Dokumentacja mechanizmu ETL Facebook dla DeepDoc

Wojciech Drężek

Spis treści

1. **Założenia ogólne1**
2. **Technologie i terminologia2**
3. **Zarys mechanizmu2**
4. Diagram2
5. Ogólny opis algorytmu mechanizmu3
6. **Dane użytkowników oraz projekt tabel w bazie danych3**
7. **Szczegółowy opis algorytmu mechanizmu7**

0. Upload danych użytkowników z Facebooka na S37

1. Wybór zdjęć do analizy7

2. Przetwarzania plików graficznych7

3. Uruchomienie serwera EC2 ETL Facebook7

4. Parsowanie i upload danych z plików JSON. Parsowanie i upload plików z wynikami analiz zdjęć 8

5. EC2UploadFBDataFromJSONs.py8

6. ECUploadImageProcessingData.py8

7. Uruchomienie serwera EC2 ETL DM9

8. Tworzenie data mart dla danych użytkownika i danych zdjęć9

9. EC2DataMartFBPicture.py9

10. EC2DataMartUser.py9

1. Założenia ogólne

Celem mechanizmu jest przekształcenie danych pochodzących z kont facebookowych do postaci tabelarycznej w relacyjnej bazie danych. Dane takie mają być gotowe do późniejszej analizy. Mechanizm zakłada przypływ danych sterowany przez zewnętrzny proces do folderu facebook w buckecie fbdeepdocdata. Każdy użytkownik ma swój folder, którego nazwa to jego id facebookowe. W folderze użytkownika znadują zdjęcia w formacie .jpg oraz pliki tekstowe w formacie .json zawierające dane użytkownika. Pojawienie się zdjęcia wywołuje odpowiednią funkcję Lambda, która dokonuje analizy owego zdjęcia, a wynik zapisuje do pliku tekstowego w tym samym folderze. Dane z plików .json oraz wyniki analizy zdjęć są zbierane przez procesy na serwerach EC2, a następnie ładowane do relacyjnej bazy danych. Serwery EC2 są uruchamiane okresowo poprzez funkcję Lambda.

1. Technologie i terminologia

Poniższe skróty będą oznaczać:

S3 – Amazon Simple Storage Service (Amazon S3)

Lambda – AWS Lambda

Baza danych – serwer MySQL Amazon Relational Database Service (Amazon RDS)

EC2 – Amazon Elastic Compute Cloud (EC2)

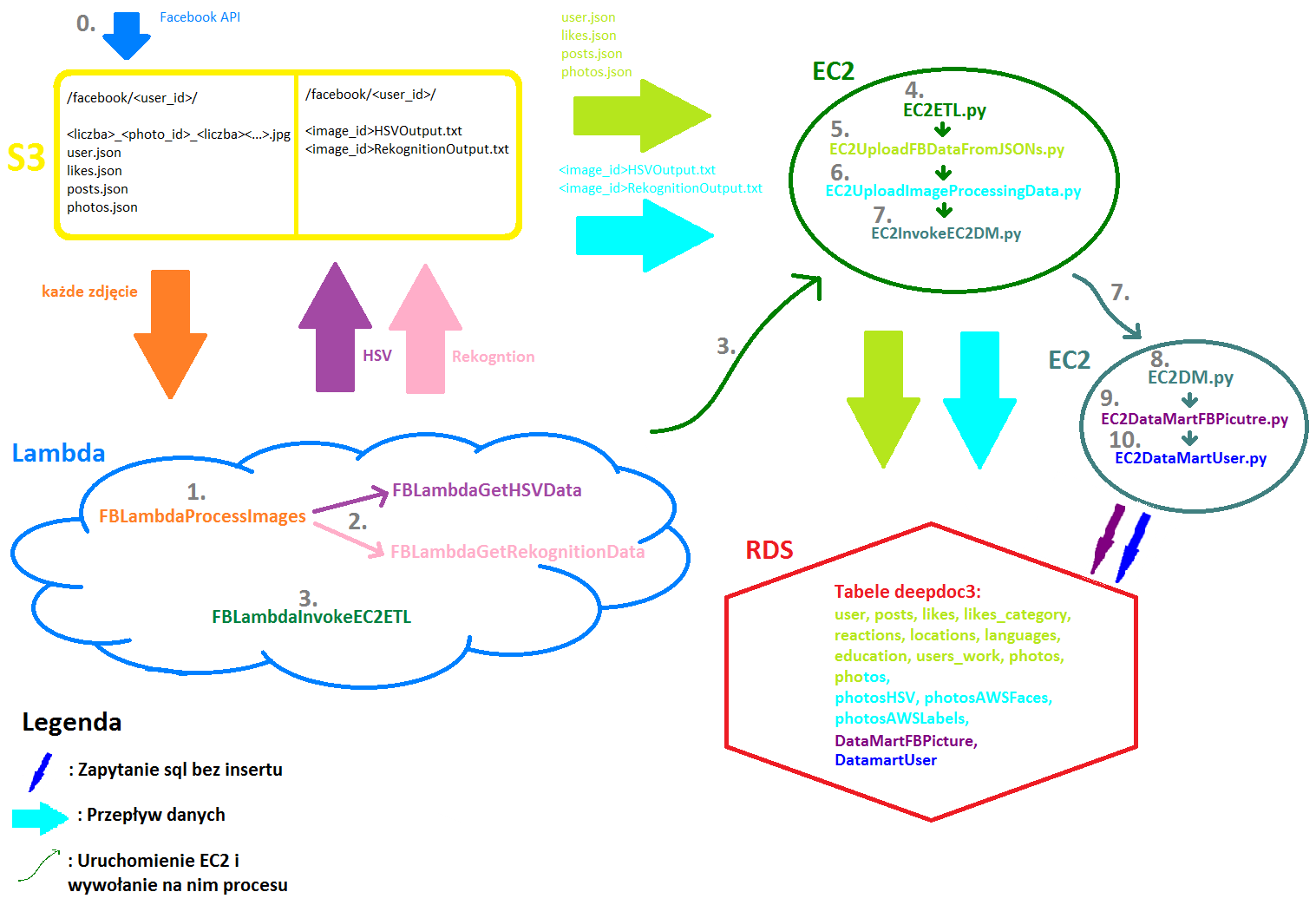
Rekognition – Amazon Rekognition

Użytkownik – użytkownik Facebooka, którego dane będą przetwarzane

Id użytkownika – id użytkownika z Facebooka

Wszystkie kody źródłowe są napisane w języku Python.

1. Zarys mechanizmu
2. Diagram (ctrl + scroll)



1. Ogólny algorytm mechanizmu:
2. Dane użytkowników Facebooka są ładowane na S3
3. Pojawienie się zdjęcia jest triggerem do funkcji Lambda FBLambdaProcessImages. Funkcja ta weryfikuje poprawność nazwy zdjęcia, a następnie przekazuje je do analizy HSV lub Rekognition.
4. Funkcje Lambda **FBLambdaGetHSVData** i **FBLambdaGetRekognitionData**, wywołane przez poprzedni proces dokonują analizy otrzymanego zdjęcia. Wynik analizy jest zapisywany w postaci gotowego zapytania SQL do pliku tekstowego.
5. Funkcja Lambda **FBLambdaInvokeEC2ETL** uruchamia zastopowany serwer EC2 oraz wywołuję na nim komendę uruchamiającą proces EC2ETL.py.
6. Proces **EC2ETL.py** uruchamia procesy w następującej kolejności **EC2UploadFBJSONs.py**, **EC2UploadImageProcessingData.py**, **EC2InvokeEC2DM.py**, po czym stopuje własny serwer.
7. Proces **EC2UploadFBJSONs.py** zbiera wszystkie dane użytkowników, a ich zawartość po odpowiedniej obróbce wysyła do bazy danych.
8. Proces **EC2UploadImageProcessingData.py** zbiera wszystkie pliki zawierające wynik analizy zdjęcia, a ich zawartość po odpowiedniej obróbce wysyła do bazy danych.
9. Proces **EC2InvokeEC2DM.py** uruchamia kolejny zastopowany serwer EC2, a na nim proces **EC2DM.py**
10. Proces **EC2DM.py** uruchamia procesy w następującej kolejności **EC2DataMartFBPicture.py**, **EC2DataMartUser.py**, po czym stopuje własny serwer.
11. Proces **EC2DataMartFBPicture.py** tworzy data mart dotyczący danych zdjęć w bazie danych poprzez zapytanie SQL.
12. **EC2DataMartUser.py** tworzy data mart dotyczący danych użytkownika w bazie danych poprzez zapytanie SQL.
13. Dane użytkowników oraz projekt tabel

Dane użytkowników występują w formacie plików graficznych oraz plików tekstowych w formacie JSON. Pliki graficzne są to zdjęcia użytkowników z rozszerzeniem .jpg.

Pliki z formatem JSON są podzielone na:

* **user.json**
* **likes.json**
* **posts.json**
* **photos.json**

Każdemu użytkownikowi są przydzielone cztery powyższe pliki oraz nieokreślona liczba zdjęć.

Dane z plików JSON po sparsowaniu będą umieszczone w bazie danych według następującego schematu:

**user.json** -> tabele: **user, photos, likes, education, languages, users\_work, locations**

**likes.json** - >tabele: **likes, likes\_category, reactions**

**posts.json** -> tabele: **posts, reactions, locations**

**photos.json** - > tabele: **photos, reactions**

Projekt tabel

(kolumna, opis)

1. Tabela **user**

|  |  |
| --- | --- |
| generationdate | data utworzenia wiersza |
| user\_id | id użytkownika |
| age\_range\_min | minimalny wiek użytkownika |
| age\_range\_max |  |
| birthday | data urodzenia |
| currency | waluta |
| devices | urządzenia, z których korzysta użytkownik np. Android |
| email |  |
| gender |  |
| interested\_in | ? |
| political | poglądy polityczne |
| quotes | cytaty użytkownika |
| relational\_status |  |
| significant\_other\_id | id drugiej połówki |
| significant\_other\_name | nazwa drugiej połówki |
| religion |  |
| is\_verified | ? |
| user\_name | nazwa użytkownika |
| user\_name\_format |  |
| secure\_browsing | ? |
| test\_group | ? liczba przydzielana użytkownikowi |
| thrid\_party\_id | third party id użytkownika |
| timezone | strefa czasowa |
| updated\_time | ? |
| user\_verified | ? |

1. Tabela **likes**

|  |  |
| --- | --- |
| generationdate |  |
| user\_id |  |
| like\_id |  |
| category | kategoria lajku |
| like\_name | nazwa lajku |
| like\_about | opis lajku |
| favorite | lajki w zakładce about w profilu na fb |

1. Tabela **likes\_category**

|  |  |
| --- | --- |
| generationdate |  |
| user\_id |  |
| like\_id |  |
| category\_id |  |
| category\_name | kategoria lajku |

1. Tabela **posts**

|  |  |
| --- | --- |
| generationdate |  |
| user\_id |  |
| post\_id |  |
| created\_time | data utworzenia postu |
| full\_picture\_source | link do zdjęcia w pełnej rozdzielczości |
| message | treść postu |
| picture\_source | link do zdjęcia |
| status\_type | typ postu, np. added photo, shared story |
| story | np. <user\_name> shared a link |
| description | opis do powyższej kolumny |
| post\_source | link jeżeli post zawiera link |
| privacy\_value | np. ALL FRIENDS, CUSTOM |
| from\_id | id autora postu |
| comments\_cnt | ilość komentarzy |
| likes\_cnt | ilość lajków |
| with\_tags\_cnt | ilość oznaczeń innych użytkowników |
| reactions\_cnt | ilość reakcji |

1. Tabela **photos**

|  |  |
| --- | --- |
| generationdate |  |
| user\_id |  |
| photo\_id |  |
| created\_time |  |
| backdate\_time | ? |
| image\_big\_height |  |
| image\_big\_width |  |
| image\_big\_source | link do zdjęcia |
| image\_small\_height |  |
| image\_small\_width |  |
| image\_small\_source | link do zdjęcia |
| image\_name | opis zdjęcia |
| picture | link do miniaturki zdjęcia |
| tags\_cnt | ilość tagów |
| comments\_cnt | ilość komentarzy |
| likes\_cnt | ilość likeów |
| from\_name | nazwa użytkownika, od którego pochodzi zdjęcie (jeśli dotyczy) |
| from\_id | id użytkownika --//-- |
| image\_type | other, cover lub profile |
| av\_HSV | zmienna pomocnicza oznaczająca czy została wykonana analiza HSV |
| av\_Labels | zmienna pomocnicza oznaczająca czy została wykonana analiza Rekognition Labels |
| av\_Faces | zmienna pomocnicza oznaczająca czy została wykonana analiza Rekogniton Faces |

1. Tabela **reactions**

|  |  |
| --- | --- |
| generationdate |  |
| user\_id |  |
| post\_id |  |
| photo\_id |  |
| reaction\_category | np post\_reaction, photo\_reaction |
| reaction\_id |  |
| from\_name | id autora |
| reaction\_type | rodzaj reakcji |
| created\_time |  |

1. Tabela **languages**

|  |  |
| --- | --- |
| generationdate |  |
| user\_id |  |
| language\_name |  |

1. Tabela **education**

|  |  |
| --- | --- |
| generation\_date |  |
| user\_id |  |
| school\_name |  |
| school\_type |  |
| degree\_name |  |
| degree\_type |  |
| concentration | kierunek studiów |

1. Tabela **locations**

|  |  |
| --- | --- |
| generationdate |  |
| user\_id |  |
| post\_id |  |
| category |  |
| city |  |
| country |  |
| created\_time |  |
| latitude | szerokość geograficzna |
| longitutde | długość geograficzna |

1. Tabela **users\_work**

|  |  |
| --- | --- |
| generation\_date |  |
| user\_id |  |
| description |  |
| employer\_name |  |
| position |  |
| location\_id |  |

Uwagi

Danych do tabel education i users\_work obecnie nie ma (30.08.2017)

1. Szczegółowy alogrytm mechanizmu
2. Upload danych użytkowników z Facebooka na S3

Przypływ danych jest obsługiwany przez zewnętrzny proces nienależący do mechanizmu ETL Facebook.

Dane dla jednego użytkownika czyli pliki JSON oraz zdjęcia mają się znajdować na S3 w buckecie ‘**fbdeepdocdata**’w folderach **facebook/<user\_id>/,** gdzie <user\_id> jest folderem o nazwie id użytkownika. To znaczy każdy użytkownik ma swój własny folder w folderze facebook.

Każde zdjęcie ma być ładowane w dwóch postaci; małej i dużej.

Nazwy małych zdjęć mają być oznaczane prefixem ‘small’.

Ogólny format nazwy zdjęcia:

<liczba 1>\_<**photo\_id** >\_<liczba 2>\_<nieistotny ciąg znaków>.jpg

lub

**small**<liczba 1>\_<**photo\_id** >\_<liczba 2>\_<ciąg znaków>.jpg

1. Wybór zdjęć do analizy

Pojawienie się jednego zdjęcia natychmiast wywołuję funkcję Lambda FBLambdaProcessImages. Zadaniem tej funkcji jest rozpoznanie czy nazwa zdjęcia jest poprawnego formatu, t.j. opisanego w poprzednim punkcie oraz przekazanie zdjęcia do analizy.

Weryfikacja poprawności formatu jest dokonywana przez sprawdzenie czy nazwa zdjęcia zawiera dwa slashe ‘/’, dwa podkreślniki ‘\_’ oraz czy <photo\_id> oraz <liczba\_2> są liczbami.

Warto tutaj zwrócić uwagę na to, iż nazwy plików w S3 zawierają w sobie pełną ścieżkę.

Przykład:

facebook/56942561789524652/1013553\_237760999763307\_7126724387170253782\_n.jpg-oh=bd6e50eb30d908ecbd3ea72e553b8c6b&oe=5A2FB9C2.jpg

Jeśli wyniki testu poprawności jest pozytywny, zdjęcia bez prefixu są przekazywane do funkcji Lambda **FBGetRekognitionData**, a zdjęcia z prefixem ‘**small**’ do funkcji Lambda **FBGetHSVData**.

1. Przetwarzanie plików graficznych

**FBGetRekognitionData**

Funkcja dostaje na wejściu obraz .jpg i wywołuje na nim funkcje usługi Rekognition, które zwracają etykiety opisujące to co znajduje się na obrazie oraz indeksy liczbowe dotyczące wykrytych twarzy. Powyższe wartości są przekształcane na zapytanie SQL, które jest zapisywane do pliku tekstowego o nazwie **<photo\_id>RekognitionOutput.txt** do folderu, w którym znajduje się dany obraz. Powyższe zapytanie po wykonaniu przez dalszy proces dokonuje wstawienia danych to tabel **photosAWSLabels** i **photosAWSFaces**. Zapytanie również aktualizuje wiersze danego obrazu kolumn **av\_Labels** i **av\_Faces** w tabeli photos.

**FBGetHSVData**

Funkcja dostaje na wejściu obraz .jpg oraz oblicza średnie R, G, B, Hue, Saturation i Value. Uzyskane wartości są przekształcane na zapytanie SQL, które jest zapisywane do pliku tekstowego o nazwie **<photo\_id>HSVOutput.txt** do folderu, w którym znajduje się dany obraz. Powyższe zapytanie po wykonaniu przez dalszy proces dokonuje wstawienia danych to tabeli **photosHSV**. Zapytanie również aktualizuje wiersze danego obrazu kolumny **av\_HSV.**

Uwagi

Bardziej optymalnym rozwiązaniem jest zapisywanie samych wartości do pliku tekstowego. Wówczas dalszy proces zbierający pliki wynikowe wykonywałby jeden insert ze wszystkich zebranych wartości do odpowiedniej tabeli. Wykonywanie insertów z danymi pojedynczych zdjęć do różnych tabel naprzemiennie jest bardzo nieefektywne. Dalszy proces **EC2UploadImageProcessingData** koryguję tę niedoskonałość poprzez ekstrakcję wartości z zapytań wygenerowanych przez powyższe funkcje.

1. Uruchomienie serwera **EC2 ETL Facebook**

Serwer EC2 o nazwie **EC2 ETL Facebook** jest postawiony na systemie operacyjnym Ubuntu Server 16.04 LTS. Typ instancji na chwilę obecną to c4.8xlarge (36 CPU i 60GB ramu). Z uwagi na koszty serwer po skończonej pracy jest stopowany. Uruchamia go Lambda **FBLambdaInvokeEC2ETL**, która jest triggerowano okresowo(obecnie co tydzień). Lambda łączy się z serwerem za pośrednictwem protokołu SSH. Kluczem do połączenia jest plik ‘**ETLFacebook.pem**’ znajdujący się się w buckecie ‘**deepdockeys**’. Lambda do połączenia potrzebuje również instance-id serwera oraz user. Instance-id jest podawany w kodzie źródłowym funkcji.

Po uruchomieniu serwera, Lambda wywołuje skrypt **EC2ETL.py**, a po 20 sekundach kończy połączenie.

Serwer posiada domyślnego użytkownika ‘ubuntu’.

Na serwerze zainstalowany jest python3.6 oraz niezbędne biblioteki do wykonywania skryptów. Skrypty znajdują się w folderze /home/ubuntu/scripts/DD/.

Logi skryptów są zapisywane w folderze /home/ubuntu/logs/<nazwa\_skryptu>/.

1. Parsowanie i upload danych z plików JSON. parsowanie i upload plików z wynikami analiz zdjęć.

Proces **EC2ETL.py** uruchamia kolejno procesy **EC2UploadFBDataFromJSONs.py**, **EC2UploadImageProcessingData**, **EC2InvokeEC2DM**.

Procesy te muszą być wykonane w takiej kolejności, gdyż zakładają dostępność danych z poprzednich procesów. Czyli proces **EC2ETL.py** czeka aż każdy z powyższych procesów skończy swoją pracę, po czym stopuje własny serwer.

1. **EC2UploadFBDataFromJSONs.py**

Celem powyższego procesu jest pobranie danych z plików JSON oraz wysłanie ich do bazy danych. Proces wykorzystuje pomocniczy skrypt **getFBDataFromJSONs.py**, ten z koleji skrypty **getFBUserFromJSON.py, getFBPhotosFromJSON.py, getFBPostsFromJSON.py, getFBLikesFromJSON.py**.

Proces przeszkuje S3 bucket ‘fbdeepdocdata’ pod kątem plików .json. Następnie dzieli je na listy nazw plików user.json, likes.json, photos.json, posts.json. Listy te są przekazywane do funkcji **getFBDataFromJSONs** ze skryptu **getFBDataFromJSONs.py**, która z kolei pojedyńczo pobiera pliki oraz przekazuje je do sparsowania przez odpowiednie funkcje ze skryptów: **getFBUserFromJSON.py, getFBPhotosFromJSON.py, getFBPostsFromJSON.py, getFBLikesFromJSON.py,** po czym łączy wynik pracy tych skryptów w tabele. Tabele te są zwracane do skryptu **EC2UploadFBDataFromJSONs.py,** a następnie przekształcane w zapytania SQL i wysyłane do bazy danych.

Proces **EC2UploadFBDataFromJSONs.py** w rzeczywistości w celu przyspieszenia działania dzieli się na 36 procesów, z których każdy z nich dostaje fragment listy plików do przetworzenia. Kolejkowanie zapytań SQL odbywa się na pozimie skryptu. Ilość procesów można zwiększyć lub zmniejszyć. Zależy to oczywiście od liczby procesorów, które posiada serwer.

1. **EC2UploadImageProcessingData.py**

Celem tego procesu jest zebranie plików wynikowych analizy Rekognition i HSV oraz wysłanie ich zawartości do bazy danych.

Jak wiadomo wynikiem przetwarzania plików graficznych są pliki **<photo\_id>RekognitionOutput.txt** oraz **<photo\_id>HSVOutput.txt**. Pliki te zawierają gotowe zapytania SQL, które wystarczy pobrać i wykonać. Okzało się to jednak bardzo nieoptymalne, gdyż wykonywanie na zmianę ‘insertów’ do różnych tabel jest nieefektywne. Zatem proces zbiera dane ze wszystkich wygenerowanych zapytań znajdującychg się w powyższych plikach oraz sam tworzy tylko trzy ‘inserty’ do tabel **photosHSV**, **photosAWSLabels**, **photosAWSFaces**.

Proces ten również dzieli swoją pracę na wiele równoległych procesów. Zasada działania jest identyczna jak w procesie poprzedzającym.

1. Uruchomienie serwera **EC2 ETL DM**

Serwer EC2 o nazwie **EC2 ETL DM** jest postawiony na systemie operacyjnym Ubuntu Server 16.04 LTS. Typ instancji na chwilę obecną to t2.micro (1 CPU i 1GB ramu). Serwer jest uruchamiany przez proces **EC2ETL.py** na serwerze **EC2 ETL Facebook**. Po skończonej pracy zostaje zastopowany.

Serwer posiada domyślnego użytkownika ‘ubuntu’.

Na serwerze zainstalowany jest python3.6 oraz niezbędne biblioteki do wykonywania skryptów. Skrypty znajdują się w folderze /home/ubuntu/scripts/DD/.

Logi skryptów są zapisywane w folderze /home/ubuntu/logs/<nazwa\_skryptu>/.

Po uruchomieniu serwera proces **EC2ETL.py** wywołuje skrypt **EC2DM.py**, po czym zrywa połączenie i kończy swoją działalność, stopując tym samym serwer **EC2 ETL Facebook**.

1. **Tworzenie data mart dla danych użytkownika i danych z zdjęć**

Proces **EC2DM.py** uruchamia kolejno procesy **EC2DataMartFBPicture.py**, **EC2DataMartUser.py**.

Procesy te muszą być wykonane w takiej kolejności, gdyż zakładają dostępność danych z poprzednich procesów. To znaczy proces **EC2DM.py** czeka aż każdy z powyższych procesów skończy swoją pracę, po czym stopuje własny serwer.

1. **EC2DataMartFBPicture.py**

Celem procesu jest stworzenie data mart, dotyczącego danych związanych ze zdjęciami. Proces ten wywołuje gotowe zapytanie SQL, które znajduje się kodzie źrodłowym skryptu. Proces czeka aż przetwarzanie zapytania przez bazę danych zostanie zakończone.

1. **EC2DataMartUser.py**

Celem procesu jest stworzenie data mart, dotyczącego danych związanych z użytkownikiem. Proces ten wywołuje gotowe zapytanie SQL, które znajduje się kodzie źrodłowym skryptu. Proces czeka aż przetwarzanie zapytania przez bazę danych zostanie zakończone.

Uwagi

Bardziej eleganckim i wygodniejszym rozwiązaniem byłoby gdyby Lambdy wszelkie zmienne wejściowe typu bucker, folder, instance-id, keyname przyjmowały jako enviromental variables. Wówczas skróciłoby to ilość ingerencji w kody źródłowe.