Jakub Adamczyk, Natalia Organek

Batch processing w Sparku

Cel projektu

Zasadniczy cel to przepisanie części batch processing istniejącego systemu o architekturze lambda, na skalowalne rozwiązanie oparte o Apache Spark.

Obecny system jest zaimplementowany jako sekwencyjne przetwarzanie w języku Ruby. Najpierw dane są pobierane z ElasticSearch'a działającego jako data sink. Każdy dokument jest przetwarzany, a następnie wrzucany do bazy typu key-value (Redis lub RocksDB), aby zapewnić unikalność według stworzonego klucza. Później dane te są wczytywane, przetwarzane do pliku CSV, a on ładowany jest bezpośrednio do bazy PostgreSQL.

Rozwiązanie to jest złożone i niewydajne do punktu nieużywalności.

Problemy koncepcyjne do rozwiązania

1. Efektywne przepisanie istniejących sekwencyjnych przepływów danych na architekturę równoległą

Dane mają często nietrywialny charakter czasowy, np. dla danej osoby (o konkretnym ID) muszą być przetwarzane sekwencyjnie, bo to eventy rozłożone na linii czasowej, natomiast dla wielu osób mogą być przetwarzane równolegle. Często należy też zapewnić filtrowanie po założonym kluczu, np. dla ofert promocyjnych interesuje nas tylko najnowsze użycie.

Przepisanie tego systemu do Sparka wymagać będzie przemyślanej implementacji, aby zapewnić efektywne przetwarzanie równoległe przy uwzględnieniu powyższych wymagań.

2. Ładowanie danych z ElasticSearch'a do Sparka

ElasticSearch wspiera odczyt batchowy, równoległy i ma plugin do Sparka. Chcemy sprawdzić, jak łatwo go wykorzystać i jak efektywne jest to rozwiązanie, w szczególności jaki wpływ mają batch size i poziom równoległości na wydajność.

3. Zapis danych ze Sparka do bazy PostgreSQL

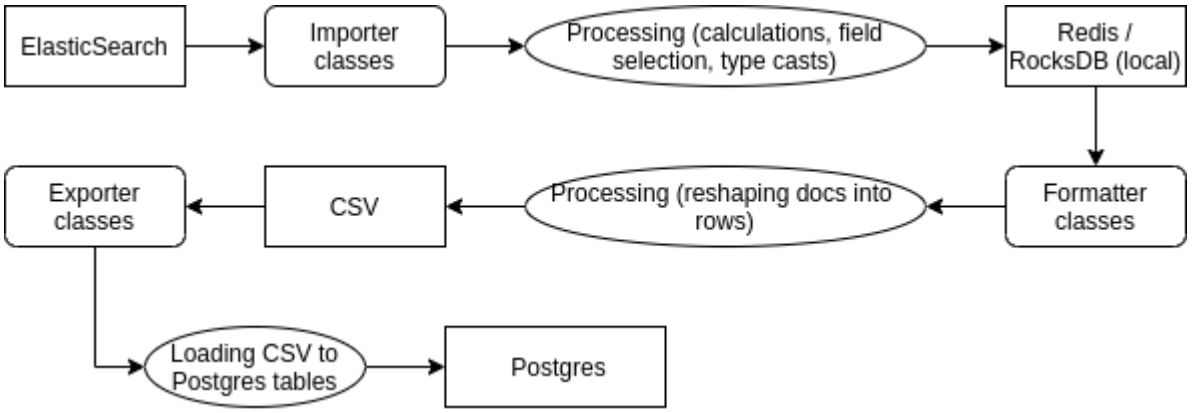
Postgres ma zapis zoptymalizowany pod ładowanie plików CSV (rozszerzenie ANSI SQL, klauzula `COPY FROM`), co jest jedną z przyczyn wykorzystania tego rozwiązania w obecnym systemie. Chcemy porównać, czy da się bezpośrednio pisać ze Sparka do bazy PostgreSQL, a jeżeli tak, to które rozwiązanie jest szybsze.

4. Konfiguracja architektury

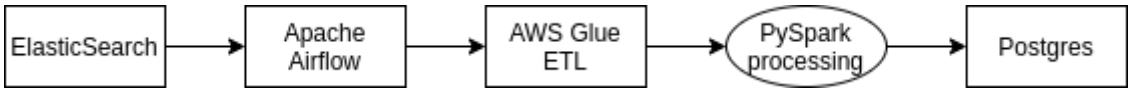
Batch processing ma być z założenia rzadko uruchamiany, więc konfiguracja i zarządzanie klastrem to dużo pracy. Chcemy sprawdzić, czy rozwiązanie serverless Spark, AWS Glue ETL, jest wydajne i czy może stanowić dobre rozwiązanie dla tego typu przetwarzania ad-hoc.

Schemat architektury

Schemat istniejącego rozwiązania:



Schemat docelowego rozwiązania opartego o PySpark:



Planowane testy

1. Infrastruktura

Na potrzeby developmentu: próbki danych testowych, lokalny system (wszystkie systemy w Docker Compose), ew. AWS EMR / Glue ETL i notebooki.

Testy skalowalności i deployment: istniejące instalacje Apache Airflow i ElasticSearch na klastrze Kubernetes, Postres w AWS RDS, AWS Glue ETL.

2. Zbiory danych

Dane będą pochodzić z danych testowych oraz produkcyjnych firmy Placewise. Dane zostały poddane anonimizacji i na całym etapie przetwarzania nie zawierają czułych danych osobowych. Są zawarte w wielu (ponad 20) indeksach ElasticSearch, gdzie każdy indeks jest mapowany na osobną tabelę w Postgresie.

3. Metryki

Dwie główne badane metryki to szczytowe oraz średnie zużycie pamięci oraz średnie zapotrzebowanie na CPU, gdyż one sterują najbardziej zapotrzebowaniem na wielkość klastra i typ instancji.

Metryki będą mierzone dla różnej liczby workerów i ich typów. Rozważany jest też pomiar wydajności na tradycyjnym klastrze AWS EMR i porównanie go z Glue ETL.

Szkic rozwiązania

Technologie i rozwiązania:

Metoda / narzędzie	**Zastosowanie**
Apache Spark + PySpark	Silnik obliczeniowy
Apache Airflow	Orkiestracja, GUI, logging
AWS Glue ETL	Serverless Spark
AWS S3	Zapis pliku CSV
ElasticSearch	Data sink
PostgreSQL	Docelowa baza danych
elasticsearch-hadoop (plugin do ES)	Przepływ danych z ES do Sparka
boto3	Komunikacja Pythona z AWS
Mozilla sops	Zabezpieczenie plików konfiguracyjnych
pipenv	Zarządzanie pakietami Pythona

Docelowe rozwiązanie ma być realizowane albo jako aplikacja Pythona uruchamiana dla wybranych indeksów z ElasticSearch'a, albo jako DAG w Apache Airflow z odpowiednią konfiguracją (wybór zostanie dokonany po dokładniejszej analizie).

Kroki docelowego rozwiązania:

1. Załadowanie konfiguracji AWS z zaszyfrowanych plików
2. Wykonanie skryptów dla wybranych indeksów z pomocą Glue ETL:
    a. Załadowanie danych z ElasticSearch'a
    b. Przetwarzanie danych w PySparku
    c. Zapis danych do bazy PostgreSQL

W eksperymentach ważnym krokiem będzie dodanie pomiędzy kroki 2b i 2c stworzenia pliku CSV (używając AWS S3 jako miejsca zapisu) w celu sprawdzenia, czy jest to szybsze niż bezpośredni zapis ze Sparka.