniki

<https://github.com/m-jachyra/spotify-stats>

Literatura

- [1] [https://en.wikipedia.org/wiki/Pipeline_\(computing\)](https://en.wikipedia.org/wiki/Pipeline_(computing)). Dostęp 7.06.2022.
- [2] <https://airflow.apache.org/docs/>. Dostęp 7.06.2022.
- [3] <https://pymongo.readthedocs.io/en/stable/>. Dostęp 7.06.2022.
- [4] <https://docs.docker.com/>. Dostęp 7.06.2022.
- [5] <https://developer.spotify.com/>. Dostęp 7.06.2022.

STRESZCZENIE PRACY PROJEKTOWEJ

DATA PIPELINE W APACHE AIRFLOW

Autor: Marek Jachyra, nr albumu: FS-160747
Opiekun: dr inż. Mariusz Borkowski, prof. PRz
Słowa kluczowe: Airflow

Data pipeline do pobierania danych z API Spotify, przetwarzania ich i zapisywania w bazie z wykorzystaniem Apache Airflow.

PROJECT ABSTRACT

DATA PIPELINE IN APACHE AIRFLOW

Author: Marek Jachyra, nr albumu: FS-160747
Supervisor: PhD Mariusz Borkowski
Key words: Airflow

Data pipeline for fetching data from Spotify API, processing it, and saving in the database with Apache Airflow.