Instytut Elektrotechniki Teoretycznej i Systemów Informacyjno-Pomiarowych

Praca dyplomowa magisterska

na kierunku Informatyka Stosowana
w specjalności Inżyniera Oprogramowania

Analiza danych w czasie rzeczywistym z systemów wieloczujnikowych z wykorzystaniem klastra Kubernetes oraz narzędzi Big Data

Damian Wójcik

promotor
dr. inż. Łukasz Oskwarek

Warszawa, 2025

Spis treści

Streszczenie

Temat: **Analiza danych w czasie rzeczywistym z systemów wieloczujnikowych z wykorzystaniem klastra Kubernetes oraz narzędzi Big Data**

Niniejsza praca magisterska przedstawia projekt (rozdział 3), implementację (rozdziały 5, 7, 8, 9) oraz analizę wydajności i działania (rozdziały 6) systemu do przetwarzania danych w czasie rzeczywistym z wielu czujników, wykorzystując architekturę mikroserwisową opartą na klastrze Kubernetes oraz narzędziach Big Data, w tym Apache Kafka i Apache Spark.

Praca koncentruje się na efektywnym pozyskiwaniu, przetwarzaniu i analizie strumieni danych generowanych przez systemy wieloczujnikowe w środowiskach przemysłowych. Zaproponowany system umożliwia monitorowanie stanu instalacji przemysłowych w czasie rzeczywistym, wykrywanie anomalii oraz przewidywanie potencjalnych awarii.

Przeprowadzone badania i analizy wykazują, że zastosowanie klastra Kubernetes jako platformy do wdrożenia systemu analitycznego zapewnia elastyczne skalowanie zasobów w zależności od obciążenia. Natomiast wykorzystanie narzędzi do przetwarzania strumieniowego, takich jak Apache Spark, umożliwia efektywną analizę dużych wolumenów danych w czasie rzeczywistym.

Praca pokazuje również potencjalne praktyczne zastosowania (rozdział 10) opracowanego rozwiązania w przemyśle, przedstawiając korzyści ekonomiczne i operacyjne wynikające z wdrożenia systemu analitycznego opartego na klastrze Kubernetes oraz narzędziach Big Data.

Słowa kluczowe: Kubernetes, analiza danych w czasie rzeczywistym, Apache Kafka, Apache Spark, systemy wieloczujnikowe, architektura mikroserwisowa, przetwarzanie strumieniowe, Big Data

Abstract

Title: **Real-Time Data Analysis from Multi-Sensor Systems Using Kubernetes Cluster and Big Data Tools**

This master's thesis presents the design (chapter 3), implementation (chapters 5, 7, 8, 9), and performance and operational analysis (chapters 6) of a real-time data processing system for multi-sensor environments, utilizing a microservice architecture based on a Kubernetes cluster and Big Data tools, including Apache Kafka and Apache Spark.

The thesis focuses on efficiently acquiring, processing, and analyzing data streams generated by multi-sensor systems in industrial environments. The proposed system enables real-time monitoring of industrial installations, anomaly detection, and prediction of potential failures.

The conducted research and analysis demonstrate that using a Kubernetes cluster as a platform for deploying an analytical system provides flexible resource scaling depending on workload. While utilizing stream processing tools such as Apache Spark enables efficient analysis of large volumes of data in real-time.

The thesis also exhibits potential practical applications (chapter 10) of the developed solution in industry, presenting economic and operational benefits resulting from the implementation of an analytical system based on a Kubernetes cluster and Big Data tools.

Keywords: Kubernetes, real-time data analysis, Apache Kafka, Apache Spark, multi-sensor systems, microservice architecture, stream processing, Big Data

1 Wprowadzenie

1.1 Kontekst i motywacja badań

W dobie czwartej rewolucji przemysłowej (Przemysł 4.0) [Przemysł 4.0(2017)] i Internetu Rzeczy (IoT, ang. Internet of Things) [Amazon Web Services(2024a)], definiowanego jako sieć połączonych ze sobą urządzeń fizycznych wyposażonych w czujniki i oprogramowanie umożliwiające zbieranie i wymianę danych, systemy wieloczujnikowe stają się nieodłącznym elementem nowoczesnych procesów produkcyjnych. Generują one ogromne ilości danych, które odpowiednio wykorzystane mogą dostarczyć cennych informacji na temat stanu i wydajności monitorowanych procesów. Analiza tych danych w czasie rzeczywistym stanowi jednak wyzwanie, zarówno pod względem technicznym, jak i organizacyjnym [Challenges(2022)].

Tradycyjne podejścia do przetwarzania danych, oparte na scentralizowanych systemach, często nie są w stanie efektywnie obsłużyć dużej liczby równoczesnych strumieni danych przy zachowaniu niskich opóźnień. Ponadto, skalowanie takich systemów w odpowiedzi na rosnące obciążenie może być problematyczne i kosztowne.

Rozproszona architektura, w połączeniu z technologiami konteneryzacji i orkiestracji, takimi jak Kubernetes [Benefits(2022)], oferuje nowe możliwości w zakresie budowy systemów do analizy danych w czasie rzeczywistym. Umożliwia ona elastyczne skalowanie, izolację usług i łatwiejsze zarządzanie złożonymi aplikacjami. Jednocześnie narzędzia Big Data, czyli technologie służące do przetwarzania i analizy dużych, złożonych zbiorów danych, takie jak: Apache Kafka czy Apache Spark [Kafka(2022), Streaming(2022)], zapewniają wydajne mechanizmy do przetwarzania strumieniowego, niezbędne w analizie danych w czasie rzeczywistym.

Motywacją do podjęcia niniejszych badań jest potrzeba sprawdzenia możliwości do opracowania wydajnego i skalowalnego do analizy danych w czasie rzeczywistym z systemów wieloczujnikowych, który mógłby być wykorzystany w różnych procesach przemysłowych.

1.2 Cel i zakres pracy

Głównym celami niniejszej pracy są zaprojektowanie, implementacja i analiza wydajności systemu do przetwarzania danych w czasie rzeczywistym z wielu czujników.

Szczegółowe cele pracy obejmują:

- opracowanie architektury systemu do analizy danych strumieniowych z systemów wieloczujnikowych, opartego na klastrze Kubernetes i narzędziach Big Data,
- implementacja systemu zgodnie z zaprojektowaną architekturą, z wykorzystaniem mikroserwisów i technologii przetwarzania strumieniowego,
- opracowanie i implementacja algorytmów do analizy danych strumieniowych, w tym algorytmów do detekcji anomalii i przewidywania awarii,
- przeprowadzenie badań eksperymentalnych w celu oceny wydajności i skalowalności opracowanego systemu,
- analiza przydatności systemu w środowisku przemysłowym.

Zakres pracy obejmuje:

- analizę istniejących rozwiązań do przetwarzania danych w czasie rzeczywistym,

- projekt i implementację systemu przetwarzania danych opartego na klastrze Kubernetes,
- opracowanie mechanizmów agregacji i analizy danych z wielu czujników,
- implementację wizualizacji danych w czasie rzeczywistym,
- testowanie wydajności i skalowalności systemu,
- ocenę przydatności zaimplementowanego rozwiązania w środowisku produkcyjnym.

1.3 Struktura pracy

Praca składa się z następujących rozdziałów:

- **rozdział 1: Wprowadzenie** - przedstawia kontekst i motywację badań, cel i zakres pracy, tezę oraz metodologię badań,
- **rozdział 2: Przegląd literatury i stan wiedzy** - omawia istniejące rozwiązania w zakresie analizy danych w czasie rzeczywistym, architektury mikroserwisowej, Kubernetes oraz narzędzi Big Data,
- **rozdział 3: Projekt systemu przetwarzania danych w czasie rzeczywistym** - przedstawia wymagania, architekturę, model danych oraz przepływ danych w projektowanym systemie,
- **rozdział 4: Implementacja systemu na klastrze Kubernetes** - opisuje konfigurację klastra, wdrożenie mikroserwisów, konfigurację Apache Kafka i Kafka Streams, implementację przetwarzania strumieniowego,
- **rozdział 5: Algorytmy analizy danych w czasie rzeczywistym** - omawia zaimplementowane algorytmy do analizy danych,
- **rozdział 6: Badania eksperymentalne i analiza wyników** - przedstawia metodologię testowania, scenariusze testowe, wyniki badań wydajnościowych,
- **rozdział 7: Zastosowania praktyczne** - omawia praktyczne zastosowania opracowanego systemu w przemyśle, korzyści ekonomiczne i operacyjne oraz wyzwania wdrożeniowe,
- **rozdział 8: Podsumowanie i wnioski** - zawiera podsumowanie pracy, główne osiągnięcia i wnioski, ograniczenia badań oraz kierunki przyszłych badań.

2 Przegląd literatury i stan wiedzy

Niniejszy rozdział przedstawia przegląd literatury oraz aktualny stan wiedzy w dziedzinach kluczowych dla realizacji celów pracy. Omówione zostaną zagadnienia związane z analizą danych w czasie rzeczywistym, architekturami systemów rozproszonych, ze szczególnym uwzględnieniem mikroserwisów i konteneryzacji, a także narzędziami Big Data wykorzystywanymi do przetwarzania i analizy dużych wolumenów informacji.

2.1 Analiza danych w czasie rzeczywistym

Analiza danych w czasie rzeczywistym (ang. Real-Time Analytics) odnosi się do procesów i technologii umożliwiających analizowanie danych natychmiast po ich wygenerowaniu lub zebraniu [Analytics(2022)]. Głównym celem jest uzyskanie natychmiastowych wniosków i reakcja na zdarzenia w możliwie najkrótszym czasie. Odróżnia to analizę w czasie rzeczywistym od tradycyjnego przetwarzania wsadowego (ang. batch processing), gdzie dane są gromadzone przez pewien okres i analizowane później [Processing(2022)].

Kluczowe aspekty analizy danych w czasie rzeczywistym to:

- **niskie opóźnienia (ang. Low Latency)**, oznaczające minimalny czas od momentu pojawienia się danych do momentu uzyskania wyników analizy,
- **wysoka przepustowość (ang. High Throughput)**, czyli zdolność systemu do przetwarzania dużej liczby zdarzeń lub transakcji w jednostce czasu,
- **ciągłe przetwarzanie (ang. Continuous Processing)**, gdzie dane są analizowane w sposób ciągły, a nie w odizolowanych partiach.

W kontekście analizy danych z systemów wieloczujnikowych, szczególnie istotne jest przetwarzanie strumieniowe, które umożliwia szybką reakcję na zmieniające się warunki procesu produkcyjnego.

2.1.1 Wyzwania w analizie danych strumieniowych

Analiza danych strumieniowych, choć oferuje wiele korzyści, stawia przed projektantami systemów szereg istotnych wyzwań technicznych - przedstawionych poniżej:

- **Duża objętość i wysoka prędkość danych** - systemy czasu rzeczywistego powinny obsługiwać ogromne ilości danych napływających z wysoką prędkością. Wymaga to wydajnej infrastruktury zdolnej do przetwarzania danych z minimalnym opóźnieniem. Rozwiązaniem jest wykorzystanie systemów rozproszonych, optymalizacja algorytmów oraz efektywne zarządzanie zasobami.
- **Wymóg niskich opóźnień** - aplikacje czasu rzeczywistego wymagają niemal natychmiastowego przetwarzania danych i generowania odpowiedzi. Opóźnienia mogą prowadzić do dezaktualizacji wyników i nieprawidłowych decyzji. Kluczowe jest zastosowanie przetwarzania w pamięci operacyjnej (zarezerwowanej dla aplikacji), optymalizacji zapytań i minimalizacji operacji wejścia/wyjścia.

- **Spójność i dokładność danych** - utrzymanie spójności danych podczas ich przetwarzania jest krytyczne dla poprawności analizy. Systemy powinny zapewnić dokładną semantykę przetwarzania (exactly-once processing) i obsługę braku uporządkowania czasowego w danych.

- **Odporność na awarie i niezawodność** - systemy analizy czasu rzeczywistego powinny działać nieprzerwanie, nawet w przypadku awarii poszczególnych komponentów. Wymagane są mechanizmy automatycznego odzyskiwania po awarii, replikacji danych i równoważenia obciążenia.

- **Przetwarzanie złożonych zdarzeń** - wykrywanie wzorców i korelacji w strumieniach danych wymaga zaawansowanych technik przetwarzania zdarzeń. Konieczne jest zastosowanie algorytmów CEP (Complex Event Processing) zdolnych do identyfikacji istotnych wzorców w czasie rzeczywistym.

- **Integracja z istniejącymi systemami** - nowe rozwiązania powinny współpracować z istniejącą infrastrukturą IT. Wymaga to wykorzystania standardowych interfejsów, brokerów wiadomości i.

- **Skalowalność** - systemy powinny płynnie dostosowywać się do zmieniających się wolumenów danych i wzorców ruchu. Kluczowe jest projektowanie architektury umożliwiającej poziome skalowanie oraz elastyczne przydzielanie zasobów.

- **Bezpieczeństwo i prywatność** - ochrona wrażliwych danych podczas przetwarzania w czasie rzeczywistym stanowi istotne wyzwanie. Systemy powinny implementować szyfrowanie, kontrolę dostępu oraz mechanizmy audytu, zapewniając jednocześnie zgodność z przepisami dotyczącymi ochrony danych.

Skuteczne rozwiązanie tych wyzwań wymaga starannego projektowania architektury systemu, doboru odpowiednich technologii oraz uwzględnienia specyficznych wymagań aplikacji. W kolejnych rozdziałach omówione zostaną narzędzia i techniki pozwalające sprostać tym wyzwaniom.

2.2 Systemy wieloczujnikowe w przemyśle

Systemy wieloczujnikowe to zbiory urządzeń pomiarowych, monitorujących różne parametry procesów przemysłowych [Systems(2022)]. W nowoczesnych zakładach produkcyjnych, systemy te generują ogromne ilości danych, które odpowiednio przetworzone mogą dostarczyć cennych informacji na temat stanu procesu.

2.2.1 Rodzaje czujników przemysłowych

W przemyśle wykorzystuje się różne rodzaje czujników, odnoszące się do różnych wielkości fizycznych, takich jak:

- **czujniki temperatury** - monitorują temperaturę w różnych punktach instalacji,

- **czujniki ciśnienia** - mierzą ciśnienie gazu lub cieczy w instalacji,

- **czujniki przepływu** - monitorują przepływ cieczy i gazów przez instalację,

- **czujniki składu gazu** - analizują skład mieszanin gazowych, np. zawartość wodoru, azotu, amoniaku, tlenu czy dwutlenku węgla,
- **czujniki drgań i hałasu** - monitorują drgania i hałas generowane przez urządzenia, co może wskazywać na potencjalne problemy,
- i inne.

2.2.2 Architektura systemów wieloczujnikowych

Tradycyjne systemy wieloczujnikowe opierają się na architekturze trójwarstwowej:

- **warstwa czujników** - obejmuje czujniki fizyczne wraz z konwerterami analogowo-cyfrowymi,
- **warstwa komunikacji** - odpowiada za przesyłanie danych z czujników do warstwy przetwarzania,
- **warstwa przetwarzania** - przetwarza dane z czujników i udostępnia je użytkownikom lub innym systemom.

W nowoczesnych systemach, warstwa komunikacji często wykorzystuje technologie IoT, takie jak: MQTT, AMQP czy OPC UA, a warstwa przetwarzania opiera się na mechanizmach przetwarzania strumieniowego, takich jak:

- Apache Kafka [Apache Spark(2024)],
- Apache Flink [Flink(2022)].

2.3 Przedstawienie architektury mikroserwisowej

Architektura mikroserwisowa to podejście do tworzenia aplikacji jako zbioru luźno powiązanych, małych, autonomicznych usług, komunikujących się ze sobą za pomocą mechanizmów, takich jak protokół HTTP [Microservices(2022)]. Każdy mikroserwis realizuje określoną funkcjonalność biznesową i może być rozwijany, wdrażany i skalowany niezależnie od innych usług.

2.3.1 Zalety architektury mikroserwisowej

Architektura mikroserwisowa oferuje szereg zalet w kontekście systemów do analizy danych w czasie rzeczywistym [Benefits(2022)]:

- **skalowalność** - możliwość niezależnego skalowania poszczególnych usług w zależności od obciążenia,
- **odporność na awarie** - awaria jednego mikroserwisu nie powoduje awarii całego systemu,
- **elastyczność technologiczna** - możliwość wykorzystania różnych technologii i języków programowania w różnych mikroserwisach,
- **szybsze wdrażanie** - możliwość niezależnego wdrażania poszczególnych mikroserwisów,
- **łatwiejsze zarządzanie kodem** - mniejsze, bardziej zrozumiałe bazy kodu dla poszczególnych usług.

2.3.2 Wyzwania architektury mikroserwisowej

Architektura mikroserwisowa stawia również pewne wyzwania, takie jak [Challenges(2022)]:

- **złożoność operacyjna** - zarządzanie wieloma niezależnymi usługami może być skomplikowane,
- **spójność danych** - z uwagi na trudności w utrzymaniu spójności danych między różnymi mikroserwisami,
- **koszty komunikacji sieciowej** - komunikacja między mikroserwisami wprowadza dodatkowe opóźnienia,
- **testowanie end-to-end** - z uwagi na trudności w testowaniu całego systemu złożonego z wielu niezależnych usług.

2.4 Kubernetes jako platforma orkiestracji kontenerów

Kubernetes to otwarta platforma do orkiestracji kontenerów, która automatyzuje wdrażanie, skalowanie i zarządzanie aplikacjami kontenerowymi [Kubernetes(2022)]. Powstała jako projekt Google, obecnie rozwijana przez [Cloud Native Computing Foundation(2024)].

2.4.1 Podstawowe koncepcje platformy Kubernetes

Kubernetes bazuje na kilku kluczowych koncepcjach [Concepts(2022)]:

- **pod** - najmniejsza jednostka wdrożeniowa w Kubernetes, składająca się z jednego lub więcej kontenerów,
- **deployment** - opakowanie na pody określający ich pożądany stan, umożliwiając ich skalowanie i aktualizacje,
- **service** - abstrakcja definiująca logiczny zestaw podów i politykę dostępu do nich,
- **ingress** - abstrakcja zarządzająca zewnętrznym dostępem do usług w klastrze,
- **configMap** i **Secret** - mechanizmy do przechowywania konfiguracji i tajnych danych,
- **namespace** - mechanizm do izolacji zasobów w klastrze.

2.4.2 Zalety Kubernetes w kontekście analizy danych strumieniowych

Kubernetes cechuje szereg zalet w kontekście rozproszonych systemów [Benefits(2022)]:

- **automatyczne skalowanie** - możliwość automatycznego dostosowywania liczby replik usług w zależności od obciążenia,
- **samonaprawianie** - automatyczne ponowne uruchamianie podów doznających awarii,
- **równoważenie obciążenia** - równomierne rozłożenie ruchu między replikami usług,
- **aktualizacje bez przestojów** - możliwość aktualizacji usług bez przerywania ich działania,
- **deklaratywna konfiguracja** - definiowanie pożądanego stanu systemu, a nie kroków do jego osiągnięcia.

2.5 Narzędzia Big Data do przetwarzania strumieniowego

W kontekście analizy danych strumieniowych, szczególnie istotne są narzędzia Big Data do przetwarzania strumieniowego [Streaming(2022)]. Poniżej omówiono kilka kluczowych technologii wykorzystywanych w tym obszarze.

2.5.1 Apache Kafka

Apache Kafka [Kafka(2022)] to rozproszona platforma do przetwarzania strumieniowego, opracowana przez LinkedIn, obecnie rozwijana jako projekt Apache Software Foundation. Kafka oferuje następujące możliwości:

- **wysoka przepustowość** - możliwość obsługi milionów wiadomości na sekundę,
- **trwałość danych** - dane są przechowywane na dysku i replikowane między brokerami,
- **skalowalność** - łatwe skalowanie poziome przez dodawanie nowych brokerów,
- **mechanizm partycjonowania** - umożliwia równoległe przetwarzanie danych,
- **gwarancje dostarczania** - co najmniej raz, co najwyżej raz lub dokładnie raz.

2.5.2 Kafka Streams

Kafka Streams to biblioteka przetwarzania strumieniowego, zintegrowana z Apache Kafka [Streams(2022)]. Oferuje następujące możliwości:

- **przetwarzanie rekord po rekordzie** - minimalne opóźnienia przetwarzania,
- **operacje stanowe i bezstanowe** - możliwość agregacji danych w czasie,
- **okna czasowe** - przetwarzanie danych w zdefiniowanych oknach czasowych,
- **łączenie strumieni** - możliwość łączenia danych z różnych strumieni,
- **Semantyka "dokładnie raz"** - gwarancje przetwarzania dokładnie raz, eliminujące duplikaty i utratę danych.

2.5.3 Apache Flink

Apache Flink to framework przetwarzania strumieniowego, oferujący możliwości podobne do Kafka Streams, ale jako oddzielna platforma [Flink(2022)]. Flink charakteryzuje się:

- **niskimi opóźnieniami** - przetwarzanie rekord po rekordzie z minimalnymi opóźnieniami,
- **wysoką przepustowością** - efektywne przetwarzanie dużych wolumenów danych,
- **Semantyka "dokładnie raz"** - gwarancje przetwarzania dokładnie raz,
- **zaawansowanym zarządzaniem stanem** - efektywne przechowywanie i dostęp do stanu przetwarzania,
- **obsługą czasu zdarzeń** - możliwość przetwarzania danych na podstawie czasu, w którym zdarzenia zostały wygenerowane.

2.5.4 Apache Spark Streaming

Apache Spark Streaming to moduł przetwarzania strumieniowego platformy Apache Spark [Streaming(2022)]. Opiera się na modelu mikrowsadowym, gdzie dane są przetwarzane w małych pakietach. Spark Streaming oferuje:

- **integrację z ekosystemem Spark** - możliwość wykorzystania bibliotek Spark do analizy danych i uczenia maszynowego - ekosystem ten obejmuje m.in. Spark SQL do przetwarzania danych strukturalnych oraz Spark ML (Machine Learning Library) do implementacji algorytmów uczenia maszynowego,
- **wysoką przepustowość** - efektywne przetwarzanie dużych wolumenów danych,
- **odporność na awarie** - automatyczne odtwarzanie stanu po awarii,
- **łatwe skalowanie** - możliwość łatwego skalowania przetwarzania przez dodawanie węzłów.

2.5.5 Apache Avro i Confluent Schema Registry

W rozproszonych systemach przetwarzania danych, zwłaszcza tych wykorzystujących Apache Kafka, kluczowe znaczenie ma efektywna serializacja danych oraz zarządzanie ich schematami. Pozwala to na minimalizację narzutu komunikacyjnego oraz zapewnienie spójności i kompatybilności danych między różnymi komponentami systemu.

Apache Avro to format serializacji danych oparty na schematach, który zapewnia kompaktową reprezentację binarną oraz bogate możliwości ewolucji schematów [Avro(2022)]. Dzięki temu możliwe jest modyfikowanie struktury danych bez zakłócania pracy istniejących producentów i konsumentów. Avro jest szczególnie popularne w ekosystemie Apache Kafka ze względu na swoją wydajność i elastyczność.

Confluent Schema Registry to usługa działająca jako centralne repozytorium schematów (m.in. Avro, JSON Schema, Protobuf) [Confluent Schema Registry(2022)]. Integruje się z klientami Kafka, umożliwiając automatyczną rejestrację, walidację i pobieranie schematów podczas serializacji i deserializacji wiadomości. Schema Registry pomaga w utrzymaniu jakości danych, zapobiega problemom związanym z niekompatybilnością schematów oraz ułatwia zarządzanie zmianami w strukturze danych w dynamicznie rozwijających się systemach. W ramach opisywanego projektu, Confluent Schema Registry zostało wykorzystane do zarządzania schematami Avro dla danych przesyłanych przez Apache Kafka.

2.6 Istniejące narzędzia analizy danych i ich ograniczenia

W literaturze i praktyce przemysłowej istnieje szereg rozwiązań do analizy danych z systemów wieloczujnikowych. Poniżej omówiono kilka z nich, wraz z ich ograniczeniami.

2.6.1 Tradycyjne systemy SCADA

Systemy SCADA (Supervisory Control and Data Acquisition) to tradycyjne rozwiązania do monitorowania i kontroli procesów przemysłowych [SCADA(2022)]. Mimo popularności, systemy te mają pewne ograniczenia w kontekście analizy danych w czasie rzeczywistym:

- **ograniczona skalowalność** - trudności w obsłudze dużej liczby czujników i strumieni danych,

- **monolityczna architektura** - utrudnia elastyczne rozwijanie i modyfikowanie systemu,
- **ograniczone możliwości analityczne** - często koncentrują się na wizualizacji danych, a nie ich głębokiej analizie,
- **wysokie koszty licencji** - komercyjne systemy SCADA często wiążą się z wysokimi kosztami licencji.

2.6.2 Platformy IoT w chmurze

Platformy IoT w chmurze, takie jak: AWS IoT, Azure IoT Hub czy Google Cloud IoT Core, oferują zaawansowane możliwości analizy danych z urządzeń IoT [IoT(2022)]. Mimo to, mają pewne ograniczenia:

- **zależność od dostawcy chmury** - trudności w migracji między różnymi dostawcami,
- **koszty transferu danych** - wysokie koszty przy dużym wolumenie danych,
- **opóźnienia sieciowe** - potencjalne opóźnienia związane z przesyłaniem danych do chmury,
- **ograniczone możliwości dostosowania** - platformy chmurowe oferują określony zestaw usług, które mogą nie spełniać wszystkich wymagań.

2.6.3 Rozwiązania Open Source do analizy danych strumieniowych

Istnieje szereg rozwiązań otwarto-źródłowych (open source) do analizy danych w czasie rzeczywistym, takich jak Apache NiFi [Apache NiFi(2024)], Apache Druid [Apache Druid(2024)] czy InfluxDB [InfluxDB(2024)]. Mimo ich zalet, mają również pewne ograniczenia:

- **złożoność wdrożenia** - konfiguracja i wdrożenie mogą być skomplikowane,
- **ograniczone wsparcie** - wsparcie techniczne może być ograniczone w porównaniu do rozwiązań komercyjnych,
- **konieczność integracji wielu narzędzi** - często wymagają integracji wielu narzędzi, co zwiększa złożoność systemu.

Podsumowując, istniejące rozwiązania do analizy danych z systemów wieloczujnikowych mają pewne ograniczenia uzasadniające potrzebę opracowania nowego systemu, opartego na architekturze mikroserwisowej, klastrze Kubernetes i narzędziach Big Data.

3 Projekt systemu przetwarzania danych w czasie rzeczywistym

W niniejszym rozdziale przedstawiono projekt systemu do analizy danych w czasie rzeczywistym z systemów wieloczujnikowych, opartego na klastrze Kubernetes oraz narzędziach Big Data. Omówiono wymagania systemu, architekturę, model danych oraz przepływ danych.

3.1 Wymagania systemu osadzonego na klastrze Kubernetes

Na podstawie analizy literatury 2 oraz istniejących rozwiązań, zidentyfikowano przedstawione poniżej wymagania dla projektowanego systemu.

3.1.1 Wymagania funkcjonalne

- **Pozyskiwanie danych** - system powinien umożliwiać pozyskiwanie danych z różnych typów czujników, w tym:
  - czujników temperatury,
  - czujników ciśnienia,
  - czujników przepływu,
  - czujników drgań.
- **Przetwarzanie danych** - system powinien umożliwiać przetwarzanie danych w czasie rzeczywistym, w tym:
  - filtrację danych,
  - agregację danych w różnych oknach czasowych,
  - klasyfikację awarii,
  - generowanie zdarzeń.
- **Wizualizacja danych** - system powinien umożliwiać wizualizację danych w czasie rzeczywistym, w tym:
  - wykresy liniowe,
  - wskaźniki i mierniki,
  - alerty i powiadomienia.
- **Zarządzanie użytkownikami** - system powinien umożliwiać zarządzanie użytkownikami, w tym:
  - logowanie użytkowników,
  - różne poziomy uprawnień (admin, operator, gość),
  - rejestrację użytkowników z poziomu panelu administracyjnego,
  - kontrolę dostępu do danych.

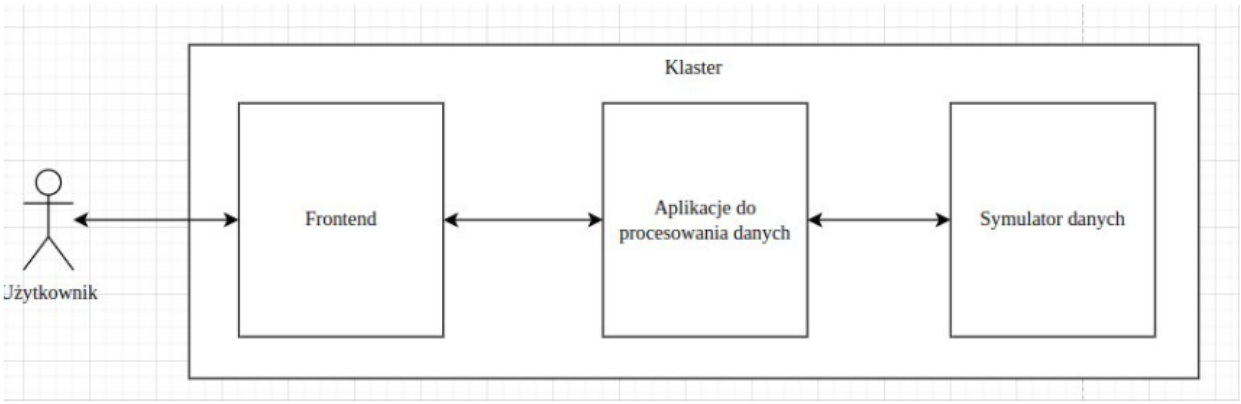
- **Raportowanie** - system powinien umożliwiać generowanie raportów, w tym:
  - tworzenie raportów dla dowolnej zakresu czasu,
  - edycję raportów,
  - wyszukiwanie raportów na podstawie różnych kryteriów.

3.1.2 Wymagania niefunkcjonalne

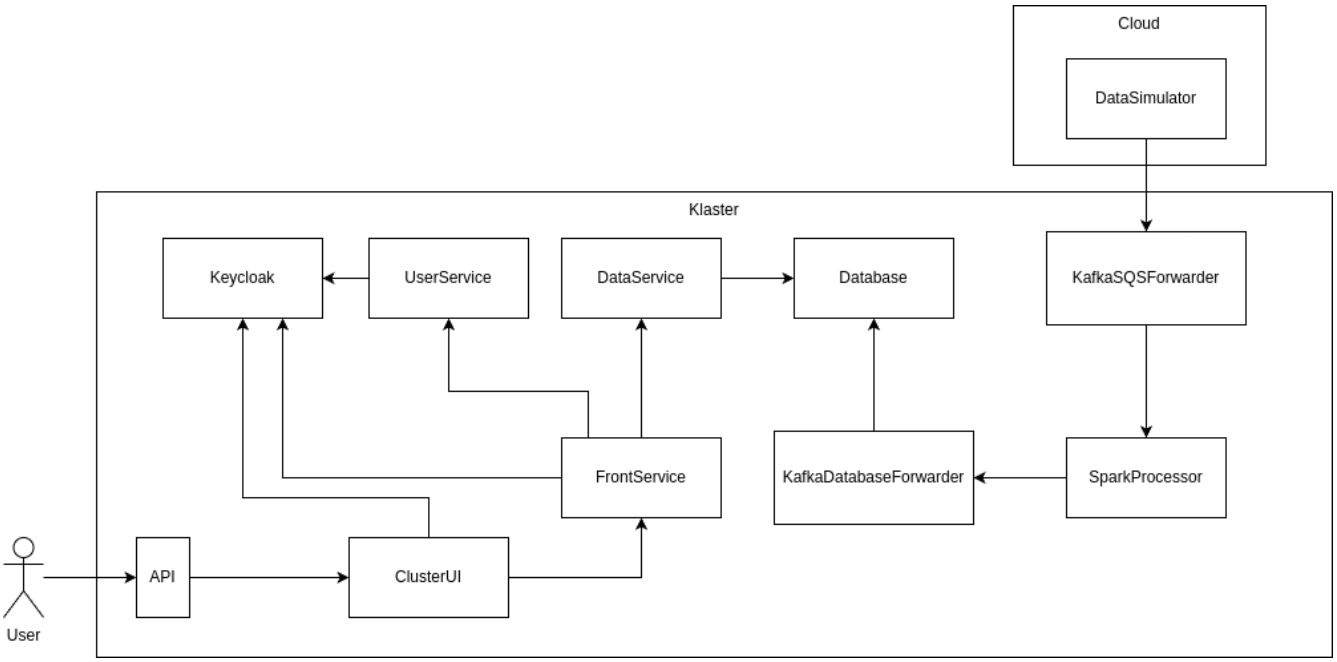
- **Skalowalność** - system powinien umożliwiać:
  - skalowanie horyzontalne (dodawanie nowych węzłów),
  - automatyczne skalowanie w zależności od obciążenia.
- **Niezawodność** - system powinien charakteryzować się:
  - odpornością na awarie pojedynczych komponentów.
- **Bezpieczeństwo** - system powinien zapewniać:
  - szyfrowanie danych w spoczynku i w transporcie,
  - uwierzytelnianie i autoryzację użytkowników,
  - audytowanie dostępu do danych.
- **Utrzymywalność** - system powinien charakteryzować się:
  - modułową architekturą,
  - dobrą dokumentacją,
  - łatwością rozszerzania i modyfikacji.

3.2 Architektura systemu

Projektowany system opiera się na architekturze mikroserwisowej, wdrożonej na klastrze Kubernetes [Benefits(2022)]. Architektura składa się z kilku kluczowych warstw, przedstawionych na rysunku 3.2.2.



Rysunek 3.2.1: Uproszczone architektura systemu do analizy danych w czasie rzeczywistym



Rysunek 3.2.2: Architektura systemu do analizy danych w czasie rzeczywistym

3.2.1 Warstwa pozyskiwania danych (symulatory czujników)

Warstwa pozyskiwania danych odpowiada za zbieranie danych z czujników i ich wstępne przetworzenie. W projektowanym systemie dane są generowane przez symulatory czujników implementowane jako funkcje AWS Lambda [Amazon Web Services(2024b)]. Generują one dane symulujące odczyty z różnych typów czujników.

Dane generowane przez symulatory są formatowane jako wiadomości JSON [JSON Schema(2024)] i publikowane na tematy SNS [AWS SNS(2024)], a następnie przenoszone do kolejek SQS (Amazon Simple Queue Service, zarządzana usługa kolejkowania wiadomości), stanowiących interfejs między AWS a klastrem Kubernetes.

3.2.2 Warstwa komunikacji (Apache Kafka)

Warstwa komunikacji odpowiada za odbieranie danych z warstwy pozyskiwania i ich dostarczanie do warstwy przetwarzania. W projektowanym systemie warstwa ta opiera się na rozproszonej platformie do przetwarzania strumieniowego Apache Kafka [Kafka(2022)].

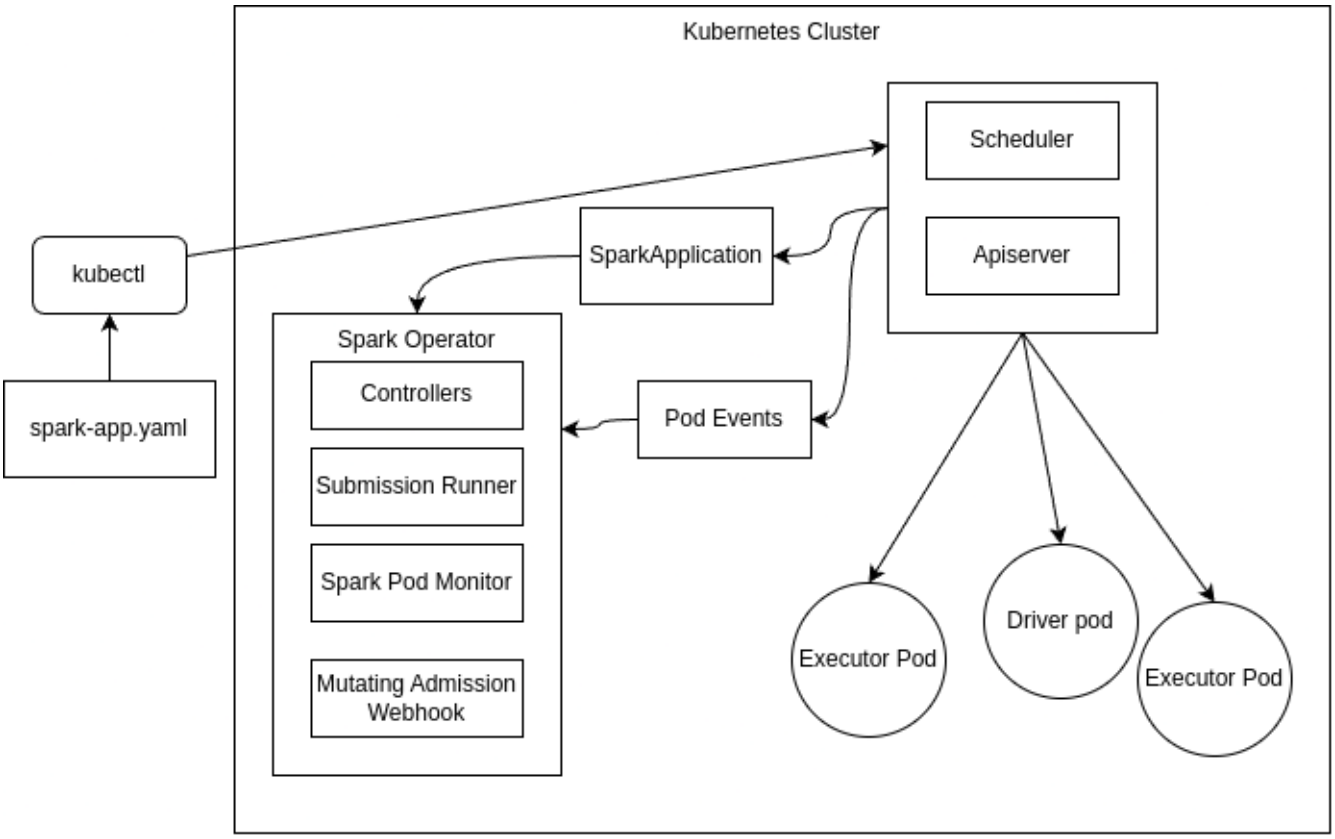
Apache Kafka zapewnia trwałość danych, wysoką przepustowość i skalowalność, co jest kluczowe w kontekście przetwarzania danych w czasie rzeczywistym z wielu czujników.

3.2.3 Warstwa przetwarzania (Apache Spark)

Warstwa przetwarzania odpowiada za analizę danych w czasie rzeczywistym. W projektowanym systemie warstwa ta opiera się na frameworku Apache Spark [Streaming(2022)], wykorzystując jego moduł Spark Structured Streaming do przetwarzania strumieni danych. Głównym komponentem tej warstwy jest aplikacja `SparkDataProcessor`, zrealizowana w języku Scala. Aplikacja ta zawiera logikę przetwarzania danych, podzieloną na wyspecjalizowane moduły takie jak: `MeanProcessor` (obliczanie średnich), `EventProcessor` (wykrywanie zdarzeń na podstawie progów) oraz `EquipmentStateProcessor` (predykcja stanu urządzenia przy użyciu modelu uczenia

maszynowego RandomForestClassifier). Szczegółowy opis tych modułów oraz ich działania znajduje się w Rozdziale 6.

Apache Spark umożliwia przetwarzanie danych strumieniowych z wykorzystaniem zaawansowanych operacji, obsługę danych opóźnionych (watermarks), zarządzanie stanem oraz integrację z bibliotekami uczenia maszynowego (Spark ML), co jest kluczowe w kontekście kompleksowej analizy danych z systemów wieloczujnikowych.



Rysunek 3.2.3: Działanie Spark Operatora z Driverami i aplikacją

3.2.4 Warstwa składowania danych

Warstwa składowania danych odpowiada za przechowywanie przetworzonych danych do dalszej analizy i wizualizacji. W projektowanym systemie warstwa ta składa się z dwóch głównych komponentów:

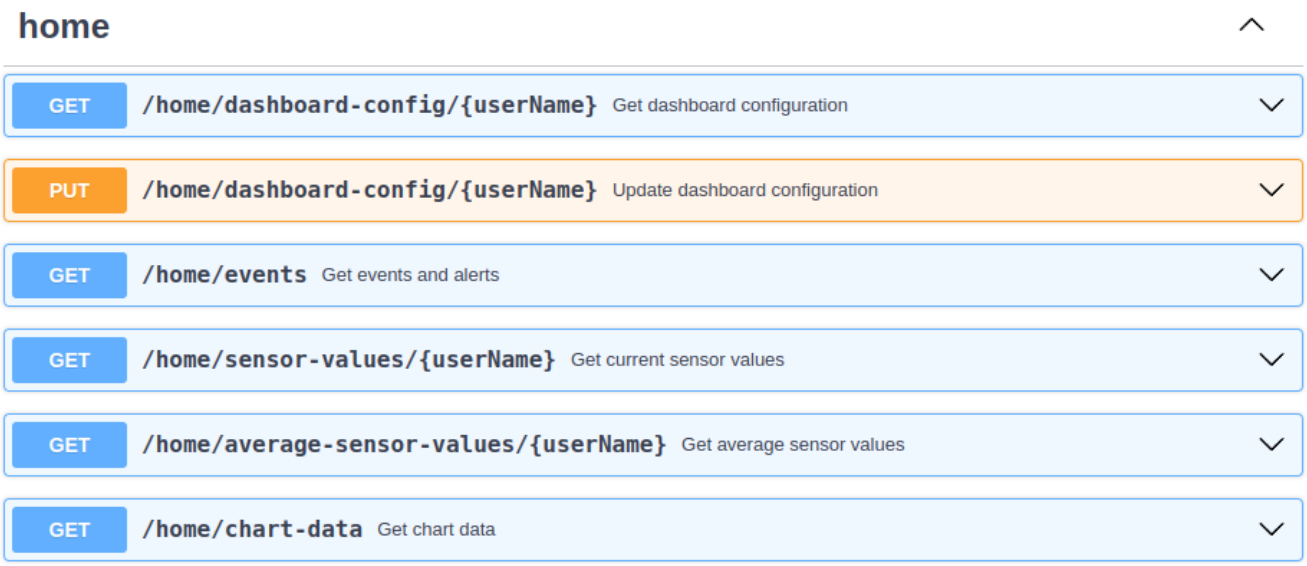
- **PostgreSQL** - relacyjna baza danych, przechowująca ustrukturyzowane dane, takie jak: odczyty czujników, metadane czy informacje o użytkownikach,
- **Elasticsearch** - baza danych NoSQL [NoSQL(2024)], umożliwiająca szybkie wyszukiwanie i analizę danych, szczególnie przydatna w kontekście wykrywania anomalii i analizy trendów.

3.2.5 Warstwa usług i interfejs API

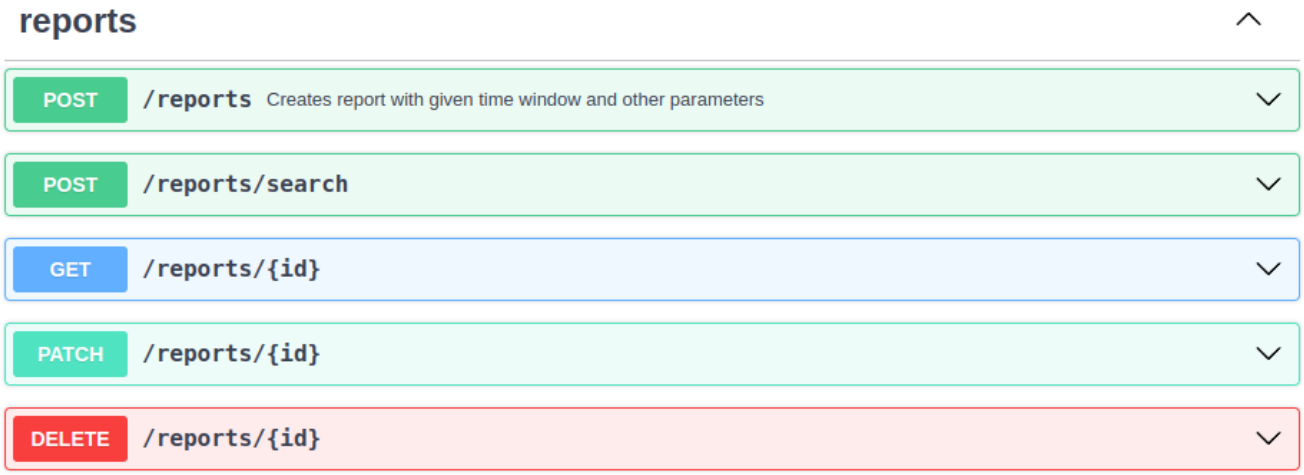
Warstwa usług i interfejs API odpowiada za udostępnianie danych i funkcjonalności systemu zewnętrznym aplikacjom i użytkownikom. W projektowanym systemie warstwa ta składa się z kilku mikroserwisów:

- **Data Service** - mikroserwis udostępniający API do dostępu do danych z czujników, umożliwiający tworzenie i pobieranie raportów,
- **Users Service** - mikroserwis zarządzający użytkownikami i uwierzytelnianiem, wykorzystujący Keycloak jako system zarządzania tożsamością,
- **Front Service** - mikroserwis pełniący rolę API Gateway [API Gateway(2024)], który wystawia interfejs API innych mikroserwisów na zewnątrz klastra.

Wszystkie mikroserwisy są implementowane jako aplikacje Spring Boot, co zapewnia łatwość rozwoju, testowania i wdrażania.



Rysunek 3.2.4: API sekcji Home



Rysunek 3.2.5: API sekcji Reports

3.2.6 Warstwa prezentacji

Warstwa prezentacji odpowiada za wizualizację danych i interakcję z użytkownikami. W projektowanym systemie warstwa ta składa się z panelu sterowania, wyświetlający dane w formie wykresów, tabel i widżetów.

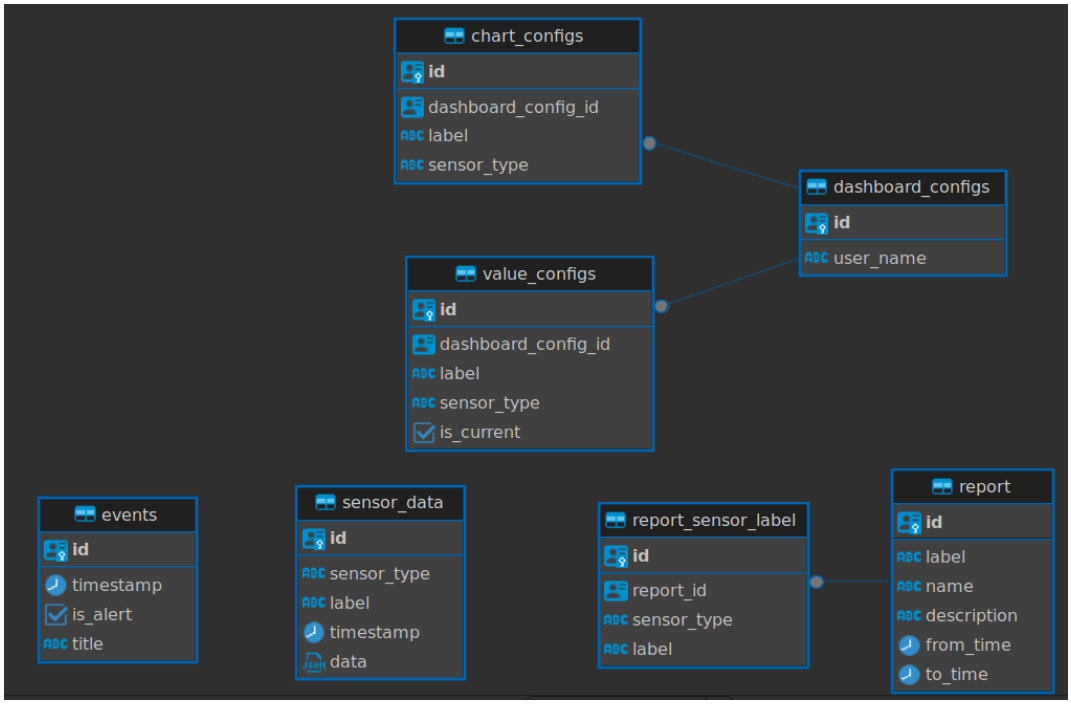
Główne narzędzia panelu sterowania to:

- **wykresy liniowe** - wizualizacja zmian parametrów w czasie,
- **wskaźniki i mierniki** - wizualizacja aktualnych wartości parametrów,
- **alerty** - powiadomienia o anomaliach i potencjalnych awariach,
- **raporty** - generowanie i przeglądanie raportów.

Panel sterowania jest implementowany jako aplikacja webowa, wykorzystująca React i biblioteki wizualizacji danych.

3.3 Model danych

Model danych projektowanego systemu składa się z kilku kluczowych encji, przedstawionych na rysunku 3.3.1.



Rysunek 3.3.1: Model danych systemu

3.3.1 Dane z czujników

Dane z czujników są modelowane jako strumienie zdarzeń, gdzie każde zdarzenie zawiera:

- **typ czujnika** - rodzaj czujnika (temperatura, ciśnienie, przepływ, skład gazu, drgania),

- **znacznik lokalizacji** - miejsce, w którym znajduje się czujnik,
- **znacznik czasu** - czas, w którym dokonano pomiaru,
- **wartość** - wartość pomiaru,
- **jednostka** - jednostka, w której wyrażona jest wartość.

Dane te są serializowane przy użyciu serializatora Apache Avro [Avro(2022)], co zapewnia efektywne kodowanie i dekodowanie wiadomości.

3.4 Model raportów w Elasticsearch

Elasticsearch jest używany do przechowywania i szybkiego wyszukiwania raportów. Używa on algorytmów przeszukiwwania pełnotekstowego, które umożliwiają szybkie wyszukiwanie raportów po różnych parametrach. Po wysłaniu przez API zapytania do DataService, serwis ten wysyła zapytanie do Elasticsearch, które zwraca raporty spełniające kryteria wyszukiwania. Reporty są przechowywane w Elasticsearch w formacie JSON i są w specjalny sposób zindeksowane tak, by dane dało się bardzo szybko wyszukać. Raporty oraz zapytania są normalizowane do małych liter za pomocą normalizatora `lower_case_normalizer` oraz analizatora `lower_case_analyzer`.

```
{
    "properties": {
            "id": {
                "type": "keyword"
            },
            "label": {
                "type": "keyword",
                "normalizer": "lower_case_normalizer"
            },
            "name": {
                "type": "text",
                "fields": {
                    "lowercase": {
                        "type": "text",
                        "analyzer": "lower_case_analyzer"
                    },
                    "sort": {
                        "type": "keyword",
                        "normalizer": "lower_case_normalizer"
                    }
                }
            },
            "description": {
                "type": "text",
                "fields": {
                    "lowercase": {
                        "type": "text",
                        "analyzer": "lower_case_analyzer"
                    }
                }
            },
            "from": {
                "type": "date",
```

```
                    "format": "epoch_millis"
                },
                "to": {
                    "type": "date",
                    "format": "epoch_millis"
                },
                "reportSensorLabels": {
                    "type": "nested",
                    "properties": {
                        "sensorType": {
                            "type": "keyword",
                            "normalizer": "lower_case_normalizer"
                        },
                        "label": {
                            "type": "keyword",
                            "normalizer": "lower_case_normalizer"
                        }
                    }
            }
        }
    }

```

Listing 1: Model raportów w Elasticsearch

```
    {
        "analysis": {
          "normalizer": {
            "lower_case_normalizer": {
              "type": "custom",
              "char_filter": [],
              "filter": [
                "lowercase"
              ]
            }
          },
          "analyzer": {
            "lower_case_analyzer": {
              "type": "custom",
              "tokenizer": "standard",
              "filter": [
                "lowercase"
              ]
            }
          }
        }
      }
```

Listing 2: Analizatory i normalizatory w Elasticsearch

3.4.1 Przetworzone dane

Przetworzone dane są wynikiem analizy danych surowych przez aplikację `SparkDataProcessor` i obejmują:

- **Agregacje (Aggregations)** - średnie wartości oraz liczba odczytów dla poszczególnych typów sensorów, obliczane przez moduł `MeanProcessor` w zdefiniowanych oknach czasowych:

  - *sensor_type* - typ sensora (np. "pressure", "temperature"),
  - *label* - etykieta urządzenia (np. "pump", "compressor"),
  - *window_start* - początek okna czasowego,
  - *window_end* - koniec okna czasowego,
  - *avg_value* - średnia wartość pomiarów w oknie,
  - *count* - liczba pomiarów w oknie.

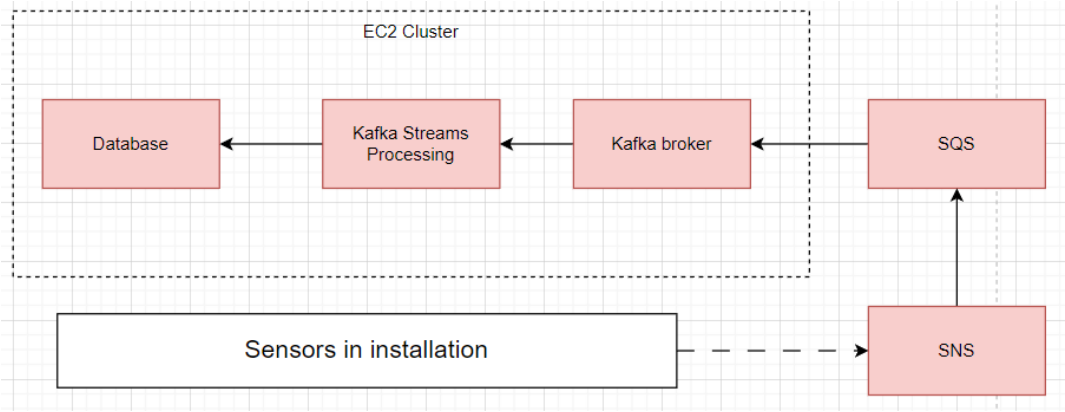
- **Zdarzenia (Events)** - wykryte przez moduł `EventProcessor` przekroczenia zdefiniowanych progów ostrzegawczych i krytycznych dla danych sensorycznych:
  - *event_id* - unikalny identyfikator zdarzenia,
  - *sensor_type* - typ sensora, którego dotyczy zdarzenie,
  - *label* - etykieta urządzenia,
  - *timestamp* - czas wykrycia zdarzenia,
  - *title* - tytuł zdarzenia (np. "Przekroczenie progu krytycznego dla ciśnienia"),
  - *status* - status alertu (np. "Ostrzeżenie", "Krytyczny"),
  - *actual_value* - rzeczywista wartość, która spowodowała zdarzenie.

- **Dane wzbogacone o predykcję stanu (Augmented Data with Predicted Status)** - oryginalne dane sensoryczne wzbogacone o sklasyfikowany stan urządzenia, generowane przez moduł `EquipmentStateProcessor` z użyciem modelu `RandomForestClassifier`:
  - *original_sensor_data* - pełny zestaw oryginalnych danych z sensora (np. ciśnienie, temperatura wraz ze znacznikiem czasu, typem urządzenia itp.),
  - *predicted_status* - przewidziany stan urządzenia (np. "Stan normalny", "Wczesne zużycie", "Stan podkrytyczny", "Stan krytyczny", "Naprawa"),
  - *model_type* - typ modelu użytego do predykcji (tutaj: `RandomForestClassifier`).

Wszystkie te dane są serializowane do formatu Avro i publikowane na dedykowane tematy Kafka. Następnie, wybrane dane (np. zdarzenia, zagregowane dane do wizualizacji) mogą być zapisywane w bazach danych takich jak PostgreSQL lub Elasticsearch przez dedykowane serwisy (np. `Kafka-DB-Forwarder`).

3.5 Przepływ danych w systemie

Przepływ danych w projektowanym systemie obejmuje kilka etapów, przedstawionych na rysunku 3.5.1.



Rysunek 3.5.1: Przepływ danych w systemie

3.5.1 Generowanie danych

Proces rozpoczyna się od generowania danych przez symulatory czujników. Symulatory te są implementowane jako funkcje AWS Lambda [Amazon Web Services(2024b)], które są wyzwalane periodycznie przez funckcje bezpośrednio osadzone na chmurze [AWS Step Functions(2024)]. Każdy symulator generuje dane symulujące odczyty z określonego typu czujnika, umieszczonego w określonym miejscu instalacji.

Dane są generowane w formacie JSON i zawierają informacje takie jak: identyfikator czujnika, typ, lokalizacja, znacznik czasu, wartość i jednostka. Dane te są następnie publikowane na temat SNS [AWS SNS(2024)], który działa jako punkt dystrybucji dla wielu subskrybentów.

3.5.2 Przesyłanie danych do klastra

Dane opublikowane na temacie SNS są automatycznie dostarczane do kolejek SQS [AWS SQS(2024)], które są subskrybentami tematu. Kolejki SQS działają jako bufor między AWS a klastrem Kubernetes, zapewniając niezawodne dostarczanie wiadomości, nawet w przypadku tymczasowej niedostępności klastra.

W klastrze Kubernetes działa mikroserwis SQS-Kafka-Forwarder, który regularnie odpytuje kolejki SQS i pobiera nowe wiadomości. Wiadomości te są następnie deserializowane i publikowane na odpowiednie kanały (ang. topics) Kafki, w zależności od typu czujnika i lokalizacji.

3.5.3 Przetwarzanie strumieni danych

Dane opublikowane na kanałach Kafki są przetwarzane przez mikroserwis `SparkDataProcessor`, który wykorzystuje silnik analityczny Apache Spark (Spark Structured Streaming) [Streaming(2022)] do analizy danych w czasie rzeczywistym. Przetwarzanie obejmuje:

- **obliczanie średnich wartości** dla poszczególnych typów sensorów w zdefiniowanych oknach czasowych (moduł `MeanProcessor`),
- **wykrywanie zdarzeń** na podstawie przekroczenia zdefiniowanych progów ostrzegawczych i krytycznych (moduł `EventProcessor`),
- **predykcję ogólnego stanu technicznego** monitorowanego urządzenia na podstawie połączonych danych z wielu sensorów, przy użyciu modelu `RandomForestClassifier` (moduł `EquipmentStateProcessor`),

- **wzbogacanie oryginalnych danych** sensorycznych o przewidzianą etykietę stanu.

Wyniki przetwarzania, w formacie Avro, są publikowane na nowe, dedykowane kanały Kafka.

3.5.4 Zapisywanie danych do bazy danych

Przetworzone dane są konsumowane przez mikroserwis Kafka-DB-Forwarder, który zapisuje je do baz danych. Dane strukturalne, takie jak: odczyty czujników, agregacje i metadane, są zapisywane do bazy danych PostgreSQL.

Zapisywanie danych do baz danych umożliwia ich późniejszą analizę, raportowanie i wizualizację, a także zapewnia trwałość danych.

3.5.5 Udostępnianie danych przez interfejs API

Zapisane dane są udostępniane przez mikroserwis Data-Service, który ekspozuje REST API do tworzenia i pobierania raportów. API umożliwia:

- **pobieranie danych surowych** - dostęp do nieprzetworzonych odczytów czujników,
- **pobieranie danych przetworzonych** - dostęp do agregacji, wykrytych anomalii, predykcji,
- **tworzenie raportów** - generowanie raportów na podstawie zadanych kryteriów,
- **pobieranie raportów** - dostęp do wcześniej wygenerowanych raportów.

API jest zabezpieczone mechanizmami uwierzytelniania i autoryzacji, implementowanymi przez mikroserwis Users-Service, który korzysta z Keycloak.

3.5.6 Wizualizacja analizowanych danych

Ostatnim etapem przepływu danych jest ich wizualizacja na dashboardzie (ekranie zbiorczym). Dashboard komunikuje się z interfejs API, pobierając dane do wyświetlenia, a następnie prezentuje je w formie wykresów, tabel i widżetów.

Dashboard umożliwia interaktywną eksplorację danych, filtrowanie, sortowanie i eksport wyników. Ponadto, dashboard może wyświetlać alerty i powiadomienia o wykrytych anomaliach i potencjalnych awariach.

Przepływ danych w projektowanym systemie zapewnia efektywne pozyskiwanie, przetwarzanie i analizę danych w czasie rzeczywistym, co jest kluczowe w kontekście monitorowania i optymalizacji procesu syntezy amoniaku.

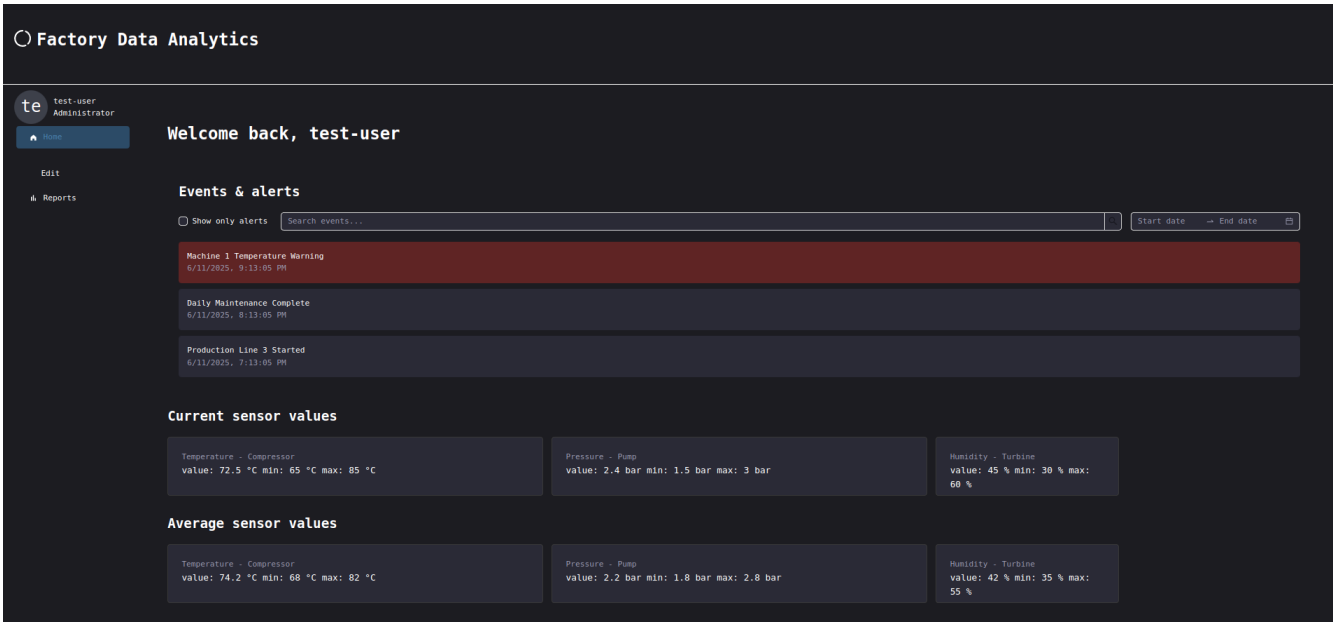
4 Prezentacja wizualna danych i funkcje biznesowe systemu

Interfejs użytkownika (UI) systemu analitycznego został starannie zaprojektowany, aby umożliwić intuicyjną i efektywną pracę z danymi telemetrycznymi. Zbudowany w oparciu o nowoczesne technologie, takie jak: React oraz biblioteka komponentów Ant Design, zapewnia responsywność i bogaty zestaw interakcji. Dostęp do systemu jest zabezpieczony poprzez integrację z serwerem Keycloak, co gwarantuje, że tylko autoryzowani użytkownicy z przypisanymi rolami `DATA_ACCESSOR` lub `ADMIN` mogą korzystać z jego zasobów. Rola `ADMIN` dodatkowo rozszerza uprawnienia o możliwość zarządzania kontami użytkowników, co typowo realizowane jest poprzez dedykowany panel administracyjny Keycloak, zapewniając centralne zarządzanie dostępem.

Celem aplikacji jest dostarczenie użytkownikom narzędzi do monitorowania procesów w czasie rzeczywistym, analizy danych historycznych oraz generowania szczegółowych raportów, co przekłada się na lepsze zrozumienie działania systemu, szybsze wykrywanie anomalii i podejmowanie świadomych decyzji biznesowych.

4.1 Pulpit Główny (Home Dashboard) - Centrum Monitorowania Operacyjnego

Pulpit główny stanowi spersonalizowane centrum dowodzenia dla każdego użytkownika, oferując natychmiastowy przegląd najważniejszych informacji o stanie systemu. Po zalogowaniu, użytkownik jest witany imiennie, co buduje przyjazne środowisko pracy.



Rysunek 4.1.1: Pulpit główny

4.1.1 Monitorowanie zdarzeń i alertów - Świadomość sytuacyjna

Kluczowym elementem pulpitu jest sekcja `Events & alerts`, agregująca i prezentuje zdarzenia systemowe w porządku chronologicznym. Mogą to być zarówno standardowe komunikaty operacyjne, jak i krytyczne alerty informujące o przekroczeniach zdefiniowanych progów dla wartości

sensorów lub innych istotnych incydentach. Funkcjonalność ta ma kluczowe znaczenie dla utrzymania ciągłości operacyjnej:

- **szybka identyfikacja problemów:** możliwość filtrowania listy w celu wyświetlenia **tylko alertów** pozwala operatorom na natychmiastowe skupienie uwagi na sytuacjach wymagających interwencji,
- **analiza incydentów:** zaawansowane opcje przeszukiwania zdarzeń według słów kluczowych oraz filtrowania ich według zdefiniowanego zakresu dat umożliwiają dogłębną analizę przyczyn źródłowych problemów oraz identyfikację wzorców i trendów w przeszłych zdarzeniach.

Informacje o zdarzeniach są dynamicznie pobierane aplikacji przetwrzającej danej `data-service`, które prawdopodobnie agreguje je z różnych źródeł, w tym z systemu przetwarzania strumieniowego (np. na podstawie logiki zaimplementowanej w aplikacji przetwarzającej dane w okienkach czasowych `kafka-data-processor`).

4.1.2 Wizualizacja wartości sensorów - Bezpośredni wgląd w parametry

Panel sterowania umożliwia ciągłe monitorowanie kluczowych parametrów operacyjnych poprzez wyświetlanie wartości z wybranych przez użytkownika sensorów. Prezentacja danych jest podzielona na dwie kategorie, każda dostarczająca innego rodzaju informacji:

- **Current Sensor Values (bieżące wartości sensorów):** ta sekcja prezentuje najnowsze, surowe odczyty bezpośrednio z sensorów. Użytkownik widzi etykietę identyfikującą konkretny punkt pomiarowy (np. *Temperatura - Przed Kotłem*) oraz jego aktualną wartość wraz z odpowiednią jednostką miary (np. °C, hPa, %, m/s). Jednostki te są dynamicznie pobierane z centralnej konfiguracji systemu, co zapewnia spójność prezentacji. Funkcja ta jest nieoceniona dla natychmiastowej oceny stanu poszczególnych komponentów systemu i umożliwia szybkie podjęcie decyzji w przypadku wykrycia nieoczekiwanych odchyleń,
- **Average Sensor Values (uśrednione wartości sensorów):** w odróżnieniu od wartości bieżących, ta sekcja pokazuje uśrednione wartości dla wybranych sensorów z określonego, niedawnego przedziału czasu. Te wartości są prawdopodobnie wynikiem agregacji danych strumieniowych (np. realizowanej przez Kafka Streams na podstawie konfiguracji w `kafka-data-processor-config`), co pozwala na wygładzenie chwilowych fluktuacji i obserwację bardziej stabilnych trendów. Jest to przydatne do oceny ogólnej wydajności i stabilności procesów w średnim okresie.

Personalizacja widoku: - obie sekcje wyświetlające wartości sensorów są w pełni konfigurowalne. W trybie edycji, dostępnym z paska bocznego interfejsu, użytkownik może dynamicznie modyfikować swój pulpit:

- dodawać nowe sensory do monitorowania poprzez wybór typu sensora (np. Temperatura, Ciśnienie, Przepływ) z predefiniowanej listy, a następnie wybór konkretnej etykiety pomiarowej (np. Przed Kotłem, Za Sprężarką Powrotną) dostępnej dla danego typu sensora,
- modyfikować lub usuwać istniejące kafelki z wartościami sensorów.

Konfiguracja pulpitu jest zapisywana indywidualnie dla każdego użytkownika, co pozwala na dostosowanie interfejsu do jego specyficznych potrzeb i zakresu odpowiedzialności. Dodatkowo, w

celu optymalizacji prezentacji, kafelki wyświetlające wartości mogą mieć dynamicznie dostosowywaną szerokość. Dla sensorów oznaczonych w konfiguracji systemowej jako `wideSensors` – typowo tych, które prezentują wiele powiązanych wartości lub dłuższe opisy tekstowe – kafelek automatycznie zajmuje podwójną standardową szerokość. Zapewnia to lepszą czytelność i przejrzystość prezentowanych informacji, eliminując konieczność przycinania tekstu lub nadmiernego zagęszczania danych.

4.1.3 Wykresy danych

Sekcja `Realtime data charts` jest narzędziem do wizualizacji dynamiki zmian wartości sensorów. Wykresy liniowe aktualizują się w czasie zbliżonym do rzeczywistego, co pozwala na proaktywne monitorowanie i wczesne wykrywanie odchyleń od normalnych warunków operacyjnych.

- **Dynamiczna wizualizacja stanu sensora:** kluczową funkcją jest możliwość wizualizacji **stanu sensora** bezpośrednio na wykresie. Realizowane jest to poprzez dynamiczną zmianę koloru tła odpowiedniego regionu wykresu, co dostarcza natychmiastowego wizualnego sygnału o kondycji monitorowanego elementu lub procesu. Umożliwia to operatorom szybką ocenę sytuacji bez konieczności analizowania dokładnych wartości liczbowych,
- **Personalizacja wykresów:** podobnie jak w przypadku wartości numerycznych, użytkownicy w trybie edycji mogą dodawać nowe wykresy (wybierając typ sensora i etykietę), modyfikować konfigurację istniejących lub je usuwać, dostosowując widok do aktualnych potrzeb analitycznych,
- **Regulacja zakresu czasowego:** możliwość zmiany zakresu czasu prezentowanego na wykresach (np. ostatni dzień, ostatnie dwa dni, tydzień) pozwala na analizę zarówno krótkoterminowych fluktuacji, jak i długoterminowych wzorców zachowań systemu.

Rysunek 4.1.2: Wykresy danych

4.2 System raportowania

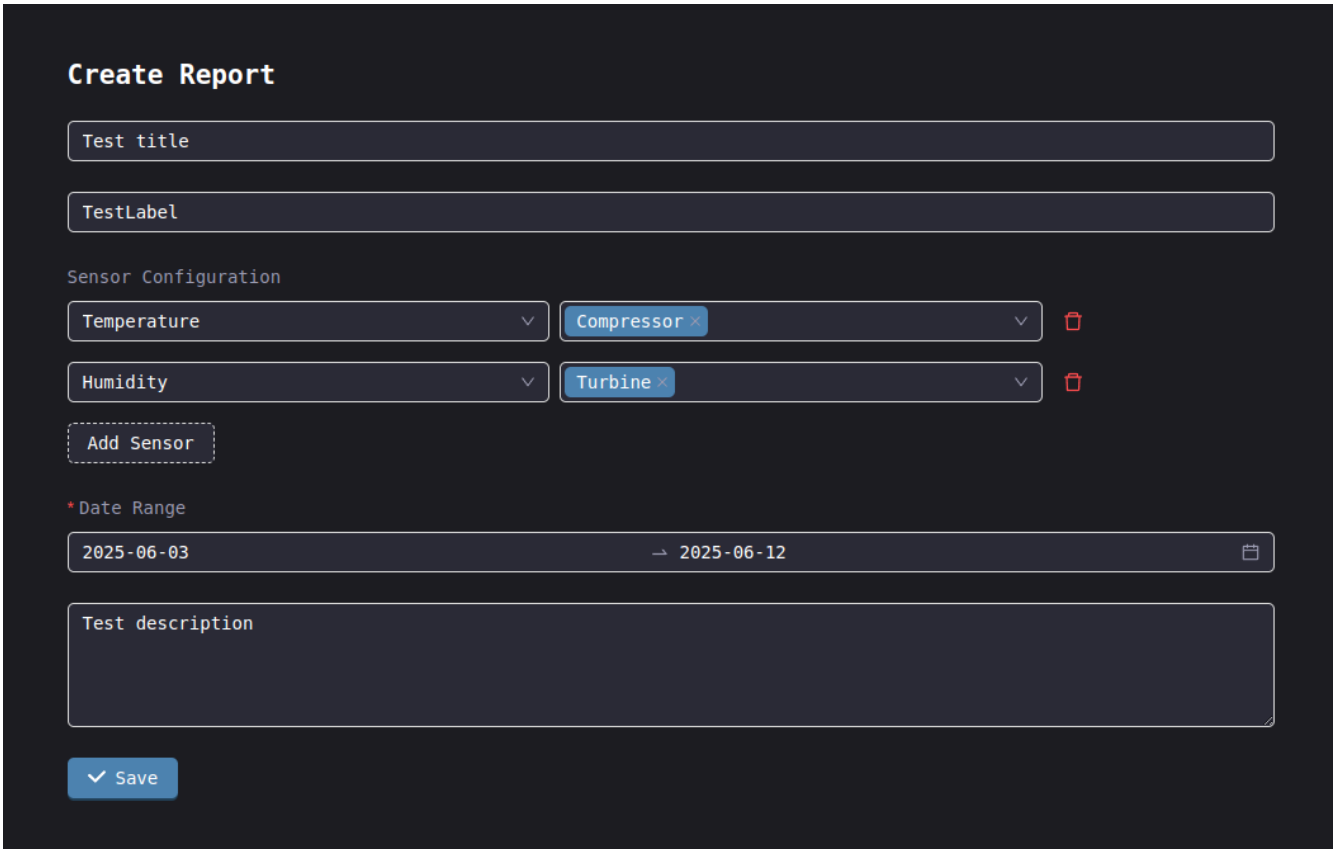
Sekcja `Reports` umożliwia użytkownikom tworzenie, przeglądanie i zarządzanie raportami opartymi na danych historycznych.

4.2.1 Definiowanie raportów

Użytkownik może zdefiniować nowy raport poprzez formularz, w którym określa:

- nazwę, etykietę oraz opis raportu,
- konfigurację sensorów: wybór jednego lub wielu typów sensorów oraz, dla każdego typu, jednej lub wielu konkretnych etykiet pomiarowych, z których dane mają być uwzględnione w raporcie,
- zakres czasowy: precyzyjny przedział dat (od-do), dla którego dane historyczne zostaną pobrane i zaprezentowane.

Zdefiniowany raport jest zapisywany w systemie, a jego konfiguracja przechowywana w bazie danych PostgreSQL oraz indeksowana w Elasticsearch w celu umożliwienia szybkiego wyszukiwania.



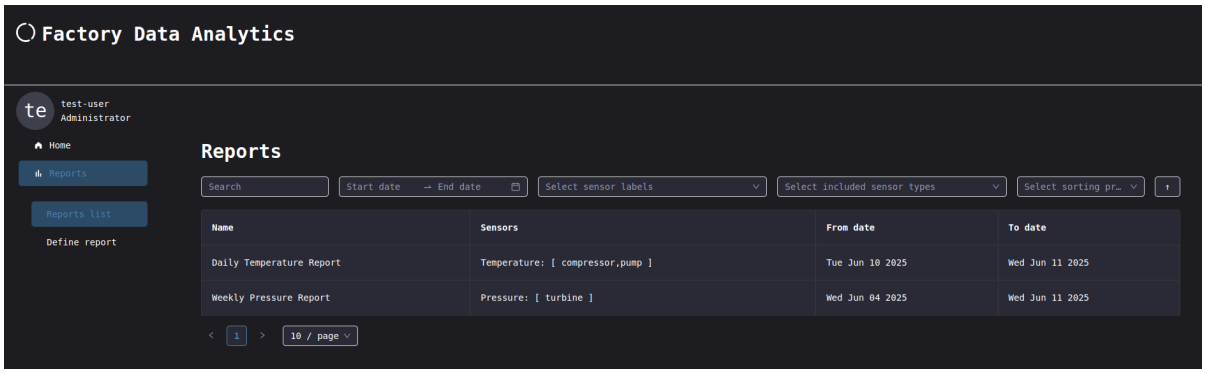
Rysunek 4.2.1: Tworzenie raportu

4.2.2 Przeglądanie i zarządzanie raportami

Lista zdefiniowanych raportów jest prezentowana w formie tabelarycznej, z możliwością paginacji. Użytkownik może:

- wyszukiwać raporty po nazwie lub opisie,
- filtrować listę raportów według zakresu dat, typów sensorów oraz etykiet pomiarowych zawartych w raportach,
- sortować raporty według różnych kryteriów (np. nazwa, data początkowa, data końcowa).

Kliknięcie na raport z listy otwiera jego szczegółowy widok.



Rysunek 4.2.2: Wyszukiwarka raportów

4.2.3 Szczegóły raportu

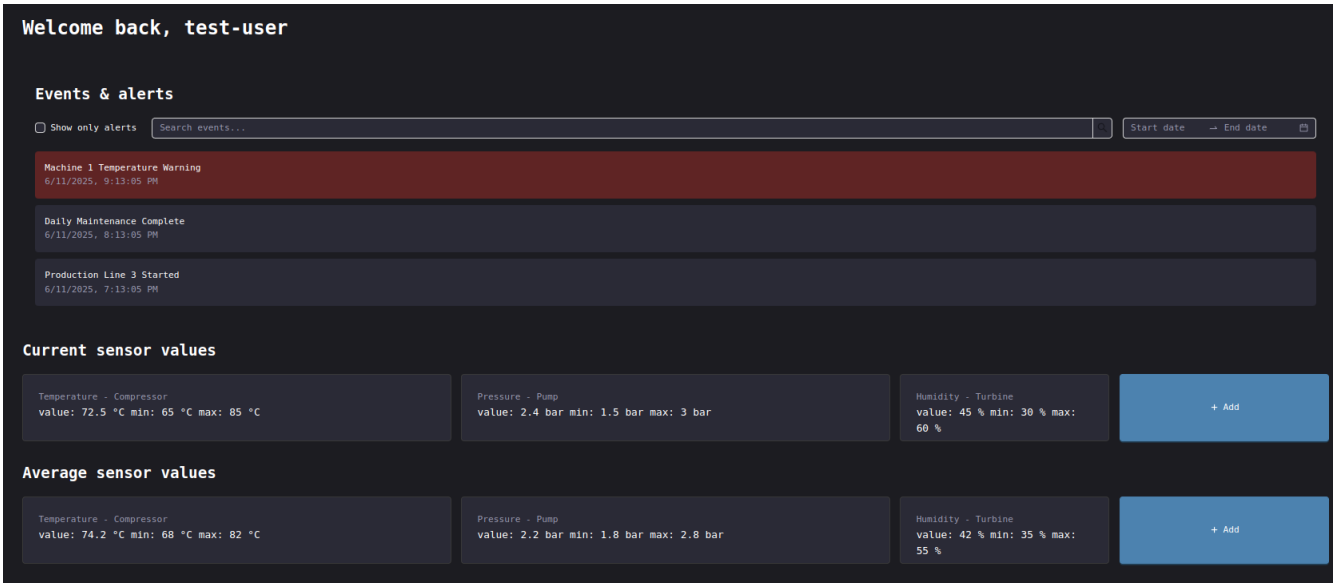
Widok szczegółowy raportu prezentuje wszystkie zdefiniowane metadane (nazwa, opis, zakres czasowy, wybrane sensory) oraz, co najważniejsze, wykresy historyczne dla każdej skonfigurowanej kombinacji sensor-etykieta w zadanym przedziale czasowym. Umożliwia to dogłębną analizę przeszłych zdarzeń i trendów. Z tego poziomu użytkownik może również przejść do edycji definicji raportu lub go usunąć.

Rysunek 4.2.3: Szczegóły raportu

4.3 Dostosowywanie interfejsu i konfiguracja

Interfejs użytkownika jest dynamicznie konfigurowany na podstawie danych pobieranych z klastra (`data-service` poprzez `front-service`). Dotyczy to m.in. listy dostępnych typów sensorów, ich etykiet, jednostek miar, opcji filtrowania czasu itp. Zapewnia to spójność i elastyczność systemu.

Choć interfejs użytkownika pozwala na edycję konfiguracji pulpitu i raportów, szczegółowe różnice w uprawnieniach między rolami `DATA_ACCESSOR` a `ADMIN` (np. kto może zapisywać zmiany) są prawdopodobnie egzekwowane po stronie klastra, poza wspomnianym już zarządzaniem użytkownikami przez administratorów.



Rysunek 4.3.1: Modyfikacja pulpitu

5 Implementacja systemu przetwarzania danych w czasie rzeczywistym

W niniejszym rozdziale przedstawiono szczegóły implementacji systemu do analizy danych w czasie rzeczywistym z systemów wieloczujnikowych, zgodnie z projektem opisanym w poprzednim rozdziale.

5.1 Wykorzystane technologie

W stworzonej autorskiej implementacji zastosowane zostały przedstawione poniżej kluczowe technologie.

5.1.1 Języki programowania

- **Java 21** - główny język programowania wykorzystany do implementacji mikroserwisów (np. `FrontService`, `DataService`),
- **Scala** - język programowania wykorzystany do implementacji aplikacji `SparkDataProcessor` służącej do przetwarzania strumieni danych za pomocą Apache Spark,
- **Python 3.9** - język wykorzystany do implementacji symulatorów czujników i skryptów pomocniczych (np. do trenowania modelu ML),
- **TypeScript** - język wykorzystany do implementacji interfejsu użytkownika.

5.1.2 Frameworki i biblioteki dla aplikacji

- **Spring Boot 3.4** - framework wykorzystany do implementacji mikroserwisów Java,
- **Apache Spark 3.x** [Streaming(2022)] - framework wykorzystany do przetwarzania strumieniowego danych (Spark Structured Streaming) oraz do zadań uczenia maszynowego (Spark ML) w aplikacji `SparkDataProcessor`,
- **Spring Cloud** - zestaw narzędzi do budowy aplikacji na chmurze,
- **React 18** - biblioteka JavaScript wykorzystana do budowy interfejsu użytkownika.

5.1.3 Bazy danych i systemy magazynowania

- **Apache Kafka 3.4** [Kafka(2022)] - platforma do przetwarzania strumieniowego,
- **PostgreSQL 15** - relacyjna baza danych,
- **Elasticsearch 8.7** - baza danych NoSQL [NoSQL(2024)] do wyszukiwania i analizy,
- **Redis 7.0** - baza danych in-memory wykorzystywana jako cache.

5.1.4 Infrastruktura i orkiestracja aplikacji

- **Kubernetes 1.26** [Kubernetes(2022)] - platforma do orkiestracji kontenerów,
- **Docker** - platforma konteneryzacji,
- **Helm 3** - menedżer pakietów dla Kubernetes,
- **AWS** (Amazon Web Services) [Amazon Web Services(2024)] - chmura obliczeniowa wykorzystana do hostowania usług zewnętrznych,
- **Terraform** - narzędzie do zarządzania infrastrukturą jako kod (IaC, ang. Infrastructure as Code) [Terraform(2024)].

5.2 Implementacja symulatora czujników

5.2.1 Architektura symulatora danych z czujników

Architektura symulatora składa się z kilku kluczowych komponentów:

- **AWS Step Functions** - usługa, która koordynuje wykonywanie funkcji Lambda, zarządzając przepływem pracy i częstotliwością generowania danych,
- **AWS Lambda** [Amazon Web Services(2024b)] - usługa bezserwerowa, która wykonuje kod symulatora,
- **AWS SNS** (Amazon Simple Notification Service) [AWS SNS(2024)] - usługa powiadomień, która działa jako punkt dystrybucji dla wygenerowanych danych,
- **AWS SQS** (Amazon Simple Queue Service) [AWS SQS(2024)] - usługa kolejek, która buforuje wiadomości przed ich przetworzeniem przez system.

5.2.2 Implementacja funkcji Lambda generujących dane z czujników

Funkcje Lambda zostały zaimplementowane w języku Python 3.9. Każda funkcja generuje dane dla określonego typu czujnika, symulując jego zachowanie w różnych warunkach pracy. Symulator uwzględnia specyfikę każdego typu czujnika i jego charakterystykę w kontekście procesu syntezy amoniaku.

Dla każdego typu czujnika zaimplementowano logikę generowania realistycznych danych, uwzględniającą:

- normalne wahania wartości w zakresie typowym dla danego parametru,
- możliwość wystąpienia anomalii z określonym prawdopodobieństwem,
- korelacje między różnymi parametrami procesu,
- symulację zakłóceń i szumów pomiarowych.

6 Zaimplementowane algorytmy analizy danych

W niniejszym rozdziale omówiono algorytmy wykorzystane do analizy danych, zaimplementowane w ramach opracowanego systemu. System ten, nazwany `SparkDataProcessor`, został zrealizowany w języku Scala z wykorzystaniem frameworka Apache Spark [Streaming(2022)]. Jego głównym zadaniem jest przetwarzanie strumieni danych sensorycznych pochodzących z symulowanych urządzeń przemysłowych, takich jak: pompy, sprężarki i turbiny, szczegółowo opisane w rozdziale 7. Aplikacja analizuje cztery podstawowe wielkości mierzone: ciśnienie, temperaturę, wilgotność oraz wibracje.

Architektura `SparkDataProcessor` opiera się na głównym obiekcie `SparkDataProcessor`, inicjalizującego i zarządzającego poszczególnymi zadaniami przetwarzania strumieniowego. Działanie aplikacji jest w pełni konfigurowalne za pomocą pliku `application.conf` w formacie HOCON (Human-Optimized Config Object Notation) [Lightbend(2024)], ładowanego przy użyciu biblioteki PureConfig [PureConfig(2024)] poprzez obiekt `AppConfig`. Konfiguracja ta pozwala na dynamiczne włączanie i wyłączanie poszczególnych modułów analitycznych oraz precyzyjne definiowanie ich parametrów, takich jak: adresy serwerów Kafka [Kafka(2022)], nazwy tematów wejściowych i wyjściowych, rozmiary okien czasowych dla agregacji, progi alarmowe dla detekcji zdarzeń, a także flagi debugowania.

Podstawą przetwarzanych informacji są modele danych zdefiniowane jako klasy przypadku (case classes) w języku Scala, znajdujące się w pakiecie `com.factory.model`. Reprezentują one odczyty z poszczególnych sensorów (np. `Pressure`, `Temperature`, `Humidity`, `Vibration`) oraz strukturę wykrytych zdarzeń (`Event`).

System `SparkDataProcessor` składa się z trzech głównych modułów przetwarzających dane:

Pierwszym z nich jest moduł do obliczania średnich wartości wielkości mierzonych przez czujniki `MeanProcessor`, odpowiedzialny za obliczanie średnich wartości dla poszczególnych typów sensorów w zdefiniowanych oknach czasowych. Przetwarzanie to odbywa się dla każdego typu urządzenia (np. "pump", "compressor", "turbine"), identyfikowanego za pomocą etykiet (labels) w danych wejściowych. Moduł ten konsumuje dane w formacie Apache Avro [Avro(2022)] z dedykowanych tematów Kafka [Kafka(2022)]. Aplikacja `MeanProcessor` wykorzystuje mechanizmy bibliotece Spark Structured Streaming, w tym funkcje okienkowania (np. `window(col("event_time"), "2 minutes", "1 minutes")`) oraz znaki wodne (watermarks) [Watermarking(2024)] do obsługi danych opóźnionych. Zagregowane wartości średnie, wraz z liczbą odczytów, są następnie serializowane z powrotem do formatu Avro i publikowane na odpowiednie tematy Kafka (np. `pressureMean`).

Następny moduł `EventProcessor`, ma za zadanie monitorowanie strumieni danych sensorycznych pod kątem przekroczenia zdefiniowanych progów ostrzegawczych i krytycznych. Analiza ta jest prowadzona indywidualnie dla każdego typu sensora i etykiety urządzenia. Ten komponent wykorzystuje zaawansowaną operację `flatMapGroupsWithState` dostępną w Spark Structured Streaming [Streaming(2022)] , która pozwala na utrzymanie stanu dla każdego klucza (czyli dla każdej etykiety sensora). Moduł ten implementuje mechanizm "cooldown", zapobiegający generowaniu nadmiernej liczby alertów dla tego samego problemu w krótkim czasie. Okres wyciszenia może być zdefiniowany globalnie lub specyficznie dla danego typu sensora. Wykryte zdarzenia, zawierające m.in. tytuł, znacznik czasowy oraz status alertu (ostrzeżenie/krytyczny), są serializowane do formatu Avro przy użyciu schematu pobranego z Confluent Schema Registry [Confluent Schema Registry(2022)] i wysyłane do wspólnego tematu Kafka o nazwie `events`.

Trzecim, najbardziej złożonym modułem, jest `EquipmentStateProcessor`. Jego celem jest predykcja ogólnego stanu technicznego monitorowanego urządzenia na podstawie połączonych danych z wielu sensorów. Moduł ten subskrybuje dane Avro [Avro(2022)] z czterech głównych tematów

Kafki odpowiadających strumieniom ciśnienia, temperatury, wilgotności i wibracji. Każdy ze strumieni jest opatrzony znakiem wodnym w celu poprawnej obsługi danych napływających z opóźnieniem. Następnie strumienie te są łączone (join) na podstawie wspólnego klucza, który, zgodnie z dostarczonymi informacjami, reprezentuje identyfikator cyklu pomiarowego (kombinacja znacznika czasowego i typu urządzenia). Pozwala to na skorelowanie odczytów z różnych sensorów dla tego samego momentu i urządzenia. Połączone dane, zawierające cechy takie jak: typ urządzenia (`equipment_type`), znacznik czasowy zdarzenia (`event_timestamp`) oraz wartości poszczególnych pomiarów, są przekazywane do wytrenowanego wcześniej modelu uczenia maszynowego. Model ten, będący klasyfikatorem `RandomForestClassifier` z biblioteki Spark ML (ang. Machine Learning Library) [Apache Spark(2024c)], został wcześniej wytrenowany oraz został ładowany do aplikacji Sparka. Na podstawie tych danych model klasyfikuje stan urządzenia. Możliwe stany, szczegółowo opisane w Rozdziale 7, obejmują: "Stan normalny", "Wczesne zużycie", "Stan podkrytyczny", "Stan krytyczny" oraz "Naprawa". Oryginalne dane sensoryczne są następnie wzbogacane o tę przewidzianą etykietę stanu (`predicted_status`). Tak przygotowane, wzbogacone dane dla każdego typu sensora (np. dane o ciśnieniu wraz z przewidywanym stanem) są serializowane do formatu Avro, z wykorzystaniem odpowiednich schematów (np. `pressureAugumented-value`) pobieranych z Confluent Schema Registry [Confluent Schema Registry(2022)] i publikowane na nowe, dedykowane tematy Kafka (np. `pressureAugumented`). Głównym zastosowaniem tych wzbogaconych danych jest ich wizualizacja na wykresach w interfejsie użytkownika.

6.1 Proces uczenia modelu predykcji stanu urządzenia

Model klasyfikacji `RandomForestClassifier`, wykorzystywany przez moduł `EquipmentStateProcessor` do predykcji stanu technicznego urządzeń, jest trenowany w osobnym procesie wsadowym przy użyciu dedykowanego skryptu Python z biblioteką PySpark (interfejs Python dla Apache Spark) [PySpark(2024)]. Proces ten można podzielić na kilka kluczowych etapów.

1. **Przygotowanie danych**

   Źródłem danych treningowych są pliki CSV generowane przez symulator. Zawierają one historyczne odczyty sensorów (temperatura, ciśnienie, wibracje, wilgotność) oraz odpowiadający im rzeczywisty stan urządzenia (np. "Stan normalny", "Wczesne zużycie", itd.). Dane dla różnych typów urządzeń są wczytywane do Spark DataFrame (rozproszona kolekcja danych zorganizowana w nazwane kolumny) [Spark DataFrame(2024)] , a następnie łączone w jeden zbiór. Przeprowadzane jest czyszczenie danych, obejmujące m.in. usunięcie rekordów ze stanem `repair`, ponieważ ten stan nie jest przedmiotem klasyfikacji w czasie rzeczywistym, lecz wynikiem działania serwisowego. Kluczowym krokiem jest konwersja kolumny znacznika czasu (np. `timestamp`) do odpowiedniego formatu Spark TimestampType i sortowanie danych chronologicznie. Cały zbiór danych jest następnie dzielony na zbiory: treningowy (70% danych), walidacyjny (15%) oraz testowy (15%). Podział ten realizowany jest w oparciu o kolejność czasową rekordów, co uzyskuje się przez dodanie kolumny z monotonicznie rosnącym ID po sortowaniu (`monotonically_increasing_id()`) i filtrowanie na podstawie progów wyliczonych z proporcji. Taki sposób podziału ma na celu lepsze symulowanie rzeczywistych warunków predykcji przyszłych stanów na podstawie danych historycznych.

2. **Inżynieria cech.**

   W tym etapie przygotowywane są cechy wejściowe dla modelu. Kolumny reprezentujące odczyty sensorów (`temperature`, `pressure`, `vibration`, `humidity`) są rzutowane na typ

`double`. Zmienna docelowa, czyli kolumna `status` opisująca stan urządzenia, jest konwertowana na wartości numeryczne przy użyciu transformatora `StringIndexer` (transformator Spark ML konwertujący kolumnę etykiet tekstowych na kolumnę indeksów liczbowych) [Spark StringIndexer(2024)], tworząc nową kolumnę `label`. Etykiety tekstowe, które `StringIndexer` mapuje na indeksy, są zapamiętywane na potrzeby późniejszej konwersji predykcji z powrotem na tekst. Następnie, wybrane cechy numeryczne są łączone w jeden wektor cech za pomocą transformatora `VectorAssembler` (transformator Spark ML łączący wiele kolumn w jedną kolumnę wektorową) [Spark VectorAssembler(2024)] , tworzący kolumnę `features`.

3. **Definicja i trening modelu.**

   Jako model klasyfikacyjny wykorzystywany jest
   `RandomForestClassifier` z biblioteki Spark MLlib [Apache Spark(2024c)], skonfigurowany m.in. z liczbą drzew decyzyjnych ustawioną na 100 (`numTrees=100`) oraz stałym ziarnem losowości (`seed=42`) dla zapewnienia reprodukowalności wyników. Cały proces transformacji danych i treningu modelu jest zdefiniowany jako potok (pipeline) Spark ML, składający się z sekwencji kroków: `StringIndexer` (dla kolumny `status`), `VectorAssembler` (dla cech wejściowych) oraz
   `RandomForestClassifier`. Model jest trenowany poprzez wywołanie metody `fit()` na tym potoku, używając przygotowanego wcześniej zbioru treningowego.

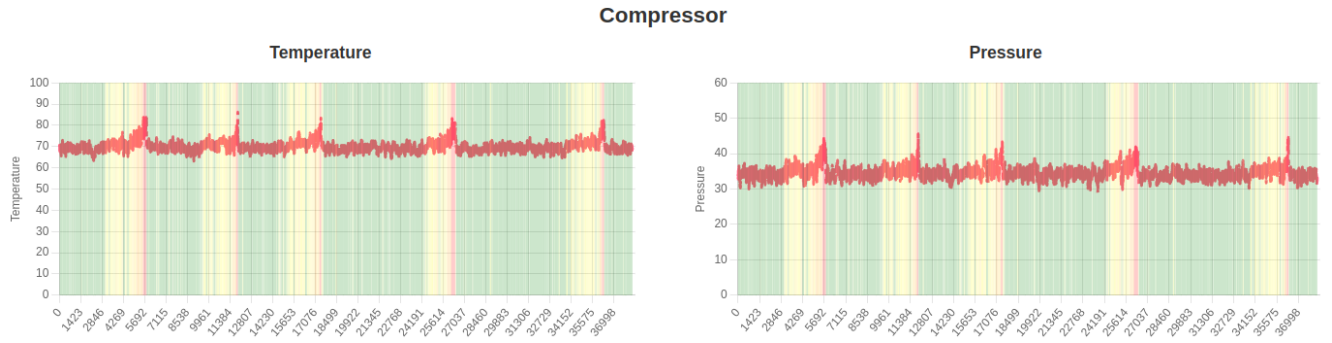
4. **Zapis i przygotowanie modelu do inferencji.**

   Po zakończeniu treningu, wytrenowany potok jest zapisywany na dysku. Skrypt treningowy generuje dwie główne struktury modelu. Pierwsza, określana jako "pełny potok deweloperski" (zapisywana w ścieżce `spark_model_path + "_dev_full_pipeline"`), zawiera wszystkie etapy, łącznie ze `StringIndexer`, i jest używana na potrzeby ewaluacji i dalszych prac deweloperskich w środowisku Python. Druga wersja, kluczowa dla aplikacji `SparkDataProcessor`, jest zapisywana bezpośrednio pod ścieżką docelową dla modelu inferencyjnego. Ten model inferencyjny jest specjalnie przygotowanym potokiem, składającego się z funkcji transformującej i agregującej wiele kolumn w jeden wektor `VectorAssembler` (z wytrenowanego pełnego potoku), wytrenowanego modelu `RandomForestClassifier` oraz, co istotne, transformatora `IndexToString`. Zadaniem `IndexToString` jest przekształcenie numerycznych predykcji modelu (kolumna `prediction`) z powrotem na etykiety tekstowe (kolumna `predicted_status`), wykorzystując mapowania (etykiety) nauczone przez `StringIndexer` podczas treningu i zapisane w pełnym potoku. Dzięki temu aplikacja Scala otrzymuje predykcje w czytelnej, tekstowej formie. Skrypt treningowy posiada również logikę umożliwiającą wczytanie wcześniej zapisanego pełnego potoku deweloperskiego, jeśli jest on dostępny, co przyspiesza iteracyjne uruchomienia i pozwala na ponowne wygenerowanie modelu inferencyjnego bez pełnego treningu.

5. **Wyniki treningu i skuteczność modelu**

   Model przetestowano wstępnie na zbiorze testowym w zadanym przedziale czasowym dla poszczególnego urządzenia (w tym przypadku kompresora), wyniki przedstawione są na rysunkach 5 oraz 5. Jak pokazano na porównaniu model wykazał się dobrym poziomem skuteczności.

Rysunek 6.1.1: Wykres pokazujący prawdziwe stany urządzenia



Rysunek 6.1.2: Dane pokazujące sklasyfikowane stany urządzenia

6. **Ewaluacja i wyniki.**

Skuteczność wytrenowanego modelu jest weryfikowana na zbiorach treningowym, walidacyjnym i testowym przy użyciu metryki dokładności (accuracy) za pomocą ewaluatora `MulticlassClassifi`

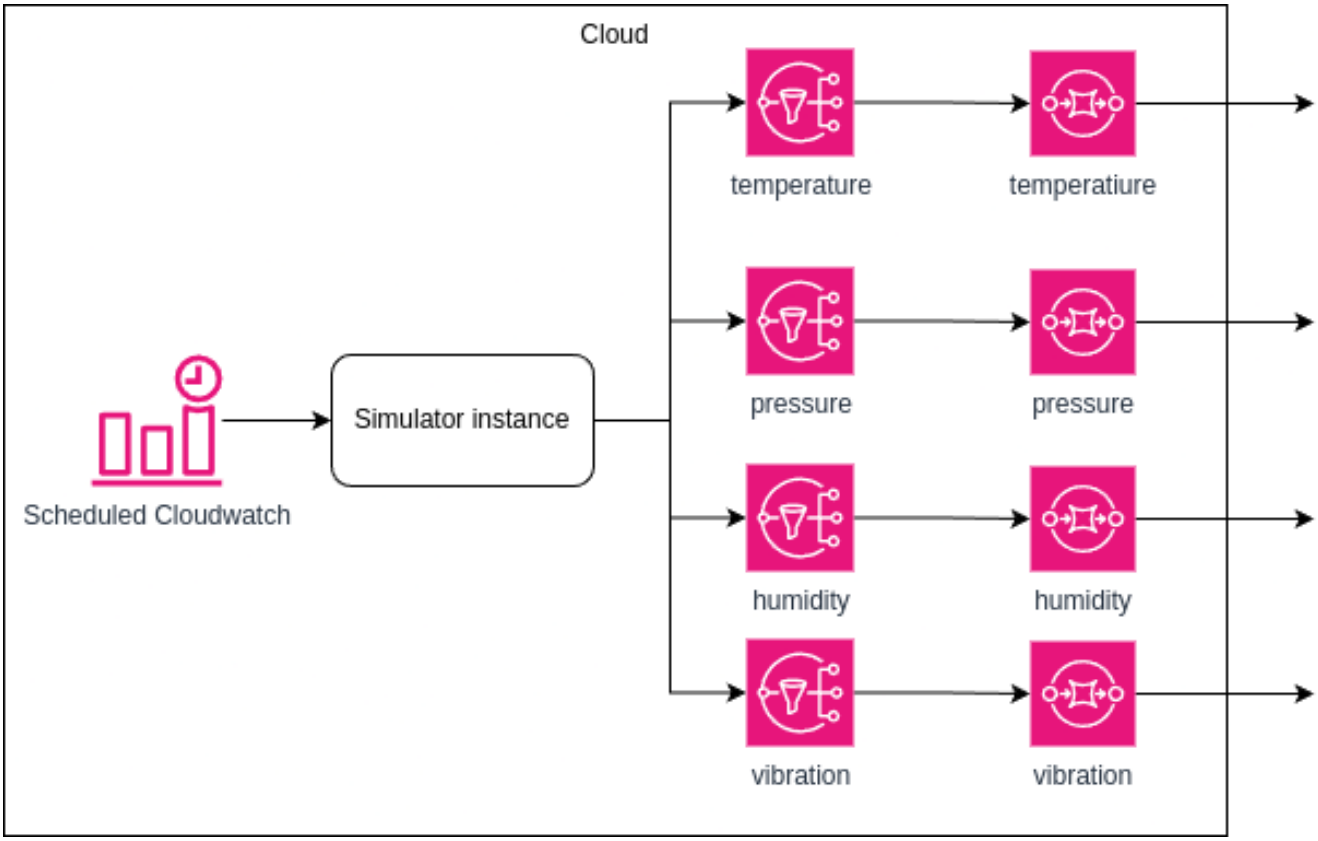
W całym systemie `SparkDataProcessor` kluczową rolę odgrywa serializacja danych do formatu Apache Avro oraz integracja z Confluent Schema Registry w celu zarządzania schematami. Elementem tej integracji, stosowanym przez wszystkie moduły przetwarzające dane Avro, jest specyficzny dla projektu mechanizm obsługi nagłówków. Przy konsumpcji danych z Kafki, przychodzące komunikaty Avro mają usuwany 6-bajtowy nagłówek (składający się z tzw. magic byte oraz ID schematu) za pomocą operacji `substring(value, 6)`. Natomiast przy wysyłaniu przetworzonych danych z powrotem do Kafki, odpowiedni nagłówek jest dodawany przed serializacją przez funkcję zdefiniowaną przez użytkownika (UDF, ang. User-Defined Function) [Spark UDF(2024)] o nazwie `addSchemaRegistryHeaderUDF`. Takie podejście jest niestandardowym obejściem, koniecznym dla zapewnienia poprawnej współpracy aplikacji Spark Structured Streaming ze schematami Avro zarządzanymi przez Confluent Schema Registry [Confluent Schema Registry(2022)] w ramach tego projektu. Aplikacja korzysta również ze standardowych funkcji Apache Spark, takich jak konfiguracja lokalizacji punktów kontrolnych (`spark.sql.streaming.checkpointLocation`) dla zapewnienia odporności na błędy w przetwarzaniu strumieniowym oraz możliwości ustawiania poziomu logowania (np. `spark.sparkContext.setLogLevel("WARN")`). Dla celów diagnostycznych, niektóre procesory mogą, w zależności od konfiguracji, wypisywać przetwarzane dane również na konsolę. Źródłem danych wejściowych dla systemu są komunikaty Kafka generowane przez symulator (opisany w rozdziale 7), naśladujący pracę rzeczywistych urządzeń przemysłowych.

7 Generator danych czujników w chmurze

W celu przeprowadzenia kompleksowych testów systemu analizy danych w czasie rzeczywistym oraz zapewnienia ciągłego strumienia danych testowych, opracowano dedykowany generator danych w chmurze. Generator ten symuluje pracę rzeczywistych urządzeń przemysłowych, generując realistyczne dane sensoryczne z uwzględnieniem korelacji między parametrami oraz ewolucji stanów technicznych urządzeń.

System generatora został zbudowany w oparciu o architekturę bezserwerową (serverless) w chmurze AWS z wykorzystaniem funkcji AWS Lambda. Generator produkuje dane dla trzech typów urządzeń przemysłowych: pomp, sprężarek i turbin, symulując cztery rodzaje parametrów sensorycznych: temperaturę, ciśnienie, wibracje oraz wilgotność.

Wartości symulowanych danych są generowane w oparciu o dostarczony wcześniej zbiór z danymi z prawdziwych urządzeń przemysłowych. Na tej podstawie generowane są przebiegi czasowe dla każdego z czterech parametrów sensorycznych.



Rysunek 7.0.1: Schemat przepływu danych w generatorze

7.1 Przygotowanie danych wstępnych do generatora

Proces przygotowania danych do generatora rozpoczyna się od analizy rzeczywistego przemysłowego zbioru danych za pomocą skryptu opartego na Apache Spark. Zbiór wejściowy zawiera dane sensoryczne z trzech typów urządzeń przemysłowych (pomp, sprężarek i turbin) rozmieszczonych w różnych lokalizacjach fabrycznych.

7.1.1 Struktura danych wejściowych

Dane wejściowe składają się z następujących kolumn:

- **temperature** - temperatura urządzenia w stopniach Celsjusza,
- **pressure** - ciśnienie w barach,
- **vibration** - poziom wibracji w jednostkach przyspieszenia,
- **humidity** - wilgotność względna w procentach,
- **equipment** - typ urządzenia (Pump, Compressor, Turbine),
- **location** - lokalizacja fabryczna (Atlanta, Chicago, San Francisco, New York, Houston),
- **faulty** - wskaźnik stanu awaryjnego (0.0 = normalny, 1.0 = awaryjny).

7.1.2 Proces analizy statystycznej

Skrypt wykonuje wieloetapową analizę danych w następujących krokach:

- **Analiza podstawowych statystyk** - obliczane są podstawowe metryki statystyczne dla każdego typu urządzenia i stanu technicznego:
  - średnie wartości,
  - odchylenia standardowe,
  - wartości minimalne i maksymalne,
  - liczebności obserwacji.

  Wyniki są grupowane według kombinacji typu urządzenia i stanu awaryjnego, co pozwala na wyznaczenie charakterystycznych zakresów parametrów dla stanów normalnych i awaryjnych każdego typu urządzenia.
- **Analiza korelacji między sensorami** - skrypt generuje macierze korelacji Pearsona dla różnych kombinacji:
  - **korelacja ogólna** - dla wszystkich danych,
  - **korelacja specyficzna dla urządzeń** - oddzielnie dla każdego typu urządzenia,
  - **korelacja specyficzna dla stanów** - oddzielnie dla stanów normalnych i awaryjnych,
  - **korelacja kombinowana** - dla każdej kombinacji typu urządzenia i stanu.

  Dodatkowo obliczana jest macierz zmian korelacji:

  $$\Delta \mathbf{R} = \mathbf{R}_{faulty} - \mathbf{R}_{normal},$$

  pokazująca, jak korelacje między parametrami zmieniają się podczas przejścia ze stanu normalnego do awaryjnego.
- **Analiza głównych składowych (PCA)** - wykonywana jest analiza PCA w celu:

– identyfikacji najważniejszych czynników wpływających na stan urządzeń,
– określenia wkładu każdego parametru sensorycznego w główne składowe,
– redukcji wymiarowości danych przy zachowaniu maksymalnej wariancji,
– wyznaczenia ważności cech (*feature importance*) na podstawie ładunków składowych.

7.1.3 Generowane pliki konfiguracyjne

Wyniki analizy są zapisywane w katalogu `analysis_output` w postaci plików wykorzystywanych przez generator:

Pliki statystyk

- statystyki podstawowe dla każdego typu urządzenia,
- statystyki dla stanów normalnych i awaryjnych,
- statystyki dla kombinacji urządzenie-stan.

Pliki korelacji

- macierz korelacji ogólnej,
- macierze specyficzne dla urządzeń i stanów,
- macierz zmian korelacji,
- słownik korelacji w formacie JSON.

Powyższe pliki konfiguracyjne stanowią bazę dla generatora danych, zapewniając, że symulowane dane zachowują realistyczne charakterystyki statystyczne i korelacyjne obserwowane w rzeczywistych danych przemysłowych.

Symulator wykorzystuje wielowymiarowe rozkłady normalne do generowania skorelowanych danych sensorycznych, zachowując realistyczne relacje między parametrami. Przejścia między stanami są modelowane probabilistycznie z określonymi czasami trwania dla każdego stanu.

7.2 Model matematyczny korelacji

Generator wykorzystuje macierze korelacji wyznaczone na podstawie analizy danych historycznych. System automatycznie ładuje macierze korelacji dla różnych kombinacji typów urządzeń i stanów technicznych z plików konfiguracyjnych wygenerowanych przez skrypt analizy.

Dla każdego typu urządzenia i i stanu s dostępne są dedykowane macierze korelacji $\mathbf{R}_{i,s}$:

$$\mathbf{R}_{i,s} = \begin{pmatrix} 1 & r_{12} & r_{13} & r_{14} \\ r_{12} & 1 & r_{23} & r_{24} \\ r_{13} & r_{23} & 1 & r_{34} \\ r_{14} & r_{24} & r_{34} & 1 \end{pmatrix},$$

gdzie r_{jk} oznacza współczynnik korelacji między parametrami j i k.

Generator implementuje adaptacyjny wybór macierzy korelacji na podstawie aktualnego stanu urządzenia:

- dla stanu normalnego: $\mathbf{R}_{i,normal}$,
- dla stanów degradacji: progresywne przejście od $\mathbf{R}_{i,normal}$ do $\mathbf{R}_{i,faulty}$,
- dla stanu krytycznego: $\mathbf{R}_{i,faulty}$,
- dla stanu naprawy: brak generowania danych sensorycznych.

7.3 Algorytm generowania skorelowanych wartości

System implementuje zaawansowany algorytm generowania skorelowanych wartości sensorycznych wykorzystujący wielowymiarowy rozkład normalny z adaptacyjnym mieszaniem statystyk.

7.3.1 Mieszanie statystyk między stanami

Dla każdego parametru sensorowego obliczane są mieszane statystyki na podstawie poziomu uszkodzenia $f \in [0, 1]$:

$$\mu_{mieszane}(f) = \mu_{normal} + f \cdot (\mu_{faulty} - \mu_{normal}),$$

$$\sigma_{mieszane}(f) = \sigma_{normal} + f \cdot (\sigma_{faulty} - \sigma_{normal}),$$

gdzie poziom uszkodzenia f jest funkcją stanu i postępu w stanie:

$$f(stan, postp) = \begin{cases} 0.0 & \text{dla stanu normalnego} \\ 0.2 + 0.2 \cdot postp & \text{dla wczesnego zużycia} \\ 0.4 + 0.3 \cdot postp & \text{dla stanu podkrytycznego} \\ 0.7 + 0.3 \cdot postp & \text{dla stanu krytycznego} \\ 1.0 & \text{dla stanu naprawy} \end{cases},$$

7.3.2 Komponenty cykliczne

System uwzględnia naturalne cykle operacyjne poprzez dodanie komponentów cyklicznych:

$$\mu_{cykliczne} = \mu_{mieszane} + A \cdot \sin\left(\frac{2\pi t}{T}\right),$$

gdzie:

- $A = 0.2 \cdot \sigma_{normal}$ - amplituda cyklu dobowego,
- $T = 24$ godziny - okres cyklu,
- t - aktualny czas w godzinach.

7.3.3 Generowanie wartości z rozkładu wielowymiarowego

Skorelowane wartości sensoryczne są generowane z wielowymiarowego rozkładu normalnego:

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_{mieszane}, \boldsymbol{\Sigma})$$

gdzie macierz kowariancji obliczana jest jako:

$$\boldsymbol{\Sigma} = \mathbf{D}\mathbf{R}_{i,s}\mathbf{D}$$

gdzie $\mathbf{D} = \text{diag}(\sigma_1, \sigma_2, \sigma_3, \sigma_4)$ to macierz diagonalna odchyleń standardowych.

7.4 Wygładzanie czasowe i ograniczenia zmian

Celemem zapewnienia realistycznego przejścia między kolejnymi odczytami, system implementuje dwuetapowe wygładzanie składające się z następujących etapów:

7.4.1 Ograniczenie szybkości zmian

Maksymalna zmiana między kolejnymi odczytami jest ograniczona do:

$$\Delta x_{max} = \sigma_{normal} \cdot r_{max} \cdot k_{stan}$$

gdzie:

- $r_{max} = 0.2$ - maksymalny współczynnik zmiany,
- k_{stan} - czynnik stanu (1.0 dla normalnego, 1.5 dla podkrytycznego, 2.0 dla krytycznego).

Rzeczywista zmiana wartości danej wielkości pochodzącej z czujnika jest ograniczana zgodnie z:

$$\Delta x_{limited} = \begin{cases} \Delta x & \text{jeśli } |\Delta x| \leqslant \Delta x_{max} \\ \Delta x_{max} \cdot \text{sign}(\Delta x) & \text{w przeciwnym przypadku} \end{cases}$$

7.4.2 Wygładzanie wykładnicze

Po przeprowadzeniu ograniczenia szybkości zmian, w kolejnym etapie stosowane jest wygładzanie wykładnicze:

$$x_{t+1} = x_t + (1 - \alpha) \cdot \Delta x_{limited}$$

gdzie $\alpha = 0.8$ to współczynnik wygładzania.

7.5 Model stanów urządzeń i przejść

System implementuje deterministyczny model maszyny stanów z probabilistycznymi czasami trwania.

7.5.1 Stany techniczne urządzeń

Generator danych modeluje następujące stany techniczne:

- **stan normalny** - prawidłowa praca urządzenia,
- **wczesne zużycie** - początkowe oznaki degradacji,
- **stan podkrytyczny** - znaczące pogorszenie parametrów,
- **stan krytyczny** - stan przed awarią,
- **naprawa** - okres nieaktywności urządzenia.

7.5.2 Parametry czasowe stanów

Każdy stan techniczny charakteryzuje się losowym czasem trwania z zakresu $[t_{min}, t_{max}]$:

Stan	Min. czas	Max. czas	Następny stan
normalny	24h	7 dni	wczesne zużycie
wczesne zużycie	12h	48h	podkrytyczny
podkrytyczny	6h	24h	krytyczny
krytyczny	1h	6h	naprawa
naprawa	2h	4h	normalny

Tabela 1: Parametry czasowe stanów urządzeń w symulatorze

7.5.3 Deterministyczne przejścia stanów

System wykorzystuje deterministyczne generowanie czasów trwania stanów na podstawie funkcji *hash*:

$$t_{stan} = t_{min} + \text{hash}(typ\_urzadzenia + stan + timestamp) \bmod (t_{max} - t_{min})$$

Zapewnia to powtarzalność generowanych danych przy wykorzystaniu tego samego ziarna losowego.

7.6 Algorytm generowania danych historycznych

Generator implementuje algorytm generowania kompletnych szeregów czasowych danych historycznych z minutową rozdzielczością.

7.6.1 Inicjalizacja deterministyczna

Każdy typ urządzenia inicjalizowany jest z deterministycznym stanem początkowym:

$$stan_{poczatkowy} = \begin{cases} \text{normalny} & \text{jeśli } h(typ) \bmod 1.0 < 0.5 \\ \text{wczesne zużycie} & \text{jeśli } 0.5 \leqslant h(typ) \bmod 1.0 < 0.8 \\ \text{podkrytyczny} & \text{w przeciwnym przypadku} \end{cases}$$

gdzie $h(typ)$ to funkcja hash typu urządzenia.

7.6.2 Iteracyjne generowanie danych

Dla każdego kroku czasowego t (minuty):

1. aktualizacja czasu w stanie: $t_{stan} = t_{stan} + \Delta t$,
2. obliczenie postępu: $progress = \min(1.0, t_{stan}/t_{trwania})$,
3. sprawdzenie konieczności przejścia stanu,
4. generowanie wartości docelowych z rozkładu wielowymiarowego,
5. zastosowanie wygładzania czasowego,
6. zapis danych do pliku CSV.

7.6.3 Obsługa stanu naprawy

Podczas stanu naprawy system:

- nie generuje wartości sensorycznych (wartości NULL),
- zachowuje informacje o stanie i czasie,
- po zakończeniu naprawy resetuje wartości do normalnych.

7.6.4 Integracja z chmurą AWS

System generatora danych został zaimplementowany jako w pełni bezserwerowa architektura w chmurze AWS, wykorzystująca usługi AWS Lambda, Amazon S3 oraz Amazon SNS. Implementacja zapewnia skalowalność, niezawodność oraz efektywność kosztową przy minimalnych wymaganiach administracyjnych.

Generator czyta wcześniej wygenerowane pliki CSV z danymi z S3 i generuje dane w odstępach minutowych.

7.6.5 Architektura bezserwerowa

System składa się z trzech głównych komponentów AWS:

- **Amazon S3** - przechowywanie danych historycznych i metadanych konfiguracyjnych,
- **AWS Lambda** - funkcja wykonująca logikę publikowania danych w czasie rzeczywistym,
- **Amazon SNS** - tematy publikacyjno-subskrypcyjne dla każdego typu sensora.

7.6.6 Przechowywanie danych w Amazon S3

System wykorzystuje Amazon S3 jako centralne repozytorium dla danych historycznych wygenerowanych przez symulator lokalny.

7.6.7 Struktura danych w S3

Dane są organizowane w następującej strukturze:

- **prefiks danych**: `prod_data/`,
- **pliki CSV**: `pump_YYYYMMDD.csv`, `compressor_YYYYMMDD.csv`, `turbine_YYYYMMDD.csv`,
- **metadane**: `prod_data/historical_metadata.json`.

Poniższy plik reprezentuje format metadanych zawierający konfigurację systemu w formacie JSON:

```
{
  "generated_at": "2023-01-15 14:30:00",
  "start_timestamp": 1672531200,
  "end_timestamp": 1675123199,
  "minute_resolution": 1,
  "equipment_types": ["Pump", "Compressor", "Turbine"],
  "sensor_types": ["temperature", "pressure", "vibration", "humidity"],
  "seed": 42,
  "files": {
    "Pump": "pump_20230101.csv",
    "Compressor": "compressor_20230101.csv",
    "Turbine": "turbine_20230101.csv"
  }
}
```

Listing 3: Plik metadanych

7.6.8 Funkcja AWS Lambda

Główna logika systemu jest zaimplementowana jako funkcja AWS Lambda napisana w języku Python z wykorzystaniem bibliotek: boto3, pandas oraz pydantic. System wykorzystuje bibliotekę Pydantic do walidacji i serializacji danych.

```
class SensorData(BaseModel):
    label: str
    timestamp: int
    event_key: str

class Temperature(SensorData):
    temperature: TemperatureData

class Pressure(SensorData):
    pressure: PressureData
```

Listing 4: Klasy modeli danych

Algorytm publikowania danych - funkcja AWS Lambda wykonuje następujący algorytm przy każdym wywołaniu. Funkcja implementuje algorytm wyszukiwania najbliższego czasowo rekordu. Składa się z następujących kroków:

1. **ładowanie metadanych** z S3 w celu określenia dostępnych plików danych
2. **ładowanie danych** dla każdego typu urządzenia z plików CSV
3. **wyznaczenie aktualnego czasu** z zaokrągleniem do pełnej minuty
4. **wyszukanie najbliższego rekordu** w danych historycznych dla każdego urządzenia
5. **generowanie unikalnych kluczy** zdarzeń dla każdego urządzenia
6. **publikowanie danych** do odpowiednich tematów SNS dla każdego sensora

```
def _find_closest_record(self, df, target_time):
    target_time_dt = pd.to_datetime(target_time)
    df['time_diff'] = abs(df['timestamp'] - target_time_dt)
    closest_record = df.loc[df['time_diff'].idxmin()]
    return closest_record
```

Listing 5: Algorytm wyszukiwania najbliższego rekordu

7.6.9 Amazon SNS dla publikowania danych

System wykorzystuje system transportowania danych Amazon Simple Notification Service (SNS) do asynchronicznego publikowania danych sensorycznych w architekturze publish-subscribe. Każda wiadomość jest publikowana w formacie JSON zgodnym z modelem Pydantic.

Dla każdego typu sensora utworzony jest dedykowany temat SNS:

- `arn:aws:sns:eu-central-1:accountid:temperature-topic`,
- `arn:aws:sns:eu-central-1:accountid:pressure-topic`,
- `arn:aws:sns:eu-central-1:accountid:vibration-topic`,
- `arn:aws:sns:eu-central-1:accountid:humidity-topic`.

Format publikowanych wiadomości do systemu SNS:

```
{
  "label": "pump",
  "timestamp": 1672531200,
  "event_key": "pump-1672531200",
  "temperature": {
    "temperature": 72.5
  }
}
```

Listing 6: Format publikowanych wiadomości

System ten stanowi kluczowy element infrastruktury testowej, umożliwiając kompleksową walidację opracowanego systemu analizy danych bez konieczności dostępu do rzeczywistych instalacji przemysłowych oraz zapewniając pełną integrację z ekosystemem chmury AWS.

8 Konfiguracja i orkiestracja klastra Kubernetes

System jest wdrożony na dwuwęzłowym klastrze Kubernetes, składającym się z węzła głównego (master) oraz węzła roboczego (worker). Klaster został skonfigurowany w celu zapewnienia wysokiej dostępności, skalowalności oraz bezpieczeństwa. W niniejszym rozdziale przedstawiono kluczowe aspekty tej konfiguracji, koncentrując się na infrastrukturze technicznej, mechanizmach komunikacji oraz wykorzystanych narzędziach i pluginach.

Wszystkie aplikacje oraz usługi systemowe są wdrażane w dedykowanej przestrzeni nazw (namespace) o nazwie `factory`. Zarządzanie manifestami Kubernetes oraz procesem wdrożenia poszczególnych komponentów jest zautomatyzowane przy użyciu narzędzia Helm. Wykorzystano wspólny, reużywalny szablon Helm (common chart), zlokalizowany w repozytorium projektu, do standaryzacji definicji zasobów Kubernetes. Ten wspólny chart dostarcza szablonów dla większości typów zasobów, w tym `Deployment`, `StatefulSet`, `Service`, `ConfigMap`, `Ingress`, `PersistentVolume`, `PersistentVolumeClaim`, `ServiceAccount`, `Role`, `RoleBinding`, `ClusterRole`, `ClusterRoleBinding`, `Job` oraz `HorizontalPodAutoscaler`, co znacząco upraszcza zarządzanie i zapewnia spójność konfiguracji w całym systemie.

8.1 Architektura sieciowa i routing

Routing ruchu zewnętrznego do usług działających w klastrze jest realizowany za pomocą kontrolera Nginx Ingress. Kontroler ten jest skonfigurowany do obsługi zarówno ruchu HTTP/HTTPS, jak i do przekazywania ruchu TCP dla specyficznych usług, takich jak: Kafka oraz PostgreSQL. Wszystkie usługi zewnętrzne są udostępniane przez ten sam kontroler Nginx Ingress, działający jako LoadBalancer, a jego zewnętrzny adres IP jest punktem wejścia do klastra.

Definicje Ingress dla poszczególnych usług (np. `front-service.local`, `cluster-ui.local`, `keycloak.local`, `elasticsearch.local`, `schema-registry.local`) wykorzystują lokalne nazwy hostów.

Komunikacja wewnętrzna między serwisami w klastrze odbywa się głównie za pomocą standardowych usług Kubernetes typu `ClusterIP`. Dla usług stanowych, takich jak: Kafka i Zookeeper, wykorzystywane są usługi typu `ClusterIP` z ustawieniem `clusterIP: None` (headless services), co pozwala na bezpośrednią komunikację z poszczególnymi podami.

8.2 Zarządzanie konfiguracją aplikacji

Konfiguracja poszczególnych aplikacji jest zarządzana za pomocą zasobów `ConfigMap`. Każda usługa (np. `front-service`, `data-service`, `kafka-db-forwarder`) posiada dedykowany `ConfigMap`, który zawiera plik `application.yaml` ze specyficznymi ustawieniami Spring Boot, takimi jak adresy URL usług zależnych, konfiguracja połączeń z bazą danych, brokerem Kafki, czy serwerem autoryzacyjnym Keycloak.

Przykładem obiektu mapy konfiguracyjnej (ang. *ConfigMap*) `front-service-config`, który definiuje trasy dla bramy API (Spring Cloud Gateway), wskazując na wewnętrzne usługi takie jak: `data-service-svc` czy `users-service-svc`. Podobnie, `kafka-data-processor-config` zawiera szczegółową konfigurację strumieni Kafka Streams, w tym definicje okien czasowych, tematów wejściowych i wyjściowych oraz progów dla generowania zdarzeń.

8.3 Bezpieczeństwo

Bezpieczeństwo klastra jest wzmocnione przez przedstawione poniżej mechanizmy:

8.3.1 Zarządzanie certyfikatami TLS

Do automatycznego zarządzania certyfikatami TLS w klastrze wykorzystywany jest `cert-manager`, instalowany przy pomocy dostarczonych plików `values.yaml`. Zdefiniowane zasoby Ingress dla usług takich jak: `front-service`, `cluster-ui` oraz `keycloak` zawierają sekcje `tls`, które wskazują na sekrety Kubernetes (np. `front-service-tls`, `cluster-ui-tls`) przechowujące certyfikaty i klucze prywatne, zarządzane przez `cert-manager`.

8.3.2 Kontrola dostępu oparta na rolach (RBAC)

System wykorzystuje mechanizmy RBAC (Role-Based Access Control) do definiowania uprawnień dla poszczególnych komponentów. Przykładem jest `front-service`, dla którego zdefiniowano `ServiceAccount` (`front-service-sa`), `Role` (`front-service-role`) oraz `RoleBinding` (`front-service-rolebinding`). Rola ta nadaje uprawnienia do odczytu i listowania podów oraz deploymentów, a także do tworzenia podów i ich wykonywania (pods/exec), co jest wykorzystywane w zadaniu (Job) `add-keycloak-host`.

Operator Spark również posiada własną konfigurację RBAC, definiującą uprawnienia niezbędne do zarządzania aplikacjami Spark w przestrzeniach nazw `factory` oraz `default`.

8.3.3 Serwer autoryzacyjny Keycloak

Keycloak jest wdrożony jako centralny serwer uwierzytelniania i autoryzacji. Jego konfiguracja obejmuje połączenie z bazą danych PostgreSQL (`database-svc`) oraz dane logowania administratora. Dostęp do interfejsu Keycloak jest zapewniony poprzez Ingress na adresie `keycloak.local`.

8.4 Platforma danych

Kluczowe komponenty platformy danych to broker Kafki, bazy danych Elasticsearch oraz PostgreSQL. Wszystkie te usługi wykorzystują mechanizmy przechowywania danych bezpośrednio na węzłach klastra.

8.4.1 Broker Apache Kafka i serwer Schema Registry

Klaster Kafka składa się z trzech brokerów oraz instancji Zookeepera. Każdy broker oraz Zookeeper są wdrażane jako `StatefulSet` i wykorzystują wolumeny typu `hostPath` do przechowywania danych bezpośrednio na fizycznych węzłach klastra: Zookeeper na węźle master, a brokery Kafka częściowo na węźle roboczym i masterze. Taka konfiguracja przypina konkretne instancje do fizycznych maszyn. Komunikacja z brokerami odbywa się poprzez dedykowane usługi `ClusterIP` (headless dla komunikacji wewnętrznej i standardowe dla ruchu zewnętrznego poprzez Nginx Ingress z mapowaniem portów TCP).

Confluent Schema Registry jest również wdrożony i połączony z Zookeeperem oraz brokerami Kafka, umożliwiając zarządzanie schematami Avro. Dostęp do Schema Registry jest zapewniony przez Ingress na adresie `schema-registry.local`. Po uruchomieniu klastra Kafka, uruchamiane jest zadanie (Job) `create-topics`, które automatycznie tworzy predefiniowaną listę tematów Kafka z odpowiednią liczbą partycji i współczynnikiem replikacji.

8.4.2 Elasticsearch

Elasticsearch jest wdrażany jako `StatefulSet` z jedną repliką, skonfigurowany do pracy w trybie `single-node`. Dane Elasticsearch są przechowywane na wolumenie trwałym (PersistentVolume) typu `local-storage`, zmapowanym do konkretnej ścieżki na węźle roboczym. Dostęp do Elasticsearch jest możliwy poprzez usługę `ClusterIP` oraz Ingress na adresie `elasticsearch.local`.

8.4.3 PostgreSQL

Baza danych PostgreSQL (wersja 16) jest wdrażana jako `StatefulSet`. Podobnie jak Elasticsearch, wykorzystuje wolumen trwały typu `local-storage` zmapowany do ścieżki na węźle roboczym do przechowywania danych. Konfiguracja bazy, w tym dane logowania, jest dostarczana poprzez `ConfigMap` o nazwie `db-credentials`. Baza danych jest wykorzystywana przez Keycloak oraz inne usługi aplikacyjne (np. `data-service`, `users-service`, `kafka-db-forwarder`). Dostęp do bazy danych wewnątrz klastra zapewnia usługa `database-svc`, a ruch zewnętrzny jest możliwy dzięki mapowaniu portu TCP w konfiguracji Nginx Ingress.

8.5 Przetwarzanie danych i logika aplikacyjna

8.5.1 Spark Operator

Do zarządzania i uruchamiania zadań Apache Spark w klastrze wykorzystywany jest Spark Operator. Jego konfiguracja obejmuje definicję kontrolera, webhooka oraz odpowiednich ról RBAC. Operator monitoruje zasoby `SparkApplication` i zarządza cyklem życia sterowników (driver) i wykonawców (executor) Spark.

Zdefiniowany jest zasób `SparkApplication` o nazwie `spark-data-processor`, który uruchamia aplikację Scala (`com.factory.SparkDataProcessor`) z obrazu Dockera. Aplikacja ta działa w trybie `cluster` i wykorzystuje zasoby zdefiniowane dla sterownika i dwóch wykonawców (executorów).

8.5.2 Usługi aplikacyjne

Pozostałe usługi, takie jak: `front-service` (brama API), `cluster-ui` (interfejs użytkownika React), `data-service`, `users-service`, `kafka-db-forwarder` oraz `sqs-kafka-forwarder` są wdrażane jako standardowe Deploymenty Kubernetes. Każda z nich posiada własną konfigurację (ConfigMap), usługę (Service) oraz, w przypadku usług frontendowych i bramy API, definicję Ingress.

Usługa `sqs-kafka-forwarder` jest skonfigurowana do integracji z rzeczywistymi kolejkami Amazon SQS i przekazywania wiadomości do odpowiednich tematów Kafka. Konfiguracja zawiera klucze dostępowe AWS, co potwierdza bezpośrednią integrację z infrastrukturą AWS.

Dodatkowym elementem jest zadanie (Job) `add-keycloak-host` powiązane z `front-service`. Zadanie to, po udanym wdrożeniu `front-service`, modyfikuje plik `/etc/hosts` w podach `front-service`, dodając wpis mapujący `keycloak.local` na adres IP Ingressa. Jest to obejście problemów z rozwiązywaniem nazw DNS dla Keycloak z poziomu tej usługi, zapewniając bezpośrednią komunikację po adresie IP kontrolera Ingress.

8.6 Zarządzanie zasobami i sondy

Wiele wdrożeń (Deployments) i StatefulSetów definiuje żądania (requests) i limity (limits) zasobów CPU i pamięci, co jest dobrą praktyką zapewniającą stabilność klastra. Przykładowo, `front-service` ma zdefiniowane limity na 500m CPU i 512Mi pamięci, a `keycloak` na 1000m CPU i 1Gi pamięci.

Dodatkowo, dla kluczowych usług takich jak: `front-service`, `cluster-ui` oraz `keycloak`, zdefiniowane są sondy żywotności (livenessProbe) i gotowości (readinessProbe). Sondy te pozwalają klastrowi Kubernetes na monitorowanie stanu aplikacji i automatyczne restartowanie kontenerów lub kierowanie ruchu tylko do zdrowych instancji.

9 System autoryzacji i zarządzania użytkownikami

System analizy danych w czasie rzeczywistym wykorzystuje nowoczesny mechanizm autoryzacji oparty na tokenach JWT (JSON Web Token) z wykorzystaniem serwera Keycloak jako dostawcy tożsamości. Takie rozwiązanie zapewnia bezpieczny dostęp do zasobów systemu oraz umożliwia skalowalne zarządzanie użytkownikami i rolami.

9.1 Architektura systemu autoryzacji

Keycloak pełni rolę centralnego serwera uwierzytelnienia, który zarządza procesami logowania, wydawania tokenów oraz kontroli dostępu. Aplikacja frontendowa komunikuje się bezpośrednio z Keycloak w celu uzyskania tokenów dostępu, które następnie są wykorzystywane do autoryzacji żądań kierowanych do usług backendowych poprzez FrontService - bramę API działającą w architekturze mikroserwisowej.

FrontService stanowi centralny punkt dostępu do całego klastra mikroserwisów, implementując jednolity system autoryzacji dla wszystkich usług backendowych. Dzięki tej architekturze, każdy mikroserwis w klastrze może polegać na walidacji tokenów przeprowadzonej przez bramę, co upraszcza implementację autoryzacji w poszczególnych serwisach oraz zapewnia spójność mechanizmów bezpieczeństwa w całym systemie.

Backend aplikacji został skonfigurowany do pracy w trybie hybrydowym - może działać zarówno z Keycloak jako zewnętrznym dostawcą uwierzytelnienia, jak i z własnym wewnętrznym systemem JWT. Konfiguracja ta jest kontrolowana przez parametr `app.security.useKeycloak`, co zapewnia elastyczność wdrożenia aplikacji w różnych środowiskach.

9.2 Implementacja po stronie frontendu

Aplikacja frontendowa wykorzystuje bibliotekę `keycloak-js` do integracji z serwerem Keycloak. Konfiguracja połączenia jest realizowana przez zmienne środowiskowe określające URL serwera, nazwę grupy oraz identyfikator klienta. System automatycznie przekierowuje użytkowników do strony logowania Keycloak w przypadku braku ważnego tokenu dostępu.

Po pomyślnym uwierzytelnieniu, aplikacja automatycznie dołącza token Bearer do nagłówka `Authorization` wszystkich żądań HTTP kierowanych do API. Implementacja obejmuje mechanizm automatycznego odświeżania tokenów, który sprawdza ważność tokenu przed każdym żądaniem i w razie potrzeby odnawia go w sposób przezroczysty dla użytkownika.

Interface `KeycloakInterface` zapewnia zunifikowane API do zarządzania sesją użytkownika, sprawdzania ról oraz wykonywania operacji logowania i wylogowania. Implementacja ta abstrahuje szczegóły komunikacji z Keycloak, ułatwiając integrację z komponentami React.

9.3 Autoryzacja po stronie backendu

FrontService implementuje wielowarstwowy system autoryzacji wykorzystujący Spring Security WebFlux. W trybie Keycloak, system wykorzystuje mechanizm OAuth2 Resource Server do walidacji tokenów JWT wydanych przez serwer uwierzytelnienia. Tokeny są weryfikowane pod kątem poprawności podpisu, ważności czasowej oraz uprawnień użytkownika.

Ekstractor uprawnień `KeycloakGrantedAuthoritiesConverter` analizuje strukturę tokenu JWT w celu pobrania ról zarówno z poziomu grupy jak i klienta. Umożliwia to precyzyjne mapowanie

uprawnień Keycloak na role używane wewnętrznie przez aplikację. System rozpoznaje hierarchię uprawnień, gdzie role grupy mają wyższy priorytet niż role specyficzne dla klienta.

9.4 Model uprawnień

System definiuje dwie podstawowe role użytkowników: `DATA_ACCESSOR` oraz `ADMIN`. Użytkownicy z rolą `DATA_ACCESSOR` mają dostęp do funkcji odczytu i analizy danych, obejmujących przeglądanie raportów, dashboardów oraz konfiguracji systemu. Rola `ADMIN` rozszerza te uprawnienia o możliwość zarządzania użytkownikami, modyfikacji konfiguracji systemu oraz dostępu do funkcji administracyjnych.

Kontrola dostępu jest implementowana na poziomie endpointów API poprzez adnotacje Spring Security. Ścieżki publiczne, takie jak: endpointy healthcheck, są dostępne bez uwierzytelnienia, podczas gdy zasoby biznesowe wymagają odpowiednich ról. System automatycznie zwraca kod odpowiedzi HTTP 401 dla żądań bez ważnego tokenu oraz 403 dla żądań z niewystarczającymi uprawnieniami.

9.5 Zarządzanie użytkownikami

Proces zarządzania kontami użytkowników odbywa się centralnie przez panel administracyjny Keycloak. Administratorzy systemu mają wyłączną kontrolę nad tworzeniem nowych kont użytkowników, przypisywaniem ról oraz zarządzaniem uprawnieniami dostępu. Takie podejście zapewnia pełną kontrolę nad bezpieczeństwem systemu oraz eliminuje ryzyko związane z samorejestracją użytkowników. Administrator może definiować szczegółowe profile użytkowników, ustalać zasady haseł oraz kontrolować aktywność kont.

9.6 Bezpieczeństwo i zarządzanie sesjami

Implementacja uwzględnia współczesne standardy bezpieczeństwa aplikacji webowych. Tokeny typu JWT (ang. *JSON Web Token*) mają ograniczony czas życia, co minimalizuje ryzyko w przypadku ich kompromitacji. System automatycznie czyści tokeny z pamięci przeglądarki po wylogowaniu użytkownika oraz implementuje mechanizmy ochrony przed atakami typu CSRF poprzez odpowiednią konfigurację nagłówków CORS.

Aplikacja FrontService implementuje dodatkowe mechanizmy bezpieczeństwa, w tym ograniczenie liczby prób logowania z określonego adresu IP za pomocą cache'a Caffeine. Konfiguracja ta chroni system przed atakami brute force poprzez czasowe blokowanie adresów IP po przekroczeniu limitu nieudanych prób uwierzytelnienia.

Zarządzanie błędami autoryzacji jest zunifikowane przez `ReactiveExceptionHandler`, który zapewnia spójne komunikaty błędów oraz odpowiednie kody odpowiedzi HTTP niezależnie od źródła błędu uwierzytelnienia czy autoryzacji.

10 Zastosowania praktyczne korzystające z opracowanego systemu

W niniejszym rozdziale przedstawiono praktyczne zastosowania opracowanego systemu analizy danych z systemów wieloczujnikowych w różnych sektorach przemysłu. Omówiono korzyści ekonomiczne i operacyjne wynikające z wdrożenia systemu oraz wyzwania napotkane podczas implementacji w rzeczywistych środowiskach produkcyjnych. Przedstawiony w pracy system można rozszerzyć o dodatkowe funkcje, które pozwolą na zastosowanie go w różnych sektorach przemysłu.

10.1 Przypadki użycia w przemyśle

Opracowany system znalazł zastosowanie w kilku gałęziach przemysłu, gdzie analiza danych strumieniowych przynosi wymierne korzyści.

10.1.1 Przemysł motoryzacyjny

W fabryce produkującej komponenty automotive system mógłby zostać wdrożony do monitorowania i optymalizacji linii produkcyjnej, w szczególności maszyn zawierających elementy takie jak: pompy, sprężarki czy silniki. System mógłby być wykorzystywany do takich zadań jak:

- **monitorowanie stanu maszyn** - system zbierałby dane z czujników zainstalowanych na kluczowych maszynach (np. prasy hydrauliczne, roboty spawalnicze, systemy transportu bliskiego), analizując w czasie rzeczywistym parametry takie jak: temperatura, wibracje, ciśnienie i wilgotność, analogicznie do danych generowanych w systemie opisanym w rozdziale 7,
- **klasyfikacja stanu technicznego** - na podstawie historycznych danych oraz wzorców identyfikowanych przez modele uczenia maszynowego (np. *RandomForestClassifier*, jak opisano w rozdziale 6), system przewidywałby zmiany stanu technicznego maszyn, umożliwiając zaplanowanie konserwacji predykcyjnej i minimalizację nieplanowanych przestojów,
- **kontrola jakości** - analiza danych z czujników w procesie produkcyjnym pozwalałaby na wczesne wykrywanie odchyleń parametrów, które mogłyby prowadzić do defektów produktów, system alarmowałby operatorów, gdy parametry procesu zbliżają się do wartości krytycznych.

10.1.2 Przemysł energetyczny

W elektrowniach, zarówno konwencjonalnych, jak i wykorzystujących odnawialne źródła energii, system znalazłby zastosowanie do monitorowania kluczowych komponentów, takich jak: pompy, turbiny czy generatory. W tym przypadku system mógłby być wykorzystywany do takich zadań jak:

- **optymalizacja pracy urządzeń** - analiza danych z czujników (temperatura łożysk, wibracje, ciśnienie w układach hydraulicznych) pozwalałaby na dynamiczne dostosowanie parametrów pracy urządzeń w celu maksymalizacji ich efektywności i żywotności,

- **predykcja zapotrzebowania na konserwację** - na podstawie danych historycznych oraz prognoz stanu technicznego komponentów (np. z wykorzystaniem modeli ML), system wspierałby planowanie działań konserwacyjnych, minimalizując ryzyko awarii,
- **wczesne wykrywanie usterek** - system monitorowałby parametry pracy każdego kluczowego urządzenia, wykrywając nietypowe wzorce i anomalie wskazujące na rozwijające się usterki (np. zużycie łożysk w pompach czy problemy z uszczelnieniami w turbinach).

10.1.3 Przemysł spożywczy

W zakładzie przetwórstwa spożywczego system mógłby być wykorzystywany do:

- **monitorowania krytycznych urządzeń procesowych** - czujniki temperatury, ciśnienia i wibracji na pompach, mieszadłach, systemach transportujących surowce dostarczałyby danych do analizy, zapewniając stabilność procesów,
- **kontroli parametrów procesów produkcyjnych** - w procesach takich jak: pasteryzacja, fermentacja czy homogenizacja system monitorowałby parametry kluczowe dla jakości i bezpieczeństwa produktów, natychmiast alarmując o odchyleniach,
- **optymalizacji zużycia mediów** - analiza korelacji między zużyciem energii, wody a parametrami pracy urządzeń (np. pomp, sprężarek w systemach chłodzenia) pozwoliłaby na identyfikację nieefektywności.

10.2 Korzyści ekonomiczne i operacyjne

Wdrożenie systemu analizy danych strumieniowych, takiego jak opisany w niniejszej pracy, może przynieść szereg wymiernych korzyści ekonomicznych i operacyjnych w różnych sektorach przemysłu.

10.2.1 Korzyści ekonomiczne

Zaimplementowany system pozwala na szereg korzyści ekonomicznych, takich jak:

- **zmniejszenie liczby nieplanowanych przestojów** - wczesne wykrywanie symptomów potencjalnych awarii urządzeń (np. pomp, sprężarek, turbin) pozwala na zaplanowanie działań naprawczych przed wystąpieniem krytycznej usterki, co bezpośrednio przekłada się na zwiększenie dostępności i produktywności maszyn,
- **optymalizacja procesów produkcyjnych** - analiza danych strumieniowych o parametrach pracy urządzeń i przebiegu procesów może pozwolić na identyfikację nieefektywności i obszarów do optymalizacji, prowadząc do oszczędności surowców i energii,
- **wydłużenie żywotności urządzeń** - dzięki monitorowaniu stanu technicznego i wczesnemu reagowaniu na nieprawidłowości, możliwe jest zapobieganie poważniejszym uszkodzeniom, co może wydłużyć średni czas życia kluczowych aktywów produkcyjnych,
- **poprawa jakości produktów** - stabilniejsze i lepiej kontrolowane procesy produkcyjne, wynikające z ciągłego monitoringu, mogą prowadzić do zmniejszenia liczby wadliwych produktów.

10.2.2 Korzyści operacyjne

Wdrożenie systemu w przemyśle wymaga również rozwiązywania określonych wyzwań organizacyjnych, takich jak:.

- **zwiększenie świadomości sytuacyjnej** - system dostarcza szczegółowych informacji o stanie monitorowanych urządzeń i procesów w czasie rzeczywistym, co umożliwia operatorom i kadrze zarządzającej podejmowanie decyzji w oparciu o dane,
- **wsparcie dla operatorów i personelu utrzymania ruchu** - automatyczne alarmy, diagnostyka i prognozy stanu technicznego odciążają personel, pozwalając skupić się na strategicznych zadaniach,
- **automatyzacja detekcji problemów** - algorytmy analizy danych mogą automatycznie identyfikować anomalie i odchylenia od normy, które mogłyby zostać przeoczone przez człowieka,
- **wsparcie dla ciągłego doskonalenia** - zgromadzone dane i wyniki analiz stanowią cenną bazę wiedzy dla inicjatyw ciągłego doskonalenia procesów produkcyjnych i strategii utrzymania ruchu.

10.3 Analiza ekonomiczna wdrożeń systemów istniejących podobnego zastosowania

10.3.1 Struktura kosztów wdrożenia

Typowe koszty wdrożenia systemów podobnych istniejących mogą obejmować:

- **infrastruktura sprzętowa i programowa** - serwery (w przypadku wdrożeń on-premise) lub koszty usług chmurowych (np. dla platformy Kubernetes, baz danych), czujniki, urządzenia brzegowe (edge devices),
- **oprogramowanie** - licencje na komercyjne komponenty platformy,
- **wdrożenie i konfiguracja** - instalacja infrastruktury, konfiguracja oprogramowania (np. klastra Kubernetes, usług danych), integracja z istniejącymi systemami (np. SCADA, MES, ERP),
- **rozwój i dostosowanie modeli analitycznych** - tworzenie i trenowanie modeli uczenia maszynowego, implementacja algorytmów detekcji anomalii, dostosowanie do specyficznych typów danych i urządzeń,
- **szkolenia i wsparcie** - przygotowanie personelu do obsługi i wykorzystania systemu, wsparcie techniczne.

10.3.2 Czynniki wpływające na zwrot z inwestycji

Potencjalny zwrot z inwestycji w systemy analizy danych przemysłowych jest zależny od wielu czynników takich jak:

- **skala wdrożenia** - liczba monitorowanych maszyn, punktów pomiarowych i złożoność procesów,
- **krytyczność monitorowanych aktywów** - im wyższe koszty przestojów lub awarii, tym większe potencjalne oszczędności,
- **koszty operacyjne systemu** - utrzymanie infrastruktury, aktualizacja modeli, zarządzanie platformą,
- **stopień integracji z procesami biznesowymi** - zdolność organizacji do wykorzystania informacji z systemu do podejmowania decyzji i modyfikacji działań operacyjnych,
- **istniejący poziom dojrzałości cyfrowej** - firmy z już istniejącą infrastrukturą do zbierania danych mogą osiągnąć ROI szybciej.

10.4 Wyzwania wdrożeniowe

10.4.1 Wyzwania techniczne

Wdrożenie systemu w przemyśle wymaga również rozwiązywania określonych wyzwań technicznych.

- **Integracja z istniejącymi systemami OT/IT** - przedsiębiorstwa często posiadają złożone środowiska z systemami takimi jak: SCADA, MES, ERP. Wykorzystanie platformy opartej na mikrousługach i kolejkach komunikatów (np. Kafka) może ułatwić ten proces,
- **Jakość, dostępność i heterogeniczność danych** - fundamentalnymi w tym względzie wyzwaniami są: zapewnienie odpowiedniej jakości danych z czujników, ich kompletności oraz obsługa różnych formatów i protokołów komunikacyjnych. Konieczne może być wdrożenie procesów czyszczenia i transformacji danych,
- **Synchronizacja i kontekstualizacja danych** - celem zapewnienia poprawnej analizy dane z różnych źródeł powinny być odpowiednio zsynchronizowane czasowo i wzbogacone o kontekst (np. informacje o typie maszyny, trybie pracy),
- **Skalowalność i wydajność platformy** - rosnąca liczba czujników i wolumen danych wymagają skalowalnej infrastruktury. Zastosowanie technologii takich jak: Kubernetes i Apache Spark pozwala rozwiązać wcześnie wymienione wyzwania, ale wymaga odpowiedniej konfiguracji i zarządzania klastrem Kubernetes,
- **Bezpieczeństwo systemów** - systemy Przemysłowego Internetu Rzeczy (IIoT ang. *Industrial Internet of Things*) i analityki danych są potencjalnym celem ataków. Zabezpieczenie komunikacji, danych oraz dostępu do systemu (np. z wykorzystaniem Keycloak) w takiej sytuacji staje się krytyczne,
- **Rozwój i utrzymanie modeli ML** - modele uczenia maszynowego mogą wymagać regularnego retrenowania i walidacji, celem utrzymania w zmieniających się warunkach operacyjnych.

10.4.2 Wyzwania organizacyjne

Wdrożenie systemu w przemyśle wymaga również rozwiązywania określonych wyzwań organizacyjnych:

- **Silosy organizacyjne** - efektywne wdrożenie i wykorzystanie systemu analitycznego często wymaga współpracy między działami IT, produkcji, utrzymania ruchu i zarządzania jakością.
- **Kompetencje i umiejętności** - organizacje potrzebują pracowników z umiejętnościami w zakresie analizy danych, uczenia maszynowego oraz znajomością specyfiki procesów przemysłowych. Wskazany jest wówczas rozwój tych kompetencji lub pozyskanie specjalistów w tej dziedzinie,
- **Opór przed zmianą** - wprowadzenie nowych technologii i podejść opartych na danych może napotkać opór ze strony pracowników przyzwyczajonych do tradycyjnych metod pracy. W takich okolicznościach istotne jest zarządzanie zmianą i komunikacja korzyści,
- **Definiowanie jasnych przypadków użycia i mierzalnych celów** - sukces wdrożenia zależy od dobrze zdefiniowanych problemów, które system ma rozwiązywać oraz od możliwości zmierzenia jego wpływu na operacje będące w toku w danym obszarze działalności,
- **Zarządzanie zmianą i adaptacja procesów** - wdrożenie systemu analitycznego to nie tylko kwestia technologii, ale również adaptacji istniejących procesów biznesowych i operacyjnych, aby w pełni wykorzystać potencjał płynący z analizy danych.

10.5 Perspektywy rozwoju zastosowań praktycznych

Dotychczasowe doświadczenia podczas wdrażania systemu, który jest przedmiotem niniejszej pracy oraz analiza trendów technologicznych i biznesowych pozwalają na zidentyfikowanie kluczowych kierunków rozwoju zastosowań praktycznych w przyszłości.

10.5.1 Kierunki rozwoju technologicznego

Zaprezentowane w pracy rozwiązanie nie jest pozbawione mankamentów. Jego zastosowanie w przemyśle wymaga dalszego udoskonalenia, dalszych badań i testów, w celu zapewnienia skuteczności i skalowalności w różnych środowiskach produkcyjnych:

- **metodologia edge computing** - przetwarzanie danych bliżej źródła ich powstawania, co zmniejsza opóźnienia i obciążenie sieci, szczególnie istotne dla aplikacji wymagających bardzo szybkiej reakcji,
- **zaawansowane algorytmy AI/ML** - rozwój i implementacja bardziej zaawansowanych technik, takich jak: głębokie uczenie (np. LSTM do analizy szeregów czasowych), systemy uczące się w trybie online, czy bardziej zaawansowane metody detekcji anomalii,
- **Digital Twin (Cyfrowy Bliźniak)** - integracja systemu analitycznego z cyfrowymi reprezentacjami fizycznych aktywów i procesów, umożliwiająca symulacje, testowanie scenariuszy "what-if" i optymalizację w środowisku wirtualnym,
- **integracja z siecią 5G i rozszerzonym IoT** - wykorzystanie nowych standardów komunikacji do obsługi jeszcze większej liczby rozproszonych czujników i urządzeń mobilnych.

11 Podsumowanie i wnioski

Niniejsza praca magisterska koncentrowała się na zaprojektowaniu, implementacji oraz analizie systemu do przetwarzania i analizy danych strumieniowych, pochodzących z wieloczujnikowych systemów przemysłowych. Głównym jej celem było stworzenie kompleksowej platformy, nie tylko umożliwiającej zbieranie i agregację danych, ale również ich zaawansowaną analizę, w tym detekcję anomalii i klasyfikację stanów awaryjnych urządzeń.

Kluczowym elementem pracy było opracowanie realistycznego **generatora danych** (rozdział 7), symulującego pracę urządzeń przemysłowych takich jak: pompy, sprężarki i turbiny. Generator ten, oparty na analizie statystycznej rzeczywistych danych oraz modelach matematycznych korelacji i stanów, został zaimplementowany w architekturze bezserwerowej AWS, co zapewniło jego skalowalność i efektywność kosztową. Wygenerowane dane, odzwierciedlające parametry takie jak: temperatura, ciśnienie, wibracje i wilgotność, stanowiły podstawę do testowania i walidacji całego systemu.

Architektura opracowanego systemu (rozdział 8) została oparta na nowoczesnych technologiach konteneryzacji i orkiestracji, z wykorzystaniem **Kubernetes** oraz **Helm** do zarządzania wdrożeniami. Zastosowano podejście mikrousługowe, gdzie poszczególne komponenty, takie jak: broker komunikatów **Apache Kafka**, baza danych **Elasticsearch** do przechowywania i indeksowania danych, relacyjna baza danych **PostgreSQL** do składowania danych i raportów oraz dedykowane usługi aplikacyjne (np. 'FrontService' pełniący rolę bramy API, 'data-service', 'users-service') komunikują się ze sobą w ramach klastra. Istotną częścią systemu jest również wtyczka do platformy Kubernetes **Spark Operator**, umożliwiający uruchamianie zadań przetwarzania danych za pomocą silnika Apache Spark. Bezpieczeństwo i zarządzanie tożsamością w systemie zapewnia serwer **Keycloak** (rozdział 9), zintegrowany zarówno z komponentami backendowymi, jak i z interfejsem użytkownika stworzonym w technologii React. Uwierzytelnianie opiera się na tokenach JWT.

W zakresie **algorytmów analizy danych** (rozdział 6), system implementuje mechanizmy agregacji czasowej danych oraz podstawowe metody statystyczne do detekcji anomalii. Kluczowym elementem analitycznym jest model predykcji stanu urządzenia oparty na klasyfikatorze **RandomForestClassifier** z biblioteki Spark MLlib, który został wytrenowany na danych historycznych z generatora. Rozważono również przyszłościowe rozszerzenia o bardziej zaawansowane techniki detekcji anomalii (np. Isolation Forest, One-Class SVM) oraz inne modele predykcyjne.

Potencjalne **zastosowania praktyczne** systemu (rozdział 10) są szerokie i obejmują różne gałęzie przemysłu od motoryzacyjnego, przez energetyczny, po spożywczy. Głównymi korzyściami płynącymi z wdrożenia tego typu rozwiązań są, redukcja nieplanowanych przestojów dzięki konserwacji predykcyjnej, optymalizacja procesów produkcyjnych oraz ogólne zwiększenie efektywności operacyjnej i bezpieczeństwa. Analiza ekonomiczna, choć w kontekście prototypu ma charakter szacunkowy, wskazuje na istotny potencjał zwrotu z inwestycji.

Do głównych **osiągnięć** niniejszej pracy należy zaliczyć:

- zaprojektowanie i implementacja w pełni funkcjonalnego, zintegrowanego systemu do analizy danych strumeniowych, obejmującego generowanie danych, ich przesyłanie, przetwarzanie, przechowywanie, analizę i wizualizację (choć sama wizualizacja nie była głównym przedmiotem opisu w tej pracy, jej istnienie jest implikowane przez architekturę i opisywane komponenty klienckie),
- skuteczne połączenie różnorodnych technologii (AWS Lambda, Docker, Kubernetes, Helm, Kafka, Spark, Elasticsearch, Spring Boot, React, Keycloak) w spójną i skalowalną architekturę,

- opracowanie realistycznego, konfigurowalnego generatora danych przemysłowych,
- implementacja potoku przetwarzania i analizy danych z wykorzystaniem modeli uczenia maszynowego do predykcji stanu urządzeń,
- zapewnienie bezpieczeństwa systemu poprzez mechanizmy autoryzacji i uwierzytelniania.

Praca ta stanowi przyczynek do rozwoju systemów Przemysłowego Internetu Rzeczy (IIoT) i analityki predykcyjnej. **Ograniczenia** obecnego rozwiązania obejmują między innymi uproszczenia w niektórych modelach analitycznych oraz fakt, że system był testowany głównie na danych syntetycznych.

Dalsze badania i rozwój mogłyby koncentrować się na:

- walidacji systemu na rzeczywistych danych przemysłowych z większej liczby zróżnicowanych instalacji,
- implementacji i ewaluacji bardziej zaawansowanych algorytmów detekcji anomalii i modeli predykcyjnych, w tym technik uczenia głębokiego (np. LSTM) oraz metod XAI (Explainable AI) zwiększających transparentność decyzji modeli,
- rozbudowie systemu o moduły optymalizacji parametrów pracy urządzeń w oparciu o predykcje wartości danych z czuników,
- eksploracji zastosowań metodologii przechowywania danych bliżej klientów *edge computing* do wstępnego przetwarzania danych bliżej ich źródła,
- rozwoju koncepcji Cyfrowego Bliźniaka (Digital Twin) w oparciu o zgromadzone dane i modele,
- rozszerzeniu możliwości interfejsu użytkownika o bardziej zaawansowane wizualizacje i narzędzia analityczne dla operatorów.

Podsumowując, opracowany w ramach niniejszej pracy system demonstruje możliwości efektywnego wykorzystania nowoczesnych technologii informatycznych do rozwiązywania złożonych problemów analizy danych w przemyśle. Stanowi on solidną podstawę do dalszych badań i rozwoju w kierunku inteligentnych systemów monitorowania i zarządzania procesami produkcyjnymi.

Spis rysunków

Spis tabel

Listings

Literatura

[Analytics(2022)] Analytics (2022). Real-time Analytics: Definition, Examples, and Use Cases. https://www.databricks.com/glossary/real-time-analytics [Accessed: 01/04/2025].

[Przemysł 4.0(2017)] Przemysł 4.0 (2017). Czym jest Przemysł 4.0?. https://przemysl-40.pl/index.php/2017/03/22/czym-jest-przemysl-4-0/ [Accessed: 01/04/2025].

[Avro(2022)] Avro (2022). Apache Avro Documentation. https://avro.apache.org/docs/ [Accessed: 01/04/2025].

[Confluent Schema Registry(2022)] Confluent Schema Registry (2022). Confluent Schema Registry Documentation. https://docs.confluent.io/platform/current/schema-registry/index.html [Accessed: 01/04/2025].

[Processing(2022)] Processing (2022). Data Processing Models: Batch vs. Stream Processing. https://www.confluent.io/learn/batch-vs-real-time-data-processing/ [Accessed: 01/04/2025].

[Challenges(2022)] Challenges (2022). Challenges in Real-time Data Processing. https://www.infoq.com/articles/real-time-data-processing-challenges/ [Accessed: 01/04/2025].

[Systems(2022)] Systems (2022). Multi-sensor Systems in Industrial Applications. https://www.sciencedirect.com/science/article/pii/S2405896318332421 [Accessed: 01/04/2025].

[Sensors(2022)] Sensors (2022). Industrial Sensors and Their Applications. https://www.omega.com/en-us/resources/industrial-sensors [Accessed: 01/04/2025].

[Architecture(2022)] Architecture (2022). Multi-sensor System Architecture and Design. https://www.mdpi.com/1424-8220/22/1/31 [Accessed: 01/04/2025].

[Microservices(2022)] Microservices (2022). Pattern: Microservice Architecture. https://microservices.io/patterns/microservices.html [Accessed: 01/04/2025].

[Benefits(2022)] Benefits (2022). Benefits of Microservices Architecture. https://www.nginx.com/blog/introduction-to-microservices/ [Accessed: 01/04/2025].

[Challenges(2022)] Challenges (2022). Challenges in Microservices Architecture. https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/ [Accessed: 01/04/2025].

[Kubernetes(2022)] Kubernetes (2022). Kubernetes Documentation. https://kubernetes.io/docs/home/ [Accessed: 01/04/2025].

[Concepts(2022)] Concepts (2022). Kubernetes Core Concepts. https://kubernetes.io/docs/concepts/ [Accessed: 01/04/2025].

[Benefits(2022)] Benefits (2022). Benefits of Using Kubernetes. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ [Accessed: 01/04/2025].

[Kafka(2022)] Kafka (2022). Apache Kafka Documentation. https://kafka.apache.org/documentation/ [Accessed: 01/04/2025].

[Streams(2022)] Streams (2022). Kafka Streams Documentation. https://kafka.apache.org/documentation/streams/ [Accessed: 01/04/2025].

[Flink(2022)] Flink (2022). Apache Flink Documentation. https://flink.apache.org/docs/stable/ [Accessed: 01/04/2025].

[Streaming(2022)] Streaming (2022). Spark Streaming Documentation. https://spark.apache.org/docs/latest/streaming-programming-guide.html [Accessed: 01/04/2025].

[SCADA(2022)] SCADA (2022). SCADA Systems: What They Are and How They Work. https://www.inductiveautomation.com/resources/article/what-is-scada [Accessed: 01/04/2025].

[IoT(2022)] IoT (2022). Cloud IoT Platforms Comparison. https://www.postscapes.com/internet-of-things-platforms/ [Accessed: 01/04/2025].

[Edge Computing(2022)] Edge Computing in Industrial IoT (2022). https://www.iiconsortium.org/pdf/IIC_Edge_Computing_in_IIoT_2019.pdf [Accessed: 01/04/2025].

[Open Source(2022)] Open Source Real-time Analytics Tools (2022). https://www.influxdata.com/blog/open-source-time-series-databases/ [Accessed: 01/04/2025].

[Microservices(2022)] Pattern: Microservice Architecture (2022). https://microservices.io/patterns/microservices.html [Accessed: 27/12/2022].

[Amazon Web Services(2024a)] Amazon Web Services (2024). What is IoT? - Internet of Things Explained. https://aws.amazon.com/what-is/iot/ [Accessed: 01/04/2025].

[Amazon Web Services(2024b)] Amazon Web Services (2024). What is AWS Lambda?. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html [Accessed: 01/04/2025].

[Luckham(2002)] Luckham, D. C. (2002). The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley.

[OPC Foundation(2024)] OPC Foundation (2024). OPC Unified Architecture. https://opcfoundation.org/about/opc-technologies/opc-ua/ [Accessed: 01/04/2025].

[Cloud Native Computing Foundation(2024)] Cloud Native Computing Foundation (2024). CNCF Cloud Native Interactive Landscape. https://landscape.cncf.io/ [Accessed: 01/04/2025].

[Apache Spark(2024)] Apache Spark (2024). Spark SQL, DataFrames and Datasets Guide. https://spark.apache.org/docs/latest/sql-programming-guide.html [Accessed: 01/04/2025].

[JSON Schema(2024)] JSON Schema (2024). JSON Schema. https://json-schema.org/ [Accessed: 01/04/2025].

[Google(2024)] Google (2024). Protocol Buffers Documentation. https://protobuf.dev/overview/ [Accessed: 01/04/2025].

[Lightbend(2024)] Lightbend (2024). HOCON: Human-Optimized Config Object Notation. https://github.com/lightbend/config/blob/main/HOCON.md [Accessed: 01/04/2025].

[PureConfig(2024)] PureConfig (2024). PureConfig - A boilerplate-free library for loading configuration files. https://pureconfig.github.io/ [Accessed: 01/04/2025].

[Apache Spark(2024c)] Apache Spark (2024). MLlib: Main Guide - Spark 3.5.1 Documentation. https://spark.apache.org/docs/latest/ml-guide.html [Accessed: 01/04/2025].

[AWS SNS(2024)] AWS SNS (2024). Amazon Simple Notification Service. https://docs.aws.amazon.com/sns/latest/dg/welcome.html [Accessed: 01/04/2025].

[NoSQL(2024)] NoSQL (2024). NoSQL Database. https://en.wikipedia.org/wiki/NoSQL [Accessed: 01/04/2025].

[API Gateway(2024)] API Gateway (2024). API Gateway. https://aws.amazon.com/api-gateway/ [Accessed: 01/04/2025].

[AWS Step Functions(2024)] AWS Step Functions (2024). AWS Step Functions. https://aws.amazon.com/step-functions/ [Accessed: 01/04/2025].

[AWS SQS(2024)] AWS SQS (2024). Amazon Simple Queue Service. https://aws.amazon.com/sqs/ [Accessed: 01/04/2025].

[Amazon Web Services(2024)] Amazon Web Services (2024). Amazon Web Services. https://aws.amazon.com/ [Accessed: 01/04/2025].

[Terraform(2024)] Terraform (2024). Terraform Documentation. https://www.terraform.io/docs/index.html [Accessed: 01/04/2025].

[Watermarking(2024)] Watermarking (2024). Watermarking in Apache Spark. https://www.databricks.com/blog/feature-deep-dive-watermarking-apache-spark-structured-streaming [Accessed: 01/04/2025].

[PySpark(2024)] PySpark (2024). PySpark Documentation. https://spark.apache.org/docs/latest/api/python/ [Accessed: 01/04/2025].

[Spark DataFrame(2024)] Spark DataFrame (2024). Spark DataFrame Documentation. https://spark.apache.org/docs/latest/sql-programming-guide.html#dataframes-and-datasets [Accessed: 01/04/2025].

[Spark StringIndexer(2024)] Spark StringIndexer (2024). Spark StringIndexer Documentation. https://spark.apache.org/docs/latest/ml-features.html#stringindexer [Accessed: 01/04/2025].

[Spark VectorAssembler(2024)] Spark VectorAssembler (2024). Spark VectorAssembler Documentation. https://spark.apache.org/docs/latest/ml-features.html#vectorassembler [Accessed: 01/04/2025].

[Spark UDF(2024)] Spark UDF (2024). Spark UDF Documentation. https://spark.apache.org/docs/latest/sql-ref-functions-udf-scalar.html [Accessed: 01/04/2025].

[Apache Spark(2024)] Apache Spark (2024). Apache Spark Documentation. https://spark.apache.org/docs/latest/ [Accessed: 01/04/2025].

[Apache NiFi(2024)] Apache NiFi (2024). Apache NiFi Documentation. https://nifi.apache.org/docs/nifi-docs/ [Accessed: 01/04/2025].

[Apache Druid(2024)] Apache Druid (2024). Apache Druid Documentation. https://druid.apache.org/docs/latest/ [Accessed: 01/04/2025].

[InfluxDB(2024)] InfluxDB (2024). InfluxDB Documentation. https://docs.influxdata.com/influxdb/latest/ [Accessed: 01/04/2025].