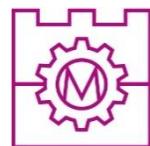




**POLITECHNIKA KRAKOWSKA im. T. Kościuszki**  
Wydział Mechaniczny  
**Katedra Informatyki Stosowanej**



Kierunek studiów: Informatyka Stosowana

STUDIA STACJONARNE

## **PRACA DYPLOMOWA**

MAGISTERSKA

**inż. Krystian Dutka**

Analiza architektur przetwarzania danych w systemach Big Data

Analysis of data processing architectures in Big Data systems

Promotor:  
dr hab. inż. **Aneta Iwona Gądek-Moszczak**, prof. PK

Kraków, rok akad. 2023/2024



<sup>\*</sup>) – niepotrzebne skreślić

## OŚWIADCZENIE O SAMODZIELNYM WYKONANIU PRACY DYPLOMOWEJ

Oświadczam, że przedkładana przeze mnie praca dyplomowa magisterska/inżynierska-została napisana przeze mnie samodzielnie. Jednocześnie oświadczam, że ww. praca:

- 1) nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz.U. z 2021 r. poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym, a także nie zawiera danych i informacji, które uzyskałem/le<sup>m</sup>\* w sposób niedozwolony,
- 2) nie była wcześniej podstawą żadnej innej procedury związanej z nadawaniem tytułów zawodowych, stopni lub tytułów naukowych.

Jednocześnie wyrażam zgodę na:

- 1) poddanie mojej pracy kontroli za pomocą systemu Antyplagiat oraz na umieszczenie tekstu pracy w bazie danych uczelni, w celu ochrony go przed nieuprawnionym wykorzystaniem. Oświadczam, że zostałem/le<sup>m</sup>\* poinformowany/-ka i wyrażam zgodę, by system Antyplagiat porównywał tekst mojej pracy z tekstem innych prac znajdujących się w bazie danych uczelni, z tekstami dostępnymi w zasobach światowego Internetu oraz z bazą porównawczą systemu Antyplagiat,
- 2) to, aby moja praca pozostała w bazie danych uczelni przez okres wynikający z przepisów prawa. Oświadczam, że zostałem poinformowany i wyrażam zgodę, że tekst mojej pracy stanie się elementem porównawczej bazy danych uczelni, która będzie wykorzystywana, w tym także udostępniana innym podmiotom, na zasadach określonych przez uczelnię, w celu dokonywania kontroli antyplagiatowej prac dyplomowych/doktorskich, a także innych tekstów, które powstaną w przyszłości.

podpis

- 1) Wyrażam zgodę na udostępnianie mojej pracy dyplomowej w Akademickim Systemie Archiwizacji Prac na PK do celów naukowo-badawczych z poszanowaniem przepisów ustawy o prawie autorskim i prawach pokrewnych (Dz.U. z 2021 r. poz. 1062)..

TAK/NIE\*

podpis

Jednocześnie przyjmuję do wiadomości, że w przypadku stwierdzenia popełnienia przez mnie czynu polegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzej pracy, lub ustalenia naukowego, Rektor PK stwierdzi nieważność postępowania w sprawie nadania mi tytułu zawodowego (art. 77 ust. 5 ustawy z dnia 18 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce, (Dz.U. z 2021 r. poz. 478, z późn. zm.)).

podpis



## SPIS TREŚCI

<b>1. WSTĘP</b>	<b>6</b>
<b>2. CEL I ZAKRES</b>	<b>7</b>
<b>3. PRZEGŁĄD LITERATURY NAUKOWEJ</b>	<b>8</b>
<b>4. WPROWADZENIE TEORETYCZNE</b>	<b>10</b>
4.1. Przetwarzanie danych oraz technologie Big Data.....	10
4.1.1. Trzy warstwy architektury w przetwarzaniu danych .....	10
4.1.2. Architektury systemów Big Data .....	13
4.1.3. Jakość danych.....	18
4.2. Cold Path i Hot Path w Przetwarzaniu Danych.....	20
4.2.1. Koncepcje Cold Path i Hot Path.....	20
4.2.2. Zastosowania Cold Path i Hot Path w Systemach Big Data .....	21
4.3. Apache Kafka & Apache ZooKeeper .....	23
4.3.1. Architektura Apache Kafka oraz Apache ZooKeeper.....	23
4.3.2. Zastosowania Apache Kafka w Przetwarzaniu Strumieniowym.....	27
4.4. Przetwarzanie danych w Chmurze .....	29
4.4.1. Microsoft Azure Cloud .....	29
4.4.2. Amazon Web Services (AWS).....	32
4.4.3. Google Cloud Platform .....	35
<b>5. CZĘŚĆ PROJEKTOWA</b>	<b>37</b>
5.1. Założenia ekosystemu do przetwarzania danych w czasie rzeczywistym.....	38
5.1.1. Biznesowe podstawy projektu.....	38
5.1.2. Wykrycie i wybór źródeł danych .....	39
5.1.3. Określenie początkowych wymagań.....	40
5.1.4. Finalne wymagania systemowe.....	41
5.2. Przedstawienie możliwych rozwiązań architektonicznych .....	43
5.2.1 Architektura Big Data z wykorzystaniem Azure .....	43
5.2.2 Architektura Big Data z wykorzystaniem Apache Kafka .....	46
5.3. Wybór Architektura Big Data .....	48
5.3.1 Architektury ekosystemu stacji meteorologicznej .....	48
5.3.2 Orchestracja, zarządzanie metadanymi oraz administracja stacji meteorologicznej .....	49
5.2.3 Przetwarzanie danych w ekosystemie .....	50
5.2.4 Magazyn danych i organizacja danych .....	50
5.2.5 Udostępnianie oraz analiza danych .....	51
5.2.6 Diagram architektury oraz tabela użytych technologii.....	52
5.4. Opis implementacji .....	54
5.4.1 Generowanie danych za pomocą Pythona.....	54
5.4.2 Przetwarzanie danych i zapisywanie danych za pomocą Apache Kafka i Apache ZooKeeper .....	56
5.4.3 Transformacja danych za pomocą Javy .....	57
5.5. Podsumowanie .....	64
5.5.1 Porównanie z architekturą wykorzystującą Azure .....	64
<b>6. WNIOSKI</b>	<b>66</b>
<b>LITERATURA</b>	<b>67</b>
<b>SUMMARY</b>	<b>68</b>

## **1. Wstęp**

W dzisiejszych czasach korporacje dążą do maksymalizacji szansy pozyskania nowych klientów oraz optymalizacji własnych produktów i linii produkcyjnych. W tym celu muszą analizować dane dotyczące swoich przedsiębiorstw, produktów oraz usług, jakie oferują. Posłużyć ma do tego przetwarzanie danych, które może być przeprowadzane w czasie rzeczywistym (lub zbliżonym do rzeczywistego). Wyróżnia się dwa typy takiej analizy: analizę danych historycznych oraz analizę danych rzeczywistych. Wiedza pozyskana za ich pomocą ma wspomóc korporację w uzyskaniu przewagi nad konkurencją oraz w podejmowaniu decyzji w procesach biznesowych.

Architektura wymagana do wykonania tego typu analiz wykorzystuje platformy Big Data ze względu na ich możliwości w przetwarzaniu, transformowaniu i przechowywaniu danych. Rozwiązanie takie zapewnia również dużą skalowalność, jeśli wymagane byłoby, aby architektura się rozbudowała. Jest ona niezbędna ze względu na operowanie na petabajtach danych, w których posiadaniu są duże firmy. Dostarczanie danych w czasie rzeczywistym odbywa się zazwyczaj za pomocą komponentów IoT (ang. *Internet of Things*) takich jak sensory, diody, czujniki, przekaźniki, mikrokontrolery itp. Następnie dane te są transformowane za pomocą procesów ELT (ang. *Extract Load and Transform*) lub ETL (ang. *Extract Transform and Load*). Struktura przetwarzania danych w architekturze Big Data ma na celu pozbycie się szumu i uzyskanie danych najlepszej jakości, tak aby można było wskazać mocne i słabe strony przedsiębiorstwa.

Praca magisterska prezentuje szczegółowe omówienie procesów przetwarzania danych w czasie rzeczywistym w kontekście architektur Big Data oraz przedstawia korzyści płynące z zastosowania takich rozwiązań w nowoczesnych korporacjach.

## **2. Cel i zakres**

Celem przedstawionej pracy magisterskiej jest analiza różnych koncepcji budowania architektury do przetwarzania danych w systemach Big Data. W szczególności skupiono się na porównaniu rozwiązań architektonicznych wykorzystujących technologie chmurowe i nie chmurowe, a przy tym przeanalizowano różnice, wady oraz zalety tych podejść. Ważny element pracy stanowi zademonstrowanie praktycznego zastosowania wybranej architektury w rozwiązaniu problemu związanego z napływem dużej ilości danych do stacji meteorologicznej. Celem tej demonstracji było pokazanie, w jaki sposób system może efektywnie wysyłać informacje ostrzegawcze przed przekroczeniem przez urządzenia IoT temperatury zagrażającej uszkodzeniu przekaźników.

Zakres pracy obejmuje szczegółową analizę procesów przetwarzania, transformowania, ładowania i magazynowania danych. Wprowadzenie teoretyczne uwzględnia:

- przedstawienie kolejnych warstw architektury systemu, na których dane są zbierane, przetwarzane, analizowane i przechowywane;
- omówienie technik czyszczenia i walidacji danych oraz usuwania szumów, które są niezbędne do zapewnienia wysokiej jakości danych oraz uzyskania wiarygodnych wyników analizy;
- prezentację dwóch różnych podejść do przetwarzania danych – Cold Path oraz Hot Path;
- wykorzystanie środowiska Apache w przetwarzaniu strumieniowym.
- analizę korzyści i wyzwań związanych z wykorzystaniem rozwiązań chmurowych do przetwarzania danych;
- omówienie usług oferowanych przez głównych dostawców chmury, takich jak Amazon Web Services (AWS) czy Microsoft Azure.

W części praktycznej pracy zaprezentowano szczegółowy przykład implementacji wybranej architektury w stacji meteorologicznej. System został zaprojektowany w taki sposób, aby był w stanie efektywnie przetwarzać napływające dane w czasie rzeczywistym, a następnie generować i wysyłać alerty w przypadku wykrycia potencjalnych zagrożeń, takich jak przekroczenie bezpiecznej temperatury przez urządzenia IoT. Przykład ten ma na celu zilustrowanie praktycznych aspektów budowy i wdrażania systemów przetwarzania danych w czasie rzeczywistym. Pokazuje również, w jaki sposób technologie Big Data mogą zostać wykorzystane do poprawy bezpieczeństwa i wydajności operacyjnej w różnych zastosowaniach przemysłowych.

### **3. Przegląd literatury naukowej**

Praca przedstawia studium architektur oraz budowy systemów przetwarzania danych w czasie rzeczywistym z wykorzystaniem technologii Big Data. Systemy tego rodzaju wykorzystywane są w wielu dziedzinach technologii, nauki oraz kultury, pomagając w gromadzeniu, zarządzaniu i przetwarzaniu danych. Kluczowe jest zwrócenie uwagi na następujące zagadnienia:

- Jakie są najważniejsze różnice między tradycyjnymi systemami przetwarzania danych a systemami Big Data przetwarzającymi dane w czasie rzeczywistym?
- Jakie modele przetwarzania wykorzystywane są w systemach Big Data w czasie rzeczywistym?
- W jaki sposób architektura systemu przetwarzania danych wpływa na jego wydajność i skalowalność?
- Jakie są korzyści i wyzwania związane z wykorzystaniem technologii chmurowych w systemach Big Data?
- Jakie rozwiązania architektoniczne są najbardziej efektywne w kontekście przetwarzania danych w czasie rzeczywistym?

Do analizy tematu wykorzystano artykuły i publikacje naukowe oraz dokumentacje techniczne. Wyszukiwanie źródeł naukowych przeprowadzono za pomocą portalu Google Scholar, który umożliwia przeszukiwanie baz publikacji w oparciu o słowa kluczowe i frazy w tytułach. Poszukiwano następujących fraz:

- „real-time data processing”
- „Big Data architecture”
- „cloud data platforms”
- „data streaming”
- „IoT data processing”

Firmy wykorzystują analizę przetwarzanych danych do podejmowania decyzji podczas procesów biznesowych. Analizy konkretnych przedsiębiorstw są zazwyczaj bardzo szczegółowe, jednak zindywidualizowane, a projekty praktycznie nigdy nie są możliwe do wykorzystania do analiz innego przedsiębiorstwa, bez konieczności wprowadzenia dużej ilości zmian. Skutkiem tego jest fakt, iż liczba przydatnych publikacji naukowych nie jest duża. W większości publikacji przedstawiono badania oraz analizy zastosowań rozwiązań dla konkretnych problemów badawczych, takich jak monitorowanie przemysłowe, analiza danych meteorologicznych lub zarządzanie infrastrukturą miejską. Spośród wyszukanych publikacji wybrane zostały te, których tytuły i streszczenia zawierały informacje o zastosowanej architekturze systemu przetwarzania danych oraz mówiły o badaniu podmiotowym systemów danych.

W pracy „Hand of multisensor data fusion” autorzy, David L. Hall oraz James Llinas [1], przedstawiają podejście do przetwarzania danych, które opiera się na wykorzystaniu tylko najważniejszych informacji. Do osiągnięcia tego celu wykorzystywana jest fuzja danych. W najprostszym możliwym ujęciu autorzy zakładają, że wszystkie wymagane informacje znajdują się w zestawie pomiarów czujników, a rolą wspomnianej fuzji jest wyodrębnianie informacji osadzonych w zestawie danych. [1]

W opracowaniu „Storm Real-time Processing Cookbook” autorstwa Quintona Andersona [2], autor postrzega potok danych w kategoriach strumieni, krotek w strumieniu oraz predykatów działających na tych krotkach. Uważa, że takie podejście umożliwia łatwe opisanie rozwiązania problemu przetwarzania danych. Dodatkowo wspiera ono kompozycyjność, ponieważ predykaty są z natury kompozycyjne, a same potoki danych mogą być komponowane, tworząc tym samym większe i bardziej złożone struktury. W opisanej przez niego architekturze kaskadowość zapewnia abstrakcję dla MapReduce w ten sam sposób, w jaki Trident robi to dla Storm. Dzięki takiemu podejściu, zastosowanym narzędziom i uwzględnieniu odpowiednich aspektów, możliwe jest stworzenie architektury przetwarzania dużych zbiorów danych w czasie rzeczywistym. [2]

P. Taylor Goetz oraz Brian O’Neill w pracy „Storm Blueprints: Patterns for Distributed Real-time Computation” [3] przedstawiają podejście do przetwarzania danych w czasie rzeczywistym, które opiera się na wykorzystaniu framework-a Storm. Ich podejście charakteryzuje się kilkoma kluczowymi założeniami. Autorzy podkreślają znaczenie ciągłego przetwarzania strumieni danych. Strumienie te mogą pochodzić z różnych źródeł, takich jak czujniki, logi systemowe czy zdarzenia generowane przez użytkowników. Storm umożliwia skalowanie w poziomie, co oznacza, że duże ilości danych mogą być przetwarzane równolegle przez podział pracy na wiele węzłów, zapewniając wysoką wydajność systemu. Niezawodność i odporność na błędy to kolejne ważne elementy podejścia autorów. Opisują mechanizmy zapewniające niezawodność przetwarzania w Storm, takie jak monitorowanie przekroczeń czasów przetwarzania krotek oraz mechanizmy powtórnego przetwarzania w przypadku awarii. Ponadto podejście do przetwarzania danych w Storm opiera się na kompozycyjności, gdzie różne komponenty (ang. *spouts* oraz *bolts*) mogą być łączone w bardziej złożone topologie przetwarzania. [3]

Literatura naukowa dostarcza licznych informacji na temat przetwarzania danych w czasie rzeczywistym w systemach Big Data, ze szczególnym uwzględnieniem zastosowań IoT. Różnorodność podejść architektonicznych oraz dostępnych narzędzi i technologii pozwala na efektywne zarządzanie danymi i optymalizację procesów biznesowych. Przyszłe badania powinny skupić się na dalszej integracji technologii chmurowych i IoT oraz rozwijaniu nowych metod zapewniających jeszcze wyższą jakość danych i wydajność przetwarzania.

## 4. Wprowadzenie teoretyczne

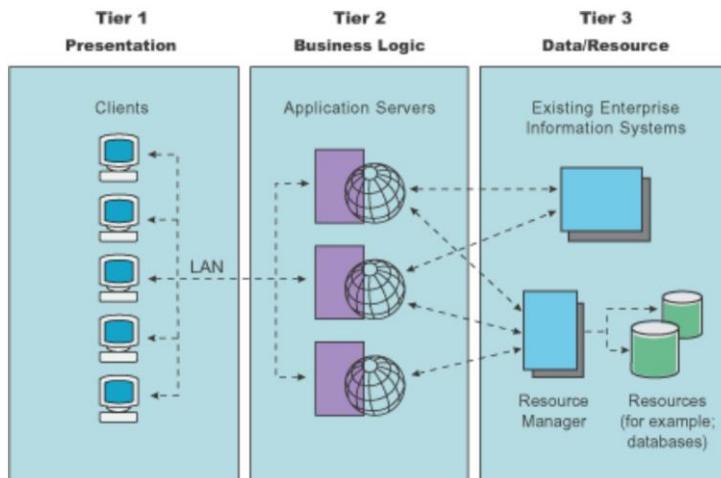
### 4.1. Przetwarzanie danych oraz technologie Big Data

W dobie intensywnego rozwoju technologicznego oraz rosnącej ilości generowanych i gromadzonych danych, przetwarzanie informacji oraz technologie Big Data odgrywają kluczową rolę w analizie i zarządzaniu danymi na szeroką skalę są one niezbędne do rozwoju wielkich przedsiębiorstw i analiza danych jakie posiada firma. Wiedza ta może dać korporacji przewagę względem innych firm na rynku.

#### 4.1.1. Trzy warstwy architektury w przetwarzaniu danych

Architektura trójwarstwowa to klasyczny model projektowania aplikacji, który dzieli je na trzy poziomy: warstwę prezentacji, czyli interfejs użytkownika; warstwę aplikacji (inaczej logika biznesowa) oraz warstwę danych, czyli warstwę odpowiedzialną za zarządzanie i przechowywanie danych. Główną zaletą tej architektury jest możliwość oddzielnego działania każdej z warstw na własnej infrastrukturze. Rozdzielenie to pozwala na pracę nad każdą z warstw z osobna, a także na wprowadzanie zmian bez ingerencji w pozostałe warstwy. Rozwiązanie to sprawiło, że architektura trójwarstwowa na wiele lat stała się standardem w aplikacjach kliencko-serwerowych. Obecnie wiele z tego typu aplikacji wykorzystuje technologie natywne dla chmury, takie jak kontenery i mikrouslugi, oraz migrację do chmury [2].

Przykładową architekturę trójwarstwową przedstawia rysunek 4.1.



Rys. 4.1. Architektura trójwarstwowa  
Źródło: <https://www.ibm.com>

Warstwa prezentacyjna, będąca interfejsem aplikacji, pełni kluczową rolę w interakcji między użytkownikiem a systemem. Jej głównym zadaniem jest wyświetlanie informacji oraz zbieranie danych wejściowych od użytkowników. W aplikacjach webowych warstwa ta tworzy strukturę, stylizację i interaktywność interfejsu, często z użyciem narzędzi, które ułatwiają

kreowanie dynamicznych i responsywnych interfejsów. Dla aplikacji desktopowych interfejs użytkownika jest tworzony przy użyciu różnych technologii, w zależności od platformy, na której działa aplikacja. Ważnym aspektem warstwy prezentacyjnej jest zapewnienie dobrego doświadczenia użytkownika (UX), co oznacza, że interfejs powinien być intuicyjny, łatwy w nawigacji i estetycznie przyjemny. Oprócz tego warstwa prezentacyjna musi być również zoptymalizowana pod kątem wydajności i dostępności [6].

Warstwa biznesowa, znana również jako warstwa logiki aplikacji lub warstwa środkowa, stanowi rdzeń każdej aplikacji. Jest odpowiedzialna za przetwarzanie danych, realizację logiki biznesowej oraz zarządzanie przepływem informacji między warstwą prezentacyjną a warstwą danych. W tej warstwie definiowane są zasady biznesowe, które determinują sposób przetwarzania danych oraz wykonywania operacji.

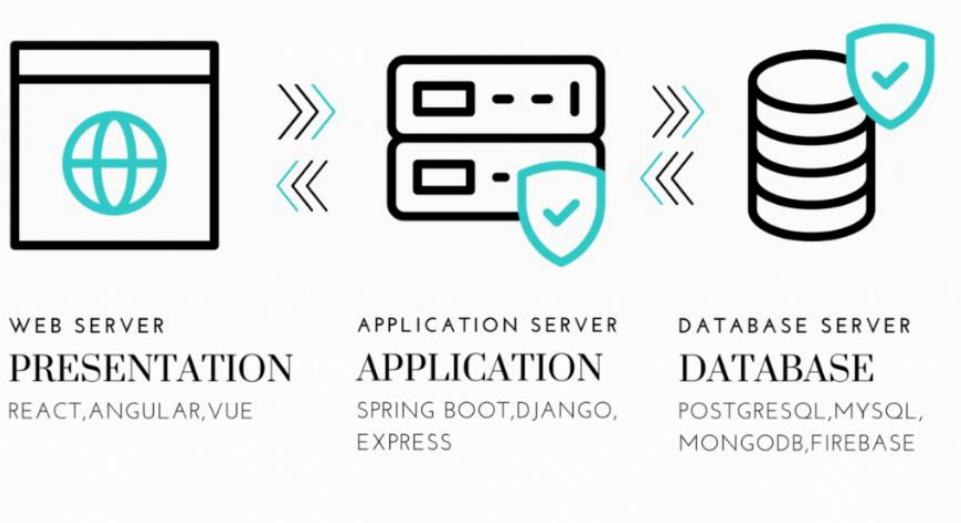
Logika biznesowa obejmuje różnorodne procesy, w tym validację danych wejściowych, kalkulację wyników, przetwarzanie zamówień, zarządzanie użytkownikami oraz inne operacje specyficzne dla danej aplikacji. Kluczowym zadaniem warstwy biznesowej jest komunikacja z pozostałymi warstwami. Omawiana warstwa pobiera dane od warstwy prezentacyjnej, przetwarza je zgodnie z logiką biznesową, a następnie zapisuje wyniki do warstwy danych lub zwraca je z powrotem do warstwy prezentacyjnej. Ważnym aspektem warstwy biznesowej jest zarządzanie transakcjami, a więc zapewnienie, że wszystkie operacje na danych będą wykonywane w sposób spójny i niezawodny. Ta część architektury powinna być skalowalna i wydajna, aby być w stanie obsługiwać rosnące obciążenia i zwiększącą się liczbę użytkowników. W praktyce oznacza to stosowanie odpowiednich wzorców architektonicznych, takich jak mikrousługi. Pozwalają one na podział aplikacji na mniejsze, niezależne komponenty, które można skalować w zależności od potrzeb. Kolejnym kluczowym elementem tej warstwy jest bezpieczeństwo. Dane muszą być chronione przed nieautoryzowanym dostępem i manipulacją. Aby to uzyskać stosowane są mechanizmy uwierzytelniania i autoryzacji, szyfrowanie danych oraz regularne audyty bezpieczeństwa, których celem jest wykrywanie i usuwanie potencjalnych luk. Warstwa biznesowa powinna być łatwa do utrzymania i rozwijania, a kod – dobrze zorganizowany, modularny i zgodny z najlepszymi praktykami programistycznymi. Stosowanie wzorców projektowych, takich jak MVC (ang. *model view control*), oraz narzędzi do automatyzacji testów i ciągłej integracji CI/CD (ang. *continuous integration and continuous delivery*) pomaga w utrzymaniu wysokiej jakości kodu oraz w szybkim wdrażaniu nowych funkcji [4].

Warstwa danych jest odpowiedzialna za trwałe przechowywanie i zarządzanie danymi używanymi przez aplikację. Warstwa ta zapewnia integralność i bezpieczeństwo danych oraz efektywny dostęp do nich. Relacyjne bazy danych, takie jak PostgreSQL, MySQL, Oracle i Microsoft SQL Server, używają języka SQL do definiowania i manipulowania danymi. Zapewniają one transakcyjność i spójność danych. Bazy NoSQL, takie jak MongoDB

i Cassandra, są stosowane w sytuacjach wymagających skalowalności i elastyczności w modelowaniu danych. Są one idealne dla aplikacji, które zajmują się obsługiwaniem dużych ilości danych niestrukturyzowanych, napływających w krótkim czasie. W kontekście przetwarzania danych rzeczywistych, charakteryzujących się wysoką przepustowością i częstym napływem nowych danych, wykorzystanie w architekturze bazy NoSQL oferuje kilka różnych korzyści, przede wszystkim zdolność do obsługi wielu węzłów oraz możliwość łatwego skalowania poziomego. Pozwala to na przetwarzanie i przechowywanie danych w czasie rzeczywistym bez obaw o utratę wydajności. Dodatkowo ich schemat danych jest elastyczny i dynamiczny, co pozwala architekturze na szybką adaptację do zmieniających się potrzeb aplikacji. Dzięki temu jest ona przygotowana na przyjęcie różnorodnych typów danych, które mogą być zarówno niestrukturyzowane, jak i półstrukturyzowane [5].

Architektura trójwarstwowa jest kluczowym wzorcem projektowym, który umożliwia efektywne zarządzanie, skalowalność oraz zapewnienie bezpieczeństwa aplikacji poprzez jasne rozdzielenie funkcjonalności na trzy główne warstwy: prezentacji, aplikacji oraz danych. Warstwy te pełnią unikalne role, co umożliwia niezależny rozwój, aktualizację oraz skalowanie poszczególnych komponentów systemu. Jasny podział obowiązków pomiędzy warstwami pozwala na wprowadzanie zmian w jednej z nich bez konieczności ingerencji w pozostałe, co zwiększa elastyczność i ułatwia utrzymanie systemu. Adaptacja różnych technologii, dostosowanych do specyficznych zadań każdej z warstw, przyczynia się do efektywności oraz wydajności całego systemu. Architektura trójwarstwowa stanowi solidny fundament dla projektowania aplikacji, umożliwiając tym samym efektywne zarządzanie złożonością, skalowanie oraz zapewnienie wysokiego poziomu bezpieczeństwa i niezawodności [2].

Ogólną koncepcję komunikacji pomiędzy warstwami przedstawia rysunek 4.2.



4.2. Komunikacja pomiędzy warstwami w architekturze  
Źródło: <https://www.linkedin.com/pulse/what-three-tier-architecture-hasindu-sithmin/>

#### 4.1.2. Architektury systemów Big Data

W obecnej erze cyfrowej dane stały się nieodzownym zasobem dla firm wszelkich rozmiarów, co podkreśla potrzebę wyboru odpowiedniej architektury przetwarzania danych. Dwa powszechnie uznawane podejścia w tej dziedzinie to architektury Lambda i Kappa, zapewniające efektywne zarządzanie dużymi i złożonymi zbiorami danych. Stały wzrost ilości danych gromadzonych w firmach otwiera nowe możliwości odkrywania nieznanych wcześniej zależności i wzorców. Przetwarzanie dużych, nieustrukturyzowanych zbiorów danych wymaga jednak stosowania zaawansowanych technologicznie rozwiązań. Pierwszą rewolucję w przetwarzaniu danych było wprowadzenie koncepcji MapReduce, popularnej w ekosystemie Hadoop. Polega ona na rozdzieleniu procesu przetwarzania danych na wiele węzłów (komputerów), z których każdy przetwarza mniejszą część całego zbioru danych, co umożliwia obliczenia rozproszone. Hadoop 1.0 był jedną z pierwszych platform do przetwarzania dużych zbiorów danych w trybie wsadowym (ang. *batch processing*), jednak proces ten mógł trwać od kilku godzin do nawet kilku dni [4].

Z czasem potrzeby biznesowe zaczęły się zmieniać, a technologie musiały nadążyć za nowymi wymaganiami. Oprócz zdolności do przetwarzania dużych zbiorów danych w trybie wsadowym, zauważono korzyści płynące z przetwarzania danych w czasie rzeczywistym (ang. *real-time processing*). W odpowiedzi na te potrzeby pojawiły się projekty takie jak Apache Kafka i Spark Streaming, które umożliwiają przetwarzanie danych na bieżąco. Niemniej jednak wciąż niezaspokojona pozostała potrzeba związana z zapewnieniem wystarczającej wydajności do szybkiego przetwarzania zarówno napływających danych, jak i dużych historycznych zbiorów. Właśnie w tym celu opracowane zostały architektury Lambda i Kappa. Architektura Lambda dzieli przetwarzanie danych na dwie równoległe warstwy: wsadową i czasu rzeczywistego, co pozwala na elastyczne zarządzanie danymi. Z kolei architektura Kappa upraszcza ten proces, eliminując potrzebę dwóch oddzielnych ścieżek i przetwarzając dane wyłącznie w czasie rzeczywistym [2].

Architektura Big Data została zaprojektowana w celu obsługi gromadzenia, przetwarzania i analizy danych, które ze względu na swój rozmiar lub złożoność, przekraczają możliwości tradycyjnych systemów baz danych. Próg wejścia do zagadnienia Big Data jest bardzo wysoki, ponieważ może obejmować zarządzanie setkami gigabajtów do nawet setek terabajtów danych, zależnie od potrzeb użytkownika oraz dostępnych narzędzi. Kluczową cechą architektury Big Data jest umiejętność ekstrakcji wartościowych informacji z danych za pomocą zaawansowanych technik analitycznych. W ostatnich latach w środowisku zarządzania danymi zaszły znaczące zmiany, które obejmują obniżenie kosztów przechowywania danych oraz wprowadzenie nowych metod ich zbierania. Nie bez znaczenia pozostaje również zróżnicowane

tempo napływu danych. Podczas gdy jedne mogą napływać szybko i wymagają ciągłej analizy, inne napływają w dużych ilościach, często obejmując dekady danych historycznych [4].

Rozwiązań opartych na architekturze Big Data zwykle obarczone są przynajmniej jednym z następujących typów obciążień:

- przetwarzanie wsadowe danych w trybie offline,
- przetwarzanie danych w czasie rzeczywistym podczas ich przesyłania,
- interaktywne eksplorowanie i analizowanie danych przez użytkownika,
- wykorzystanie danych w analizie predykcyjnej oraz w uczeniu maszynowym.

Mimo to architektury Big Data znajdują szczególną zastosowanie w przypadkach, gdy niezbędne jest:

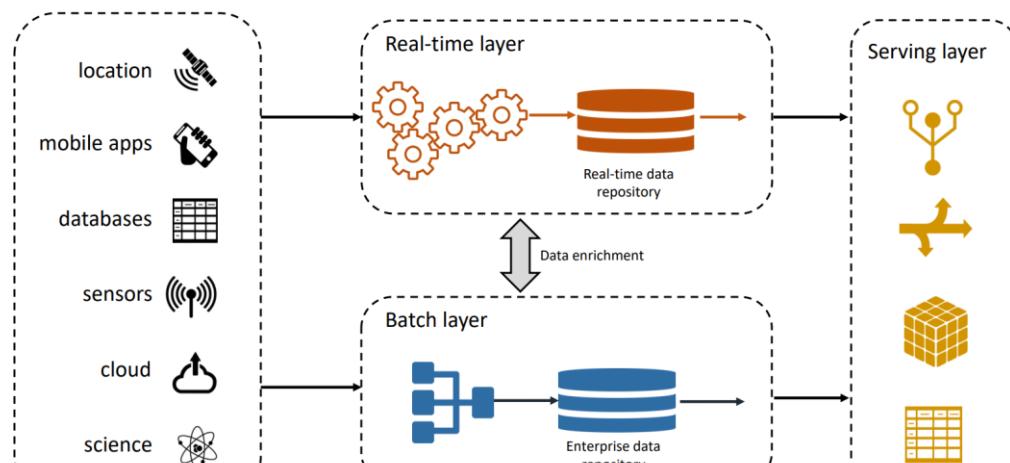
- przechowywanie i przetwarzanie dużych ilości danych, których rozmiar przekracza możliwości tradycyjnych baz danych,
- przekształcanie danych niestrukturyzowanych i półstrukturyzowanych w dane gotowe do analizy i raportowania,
- przechwytywanie, przetwarzanie i analizowanie danych z różnych źródeł w czasie rzeczywistym lub z minimalnym opóźnieniem.

Architektura Lambda stanowi zaawansowany wzorzec architektoniczny, opracowany z myślą o efektywnym zarządzaniu i przetwarzaniu dużych zbiorów danych. Jej unikalność polega na zdolności do równoczesnego obsługiwania zarówno przetwarzania wsadowego (ang. *batch processing*), jak i przetwarzania w czasie rzeczywistym (ang. *real-time processing*), co jest kluczowe dla elastycznego reagowania na zróżnicowane potrzeby i wymagania dotyczące danych w przedsiębiorstwach. Centralnym elementem architektury Lambda jest Warstwa Ingestii Danych, która pełni fundamentalną rolę w procesie zbierania i przygotowywania danych do dalszego przetwarzania. Odpowiada ona nie tylko za wysoko skalową część systemu, dostosowującą się dynamicznie do zmieniających się obciążzeń, ale również za odporność na awarie zapewniającą nieprzerwaną ciągłość operacyjną oraz za mechanizmy odzyskiwania danych po ewentualnych awariach. Warstwa ta obsługuje wielowątkowe przetwarzanie oraz szybką transformację danych do formatu wymaganego przez kolejne etapy przetwarzania, co jest kluczowe dla zapewnienia wysokiej jakości danych analizy. Drugim istotnym komponentem jest warstwa przetwarzania wsadowego, odpowiedzialna za dokładne czyszczenie, modelowanie oraz przetwarzanie surowych danych. Wykorzystuje ona mechanizmy umożliwiające ponowne przetworzenie partii danych oraz zaawansowane algorytmy poprawiające jakość danych. Warstwa ta jest nieodzowna dla uzyskania wysokiej precyzji w analizach i prognozach. Przetwarzanie w czasie rzeczywistym jest realizowane przez dedykowaną warstwę przetwarzania w czasie rzeczywistym, która obsługuje szybkie operacje na strumieniach danych, tworząc

odpowiednie modele danych i zapewniając wysoką wydajność operacyjną. Kluczową cechą tej warstwy jest separacja od Warstwy Ingestii Danych, co umożliwia niezależne zarządzanie i skalowanie obiema ścieżkami przetwarzania. Jest to istotne w kontekście dynamicznych potrzeb biznesowych. Warstwa Przechowywania Danych odpowiada za efektywne zarządzanie dużymi wolumenami danych oraz obsługę różnorodnych operacji, zarówno seryjnych, jak i losowych. Struktura warstwowa umożliwia optymalne zarządzanie zasobami oraz zapewnia elastyczność w zarządzaniu różnorodnymi typami danych, co jest niezbędne dla złożonych potrzeb analitycznych i operacyjnych. Ostatnia warstwa, Warstwa Usługowa, dostarcza przetworzone dane do aplikacji końcowych, zapewniając zgodność z ustalonymi kontraktami danych oraz reagując na potrzeby aplikacji w czasie rzeczywistym. Eластyczność i reaktywność są niezbędne do zapewnienia szybkiego dostępu do danych oraz wsparcia dla różnorodnych aplikacji biznesowych [7].

Architektura Lambda reprezentuje zaawansowane podejście do przetwarzania dużych zbiorów danych, które integruje przetwarzanie wsadowe i przetwarzanie w czasie rzeczywistym. Jest to rozwiązanie, które umożliwia organizacjom na efektywne zarządzanie i analizowanie danych o różnorodnym charakterze i pochodzeniu, odpowiadając tym samym na rosnące potrzeby w zakresie analizy danych i generowania wartości biznesowej [7].

Architektura Lambda dla systemów Big Data została przedstawiona na rysunku 4.3.



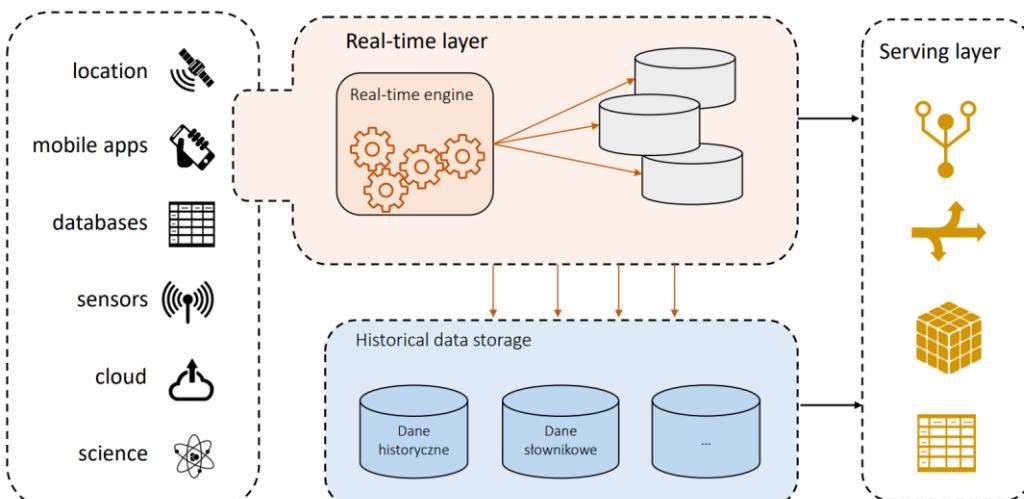
4.3. Architektura Lambda  
Źródło: <https://www.erp-view.pl>

Architektura Kappa stanowi zaawansowane podejście architektoniczne do przetwarzania dużych strumieni danych w czasie rzeczywistym. Opiera się na uproszczeniu tradycyjnej struktury (charakterystycznej dla architektury Lambda) poprzez eliminację warstwy przetwarzania wsadowego,. Główną cechą wyróżniającą Architekturę Kappa jest skupienie się wyłącznie na warstwie przetwarzania w czasie rzeczywistym (ang. *speed layer*), co pozwala natychmiastowo reagować na napływające dane, a także szybko generować wyniki.

W przetwarzaniu danych Architektura Kappa kładzie duży nacisk na ciągłość i szybkość operacji. Dane są analizowane bezpośrednio po ich przyjęciu, co minimalizuje opóźnienia i umożliwia operowanie na danych niemalże w czasie rzeczywistym. Implementacja tej architektury jest zalecana w aplikacjach, które wymagają natychmiastowej reakcji na zmieniające się warunki rynkowe, śledzenie trendów w czasie rzeczywistym oraz monitorowanie bezpieczeństwa. Architektura Kappa często wykorzystuje zaawansowane narzędzia i platformy, takie jak Apache Kafka (do zarządzania strumieniami danych) oraz Apache Storm lub Apache Flink (do ich przetwarzania w czasie rzeczywistym). Te technologie pozwalają na skalowanie infrastruktury w zależności od potrzeb użytkowników oraz zapewniają wysoką niezawodność i odporność na awarie. Zaletą tej architektury jest eliminacja potrzeby zarządzania dwoma oddzielnymi kodami i logiką dla warstw wsadowej i czasu rzeczywistego, co znaczaco upraszcza złożoność systemu. W porównaniu do Architektury Lambda, Kappa może być bardziej efektywna w środowiskach, gdzie dominuje przetwarzanie w czasie rzeczywistym i nie ma potrzeby przechodzenia przez kompleksowe etapy przeliczania i przetwarzania wsadowego. Skuteczne wdrożenie Architektury Kappa wymaga starannego planowania i optymalizacji infrastruktury, aby sprostać wymaganiom co do szybkości, niezawodności i skuteczności przetwarzania danych. Architektura Kappa została przedstawiona na rysunku 4.4 [7].

Architektura Lambda jest bardziej kompleksowym podejściem, pozwalającym na oddzielne zarządzanie wsadowymi strumieniami danych, co jest preferowane w budowie rozbudowanych jezior danych (ang. *Data Lake*). Architektura Kappa, mimo swojej prostoty, może być odpowiednia dla aplikacji wymagających szybkiej i efektywnej analizy strumieniowej danych, bez konieczności przetwarzania wsadowego. Wybór odpowiedniej architektury zależy od konkretnych potrzeb biznesowych [3].

Obie architektury zostały bardziej szczegółowo opisane w pracy inżynierskiej.



4.4. Architektura Kappa  
Źródło: <https://www.erp-view.pl>

Architektura Medalionu to wzorzec projektowy używany do logicznego organizowania danych w Lakehouse, mający na celu stopniowe i progresywne poprawianie struktury i jakości danych w miarę ich przepływu przez kolejne warstwy architektury od warstwy Brązowej przez Srebrną do Złotej. Architektura Medalionu jest nazywana także „wielokrotną” architekturą. Architektura ta została przedstawiona na rysunku 4.5 [11].



4.5. Architektura Medalionu  
Źródło: <https://www.databricks.com>

Databricks oferuje narzędzia takie jak Delta Live Tables (DLT), które umożliwiają użytkownikom natychmiastowe budowanie potoków danych z tabelami Brązowymi, Srebrnymi i Złotymi. Dzięki tabelom strumieniowym i materializowanym widokom, użytkownicy mogą tworzyć strumieniowe potoki DLT oparte na Apache Spark, które są inkrementalnie odświeżane i aktualizowane.

Użycie architektury Lakehouse przynosi następujące korzyści:

- prosty model danych, łatwy do zrozumienia i wdrożenia,
- możliwość przyrostowego ETL,
- możliwość odtworzenia tabel z surowych danych w dowolnym momencie,
- transakcje ACID.

Architektura Medalionu zapewnia stopniowe oczyszczanie i przygotowanie danych w miarę ich przepływu przez kolejne warstwy Lakehouse. Zastosowanie warstw Brązowej, Srebrnej i Złotej pozwala na efektywne zarządzanie danymi, zapewniając ich wysoką jakość, przejrzystość i dostępność dla różnych potrzeb biznesowych. Architektura Lakehouse łączy najlepsze cechy jezior danych i hurtowni danych, tworząc skalową i wydajną platformę gotową do zaawansowanej analizy i podejmowania decyzji biznesowych [10].

#### 4.1.3. Jakość danych

W procesie analizy istotną rolę odgrywa jakość danych, szczególnie tych generowanych w czasie rzeczywistym. Dane napływające z urządzeń IoT do magazynów danych zawierają bardzo dużą ilość tak zwanych szumów, a zadaniem procesowania danych jest ich oczyszczenie ze zbędnych z punktu widzenia analityków informacji. Proces ten został przedstawiony na rysunku 4.6.



4.6. Proces przetwarzania danych  
Źródło: <https://www.databricks.com>

Warstwa Brązowa to pierwsza warstwa, która zawiera surowe dane. Jest ona miejscem, do którego lądują wszystkie dane z zewnętrznych systemów źródłowych. Struktury tabel w tej warstwie odpowiadają strukturze tabel w systemie źródłowym wraz z dodatkowymi kolumnami metadanych, które rejestrują datę/godzinę ładowania, ID procesu itp. Zadaniem tej warstwy jest szybkie uchwycenie zmian danych i zapewnienie historycznego archiwum źródła, rodowodu danych oraz możliwości ich ponownego przetworzenia bez konieczności zaczytywania z systemu źródłowego. Dane magazynowane w tej warstwie są ustrukturyzowane w ten sam sposób co dane zapisane na urządzeniach IoT. Brązowy etap służy jako początkowy punkt pozyskiwania i przechowywania danych. Są one zapisywane bez przetwarzania lub transformacji. Przykładem surowych danych jest zapisywanie logów z aplikacji w rozproszonym systemie plików lub strumieniowe przesyłanie zdarzeń z Kafki [10].

Drugą warstwą jest Warstwa Srebrna. Jej zadaniem jest oczyszczenie i ujednolicenie danych pochodzących z warstwy Brązowej na tyle, by zapewnić przedsiębiorstwu wgląd we wszystkie kluczowe jednostki biznesowe i transakcje (np. główni klienci, sklepy, niepowielone transakcje i tabele krzyżowych odniesień).

Warstwa Srebrna łączy dane z różnych źródeł w widoku przedsiębiorstwa i umożliwia samoobsługową analitykę dla raportów zaawansowanej analityki. Ponadto służy ona jako źródło dla analityków działowych, inżynierów danych i analityków danych, aby mogli oni dalej tworzyć projekty i analizy w celu odpowiedzi na problemy biznesowe w warstwie Złotej. Transformacje tutaj powinny być jedynie lekkimi modyfikacjami, a nie agregacjami lub wzbogaceniami. Dane przedsiębiorstwa mogą zostać ustANDARDYZOWANE w celu ujednolicenia konwencji nazewnictwa lub podzielenia pojedynczego strumienia na wiele tabel.

Warstwa Złota jest warstwą finalną, w której docelowo przechowywane mają być tabele zawierające dane przetworzone na poziomie biznesowym. Dane w warstwie Złotej Lakehouse są zazwyczaj organizowane w bazy danych gotowe do implementacji do projektów. Warstwa Złota jest przeznaczona do raportowania i używa modeli danych z mniejszą liczbą złączeń, bardziej wartościowych i zoptymalizowanych pod kątem odczytu. Na tym etapie stosowana jest ostateczna warstwa transformacji danych i reguł jakości danych. Warstwa prezentacji końcowych projektów, takich jak analityka klientów, analityka jakości produktów, analityka zapasów, segmentacja klientów, rekomendacje produktów, analityka marketingu/sprzedaży itp. jest produktem końcowym tej warstwy. W Warstwie Złotej znaleźć można dużą ilość modeli danych opartych na schematach gwiazdy (w stylu Kimballa) lub hurtowni danych (w stylu Inmona).

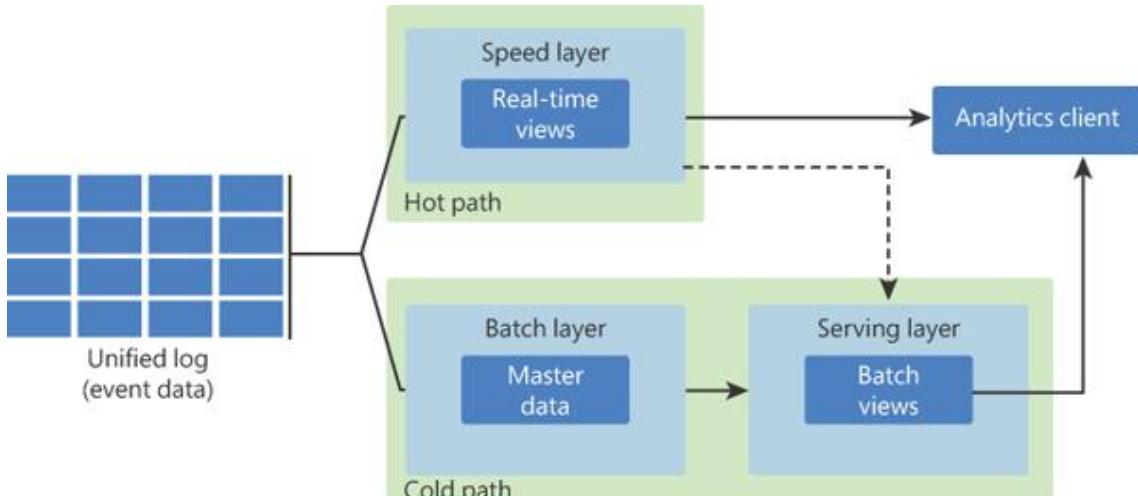
## 4.2. Cold Path i Hot Path w Przetwarzaniu Danych

### 4.2.1. Koncepcje Cold Path i Hot Path

Badania nad koncepcją gorącej, ciepłej i zimnej ścieżki (ang. *Hot Path*, *Warm Path* i *Cold Path*) danych są istotne w kontekście skutecznego zarządzania danymi w różnych kontekstach biznesowych, w tym w obszarze Internetu Rzeczy (IoT) oraz przetwarzania danych w chmurze. Koncepcja ta odnosi się do strategii przechowywania i przetwarzania danych w zależności od ich aktualności, charakterystyki oraz wymagań biznesowych. Gorąca ścieżka danych, będąca przedłużeniem koncepcji przetwarzania w czasie rzeczywistym, odgrywa kluczową rolę w obszarach, gdzie natychmiastowa dostępność informacji jest priorytetem. Dane przetwarzane w gorącej ścieżce są dostępne niemal natychmiast i wykorzystywane do operacji, które wymagają błyskawicznej reakcji, takich jak monitorowanie awaryjne czy alertowanie. Z kolei ciepła ścieżka koncentruje się na przechowywaniu i analizie najnowszych danych, które choć nie wymagają natychmiastowej reakcji, są nadal istotne dla operacji biznesowych. W porównaniu do gorącej ścieżki proces przetwarzania w ścieżce ciepłej może być bardziej złożony i obejmować analizę danych z prędkością zbliżoną do czasu rzeczywistego. Szczególnie istotne jest to dla przypadków biznesowych, które chcą śledzić na bieżąco zmiany w danych oraz podejmować na ich podstawie szybkie decyzje [12].

Zimna ścieżka danych, nazywana także długoterminowym przechowywaniem danych, skupia się na przechowywaniu dużych ilości danych historycznych oraz wykonywaniu bardziej zaawansowanych analiz, które nie wymagają natychmiastowej reakcji. Jest to kluczowe dla biznesów, które chcą analizować długoterminowe trendy, wykrywać wzorce oraz prognozować przyszłe tendencje na podstawie danych z przeszłości [13].

Wybór odpowiedniej ścieżki danych zależy od wielu czynników, takich jak charakterystyka danych, wymagania biznesowe oraz dostępność technologii. Kluczowym wyzwaniem jest skuteczne zbalansowanie potrzeb operacyjnych z możliwościami technologicznymi oraz kosztami przechowywania i przetwarzania danych. Dlatego też badania nad koncepcją gorącej, ciepłej i zimnej ścieżki danych mają istotne znaczenie dla efektywnego zarządzania danymi oraz wykorzystania ich potencjału analitycznego dla potrzeb biznesowych. Przetwarzanie danych w zależności od wybranej ścieżki zostało przedstawione na rysunku 4.7 [12].



4.7. Ścieżka gorąca oraz zimna w Przetwarzaniu Danych  
Źródło: <https://microsoft.com>

#### 4.2.2. Zastosowania Cold Path i Hot Path w Systemach Big Data

Systemy Big Data często wykorzystują koncepcje gorącej ścieżki danych do efektywnego zarządzania i przetwarzania dużych ilości danych. Gorąca ścieżka danych jest wykorzystywana do przetwarzania w czasie rzeczywistym, co umożliwia natychmiastową analizę danych po ich przybyciu. Jest to szczególnie ważne dla zastosowań, które wymagają szybkiej reakcji, takich jak monitorowanie i alertowanie w przypadku wykrycia nietypowych lub krytycznych zdarzeń. Analiza danych wrażliwych na opóźnienia, takich jak dane transakcyjne w handlu elektronicznym czy dane telemetryczne w systemach IoT, również korzysta z gorącej ścieżki. Zimna ścieżka danych służy do długoterminowego przechowywania dużych ilości danych historycznych, które nie wymagają natychmiastowej analizy. Takie dane są przechowywane na długie okresy i wykorzystywane do analizy długoterminowych trendów oraz tworzenia prognoz. Ponadto zimna ścieżka umożliwia zaawansowaną analizę danych (w tym identyfikację wzorców), wykrywanie anomalii oraz prognozowanie przyszłych tendencji przy użyciu złożonych algorytmów uczenia maszynowego i analiz predykcyjnych. Ścieżka pośrednia, czyli ciepła ścieżka danych, może służyć do przechowywania danych o średnim terminie, które choć nie wymagają natychmiastowej analizy, nadal są stosunkowo świeże i użyteczne. Analiza danych z ograniczonym opóźnieniem jest możliwa właśnie dzięki ciepłej ścieżce, która pozwala na szybką analizę trendów i wykrywanie wzorców w danych. Zastosowania gorącej i zimnej ścieżki danych w systemach Big Data są zróżnicowane i zależą od konkretnych wymagań biznesowych oraz charakterystyki danych. Kluczowe jest efektywne wykorzystanie obu ścieżek w celu zapewnienia kompleksowego zarządzania danymi oraz wykorzystania ich potencjału analitycznego dla potrzeb biznesowych [13].

Doboru ścieżki należy dokonać zależności od rodzaju posiadanych danych oraz celu, jaki chce się osiągnąć. Poniżej przedstawiono przypadki, dla których dedykowana jest dana ścieżka.

- Gorąca ścieżka
  - Przetwarzanie lub wyświetlanie danych w czasie rzeczywistym.
  - Operacje alertów i przesyłania strumieniowego w czasie rzeczywistym.
  - Systemy monitorowania zdrowia i wydajności serwerów, sieci oraz aplikacji wymagające natychmiastowego wykrywania i reagowania na problemy.
  - Systemy handlu algorytmicznego analizujące rynki finansowe w czasie rzeczywistym w celu umożliwienia natychmiastowego podejmowania decyzji inwestycyjnych.
  - Systemy zarządzania magazynem, które w czasie rzeczywistym wykorzystują dane do optymalizacji operacji logistycznych.
- Ciepła ścieżka
  - Przechowywanie lub wyświetlanie tylko ostatniego podzbioru danych, na którym wykonywane są niewielkie operacje analityczne i przetwarzania wsadowego.
  - Wykorzystanie danych z ostatnich dni do tworzenia modeli predykcyjnych, które pomagają przewidywać przyszłe zdarzenia lub potrzeby.
  - Analiza danych z ostatnich interakcji użytkowników w celu dostosowania treści lub rekomendacji w aplikacjach internetowych i mobilnych.
- Zimna ścieżka
  - Długoterminowe przechowywanie danych, na których wykonywane są czasochłonne analizy i przetwarzanie wsadowe.
  - Duże ilości danych historycznych, które nie wymagają natychmiastowej analizy [12].

## 4.3. Apache Kafka & Apache ZooKeeper

### 4.3.1. Architektura Apache Kafka oraz Apache ZooKeeper

W dzisiejszych czasach informacje w czasie rzeczywistym są nieustannie generowane przez różne aplikacje i potrzeba łatwego sposobu na niezawodne i szybkie przekazywanie ich do wielu typów odbiorców. Często aplikacje produkujące oraz konsumujące informacje są od siebie oddalone i niedostępne. Prowadzi to czasami do konieczności ponownego tworzenia połączeń w celu zapewnienia punktu integracji między nimi. Potrzebny jest więc mechanizm umożliwiający płynną integrację informacji między producentami i konsumentami, który eliminowałby konieczność modyfikacji aplikacji po obu stronach. W obecnej erze Big Data pierwszym wyzwaniem jest zebranie danych, gdyż jest ich ogromna ilość, a drugim – ich analiza. Analiza ta zazwyczaj obejmuje następujące rodzaje danych:

- dane o zachowaniu użytkowników,
- śledzenie wydajności aplikacji,
- dane o aktywności w formie logów,
- komunikaty zdarzeń.

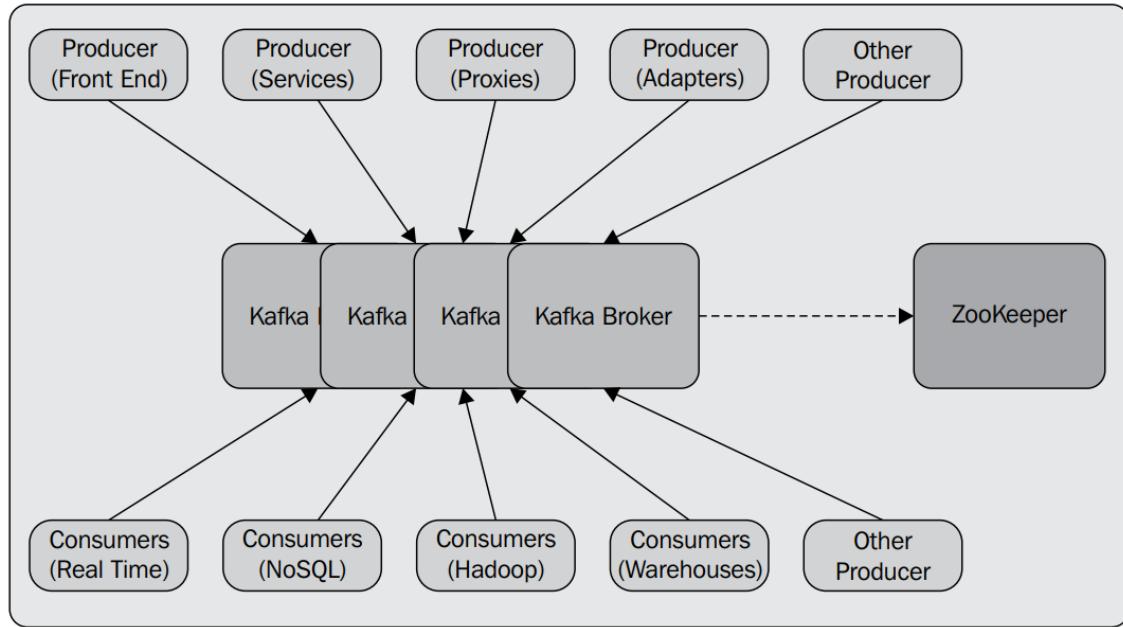
Apache Kafka jest rozwiązaniem dla problemów w czasie rzeczywistym każdej aplikacji, która zajmuje się obsługą ogromnych ilości informacji w czasie rzeczywistym i szybkim przekazywaniem ich do wielu konsumentów. Kafka zapewnia płynną integrację między informacjami producentów i konsumentów bez blokowania informacji producentów oraz bez konieczności posiadania przez nich wiedzy o tym, kim są końcowi konsumenci [15].

Apache Kafka to otwarty źródłowy, rozproszony system publikacji-subskrypcji wiadomości, zaprojektowany z myślą o następujących właściwościach:

- stała wydajność czasowa nawet przy wieloterabajtowych wolumenach przechowywanych wiadomości,
- wysoka przepustowość,
- partycjonowanie wiadomości oraz rozproszone konsumowanie ich w klastrze maszyn konsumenckich przy zachowaniu zamówienia w ramach partycji.
- łatwa integracja klientów z różnych platform (np. Java, .NET, PHP, Ruby, Python),
- widoczność w czasie rzeczywistym wiadomości produkowanych przez wątki producentów

Apache Kafka zapewnia rozwiązanie publikacji-subskrypcji w czasie rzeczywistym, które skutecznie radzi sobie z wyzwaniami związanymi z przetwarzaniem danych w czasie rzeczywistym, nawet gdy wolumen danych rośnie o rzędy wielkości bardziej niż pierwotnie zakładano. Kafka wspiera także równoległe ładowanie danych do systemów Hadoop.

Poniższy diagram (rysunek 4.8) pokazuje typowy scenariusz agregacji i analizy Big Data wspierany przez system wiadomości Apache Kafka:



4.8. Scenariusz analizy architektury Big Data w oparciu o Apache Kafka

Po stronie produkcyjnej istnieją różne rodzaje producentów, na przykład:

- frontendowe aplikacje webowe generujące logi aplikacji,
- producenci proxy generujący dzienniki analityki internetowej,
- adaptery producentów generujące dzienniki transformacji,
- usługi producenta generujące dzienniki śledzenia wywołań.

Z kolei po stronie konsumpcji można znaleźć konsumentów, takich jak:

- konsumenci offline, którzy konsumują wiadomości i przechowują je w Hadoop lub tradycyjnej hurtowni danych do analizy offline;
- konsumenci działający w czasie zbliżonym do rzeczywistego, którzy pobierają wiadomości i przechowują je w dowolnym magazynie danych NoSQL;
- konsumenci działający w czasie rzeczywistym, którzy filtryują wiadomości w bazie danych w pamięci (ang. *in-memory*) i wyzwalają zdarzenia alertów dla powiązanych grup [13].

Firmy w jakikolwiek sposób obecne przestrzeni internetowej generują dużą ilość danych.

Dane są jednym z nowszych surowców powstały dzięki internetowi. Zazwyczaj obejmują zdarzenia związane z aktywnością użytkowników: logowania, odwiedziny stron, aktywności w sieciach społecznościowych (polubienia, udostępnienia i komentarze) oraz metryki operacyjne i systemowe. Dane te są zazwyczaj obsługiwane przez rejestrowanie i tradycyjne rozwiązania do agregacji logów ze względu na ich wysoką przepustowość (miliony komunikatów na sekundę).

Są to realne rozwiązania do przesyłania danych z logowania do systemów analizy offline, takich jak Hadoop. Rozwiązania te są istotnym ograniczeniem dla budowania systemów przetwarzania w czasie rzeczywistym. Zgodnie z nowymi trendami w aplikacjach internetowych, dane dotyczące aktywności stały się częścią danych produkcyjnych i są wykorzystywane do uruchamiania analiz w czasie rzeczywistym. Wyróżnia się następujące typy analiz:

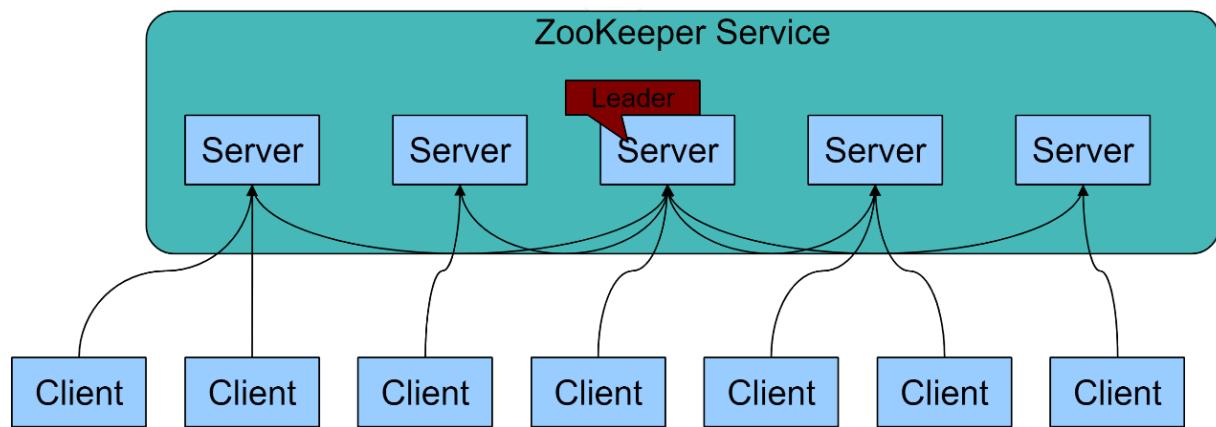
- wyszukiwanie oparte na trafności,
- rekomendacje oparte na popularności, współwystępowaniu lub analizie sentymentalnej,
- dostarczanie reklam do mas,
- bezpieczeństwo aplikacji internetowych przed spamem lub nieautoryzowanym skrobaniem danych [13].

Wykorzystanie w czasie rzeczywistym wielu zestawów danych zebranych z systemów produkcyjnych stało się wyzwaniem ze względu na ilość gromadzonych i przetwarzanych danych. Apache Kafka ma na celu ujednolicenie przetwarzania offline i online poprzez zapewnienie mechanizmu równoległego obciążenia w systemach Hadoop, a także możliwość partycjonowania zużycia na klaster maszyn w czasie rzeczywistym. Kafkę można porównać do Scribe lub Flume, ponieważ jest przydatna do przetwarzania danych strumieniowych aktywności; jednak z perspektywy architektury, jest ona bliższa tradycyjnym systemom przesyłania wiadomości, takim jak ActiveMQ czy RabbitMQ [14].

Apache ZooKeeper to projekt oprogramowania Apache Software Foundation, który zapewnia rozwiązanie typu open source dla różnych problemów związanych z koordynacją w dużych systemach rozproszonych. ZooKeeper został pierwotnie opracowany w Yahoo! Jako decentralizowana usługa koordynacji jest rozproszona i wysoce niezawodna. Działa na klastrze serwerów zwany ZooKeeper ensemble. Zarządzanie grupami, protokoły obecności i wybory liderów są implementowane przez usługę, aby twórcy aplikacji nie musieli wdrażać tych funkcjonalności samodzielnie. Co więcej prymitywy udostępniane przez ZooKeeper mogą zostać wykorzystywane przez aplikacje do tworzenia znacznie potężniejszych abstrakcji w celu rozwiązywania szerokiej gamy różnorodnych problemów.

Apache ZooKeeper został zaimplementowany w języku Java. Dostarczany jest z rozwiązaniami klienckimi w językach C, Java, Perl i Python. Współtworzone przez społeczność biblioteki klienckie są dostępne dla wielu języków, takich jak Go, Scala czy Erlang. Apache ZooKeeper jest usługą koordynacyjną dla aplikacji rozproszonych. Ma na celu rozwiązanie trudnych problemów związanych z koordynacją komponentów w aplikacjach rozproszonych. Realizuje to poprzez udostępnienie prostego, ale obszernego interfejsu prymitywów. Aplikacje mogą być projektowane na tych prymitywach, które są implementowane za pomocą interfejsów API ZooKeeper, aby rozwiązywać problemy synchronizacji rozproszonej, zarządzania

konfiguracją klastra, członkostwa w grupach i wiele innych. ZooKeeper sam w sobie jest replikowaną i rozproszoną aplikacją, zaprojektowaną do działania jako usługa, podobnie jak DNS lub inne centralne usługi. Na poniższym rysunku 4.9 przedstawiono usługę ZooKeeper. [13]



4.9. Usługa ZooKeeper i sposób, w jaki klienci łączą się z usługą

#### 4.3.2. Zastosowania Apache Kafka w Przetwarzaniu Strumieniowym

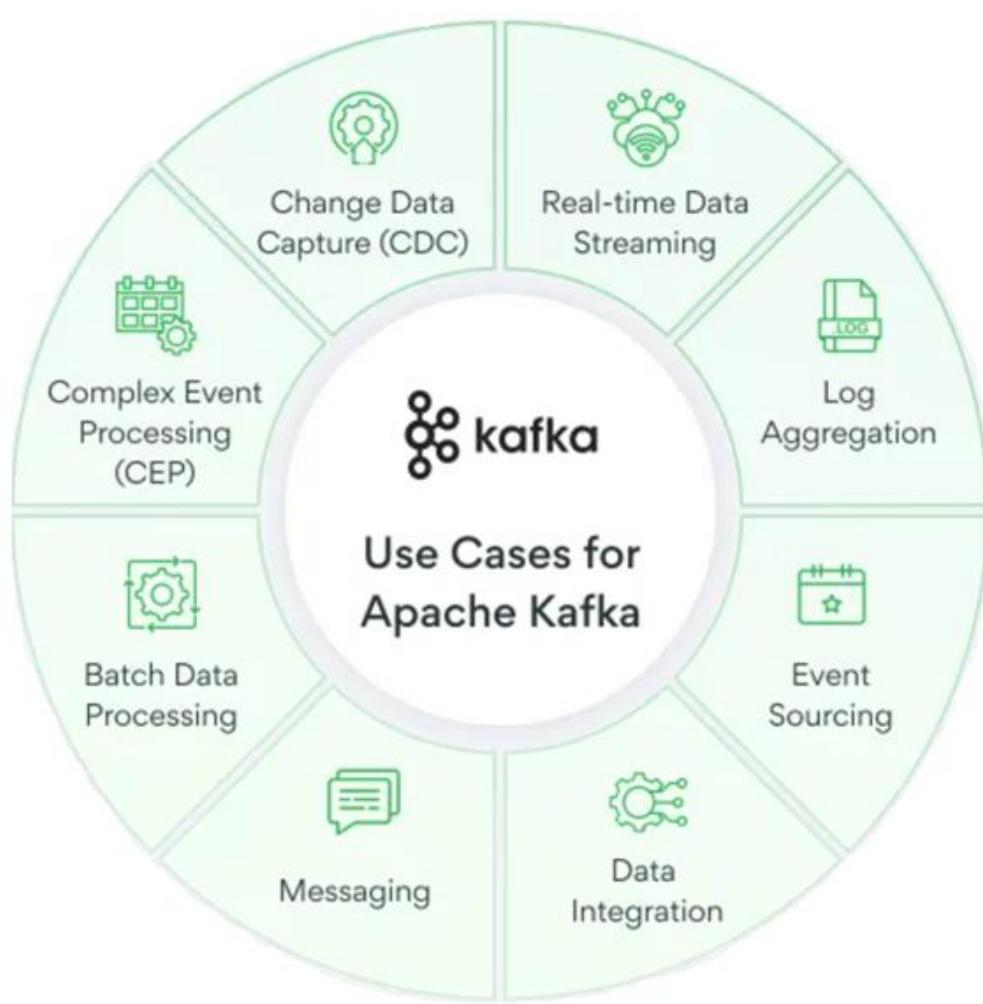
Apache Kafka może być skutecznie wykorzystana w przetwarzaniu strumieniowym na stronach internetowych, gdzie ciągłe zdarzenia dotyczące bezpieczeństwa, takie jak uwierzytelnianie użytkowników i autoryzacja dostępu do zasobów, muszą być monitorowane w czasie rzeczywistym, aby natychmiast reagować na wszelkie zagrożenia. Użycie tradycyjnych systemów przetwarzania danych wsadowego (np. Hadoop), które wymagają zbierania wszystkich danych przed ich przetworzeniem w celu identyfikacji wzorców, prowadziłoby do zbyt dużych opóźnień w wykrywaniu potencjalnych problemów z bezpieczeństwem aplikacji internetowej. Dlatego właśnie w takich scenariuszach istotne jest wykorzystanie przetwarzania danych w czasie rzeczywistym [15].

Kolejnym przypadkiem użycia jest analiza surowych danych kliknięć generowanych przez klientów podczas korzystania z witryny internetowej. Przechwytywanie i wstępne przetwarzanie tych kliknięć pozwala na uzyskanie cennych informacji dotyczących preferencji użytkowników. Wnioski z analiz mogą zostać później wykorzystane w kampaniach marketingowych oraz silnikach rekomendacyjnych, co umożliwiłoby pełniejszą analizę zachowań konsumentów. Chociaż duże ilości danych kliknięć przechowywanych w Hadoopie są przetwarzane przez zadania Hadoop MapReduce w trybie wsadowym, nie umożliwia to szybkiej analizy w czasie rzeczywistym. Apache Kafka w celu rozwiązywania różnych przypadków użycia może być zintegrowana z przetwarzaniem w czasie rzeczywistym za pomocą Storm oraz przetwarzaniem wsadowym za pomocą Hadoopa. Te integracje pozwalają na efektywne zarządzanie danymi strumieniowymi, zapewniając jednocześnie natychmiastową reakcję i kompleksowe przetwarzanie wsadowe spełniające różnorodne wymagania aplikacji opartych na danych [14].

Istnieje wiele firm korzystających w swoich przypadkach użycia z Apache Kafka. Wśród nich znajdują się:

- LinkedIn, gdzie Apache Kafka jest używany do strumieniowego przesyłania danych dotyczących aktywności i wskaźników operacyjnych.
- DataSift, w którym Kafka używana jest jako kolektor do monitorowania zdarzeń i jako narzędzie do śledzenia zużycia strumieni danych przez użytkowników w czasie rzeczywistym.
- Twitter, który używa Kafki jako części swojej infrastruktury Storm.
- Foursquare, gdzie Kafka stosowana jest do obsługi komunikacji online-to-online i online-to-offline oraz do integracji z systemami monitorowania i produkcji.
- Square, które używa Kafki jako magistrali do przenoszenia wszystkich zdarzeń systemowych przez różne centra danych. Obejmuje to metryki, dzienniki, niestandardowe zdarzenia itd. Po stronie konsumenta dane wyjściowe są przesyłane do Splunk, Graphite lub Esper, podobnych do alertów w czasie rzeczywistym.

Apache Kafka jest niezwykle przydatnym narzędziem w nowoczesnych aplikacjach i systemach, gdzie przetwarzanie danych odgrywa kluczową rolę w podejmowaniu szybkich decyzji biznesowych oraz dostarczaniu wartościowych analiz i prognoz na podstawie napływających danych. Możliwe przypadki użycia, w których sprawdzi się Apache Kafka, zostały przedstawione na rysunku 4.10 [14].



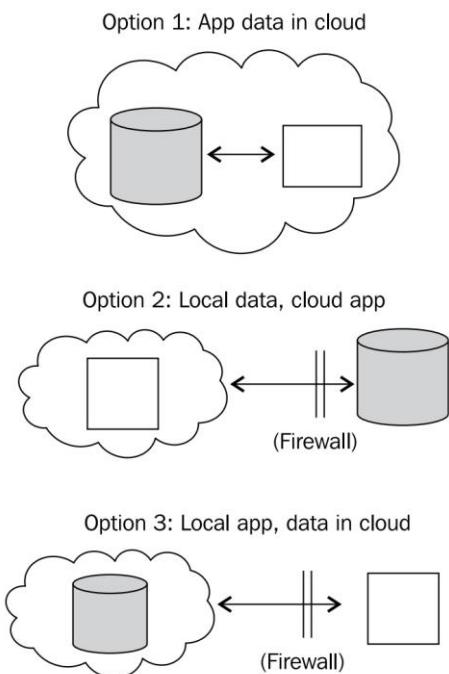
4.10. Przypadki użycia Apache Kafka  
Źródło: <https://kafka.apache.org>

## 4.4. Przetwarzanie danych w Chmurze

### 4.4.1. Microsoft Azure Cloud

Chmura obliczeniowa to termin, który w bardzo krótkim czasie awansował w branży informatycznej do czołówki najczęściej używanych. Amazon, Google i Microsoft (wśród wielu innych) – każda z nich oferuje usługi przetwarzania w chmurze. Obliczenia w chmurze polegają na przenoszeniu aplikacji dostępnych wewnętrznej sieci do przestrzeni dostępnej przez Internet (chmury). W zasadzie jest to wynajęcie wirtualnych maszyn w cudzym centrum danych z możliwością natychmiastowego skalowania, przełączania awaryjnego i synchronizacji danych. W przeszłości chęć posiadania aplikacji dostępnej przez Internet oznaczała potrzebę zbudowania strony internetowej z hostowaną bazą danych. Chmura obliczeniowa zmienia ten paradymat – aplikacja może być witryną internetową lub klientem zainstalowanym na lokalnym komputerze uzyskującym dostęp do wspólnego magazynu danych z dowolnego miejsca na świecie. Magazyn danych może istnieć wewnętrznej sieci lub być hostowany w chmurze [17].

Diagram umieszczony na poniższym rysunku 4.11 przedstawia trzy sposoby, w jakie chmura obliczeniowa może być wykorzystywana przez aplikację. W opcji pierwszej dane, jak i aplikacja, są hostowane w chmurze; druga opcja polega na hostowaniu aplikacji w chmurze, a danych lokalnie; trzecia natomiast to hostowanie danych w chmurze a aplikacji lokalnie [18].



4.11. Komunikacja pomiędzy chmurą obliczeniową a aplikacją

Zaletami takiego rozwiązania jest niski koszt początkowy. W przypadku chmury ktoś inny kupuje i instaluje serwery, przełączniki i zapory sieciowe. Oprócz sprzętu na poziomie

przedsiębiorstwa drogie są również licencje na oprogramowanie i plany asekuracyjne, nawet z umową zakupu. W przypadku większości usług w chmurze, w tym platformy Microsoft Azure, nie trzeba kupować oddzielnych licencji na systemy operacyjne czy bazy danych. W przypadku platformy Azure koszty obejmują licencje na system operacyjny Windows Azure i SQL Azure. Drugą zaletą takiego rozwiązania jest łatwiejsze odzyskiwanie danych po awarii i zarządzanie pamięcią masową. Zsynchronizowana pamięć masowa w wielu centrach danych, zlokalizowanych w różnych regionach tego samego kraju lub nawet w różnych krajach, ułatwia planowanie i sprawia, że odzyskiwanie danych po awarii staje się znacznie mniej problematyczne. Kolejną zaletą takiego rozwiązania jest migracja ze środowiska testowego do produkcyjnego, która jest znacznie uproszczona. Windows Azure umożliwia przetestowanie zaktualizowanej wersji naszej aplikacji w lokalnej piaskownicy (ang. *sandbox*). Migrację można przeprowadzić z dużym wyprzedzeniem, co umożliwia wykonanie migracji w ciągu dnia, a przełączenia o północy. Znane środowisko to kolejna zaleta rozwiązania korzystającego z obliczeń chmurowych. Jedną ze cech systemu Azure jest możliwość skonfigurowania go na wiele sposobów [17].

Chmura obliczeniowa brzmi wspaniale, ale nic nie jest idealne. Istnieją aspekty przetwarzania w chmurze, które wymagają kompromisów, a w niektórych przypadkach mogą nawet sprawić, że niewykonalne stanie się przeprowadzenie obliczeń chmurowych dla firmy. Wadą jest również mniejsza kontrola nad środowiskiem aplikacji. Rezygnacja z kontroli nad utrzymaniem zapór ogniwowych, serwerów i systemu operacyjnego może być kłopotliwa, szczególnie dla instytucji wrażliwych, takich jak służba zdrowia czy bankowość. Zastosowanie chmury powoduje, że dane i aplikacja przechowywane są w publicznie dostępnej przestrzeni. W przypadku wielu aplikacji internetowych koszty aplikacji w chmurze są wyższe niż w przypadku standardowego hostingu współdzielonego. W oparciu o ceny ogłoszone na PDC 2009, prosta aplikacja internetowa z pojedynczą instancją kosztowałaby około 100 USD/miesiąc. Dla porównania w przypadku standardowego hostingu współdzielonego kwota ta wyniosłaby około 5-20 USD/miesiąc. W większości przypadków hosting aplikacji w chmurze nie jest tak prosty jak zwykłe wdrożenie jej na zdalnym serwerze. W przypadku istniejących aplikacji wystąpić mogą znaczące zmiany, jak choćby zastąpienie lokalnych ciągów połączeń z architekturą zorientowaną na usługi lub wykorzystanie wysokowydajnej pamięci masowej (tabel i obiektów blob) zamiast systemu plików [17].

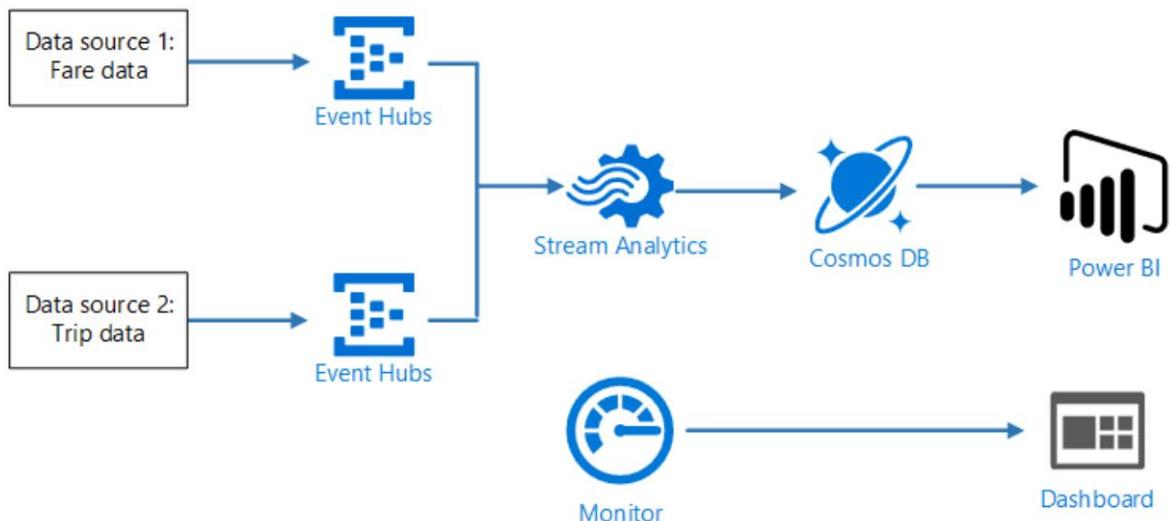
Microsoft Azure w rzeczywistości odwołuje się do palety usług oferowanych przez Microsoft. Każda z nich zawarta jest w Azure Fabric. Azure Fabric to zasadniczo infrastruktura sprzętowa i oprogramowanie, które monitorują i kontrolują sprzęt. Każdy serwer, każda zapora sieciowa, każdy moduł równoważenia obciążenia, usługi przełączania awaryjnego w przypadku awarii – wszystko to jest częścią Azure Fabric. W jego skład wchodzi także Azure Portal, z pomocą którego wdrażana jest aplikacja, tworzone są bieżące usługi i sprawdzana jest ich bezawaryjność.

Przetwarzanie strumieniowe to technologia Big Data, która pozwala na przetworzenie danych w czasie rzeczywistym w miarę ich napływu w krótkim czasie od momentu otrzymania danych.

Możliwymi komponentami architektury do przetwarzania strumieniowego są:

- Azure Stream Analytics – mechanizm analizy i przetwarzania zdarzeń w czasie rzeczywistym przeznaczony do analizowania i przetwarzania dużych ilości szybkich danych strumieniowych z wielu źródeł;
- HDInsight with Storm – odporny na błędy, rozproszony, i otwarty system obliczeniowy służący do przetwarzania strumieni danych w czasie rzeczywistym za pomocą Apache Hadoop;
- Apache Spark w usłudze Azure Databricks;
- Interfejsy API usługi Azure Kafka Stream;
- HDInsight z usługą Spark Streaming.

Na rysunku 4.12 przedstawiono przykładowe przetwarzanie strumieniowe w architekturze Azure [19].



4.12 Przepływ pracy przetwarzania strumieniowego  
Źródło: k21.academy.com

#### 4.4.2. Amazon Web Services (AWS)

Amazon Web Services (AWS) oferuje usługi przetwarzania danych, które spełniają różne potrzeby i realizują różne scenariusze. Wybór odpowiedniej usługi zależy od czynników takich jak ilość danych, wymagania dotyczące przetwarzania i pożądany rezultat.

Architektura Big Data, która korzystałaby z AWS do przetwarzania danych, powinna składać się z Amazon S3 (skrót od ang. *Simple Storage Service*). S3 to usługa obiektowej pamięci masowej często używana jako jezioro danych do przechowywania dużych ilości nieprzetworzonych danych. Drugim kluczowym elementem architektury jest AWS Glue, usługa ETL (Extract, Transform, Load) przeznaczona do przygotowania i ładowania danych do data lake'a lub hurtowni danych. AWS Glue umożliwia definiowanie logiki transformacji danych za pomocą skryptów ETL oraz harmonogramowanie zadań w zależności od częstotliwości aktualizacji danych. Następnym elementem jest Amazon EMR (Elastic MapReduce). Wykorzystywany jest on do przetwarzania dużych zbiorów danych przy użyciu popularnych framework'ów, takich jak Apache Spark i Apache Hadoop. Pomaga także przy wyborze odpowiednich typów oraz liczby instancji i wybór odpowiedniego rozmiaru klastra w oparciu o wymagania dotyczące przetwarzania danych. AWS Lambda, bezserwerowa usługa obliczeniowa do wykonywania kodu w odpowiedzi na zdarzenia, idealnie nadaje się do przetwarzania danych w małych, dyskretnych zadaniach. Amazon Redshift jest zarządzaną hurtownią danych do analiz i raportowania. Wymaga ona jednak optymalizacji dystrybucji danych, kluczy sortowania oraz zapytań. Ostatnimi dwoma elementami niezbędnymi w budowaniu architektury są Amazon Kinesis oraz Amazon Athena. Pierwsze z nich odpowiada za strumieniowe przetwarzanie danych w czasie rzeczywistym dla architektury takich jak analityka, monitorowanie i uczenie maszynowe. Amazon Athena to bezserwerowa usługa zapytań do analizy danych przechowywanych w Amazon S3 przy użyciu SQL. Służy ona do optymalizacji formatów danych (np. Parquet) i partycjonowania w celu poprawy wydajności zapytań i obniżenia kosztów. Architektura zbudowana z takich elektów pozwala na efektywne i skrojone na miarę rozwiązania przetwarzania danych w środowisku AWS, zapewniając skalowalność, wydajność i bezpieczeństwo [22].

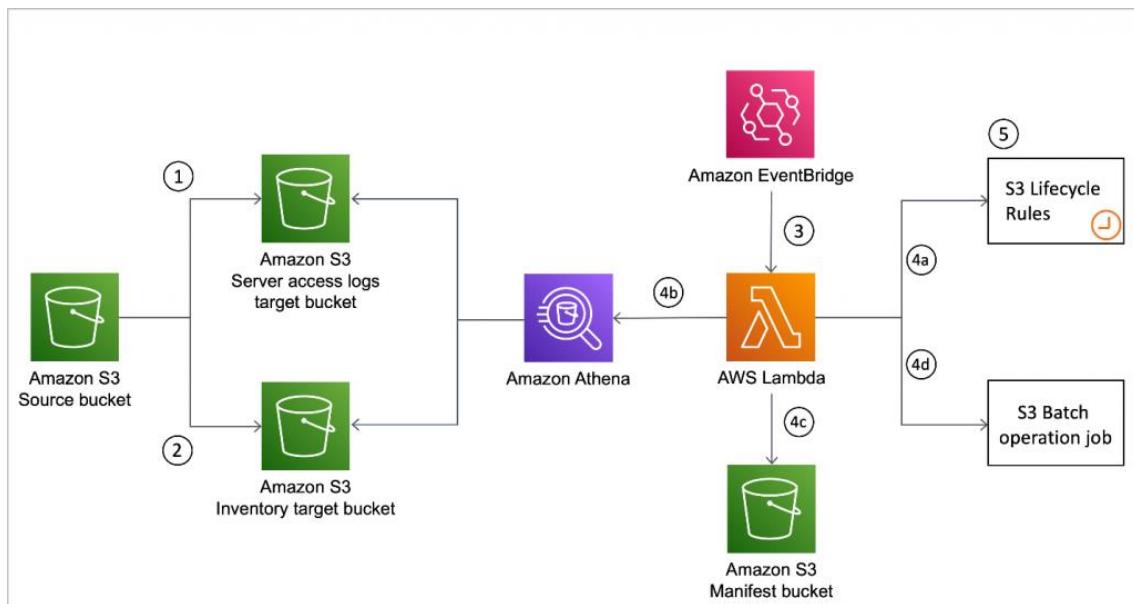
Amazon Kinesis Services i Amazon MSK to narzędzia wykorzystywane do przechwytywania i przechowywania danych strumieniowych. Dane te można przetwarzać sekwencyjnie i przyrostowo, rekord po rekordzie, lub w przesuwnych oknach czasowych. Przetworzone dane można wykorzystać do różnego typu analiz, w tym korelacji, agregacji, filtrowania i próbkowania. Informacje pochodzące z przetwarzania w czasie rzeczywistym zapewniają firmom wgląd w wiele aspektów ich działalności i aktywności klientów, takich jak wykorzystanie usług (do pomiaru lub fakturowania), aktywność serwerów, kliknięcia witryn internetowych oraz geolokalizacja urządzeń, osób i towarów fizycznych. Umożliwia im to

szynkowe reagowanie na zaistniałe zdarzenia. Przetwarzanie w czasie rzeczywistym wymaga użycia AWS Lambda, które jest w stanie przetwarzać dane bezpośrednio z AWS, IoT lub Amazon Kinesis Data Streams. Lambda umożliwia uruchamianie kodu bez konieczności udostępniania serwerów lub zarządzania nimi. Amazon Kinesis Client Library (KCL) to kolejny sposób na przetwarzanie danych z Amazon Kinesis Streams. KCL w zakresie grupowania przychodzących danych do dalszego przetwarzania zapewnia większą elastyczność niż Lambda. KCL można również wykorzystać do zastosowania obszernych transformacji i dostosowań w logice przetwarzania. Amazon Data Firehose to najprostszy sposób na załadowanie danych strumieniowych do AWS. Może on przechwytywać dane strumieniowe i automatycznie ładować je do Amazon Redshift, umożliwiając analizę w czasie zbliżonym do rzeczywistego za pomocą istniejących narzędzi BI i pulpitów nawigacyjnych. Amazon MSK to łatwy sposób na tworzenie i uruchamianie aplikacji do przetwarzania danych strumieniowych, które wykorzystują Apache Kafka. Zadania strumieniowe AWS Glue umożliwiają wykonywanie złożonych operacji ETL na danych strumieniowych. Strumieniowe zadania ETL w AWS Glue mogą pobierać dane ze źródeł strumieniowych, takich jak Amazon Kinesis Data Streams i Amazon MSK, czyścić i przekształcać te strumienie danych w locie oraz stale ładować wyniki do jezior danych S3, hurtowni danych lub innych magazynów danych. Podczas przetwarzania danych strumieniowych w zadaniu AWS Glue, użytkownik ma dostęp do pełnych możliwości Spark Structured Streaming w celu implementacji transformacji danych, takich jak agregacja, partycjonowanie i formatowanie, a także łączenie z innymi zestawami danych w celu wzbogacenia lub oczyszczania danych w celu łatwiejszej analizy [23].

Amazon Kinesis Services i Amazon MSK oferują wszechstronne narzędzia do przetwarzania danych strumieniowych w chmurze, umożliwiając firmom uzyskiwanie wglądu w czasie rzeczywistym w różne aspekty ich działalności. Usługi te zapewniają elastyczność i skalowalność w przechwytywaniu, przechowywaniu i przetwarzaniu danych strumieniowych, pozwalając firmom na skalowanie operacji zgodnie z potrzebami, bez konieczności zarządzania infrastrukturą.

Przetwarzanie danych w czasie rzeczywistym pozwala firmom szybko reagować na zmieniające się warunki, takie jak wzrost aktywności klientów, zmiany w geolokalizacji czy anomalie w danych serwerowych. Integracja i automatyzacja z AWS Lambda umożliwia bezserwerowe przetwarzanie danych, eliminując potrzebę zarządzania infrastrukturą. Amazon Kinesis Client Library (KCL) oferuje większą elastyczność w przetwarzaniu i transformacji danych, a Amazon Data Firehose upraszcza proces ładowania danych strumieniowych do innych usług AWS, takich jak Amazon Redshift, co pozwala na analizę danych niemal w czasie rzeczywistym. AWS Glue, dzięki zadaniom ETL, pozwala na kompleksowe przetwarzanie i transformację danych strumieniowych, umożliwiając firmom wykonywanie zaawansowanych operacji na danych, takich jak agregacja, partycjonowanie i łączenie z innymi zestawami danych.

Amazon MSK ułatwia tworzenie i zarządzanie aplikacjami wykorzystującymi Apache Kafka, co umożliwia firmom korzystanie z natywnych interfejsów API Apache Kafka do różnych zadań, takich jak zasilanie aplikacji uczenia maszynowego i analitycznych. Usługi te są kluczowe dla analiz w czasie rzeczywistym, które mogą obejmować korelacje, agregacje, filtrowanie i próbkowanie danych. Przetworzone dane dostarczają cennych informacji, które mogą zostać wykorzystywane do pomiaru wykorzystania usług, monitorowania aktywności serwerów, analiz kliknięć na stronach internetowych oraz śledzenia geolokalizacji. Podsumowując, Amazon Kinesis Services i Amazon MSK stanowią solidne fundamenty do budowy skalowalnych i efektywnych rozwiązań przetwarzania danych strumieniowych w chmurze, umożliwiając firmom uzyskiwanie natychmiastowych i cennych informacji o ich działalności. Na rysunku 4.13 zostało przedstawione zarządzanie danych z wykorzystaniem Amazon S3 [24].



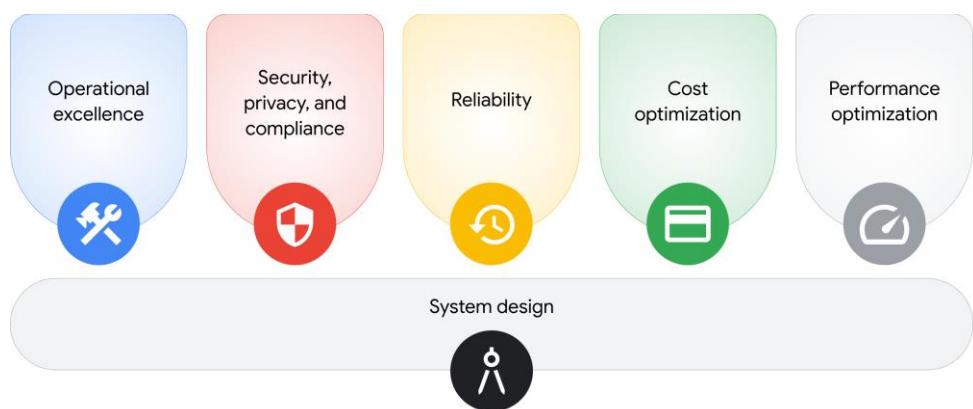
4.13. Przepływ pracy zarządzania danymi Amazon S3

Źródło: <https://aws.amazon.com>

#### 4.4.3. Google Cloud Platform

Korzystając z Google Cloud Architecture Framework, architekci, programiści, administratorzy i inni praktycy chmury mogą tworzyć i zarządzać topografią chmury, która wyróżnia się bezpieczeństwem, wydajnością, odpornością, wydajnością i opłacalnością. Ramy te ucieleśniają interpretację dobrze zaprojektowanej przez Google Cloud Platform struktury. Wytyczne projektowe w ramach Architecture Framework są dostosowane do szerokiego zakresu scenariuszy, w tym rozwoju aplikacji natywnych dla chmury, migracji obciążień lokalnych do Google Cloud, wdrożeń chmury hybrydowej i środowisk wielochmurowych [22].

Na rysunku 4.14 został przedstawiony podział Google Cloud Architecture Framework na sześć kategorii znanych również jako filary.



4.14. Architektura Google Cloud Platform  
Źródło: <https://cloud.google.com>

Projekt systemu jest podstawową kategorią. Opisuje ona najlepsze praktyki, zasady i zalecenia projektowe, które mają pomóc w projektowaniu architektury, komponentów, modułów, interfejsów i danych na platformie chmurowej spełniających wymagania systemowe. W tej kategorii można zrozumieć podstawowe założenia projektowania systemu na GCP. Pomaga ona również zdecydować, jaki typ architektury chmury będzie odpowiedni, a także zawiera ogólne wskazówki dotyczące wyboru zasobów Google Cloud i zarządzania nimi. Kolejną kategorią jest doskonałość operacyjna, która pokazuje, jak efektywnie korzystać z usług Google Cloud. Obejmuje ona sposób obsługi, nadzorowania i zarządzania systemami, które są wartościowe dla firmy. Zawiera również oferowane przez Google Cloud funkcje i produkty, które pomagają w osiągnięciu doskonałości operacyjnej. Stosowanie tej koncepcji kładzie podwaliny pod niezawodność. Doskonałość operacyjna osiągana jest poprzez ustanowienie podstawowych komponentów, takich jak automatyzacja, skalowalność i obserwonalność. W celu ochrony infrastruktury, aplikacji i danych opartych na chmurze zaprojektowano zbiór środków bezpieczeństwa, na które składa się bezpieczeństwo, prywatność i zgodność w chmurze.

Kategoria ta pokazuje, jak projektować i obsługiwać bezpieczne usługi w Google Cloud, a także jest źródłem wiedzy na temat funkcji i produktów, które pomagają w zapewnieniu bezpieczeństwa i zgodności. Niezawodność pokazuje, jak tworzyć i obsługiwać niezawodne usługi na platformie chmurowej. Ta kategoria obejmuje kluczowe elementy zarządzania niezawodną aplikacją w chmurze, takie jak SLI, SLO, SLA, skalowalność i wysoka dostępność. Optymalizacja kosztów przeniesienia obciążen IT do chmury może ułatwić wprowadzanie innowacji na dużą skalę, przyspieszyć dostarczanie funkcji i umożliwić dostosowanie się do zmieniających się wymagań klientów. Topologia zoptymalizowana pod kątem bezpieczeństwa, odporności, doskonałości operacyjnej, kosztów i wydajności jest niezbędna do przeniesienia bieżących obciążen lub wdrożenia aplikacji natywnych dla chmury. Architekci, programiści, administratorzy i inni praccy chmury mogą zoptymalizować koszty obciążen w Google Cloud za pomocą zaleceń projektowych i najlepszych praktyk opisanych w tej kategorii. Kategoria optymalizacji wydajności opisuje proces optymalizacji oraz najlepsze praktyki, których celem jest optymalizacja wydajności obciążen w Google Cloud. Dzięki temu firma może działać płynniej, bardziej zadowolić klientów, zarobić więcej pieniędzy i obniżyć wydatki [26].

Google Cloud Architecture Framework dostarcza narzędzi i wskazówek, które wspierają skuteczne zarządzanie chmurą, umożliwiając firmom osiąganie ich celów biznesowych w sposób zoptymalizowany pod kątem bezpieczeństwa, kosztów, wydajności i niezawodności [22].

## **5. Część projektowa**

Celem projektu była krytyczna analiza wybranych architektur systemów przetwarzania danych. Zostały zaprojektowane dwie architektury przetwarzania danych w systemach Big Data. Następnie wybrana jedna z nich i zaimplementowana pod kątem możliwości jej zastosowania w systemach meteorologicznych z analizą danych IoT w czasie rzeczywistym. Wybór padł na taki przykład, ponieważ stacje meteorologiczne posiadają wiele danych które można zbierać w czasie rzeczywistym lub zbliżonym do niego. Dane GPS z konkretnej lokalizacji geograficznej jak wilgotność w konkretnym miejscu, temperatura i tym podobne. Architektura taka w pełni pokazuje swoje możliwości właśnie w takich miejsca jak manufaktury, banki, obszar telekomunikacji, stacje pomiarowe ze względu na to że możemy zareagować bardzo szybko w momencie w którym pojawia się jakaś awaria, którą będziemy w stanie od razu zobaczyć na pulpity nawigacyjne Pulpity nawigacyjne są artefaktem czyli w kontekście IT finalnym produktem który zwraca architektura. Zakres działań projektowych obejmował stworzenie dwóch architektur, aby pokazać różnice pomiędzy nimi wybranie jednej z nich która bardziej pasuje do założeń celów biznesowych, a następnie jej implementacja na fikcyjnych danych, aby pokazać możliwości architektury do przetwarzania danych w systemach Big Data. Cześć projektowa została podzielona na pięć podrozdziałów dla bardziej szczegółowego i uporządkowanego opisania wykorzystania architektury dla przypadku biznesowego. Rozdział założenia ekosystemu do przetwarzania danych w czasie rzeczywistym skupia się na wybraniu przypadku użycia dla naszej architektury. Wybór padł na wcześniej wspomniana stacje meteorologiczna i zagadnienie zostało opisane w sposób teoretyczny zawiera rozważania teoretyczne jakie dane oraz w jaki sposób są generowane i magazynowane. Opisane zostały źródła danych oraz jakie funkcjonalności powiewnie zawierać system. Drugi podrozdział skupił się na przedstawieniu i zaprojektowaniu dwóch różnych architektur, które można by było zaimplementować dla stacji meteorologicznej. Następnie została wybrana jedna z architektura opisana bardzo szczegółowo w raz z informacją jaki komponent architektury odpowiada za konkretna funkcje i jakie ma zadanie. Przedostatni rozdział pokazał budowę całej architektury wraz z prezentacja jej możliwości dla celów projektowych zostały wykorzystane fikcyjne dane, ponieważ celem projektu była analiza architektur przetwarzania danych w systemach Big Data, a budowa i prezentacja możliwości architektury miała na celu wizualizacje. Ostatni rozdział skupił się na podsumowaniu w którym zawarte zostały mocne i słabe strony każdego z rozwiązań.

## **5.1. Założenia ekosystemu do przetwarzania danych w czasie rzeczywistym**

### **5.1.1. Biznesowe podstawy projektu**

W dzisiejszych czasach precyzyjne i aktualne dane pogodowe są niezbędne w wielu dziedzinach życia, od rolnictwa przez transport, aż po zarządzanie kryzysowe i ochronę środowiska. Tradycyjne metody zbierania danych meteorologicznych często okazywały się niewystarczające, szczególnie w kontekście potrzeby monitorowania lokalnych mikroklimatów i szybkiego reagowania na zmiany warunków pogodowych.

Architektura do przetwarzania danych w systemach Big Data realizuje najważniejsze wymagania projektowe dla stacji meteorologicznej z analizą danych IoT w czasie rzeczywistym. Architektura ma na celu zapelenienie tej luki poprzez wykorzystanie nowoczesnych technologii Internetu Rzeczy, które umożliwiały zbieranie, przetwarzanie i analizowanie danych pogodowych w czasie rzeczywistym. Stacja meteorologiczna są wyposażone w szereg czujników IoT, które mierzą różne parametry środowiskowe, takie jak temperatura, wilgotność, ciśnienie atmosferyczne, prędkość i kierunek wiatru oraz poziom opadów. Dane te mogą być przesyłane bezprzewodowo do centralnego serwera, gdzie jest przetwarzane i analizowane. Systemy oferują intuicyjny interfejs użytkownika, dostępny zarówno jako aplikacja webowa, jak i mobilna, umożliwiający dostęp do danych w czasie rzeczywistym, wizualizacji oraz generowania raportów.

System analizy danych meteorologicznych powinien być środowiska, w którym użytkownik będzie mógł analizować dane meteorologiczne w czasie rzeczywistym w celu poprawy funkcjonowania różnych dziedzin życia. System stacji meteorologicznej powinien dostarczać precyzyjne i aktualne dane pogodowe oraz narzędzia do ich efektywnego wykorzystania. Głównym celem takiego systemu jest automatyczne i ciągłe zbieranie danych meteorologicznych z różnych czujników rozmieszczonych na stacji meteorologicznej. Dzięki wykorzystaniu technologii IoT, dane są przesyłane bezprzewodowo do centralnego serwera, co zapewniało ich aktualność i precyzję. System analizuje zebrane dane w celu wykrywania wzorców i anomalii. Wykorzystanie zaawansowanych algorytmów analizy danych i uczenia maszynowego pozwala na przewidywanie przyszłych warunków pogodowych oraz identyfikację niebezpiecznych zjawisk pogodowych, takich jak gwałtowne burze czy nagłe spadki temperatury. System powinien oferować zaawansowane narzędzia do wizualizacji danych, w tym interaktywne wykresy, mapy i pulpit nawigacyjny (ang. *dashboard*). Umożliwi to użytkownikom łatwe monitorowanie aktualnych warunków pogodowych oraz śledzenie zmian w czasie. Wizualizacje powinny być dostępne zarówno na stronie internetowej, jak i w aplikacji mobilnej, co zapewni wygodny dostęp do informacji. Kluczowym elementem systemu jest moduł powiadomień, który informuje użytkowników o nagłych zmianach pogodowych lub wykrytych anomaliach. Integracja powinna być zaprojektowana z myślą o innych platformach i systemach, takich jak zewnętrzne

systemy prognoz pogody, aplikacje mobilne czy systemy zarządzania kryzysowego. Pozwoli to na stworzenie bardziej kompleksowych rozwiązań i zwiększenie użyteczności zgromadzonych danych. System powinien zbierać i analizować dane na bieżąco, co umożliwi optymalizację różnych procesów. Architektura Big Data dostarcza cennych informacji, które pomagają w podejmowaniu bardziej świadomych decyzji.

Projekt systemu obsługującego stacji meteorologicznej z analizą danych IoT w czasie rzeczywistym powinien mieć na celu nie tylko dostarczenie precyzyjnych i aktualnych danych pogodowych, ale również zapewnienie narzędzi do ich efektywnego wykorzystania, co przyczyniło się do poprawy zarządzania różnymi zasobami i zwiększenia bezpieczeństwa w wielu dziedzinach życia.

### 5.1.2. Wykrycie i wybór źródeł danych

Dane z bazy danych są dostarczane do systemu wraz z informacjami dotyczącymi różnorodnych parametrów. Ponadto, zawierają odwołania do źródła danych. Identyfikacja źródeł danych jest krokiem niezbędnym przed wykonaniem analizy, aby można było zadecydować, które dane warto przeanalizować. W niniejszym projekcie wykorzystane zostały wyłącznie dane fikcyjne (sztucznie wygenerowane) jednak architektura została zaprojektowana tak, aby w łatwy sposób można było zmienić źródło danych na dane pochodzące z urządzeń IoT, takich jak sensory dostarczającej dane w czasie rzeczywistym. W przyszłości mogłyby zostać przeprowadzona rozbudowa o dane historyczne.

Dane pogodowe są zazwyczaj zbierane przez różne czujniki IoT rozmieszczone na stacji meteorologicznej. Czujniki te mierzą podstawowe parametry środowiskowe, takie jak temperatura, wilgotność, ciśnienie atmosferyczne, prędkość i kierunek wiatru oraz poziom opadów. Zebrane dane przesyłane są w czasie rzeczywistym do centralnego serwera, gdzie następnie podlegają przetworzeniu i analizie. Dane te są kluczowe do monitorowania aktualnych warunków pogodowych i przewidywania zmian. Ich analiza jest niezbędna do reagowania w razie wystąpienia sytuacji awaryjnej, jaka mogła się wydarzyć. Dane pochodzące z urządzeń IoT w czasie rzeczywistym są dla takich projektów kluczowe, ponieważ dostarczają one najbardziej aktualnych informacji. Potencjalny przyszły rozwój o historyczne dane mógłby umożliwić bardziej zaawansowane analizy trendów i wzorców. Różne źródła danych IoT zostały przedstawione na rysunku 5.1.



5.1. IoT źródła danych  
Źródło: <https://amendllc.com>

### 5.1.3. Określenie początkowych wymagań

Stacje meteorologiczne gromadzą ogromne ilości danych pochodzących z różnorodnych źródeł, takich jak sensory, satelity, radary i modele numeryczne. Analiza tych danych w czasie rzeczywistym jest kluczowa dla precyzyjnych prognoz pogody, monitorowania warunków atmosferycznych oraz ostrzegania przed ekstremalnymi zjawiskami pogodowymi. W celu sprostania tym wymaganiom, projekt zakłada wdrożenie architektury Big Data, która umożliwia efektywne przetwarzanie i analizę danych w czasie rzeczywistym.

W założeniu architektura przetwarzania danych w czasie rzeczywistym dla stacji meteorologicznej ma składać się z kilku kluczowych komponentów. Dane mogą być zbierane z różnych źródeł, takich jak czujniki rozmieszczone na stacjach meteorologicznych, satelity, radary i inne urządzenia pomiarowe. Następnie są one przesyłane do systemu przetwarzania danych za pomocą gorącej ścieżki, która umożliwia natychmiastowe przetwarzanie i analizę danych. Jest to kluczowe w kontekście prognozowania pogody i monitorowania warunków atmosferycznych w czasie rzeczywistym. Dane przepływają do systemu przetwarzania strumieniowego, gdzie podlegają natychmiastowej analizie. System jest wyposażony w zaawansowane algorytmy analizy strumieniowej, które pozwalają na wykrywanie anomalii, identyfikację trendów oraz generowanie prognoz w czasie rzeczywistym. Następnie przetworzone dane są przechowywane w magazynach danych, gdzie podlegają modelowaniu oraz organizowaniu w sposób umożliwiający łatwe i szybkie wyszukiwanie oraz analizę historycznych danych meteorologicznych. Zarządzanie metadanymi stało się kluczowym elementem systemu umożliwiającym dokładne śledzenie pochodzenia, jakości i aktualności danych. Standardem powinno być wdrożenie mechanizmów, takich jak kontrola dostępu, szyfrowanie danych oraz regularne audyty systemu, które zapewnią bezpieczeństwo i integralność danych. Ostatecznym etapem budowy architektury jest udostępnienie przetworzonych danych narzędziom analitycznym i wizualizacyjnym. Umożliwia to tworzenie szczegółowych raportów, interaktywnych wizualizacji oraz dostarczanie informacji kluczowych dla podejmowania decyzji w czasie rzeczywistym. Wizualizacje te są dostępne dla meteorologów, decydentów oraz innych zainteresowanych stron, wspomagając ich w podejmowaniu trafnych i szybkich decyzji.

Docelowo implementacja systemu przetwarzania danych w czasie rzeczywistym dla stacji meteorologicznej ma przynieść liczne korzyści, w tym poprawę dokładności prognoz pogody, szybsze wykrywanie i reagowanie na ekstremalne zjawiska pogodowe oraz lepsze zarządzanie zasobami. Dzięki wykorzystaniu zaawansowanych technologii Big Data stacja meteorologiczna zyska możliwość efektywnego przetwarzania ogromnych ilości danych, a tym samym dostarczania wartościowych informacji w czasie rzeczywistym.

#### 5.1.4. Finalne wymagania systemowe

Przypadek użycia (ang. *use case*) opisuje specyficzny zestaw działań podejmowanych w celu osiągnięcia określonego zadania. W kontekście stacji meteorologicznej, przypadek użycia pomaga zrozumieć, jak różne funkcje i komponenty systemu będą wykorzystywane przez użytkowników do zbierania, przetwarzania i analizowania danych meteorologicznych. Poniższy opis przedstawia, jak zidentyfikowano końcowe wymagania dla stacji meteorologicznej, która miałaby korzystać z przetwarzania danych w czasie rzeczywistym w ekosystemie Big Data.

Wymagania stawiane stacji meteorologicznej obejmują:

- zbieranie danych meteorologicznych na temat temperatury, wilgotności, ciśnienia atmosferycznego, prędkości i kierunku wiatru oraz innych parametrów pogodowych z wykorzystaniem sensorów i urządzeń pomiarowych rozmieszczonych w różnych lokalizacjach,
- przesyłanie danych w czasie rzeczywistym przy użyciu gorącej ścieżki, co umożliwioby ich bieżącą analizę,
- przetwarzanie i analiza danych przy wykorzystaniu zaawansowanych algorytmów do wykrywania anomalii pogodowych, przewidywania zmian pogody i generowania ostrzeżeń,
- wizualizacja danych w postaci wykresów, diagramów, map termicznych i interaktywnych pulpitów nawigacyjnych, co ułatwioby ich interpretację i wykorzystanie przez meteorologów i decydentów,
- automatyczne generowanie raportów pogodowych, które stanowiłyby podsumowanie najważniejszych informacji i prognoz,
- bezpieczne przechowywanie danych w bazach z możliwością łatwego wyszukiwania i przetwarzania historycznych danych meteorologicznych,
- posiadanie mechanizmów kontroli dostępu, które zapewnią przeglądanie i modyfikację danych tylko przez upoważnione osoby.

Metody prezentacji i wykorzystania danych zastosowane w stacji meteorologicznej:

- tabele (w celu ułatwienia przeglądania oraz analizy danych),
- diagramy i wykresy (w celu ułatwienia interpretacji danych),
- mapy termiczne (w celu ułatwienia identyfikacji wzorców pogodowych),
- techniki analizy statystycznej i uczenia maszynowego do przetwarzania i interpretacji danych,
- automatyczne generowanie raportów, które przedstawałyby kluczowe informacje w przystępny sposób,
- bezpieczne przechowywanie danych w bazach danych i systemach przechowywania z możliwością szybkiego wyszukiwania.

Finalna architektura systemu stacji meteorologicznej ma zapewniać:

- efektywne gromadzenie i archiwizowanie danych meteorologicznych,
- integrację danych z różnych źródeł i ich spójne prezentowanie,
- strukturalizowanie danych w sposób ułatwiający ich analizę,
- możliwość łatwego dostępu do danych przez uprawnionych użytkowników,
- prowadzenie dokładnej ewidencji danych i ich źródeł,
- bieżącą analizę danych w celu generowania prognoz i wykrywania anomalii,
- tworzenie czytelnych i interaktywnych wizualizacji danych,
- automatyczne tworzenie raportów na podstawie zebranych danych.

Implementacja omówionej architektury w stacji meteorologicznej ma zapewnić bieżące monitorowanie warunków pogodowych, dokładne prognozy oraz szybkie reagowanie na zmieniające się warunki atmosferyczne, a tym samym wesprzeć działania meteorologów i decydentów.

## **5.2. Przedstawienie możliwych rozwiązań architektonicznych**

### **5.2.1 Architektura Big Data z wykorzystaniem Azure**

Pierwsze prezentowane rozwiązanie architektoniczne dla stacji meteorologicznej wykorzystuje platformę Microsoft Azure do przetwarzania danych w czasie rzeczywistym. Głównym komponentem architektury jest Azure IoT Hub – usługa w chmurze umożliwiająca komunikację między urządzeniami IoT a platformą Azure. W przypadku omawianego projektu, IoT Hub ma za zadanie odbierać dane meteorologiczne z różnych sensorów i urządzeń rozmieszczonych w terenie. Odpowiedzialny jest za: zbieranie i przesyłanie danych w czasie rzeczywistym, zarządzanie komunikacją urządzeń IoT, monitorowanie stanu urządzeń. Drugi komponent stanowiło Azure Stream Analytics – usługa do strumieniowego przetwarzania danych w czasie rzeczywistym. Umożliwiła ona analizę napływających danych i generowanie wyników w czasie rzeczywistym. Przetwarzanie i analiza strumieni danych meteorologicznych, wykrywanie wzorców i anomalii, generowanie alertów i powiadomień zostały wykonane właśnie w tym komponencie architektury. Kolejny komponent, Azure Data Lake Storage, to skalowalne i bezpieczne repozytorium danych, które umożliwia przechowywanie dużych ilości danych w różnych formatach. Zapewnia funkcjonalności takie jak przechowywanie surowych i przetworzonych danych meteorologicznych, łatwy dostęp do danych dla analiz i raportowania. Azure SQL Database, relacyjna baza danych oferowana przez Azure jako usługa (DBaaS). Umożliwia przechowywanie i zarządzanie danymi w relacyjnym formacie, przechowywanie przetworzonych danych, a także szybkie wyszukiwanie i analizę danych oraz generowanie raportów. Azure Machine Learning – usługa do budowy, trenowania i wdrażania modeli uczenia maszynowego – umożliwia integrację modeli z aplikacjami i usługami w chmurze. Ten element architektury odpowiedzialny jest za budowanie i trenowanie modeli prognozowania pogody, analizę danych historycznych i bieżących, a także za wykrywanie wzorców i anomalii pogodowych. Power BI, jako narzędzie do analizy biznesowej i wizualizacji danych, umożliwia tworzenie interaktywnych raportów i pulpitów nawigacyjnych i zapewni w architekturze wizualizację danych meteorologicznych, tworzenie interaktywnych raportów i pulpitów nawigacyjnych, jak również udostępnianie wyników analiz zainteresowanym stronom.

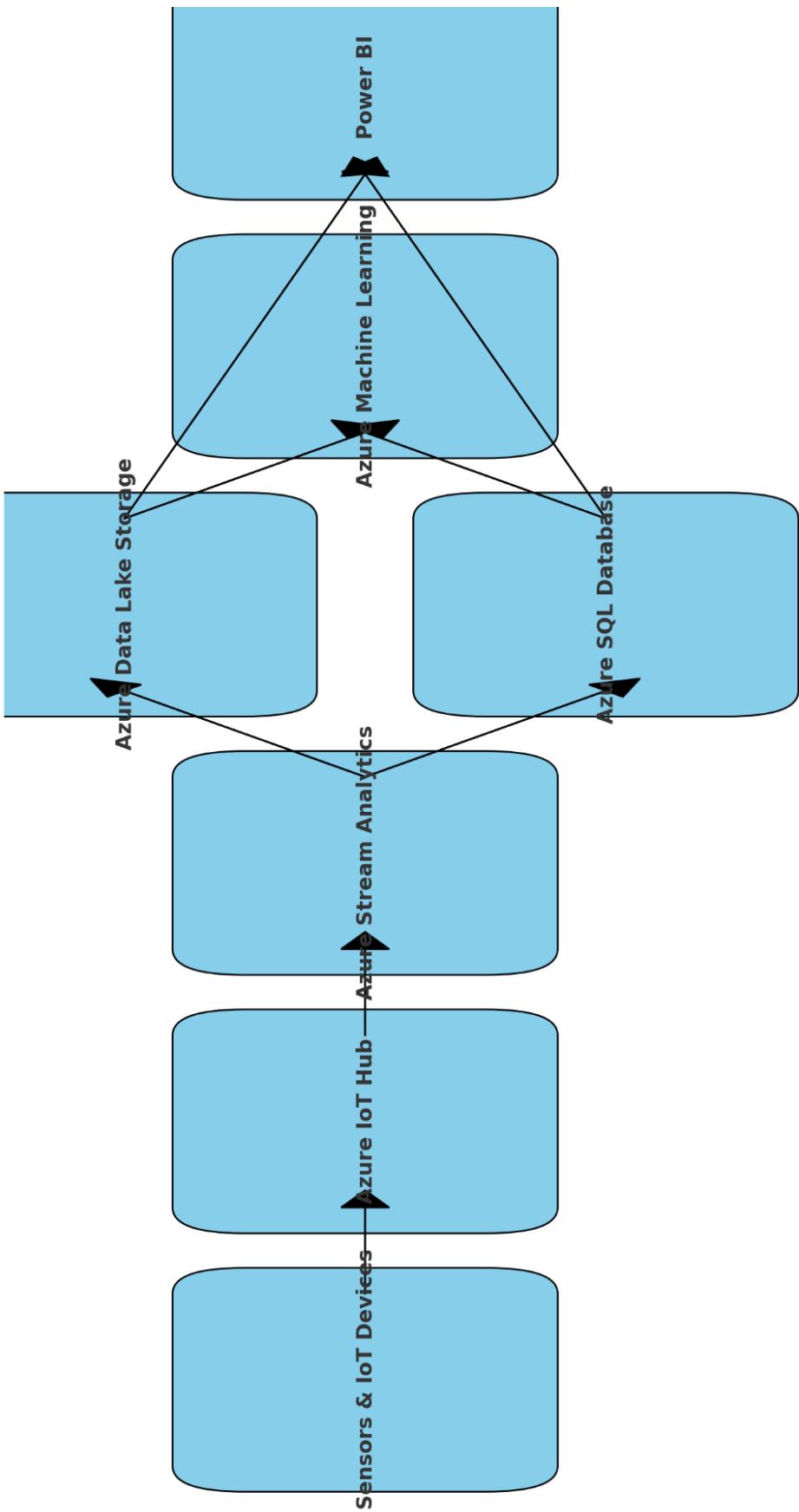
W pierwszym rozwiązaniu architektonicznym dla stacji meteorologicznej do przetwarzania danych w czasie rzeczywistym wykorzystano platformę Microsoft Azure. Zbieranie danych rozpoczyna się od przesyłania danych meteorologicznych z różnych sensorów i urządzeń IoT do Azure IoT Hub w czasie rzeczywistym. Azure IoT Hub, jako centralny punkt komunikacyjny, odpowiada za zbieranie i przesyłanie danych, zarządzanie komunikacją urządzeń IoT oraz monitorowanie ich stanu. Azure IoT Hub w celu przetworzenia i analizy przesyła zebrane dane do Azure Stream Analytics – usługi do przetwarzania strumieniowego danych

w czasie rzeczywistym. Azure Stream Analytics analizuje napływające dane meteorologiczne, wykrywa wzorce i anomalie, a następnie przekazuje wyniki do Azure Data Lake Storage oraz Azure SQL Database. Pierwsze z nich jest skalowalnym i bezpiecznym repozytorium danych, umożliwiającym przechowywanie dużych ilości danych w różnych formatach. W tej architekturze, Azure Data Lake Storage przechowuje surowe i przetworzone dane meteorologiczne, zapewniając łatwy dostęp do nich dla celów analizy i raportowania. Z kolei Azure SQL Database umożliwia przechowywanie i zarządzanie danymi w relacyjnym formacie, co pozwala na szybkie wyszukiwanie, analizę danych i generowanie raportów. Analiza danych odbywa się z wykorzystaniem modeli uczenia maszynowego stworzonych w Azure Machine Learning. Powstałe w ten sposób modele analizują dane historyczne i bieżące, prognozując przyszłe warunki pogodowe oraz wykrywając wzorce i anomalie. Wizualizacja danych realizowana jest za pomocą Power BI. Pobiera ono dane z Azure SQL Database i Azure Data Lake Storage, a następnie tworzy interaktywne raporty i dashboardy, które są udostępniane zainteresowanym stronom. Dzięki dane prezentowane są w sposób przystępny i zrozumiały dla użytkowników.

Architektura oparta na usługach Azure posiada wiele zalet, takich jak skalowalność, bezpieczeństwo, elastyczność oraz optymalizacja kosztów. Możliwość skalowania usług w zależności od potrzeb i ilości przetwarzanych danych, zaawansowane mechanizmy zabezpieczeń, integracja z różnymi źródłami danych oraz elastyczny model płatności w zależności od wykorzystanych zasobów sprawiają, że jest to efektywne rozwiązanie dla zarządzania danymi meteorologicznymi. Implementacja tej architektury zapewnia stacji meteorologicznej efektywne zarządzanie danymi, poprawę dokładności prognoz pogodowych oraz szybką reakcję na zmieniające się warunki atmosferyczne.

Diagram architektury przetwarzania danych meteorologicznych opartej na platformie Azure, przedstawiający przepływ danych między poszczególnymi komponentami został przedstawiony na rysunku 5.2. Na początku Azure IoT Hub odbiera dane z czujników i urządzeń IoT, przesyłając je do dalszego przetwarzania. Napływające dane przetwarzane są w czasie rzeczywistym przez Azure Stream Analytics, które wykrywa wzorce i anomalie. Następnie dane rozdzielane są między Azure Data Lake Storage (które przechowuje surowe i przetworzone dane, umożliwiając łatwy dostęp do nich) oraz Azure SQL Database (przechowujące przetworzone dane w relacyjnym formacie, a tym samym umożliwiające szybkie wyszukiwanie i analizę). W kolejnej fazie Azure Machine Learning analizuje dane historyczne i bieżące za pomocą modeli uczenia maszynowego, prognozując przyszłe warunki pogodowe. Na koniec Power BI tworzy interaktywne raporty i dashboardy, wizualizując dane dla zainteresowanych stron.

## Azure-Based Meteorological Data Processing Architecture



Rys. 5.2 Diagram architektury przetwarzania danych meteorologicznych opartej na platformie Azure

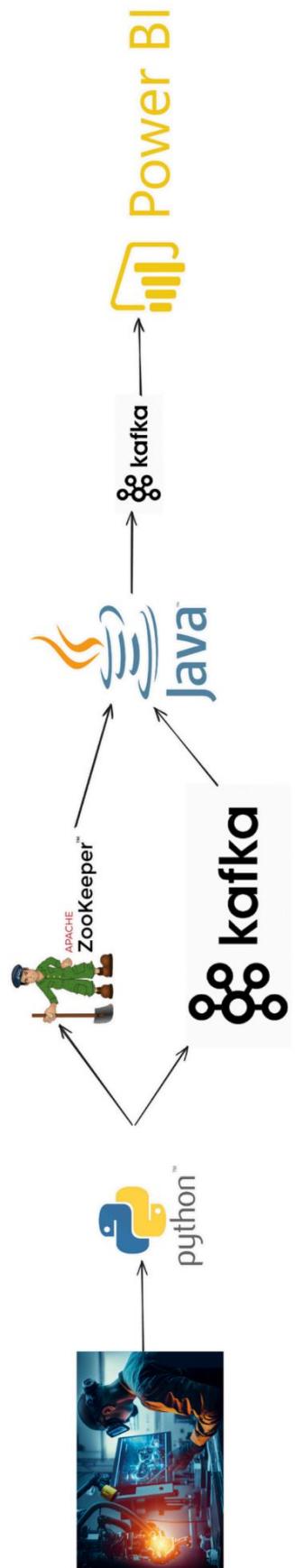
### 5.2.2 Architektura Big Data z wykorzystaniem Apache Kafka

Architektura ta wykorzystuje różne technologie do zbierania, przetwarzania, zarządzania i wizualizacji danych meteorologicznych. Dane meteorologiczne mogą być zbierane z różnych sensorów IoT za pomocą skryptów napisanych w Pythonie. Skrypty w Pythonie komunikują się z sensorami, odbierając dane dotyczące temperatury, wilgotności, ciśnienia atmosferycznego, prędkości i kierunku wiatru oraz innych parametrów pogodowych. Aplikacje napisane w języku Java przetwarzają surowe dane meteorologiczne, filtryują je, agregują i przekształcają w struktury gotowe do dalszej analizy. Apache ZooKeeper zarządza metadanymi i koordynuje pracę klastrów. ZooKeeper zapewnia synchronizację między różnymi komponentami systemu, zarządza konfiguracją i monitoruje stan usług oraz klastrów. Apache Kafka przechowuje przetworzone dane meteorologiczne i zapewnia niezawodne oraz skalowalne przesyłanie strumieniowe danych do dalszego przetwarzania i analizy. Command Prompt (w oknie terminala) jest używany do uruchamiania i monitorowania różnych usług systemu. Za pomocą poleceń w terminalu administratorzy mogą uruchamiać aplikacje, monitorować ich działanie oraz zarządzać nimi. Power BI działa analogicznie jak na platformie Azure, a więc pobiera dane z Apache Kafka, po czym tworzy interaktywne raporty i dashboardy, które przedstawiają aktualne i historyczne dane meteorologiczne w przystępny sposób.

Przepływ danych w omawianej architekturze przebiega następująco:

1. Sensory IoT zbierają dane meteorologiczne i przesyłają je do skryptów Python.
2. Skrypty Python przesyłają surowe dane do aplikacji Java. Aplikacje Java przetwarzają dane, a następnie przesyłają je do Apache Kafka.
3. Apache ZooKeeper koordynuje i zarządza klastrami oraz metadanymi systemu.
4. Apache Kafka przechowuje przetworzone dane i udostępnia je do dalszej analizy.
5. Power BI prezentuje użytkownikom dane w czytelnej formie.

Zaletami tej architektury są skalowalność, elastyczność, niezawodność i interaktywność. Możliwość skalowania każdego komponentu niezależnie pozwala sprostać rosnącym wymaganiom danych. Integracja różnych technologii umożliwia elastyczne zarządzanie danymi i dostosowywanie systemu do potrzeb, a wizualizacja danych wspomaga analizę i podejmowanie decyzji. Implementacja tej architektury zapewnia stacji meteorologicznej efektywne zbieranie, przetwarzanie, zarządzanie i wizualizację danych meteorologicznych, wspierając dokładność prognoz i umożliwiając szybkie reagowanie na zmieniające się warunki pogodowe. Architektura została przedstawiona na rysunku 5.3.



Rys. 5.3. Diagram architektury przetwarzania danych meteorologicznych opartej na Apache Kafka

## **5.3. Wybór Architektura Big Data**

### **5.3.1 Architektury ekosystemu stacji meteorologicznej**

Dla celów biznesowych i ich przypadków użycia wybrano architekturę opartą na technologii Apache Kafka, która integruje różne technologie do zbierania, przetwarzania, zarządzania i wizualizacji danych meteorologicznych. Architektura została rozbudowana o dodatkową warstwę zarządzania, co doprowadziło do podziału na dwie główne warstwy: warstwę zarządzania oraz warstwę przetwarzania, magazynowania i wizualizacji danych. Taki podział jest kluczowy dla lepszego zorganizowania zadań w całym ekosystemie.

Warstwa zarządzania składa się z następujących komponentów: Apache Airflow, Apache Atlas, Apache Ambari oraz konsola CMD. Natomiast w drugiej warstwie znajdują się takie oprogramowania, komponenty i technologie jak: Apache Kafka, Apache ZooKeeper oraz skrypty napisane w językach Java i Python, wspierane przez Power BI.

W poprzednim rozdziale opisano komponenty należące do warstwy przetwarzania, magazynowania i wizualizacji danych, szczegółowo omawiając ich funkcjonalność i rolę w architekturze. Warstwa zarządzania, dodana do całego ekosystemu, została szczegółowo opisana w kolejnych podrozdziałach.

W ramach projektu zostały stworzone poniższe komponenty architektury Big Data:

- orchestracja, zarządzanie metadanymi oraz administracja stacji meteorologicznej,
- przetwarzanie danych w ekosystemie,
- magazyn danych i organizacja danych,
- udostępnianie oraz analiza danych,
- diagram architektury oraz tabela użytych technologii.

Poniżej zostało opisane wykonywanie poszczególnych kroków.

### 5.3.2 Orquestracja, zarządzanie metadanymi oraz administracja stacji meteorologicznej

Do architektury systemu meteorologicznego została wybrana technologia Apache Airflow. Apache Airflow umożliwia harmonogramowanie, monitorowanie i zarządzanie przepływami pracy związanymi z przetwarzaniem danych meteorologicznych. Dzięki Airflow możliwe jest zautomatyzowanie procesów zbierania, przetwarzania i analizy danych, co zwiększa efektywność i niezawodność systemu. Umożliwia on również łatwe integrowanie różnych narzędzi i technologii używanych w architekturze, takich jak Apache Kafka, Python czy Power BI. Wybór Apache Airflow pozwala na skalowanie systemu i lepsze zarządzanie skomplikowanymi przepływami danych. Apache Atlas to zaawansowane narzędzie do zarządzania metadanymi, które oferuje funkcje katalogowania, zarządzania linią przetwarzania danych oraz kontroli dostępu. Umożliwia również śledzenie przepływu danych w ekosystemie Hadoop, integrując się z takimi narzędziami jak Apache Hive, Apache HBase i Apache Kafka. Apache Atlas został wykorzystany do skutecznego zarządzania danymi, zapewniając zgodność i pełną widoczność procesów przetwarzania danych. Apache Ambari to kompleksowa platforma do zarządzania, monitorowania i zabezpieczania klastrów Hadoop, oferująca intuicyjny interfejs użytkownika. Umożliwia ona automatyzację konfiguracji, zarządzania i monitorowania klastrów, zapewniając narzędzia do monitorowania wydajności i stanu komponentów. Dzięki Apache Ambari, administratorzy mogą skutecznie zarządzać dużymi infrastrukturami Big Data, minimalizując czas i wysiłek potrzebny do administracji systemem.

Wybór Apache Airflow, Apache Atlas i Apache Ambari w budowie architektury systemu meteorologicznego zapewnia efektywne i zautomatyzowane zarządzanie danymi, możliwość ich przetwarzania oraz monitorowania, co przekłada się na zwiększoną niezawodność i skalowalność systemu.

### 5.2.3 Przetwarzanie danych w ekosystemie

Przetwarzanie danych w ekosystemie systemu meteorologicznego oparte zostało na zintegrowanej architekturze Big Data, która zapewnia efektywne zbieranie, przetwarzanie, zarządzanie i analizę danych meteorologicznych. Kluczowe elementy tego ekosystemu to Apache Airflow, Apache Kafka, Apache Atlas, które współpracują, aby zapewnić płynny i niezawodny przepływ danych w czasie rzeczywistym. Proces przetwarzania danych rozpoczyna się od zbierania danych meteorologicznych za pomocą skryptów Python, które komunikują się z różnymi sensorami IoT. Dane dotyczące temperatury, wilgotności, ciśnienia atmosferycznego, prędkości i kierunku wiatru oraz innych parametrów pogodowych są zbierane w czasie rzeczywistym i przesyłane do centralnego systemu za pomocą Apache Kafka, który zapewnia skalowalne i niezawodne przesyłanie strumieniowe danych. Kafka przechowuje dane w formie strumieni, umożliwiając ich przetwarzanie w czasie rzeczywistym lub późniejsze odczytywanie.

### 5.2.4 Magazyn danych i organizacja danych

W architekturze systemu meteorologicznego kluczową rolę w magazynowaniu i organizacji danych pełnią Apache Kafka oraz Apache ZooKeeper. Te technologie współpracują, aby zapewnić niezawodne, skalowalne i dobrze zorganizowane przechowywanie oraz przesyłanie danych meteorologicznych.

Apache Kafka służy jako centralny magazyn danych w systemie meteorologicznym. Jest to platforma do przetwarzania strumieniowego, która cechuje się wysoką skalowalnością i niezawodnością. Dane meteorologiczne, zbierane z różnych sensorów IoT, są przesyłane do Kafki, gdzie są przechowywane w formie strumieni. Każdy strumień danych reprezentuje konkretne wydarzenie, takie jak odczyt temperatury, wilgotności, ciśnienia atmosferycznego, prędkości wiatru, itp. Kafka obsługuje ogromne ilości danych, dzięki czemu jest idealnym rozwiązaniem dla systemów wymagających przetwarzania dużych strumieni danych w czasie rzeczywistym. Zapewnia niezawodne przechowywanie danych dzięki mechanizmowi replikacji i tolerancji błędów, a także umożliwia łatwą integrację z innymi systemami przetwarzania danych i analitycznymi, co czyni ją centralnym punktem w architekturze przetwarzania danych.

Apache ZooKeeper pełni kluczową rolę w zarządzaniu metadanymi i synchronizacją komponentów systemu. ZooKeeper koordynuje operacje między rozproszonymi procesami, zapewniając spójność danych i stanów systemu. Przechowuje informacje o stanie klastrów, dostępnych zasobach i innych istotnych metadanych, które są niezbędne do prawidłowego działania systemu. Umożliwia zarządzanie stanem klastrów, monitorowanie dostępności węzłów i zarządzanie procesami przetwarzania danych, a także zapewnia kontrolę dostępu do danych i zasobów, co zwiększa bezpieczeństwo systemu.

W systemie meteorologicznym Apache Kafka i Apache ZooKeeper współpracują, aby zapewnić płynny przepływ danych oraz zarządzanie metadanymi. Kafka przechowuje i przesyła strumienie danych, podczas gdy ZooKeeper zarządza metadanymi oraz koordynuje operacje między różnymi komponentami systemu. Ta integracja zapewnia, że dane są zawsze aktualne, spójne i łatwo dostępne do dalszej analizy i przetwarzania. Wybór Apache Kafka jako magazynu danych oraz Apache ZooKeeper do zarządzania metadanymi i synchronizacją komponentów systemu meteorologicznego zapewnia niezawodne, skalowalne i dobrze zorganizowane przetwarzanie danych. Kafka umożliwia efektywne przechowywanie i przesyłanie strumieni danych, podczas gdy ZooKeeper zapewnia koordynację i zarządzanie stanem systemu, co jest kluczowe dla utrzymania spójności i niezawodności całego ekosystemu.

### 5.2.5 Udostępnianie oraz analiza danych

W systemie meteorologicznym udostępnianie oraz analiza danych są kluczowymi etapami, które zapewniają użytkownikom dostęp do przetworzonych informacji i umożliwiają podejmowanie na podstawie tych danych świadomych decyzji. Wykorzystanie odpowiednich narzędzi i technologii umożliwia sprawne i efektywne zarządzanie tymi procesami.

Do udostępniania i analizy danych w systemie meteorologicznym zostały wybrane technologie Apache Kafka, Power BI oraz Apache ZooKeeper. Apache Kafka, jako platforma do przetwarzania strumieniowego, odgrywa kluczową rolę w udostępnianiu danych. Przetworzone dane meteorologiczne są przechowywane w formie strumieni, co umożliwia ich szybkie i efektywne przesyłanie do dalszych systemów analitycznych. Kafka działa jako pośrednik, który dystrybuje dane do różnych narzędzi analitycznych i wizualizacyjnych, zapewniając, że wszystkie zainteresowane strony mają dostęp do aktualnych i dokładnych informacji. Power BI, który technicznie jest aspektem SharePoint online, umożliwia ładowanie skoroszytów programu Excel do chmury i udostępnianie ich wybranej grupie współpracowników. Współpracownicy mogą wchodzić w interakcje z rapportami, aby stosować filtry i podświetlać dane. Power BI pozwala również pracownikom informacyjnym udostępniać zapytania i, być może złożone, procedury pozyskiwania danych, które złożonymi procedurami pozyskiwania danych, które utworzyli za pomocą Power Query. W ten sposób organizacja może uniknąć powielania wysiłków, które mogą powstać, gdy pracownicy pracują w silosach danych. Ponadto można zweryfikować określone źródła danych jako kluczowe dla zatwierzonego zestawu danych. Power BI może również zapewnić, że udostępnione skoroszyty programu Excel są automatycznie i regularnie aktualizowane, aby użytkownicy zawsze mieli dostęp do najnowszych danych. Zaawansowanym funkcje wizualizacyjne zapewniają użytkownika łatwą możliwość dostosowywania widoki danych do swoich potrzeb i podejmować lepsze decyzje na podstawie dostępnych informacji.

Apache ZooKeeper pełni funkcję zarządzania i koordynacji w systemie, zapewniając, że wszystkie komponenty działają w harmonii. ZooKeeper zarządza metadanymi oraz synchronizuje operacje między różnymi narzędziami i usługami, co jest kluczowe dla utrzymania spójności danych i efektywnego udostępniania informacji. Dzięki ZooKeeper, system jest w stanie automatycznie reagować na zmiany w danych, skalować się w odpowiedzi na rosnące obciążenia i utrzymywać wysoką dostępność oraz niezawodność.

W systemie meteorologicznym udostępnianie i analiza danych są zatem realizowane poprzez harmonijną współpracę Apache Kafka, Power BI i Apache ZooKeeper. Kafka zapewnia niezawodne i skalowalne przesyłanie danych, Power BI umożliwia zaawansowaną wizualizację i analizę, a ZooKeeper zarządza metadanymi i koordynuje operacje systemowe. Ta kombinacja technologii pozwala na efektywne zarządzanie danymi meteorologicznymi, dostarczając użytkownikom wartościowych informacji w czasie rzeczywistym oraz umożliwiając dokładne analizy i prognozy pogodowe.

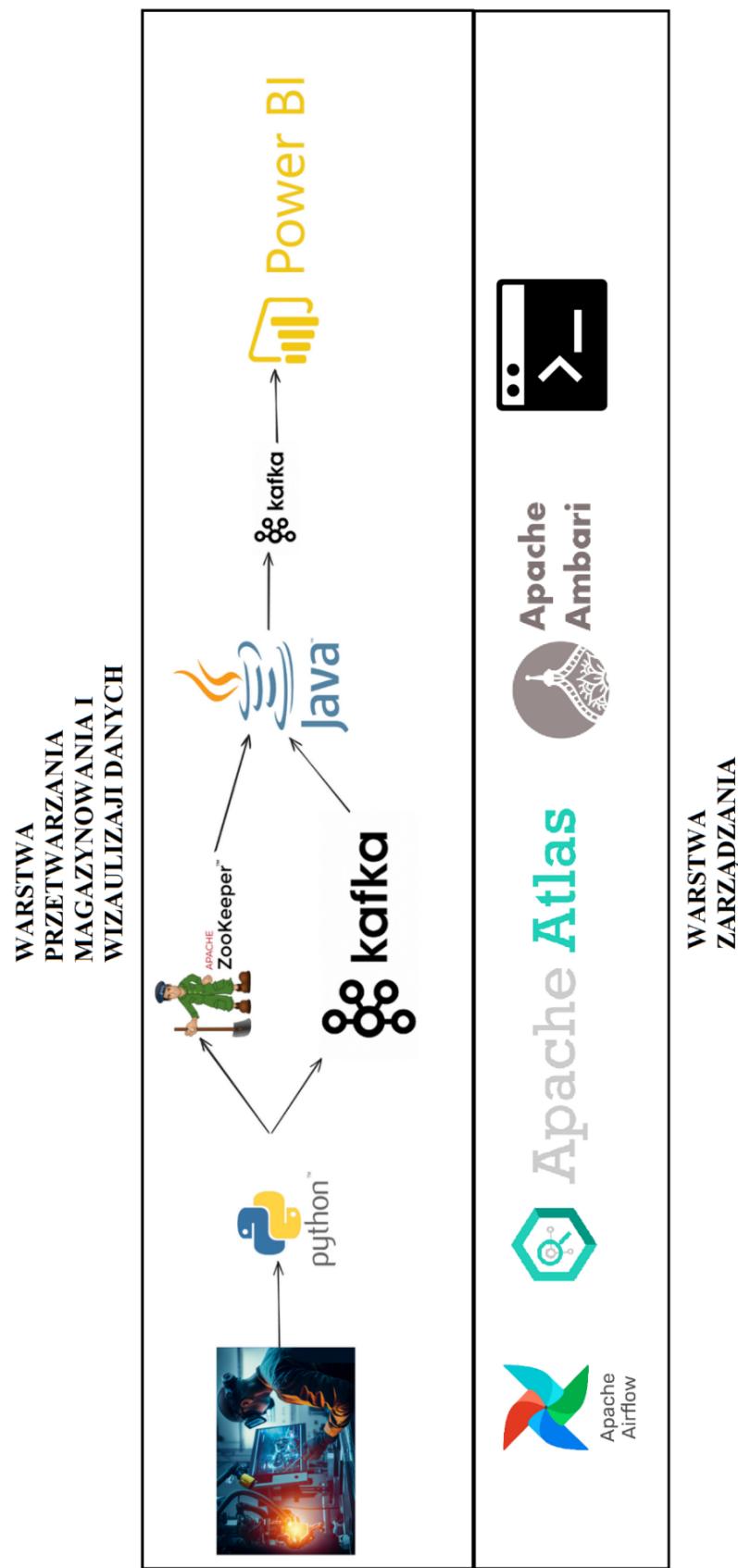
### 5.2.6 Diagram architektury oraz tabela użytych technologii

Tabela 5.4 przedstawiono technologie używane w architekturze ekosystemu stacji meteorologicznej. Każda kategoria opisuje konkretne zadanie w systemie, wraz z odpowiednią technologią i jej krótkim opisem.

Kategoria	Technologia	Opis
Zbieranie danych	Skrypty Python	Skrypty do komunikacji z sensorami IoT, zbierające dane dotyczące temperatury, wilgotności, ciśnienia atmosferycznego, prędkości i kierunku wiatru.
Przetwarzanie danych	Aplikacje Java	Przetwarzanie surowych danych meteorologicznych, filtrowanie, agregowanie i przekształcanie w struktury gotowe do dalszej analizy.
Magazynowanie i przesyłanie danych	Apache Kafka	Centralny magazyn danych przechowujący przetworzone dane meteorologiczne i zapewniający niezawodne, skalowalne przesyłanie strumieniowe danych.
Zarządzanie metadanymi i synchronizacja	Apache ZooKeeper	Zarządzanie metadanymi, synchronizacja komponentów systemu, monitorowanie stanu usług oraz klastrów, zapewnianie spójności danych i stanów systemu.
Orchestracja i zarządzanie przepływami	Apache Airflow	Harmonogramowanie, monitorowanie i zarządzanie przepływami pracy związanymi z przetwarzaniem danych meteorologicznych, automatyzacja procesów.
Zarządzanie metadanymi	Apache Atlas	Katalogowanie, zarządzanie linią przetwarzania danych, kontrola dostępu, śledzenie przepływu danych w ekosystemie Hadoop, integracja z innymi narzędziami.
Zarządzanie klastrami	Apache Ambari	Kompleksowa platforma do zarządzania, monitorowania i zabezpieczania klastrów Hadoop, automatyzacja konfiguracji, zarządzania i monitorowania klastrów.
Wizualizacja i analiza danych	Power BI	Narzędzie do analizy biznesowej i wizualizacji danych, tworzenie interaktywnych raportów i dashboardów na podstawie danych meteorologicznych.
Interfejs zarządzania systemem	Command Prompt (terminal CMD)	Narzędzie do uruchamiania i monitorowania różnych usług systemu, zarządzanie aplikacjami, monitorowanie ich działania.

Tabela 5.4 Technologie używane w architekturze ekosystemu stacji meteorologicznej

Na rys 5.5 został przedstawiony diagram architektury dla stacji meteorologicznej.



Rys. 5.5. Diagram architektury przetwarzania danych meteorologicznych opartej na platformie Azure

## 5.4. Opis implementacji

### 5.4.1 Generowanie danych za pomocą Pythona

Do generowania danych zostały wykorzystane fikcyjne dane (ang. *dummy data*) dla możliwości stworzenia architektury ze względu na łatwiejszą implementację względem wykorzystania prawdziwych sensorów IoT. Zostały one wygenerowane za pomocą skryptu Python. Dane te są w formacie surowych i napływają w czasie rzeczywistym, jeden odczyt co 15 sekund. Skrypt został przedstawiony na rysunku 5.6.

```
⚡ dummy_data.py > ...
1   from time import sleep
2   from kafka import KafkaProducer
3   import random
4   from datetime import datetime
5
6   producer = None
7
8   try:
9       producer = KafkaProducer(bootstrap_servers='localhost:9092')
10
11      while True:
12          temperature = round(random.uniform(-20, 40), 2)
13          humidity = round(random.uniform(30, 90), 2)
14          timestamp = datetime.now().isoformat()
15          data = f'{timestamp},{temperature},{humidity}'
16          producer.send('real-time-data', data.encode())
17          sleep(15)
18          print(data)
19
20      except Exception as e:
21          print(f"Error: {e}")
22      finally:
23          if producer is not None:
24              producer.close()
```

5.6. Skrypt do generowania dummy data

Dane jakie są generowane to temperatura oraz wilgotność. Ze względu na to, że głównym celem projektu była budowa ogólnej architektury, implantacja jest tylko przedstawieniem jej możliwości. Istnieje wiele projektów, w których może zostać wykorzystana zaprezentowana architektura, a stacja meteorologiczna to tylko jedna z możliwości. Generowane dane dla temperatury mieszczą się w przedziale od -20 do 40 stopni Celsjusza, a wilgotność – w przedziale od 30 do 90 procent. Obie wartości są zaokrąglane do dwóch miejsc po przecinku. Oprócz tego wraz ze wspomnianymi danymi zapisywany jest znak czasu (ang. *timestamp*). Jest on niezbędny do określenia, w którym momencie dane zostały zapisane.

Uruchomiony skrypt od generowania danych w czasie rzeczywistym został przedstawiony na rys. 5.7.

```
PS D:\IoT> & C:/Users/Krystian/AppData/Local/Programs/Python/Python311/python.exe d:/IoT/dummy_data.py
2024-07-09T21:19:01.947861,9.25,68.46
2024-07-09T21:19:19.470608,-7.01,66.54
2024-07-09T21:19:34.472250,12.46,69.93
2024-07-09T21:19:49.472922,-9.48,81.53
2024-07-09T21:20:04.474662,-10.77,48.56
2024-07-09T21:20:19.475826,7.58,45.5
2024-07-09T21:20:34.476850,19.53,89.26
2024-07-09T21:20:49.477339,15.21,85.28
2024-07-09T21:21:04.478426,19.4,51.64
2024-07-09T21:21:19.480017,-14.79,59.71
2024-07-09T21:21:34.480862,10.81,65.52
2024-07-09T21:21:49.481298,29.8,44.46
2024-07-09T21:22:04.483895,20.39,32.46
2024-07-09T21:22:19.485455,-9.05,86.29
2024-07-09T21:22:34.487197,15.04,35.79
2024-07-09T21:22:49.488057,-16.79,31.92
2024-07-09T21:23:04.488472,-6.42,40.03
2024-07-09T21:23:19.489765,11.46,76.45
2024-07-09T21:23:34.490649,10.01,86.79
2024-07-09T21:23:49.491920,-15.06,57.14
2024-07-09T21:24:04.492628,28.02,74.25
2024-07-09T21:24:19.493346,-11.67,38.11
2024-07-09T21:24:34.494599,-19.51,48.64
2024-07-09T21:24:49.495488,38.52,47.56
2024-07-09T21:25:04.496309,11.57,49.65
```

5.7. Uruchomienie skryptu do generowanie danych IoT w czasie rzeczywistym

Cały kod wykonuje się w nieskończonej pętli while. Dopóki nie zostanie ona zatrzymana skrypt cały czas będzie tworzył surowe dane. Import Kafka z KafkaProducer był niezbędny do współpracy z Apache Kafka. Skrypt tworzy obiekt KafkaProducer i łączy się z lokalnym serwerem Kafka działającym na porcie 9092. Inicjalizacja producenta jest umieszczona w bloku próby i wyjątku (ang. *try-except*) w celu obsługi potencjalnych błędów podczas połączenia. Kod ten demonstruje podstawowe użycie Apache Kafka do przetwarzania strumieniowego danych w czasie rzeczywistym, co jest szczególnie przydatne w systemach wymagających ciągłego monitorowania i analizy danych, takich jak systemy meteorologiczne.

## 5.4.2 Przetwarzanie danych i zapisywanie danych za pomocą Apache Kafka i Apache ZooKeeper

Aby transformacja danych była możliwa, należy stworzyć platformę do przetwarzania i magazynowania danych. Do tego celu posłuży Apache Kafka oraz Apache ZooKeeper. W pierwszej kolejności została uruchomiona platforma Apache ZooKeeper (rys. 5.8), do czego posłużyła komenda: `.\bin\windows\zookeeper-server-start.bat.\config\zookeeper.properties`

```
C:\Windows\System32\cmd.exe -l <bin\windows\zookeeper-server-start.bat <config\zookeeper.properties

Microsoft Windows [Version 10.0.19045.4529]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

:[kafka]> <bin\windows\zookeeper-server-start.bat <config\zookeeper.properties
[2024-07-09 22:44:38,861] INFO Reading configuration from: <config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-07-09 22:44:38,869] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-07-09 22:44:38,869] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-07-09 22:44:38,870] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-07-09 22:44:38,870] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-07-09 22:44:38,873] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-07-09 22:44:38,873] INFO autopurge.purgeInterval set to 600000 (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-07-09 22:44:38,873] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-07-09 22:44:38,873] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-07-09 22:44:38,874] WARN Either no config or no quorum defined in config, running in standalone mode! (org.apache.zookeeper.server.quorum.QuorumPeerMain)
[2024-07-09 22:44:38,874] INFO Log: 1.2.3mx support not found; jmx disabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2024-07-09 22:44:38,875] INFO Reading configuration from: <config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-07-09 22:44:38,875] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-07-09 22:44:38,875] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-07-09 22:44:38,875] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-07-09 22:44:38,875] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-07-09 22:44:38,876] INFO MetricsServer is initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider@6b143e9 (org.apache.zookeeper.server.ServerMetrics)
[2024-07-09 22:44:38,891] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider@6b143e9 (org.apache.zookeeper.server.ServerMetrics)
[2024-07-09 22:44:38,993] INFO ACL digest algorithm is: SHA1 (org.apache.zookeeper.server.auth.DigestAuthenticationProvider)
[2024-07-09 22:44:38,993] INFO DigestAuthenticationProvider.enabled = true (org.apache.zookeeper.server.auth.DigestAuthenticationProvider)
[2024-07-09 22:44:38,998] INFO zookeeper.snapshot.trustEmpty = false (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2024-07-09 22:44:38,919] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:38,919] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:38,920] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:38,920] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:38,921] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:38,921] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:38,922] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:38,922] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:38,923] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:38,923] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:43,462] INFO Server environment.version=3.8.0.202 (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:43,462] INFO Server environment.host=DESKTOP-1H5A477-B (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:43,462] INFO Server environment.environment=java version=1.8_202 (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:43,462] INFO Server environment.java.vendor=Oracle Corporation (org.apache.zookeeper.server.ZooKeeperServer)
[2024-07-09 22:44:43,462] INFO Server environment.java.home=C:\Program Files\Java\jdk_8.0_202\jre (org.apache.zookeeper.server.ZooKeeperServer)
```

## 5.8. Uruchomienie Apache Zookeeper

W momencie, kiedy ZooKeeper już działa można uruchomić Apache Kafka, do czego służy komenda: `.\bin\windows\kafka-server-start.bat.\config\server.properties`, co zostało przedstawione na rys. 5.9.

## 5.9. Uruchomienie Apache Kafka

W momencie, w którym Apache Kafka oraz ZooKeeper są uruchomione, możliwe jest uruchomienie skryptu Python, który zapisuje dane w Apache Kafka w temacie real-time-data. Zostało to przedstawione na rys. 5.10. Za pomocą komendy: kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic real-time-data jesteśmy w stanie zobaczyć zawartość, która obecnie jest przetwarzana i zapisywana w Apache Kafka pod localhostem na porcie 9092.

```
D:\kafka\bin\windows>
D:\kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic real-time-data
2024-07-08T22:27:29.237217,39.36,38.45
2024-07-08T22:27:44.237788,33.02,45.24
2024-07-08T22:27:59.238602,-3.21,38.66
2024-07-08T22:28:14.239930,14.04,42.44
```

5.10. Nasłuchiwanie tematu w Apache Kafka

#### 5.4.3 Transformacja danych za pomocą Javy

Kolejnym krokiem jest transformacja danych, co oznacza, że surowe dane będą przetwarzane według konkretnych zasad i reguł (usuwanie szumów, zmienianie typów danych itp.) w celu osiągnięcia konkretnego rezultatu. Na rys. 5.11 przedstawiony został najważniejszy fragment kodu, który odpowiada za zapisywanie, magazynowanie oraz transformowanie danych. Po zimportowaniu najważniejszych bibliotek i modułów nastepnym krokiem jest połącznie z nowym tematem w Apache Kafka, jakim jest real-time-data-processing na porcie 9092 local hosta. Kolejnym etapem jest sprawdzenie, czy dane które obecnie są zapisane w temacie real-time-data, są zapisane w prawidłowym formacie (nie są puste oraz zawierają trzy informacje: znak czasu, temperatura oraz wilgotność). Następnie kolejnym krokiem jest transformacja, co oznacza sprawdzenie dla konkretnego przypadku biznesowego wartości krytycznej, tj. sytuacji, w której temperatura przekraczała 38 stopni Celsjusza. Jeżeli taka sytuacja by wystąpiła, zapisany zostałby alert z informacją ostrzegawczą dla osoby przeglądającej dane, aby sprawdziła przyczynę wystąpienia awarii. Takie alerty są niezbędne do kontrolowania i monitorowania sprawności architektury i systemu. Dzięki nim istnieje możliwość reagowania z wyprzedzeniem na ryzyko np. przegrzania sensora. Ostatnim, lecz nie mniej ważnym, krokiem było zapisanie uzyskanych danych w lokalizacji D:\\kafka\\processed-data.json w formacie JSON, dzięki czemu możliwa była aktualizacja w czasie rzeczywistym, gdy tylko nowe dane zostały dopisane do bazy danych.

```

16  ▶  no usages
17  public static void main(String[] args) {
18      Properties props = new Properties();
19      props.put(StreamsConfig.APPLICATION_ID_CONFIG, "real-time-data-processing");
20      props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
21      props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
22      props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
23
24      StreamsBuilder builder = new StreamsBuilder();
25
26      KStream<String, String> inputStream = builder.stream("topic:real-time-data");
27
28      KStream<String, String> processedStream = inputStream.mapValues(value -> {
29          if (value.isEmpty()) {
30              System.err.println("An empty string was received");
31              return null;
32          }
33
34          String[] parts = value.split(",");
35          if (parts.length != 3) { // Expecting timestamp, temperature, humidity
36              System.err.println("Incorrect data format: " + value);
37              return null;
38          }
39
40          try {
41              String timestamp = parts[0];
42              double temperature = Double.parseDouble(parts[1]);
43              double humidity = Double.parseDouble(parts[2]);
44              if (temperature > 38) {
45                  return "{\"timestamp\": " + timestamp + ", \"temperature\": " + temperature + ", \"humidity\": " + humidity + ", \"alert\": \"ALERT! High temperature\"}";
46              } else {
47                  return "{\"timestamp\": " + timestamp + ", \"temperature\": " + temperature + ", \"humidity\": " + humidity + "}";
48              }
49          } catch (NumberFormatException e) {
50              // Number parsing error handling
51              System.err.println("Number parsing error: " + e.getMessage());
52              return null;
53          }
54      });
55
56      processedStream.to("processed-real-time-data");
57
58      processedStream.foreach((key, value) -> {
59          if (value != null) {
60              try (PrintWriter writer = new PrintWriter(new FileWriter(fileName, true))) {
61                  writer.println(value);
62              } catch (IOException e) {
63                  System.err.println("Error while writing to file: " + e.getMessage());
64              }
65          }
66      });
67
68      KafkaStreams streams = new KafkaStreams(builder.build(), props);
69      streams.start();
70  }

```

5.11. Skrypt napisany w języku Java do transformacji danych

Uruchomiony skrypt nasłuchiwał w czasie rzeczywistym. Każda nowa informacja zostaje dodana jako surowa do tematu real-time-data i od razu jest transformowana do prawidłowego formatu danych przetworzonych na temacie real-time-data-processing. Proces przetwarzania danych został przedstawiony na rys. 5.12.

5.12. Uruchomienie skryptu do przetwarzania danych

Następnym krokiem jest nasłuchiwanie danych, czy są one przetwarzane w prawidłowym formacie. Uzyskiwane jest to za pomocą komendy: kafka-console-consumer --bootstrap-server localhost:9092 --topic processed-real-time-data w konsoli CMD. Proces ten został przedstawiony na rysunku 5.13. Odczyt dla sensora zanotował wartość temperatury 39.36 wartość ta przekracza wartość 38 stopni Celsjusza, co zostało odnotowane jako przekroczenie

wartości krytycznej, dlatego obok rekordu zawierających informacje o odczycie danych z tej godziny została dodana informacja w alercie o tym, że temperatura jest za wysoka.

```
{"timestamp": "2024-07-08T22:24:59.225860", "temperature": 2.63, "humidity": 89.28}
{"timestamp": "2024-07-08T22:25:14.227114", "temperature": 0.75, "humidity": 89.67}
{"timestamp": "2024-07-08T22:25:29.228081", "temperature": -13.24, "humidity": 41.56}
{"timestamp": "2024-07-08T22:25:44.229247", "temperature": -1.05, "humidity": 59.4}
{"timestamp": "2024-07-08T22:25:59.230422", "temperature": 0.04, "humidity": 74.2}
{"timestamp": "2024-07-08T22:26:14.231893", "temperature": 25.11, "humidity": 80.69}
{"timestamp": "2024-07-08T22:26:29.232879", "temperature": 20.51, "humidity": 54.68}
{"timestamp": "2024-07-08T22:26:44.233518", "temperature": -12.24, "humidity": 56.39}
{"timestamp": "2024-07-08T22:26:59.234250", "temperature": 14.93, "humidity": 84.96}
 {"timestamp": "2024-07-08T22:27:14.235333", "temperature": 20.73, "humidity": 44.96}
 {"timestamp": "2024-07-08T22:27:29.237217", "temperature": 39.36, "humidity": 38.45, "alert": "ALERT! High temperature"}
 {"timestamp": "2024-07-08T22:27:44.237788", "temperature": 33.02, "humidity": 45.24}
 {"timestamp": "2024-07-08T22:27:59.238602", "temperature": -3.21, "humidity": 61.84}
```

### 5.13. Nasłuchiwanie real-time-data-processing w Apache Kafka

Przetworzone dane zostają zapisane w pliku JSON, co przedstawiono na rysunku 5.14. Następnym krokiem było utworzenie skryptu w Pythonie, który w Power Bi odświeża dane, jeśli do pliku zostaną dodane nowe. W momencie, w którym dane zostały dostosowane do aplikacji możliwe jest wykorzystać ich do analizy danych. Zadanie te zostanie wykonane w Power Bi, gdzie dane zostały zainportowane z magazynu danych.

```
d: > kafka > | processed-data.json | ●
dummy_data.py • 0 processed-data.json •
1 {"timestamp": "2024-07-08T22:22:59.215426", "temperature": 9.57, "humidity": 35.66}
2 {"timestamp": "2024-07-08T22:23:14.216247", "temperature": -4.52, "humidity": 76.79}
3 {"timestamp": "2024-07-08T22:23:29.217049", "temperature": 36.1, "humidity": 81.05}
4 {"timestamp": "2024-07-08T22:23:44.218471", "temperature": -9.66, "humidity": 44.07}
5 {"timestamp": "2024-07-08T22:23:59.219596", "temperature": 34.87, "humidity": 48.52}
6 {"timestamp": "2024-07-08T22:24:14.221079", "temperature": 30.55, "humidity": 63.73}
7 {"timestamp": "2024-07-08T22:24:29.222741", "temperature": 23.98, "humidity": 38.48}
8 {"timestamp": "2024-07-08T22:24:44.223634", "temperature": 23.3, "humidity": 32.89}
9 {"timestamp": "2024-07-08T22:24:59.225860", "temperature": 2.63, "humidity": 89.28}
10 {"timestamp": "2024-07-08T22:25:14.227114", "temperature": 0.75, "humidity": 89.67}
11 {"timestamp": "2024-07-08T22:25:29.228081", "temperature": -13.24, "humidity": 41.56}
12 {"timestamp": "2024-07-08T22:25:44.229247", "temperature": -1.05, "humidity": 59.4}
13 {"timestamp": "2024-07-08T22:25:59.230422", "temperature": 0.04, "humidity": 74.2}
14 {"timestamp": "2024-07-08T22:26:14.231893", "temperature": 25.11, "humidity": 80.69}
15 {"timestamp": "2024-07-08T22:26:29.232879", "temperature": 20.51, "humidity": 54.68}
16 {"timestamp": "2024-07-08T22:26:44.233518", "temperature": -12.24, "humidity": 56.39}
17 {"timestamp": "2024-07-08T22:26:59.234250", "temperature": 14.93, "humidity": 84.96}
18 {"timestamp": "2024-07-08T22:27:14.235333", "temperature": 20.73, "humidity": 44.96}
19 {"timestamp": "2024-07-08T22:27:29.237217", "temperature": 39.36, "humidity": 38.45, "alert": "ALERT! High temperature"}
20 {"timestamp": "2024-07-08T22:27:44.237788", "temperature": 33.02, "humidity": 45.24}
21 {"timestamp": "2024-07-08T22:27:59.238602", "temperature": -3.21, "humidity": 61.84}
22 {"timestamp": "2024-07-08T22:28:14.239930", "temperature": 14.04, "humidity": 42.44}
23 {"timestamp": "2024-07-08T22:28:29.240941", "temperature": -15.96, "humidity": 41.77}
24 {"timestamp": "2024-07-08T22:28:44.241982", "temperature": 23.13, "humidity": 41.08}
25 {"timestamp": "2024-07-08T22:28:59.242959", "temperature": 37.28, "humidity": 38.85}
26 {"timestamp": "2024-07-08T22:29:14.244547", "temperature": 11.91, "humidity": 57.03}
27 {"timestamp": "2024-07-08T22:29:29.245566", "temperature": 13.92, "humidity": 66.89}
28 {"timestamp": "2024-07-08T22:29:44.246566", "temperature": 19.62, "humidity": 78.46}
29 {"timestamp": "2024-07-08T22:29:59.247780", "temperature": 17.61, "humidity": 34.88}
30 {"timestamp": "2024-07-08T22:30:14.249866", "temperature": 39.29, "humidity": 47.19, "alert": "ALERT! High temperature"}
31 {"timestamp": "2024-07-08T22:30:29.150284", "temperature": 15.4, "humidity": 82.13}
32 {"timestamp": "2024-07-08T22:30:44.251475", "temperature": 18.6, "humidity": 88.48}
33 {"timestamp": "2024-07-08T22:30:59.252532", "temperature": -16.4, "humidity": 81.34}
34 {"timestamp": "2024-07-08T22:31:14.253956", "temperature": -7.03, "humidity": 60.3}
35 {"timestamp": "2024-07-08T22:31:29.254890", "temperature": 3.54, "humidity": 78.17}
36 {"timestamp": "2024-07-08T22:31:44.255643", "temperature": 14.19, "humidity": 67.14}
37 {"timestamp": "2024-07-08T22:31:59.256559", "temperature": 16.64, "humidity": 42.15}
38 {"timestamp": "2024-07-08T22:32:14.257947", "temperature": 9.05, "humidity": 72.91}
39 {"timestamp": "2024-07-08T22:32:29.259398", "temperature": 37.48, "humidity": 65.71}
40 {"timestamp": "2024-07-08T22:32:44.261102", "temperature": 4.27, "humidity": 84.7}
41 {"timestamp": "2024-07-08T22:32:59.261724", "temperature": -3.21, "humidity": 56.62}
42 {"timestamp": "2024-07-08T22:33:14.262778", "temperature": -5.86, "humidity": 74.71}
43 {"timestamp": "2024-07-08T22:33:29.264413", "temperature": 30.82, "humidity": 47.25}
44 {"timestamp": "2024-07-08T22:33:44.264977", "temperature": 11.18, "humidity": 78.44}
45 {"timestamp": "2024-07-08T22:33:59.266169", "temperature": 19.78, "humidity": 74.86}
46 {"timestamp": "2024-07-08T22:34:14.267113", "temperature": -3.13, "humidity": 89.82}
47 {"timestamp": "2024-07-08T22:34:29.268894", "temperature": -12.26, "humidity": 70.34}
48 {"timestamp": "2024-07-08T22:34:44.269965", "temperature": -17.01, "humidity": 34.62}
49 {"timestamp": "2024-07-08T22:34:59.270396", "temperature": -18.85, "humidity": 79.15}
50 {"timestamp": "2024-07-08T22:35:14.271786", "temperature": 3.81, "humidity": 35.08}
51 {"timestamp": "2024-07-08T22:35:29.273196", "temperature": -19.79, "humidity": 34.96}
52 {"timestamp": "2024-07-08T22:35:44.274189", "temperature": 22.15, "humidity": 75.26}
53 {"timestamp": "2024-07-08T22:35:59.274784", "temperature": -1.42, "humidity": 79.17}
54 {"timestamp": "2024-07-08T22:36:14.275613", "temperature": 0.67, "humidity": 88.79}
55 {"timestamp": "2024-07-08T22:36:29.276282", "temperature": -2.33, "humidity": 69.07}
56 {"timestamp": "2024-07-08T22:36:44.277522", "temperature": 36.71, "humidity": 54.51}
57 {"timestamp": "2024-07-08T22:36:59.278393", "temperature": -0.23, "humidity": 62.63}
58 {"timestamp": "2024-07-08T22:37:14.279013", "temperature": -11.49, "humidity": 69.66}
59 {"timestamp": "2024-07-08T22:37:29.280592", "temperature": -15.16, "humidity": 45.73}
60 {"timestamp": "2024-07-08T22:37:44.282142", "temperature": -11.24, "humidity": 89.25}
61 {"timestamp": "2024-07-08T22:37:59.283277", "temperature": 37.83, "humidity": 88.87}
62 {"timestamp": "2024-07-08T22:38:14.284751", "temperature": 23.88, "humidity": 66.83}
63 {"timestamp": "2024-07-08T22:38:29.286301", "temperature": 36.71, "humidity": 47.4}
64 {"timestamp": "2024-07-08T22:38:44.287448", "temperature": 13.45, "humidity": 84.02}
65 {"timestamp": "2024-07-08T22:38:59.288346", "temperature": 15.54, "humidity": 60.45}
66 {"timestamp": "2024-07-08T22:39:14.289978", "temperature": -4.96, "humidity": 45.62}
67 {"timestamp": "2024-07-08T22:39:29.291276", "temperature": -17.08, "humidity": 61.84}
68 {"timestamp": "2024-07-08T22:39:44.292401", "temperature": 34.28, "humidity": 84.39}
```

5.14. Przetworzone dane zapisane w formacie JSON.

#### 5.3.4 Analiza danych w Power Bi

Ostatnim krokiem jest wykorzystanie Power Bi który w prezentowanej architekturze posłuży jako artefakt, czyli to, co zwraca architektura do przetwarzania danych w czasie rzeczywistym. Import danych został wykonany na rysunku 5.15, jednak dane nie były wystarczające, dlatego zostały rozwinięte, co przedstawia rysunek 5.16.

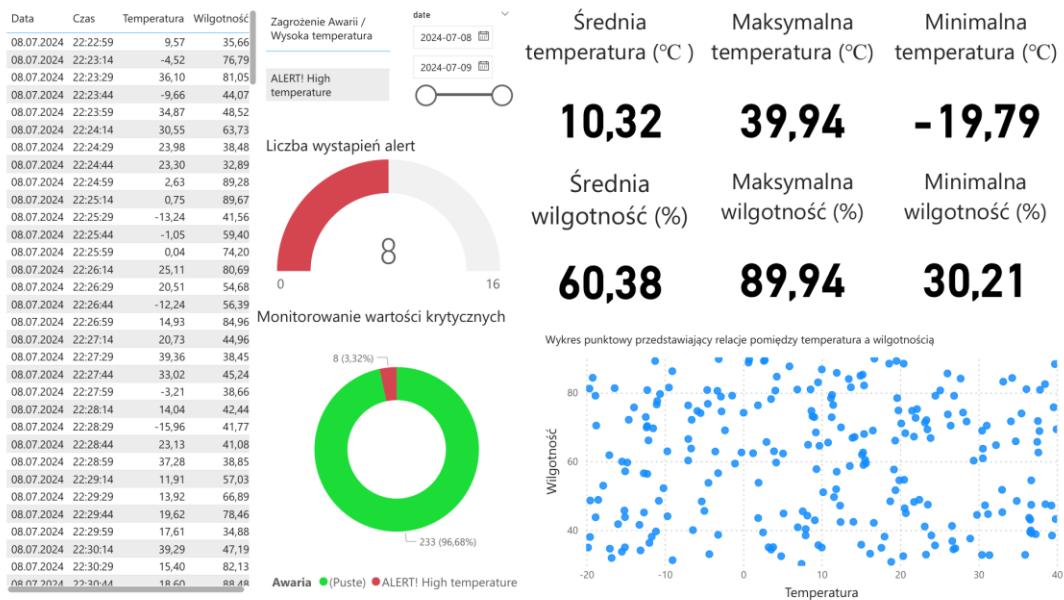
	timestamp	temperature	humidity	alert
	08.07.2024 22:22:59	9,57	35,66	
	08.07.2024 22:23:14	-4,52	76,79	
	08.07.2024 22:23:29	36,1	81,05	
	08.07.2024 22:23:44	-9,66	44,07	
	08.07.2024 22:23:59	34,87	48,52	
	08.07.2024 22:24:14	30,55	63,73	
	08.07.2024 22:24:29	23,98	38,48	
	08.07.2024 22:24:44	23,3	32,89	
	08.07.2024 22:24:59	2,63	89,28	
	08.07.2024 22:25:14	0,75	89,67	
	08.07.2024 22:25:29	-13,24	41,56	
	08.07.2024 22:25:44	-1,05	59,4	
	08.07.2024 22:25:59	0,04	74,2	
	08.07.2024 22:26:14	25,11	80,69	
	08.07.2024 22:26:29	20,51	54,68	
	08.07.2024 22:26:44	-12,24	56,39	
	08.07.2024 22:26:59	14,93	84,96	
	08.07.2024 22:27:14	20,73	44,96	
	08.07.2024 22:27:29	39,36	38,45	ALERT! High temperature
	08.07.2024 22:27:44	33,02	45,24	
	08.07.2024 22:27:59	-3,21	38,66	
	08.07.2024 22:28:14	14,04	42,44	
	08.07.2024 22:28:29	-15,96	41,77	
	08.07.2024 22:28:44	23,13	41,08	
	08.07.2024 22:28:59	37,28	38,85	
	08.07.2024 22:29:14	11,91	57,03	
	08.07.2024 22:29:29	13,92	66,89	
	08.07.2024 22:29:44	19,62	78,46	
	08.07.2024 22:29:59	17,61	34,88	
	08.07.2024 22:30:14	39,29	47,19	ALERT! High temperature
	08.07.2024 22:30:29	15,4	82,13	
	08.07.2024 22:30:44	18,6	88,48	
	08.07.2024 22:30:59	-16,4	81,34	
	08.07.2024 22:31:14	-7,03	60,3	
	08.07.2024 22:31:29	3,54	78,17	
	08.07.2024 22:31:44	14,19	67,14	
	08.07.2024 22:31:59	16,64	42,15	
	08.07.2024 22:32:14	9,05	72,91	
	08.07.2024 22:32:29	37,48	65,71	
	08.07.2024 22:32:44	4,27	84,7	

5.15. Zimportowane dane po transformacji

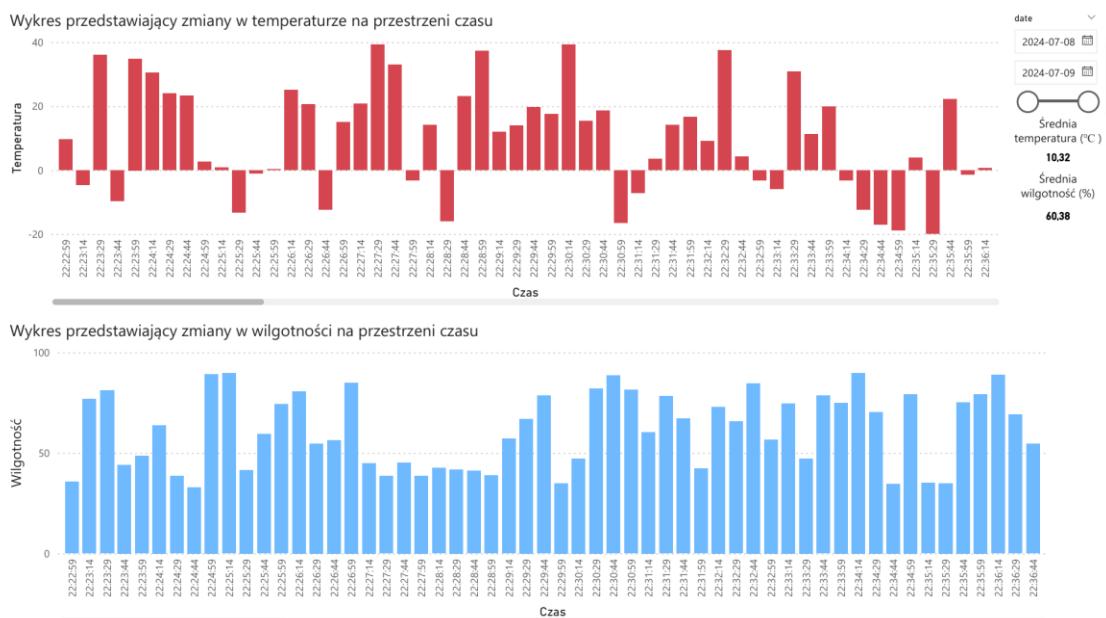
Struktura	Formatowanie	Właściwości	Sortuj	Grupy	Relacje	Oblizienia					
timestamp	temperature	humidity	alert	time	date	temperature_AVG	humidity_AVG	temperature_Max	temperature_Min	humidity_Max	humidity_Min
2024-07-08T22:22:59	9.57	35.66		22:22:59	08.07.2024	9.57	35.66	9.57	9.57	35.66	35.66
2024-07-08T22:23:14	-4.52	76.79		22:23:14	08.07.2024	-4.52	76.79	-4.52	-4.52	76.79	76.79
2024-07-08T22:23:29	36.1	81.05		22:23:29	08.07.2024	36.1	81.05	36.1	36.1	81.05	81.05
2024-07-08T22:23:44	-9.66	44.07		22:23:44	08.07.2024	-9.66	44.07	-9.66	-9.66	44.07	44.07
2024-07-08T22:23:59	34.87	48.52		22:23:59	08.07.2024	34.87	48.52	34.87	34.87	48.52	48.52
2024-07-08T22:24:14	30.55	63.73		22:24:14	08.07.2024	30.55	63.73	30.55	30.55	63.73	63.73
2024-07-08T22:24:29	23.98	38.48		22:24:29	08.07.2024	23.98	38.48	23.98	23.98	38.48	38.48
2024-07-08T22:24:44	23.3	32.89		22:24:44	08.07.2024	23.3	32.89	23.3	23.3	32.89	32.89
2024-07-08T22:24:59	2.63	89.28		22:24:59	08.07.2024	2.63	89.28	2.63	2.63	89.28	89.28
2024-07-08T22:25:14	0.75	89.67		22:25:14	08.07.2024	0.75	89.67	0.75	0.75	89.67	89.67
2024-07-08T22:25:29	-13.24	41.56		22:25:29	08.07.2024	-13.24	41.56	-13.24	-13.24	41.56	41.56
2024-07-08T22:25:44	-1.05	59.4		22:25:44	08.07.2024	-1.05	59.4	-1.05	-1.05	59.4	59.4
2024-07-08T22:25:59	0.04	74.2		22:25:59	08.07.2024	0.04	74.2	0.04	0.04	74.2	74.2
2024-07-08T22:26:14	25.11	80.69		22:26:14	08.07.2024	25.11	80.69	25.11	25.11	80.69	80.69
2024-07-08T22:26:29	20.51	54.68		22:26:29	08.07.2024	20.51	54.68	20.51	20.51	54.68	54.68
2024-07-08T22:26:44	-12.24	56.39		22:26:44	08.07.2024	-12.24	56.39	-12.24	-12.24	56.39	56.39
2024-07-08T22:26:59	14.93	84.96		22:26:59	08.07.2024	14.93	84.96	14.93	14.93	84.96	84.96
2024-07-08T22:27:14	20.73	44.96		22:27:14	08.07.2024	20.73	44.96	20.73	20.73	44.96	44.96
2024-07-08T22:27:29	33.02	45.24		22:27:24	08.07.2024	33.02	45.24	33.02	33.02	45.24	45.24
2024-07-08T22:27:39	-3.21	36.66		22:27:39	08.07.2024	-3.21	36.66	-3.21	-3.21	36.66	36.66
2024-07-08T22:28:14	14.04	42.44		22:28:14	08.07.2024	14.04	42.44	14.04	14.04	42.44	42.44
2024-07-08T22:28:29	-15.96	41.77		22:28:29	08.07.2024	-15.96	41.77	-15.96	-15.96	41.77	41.77
2024-07-08T22:28:44	23.13	41.08		22:28:44	08.07.2024	23.13	41.08	23.13	23.13	41.08	41.08
2024-07-08T22:28:59	37.28	38.85		22:28:59	08.07.2024	37.28	38.85	37.28	37.28	38.85	38.85
2024-07-08T22:29:14	11.91	57.03		22:29:14	08.07.2024	11.91	57.03	11.91	11.91	57.03	57.03
2024-07-08T22:29:29	13.92	66.89		22:29:29	08.07.2024	13.92	66.89	13.92	13.92	66.89	66.89
2024-07-08T22:29:44	19.62	78.46		22:29:44	08.07.2024	19.62	78.46	19.62	19.62	78.46	78.46
2024-07-08T22:29:59	17.61	34.88		22:29:59	08.07.2024	17.61	34.88	17.61	17.61	34.88	34.88
2024-07-08T22:30:29	15.4	82.13		22:30:29	08.07.2024	15.4	82.13	15.4	15.4	82.13	82.13
2024-07-08T22:30:44	18.6	88.48		22:30:44	08.07.2024	18.6	88.48	18.6	18.6	88.48	88.48
2024-07-08T22:30:59	-16.4	81.34		22:30:59	08.07.2024	-16.4	81.34	-16.4	-16.4	81.34	81.34
2024-07-08T22:31:14	-7.03	60.3		22:31:14	08.07.2024	-7.03	60.3	-7.03	-7.03	60.3	60.3
2024-07-08T22:31:29	3.54	78.17		22:31:29	08.07.2024	3.54	78.17	3.54	3.54	78.17	78.17
2024-07-08T22:31:44	14.19	67.14		22:31:44	08.07.2024	14.19	67.14	14.19	14.19	67.14	67.14
2024-07-08T22:31:59	16.64	42.15		22:31:59	08.07.2024	16.64	42.15	16.64	16.64	42.15	42.15
2024-07-08T22:32:14	9.05	72.91		22:32:14	08.07.2024	9.05	72.91	9.05	9.05	72.91	72.91
2024-07-08T22:32:29	37.48	65.71		22:32:29	08.07.2024	37.48	65.71	37.48	37.48	65.71	65.71
2024-07-08T22:32:44	4.27	84.7		22:32:44	08.07.2024	4.27	84.7	4.27	4.27	84.7	84.7
2024-07-08T22:32:59	-3.21	56.62		22:32:59	08.07.2024	-3.21	56.62	-3.21	-3.21	56.62	56.62
2024-07-08T22:33:14	-5.86	74.71		22:33:14	08.07.2024	-5.86	74.71	-5.86	-5.86	74.71	74.71
2024-07-08T22:33:29	30.82	47.25		22:33:29	08.07.2024	30.82	47.25	30.82	30.82	47.25	47.25
2024-07-08T22:33:44	11.18	78.44		22:33:44	08.07.2024	11.18	78.44	11.18	11.18	78.44	78.44
2024-07-08T22:33:59	19.78	74.86		22:33:59	08.07.2024	19.78	74.86	19.78	19.78	74.86	74.86
2024-07-08T22:34:14	-3.13	89.82		22:34:14	08.07.2024	-3.13	89.82	-3.13	-3.13	89.82	89.82
2024-07-08T22:34:29	-12.26	70.34		22:34:29	08.07.2024	-12.26	70.34	-12.26	-12.26	70.34	70.34
2024-07-08T22:34:44	-17.01	34.62		22:34:44	08.07.2024	-17.01	34.62	-17.01	-17.01	34.62	34.62
2024-07-08T22:34:59	-18.85	79.15		22:34:59	08.07.2024	-18.85	79.15	-18.85	-18.85	79.15	79.15
2024-07-08T22:35:14	3.81	35.08		22:35:14	08.07.2024	3.81	35.08	3.81	3.81	35.08	35.08
2024-07-08T22:35:29	-19.79	34.96		22:35:29	08.07.2024	-19.79	34.96	-19.79	-19.79	34.96	34.96
2024-07-08T22:35:44	22.15	75.26		22:35:44	08.07.2024	22.15	75.26	22.15	22.15	75.26	75.26
2024-07-08T22:35:59	-1.42	79.17		22:35:59	08.07.2024	-1.42	79.17	-1.42	-1.42	79.17	79.17
2024-07-08T22:36:14	0.67	88.79		22:36:14	08.07.2024	0.67	88.79	0.67	0.67	88.79	88.79
2024-07-08T22:36:29	-2.33	69.07		22:36:29	08.07.2024	-2.33	69.07	-2.33	-2.33	69.07	69.07
2024-07-08T22:36:44	36.71	54.51		22:36:44	08.07.2024	36.71	54.51	36.71	36.71	54.51	54.51
2024-07-08T22:36:59	-0.23	62.63		22:36:59	08.07.2024	-0.23	62.63	-0.23	-0.23	62.63	62.63
2024-07-08T22:37:14	-11.49	69.66		22:37:14	08.07.2024	-11.49	69.66	-11.49	-11.49	69.66	69.66

5.16. Tabela zawierająca dane po filtrowaniu i agregacji

Po zaimportowaniu danych kolejnym krokiem była ich rozbudowa. Timestamp został podzielony na dzień, miesiąc, rok oraz godzinę, minutę i sekundę. Krok ten był niezbędny do dokładniejszej analizy danych. Następnie zostało dodanych 6 nowych kolumn – 3 na temperaturę oraz 3 dla wilgotności. Zawierały one informacje o średniej, minimalnej oraz maksymalnej wartości dla tych parametrów. Tabela zawierająca informacje z naszych urządzeń IoT była gotowa do stworzenia wykresów oraz wskaźników niezbędnych do prawidłowej analizy. W pierwszej kolejności zostało zdefiniowane, jakie dane uważane są za konkretne oraz sposób, w jaki potencjalni klient (lub np. analityk danych) chciałby na nich operować. Następnie zostały stworzone dwa pulpity nawigacyjne przystosowane do konkretnych analiz. Rysunek 5.17 przedstawia główny pulpit nawigacyjny, a Rysunek 5.18 zawiera bardziej szczegółowe informacje dotyczące zmian w temperaturze oraz wilgotności na przestrzeni czasu. Oprócz tego dwa dashboardy zawierają możliwość implementacji filtrów w zależności od potrzeb użytkownika. Dzięki temu w zależności od sytuacji możliwe jest przejście od ogólnej analizy do bardziej szczegółowej z dokładnością do konkretnego momentu w historii poprzez podanie konkretnej daty i godziny.



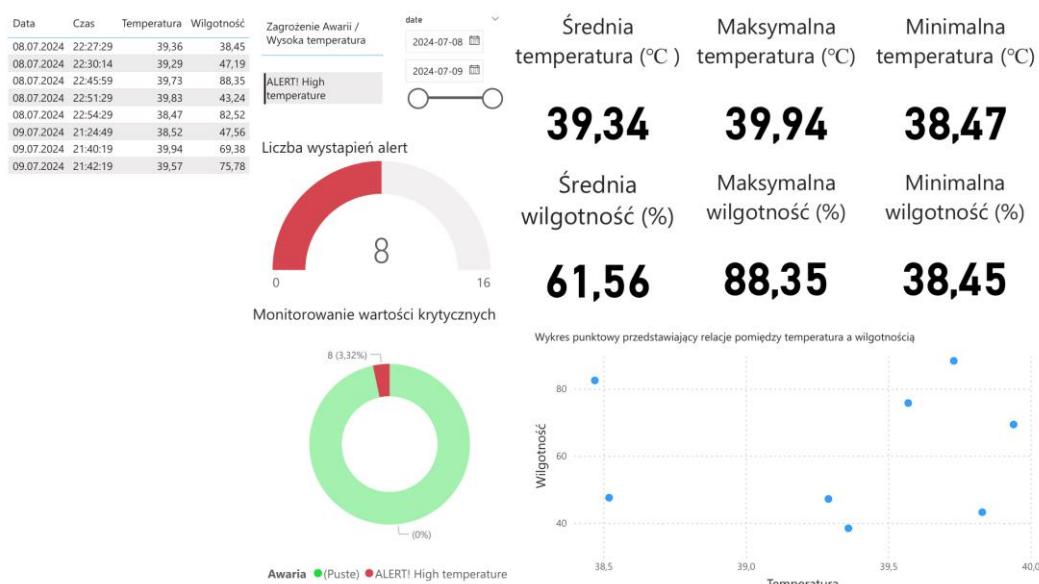
5.17. Pulpit głównej aplikacji do sprawdzania danych w czasie rzeczywistym



5.18. Pulpit pomocniczy do analizy konkretnych zmian w temperaturze i wilgotności na przestrzeni czasu

Zaczynając od lewej strony głównego pulpitu nawigacyjnego, posiadamy tabele zawierające kolejno datę, czas, temperaturę i wilgotność w konkretnym momencie. Od góry po środku widoczny jest katalog dwóch typów alertów – brak (ang. *NULL*) lub **ALERT!** High temperaturę oznaczający wartość krytyczną. Obok niego znajduje się data, która również służy do filtrowania w określonym czasie. Niżej zlokalizowane są dwa alerty, które odnoszą się do ilości wystąpień alertów i tego, czy maksymalna ilość została przekroczona; oraz monitorowanie, ile procent danych z IoT przekroczyło wartość krytyczną. W prawej części pulpitu zostało utworzone sześć kart informacyjnych, które zawierają informacje o średniej, minimalnej

i maksymalnej temperaturze oraz wilgotności w danym odstępie czasowym. Pod kartami znajduje się wykres punktowy, prezentujący wartości temperatury oraz wilgotności. Drugi pulpit nawigacyjny również zawiera informacje dotyczące wilgotności i temperatury, jednak ponadto pokazuje na dwóch wykresach, jak na przestrzeni czasu te wartości ulegały zmianie. Na potrzeby sprawdzenia działania architektury zbierane były dane na przestrzeni dwóch dni. Pomimo krótkiego czasu działania projekt mieści się w kategorii Big Data, dzięki czemu w bardzo krótkim czasie zostało uzyskanych około 250 rekordów, a wartość ta rosłaby w tempie wykładowiczym, jeśli proces pozyskiwania danych w czasie rzeczywistym pozostałyby włączony na dłużej. Jedną z najważniejszych informacji, jakie możemy odczytać z zaprezentowanych pulpitów jest to, że na przestrzeni obserwowanego czasu wartość krytyczna została przekroczona 8 razy, co stanowiło 3.32% wszystkich rekordów, a więc awarie nie występowały często, jednak się zdarzały. Oprócz tego średnia temperatura wyniosła 10.32°C, minimalna – -19.79°C, a maksymalna – 39.94°C. Dla wilgotności wartości te były następujące: średnia wilgotność wyniosła 60.38%, maksymalna – 89,94%, a minimalna 30.21%. Jeśli chodzi o wykres punktowy, ciężko dopatrzyć się jakiejś konkretnej korelacji pomiędzy danymi. Było to prawdopodobnie spowodowane przez fikcyjne dane generowane losowo. Wykres ten mógł pokazać jakąś korelację, na przykład, że wraz z wysoką wartością występuje też wysoka wilgotność. Również na bardziej szczegółowym pulpicie drugim dane zostały przedstawione w sposób ogólny ze względu na fikcyjne dane. Po włączeniu filtra jesteśmy w stanie poddać analizie konkretne przypadki dotyczące wystąpienie alertu. Wartości na pozostałych wykresach zostają zmienione, dzięki czemu możemy bardziej indywidualnie rozpatrzyć każdy przypadek, co zostało przedstawione na rysunku 5.19. Platforma w Power Bi może być bardziej rozwijana, a przedstawione wizualizacje danych są tylko przedstawieniem możliwości, jakie daje nam architektura Big Data do przetwarzania danych w czasie rzeczywistym.



5.19. Główny pulpit nawigacyjny z użyciem filtr

## 5.5. Podsumowanie

### 5.5.1 Porównanie z architekturą wykorzystującą Azure

W rozważaniach nad wyborem odpowiedniej architektury dla systemu meteorologicznego przedstawiono dwie alternatywy: architekturę wykorzystującą platformę Azure oraz lokalną architekturę dla urządzeń IoT. Architektury mają swoje unikalne cechy, które wpływają na sposób zbierania, przetwarzania, przechowywania i wizualizacji danych.

Architektura oparta na platformie Azure wykorzystuje zaawansowane technologie chmurowe do obsługi rzeczywistych danych zbieranych za pomocą sensorów IoT. Dane są przesyłane do chmury, gdzie są przetwarzane w czasie rzeczywistym za pomocą usług takich jak Azure Stream Analytics i Azure Databricks. Przetworzone dane są przechowywane w Azure Data Lake i Azure SQL Database, co zapewnia skalowalność i dostępność. Wizualizacja danych odbywa się za pomocą Power BI, który umożliwia tworzenie interaktywnych raportów i pulpitów nawigacyjnych. Zarządzanie metadanymi i monitorowanie systemu są realizowane przez narzędzia dostarczane przez Azure, co odciąża zespół IT z wykonywania wielu zadań administracyjnych.

Druga rozważana architektura to lokalne rozwiązanie, które wykorzystuje fikcyjne dane generowane przez skrypty Python. Dane te symulują odczyty z sensorów IoT, co pozwala na przeprowadzanie testów i prototypowanie bez konieczności korzystania z rzeczywistych sensorów. Przetwarzanie danych odbywa się lokalnie za pomocą aplikacji napisanych w języku Java, a przechowywanie danych realizowane jest przy użyciu Apache Kafka. Zarządzanie metadanymi i synchronizacją klastrów zapewnia Apache ZooKeeper. Wizualizacja danych jest realizowana lokalnie za pomocą Power BI, co umożliwia tworzenie raportów i analiz na podstawie symulowanych danych.

W kontekście zbierania danych, w architekturze Azure dane rzeczywiste zbierane są za pomocą sensorów IoT, podczas gdy w lokalnej architekturze wykorzystywane są symulowane dane generowane lokalnie. Jeśli chodzi o przetwarzanie danych, w architekturze Azure odbywa się ono w czasie rzeczywistym z użyciem zaawansowanych usług chmurowych, natomiast w lokalnej architekturze dane są przetwarzane lokalnie za pomocą aplikacji Java. Magazynowanie danych w architekturze Azure realizowane jest w skalowalnych magazynach danych Azure, takich jak Azure Data Lake i Azure SQL Database, podczas gdy w lokalnej architekturze dane są przechowywane w Apache Kafka. Wizualizacja danych w architekturze Azure odbywa się za pomocą Power BI na platformie Azure, co zapewnia dostęp do zaawansowanych funkcji chmurowych, podczas gdy w lokalnej architekturze Power BI jest używany lokalnie, co może ograniczać niektóre funkcje w porównaniu z chmurą. Zarządzanie w architekturze Azure korzysta z zarządzania usługami, bezpieczeństwem i skalowalnością realizowanego przez platformę

Azure, natomiast w lokalnej architekturze zarządzanie infrastrukturą i usługami odbywa się lokalnie, co wymaga większej ilości zasobów i umiejętności administracyjnych.

Na korzyść architektury korzystającej z Azure przemawia to, że jest ona bardziej skalowalna, działa zdecydowanie szybciej oraz ma potencjalnie nieograniczone miejsce do składowania i zapisywania danych i zasobów. Do jej zalet można zaliczyć również to, że ryzyko wystąpienia awarii jest zdecydowanie mniejsze niż dla lokalnej architektury. Największą wadą takiej architektury są jednak zawsze koszty, które mogą rosnąć w sposób wykładniczy. Oznacza to, że jedna z największych zalet architektury, skalowalność, może stać się jej wadą. Dodatkowo tak zwany próg wejścia, który jest wymagany do zaimplementowania architektury Azure jest bardzo wysoki, co może stanowić wyzwanie. Dla porównania mocnymi stronami lokalnej architektury z symulowanymi są: niskie koszty; brak kosztów związanych z usługami chmurowymi, co może znaczco obniżyć koszty operacyjne; oraz pełna kontrola nad danymi i procesami przetwarzania, co jest kluczowe dla niektórych zastosowań. System ten może działać bez stałego połączenia z Internetem, co zwiększa jego niezawodność w warunkach lokalnych. Jednakże lokalna architektura również posiada słabe strony. Są nimi ograniczone możliwości skalowania w porównaniu do chmury, co może być problematyczne przy wzroście ilości danych, oraz konieczność lokalnego zarządzania infrastrukturą i usługami, co wymaga dodatkowych zasobów i umiejętności. Dodatkowo brak rzeczywistych danych może ograniczać realizm testów i dokładność symulacji, co może wpływać na jakość wyników.

Podsumowując, architektura wykorzystująca Azure oferuje dużą skalowalność, dostępność i bezpieczeństwo, ale wiąże się z wyższymi kosztami i złożonością. Lokalna architektura z symulowanymi danymi jest kosztowo efektywna i daje większą kontrolę, ale jest ograniczona pod względem skalowalności i wymaga znacznych zasobów do zarządzania. Przedstawione zostały dwie architektury: architektura chmurowa wykorzystującą platformę Microsoft Azure Cloud oraz architektura lokalna z wykorzystaniem Apache Kafka. Do realizacji projektu wybrana została architektura wykorzystująca Apache Kafka, ponieważ bardziej pasowała do założeń biznesowych. Głównym celem pracy było stworzenie uniwersalnej architektury, która mogłaby zostać zaimplementowana do różnych przypadków biznesowych. Cel ten został osiągnięty, a ponadto praca została rozwinięta o implementacje.

## **6. Wnioski**

Oba podejścia mają swoje unikalne zalety i wady, które muszą być rozważone w kontekście specyficznych wymagań i ograniczeń projektu. Architektura wykorzystująca Azure oferuje zaawansowane możliwości skalowania, bezpieczeństwo i integrację, co czyni ją odpowiednią dla dużych i dynamicznych systemów meteorologicznych, gdzie kluczowe jest przetwarzanie danych w czasie rzeczywistym. Z kolei lokalna architektura z symulowanymi danymi jest bardziej ekonomiczna i elastyczna, dając większą kontrolę nad danymi i procesami przetwarzania, co może być korzystne w fazie testowania i prototypowania.

Ostateczny wybór architektury powinien być dokonany na podstawie szczegółowej analizy wymagań projektowych, budżetu, zasobów dostępnych do zarządzania systemem oraz przewidywanego wzrostu skali operacji. Dla zrealizowanego projektu została wybrana architektura Big Data z wykorzystaniem Apache Kafka ze względu na to, że bardziej pasowała do wymagań biznesowych.

## Literatura

- [1] David L. Hall, James Llinas: Handbook of Multisensor Data Fusion, CRC Press, Florida, 2001
- [2] Quinton Anderson: Storm Real-time Processing Cookbook, Packt Publishing, Birmingham, 2013
- [3] P. Taylor Goetz, Brian: Storm Blueprints: Patterns for Distributed Real-time Computation, Packt Publishing, Birmingham, 2014
- [4] James Morle: Scaling Oracle8i(TM): Building Highly Scalable OLTP System Architectures, Addison-Wesley Professional, Maaloev, 2000
- [5] IBM, <https://www.ibm.com/docs/pl/was/9.0.5> (dostęp 16.6.2024)
- [6] Clockworjava, <https://clockworkjava.pl/2021/01/architektura-trojwarstwowa/> (dostęp 16.6.2024)
- [7] Pete Warden: Big Data Glossary, O'Reilly Media, Sebastopol, 2011
- [8] Terence Craig, Mary E. Ludloff: Privacy and Big Data, O'Reilly Media, Sebastopol 2011
- [9] O'Reilly Radar Team: Big Data now, O'Reilly Media, Sebastopol, 2012
- [10] Databricks, <https://www.databricks.com/glossary/medallion-architecture> (dostęp 16.06.2024)
- [11] Dataengineering, <https://dataengineering.wiki/Concepts/Medallion+Architecture> (dostęp 16.06.2024)
- [12] Microsoft, <https://techcommunity.microsoft.com> (dotęp 16.06.2024)
- [13] Future Learn, <https://www.futurelearn.com/info/courses/> (dostęp 16.06.2024)
- [14] Nishant Garg: Apache Kafka, Packt Publishing, Birmingham, 2013
- [15] Kumar, Manish; Singh, Chanchal: Building Data Streaming Applications with Apache Kafka, Packt Publishing Birmingham, 2017
- [16] Saurav Haloi: Apache ZooKeeper Essentials, Packt Publishing, Birmingham, 2015
- [17] Bill Wilder: Cloud Architecture Patterns: Using Microsoft Azure, O'Reilly Media, Sebastopol, 2012
- [18] Henry Li: Introduction to Windows Azure: an introduction to cloud computing using Microsoft Windows Azure, Apress; Distributed by Springer-Verlag, New York, 2009
- [19] Richard J. Dudley, Nathan Duchene: Microsoft Azure: Enterprise Application Development, Packt Publishing, Birmingham, 2010
- [20] Anindita Basak, Krishna Venkataraman, Ryan Murphy, Manpreet Singh: Stream Analytics with Microsoft Azure: Real-time data processing for quick insights using Azure Stream Analytics, Packt Publishing, Birmingham, 2017
- [21] Medium, <https://medium.com/data-processing-architecture-on-microsoft-azure> (dostęp 23.06.2024)
- [22] Mark Wickham: Practical Java Machine Learning: Projects with Google Cloud Platform and Amazon Web Services, Apress, New York, 2018
- [23] David K. Rensin: Building a Windows IT Infrastructure in the Cloud: Distributed Hosted Environments with AWS, O'Reilly Media, Sebastopol, 2012
- [24] Mike Ryan, Federico Lucifredi: AWS System Administration: Best Practices for Sysadmins in the Amazon Cloud, O'Reilly Media, Sebastopol, 2015
- [25] Amazon, <https://docs.aws.amazon.com//data-warehousing-on-aws/data-processing.html> (dostęp 23.06.2024)
- James Murty: Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB, O'Reilly Media, Sebastopol, 2008
- [26] JJ Geewax: Google Cloud Platform in Action, Manning Publications, Nowy Jork, 2018
- [27] James Beswick: Google Apps Express: The Fast Way To Start Working in the Cloud, CreateSpace Independent Publishing Platform, South Carolina, 2011

## Summary

The scope of the thesis was a very detailed analysis on data processing. The main purpose of the thesis was to compare two different architectures one with Apache Kafka the other using Azure for data processing and visualization. The thesis was divided into two parts a practical part which focused on describing the whole issue and a practical part.

The theoretical part, after a review of available reading, was divided into 4 sub-chapters on data processing, the concept of Cold, Warm and Hot path and its use case, Apache Kafka and ZooKeeper and cloud solutions for data processing. The first part introduced the concept of three layers: presentation layer, business layer and application layer. Then the concept of Big Data and its two main architectures Kappa, Lambda and the fairly new concept of medallion architecture were presented. The chapter concluded with data quality and the 3 levels of data: bronze, silver and gold. The next chapter described the differences between historical and real-time data processing for which the cold and warm path is used. At the very end of the theoretical chapter Apache Kafka and Apache ZooKeeper and cloud concepts were antedated.

The practical part was divided into several smaller parts. The first was on the assumptions for building a real-time data processing ecosystem for a weather station. Two architectures were presented the first one was a cloud architecture using the Microsoft Azure Cloud platform, and the second one was an on-premises architecture using Apache Kafka. Then one of them was selected and implemented. The architecture that was chosen was the architecture using Apache Kafka, because it more closely matched the business objectives of the project. This architecture consisted of the following components:

- Data collection - Python scripts
- Data processing - Java applications
- Storage and transfer - Apache Kafka
- Metadata and data management - Apache ZooKeeper
- Orchestration synchronization and flow management - Apache Airflow
- Metadata management - Apache Atlas
- Cluster management - Apache Ambari
- Data visualization and analysis - Power BI
- System management interface - Command Prompt (CMD terminal)

The last step after the implementation was the conclusion and summary where its advantages and disadvantages were compared.

The goal of the work was achieved and in addition, the work was expanded by implementation because the main goal was to create a universal architecture that could be implemented for different business cases.