Politechnika Rzeszowska im. Ignacego Łukasiewicza
Wydział Matematyki i Fizyki Stosowanej

Piotr Krawiec 164165

Procesy ETL w Apache Airflow

Rzeszów 2022

Spis treści

1 Wstęp

W firmach gromadzi się coraz więcej danych. Dane te mogą znajdować się w różnych źródłach typu bazy danych, pliki tekstowe itp. Do gromadzenia i łączenia tych danych wykorzystuje się procesy ETL (Extract Transform Load). Praca ta pokazuje jak stworzyć system łączenia różnych źródeł danych, przekształcania ich oraz umieszczania w hurtowni danych (ETL) z pomocą Apache Airflow. Ponadto demonstruje, w jaki sposób wizualizować dane umieszczane w hurtowniach celem ich analizy.

2 Użyte oprogramowanie

2.1 Wirtualizacja

Wirtualizacja jest to narzędzie pozwalające na uruchamianie oprogramowania w odseparowanym środowisku. Narzędzia takie jak Docker pozwalają uruchamiać raz stworzone i skonteneryzowane oprogramowanie na wielu urządzeniach[1], natomiast Docker-compose pozwala definiować środowiska, w których działać może wiele skonteneryzowanych aplikacji[2]. Ułatwia to tworzenie architektury, gdyż:

1. Nie musimy instalować aplikacji, gdyż istnieją gotowe kontenery wymagające wyłącznie konfiguracji
2. Docker-compose umożliwia definicję całej architektury (wszystkich kontenerów) w jednym pliku
3. Uruchomienie aplikacji sprowadza się do jednego polecenia, niezależnie od komputera, na którym pracujemy. Ułatwia to pracę w grupie, gdyż każdy pracuje dokładnie z takim samym środowiskiem.

2.2 Wizualizacje

Jest to kluczowa część procesów Business Intelligence, to one pozwalają podejmować lepsze decyzje. Istnieje wiele gotowych narzędzi do tworzenia wizualizacji. Są to m.in.:

1. Tableau
2. Grafana

[1] `https://www.docker.com/`
[2] `https://docs.docker.com/compose/`

3. Apache Superset

i wiele innych. W wymienionych tutaj Grafana jest najmniej wymagająca oraz do działania wymaga wyłącznie jednego kontenera[3] [4]. Grafana pozwala na tworzenie interaktywnych tablic, które składają się z paneli, na których są wykresy. Wspiera ona wiele baz danych takich jak PostgreSQL czy mysql, a ponadto posiada system pluginów, który umożliwia zainstalowanie wtyczek, które pozwalają na połączenie się z niemal każdą bazą danych. System ten pozwala także na dodawanie nowych rodzajów wizualizacji.

2.3 Procesy ETL

W skrócie są to procesy pobierania, transformacji oraz umieszczania danych w bazie. Procesy ETL można realizować, korzystając z wielu narzędzi, nawet prosty skrypt w języku Python jest już w stanie takie procesy realizować. Co odróżnia taki skrypt od zaawansowanego oprogramowania to skalowalność oraz zarządzalność. Apache Airflow zapewnia obie te rzeczy. Składa on się z wielu serwisów, które odpowiadają za zadania takie jak:

- Zarządzanie i konfigurowanie procesów ETL
- Obserwowanie krok po kroku jak procesy te przebiegają
- Zlecanie wykonania określonych procesów ETl zgodnie z wyznaczonym harmonogramem
- Wykonywanie operacji ETL

W Apache Airflow proces ETL dzielony jest na małe zadania, które wykonywane są przez *operatory*. Zadania mają określoną kolejność wykonania, oraz zależności co do stanu wykonania innych zadań. Zbiór zadań odpowiadających za jeden proces ETL tworzy *DAG (Directed Acyclic Graph)*.

2.4 Bazy danych

Stanową podstawę całego systemu. W nich przechowywane będą dane potrzebne do działania każdego z serwisów oraz dane, które będą przetwarzane w procesach ETL. W projekcie wykorzystanych zostało kilka baz danych:

- MinIO — baza przechowująca obiekty. Tymi obiektami mogą być np. zbiory danych w postaci plików.

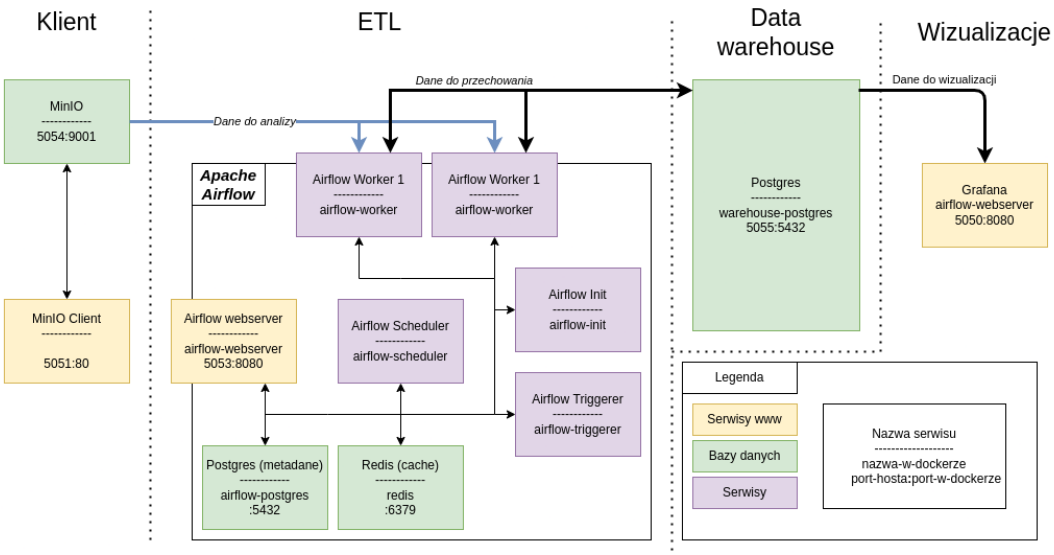
[3] Tableau wymaga licencji, aby działać w kontenerze

[4] Apache Superset potrzebuje 5 kontenerów, co zwiększa zapotrzebowanie na zasoby

- PostgreSQL — relacyjna baza danych, mogąca służyć jako hurtownia danych.
- Redis — nierelacyjna baza danych, przechowująca dane wyłącznie w pamięcy operacyjnej komputera. Może być używana jako cache lub niewielka baza danych. Wykorzystywana jest przez Apache Ariflow jako cache.

3 Opis architektury

Podstawą architektury jest Apache Airflow oraz Postgres (Data Warehouse). Dane, które będą obrabiane przez Airflow znajdują się w MinIO. Po przetworzeniu dane trafiają do Postgresa (Data Warehouse), skąd będą wizualizowane przez Grafanę.



Rysunek 1: Opis architektury

3.1 Uruchamianie

Cała architektura została zaimplementwana z pomocą Dockera i Docker-compose. Każdy jej element to pewien serwis w Docker-compose. Serwisy komunikują się ze sobą z pomocą nazw serwisów, co ułatwia przenoszenie jej między maszynami (nie jesteśmy zdani na przydzielone adresy IP kontenerów. Aby uruchomić serwisy należy:

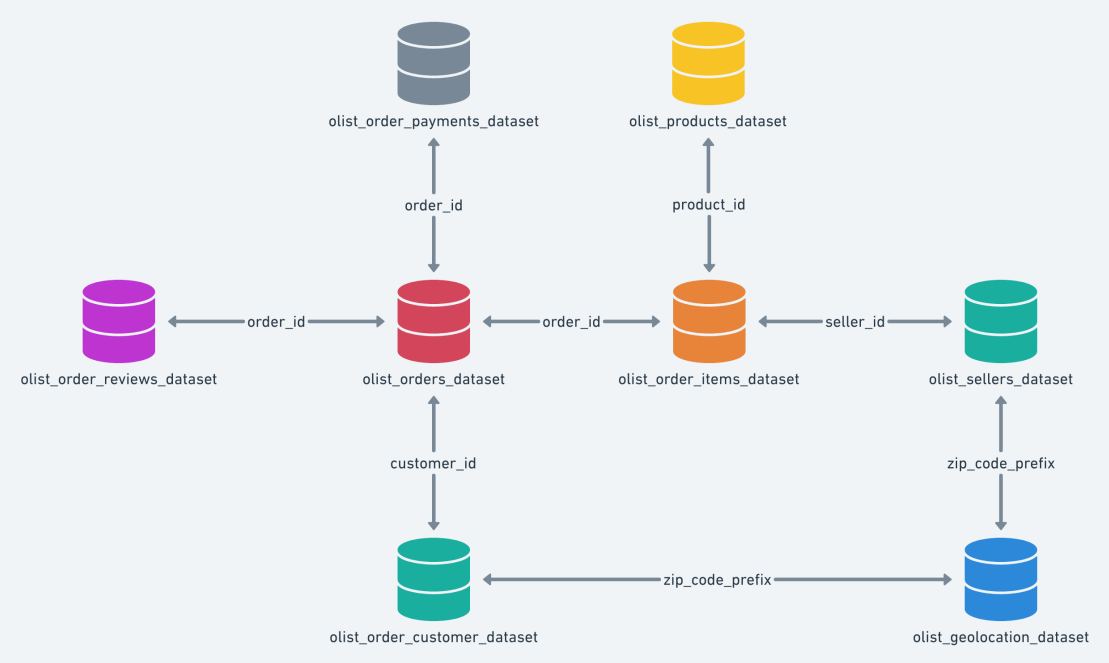
```
git clone https://github.com/finloop/grafana-airflow-etl
docker-compose up
```

Po uruchomieniu z poziomu przeglądarki uzyskami dostęp do kilku serwisów. Oto ich lista:

- MinIO Webserver `http://localhost:5051`. Domyślne dane dostępu: minioadmin:minioadmin
- ApacheAirflow Webserver `http://localhost:5053` Domyślne dane dostępu: airflow:airflow
- Grafana `http://localhost:5050` Domyślne dane dostępu: (brak)

4 Użyte dane

Dane użyte mają za symulować dane, z którymi można się spotkać często w biznesie. Są to dane dotyczące sklepu internetowego. Pochodzą one ze zbioru *Brazilian E-Commerce Public Dataset by Olist*. Zbiór ten obejmuje zanonimizowane dane pochodzące ze sklepu internetowego Olist z lat 2016-2018. Sklep Olist to sklep gdzie sprzedawcy tworzą własne sklepy, a klienci poprzez Olist store mogą w nich robić zakupy. Dane te zostały umieszczone w MinIO w "kubełku" *Datasets* w postaci



Rysunek 2: Schemat zbioru danych klienta

orginalnej tj plików csv. o niezmienionych nazwach.

Uzyskanie dostępu do danych odbywa się poprzez bibliotekę *boto3*, która współpracuje z dowolnymi Object Storage posiadającymi interfejsy kompatybilne z Amazon S3.

5 Procesy ETL w Apache Airflow

5.1 Operatory

Podstawową jednostką wykonującą zadania w Ariflow jest Operator. Operator powinien wykonywać jedno określone z góry zadanie, oczywiście to zadanie może zostać sparametryzowane, co sprawia, że operatory mogą być wykorzystane wielokrotnie [5]. Tych parametrów może być wiele, najczęściej występującymi parametrami są adresy oraz klucze API do serwisów. Operator po wykonaniu zadania może zwrócić wynik w kilku postaciach, najczęściej wykorzystywaną z nich jest XCOMS (cross-communications) [6]. XCOMS gwarantuje, że dane będą przekazywane między operatorami, pomimo tego, iż nie mamy gwarancji, że każdy operator uruchomi się na tej samej maszynie.

Airflow posiada bogatą bibliotekę gotowych operatorów pozwalających na pobieranie danych z niemal każdej bazy danych. Przykładem może być tu *PostgresOpetator*[7]. Operator ten wykonuje dowolne polecenie w SQL na bazie PostgreSQL. Ale oprócz gotowych operatorów można tworzyć własne, wykonujące dowolne zadania.

5.2 Extract

To wszystkie procesy pobierające dane z zewnętrznych źródeł. W projekcie stworzyłem taki operator **S3toDataFrameOperator**. Operator ten pobiera podany zestaw danych z MinIO, pobiera i zwraca w postaci *Pandas.DataFrame*, które są przetwarzane w kolejnych etapach.

Czasami, aby uruchomić pewne zadanie, musimy czekać na zdarzenie lub wystąpienie kilku warunków np. pojawienie się nowego pliku w folderze. W Airflow mechanizm ten realizowany jest z pomocą *Sensorów* [8]. Jednak w tym projekcie nie były wykorzystane.

[5] Operatory to jedynie podzbiór jednostek mogących tworzyć DAG, są jeszcze *sensory* i Taskflow Tasks

[6] `https://airflow.apache.org/docs/apache-airflow/stable/concepts/xcoms.html`

[7] `https://airflow.apache.org/docs/apache-airflow-providers-postgres/stable/operators/postgres_operator_howto_guide.html`

[8] `https://airflow.apache.org/docs/apache-airflow/stable/concepts/sensors.html`

```
import io
import boto3
import pandas as pd
s3 = boto3.client(
    "s3",
    endpoint_url=self.endpoint_url,
    aws_access_key_id=self.aws_access_key_id,
    aws_secret_access_key=self.aws_secret_access_key,
)
obj = s3.get_object(Bucket=self.bucket, Key=self.filename)
data = pd.read_csv(io.BytesIO(obj["Body"].read()))
return data
```

Listing 1: Fragment kodu **S3toDataFrameOperator** pobierającego dane z MinIO

5.3 Transform

Apache Airflow w raz z wejściem wersji 2.0 wprowadził TaskFlow API[9]. Rozszerzenie to pozwala na łatwiejsze tworzenie małych operatorów w Pythonie. Operatory te zostały przeze mnie wykorzystane do stworzenia procesów dokonujących transformacji na danych. Przykład takiego operatora znajduje się na Listing. 2.

Aby funkcja *transform* przetworzyła dane, musimy je pobrać. Dzięki parametryzacji Operatorów, pobranie danych i ich transformacja sprowadza się do przekazania wyników do kolejnej funkcji (patrz Listing. 3). Podobnie zostały stworzone pozostałe funkcje transformujące dane.

```
sellers = S3toDataFrame(
    filename="olist_sellers_dataset.csv",
    task_id="extract_olist_sellers_dataset",
)

transformed = transform(df_sellers=sellers.output)
```

Listing 3: Extract i transform

5.4 Load

To procesy zapisu przetworzonych danych. W Airflow realizuje się je z pomocą operatorów, podobnie jak procesy *Extract*. W projekcie stworzony został operator **Da-**

[9] https://airflow.apache.org/docs/apache-airflow/stable/concepts/taskflow.html

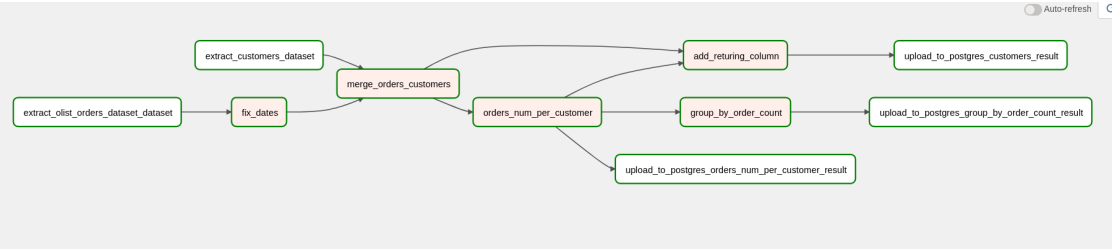
```python
from airflow.decorators import task

@task()
def transform(df_sellers):
    import pandas as pd
    sellers = df_sellers
    # Select 5 biggest states
    top_sellers = sellers.seller_state.value_counts()[:5]
    # Mark rest of them as 'other'
    sellers.loc[~sellers.seller_state.isin(top_sellers.index),
        "seller_state"] = "other"
    top_sellers = pd.DataFrame(sellers.seller_state.value_counts())
    top_sellers.columns = ["sellers_count"]

    return top_sellers
```

Listing 2: Funkcja transform wykorzystujące TaskFlow API

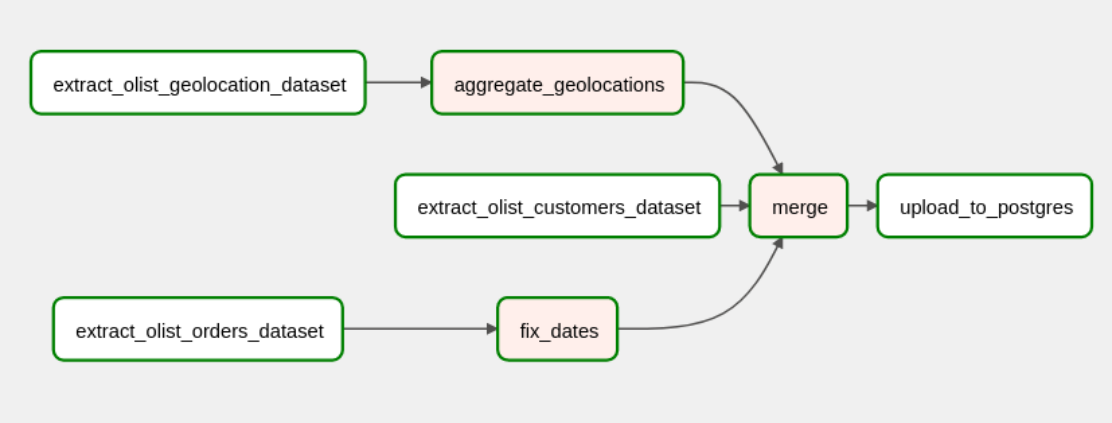
taFrametoPostgresOverrideOperator. Operator przyjmuje *pandas.DataFrame* i wysyła go do bazy danych (postgres), nadpisując przy tym tabelę, o ile już tam istniała.

5.5 DAG

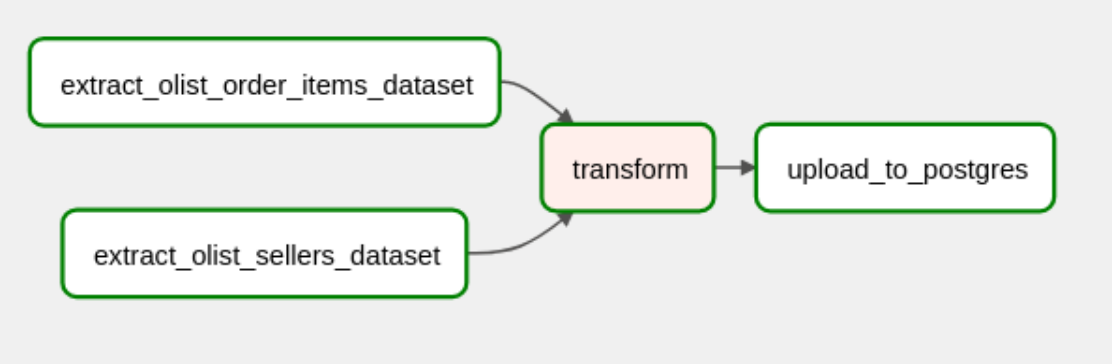
Z omówionych powyżej operatorów stworzone zostały DAGi tj. skierowane acykliczne grafy przedstawiające zależności pomiędzy zadaniami.



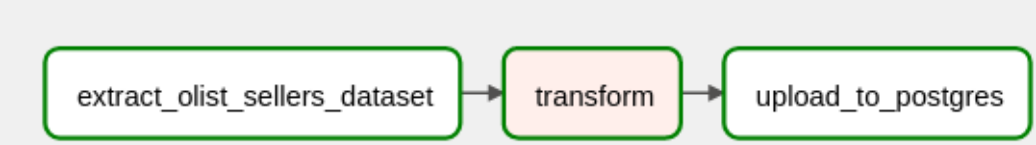
Rysunek 3: DAG Klienci



Rysunek 4: DAG Zamówienia i lokalizacje



Rysunek 5: DAG Sprzedający, zamówienia i przedmioty

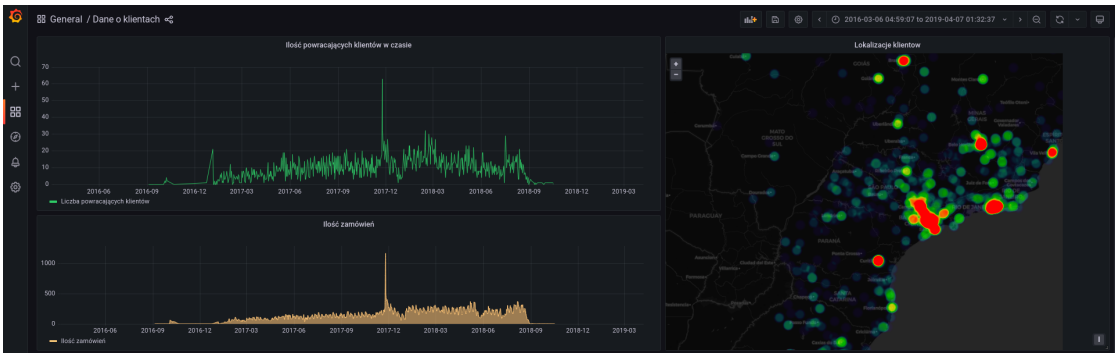


Rysunek 6: DAG Najwięksi sprzedawcy

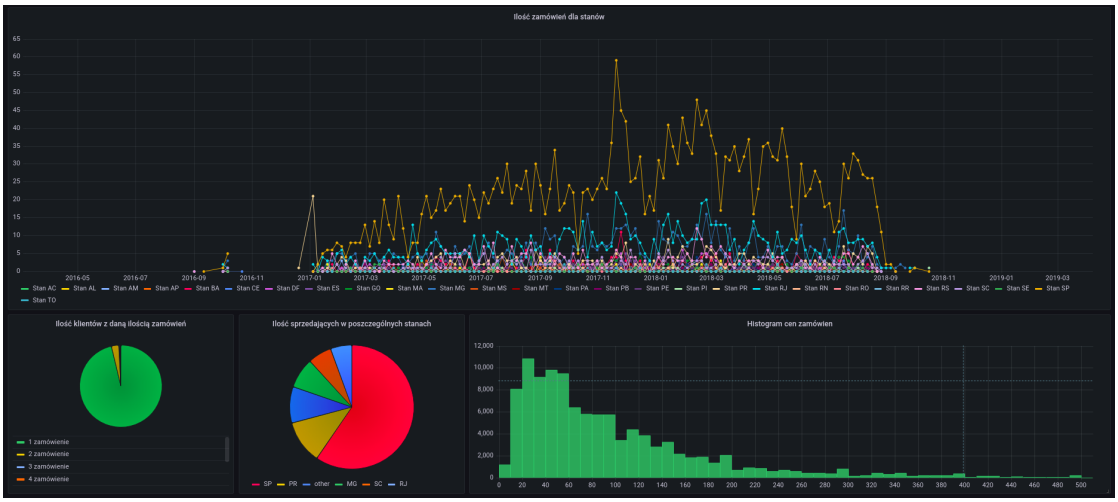
Zależności między zadaniami generowane są automatycznie przez Airflow.

6 Wizualizacje w Grafanie

Wszystkie te DAGi umieszczają swoje dane w tabelach, w data warehouse. Aby je wizualizować, połączyłem Grafanę z bazą danych i utworzyłem panel. Jego wygląd został pokazany na Rys. 7 i Rys. 8.



Rysunek 7: Panel część 1



Rysunek 8: Panel część 2

7 Podsumowanie

W projekcie pokazane zostało jak praktycznie budować procesy przetwarzające dane z wielu źródeł. Apache Airflow pozwala nie tylko tworzyć te procesy, ale także zarządzać nimi z poziomu przeglądarki, uruchamiać je w dowolnym interwale

z wieloma wyjątkami. Zaprezentowana architektura została skonteneryzowana, co pozwala na odtworzenie stworzonych analiz na dowolnym sprzęcie. A kod źródłowy rozwiązania umieszczony został w repozytorium na platformie GitHub [10].

[10] `https://github.com/finloop/grafana-airflow-etl`