Event stream processing

Łukasz Wasilewski, Szewczyk Bartłomiej, Patrycja Abramowska

POLITECHNIKA WARSZAWSKA OKNO PW

Projekt zespołowy, 2021/2022

Spis treści

[1) Wprowadzenie 2](#_Toc105224739)

[a) Cel projektu 2](#_Toc105224740)

[b) Podobne rozwiązania 2](#_Toc105224741)

[2) Założenia projektowe 2](#_Toc105224742)

[3) Podział ról i harmonogram prac 2](#_Toc105224743)

[4) Dyskusja sposobu rozwiązania problemu 3](#_Toc105224744)

[5) Prezentacja i opis wyników 4](#_Toc105224745)

# Wprowadzenie

## Cel projektu

Projekt realizuje system służący do strumieniowego przetwarzania zdarzeń generowanych przez aplikacje internetowe w celu prezentacji i wizualizacji danych statystycznych. Tworzona aplikacja udostępnia interfejs do którego aplikacja może wysyłać asynchroniczne lub synchroniczne żądania z opisem zdarzenia w ustalonym formacie. Program składa się z zestawu usług które współpracują ze sobą, żeby niemal w czasie rzeczywistym prezentować stan obserwowanej aplikacji. Przykładowe zastosowania uzyskanych danych to:

* Utrzymanie - monitoring – reagowanie na występowanie anomalii
* Rozwój - ulepszanie - analiza przeszłych awarii, namierzenie przyczyn i bottleneck
* Biznes – analiza trendów i zachowania klientów

## Podobne rozwiązania

Obecnie wykorzystywane jest klika rozwiązań oferujących podobne możliwości:

* Jako gotowy produkt można wykorzystać stack Elasticsearch którego główny komponent „Elasticsearch” jest w stanie samodzielnie lub z pomocą dodatkowych integracji oferowanych przez producenta pobierać i przeszukiwać dane ze strumienia a następnie wyświetlać je z wykorzystaniem samodzielnie przygotowanych narzędzi lub z pomocą komponentu „Kibana” oferowanego przez dostawcę.  
  Elasticsearch oferowany jako płatna usługa w wersji chmurowej lub jako open-source w przypadku instalacji na własnych maszynach
* Innym przykładem jest Power BI oferowany jako usługa w chmurze Microsoftu pozwalającą na przygotowanie dashboardów na podstawie danych otrzymywanych z różnych źródeł m.in. z plików, z chmurowej bazy danych Microsoftu, czy właśnie ze strumienia danych

# Założenia projektowe

Przy projektowaniu i wykonywaniu systemu przyjęliśmy następujące założenia:

* Poszczególne elementy systemu odpowiadające danej funkcji np. przechowywanie danych, wyświetlanie danych czy analiza zostaną wydzielone jako moduły
* Moduły zostaną dostarczone jako kontenery systemu Docker
* System będzie składał się z następujących elementów:
  + symulatora aplikacji webowej,
  + systemu kolejek,
  + magazynu danych,
  + modułu przetwarzającego dane,
  + aplikacji pozwalającej na wizualizację danych
* Dane wyświetlane będą w czasie rzeczywistym
* System będzie przenośny i łatwy w uruchomieniu
* Przetwarzane dane będą przechowywane w magazynie danych

# Podział ról i harmonogram prac

Pracę wykonujemy w metodologii scrum.

## Rejestr produktów ( backlog )

|  |  |  |
| --- | --- | --- |
| NR SPRINT | Nazwa zadania | Priorytet |
| **1** | Stworzenie kontenera dockera do symulatora | High |
| Wytworzenie orkiestratora z pierwszą usługą i systemem kolejek | High |
| Ustalenie formatu zdarzeń wysyłanych przez aplikacje | High |
| **2** | Symulator aplikacji internetowej | High |
| Stworzenie kontenera z Apache Spark | High |
| Przetwarzanie zdarzeń przez PySpark | Medium |
| **3** | Stworzenie kontenera z magazynem danych | High |
| Program do odbierania przetworzonych zdarzeń | Medium |
| Usługa do wizualizacji danych | High |
| **4** | Opracowanie dokumentacji | Medium |

## Role

Z racji małego zespołu nie mogliśmy wypracować typowego podziału dlatego pracowaliśmy wszyscy jako właściciele produktu i developerzy jednocześnie.

* Łukasz Wasilewski – programista, członek zespołu
* Szewczyk Bartłomiej – programista, członek zespołu
* Patrycja Abramowska – programista, członek zespołu

## Harmonogram prac

Pracę podzieliśmy na 4 sprinty, z czego każdy sprint trwał 4 tygodnie.

# Dyskusja sposobu rozwiązania problemu

Projekt rozpoczynaliśmy od ustalenia celu opisanego w 1 punkcie, następnie bazując na doświadczeniu własnym z pracy i innych projektów opisaliśmy wymagania oraz wstępny wygląd architektury. W celu ułatwienia rozmów przygotowaliśmy diagram systemu:

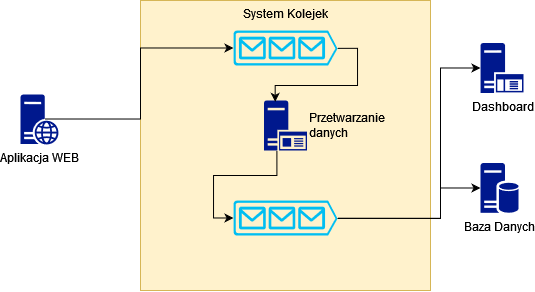


Diagram 1 Ogólny diagram architektury systemu

Po doprecyzowaniu wymagań podzieliśmy się modułami i indywidualnie wyszukiwaliśmy oprogramowanie które realizuje ustalone zadania lub wymaga małej modyfikacji aby tak było. W drodze dyskusji wybrane zostały poszczególne produkty: Grafana do wizualizacji ze względu na popularność i naszą znajomość oprogramowania, Postgres jako baza danych (pierwotnie miało być zastosowane MongoDB, plugin do Grafany pozwalający ustawić MongoDB jako źródło danych jest płatny), Kafka i Spark do obsługi kolejek oraz przetwarzania danych i generator zdarzeń, który został napisany samodzielnie w pythonie.

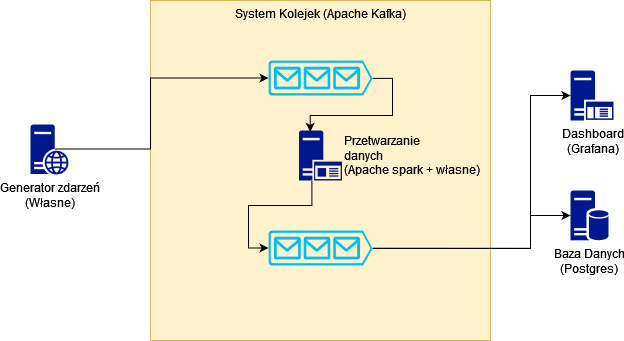
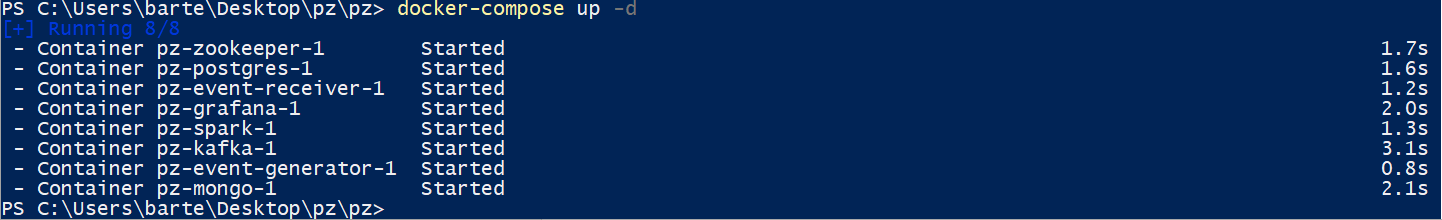


Diagram 2 Ogólny diagram architektury z systemu z wybranym oprogramowaniem

# Prezentacja i opis wyników

Przygotowane przez nas oprogramowanie wymaga do uruchomienia oprogramowania docker oraz docker-compose, a samo uruchomienie przebiega łatwo i szybko. Najpierw należy pobrać kod z repozytorium projektu[link], a następnie przemieścić się do pobranego folderu i wydać komendę która automatycznie zbuduje potrzebne kontenery i uruchomi je na naszej maszynie.  
 Gdy wszystkie kontenery się uruchomią zostanie nam zwrócona informacja że wszystkie kontenery są w stanie „Started” i otrzymamy z powrotem kontrolę w terminalu:

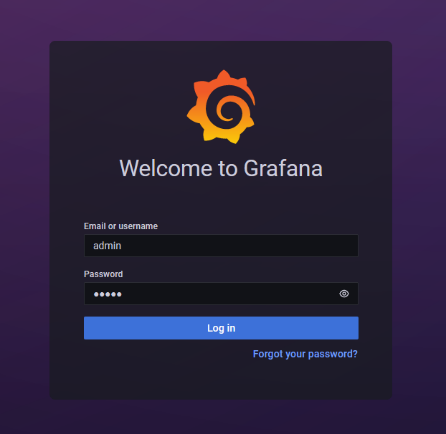


Normalnie aplikacja podłączała by się do innych działających już aplikacji, jednak w naszym przypadku do celów prezentacyjnych przygotowaliśmy generator zdarzeń który teraz należy uruchomić. Skrypt generatora uruchamia się poprzez zalogowanie się do kontenera pz-event-generator i wykonanie komendy:



Obraz zawierający tekst

Opis wygenerowany automatycznie



Podczas gdy skrypt działa w tle można zaobserwować wyniki na dashboardzie dostępnym po zalogowaniu jako admin (hasło admin) pod <http://localhost:3000> .

Obraz zawierający tekst, monitor, wewnątrz, komputer

Opis wygenerowany automatycznie