Cloud Engineer Intern

Zadanie Rekrutacyjne

Wstęp

Cześć!

Już tradycyjnie, jak co roku, mamy do rozwiązania poważny problem, tym razem dla przyszłych inżynierów Chmury.

W świecie tworzenia oprogramowania istnieje jedna rzecz, która cieszy się wyjątkowym uznaniem - wspólny lunch. Jednak z przyjemnością jedzenia często wiążą się kwestie finansowe, zwłaszcza jeśli chodzi o podział rachunku. W niektórych knajpach we Wrocławiu nie można podzielić płatności. Prowadzi to do sytuacji, w której jedna osoba płaci całość, a pozostali muszą się później z nią rozliczyć. W zgiełku transakcji łatwo o pomyłki w kwotach.

Rozwiązaniem tego problemu ma być `DebtSimplifier!`

DebtSimplifier

`DebtSimplifier` to usługa, która ułatwia proces rozliczania długów. Jej celem jest przyjęcie listy transakcji i wygenerowanie listy przelewów do wykonania, aby każda strona zakończyła rozliczenie na zero, oraz aby liczba przelewów była możliwie jak najmniejsza.

Zadania

Zadanie 1

W tej części twoim zadaniem będzie napisanie serca `DebtSimplifier'a` czyli samego algorytmu optymalizacyjnego. Dążymy do tego żeby algorytm zwrócił możliwie jak najmniejszą liczbę przelewów.

Projekt

W celu wykonania tego zadania przygotowaliśmy dla Ciebie:

- Katalog `/part_1` z:
  - plikiem `main.py` w którym powinieneś zaimplementować zadanie
  - plikiem `requirements.txt` w którym w razie potrzeby możesz zdefiniować dodatkowe zależności
- Katalog `/test_data` z plikami testowymi oraz przykładowymi rezultatami

Załączone dane testowe są bardzo proste i pozwolą na zweryfikowanie algorytmu w ograniczonym zakresie. Pamiętaj że system docelowo będzie działał w większej skali – więcej ludzi i transakcji.

Ważne: Jeśli Twoje rozwiązanie nie daje wyników w pełni zgodnych z przykładowymi, to niekoniecznie jest nieprawidłowe!

Dane wejściowe

Do uruchomienia aplikacji wymagany będzie plik CSV z listą transakcji. Uruchamiając aplikację musimy mieć możliwość określenia bezwzględnej ścieżki pliku jako argumentu w wierszu poleceń.

Spójrzmy na przykładowy plik wejściowy (`debts.csv`)

```
Jacek,Dominik,10
Dominik,Jacek,5
Kasia,Dominik,5
Michał,Kamil,13
```

Oznaczenia kolumn, od lewej:

- wierzyciel - osoba, która dokonała transakcji
- dłużnik - osoba, za którą zapłacił wierzyciel
- kwota (liczba naturalna)

Zwróć uwagę że plik CSV nie zawiera headera!

Uruchomienie aplikacji i spodziewany wynik

Aplikację będziemy uruchamiać używając Pythona 3.12:

```
python /home/…/part_1/main.py /home/…/test_data/debts_1.csv
/home/…/part_1/output.csv
```

Wynik działania powinien zostać zapisany w osobnym pliku csv pod ścieżką podaną jako kolejny argument w wierszu poleceń.

Format wyniku:

```
Kamil,Michał,13
Dominik,Jacek,5
Dominik,Kasia,5
```

Oznaczenia kolumn, od lewej:

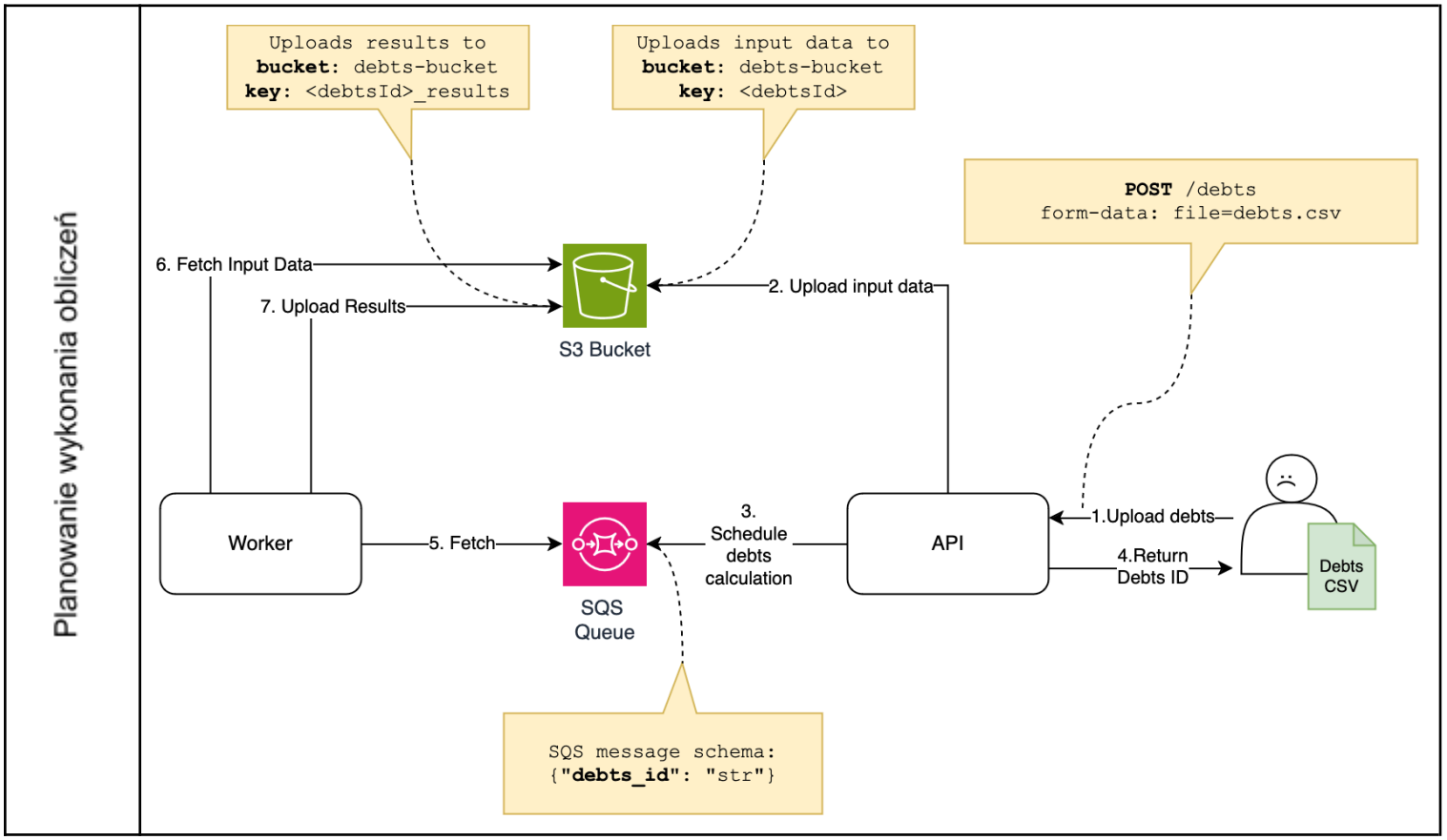
- dłużnik - osoba, która powinna wykonać przelew Osobie płacącej
- wierzyciel - adresat przelewu
- kwota

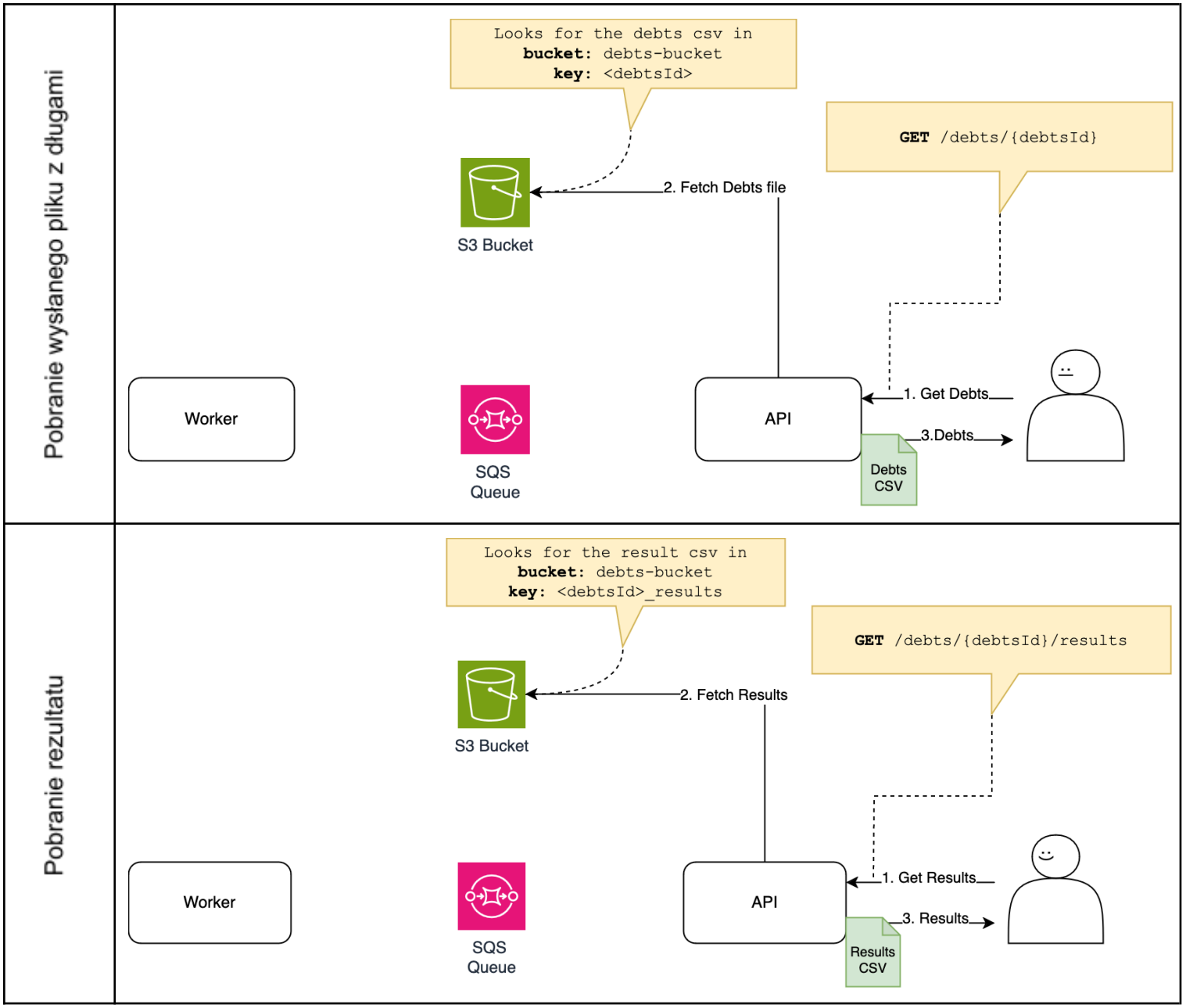
Zadanie 2

Czas zintegrować algorytm w prawdziwą usługę. Usługa będzie opierała się o cztery komponenty:

- **API** - web serwis przyjmujący requesty od użytkownika
- **Worker** - serwis obsługujący procesowanie algorytmu
- **SQS Queue** - kolejka zawierająca listę zadań do wykonania
- **S3 Bucket** - bucket przechowujący dane od użytkownika otrzymane od API oraz rezultaty wyliczone przez Worker'a

Poniższa tabela zawiera 3 procesy obsługiwane przez usługę `DebtSimplifier`:





Twoim zadaniem będzie zaimplementowanie Worker'a który:

1. **Odbierze dane z kolejki SQS**
2. **Zoptymalizuje przelewy**
3. **Zapisze rezultat w S3**

Projekt

Przygotowanie środowiska

Przed przystąpieniem do tego zadania będziesz potrzebował kilku dodatkowych narzędzi.

1. Docker - do skonteneryzowania i uruchomienia aplikacji
2. Docker Compose - do zarządzania uruchomieniem poszczególnych kontenerów
3. Poetry - do zarządzania zależnościami

Struktura projektu

Do wykonania zadania nie potrzebujesz konta AWS'owego. Zarówno S3 jak i SQS będą uruchomione lokalnie za pomocą LocalStacka.

W celu wykonania tego zadania przygotowaliśmy dla Ciebie:

- Katalog `/part_2` z:
  - Katalogiem `/api` - tutaj znajduje się kod źródłowy web serwisu odbierającego requesty od użytkownika
  - Katalogiem `/worker` - tutaj powinno znaleźć się twoje rozwiązanie
  - Plikiem `compose.yaml` zawierającym definicję serwisów uruchomionych w ramach usługi `DebtSimplifier`
- Katalog `/test_data` z plikami testowymi oraz przykładowymi rezultatami

Konfiguracja workera

Po uruchomieniu usługi za pomocą docker compose'a (patrz Uruchomienie aplikacji i spodziewany wynik) worker dostanie zestaw zmiennych środowiskowych zdefiniowanych w pliku `worker/config.env`.
Zmienne te będą potrzebne do połączenia się z zasobami AWS'owymi postawionymi za pomocą LocalStacka (S3 i SQS).

Dane wejściowe

Twoim zadaniem jest implementacja **Workera**, więc dane wejściowe należy sczytać z S3 spod klucza `<debtsId>`. Format danych jest taki sam jak w zadaniu pierwszym:

```
Jacek,Dominik,10
Dominik,Jacek,5
Kasia,Dominik,5
Michał,Kamil,13
```

Oznaczenia kolumn, od lewej:

- wierzyciel - osoba, która dokonała transakcji
- dłużnik - osoba, za którą zapłacił wierzyciel
- kwota (liczba naturalna)

Uruchomienie aplikacji i spodziewany wynik

Aby uruchomić aplikację z poziomu terminala, przejdź do katalogu `part_2` a następnie wystartuj całość za pomocą komendy:

```
docker-compose up --build
```

Na potrzeby testów możesz użyć dokumentacji API hostowanej pod adresem http://localhost:8000/docs#/ po uruchomieniu aplikacji.

Wynikiem optymalizacji wykonanej przez Workera powinien być plik csv zapisany w S3 pod kluczem `<debtsId>_results`. Format pliku powinien być analogiczny do rezultatu z zadania pierwszego tj.

Format wyniku:

```
Kamil,Michał,13
Dominik,Jacek,5
Dominik,Kasia,5
```

Oznaczenia kolumn, od lewej:

- dłużnik - osoba, która powinna wykonać przelew Osobie płacącej
- wierzyciel - adresat przelewu
- kwota (liczba naturalna)

Worker używa narzędzia do zarządzania zależnościami o nazwie poetry. Aby za pomocą poetry dodać bibliotekę do projektu należy skorzystać z komendy `poetry add`.

Nasze oczekiwania

- W odpowiedzi na maila z zadaniem spodziewamy się archiwum .zip z pełnym projektem, tj.
  - Katalog /`part_1` z rozwiązaniem zadania pierwszego
  - Katalog /`part_2` z rozwiązaniem zadania drugiego
  - Archiwum nie powinno zawierać pobranych lokalnie zależności (np. katalogu `.venv` i `.localstack`). Archiwum nie powinno być większe niż 1MB!
- Aplikacja powinna być napisana w Pythonie i być kompatybilna z Pythonem w wersji 3.12.
- Można korzystać z dowolnych bibliotek, należy jednak wiedzieć, co robią.
- To, na co zwracamy uwagę:
  - poprawność wyników
  - czytelność kodu
  - jakość projektu
  - chcielibyśmy aby kluczowe fragmenty kodu były pokryte testami
- Możesz uzyskać dodatkowe punkty za rozszerzenie pliku README wraz z kluczowymi informacjami. Spróbuj uzasadnić swoje decyzje projektowe.