POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Paweł Celejewski
Nr albumu s15799

Przetwarzanie danych z systemów IoT z wykorzystaniem rozwiązań chmurowych oraz rozproszonych

Praca magisterska
dr inż. Mariusz Trzaska

Warszawa, lipiec, 2022

Streszczenie

Praca dotyczy porównania procesu budowy i utrzymania nowoczesnych systemów informatycznych z dziedziny IoT. Wykonana została analiza charakterystyk i dziedziny problemowej rozwiązań w obszarze IoT. Dokonany został przegląd koncepcji i narzędzi do budowy skalowanych i wysoko dostępnych systemów informatycznych, które przetwarzają dane w czasie rzeczywistym. Przybliżono pojęcia określające usługi chmurowe oraz dokonano analizy i wyróżnienia popularnych rozwiązań. Opisane narzędzia dotyczą usług chmurowych, w szczególności komponentów bez serwerowych, które minimalizują nakłady pracy związane z konfiguracją oraz utrzymaniem systemu. Konkurencyjną koncepcją budowy skalowanych systemów jest wykorzystanie technologii konteneryzacji na przykładzie platformy Docker.

W pracy dokonuje się przeglądu metodologii i wzorców projektowych do wytwarzania oprogramowania, w tym skalowanej architektury systemu opartej na przetwarzaniu zdarzeń. Wyróżniono wykorzystane narzędzia takie jak język programowania, silniki bazodanowe, konsole administracyjne oraz zastosowane metodyki takie jak proces CI/CD.

Podczas tworzenia pracy zostały wykonane prototypy systemów informatycznych w technologii usług chmurowych oraz w technologii skonteneryzowanych rozporoszonych aplikacji. Dokonano opisu wymagań biznesowych systemu IoT oraz zaprojektowano architekturę umożliwiającą realizację stawianych celów. Podjęta została analiza procesu budowy, rozszerzania oraz monitorowania systemów z wyszczególnieniem dostępnych udogodnień i trudności związanych z każdą z tych technologii.

Słowa kluczowe: usługi chmurowe, AWS, IoT, inżynieria oprogramowania, projektowanie systemów, Serverless, Docker, telemetria, skalowalność, monitoring, CI/CD

Spis treści

1. Wstęp

Wszechobecna cyfryzacja procesów biznesowych ewoluuje wraz z pojawianiem się nowych rozwiązań technologicznych. Krytyczność działania systemów informatycznych wymusza projektowanie, rozwój i utrzymanie systemów, które potrafią dostosować się do liczby użytkowników. Ponadto wymagane jest, aby systemy były odporne na awarie, przez co system spełnia stawiane przed nim wymagania biznesowe bez większych zakłóceń. Ostatnie dwie dekady stanowią okres intensywnego rozwoju i popularyzacji konceptów, które znacznie ułatwiają i przyspieszają rozwój wysokoskalowanych systemów informatycznych. Mowa tutaj o technologii usług chmurowych oraz konteneryzacji. Technologie te są ze sobą ściśle powiązane, gdyż konteneryzacja jest pospolicie wykorzystywana w realizacji usług chmurowych. Skorzystanie z usług chmurowych znacznie ułatwia budowę systemów, gdyż wynajęcie zasobów komputerowych jest szybkie oraz wymaga znacznie mniejszych nakładów pracy administracyjnej. W związku z tym zespół programistów może znacznie szybciej skupić się na realizacji wymagań biznesowych.

1.1. Skalowane systemy informatyczne w obszarze IoT

Rozwój technologii Internetu oraz usług chmurowych znacząco zwiększyły możliwości rozwiązań z obszaru IoT (ang. *Internet of Things*). Budowa systemów charakteryzujących się wysoką skalowalnością i odpornością na awarie jest znacząco ułatwiona poprzez wykorzystanie technologii rozwiązań chmurowych, konteneryzacji oraz wirtualizacji zasobów obliczeniowych. Systemy z obszaru IoT mogą charakteryzować się koniecznością zarządzania i komunikacji z bardzo dużą ilością urządzeń. To stawia wyzwania przed projektantami systemów informatycznych, aby spełnić stawiane wymagania biznesowe. Skalowalność systemów oznacza cechę systemu, która uwzględnia długoterminowo rozwój systemu i potencjalnie znacząco zwiększoną liczbę zadań do obsłużenia. Wiąże się to z posiadaniem ładu architektonicznego oraz mechanizmów umożliwiających dynamiczne dostosowanie zasobów obliczeniowych do chwilowego zapotrzebowania. Kolejną pożądaną cechą systemów jest zapewnienie wysokiej dostępności i odporności na awarie systemu. Wskazane cechy są możliwe do zaspokojenia oraz znacząco ułatwione do wdrożenia przy zastosowaniu usług chmurowych z wyszczególnieniem usług bezserwerowych (ang. *Serverless*) lub technologii konteneryzacji z zastosowaniem systemów do zarządzania kontenerami typu Kubernetes lub Docker Swarm.

1.2. Cel pracy

Celem pracy jest zgłębienie wiedzy na temat budowy i utrzymania nowoczesnych systemów informatycznych. Osiągnięcie tego celu możliwe jest poprzez porównanie i przeanalizowanie procesu tworzenia skalowalnego i odpornego na awarie systemu. Szczególnie interesujące jest wykorzystanie usług chmurowych oferowanych przez AWS oraz rozproszonej technologii konteneryzacji zarządzanej na własnej infrastrukturze komputerowej. Ponadto ideą pracy jest lepsze zrozumienie mechanizmów wykorzystanych w budowie usług chmurowych. Ma to na celu bardziej świadome wykorzystywanie tego typu usług. Wiedza ta dostarcza możliwości wyboru pomiędzy dostawcami

usług chmurowych lub budową własnego systemu. Jest to istotne, aby nie być całkowicie uzależnionym od wybranego dostawcy usług chmurowych. W tym celu wykonany został projekt implementacyjny, który obejmuje stworzenie średnio-zaawansowanego systemu do zarządzenia urządzeniami IoT w dwóch wariantach. Pierwszy wariant wykorzystuje usługi chmurowe. Natomiast drugi wariant używa technologię konteneryzacji wdrożoną na własnej infrastrukturze. W wykonanym projekcie podjęta została próba zmapowania komponentów oferowanych przez usługi chmurowe AWS na skonteneryzowane komponenty wykorzystane do samodzielnego zbudowania systemu. Obydwa systemy realizują zbliżone funkcjonalności biznesowe.

Kolejnym celem pracy jest zgłębienie technik architektonicznych i wzorców projektowych wykorzystanych w budowie usług chmurowych, co pozwoli czytelnikowi uzyskać pogląd na temat budowy systemów chmurowych. Celem analizy procesu budowy i utrzymania podobnych systemów w różnych technologiach jest uzyskanie wiedzy na temat udogodnień i ograniczeń związanych z budową systemów w chmurze w kontraście do systemów na własnych serwerach.

1.3. Rozwiązanie przyjęte w pracy

Rozwiązania przyjęte w pracy skupiają się wokół technologii i usług oferowanych przez ekosystem chmurowy AWS oraz technologie konteneryzacji z wykorzystaniem narzędzia Docker. Nacisk został położony na wykorzystanie nowoczesnych rozwiązań takich jak usługi typu *Serverless*. Założenia teoretyczne pracy obejmują wykorzystanie wzorców projektowych, które zapewniają cechy systemów takie jak skalowalność, wysoka dostępność i odporność na awarie. Rozwiązania technologiczne spełniające te wymagania odnoszą się do technologii i strategii autoskalowania komponentów, zarządzenie infrastrukturą poprzez wykorzystanie metodologii definiowania infrastruktury jako kodu (ang. *Infrastructure as a Code - IaaC*), replikacji komponentów oraz wdrożenia systemu w odizolowanych geograficznie regionach. W projekcie została wdrożona metodologia ciągłego dostarczania oprogramowania CI/CD (ang. *Continuous integration Continuous Delivery*) poprzez zdefiniowanie procesu do wdrażania nowych wersji oprogramowania. Na tą technologię składa się system wersjonowania Git. Do utworzenia procesu CI/CD wykorzystane zostały usługi takie jak AWS CodeCommit, AWS CodeBuild oraz AWS CodePipeline. Usługi te wykorzystują dostarczone skrypty na platformę Unix w powłoce bash odpowiedzialne za spakowanie kodu do wymaganego formatu. Wykorzystanie technologii konteneryzacji stanowi użycie narzędzi Docker, Docker Hub, Docker Compose. Logika biznesowa systemu została zaimplementowana z wykorzystaniem języka Python w wersji 3.8 . Do zdefiniowania interfejsu REST wykorzystano standard OpenAPI poprzez użycie pliku swagger. Systemy umożliwiają monitoring aplikacji poprzez gromadzenie i wizualizację logów aplikacyjnych. Wykorzystano technologie nierelacyjnych bazy danych: AWS DynamoDB, MongoDB, RedisDB oraz baz danych typu *time-series*: AWS Timestream, InfluxDB. Przy implementacji kodu źródłowego wykorzystano IDE PyCharm. Do wykonania diagramów architektonicznych skorzystano z usług serwisu drawio.

1.4. Rezultaty pracy

W rezultacie wykonanych prac nad niniejszą pracą magisterską, autor dokonał analizy i refleksji nad możliwościami technologicznymi ekosystemów usług chmurowych oraz technologii konteneryzacji. Pozwoliło to zbudować wiedzę, zdefiniować wnioski oraz pogłębić świadomość wykorzystania tych narzędzi. Zrozumienie tych mechanizmów poszerza gamę możliwości wyboru rozwiązań, co z kolei pomaga uniknąć sytuacji uzależnienia od jednej technologii lub dostawcy (ang. *vendor lock*). W procesie zgłębiania wiedzy powstały dwa prototypy systemów informatycznych realizujących zbliżoną funkcjonalność biznesową. Systemy te zostały zbudowane wykorzystując odpowiednio technologie usług chmurowych AWS oraz konteneryzacji z użyciem platformy Docker. Proces budowy dostarczył licznych analiz i wniosków na temat wykorzystanych podejść, uwypuklając napotkane ograniczenia, możliwości oraz trudności. Tekst pracy podejmuje próbę opisania, analizy i wyciągnięcia wniosków z doświadczeń zebranych podczas analizy rozwiązań i tworzenia prototypów.

Nadrzędne wnioski zdefiniowane podczas pracy obejmują:

- rozwój technologii chmurowej i konteneryzacji znacząco usprawnił i przyspieszył budowę systemów informatycznych;
- systemy z domeny IoT zyskują na znaczeniu na rynku poprzez oferowane możliwości, jednakże rozwiązania te borykają się z pewnym sceptycyzmem ze strony klientów ze względu aspekty bezpieczeństwa lub problemowego działania urządzeń końcowych;
- usługi chmurowe powszechnie wykorzystują konteneryzację i wirtualizację przy tworzeniu swoich usług, więc omawiane technologie są ze sobą blisko powiązane;
- wykorzystanie usług chmurowych w znacznym stopniu odciąża z wykonywania czynności, które są niezbędne przy budowie systemu na własnych zasobach komputerowych takich jak zarządzenie infrastrukturą i siecią;
- stosunkowo łatwo można zbudować wysokodostępny, skalowany i odporny na awarie system poprzez wykorzystanie usług chmurowych poprzez rozmieszczenie systemów w różnych regionach z wykorzystaniem replikacji i strategii autoskalowania opartych na zdefiniowanych zdarzeniach. Osiągnięcie tego samego efektu z wykorzystaniem technologii konteneryzacji jest możliwe, lecz wymaga znacznie większych nakładów prac związanych z konfiguracją infrastruktury komputerowej oraz sieciowej;
- technologie chmurowe dynamicznie się rozwijają, co wymaga śledzenia zmian i trendów, aby pozostać na bieżąco;
- skorzystanie z usług chmurowych pozwala obniżyć koszty i szybciej wdrożyć system od pomysłu do rynku;
- przy strategicznym planowaniu systemu należy wziąć pod uwagę aspekt uzależnienia się od danej technologii lub dostawcy usług. Należy dążyć do minimalizacji tego ryzyka poprzez dywersyfikację możliwych rozwiązań;
- usługi chmurowe oferują ujednolicony sposób korzystania z dostępnych narzędzi takich jak kokpity administracyjne, wizualizacje czy metody autentykacji. Budowa systemu *on-premise* wykorzystujące różne technologie powoduje rozbieżności w sposobie

administracji danych komponentów oraz zwiększa nakłady pracy potrzebne do zintegrowania usług;

- prototypowanie oraz zadania praktyczne to bardzo efektywna metoda nauki nowych technologii.

1.5. Organizacja tekstu pracy

Tekst pracy magisterskiej zorganizowany jest w sposób wprowadzający czytelnika w problematykę podjętego tematu. Wyszczególniona jest specyfika systemów z obszaru IoT. Następnie dokonana jest analiza i możliwości systemów w technologii chmurowej. Tekst pracy opisuje również koncepcję technologii konteneryzacji. Następnie dokonany jest opis i analiza wykonanych prototypów systemów informatycznych z zagłębieniem się w szczegóły wykorzystanych usług chmurowych AWS oraz Docker. Praca jest podsumowana opisanymi wnioskami nabytymi przy realizacji pracy. Ponadto tekst pracy zawiera spis wykorzystanej literatury i innych źródeł wiedzy.

2. Tworzenie systemów informatycznych w obszarze IoT

Cele biznesowe stawiane przed systemami z obszaru IoT dotyczą zarządzania, komunikacji i integracji urządzeń, które razem składają się na współpracujący system. Przy projektowaniu tego typu rozwiązań należy zwrócić szczególną uwagę na aspekt skalowalności systemu, aby było możliwe podłączenie większej liczby urządzeń końcowych. Usługi oferowane przez dostawców rozwiązań chmurowych lub platformy do zarządzania usługami skonteneryzowanymi są w stanie spełnić te wymagania poprzez dostarczenie odpowiednych rozwiązań technologicznych.

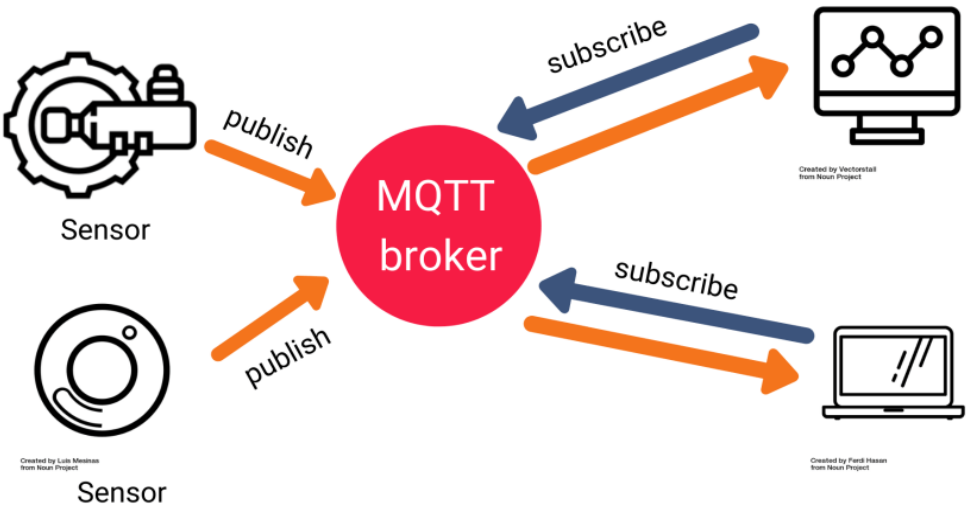
2.1. Problematyka systemów z obszaru IoT

Systemy IoT (ang. *Internet of Things*) charakteryzują się integracją urządzeń różnego typu i zastosowania, które współpracują ze sobą poprzez zdalną wymianę komunikacji za pośrednictwem sieci. Urządzenia pełniące konkretne funkcje mają możliwość komunikacji z centralnym komponentem, który integruje funkcjonalne urządzenia w spójny system. Taki ekosystem tworzy całą gamę możliwości zastosowań. Osiągnięcie takiego rozwiązania byłoby utrudnione lub nieoptymalne, gdyby urządzenia pracowały oddzielnie. Rozwiązania IoT zyskały na popularności oraz notują dynamiczny rozwój w związku z miniaturyzacją mikrokontrolerów oraz powszechnym dostępem do szerokopasmowej sieci Internetu. W rozwoju dziedziny przyczyniły się projekty takie jak zminiaturyzowane komputery (Raspberry PI, Arduino) lub szybki internet w technologii LTE, 5G. W polskiej literaturze znajdziemy zwrot IoT tłumaczony jako internet rzeczy. Dziedzina zastosowań internetu rzeczy jest szeroka i obejmuje kilka gałęzi, z których można wyróżnić inteligentne domy, samochody, gadżety, systemy czujników, asystenci głosowi czy też zarządzenie flotami urządzeń elektronicznych.

Jedną z motywacji do powstawania systemów IoT jest zbieranie i analizowanie danych z urządzeń. Producenci urządzeń elektronicznych chętnie umieszczają moduły sieciowe w swoich produktach w celu umożliwienia komunikacji za pośrednictwem Internetu. Przetworzone dane mogą być podstawą do diagnostyki funkcjonowania urządzenia lub procesu biznesowego, czego przykładem są zastosowania farm czujników monitorujących procesy wytwórcze w przemyśle – w przypadku wykrycia alarmujących wartości odczytach z czujników, operatorzy mają możliwość zareagować i uniknąć awarii. W dłuższej perspektywie czasu, zebrane i przetworzone dane stanowią bardzo ważny element przy podejmowaniu strategicznych decyzji biznesowych, które wytyczają kierunek rozwoju biznesu. Strategia ta nazwana jest biznesem zorientowanym na dane (ang. *Data Driven Business)*. Strategia ta zyskuje bardzo na znaczeniu w wyniku rozwoju dziedziny uczenia maszynowego i algorytmów sztucznej inteligencji. Stąd motywacja przedsiębiorstw do gromadzenia danych na temat swojej działalności jest bardzo duża. Z kolei cecha ta naturalnie wpisuje się w możliwości oferowane przez systemy IoT. W wyniku analizy danych w procesie głębokiego uczenia maszynowego można lepiej poznać trendy użytkowania, wskazać funkcjonalności nieużywane lub te które sprawiają najwięcej awarii. Uzyskana wiedza pozwala przedsiębiorstwom udoskonalać model biznesowy i w konsekwencji zaoferować nowe funkcjonalności w podwyższonej jakości. Potencjalnie skutkuje to zwiększonymi zyskami z prowadzenia działalności.

Urządzenia włączone w sieć internetu rzeczy mają możliwość przesyłania danych odczytanych z urządzenia w czasie prawie rzeczywistym. W związku z rozwojem infrastruktury sieci internetu, prędkości przesyłania danych zwiększają się, czego przykładem jest rozwój technologii 5G. Kreuje to możliwości budowania coraz bardziej złożonych i krytycznych systemów IoT, które wymagają szybkiego czasu reagowania. Trend ten nazywany jest procesowaniem na urządzaniach końcowych (ang. *Edge Computing*) Przykładem zastosowania jest autonomiczny transport drogowy. Obecnie systemy tego typu borykają się z ograniczeniami technologicznymi takimi jak utrata odpowiedniej przepustowości lub zasięgu sieci w trudnym terenie takim jak tunele lub obszary niezurbanizowane.

Zdalnie przesyłane dane pomiarowe z urządzeń nazywane są danymi telemetrycznymi. Urządzenia podłączone do sieci internetu rzeczy przesyłają dane do centralnego systemu, który gromadzi i przetwarza zebrane dane. Urządzenia terenowe najczęściej dysponują stosunkowo niewielką mocą obliczeniową. W związku z tym do przesyłania danych telemetrycznych stosuje się lekkie protokoły takie jak protokół MQTT (ang. *Message Queue Telemetry Transport*). Ma to na celu zapewnić wysoką przepustowość danych w czasie prawie rzeczywistym [14]. Protokół MQTT bazuje na wzorcu projektowym komunikacji o nazwie publikacja – subskrypcja. Za zarządzenie komunikacją odpowiedzialna jest usługa brokera połączeń. Broker zarządza połączeniami oraz tworzy dedykowane kanały komunikacyjne nazwane tematami (ang. *topic*). Ponadto pośredniczy w komunikacji pomiędzy nadawcą i odbiorcą komunikacji jak pokazano na Rysunku 1. Urządzenia terenowe wysyłają komunikaty na zdefiniowany temat, do którego zasubskrybowany jest odbiorca w postaci programu przetwarzającego dane w centrum obliczeniowym systemu IoT. Protokół MQTT charakteryzuje się prostotą i łatwością zastosowania. W przypadku potrzeby zastosowania bardziej rozbudowanych mechanizmów komunikacji lub bezpieczeństwa, alternatywą jest protokół AMQP (ang. *Advanced Message Queuing Protocol*). Porównania protokołów MQTT oraz AMQP dokonano w pracy [6].



Rysunek 1. Zasada działania brokera protokołu MQTT. Źródło: [14]

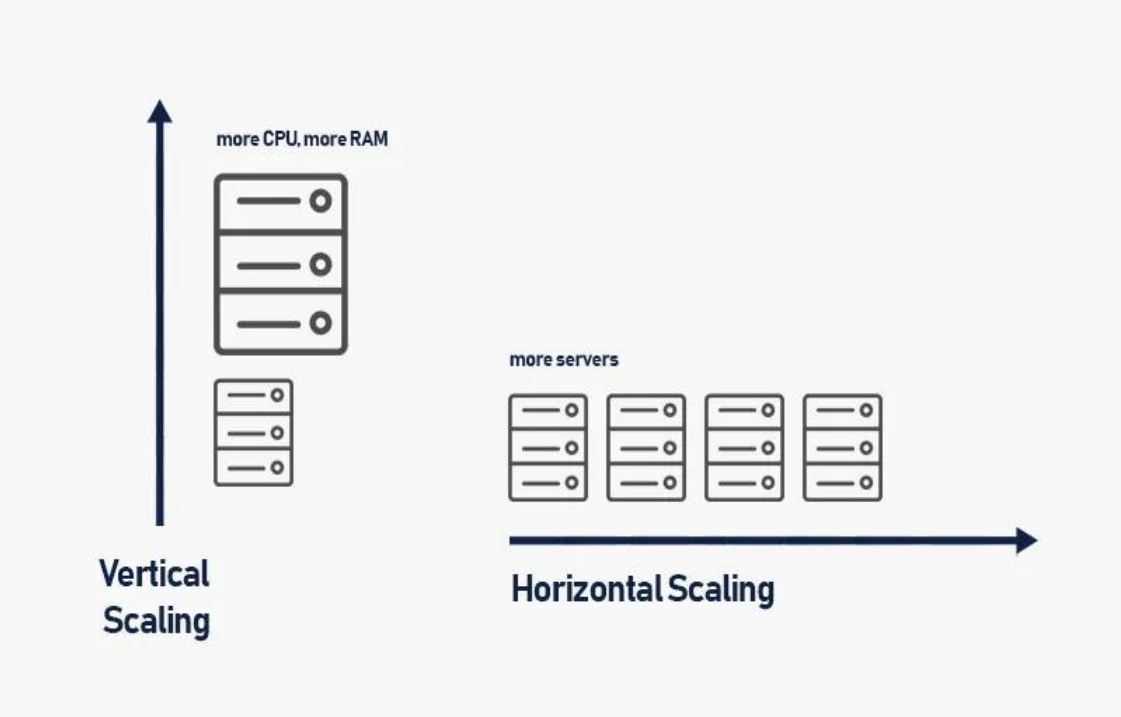
Odbierane dane z urządzeń terenowych są przetwarzane w centralnym komponencie obliczeniowym. Komponent ten pełni funkcję huba i stanowi cel połączenia dla urządzeń terenowych.

Centralny komponent obliczeniowy posiada znacznie większe możliwości obliczeniowe w stosunku do urządzeń, które się do niego podłączone. Ponadto można wyróżnić hierarchię hubów, gdyż systemy IoT mogą posiadać wielopoziomowe komponenty obliczeniowe. Przykładem może być system inteligentnego domu, który wykorzystuje centralny komponent niskiego poziomu do zarządzania urządzeniami znajdującymi się w domu takie jak oświetlenie, ogrzewanie, sprzęt AGD. Komponent obliczeniowy niskiego poziomu jest natomiast skomunikowany z komponentem obliczeniowym wyższego poziomu, który zarządza wieloma komponentami niższego poziomu. W tym przypadku komponentem obliczeniowym wyższego poziomu może być centralny system oferujących zdefiniowane usługi. Komponenty obliczeniowe poprzez odbieranie i przetwarzanie danych mają możliwość wizualizacji danych w postaci wykresów i tabel, co umożliwia monitoring, wykrywanie anomalii, kontrolę jakości oraz podejmowanie reakcji w razie przekroczenia dopuszczalnych norm wartości odczytanych z urządzeń.

Systemy IoT mogą charakteryzować się bardzo dużą ilością podłączonych urządzeń, które z kolei są w stanie generować znaczne ilości danych w zadanym interwale czasu [15]. W związku z tym systemy IoT powinny cechować się wysoką skalowalnością. Skalowalność systemów informatycznych definiuje się jako możliwość obsłużenia zwiększonej liczby użytkowników czy też zapytań do usług systemu wraz z perspektywą rozwoju i rozrostu systemu [11].

W kontekście skalowalności systemów wyróżnia się następujące strategie, które uwidocznione są również na Rysunku 2:

- Skalowanie horyzontalne - definiowane jest jako zwiększenie ilości podstawowych jednostek procesowania zdefiniowanego zadania. Może to zostać zrealizowane poprzez zwiększenie ilości działających kontenerów lub procesów. Podstawą do zwiększenia ilości jednostek procesowania może być ilość danych oczekujących na procesowanie w kolejce;
- Skalowanie wertykalne - definiuje się jako zwiększenie parametrów mocy obliczeniowej pojedynczej jednostki procesującej. Realizuje się to poprzez zwiększenie ilości pamięci lub zastosowania procesora o większych osiągach. W przypadku systemów chmurowych przykładem zastosowania jest zmiana typu maszyny wirtualnej na taką o większych parametrach. W przypadku systemu *on-premise* należy wymienić fizyczny zasób serwera na potężniejszą maszynę.



Rysunek 2. Porównanie strategii skalowania wertykalnego i horyzontalnego. Źródło: [11]

Systemy IoT powinny również charakteryzować się elastycznością, co związane jest z odpowiednim dostosowaniem mocy obliczeniowej systemu do chwilowej ilości danych do przetworzenia. Realizuje się to przez odpowiednie zdefiniowanie reguł do automatycznego skalowania systemu, które na podstawie ilości danych oczekujących w kolejce do procesowania, odpowiednio dostosowują ilość mocy obliczeniowej. W związku z charakterystyką komunikatów z urządzeń IoT, które są ustrukturyzowane i stosunkowo małych rozmiarów, dobrze sprawdza się zastosowanie strategii skalowania horyzontalnego. Naturalnie wpisuje się w to wykorzystanie techniki konteneryzacji do budowy aplikacji przetwarzającej dane. Wykorzystanie zarządcy kontenerów lub bezserwerowych usług chmurowych wraz z odpowiednią konfiguracją automatycznego skalowania zapewnia osiągnięcie skalowalności i elastyczności systemów.

Rozwiązania z domeny IoT mimo swoich licznych zalet i możliwości borykają się z wieloma istotnymi problemami. Wpływa to na odbiór i zaufanie klientów do tego typu technologii. Jak zaznacza autor w publikacji [2] mimo znaczącej popularyzacji internetu rzeczy, w dalszym ciągu większość odbiorców nie ma świadomości na temat prawidłowości w funkcjonowaniu tego typu rozwiązań. Eksplozja popularności IoT spowodowała przerodzenie się tej domeny z niszowego obszaru do szybko rosnącego trendu w stosunkowo krótkim czasie. Efektem tego zjawiska jest fakt, że bardzo dużo podmiotów niezależnie od siebie rozwija produkty i usługi pod szyldem IoT. Objawia się to brakiem standaryzacji i wysokim zróżnicowaniem jakości usług producentów. Bardzo istotnym problemem rozwiązań IoT jest zapewnienie bezpieczeństwa. W przypadku włamania do systemu, atakujący może przejąć sterowanie i dane wielu urządzeń [2, 15]. Kolejnym aspektem, który nasila brak zaufania do IoT są ciągłe problemy z prawidłowym funkcjonowaniem urządzeń końcowych [2]. Wynika to z szybkiej dynamiki powstawania produktów i oprogramowania, wydawania nowych wersji oraz problemy z kompatybilnością dla bardzo wielu urządzeń końcowych. Zgłaszane zastrzeżenia względem systemów IoT dotyczą także niechęci użytkowników do gromadzenia i przetwarzania danych. Aspekt zbierania danych jest regulowany prawnie w niektórych obszarach

geograficznych na przykład w postaci ogólnego rozporządzenia o ochronie danych osobowych (RODO) w Unii Europejskiej. Jednakże aspekt etyczny i realne wykorzystanie danych zgodnie z prawem pozostaje w gestii dostawy usług co wpływa na zaufanie do tego typu usług.

2.2. Charakterystyka systemów i usług chmurowych na przykładzie AWS

Usługi chmurowe są stosunkowo nowym rozwiązaniem w dziedzinie informatyki. Ich intensywny rozwój przypada na ostatnie dwie dekady. Mimo to technologia ta zdążyła już zrewolucjonizować środowisko IT i obecnie upatrywana jest jako jedna z najbardziej perspektywicznych obszarów z branży informatycznej [1]. Skorzystanie z usług chmurowych można rozumieć poprzez zdalne wynajęcie mocy obliczeniowej do określonych celów za pośrednictwem dostarczonych interfejsów takich jak przeglądarka internetowa lub komendy wiersza poleceń. Przykładami użycia wynajętej mocy obliczeniowej może być uruchomienie bazy danych, serwera udostępniającego stronę internetową, przechowywanie plików takich jak zdjęcia czy filmy, korzystanie z gotowych usług oferowanych z chmurowej mocy obliczeniowej takich jak pakiet MS Office 365 czy usługi konta Google. Głównymi motywacjami do wyboru usług chmurowych jest oszczędność czasu przy powstawaniu i wdrażania nowych systemów, ograniczenie kosztów związanych z nieużywaną infrastrukturą komputerową, większe możliwości obliczeniowe, skalowalność, wysoka dostępność usług, ochrona systemów przed katastrofami, bezpieczeństwo, wygodne interfejsy do zarządzania zasobami. Te czynniki powodują, że coraz więcej przedsiębiorstw buduje swoje nowe systemy w chmurze oraz migruje obecne rozwiązania *on-premise* do rozwiązań z wykorzystaniem usług chmurowych [16, 17].

2.2.1 Modele usług chmurowych

W modelu oferowanych usług chmurowych dokonywany jest następujący podział:

- IaaS - Infrastructure as a Service;
- PaaS - Platform as a Service;
- SaaS – Software as a Service.

Model ten dokonuje podziału ze względu na poziom abstrakcji i sposób korzystania z usług. Model ten został zaprezentowany przez organizację NIST (ang. *National Institute of Standards and Technology*) [12].

Najniższym poziomem abstrakcji charakteryzują się usługi IaaS. Usługi tego typu są rozumiane jako wynajęcie fizycznej infrastruktury komputerowej umieszczonej w centrum obliczeniowym dostawy usług chmurowych. Na usługi tego typu składa się wynajęcie maszyn wirtualnych lub dysków twardych. Tego typu usługi dostarczają największą swobodę w konfiguracji zasobów, jednakże wymagają także samodzielnego skonfigurowania infrastruktury i zapewnienia bezpieczeństwa do stawianych celów biznesowych. Obowiązki wynikające ze skorzystania z IaaS dotyczą akcji takich jak wybór systemu operacyjnego, doinstalowanie wymaganych aplikacji, skonfigurowanie zabezpieczeń sieciowych.

Usługi sklasyfikowane jako PaaS oferują skonfigurowaną infrastrukturę przez dostawcę usług chmurowych, na której zainstalowane są komponenty oprogramowania umożliwiające

skonfigurowanie oferowanej usługi do określonych celów. Na tego typu usługi składają się platformy baz danych, platformy do wdrażania i serwowania usług stron internetowych czy też platformy do zarządzania zasobami i klientami. Zastosowanie tych usług odciąża użytkownika z wysiłku zarządzania infrastrukturą, przez co może skupić się bezpośrednio na realizacji celów biznesowych. Jednakże zainstalowanie oprogramowanie, mechanizmy bezpieczeństwa oraz ich aktualizacje pozostają wyłącznie w jurysdykcji dostawcy usług chmurowych, więc swoboda administracyjna jest ograniczona w porównaniu do usług IaaS.

W przypadku usług typu SaaS mamy do czynienia z gotowym produktem w postaci oprogramowania, które po nabyciu odpowiednich praw, jest możliwe do użytkowania. W tym rodzaju usług użytkownik napotyka najmniejsze narzuty konfiguracyjne, stąd poziom abstrakcji względem infrastruktury jest najwyższy. Cała odpowiedzialność za prawidłowe działanie usługi spoczywa na dostawy usług. Przykładami usług SaaS są usługi pocztowe, serwery plików, usługi do zarządzania zdjęciami itp.

2.2.2 Modele wdrożenia systemów chmurowych

Kolejnym aspektem charakteryzacji systemów chmurowych jest zaproponowany przez NIST model wdrożenia [12]. Wyróżnia się następujące modele wdrożenia:

- chmura publiczna;
- chmura prywatna;
- chmura hybrydowa;
- chmura społeczności.

System wdrożony z wykorzystaniem usług chmury publicznej charakteryzuje się tym, że całość systemu funkcjonuje na infrastrukturze i usługach dostawy usług chmurowych. Domyślnie wiąże się to z korzystaniem z powszechnie dostępnych usług, ze współdzieleniem infrastruktury komputerowej z innymi klientami dostawy usług chmurowych (ang. *resource multitenancy*) oraz przechowywaniem danych na zasobach w centrum obliczeniowym dostawy usług chmurowych.

W przypadku modelu wdrożenia systemu z wykorzystaniem chmury prywatnej mamy do czynienia z usługami typowymi dla chmury publicznej. Jednakże usługi te oferowane są dla ograniczonej liczby użytkowników, najczęściej w obrębie danego przedsiębiorstwa. Instytucja ta zainwestowała w stworzenie usług chmurowych wyłączenie dla swoich pracowników lub określonej grupy docelowej w celu realizacji określonych celów biznesowych. Wiąże się to z pełnymi mocami administracyjnymi nad wykorzystaną infrastrukturą oraz przechowywaniem danych.

Systemy wdrożone w modelu chmury hybrydowej korzystają z usług zarówno chmury publicznej oraz prywatnej. Wykorzystanie tego modelu często ma zastosowanie podczas procesu migracji do chmury publicznej lub w sytuacji gdy istnieje wymaganie przechowywania danych w konkretnej, niepublicznej lokalizacji ze względu prawnych.

Systemy w chmurze społeczności posiadają podobną charakterystykę jak chmury prywatne, jednakże są wykorzystywane przez kilka przedsiębiorstw lub organizacji. Podmioty te współpracują ze sobą w ramach wspólnego celu biznesowego.

2.2.3 Ogólna charakterystyka usług chmurowych

Usługi oraz systemy chmurowe posiadają kilka charakterystyk, które wyróżniają je na tle systemów z wykorzystaniem zastosowań sprzed epoki obliczeń w chmurze, takich jak systemy wdrożone w serwerowniach przedsiębiorstw (ang. *on-premise*).

Usługi chmurowe charakteryzują się dostępnością w wielu regionach, które są geograficznie odseparowane np. region Europa Zachodnia, region Azja-Pacyfik [1]. Z kolei każdy dostępny region podzielony jest na kilka stref dostępności (ang. *Availablity Zones*). W każdym regionie znajdują się minimalnie 3 strefy dostępności. Pozwala to na osiągnięcie prawdopodobieństwa bliskiemu zeru, zgodnie z rozkładem normalnym, że usługi nie będą dostępne w całym regionie [18]. Aby to się wydarzyło, wszystkie strefy dostępności musiały by przestać funkcjonować. Każda strefa dostępności to znacznych rozmiarów centrum obliczeniowe, składające się z hal wypełnionych infrastrukturą komputerową, połączoną szybką wewnętrzną siecią. Centra obliczeniowe zapewniają konserwację, chłodzenie, stałe oraz zapasowe źródło zasilania, fizyczne bezpieczeństwo i ochronę zasobów obliczeniowych na przykład przed fizyczną kradzieżą. Zasoby ze wszystkich regionów składają się na wspólny system usług oferowanych przez dostawcę usług chmurowych, do których klient ma łatwy dostęp za pomocą scentralizowanych kanałów dostępu takich jak przeglądarka internetowa czy interfejs wiersza poleceń. Rysunek 3 obrazuje rozmieszczenie centrów obliczeniowych firmy AWS na globie.

Rysunek 3. Wykaz dostępnych i planowanych regionów AWS. Źródło: [19]

Zastosowanie rozmieszczenia zasobów obliczeniowych w regionach oraz strefach dostępności pozwala osiągnąć niezwykle pożądane właściwości systemów takich jak:

- skalowalność;
- wysoka dostępność;
- wysoka niezawodność danych;
- odporność na katastrofy.

Skalowalność oznacza zdolność systemu do obsłużenia większej ilości zapytań na przykład wraz ze wzrostem liczby użytkowników lub zwiększeniem funkcjonalności systemu [11]. Dostawy usług chmurowych posiadają bardzo duże ilości zasobów obliczeniowych do wynajęcia, więc z perspektywy przedsiębiorstwa istnieje możliwość praktycznie nieograniczonego zwiększania zasobów. Przykładami usług do skalowania są:

- maszyny wirtualne;
- skonteneryzowane instancje aplikacji;
- funkcje (FaaS);
- bazy danych.

Dokonanie skalowania następuje przy obserwacji zwiększonego ruchu w systemie. Ponadto rozwiązania chmurowe dostarczają usługi oraz strategie do autoskalowania zasobów oraz balansowania ruchem sieciowym. Przykładem wykorzystania strategii autoskalowania może być system przetwarzający dane z kolejki, który zawsze ma uruchomiony jeden kontener do procesowania zapytań. W przypadku wykrytej zwiększonej liczby wiadomości oczekujących w kolejce, system może uruchomiać kolejne instancje kontenerów, aby szybciej obsłużyć zwiększony ruch. Gdy ilość zapytań do systemu się unormuje, przez co zwiększona liczba wiadomości w kolejce nie będzie się odkładać, to system wyłączy dodatkowe instancje kontenerów. Dobór odpowiedniej strategii wymaga analizy trendów zachowań użytkowników oraz metryk procesowania systemu. Wykorzystanie skalowalności znacząco poprawia wydajność systemu.

Rodzaje strategii autoskalowania:

- na podstawie metryk i zdarzeń;
- konkretnie wyznaczone pory dnia lub miesiąca (ang. *scheduled scaling*);
- w oparciu o najkorzystniejszą cenę wynajęcia zasobów (AWS EC2 Spot Instances).

Wysoka dostępność (ang. *high availability*) oznacza cechę, dzięki której użytkownik ma zapewnienie od dostawy usług, że usługa będzie dostępna nie mniej niż określona ilość czasu w danym interwale czasu np. usługa będzie dostępna przez 99,99% czasu w miesiącu. Kontrakt pomiędzy dostawcą usług a klientem jest zawarty w dokumencie zwanym *Service Level Agreement.* W przypadku niewywiązania się z zapisów umowy zawartych w SLA, dostawa usług oferuje zniżkę lub w ogóle nie pobiera opłat za usługę w danym okresie rozliczeniowym w zależności od tego jak bardzo zapisy SLA zostały naruszone co ukazuje Tabela 1 dla przykładowej usługi AWS RDS. Na czas niedostępności usług mogą składać się niespodziewane awarie infrastruktury czy też

zaplanowane prace serwisowe. Wysoka dostępność jest możliwa do uzyskania poprzez replikację usług i danych w różnych, niezależnych od siebie strefach dostępności.

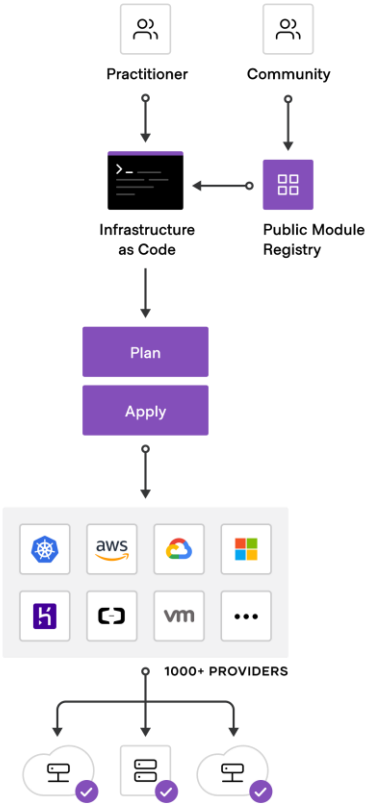
Tabela 1. Wartości zniżek dla usługi AWS RDS przy naruszeniu SLA. Źródło: [20]

Monthly Uptime Percentage	**Service Credit Percentage**
Less than 99.95% but equal to or greater than 99.0%	10%
Less than 99.0% but equal to or greater than 95.0%	25%
Less than 95.0%	100%

Powiązaną cechą do wysokiej dostępności jest wysoka niezawodność danych (ang. *high data durability*). Cecha ta określa, że dane nie zostaną utracone i będą bezpiecznie przechowane z prawdopodobieństwem zawartym w SLA. Dla przykładu usługa AWS S3 (*Simple Storage Service*) w trybie Standard oferuje niezawodność danych na poziomie 99,999999999 %. Związane jest to z replikacją danych w co najmniej 3 strefach dostępności dla każdego przechowywanego obiektu danych.

Cecha definiowana jako odporność na katastrofy (ang. *disaster recovery*) wiąże się zapobieganiem utraty funkcjonowania systemu w przypadku katastrof naturalnych (trąby powietrzne, powódź, pożary) lub antropogenicznych (wojna, terroryzm). Jest to cecha niezwykle pożądana dla krytycznych systemów, takie jak systemy obronności państwa lub systemy bankowe [18]. Dostawy usług chmurowych umożliwiają budowanie systemów odpornych na katastrofy poprzez replikację systemu w różnych regionach, które są odizolowane geograficznie i mało prawdopodobne jest, aby dwa niezależne regiony przestały funkcjonować równocześnie.

Replikację systemów w odosobnione regiony bardzo ułatwia metodologia infrastruktury jako kodu (ang. *Infrastructure as a Code*). Metodologia ta zakłada, że wszystkie komponenty i usługi, które składają się na system, są zdefiniowane w określonym formacie (JSON, YAML) w postaci zdefiniowanych komend składających się na wzorzec lub manifest systemu. Pozwala to na automatyzację replikacji danej platformy na inne środowisko przy minimalnym narzucie konfiguracyjnym. Przykładami usług oferujących tę metodologię są AWS CloudFormation, MS Azure Deployment Templates czy też Terraform. W przypadku narzędzia Terraform możliwe jest zdefiniowanie infrastruktury jako kod dla wielu dostawców usług chmurowych [21] co ukazuje Rysunek 4.



Rysunek 4. Schemat wdrożenia systemu z wykorzystaniem IaaC oraz Terraform. Źródło: [21]

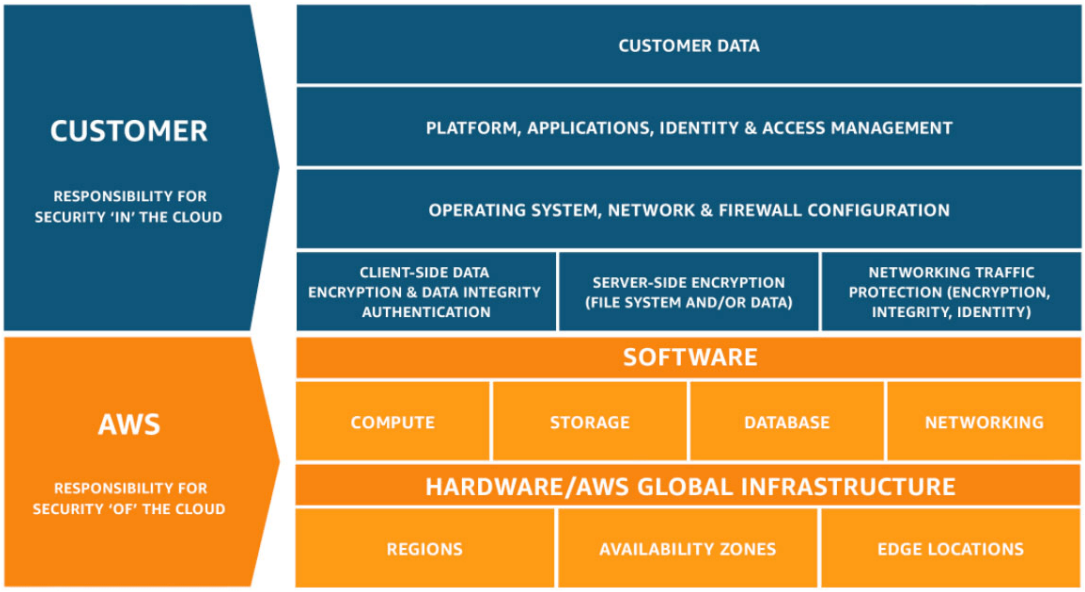
Dla części usług chmurowych zdefiniowany jest model rozliczeniowy określany jako „Płać tylko gdy korzystasz” (ang. *Pay as you go*) [8, 22]. W tym modelu klient płaci za usługę tylko i wyłącznie kiedy wykorzystuje moc obliczeniową po cenach ustalonych w taryfie usługi. Rozliczenie może być dokonywane na podstawie czasu korzystania z usługi (milisekundowe dla AWS Lambda, godzinne dla AWS EC2) lub liczby obsłużonych zapytań (AWS DynamoDB, AWS SQS). Model ten jest bardzo korzystny dla klientów usług chmurowych w porównaniu z wykorzystaniem własnej infrastruktury, za którą najpierw trzeba było zapłacić, aby uruchomić usługę, a następnie przez większość czasu nie jest wykorzystywana cała dostępna moc obliczeniowa. Rozwiązanie problemu niewykorzystywania dostępnej mocy obliczeniowej znacząco przyczyniło się na rozwoju technologii konteneryzacji oraz usług chmurowych. Główne nurty modeli rozliczeniowych AWS [22]:

- *Pay as You Go*;
- *Pay Less by Using More*;
- *Save When You Reserve*;
- *Free Usage Tier*.

Zaoferowanie korzystanego dla klienta modelu rozliczeń „płać tylko gdy korzystasz” jest możliwe poprzez optymalizację wykorzystania zasobów obliczeniowych. Wiąże się to z kolejną standardową cechą usług chmurowych, czyli wspólnego wykorzystywania infrastruktury dostawcy usług chmurowych przez wielu niezależnych od siebie klientów (ang. *multi-tenancy model*). Cecha ta oznacza, że zasoby obliczeniowe w obrębie pojedynczej szafy serwerowej w centrum obliczeniowym, są wykorzystywane przez wielu klientów. Jest to domyślny model funkcjonowania większości usług chmurowych. W przypadkach systemów wymagających dedykowanych zasobów komputerowych z dedykowaną siecią to istnieje możliwość takiego wynajmu, lecz wiąże się to ze zwiększonymi

kosztami. W niektórych sytuacjach, związanych z aspektami prawnymi, model współdzielonego najmu nie jest możliwy na przykład ze względu na zapisy licencji na oprogramowanie.

W kontekście bezpieczeństwa usług chmurowych funkcjonuje model określany jako współdzielona odpowiedzialność (ang. *shared responsiblity model*). W modelu tym wyszczególniona część odpowiedzialności za zachowanie bezpieczeństwo jest obowiązkiem dostawy usług chmurowych, a część jest obowiązkiem klienta [23] co unaocznione jest na Rysunku 5. Podział odpowiedzialności różni się pomiędzy usługami, gdyż ten model przyjmuje inne charakterystyki pomiędzy typami usług IaaS, PaaS oraz SaaS. W ogólności to na dostawy usług chmurowych spoczywają takie odpowiedzialności jak fizyczna ochrona zasobów komputerowych, szyfrowanie danych transportowanych siecią wewnętrzną, zapewnianie wymaganych aktualizacji na systemach operacyjnych, rotacja i zabezpieczenie kluczy do szyfrowania danych zarządzanych przez dostawcę usług. Natomiast odpowiedzialnością klienta jest odpowiednie skonfigurowanie użytkowników i ich pozwoleń, ochrona haseł, konfiguracja dodatkowych zabezpieczeń jak szyfrowanie danych na dyskach lub wykorzystanie bezpiecznego protokołu do transportowania danych w publicznej sieci Internetu, stosowanie konfiguracji usług wymuszających bezpieczne standardy takie jak TLS v1.2 czy też monitorowanie i audyt systemu pod kątem bezpieczeństwa.



Rysunek 5. Schemat modelu współodpowiedzialności za bezpieczeństwo w AWS. Źródło: [23]

Przykładami usług wymagających dodatkowych konfiguracji bezpieczeństwa są AWS API Gateway czy też AWS CLI w wersji pierwszej – obie te usługi domyślnie korzystają ze standardu TLS v1.0, który zgodnie ze standardem PCI jest nie rekomendowany do użytku, gdyż odnotowano podatności naruszenia bezpieczeństwa danych z wykorzystaniem tego standardu [13]. W przypadku usługi AWS API Gateway należy skonfigurować usługę DNS, dzięki której możliwe jest użycie standardu TLS v1.2. W przypadku AWS CLI należy zaktualizować wersję oprogramowania do wersji nie mniejszej niż 2.0.

Kolejną charakterystyką usług chmurowych jest kwestia bezpieczeństwa i prywatności danych. Jeśli mamy do czynienia z wykorzystaniem chmury publicznej lub chmury hybrydowej to dane będą

przechowywane na infrastrukturze dostawy usług chmurowych w zdefiniowanej lokalizacji czy też regionie. Zgodnie z modelem współdzielonej odpowiedzialności, do zapewnienia pełnego bezpieczeństwa danych zobowiązani są zarówno klient oraz dostawa usług chmurowych. Jednakże istnieją akty prawne w kilku państwach, które nakazują przechowywanie danych wyłącznie w zdefiniowanych regionach geograficznych, najczęściej w obrębie granic danego państwa. Przykładami państw, które posiadają takie regulacje prawne to USA, Rosja, Niemcy, Chiny. Niektórzy dostawcy usług chmurowych oferują dedykowane regiony, aby sprostać tym regulacjom. Przykładami są regiony AWS China dla zaspokojenia przepisów prawa Chin lub Microsoft Azure GovCloud oferujące specjalne regiony dla rządu Stanów Zjednoczonych Ameryki [24].

Aby zapewnić bezpieczeństwo danych należy dokonywać szyfrowania danych z wykorzystaniem zatwierdzonych standardów kryptograficznych (AES256) podczas przechowywania danych na dyskach (ang. *encryption on rest*) oraz podczas transportu sieciowego (ang. *encryption on transit*). Do zaszyfrowania danych można wykorzystać predefiniowane klucze utrzymywane przez dostawcę usług chmurowych lub klient może dostarczyć własne klucze do szyfrowania. Kolejnym bardzo istotnym aspektem zapewnienia bezpieczeństwa w systemach chmurowych jest odpowiednie zarządzanie pozwoleniami użytkowników zgodnie z zasadą najmniejszych pozwoleń (ang. *at least privileges principle*) [25]. Zgodnie z tą zasadą użytkownicy otrzymują minimalny zestaw pozwoleń, dzięki którym będą w stanie wykonywać swoje obowiązki. Stosowanie się do najlepszych praktyk zasad bezpieczeństwa oraz wykorzystywanie licznych usług do zarządzania bezpieczeństwem pozwala osiągnąć bezpieczeństwo danych w systemach chmurowych.

W przypadku omawiania systemów chmurowych można napotkać się na określenie usług bezserwerowych (ang. *serverless*). Jest to grupa usług chmurowych do budowy systemów, które nie wymagają zarządzania infrastrukturą [8]. Wykorzystanie i zintegrowanie różnych usług o charakterze *serverless* pozwoli zbudować kompletny system w krótszym czasie, gdyż zespół programistów będzie mógł skupić się na rozwiązywaniu problemów biznesowych zamiast zajmować się konfiguracją wymaganej infrastruktury. Ponadto usługi typu *serverless* domyślnie oferują lub w znacznym stopniu upraszczają stworzenie skalowanego i łatwo replikowanego systemu. Do sprawnego zarządzania komponentami usług typu *serverless*, AWS dostarczył *framework* SAM (ang. *Serverless Application Model*). Dostarcza on interfejs wiersza poleceń z komendami, które łatwo mogą zbudować skrypt infrastruktury jako kodu, a następnie dokonać wdrożenia aplikacji na środowisko [8]. Tabela 2 ukazuje zbiór popularnych usług *Serverless* różnego typu co potwierdza mnogość zastosować.

Tabela 2. Usługi AWS z dziedziny *Serverless*. Źródło: [8]

Nazwa usługi	Typ
AWS Lambda	moc obliczeniowa
AWS ECS Fargate	moc obliczeniowa
AWS Event Bridge	integracja aplikacji
AWS SQS	integracja aplikacji
AWS SNS	integracja aplikacji
AWS API Gateway	integracja aplikacji
AWS S3	przechowywanie danych
AWS DynamoDB	przechowywanie danych
AWS RDS Aurora	przechowywanie danych

Usługi chmurowe charakteryzują się także dostępnym wsparciem technicznym. W przypadku napotkania problemu technicznego można zgłosić swój problem z pomocą dostępnych kanałów

pomocy. Zakres pomocy różni się w zależności od wykupionego pakietu, co skutkuje określonych czasem otrzymania pomocy [26]. Jest to atrakcyjna cecha z perspektywy przedsiębiorstw, gdyż w razie problemów zawsze jest możliwość poproszenia o pomoc ekspertów. Ponadto rosnąca popularność oraz wzrost liczby użytkowników usług chmurowych wpływają na fakt, że duża dziedzina standardowych problemów jest omawiana i udokumentowana w przestrzeni Internetu po przez społeczność. Fakt ten znacząco przyspiesza wyszukanie rozwiązania na zaistniałe problemy.

2.2.4 Dostawcy usług chmurowych

Rynek dostawców usług chmurowych nieustannie zwiększa się o kolejne przedsiębiorstwa oferujące usługi chmurowe w swojej ofercie. W obecnej chwili udział w rynku dostawy usług chmurowych wygląda następująco [7]:

- Amazon Web Services (AWS) – 40%;
- Microsoft Azure - 20%;
- Google Cloud Platform (GCP) – 8%;
- Alibaba Group – 5%;
- Inne – 27%.

Amazon był pierwszym dużym przedsiębiorstwem, który oferował usługi chmurowe określane jako AWS. Jedną z pierwszych usług była usługa AWS S3. Obecni główni konkurenci dla AWS, czyli Microsoft Azure oraz Google Cloud Computing zaczęli wdrażać swoje usługi kilka lat po firmie Amazon. Pozwoliło to znacząco umocnić się na rynku i zyskać rozpoznawalność lidera rynku cytując ówczesnego CEO Amazona, Jeffa Bezosa: „*AWS had the unusual adventage of a seven-year head start before facing like-minded competition. As a result, the AWS services are by far the most evolved and most functionality-rich.*" [7]. Obecnie obserwuje się zmniejszanie dystansu w udziale w rynku pomiędzy głównymi konkurentami, a AWS. Usługi chmurowe firmy Amazon skupiają się na zasobach obliczeniowych, bazach danych, integracjach oraz migracjach. Przewagą Microsoft Azure jest natomiast fakt, że Microsoft posiada dużo własnych produktów, które są zintegrowane w usługach chmurowych. Natomiast firma Google szuka swojej przewagi rynkowej w usługach związanych z Big Data oraz nauczaniem maszynowym. Alibaba Group jest chińskim przedsiębiorstwem oferującym usługi chmurowe i właśnie na tym rynku posiada najwięcej klientów.

2.2.5 Rodzaje usług chmurowych oferowanych przez AWS

W ofercie usług AWS dostępne są liczne serwisy i platformy, które umożliwiają prowadzenie biznesu w oparciu o system zbudowany w chmurze. W menu konsoli AWS usługi te pogrupowane są względem potencjalnych zastosowań takich jak zasoby obliczeniowe, bazy danych, sieci, migracje, zarządzanie chmurą, uczenie maszynowe czy też usługi deweloperskie. Poniżej dokonany jest opis charakterystyk usług, które są powszechnie wykorzystywane przy budowanie natywnych systemów chmurowych.

- AWS Lambda – usługa umożliwiająca uruchomienie funkcji z dostarczonym kodem. Wspiera większość popularnych języków programowania takich jak Python, Java, NodeJS, C#, Go. Maksymalny czas wykonywania funkcji to 15 minut, więc usługa ta przeznaczona jest do wykonywania niewielkich zadań obliczeniowych. Funkcje te są uruchamiane na podstawie zdefiniowanych zdarzeń. Funkcje Lambda bardzo efektywnie skalują się horyzontalnie. Ponadto są bardzo tanie, gdyż oferują rozliczanie

milisekundowe w modelu „płać tylko gdy korzystasz”. Jest to specyficzna odmiana usług typu PaaS, która zyskując na popularności, została nazwana FaaS (ang. *Function as a Service*);

- AWS DynamoDB – usługa nierelacyjnej bazy danych typu klucz – wartość (ang. *key - value database*). Dostęp do danych wynosi do 50 ms, a w przypadku wykorzystania akceleratora wykorzystującego pamięć podręczną, dostęp do danych wynosi około 1ms. Umożliwia automatyczne tworzenie kopii zapasowych z możliwością przywrócenia do danego punktu w czasie, wspiera tworzenie globalnych tabel w wielu regionach oraz umożliwia śledzenia zmian w danych co umożliwia zastosowanie wyzwalaczy;
- AWS S3 – (ang. *Simple Storage Service*) usługa do przechowywania obiektów takich jak pliki tekstowe, zdjęcia, filmy. Charakteryzuje się bardzo wysoką dostępnością i niezawodnością przechowywania danych. Obiekty przechowywane są w formie prefiksów w obrębie przestrzeni nazw danego pojemnika S3 (ang. *S3 Bucket*). Charakteryzuje się bardzo wysoką wydajnością – potrafi obsłużyć do 5000 zapytań na sekundę do każdego prefiksu. Posiada różne opcje przechowywania danych, które wpływają na czas dostępu do danych oraz cenę za przechowywanie danych na przykład S3 Standard, S3 Glacier, S3 Infrequent Access. Była to pierwsza usługa chmurowa uruchomiona przez AWS w 2006 roku;
- AWS API Gateway – usługa do zbudowania i udostępnienia interfejsu API typu REST lub WebSocket. Umożliwia wykorzystanie rozszerzonych mechanizmów autentykacji w dostępie do udostępnionych zasobów. Dostarcza ustandaryzowany sposób budowy programowanych punktów końcowych (ang. *API endpoint*). Wspiera standardy OpenAPI dzięki czemu możliwe jest zaimportowanie pliku swagger.yaml. Usługa ta implementuje wzorzec projektowy „Fasada” w dostępie do usług systemu [5];
- AWS Timestream – usługa baz danych zorientowana na wymiar czasowy (ang. *time-series*). Udostępnia ona wbudowane wizualizacje z wyszczególnieniem wymiaru czasu oraz monitorowanego aspektu np. temperatury powietrza;
- AWS SQS – (ang. *Simple Queue Service*) - usługa komunikacji pomiędzy komponentami w postaci kolejki. Udostępnia tryb Standard oraz FIFO (ang. *First In First Out*). Służy do kolejkowania danych do przetworzenia oraz umożliwia architektoniczne oddzielanie komponentów (ang. *decoupling*) poprzez zastosowanie kanału komunikacji opartego na zdarzeniach (ang. *event driven architecture*);
- AWS CloudWatch – usługa do monitorowania ekosystemu usług AWS oferująca metryki usług, alarmy, przechowywanie i przeszukiwanie logów aplikacyjnych, wizualizacje i kokpity administracyjne. Zawiera usługę AWS CloudWatch Insights, która jest bardzo potężnym silnikiem wyszukiwania. Ponadto usługa ta udostępnia deklaratywny język zapytań oraz efektywne wyszukiwanie pełnotekstowe;
- AWS IAM – (ang. *Identity Access Management*) – usługa dzięki której możliwe jest zarządzanie związanie autentykacją i autoryzacją encji korzystających z usług i zasobów chmurowych. Pozwala na stworzenie użytkowników, grup , ról oraz polityk dostępu do zasobów. Zapewnia ziarniste przyznawanie pozwoleń zgodnie z zasadą najmniejszych pozwoleń;

- AWS EC2 – (ang. *Elastic Compute Cloud*) – usługa pozwalająca wynająć zasoby obliczeniowe w postaci maszyn wirtualnych. Dostępne są różne systemy operacyjne i różne parametry oferowanych zasobów takie jak zoptymalizowane do operacji wejścia i wyjścia czy też do ogólnego przeznaczenia. Podstawowa usługa AWS z obszaru IaaS;
- AWS ECS – (ang. *Elastic Container Service*) – usługa umożliwiająca wykorzystanie technologii konteneryzacji do budowy aplikacji. Oferuje wersję bez serwerową (AWS ECS Fargate) oraz wersję serwerową opartą na wykorzystaniu usługi AWS EC2;
- AWS CloudFormation – usługa umożliwiająca tworzenie i zarządzanie infrastrukturą systemu w postaci zdefiniowanego szablonu i definiowanie infrastruktury jako kod. Pozwala to na automatyzację procesu tworzenia systemu oraz wdrożenie metodologii CI/CD;
- AWS CodeCommit – usługa oferująca wersjonowanie plików z wykorzystaniem technologii Git. Usługa jest dobrze zintegrowana z innymi komponentami pomocnymi przy budowie systemu CICD;
- AWS CodePipeline – usługa umożliwiająca zbudowanie potoku automatycznych operacji niezbędnych do wdrożenia nowej wersji oprogramowania. Pozwala utworzyć reakcje systemu na pojawienie się nowego kodu źródłowego w repozytorium w usłudze AWS CodeCommit, a następnie zbudowania spakowanych paczek z kodem źródłowym i niezbędnymi zależnościami, które to zostaną przekazane do usługi AWS CloudFormation celem wdrożenia zasobów na środowisko.

2.2.6 Interfejsy dostępu do usług AWS

Użytkownik ma możliwość interakcji z usługami chmurowymi AWS za pośrednictwem konsoli udostępnionej w przeglądarce internetowej, z wykorzystaniem komend wiersza poleceń lub SDK (ang. *Software Development Kit*) dla danego języka programowania. Wykorzystanie przeglądarkowej konsoli jest bardzo wygodne i umożliwia dostęp do zasobów z każdego miejsca na świecie z dostępem do Internetu. Wygląd i funkcjonalności są nieustannie rozwijane, lecz już w chwili obecnej praca z konsolą jest subiektywnie intuicyjna. W przypadku automatyzacji czynności administracyjnych lub implementacji logiki biznesowej wymagającej interakcji z usługami AWS należy skorzystać z AWS CLI (ang. *Command Line Interface*) lub z SDK dla wybranego języka programowania. W przypadku interfejsu wiersza poleceń należy pobrać pakiet instalacyjny i zainstalować go na swoim systemie operacyjnym, aby móc z niego skorzystać zgodnie z udokumentowanymi komendami dostępu. Zaleca się stosowanie AWS CLI w wersji drugiej, gdyż ta wersja jest rozwijana i zapewnia rekomendowane standardy bezpieczeństwa.

2.3. Charakterystyka systemów rozproszonych wykorzystujących technologię konteneryzacji

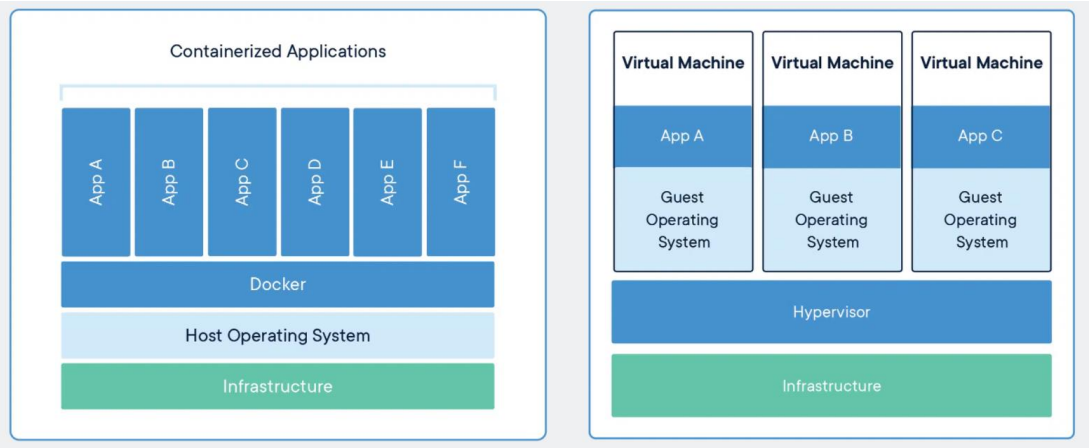
Technologia konteneryzacji polega na umieszczaniu aplikacji w niezależnych, wyizolowanych jednostkach obliczeniowych zwanych kontenerami. Do rozwoju tego podejścia przyczyniła się chęć optymalizacji wykorzystania fizycznych zasobów komputerowych. Proces ten ewoluował i przyczynił się do powstania konceptu wirtualizacji oraz konteneryzacji. Konteneryzacja izoluje oprogramowanie od fizycznego sprzętu przez co kontenery są łatwo przenoszalne. Ponadto dobrze sprawdzają się w izolowaniu funkcjonalności wyznaczając granice architektoniczne systemu. Cechą kontenerów jest łatwość skalowania horyzontalnego co uzyskuje się wykorzystując oprogramowanie do zarządzania kontenerami.

2.3.1 Ewolucja optymalizacji wykorzystania zasobów obliczeniowych

W epoce przed chmurą obliczeniową, przedsiębiorstwa były zmuszone nabyć sprzęt komputerowy dla swoich usług komputerowych już na początku projektu w celu wdrożenia swojego oprogramowania. Koszt zakupu sprzętu musiał być pokryty przed dokładnym poznaniem dokładnych wymogów i charakterystyk użytkowania serwowanych usług. Dokonywano zatem szacowania spodziewanego wykorzystania sprzętu. Aby poprawnie obsłużyć okresy szczególnie intensywnego wykorzystywania usługi, dokonywano zakupu sprzętu o parametrach znacznie większych niż wymaga tego standardowe procesowanie. To powodowało niewykorzystane potencjału zasobów obliczeniowych przez znakomitą większość czasu użytkowania sprzętu. W tym trybie, aby izolować niezależne od siebie aplikacje, należało zakupić kolejny sprzęt komputerowy w celu uruchomienia nowej aplikacji co potęgowało nieracjonalność ekonomiczną.

Chęć optymalizacji zasobów obliczeniowych doprowadziła do rozwoju technologii wirtualizacji, dzięki której było możliwe uruchomienie kilku niezależnych systemów operacyjnych na jednej fizycznej maszynie. Na każdym z tych wirtualnych systemów można było uruchamiać osobne aplikacje. Taka izolacja aplikacji na tym samym fizycznym sprzęcie była dużą optymalizacją wykorzystania potencjału obliczeniowego. Dzięki temu w większości przypadków nie było potrzeby zakupu dodatkowych maszyn, aby uruchomić nową aplikację. Popularnymi narzędziami do wirtualizacji sprzętu komputerowego są VMware lub VirtualBox.

Mimo znacznej optymalizacji uzyskanej dzięki technologii wirtualizacji to inżynierowie spoglądali w kierunku dalszych optymalizacji wynikających z właściwości wirtualizacji. Mianowicie każda maszyna wirtualna musiała rezerwować miejsce na swój system operacyjny. Częstym przypadkiem jest fakt, że wszystkie aplikacje działające na maszynach wirtualnych na tym samym fizycznym zasobie komputerowym używają dokładnie tego samego systemu operacyjnego. Powoduje to redundancję plików systemu operacyjnego co zostało zobrazowane na Rysunku 6. Kolejnym aspektem optymalizacyjnym był czas uruchomienia aplikacji działającej na maszynie wirtualnej. Aby nowa instancja aplikacji została uruchomiona, na przykład w wyniku skalowania, należało odczekać aż wykonają się wszystkie procesy odpowiedzialne za uruchomienie systemu operacyjnego. Te możliwe aspekty optymalizacyjne doprowadziły do popularyzacji technologii konteneryzacji, która wyewoluowała z technologii wirtualizacji.



Rysunek 6. Porównanie konteneryzacji i wirtualizacji. Źródło: [27]

Kontenery wykorzystują system operacyjny hosta na którym się znajdują [27]. Przy pomocy procesu zarządzającego kontenerami, możliwa jest koordynacja pracy wielu kontenerów. Ten proces działa w tle i w przypadku platformy Docker nazywa się on Docker Deamon. Skorzystanie z kontenerów względem maszyn wirtualnych oferuje szybszy czas uruchomienia aplikacji, gdyż nie wymagany jest rozruch systemu operacyjnego. Ponadto możliwe jest wykorzystanie bardzo lekkich kontenerów z minimalnym zestawem funkcjonalności, które również przyspieszają uruchomienie danej funkcjonalności. Każdy kontener jest rozpatrywany jako wyizolowany komponent realizujący dane zadanie. Kontenery posiadają własną przestrzeń procesów oraz system plików. Aby umożliwić dostęp do plików kontenerów należy dokonać specjalnej konfiguracji mapującej przestrzeń dyskową hosta na wirtualny system plików kontenera, który domyślnie jest efemeryczny. Potencjalnymi wadami technologii konteneryzacji jest aspekt bezpieczeństwa, gdyż nieprawidłowa konfiguracja zabezpieczeń i pozwoleń może doprowadzić do uzyskania dostępu do hosta kontenerów.

W związku z tym, że kontenery szybko uruchamiają kolejne instancje oraz łatwo izolują architektonicznie funkcjonalności systemu to bardzo dobrze wpisują się w metodykę skalowania horyzontalnego. Maszyny wirtualne jednakże uważane są za technologię oferującą większe bezpieczeństwo. Technologie te uzupełniają się i obecnie są powszechnie wykorzystywane przy budowie systemów informatycznych.

2.3.2 Charakterystyka technologii konteneryzacji na przykładzie platformy Docker

Podstawową, niezależną jednostką oprogramowania w technologii konteneryzacji jest instancja kontenera (ang. *container instance*). Funkcjonalność i zależności kontenera zdefiniowane są w obrazie kontenera (ang. *container image*). W obrazie kontenera definiuje się przepis na kontener. Składa się na to referencja do obrazu źródłowego, następnie dodaje się wszystkie dodatkowe zależności oraz pliki źródłowe aplikacji. Ponadto można wyspecyfikować konfigurację sieciową (*docker networks*), podpięcie przestrzeni dyskowej hosta (*docker volumes*) w celu utrwalenia danych czy też dokonać mapowania portów hosta i kontenera. Po zbudowaniu obrazu kontenera staje się możliwe powoływanie instancji kontenerów wykorzystujących dany obraz. Dokonana jest w ten sposób izolacja oprogramowania od systemu operacyjnego. Dzięki temu uzyskuje się bardzo pożądaną cechę przenoszalności oprogramowania pomiędzy urządzeniami, gdyż wszystkie zależności są zdefiniowane w definicji obrazu i mogą zostać odtworzone w dokładnie ten sam sposób na każdym zasobie

komputerowym korzystającym z platformy Docker. Rozwiązuje to problem działania aplikacji na danej maszynie i problemie w działaniu aplikacji po przeniesieniu na inną maszynę w związku z ukrytymi zależnościami.

Platforma Docker została uruchomiona w 2013 roku [27] i systematycznie zyskiwała na popularności. Jest to otwarte i publicznie dostępne oprogramowanie (ang. *open source*). Skutkuje to bardzo dużą liczbą gotowych obrazów kontenerów różnych aplikacji, które można od razu wykorzystać lub rozbudować. Składają się na to skonteneryzowane silniki baz danych, środowiska programistyczne, komponenty architektoniczne takie jak kolejki, serwery sieciowe. Obrazy kontenerów przechowywane są w repozytorium obrazów. Publiczne repozytorium producenta platformy Docker nazywa się DockerHub.

Definicję pojedynczego obrazu kontenera umieszcza się w pliku zwanym domyślnie Dockerfile. Przykładową zawartość tego pliku ukazano w Listingu 1. Wykorzystując zdefiniowane komendy można dostarczyć przepis na swój własny kontener lub rozbudować istniejący obraz źródłowy. Narzędziem do definiowania wielu obrazów kontenerów, które współpracując ze sobą tworzą spójny komponent, jest Docker Compose. Taki sposób definicji obrazów pozwala zautomatyzować konfigurację połączeń sieciowych czy też współdzielenie zmiennych środowiskowych. Jest to możliwe poprzez bezpośrednie odwoływanie się do innych obrazów kontenerów poprzez nazwy zdefiniowane w pliku Docker Compose. Znacząco to ułatwia integrację kontenerów.

Listing 1. Przykładowa zawartość pliku Docker dla aplikacji napisanej w języku Python. Źródło: opracowanie własne

```
FROM python:3.8-slim-buster
VOLUME ["/log_storage"]
ENV SERVER_URL url
WORKDIR /app
EXPOSE 5000:5000
COPY requirements.txt ./
RUN pip3 install -r requirements.txt
COPY . .
CMD [ "python", "./app.py" ]
```

Możliwość zdefiniowania obrazu źródłowego wpływa na rozmiar ostatecznej instancji kontenera. Istnieje specjalna rodzina obrazów zwana *docker alpine*, która charakteryzuje się minimalnym zestawem funkcjonalności co sprawia, że rozmiar kontenera jest bardzo mały. Takie obrazy idealnie pasują do urządzeń końcowych IoT, które posiadają ograniczone moce obliczeniowe i pamięć.

Obraz kontenera budowany jest z wykorzystaniem współdzielonych warstw. Każda komenda zdefiniowana w pliku Dockerfile tworzy osobną warstwę obrazu. Efektywne budowanie obrazów wymaga zabiegów, które umożliwiają ponowne użycie już zbudowanych warstw niewymagających przebudowy [28]. Stąd zaleca się deklarowanie poleceń w pliku Dockerfile w kolejności od najmniej do najczęściej wpływających na zmianę plików, co wymaga przebudowy warstwy zamiast sięgnąć do pamięci podręcznej.

Kontenery posiadają efemeryczną pamięć dyskową. Domyślnie dane wewnątrz instancji kontenera zostaną utracone w momencie zatrzymania lub usunięcia kontenera. Aby utrwalić dane

generowane przez kontener należy wykorzystać mechanizm zwany *docker vokumes*. Dostarcza to możliwości umieszczania danych wytworzonych przez kontener na dysku hosta z możliwością ponownego dostępu wraz z uruchomieniem nowej instancji kontenera.

Platforma Docker udostępnia komendy wiersza poleceń, które pozwalają zautomatyzować procesy związane z budowaniem obrazów, uruchomieniem i zatrzymywaniem instancji. Za obsługę poleceń odpowiedzialny jest proces zwany Docker Demon. Pośredniczy on pomiędzy systemem operacyjnym hosta, a ekosystemem kontenerów. Platforma Docker jest dostępna na systemy operacyjne Linux oraz Windows. Charakterystyka działania procesu Dockera wymaga, aby na pojedynczym hoście obsługiwane były kontenery na system Windows albo Linux, lecz nie oba jednocześnie.

Do zarządzania dużą liczbą kontenerów wykorzystuje się usługi zarządców kontenerów:

- Docker Swarm;
- Kubernetes.

Oprogramowanie to dostarcza mechanizmów do strategii autoskalowania, samo regeneracji kontenerów, alarmowania i monitoringu floty kontenerów.

Kontenery są powszechnie stosowane w budowie systemów informatycznych takich jak architektury mikro serwisowe. Powszechnie stosowane są w implementacji rozwiązań chmurowych takich jak usługi bez serwerowe [27].

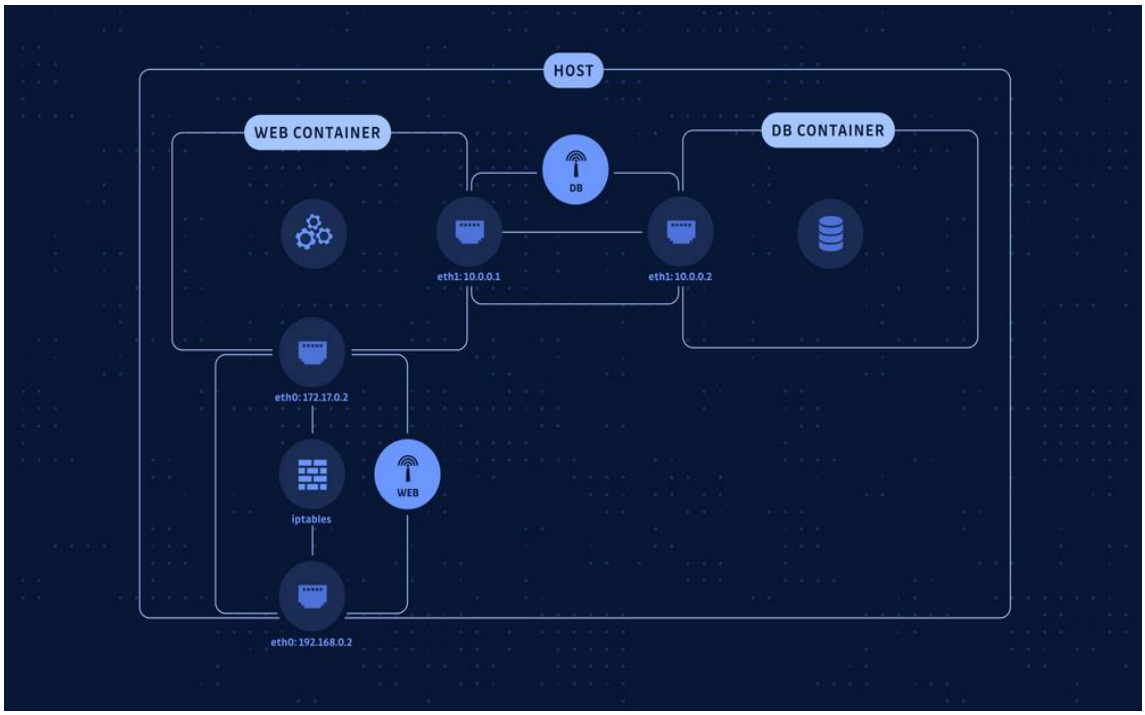
2.3.3 Aspekty bezpieczeństwa technologii konteneryzacji

Twórcy platformy Docker oświadczają, że kontenery są stosunkowo bezpiecznie. Argumentują to faktem, że główne technologie (*cgroup* oraz *namespaces*) wpływające na bezpieczeństwo kontenerów powstały wraz z nową wersją Linuxa w roku 2008, przez co technologie te są dojrzałe i zostały skrupulatnie przetestowane na wielu systemach produkcyjnych [29]. Narzędzia te służą do izolacji procesów i dostępu do zasobów. Każda powstała instancja kontenera porusza się w swojej przestrzeni nazw procesów i nie ma dostępu do procesów hosta lub innych kontenerów. Funkcjonalność cgroups warunkuje sprawiedliwy dostęp do procesora i pamięci dla wszystkich kontenerów przez co znacząco ogranicza podatność na ataki typu DoS. Jest to szczególnie istotne dla kontenerów wykorzystywanych jako część systemu chmury publicznej lub prywatnej. Dla początkujących użytkowników Dockera to pojęcie izolacji może złudnie gwarantować niezawodne bezpieczeństwo aplikacji. Należy przestrzegać kilku bardzo istotnych praktyk, aby uznać skonteneryzowaną aplikację za bezpieczną [30].

Głównym niebezpieczeństwem jest uzyskanie dostępu do super użytkownika hosta na którym znajduje się kontener. Proces Docker Deamon wymaga domyślnie pozwoleń super użytkownika. W efekcie atakujący może potencjalnie przejąć kontrolę nad całym hostem oraz wszystkimi kontenerami, które są tam ulokowane. Aby temu zapobiec należy uruchamiać kontenery z użytkownikiem, który nie jest super użytkownikiem oraz ma ograniczone uprawnienia zgodnie z zasadą najmniejszych pozwoleń. Ponadto należy nie udostępniać publicznej lub niezaufanej komunikacji z gniazdem API dla Docker Deamon (*/var/run/docker.sock*) , gdyż to efektywnie pozwala uzyskać prawa super użytkownika [30].

Kolejnym aspektem jest montowanie dostępu dla kontenera do wrażliwych plików hosta takie jakie pliki systemowe. Technicznie nic nie stoi na przeszkodzie, aby wdrożyć taką konfigurację, lecz może ona spowodować utratę funkcjonowania całego hosta, gdy krytyczne pliki zostaną zmodyfikowane lub usunięte. Jeżeli istnieje potrzeba dostępu do dysku twardego hosta, należy wykorzystać opcję dostępu do plików tylko do odczytu lub narzędzie Docker Volumes,

Zaleca się stworzenie dedykowanej sieci dockera (*docker networks*) na każde połączenie sieciowe, które jest wymagane przez dany kontener. Prawidłowa praktyka ustawienia sieci została pokazana na Rysunku 7. Stworzenie nowego kontenera domyślnie przypisuje go do domyślnej sieci nazwanej *docker0.* Wszystkie kontenery w tej samej sieci mogą się ze sobą komunikować. Zatem zaniechanie tej uwagi może prowadzić do nieuprawnionego dostępu ze strony innych kontenerów.



Rysunek 7. Wykorzystanie dedykowanych sieci na każde połączenie kontenera. Źródło: [30]

Niemal oczywistą kwestią jest nieumieszczenie zapisanych na sztywno sekretnych wartości w definicji obrazów kontenerów w plikach Dockerfile takich jest hasła dostępu. Tego typu naruszenia bezpieczeństwa stały się plagą, co nakłoniło twórców Dockera to uruchomienia usługi skanowania repozytorium obrazów pod tym kątem, co tylko potwierdziło tezę, że skala zjawiska jest wysoka. Poufne wartości powinny być przekazywane do obrazu w postaci zmiennych środowiskowych, które umieszcza się w dołączonym pliku konfiguracyjnym. Plik ten nie powinien być nigdzie udostępniany, na przykład w repozytorium kodu.

Dobrą praktyką jest korzystanie tylko z obrazów kontenerów, które pochodzą z zweryfikowanych źródeł takich jak DockerHub oraz mają prawidłowy podpis cyfrowy. Opcja ta jest domyślnie wyłączona i nie należy nie rezygnować z weryfikowania podpisu. Zaniechanie tej praktyki może doprowadzić do uruchomienia złośliwego oprogramowania. Jeśli to możliwe należy korzystać z obrazów typu *alpine* , gdyż posiadają one najmniejszy zestaw funkcjonalności, zatem pole manewru do ataku na aplikację również jest zmniejszone [30].

3. Metodologie i narzędzia zastosowane w pracy

Budowa współczesnych systemów informatycznych charakteryzuje się stosowaniem licznych metodologii i wzorców projektowych, których zadaniem jest uzyskanie ładu architektonicznego i elastyczności budowanego rozwiązania technicznego. Ma to za zadanie ułatwić budowę, utrzymanie oraz rozwój funkcjonalności. Szczególnie istotny jest aspekt wdrażania nowych wymagań biznesowych bez konieczności znacznej przebudowy dotychczasowej konfiguracji. Metodologie i narzędzia wykorzystane przy realizacji pracy wpisują się w powszechnie stosowane najlepsze praktyki czego przykładami są CI/CD, zasady SOLID czy też procesowanie oparte na zdarzeniach. Wykorzystane narzędzia znacząco ułatwiają pracę programisty oraz zarządcy systemu informatycznego na co składają się zintegrowane środowiska programistyczne, kokpity i wizualizacje administracyjne czy też systemy wersjonowania plików.

3.1. Charakterystyka zastosowanych metodologii

Wraz z dynamicznym rozwojem obszaru informatyki powstają metodyki, które starają się rozwiązać daną dziedzinę problemów pojawiających się przy prowadzeniu projektów informatycznych. Dla każdej fazy powstawania projektu znajdą się liczne metodyki, które pozwolą usprawnić proces. Istnieją metodyki organizacji cyklu dostarczania oprogramowania takie jak metodyki zwinne (ang. *agile*) lub metodyki kaskadowe (ang. *waterfall*). Ponadto są różne metodologie do pisania logiki biznesowej aplikacji w oparciu o programowanie obiektowe czy też funkcyjne. W obszarze testowania systemu również dostępne są różne podejścia takie jak dostarczanie oprogramowania w oparciu o testy (ang. *test-driven development*), testy automatyczne, testy penetracyjne czy też zamierzone wstrzykiwanie błędów do aplikacji, aby poznać odporność systemu na awarie. Ma to zastosowanie w systemach produkcyjnych firmy Netflix [36]. W przypadku metodologii definiowania architektury należy wyróżnić systemy monolityczne, zmodularyzowane monolity czy też systemy rozproszone. Utrzymanie systemu również doprowadziło do zdefiniowania metodyk, które umożliwiają sposoby odbudowy systemu w danym czasie (ang. *RTO – recovery time objective*) oraz zapewniają monitoring, definicję alarmów i procedury reakcji na alarmy, na przykład systemy obsługi zgłoszeń. Spośród szerokiej gamy dostępnych możliwości, projektant systemu musi podjąć decyzję na temat zbioru metodologii, które w najefektywniejszy sposób pomogą zrealizować założone cele biznesowe.

3.1.1 Metodologia ciągłej integracji i dostarczania oprogramowania

Wzrost skomplikowania systemów i aplikacji spowodował konieczność zatrudnienia większej liczby inżynierów do rozwoju oprogramowania. Aby skoordynować i uczynić efektowną pracę wielu zespołów nad tym samym produktem, zaistniała potrzeba stworzenia reguł i narzędzi, które by to umożliwiły. W związku z tym powstała metodyka ciągłej integracji i dostarczania oprogramowania (ang. *Continuous Integration Continuous Delivery*), w skrócie CI/CD.

Podstawami założeń metodyki CI/CD jest automatyzacja i kontrola procesów, które umożliwiają dostarczanie nowych wersji oprogramowania w ustrukturyzowany, powtarzalny sposób, który zapewnia kontrolę jakości oprogramowania i założeń biznesowych [32]. Zrealizowane jest to poprzez zdefiniowanie i zaprogramowanie potoków wdrożenia (ang. *deployment pipelines*), które składają się z wielu kroków procesu. Końcowym krokiem procesu jest bezpieczne wgranie nowej

wersji na środowisko produkcyjne przy zachowaniu jakości i kontroli produktu oraz przy minimalnym nakładzie administracyjnym. Pełna lub znaczna automatyzacja procesu minimalizuje błędy ludzkie oraz zmniejsza czas realizacji wdrożenia. Zdefiniowanie poszczególnych kroków potoku wdrożenia zależy od specyfiki projektu oraz wykorzystywanych technologii. W ogólności można wyróżnić następujące fazy :

- rewizja wprowadzonych zmian w kodzie źródłowym;
- pobranie plików kodu źródłowego z repozytorium;
- zbudowanie obrazów kontenerów;
- spakowanie i transformacja plików kodu źródłowego do formatu wykonywalnego;
- pobranie dynamicznych konfiguracji, wstrzyknięcie zmiennych środowiskowych;
- uruchomienie testów jednostkowych;
- uruchomienie testów regresyjnych oraz integracyjnych;
- zatwierdzenie wersji oprogramowania do wdrożenia przez osobę zatwierdzającą;
- notyfikacje o wdrożeniu do osób zainteresowanych;
- wgranie nowej wersji na środowisko;
- automatyczne wycofanie zmian w przypadku niepowodzenia wdrożenia.

Przykładowymi narzędziami do definiowania potoków wdrożenia są produkty takie jak Jenkins, Buddy, AWS CodePipelines, Microsoft Azure Pipelines [45, 46, 47]. Narzędzia powiązane z metodologią CI/CD to systemy wersjonowania plików takie jak Git oraz repozytoria kodu – Github, Bitbucket, Gitlab. Zakres procesu CI/CD obejmuje także określenie tzw. *git flow*. Oznacza to zdefiniowanie zasad tworzenia tematycznych gałęzi kodu do zaimplementowania nowej funkcjonalności, a następie ich rewizji i akceptacji. Przykładowymi aspektem jest ustawienie wymaganej liczby osób do potwierdzenia jakości kodu, w tym co najmniej jedną osobę na stanowisku seniorskim, aby uznać kod jako gotowy do wdrożenia. Narzędzia CI/CD mogą automatycznie reagować na zmiany w wersji kodu na repozytorium za pomocą tzw. *web-hooks*. Umożliwia to automatyzację procesu w wyniku zareagowania na zdarzenie takie jak połączenie kodu z gałęzi pobocznej do gałęzi głównej.

3.1.2 Systemy oparte na procesowaniu zdarzeń

Metodologia projektowania architektury systemów w oparciu o reagowanie na zaistniałe zdarzenia posiada wiele istotnych zalet. Zdarzenie posiada formę dokonaną, która powoduje uruchomienie danego procesu. Zatem system reaguje na fakty, a nie na domniemania i oczekiwania, że coś się wydarzy w określony sposób. Zdarzenia są ponadto niezmienne (ang. *immutable*) i jednoznaczne co oznacza, że dokładnie jest wiadomo jaka zmiana stanu systemu spowodowała powstanie zdarzenia. Projektowanie systemów opartych na zdarzeniach sprowadza się do zdefiniowania komponentów, które generują zdarzenia (ang. *event producers*) oraz te, które je odbierają i procesują (ang. *event consumers*). Wzorce projektowe opisujące tego typu rodzaje komunikacji to publikacja-subskrypcja w przypadku wielu odbiorców lub kolejka zdarzeń w przypadku relacji jeden do jednego pomiędzy producentem i konsumentem [5]. Ten rodzaj komunikacji pomiędzy komponentami umożliwia luźne powiązania oraz rozdzielanie komponentów

(ang. *decoupling*). Pozwala to na niezależny rozwój tych komponentów, jasno zdefiniowany interfejs komunikacyjny czy też możliwość rozwoju w dowolnych technologiach i językach programowania. Pożądaną cechą architektoniczną jest fakt, że producenci nie wiedzą o konsumentach i nie muszą się martwić o to w jaki sposób zdarzenie zostanie przeprocesowane. Natomiast konsumenci nie muszą się zastanawiać w jaki sposób zdarzenie zostało wygenerowane, tylko skupiają się na tym jak powinno zostać przeprocesowane [31].

Procesowanie w oparciu o zdarzenia bardzo ułatwia skalowanie systemu. Można przyjąć, że każda instancja zdarzenia to pojedyncze, niezależne zadanie do wykonania. Pozwala to dokonywać zrównoleglenia przetwarzania zdarzeń jako podstawowej jednostki pracy. Ilość zdarzeń oczekujących do obsłużenia może być kryterium do reguł autoskalowania.

Metodologia budowania architektury w oparciu o zdarzenia ułatwia logikę obsługi błędów procesowania. Zdarzenia można w stosunkowo łatwy sposób przeprocesować ponownie lub dostosować mechanizmy do ponownej próby obsługi zdarzenia. W sytuacji, gdy zdarzenie nie może być prawidłowo przeprocesowane, może ono trafić do potoku procesowania niewłaściwych zdarzeń (ang. *dead letter queue*).

Metodyka przetwarzania zdarzeń została szeroko analizowana i wykorzystana korzystając z udogodnień tego typu podejścia w budowie luźno powiązanych komponentów. W przypadku systemu chmurowego procesowanie systemu determinowane jest zdarzeniami generowanymi przez wyzwalacze w wyniku interakcji użytkowników lub zdarzeń generowanych automatycznie takich jak umieszczenie pliku w pojemniku S3. W przypadku systemu rozproszonego wykorzystano brokera komunikacji w oparciu o model publikacja-subskrypcja, który wyzwalał dalsze procesowania w systemie.

3.1.3 Wzorce projektowe

Stosowanie wzorców projektowych pomaga rozwiązywać pospolite problemy przy budowie systemów informatycznych w sprawdzony i opisany sposób. Definiowanie wzorców projektowych jest istotne, aby specjaliści mogli się posługiwać zrozumiałą nomenklaturą przy opisie pewnych rozwiązań. Wzorce projektowe różnią się od algorytmów tym, że nie są ściśle zdefiniowane w postaci listy konkretnych instrukcji. Podają one ogólny opis napotkanego problemu technicznego i sposobu na jego rozwiązanie. Stąd implementacja tego samego wzorca projektowego może wyglądać inaczej w różnych systemach [33].

Wykorzystywanie metodologii wzorców projektowych jest korzystne dla budowy i rozwoju systemu. Strukturyzują one architekturę, wymianę komunikatów, niwelują redundancję kodu lub odpowiedzialności komponentów. Ponadto rozwiązania powszechnych problemów są dobrze udokumentowane i zalecane jest wykorzystywać ponownie sprawdzone rozwiązania. Kolejną zaletą stosowania wzorców projektowych jest fakt, że nowe osoby, które dołączają do zespołu programistów, mogą się szybciej odnaleźć w strukturze kodu.

Wzorce projektowe dzieli się na 3 kategorie [5]:

- wzorce kreacyjne;
- wzorce strukturalne;
- wzorce behawioralne.

Wzorce kreacyjne pozwalają na tworzenie nowych lub rozszerzanie instancji obiektów. Dostarcza to mechanizmów do ustrukturyzowania procesu wytwarzania obiektów i pozwala wykorzystać ponowie kod, co redukuje redundancję oraz duplikaty bloków kodu. Przykładami wzorców projektowych z tej kategorii są fabryka, budowniczy czy singleton.

Do zadań wzorców strukturalnych należy skomponowanie komponentów w dany sposób, który pozwoli ułatwić rozwiązanie problemu skomplikowanego. Główną zaletą stosowania wzorców z tej kategorii jest ład i porządek architektoniczny w systemie. Ponadto wzorce strukturalne pomagają w osiągnięciu integracji między systemami w wyniku ułatwienia definicji interfejsów z wykorzystaniem komponentu pośredniego. Przykładowymi wzorcami są fasada, adapter, dekorator czy też kompozyt.

Celem wzorców behawioralnych jest ustalenie odpowiedzialności, zakresu obowiązków i efektywnej komunikacji pomiędzy komponentami. Wzorcami należącymi do tej kategorii są łańcuch odpowiedzialności, obserwator, mediator, iterator czy też komenda. Zbiór popularnych wzorców projektowych każdego typu został ukazany na Rysunku 8.

Rysunek 8. Katalog podziału wzorców projektowych. Źródło: [33]

Prawidłowe stosowanie wzorców projektowych wymaga pewnej wprawy i doświadczenia. Standardowym problemem jest nadużywanie danego wzorca projektowego lub używanie go tam, gdzie nie był on konieczny. Powoduje to większe skomplikowanie systemu niż jest to wymagane. Nadużywanie wzorca singleton doprowadziło do nazywania go antywzorcem przez część środowiska specjalistów inżynierii oprogramowania [4, 5].

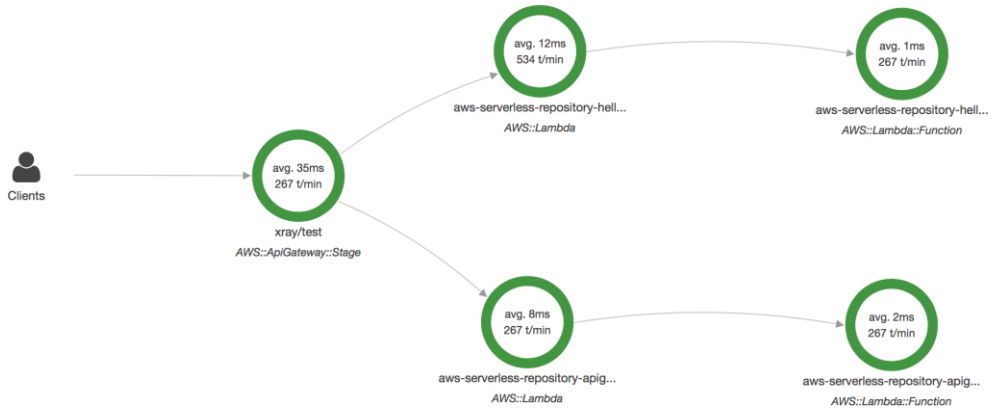
Poszczególne usługi chmurowe AWS implementują dane wzorce projektowe w celu gotowego wykomponowania w architekturę systemu rozwiązania opartego na wzorcu projektowym. Usługa API Gateway implementuje wzorzec fasady. Z kolei usługa AWS StepFunction realizuje wzorce łańcucha zobowiązań oraz stanu. Usługi monitoringu i alarmowania udostępnione przez AWS CloudWatch wpisują się we wzorzec obserwatora. Natomiast użycie funkcjonalności AWS XRay zrealizowane jest przez wzorzec projektowy dekorator.

3.1.4 Metodyki monitoringu oraz alarmowania

Monitorowanie funkcjonowania systemu jest krytyczne dla utrzymania ciągłości biznesu opartego na wykorzystaniu technologii informatycznej. Pozwala to na wczesne wykrywanie awarii co umożliwia reakcję i minimalizację strat wynikłych z niedostępności systemu. Istnieje wiele wskaźników, które definiują poziom zaawansowania monitoringu i oczekiwanego czasu reakcji na niepożądane zdarzenia. Takimi wskaźnikami są docelowy czas odzyskiwania (ang. *Recovery Time Objective*) oraz docelowy punkt odzyskiwania (ang. *Recovery Point Objective*). Wskaźniki te definiują strategię przywrócenia system do funkcjonowania po awarii. Stosowanie rozwiązań chmurowych pozwala opracować rozwiązania, które gwarantują RTO oraz RPO bliskie zeru ze względu na redundancję architektury oraz wersjonowanie danych wraz z tworzeniem kopii zapasowych w czasie rzeczywistym poprzez replikację [34]. Starty wynikłe z braku funkcjonowania systemu są szczególnie dotkliwe w krytycznych systemach takich jak obronność kraju, transport, służba zdrowia lub też systemach komercyjnych takie jak duże sklepy internetowe lub oferenci różnych usług sieciowych. Wpływa to bardzo poważnie na straty finansowe oraz wizerunkowe danego podmiotu.

Poza krytycznością monitorowania podstawowych funkcji systemu umożliwiających jego prosperowanie, odpowiedzialnością monitoringu jest również zaspokajanie potrzeb takich jak zbieranie danych logów aplikacyjnych umożliwiających różne analizy zachowania systemu, prowadzenie audytu dostępu, śledzenie wydajności systemu czy też analiza kluczowych z perspektywy biznesu wskaźników. Kluczowe wskaźniki nazywa się KPI (ang. *Key Performance Indicators*). Dla zdefiniowanych przez biznes wskaźników tworzy się kokpity administracyjne, które oferują wizualizacje zbieranych danych. Pomaga to w śledzeniu trendów, reakcji na zdarzenia oraz dostarcza narzędzia do tworzenia raportów okresowych.

Narzędzia monitoringu pomagają zidentyfikować, które operacje w procesie wykonują się szybciej lub wolnej niż pozostałe. Pomaga to zaplanować akcje wymagane do poprawy czasu reakcji aplikacji. Z oferty usług chmurowych AWS należy wyróżnić usługę AWS XRay, która oferuje pomiar czasu wykonania operacji na poziomie metody w kodzie źródłowym aplikacji. Ponadto generuje mapę interakcji i zależności danego procesu pomiędzy serwisami i komponentami co ukazuje Rysunek 9. Usługa ta dostarcza możliwość przekazywania specjalnego atrybutu, zwanego *traceId,* który ułatwia korelację podprocesów w obrębie procesu głównego. Jest to szczególnie istotne w wielowarstwowych systemach rozporoszonych, aby efektywnie monitorować proces od początku do końca poprzez wszystkie komponenty. Atrybut ten przekazywany jest w nagłówku zapytań protokołu HTTP.



Rysunek 9. Mapa wywołanych serwisów z użyciem AWS XRay. Źródło: [37]

Wykrycie anomalii w pracy systemu na podstawie metryk lub logów aplikacyjnych może skutkować podniesieniem alarmu. Zdefiniowanie procesu reakcji na podniesiony alarm jest bardzo istotne. Przykładowymi akcjami reakcji na zaistniały alarm może być wysłanie wiadomości email do administratorów systemu, przepięcie ruchu sieciowego na zapasową infrastrukturę (ang. *stand by infrastructure*), czy też odcięcie danego wyzwalacza procesu w celu minimalizowania negatywnych efektów powstałego błędu. Większe systemy informatyczne zatrudniają zespoły, które przez całą dobę dokonują monitoringu i są odpowiedzialni za reakcję na zaistniałe alarmy.

Popularnymi narzędziami do wdrożenia metodyki monitoringu są:

- Elasticsearch;
- AWS CloudWatch;
- Google Cloud Search;
- Apache Solr.

Rozwiązanie te oferują możliwość wyszukiwania interesujących danych w logach aplikacyjnych w dostarczonych deklaratywnych językach zapytań. Zoptymalizowane silniki przeszukiwań tych narzędzi są szczególnie istotne przy zapytaniach pełno tekstowych. Efektywność zapytań jest możliwa poprzez architekturę opartą na węzłach. W przypadku narzędzia Elasticsearch, mamy do czynienia z węzłami zarządców (ang. *master nodes*) oraz z węzłami przeznaczonymi na dane (ang. *data nodes*). Węzły podzielone są na mniejsze struktury zwane odłamkami (ang. *shards)*. Dane zapisywane są w logicznych strukturach zwanymi indeksami, które przechowują dokumenty [35]. Optymalne korzystanie z silnika zapytań wymaga podejścia atomowego definiowania wartości atrybutów. Preferowanym formatem danych wykorzystywanym w narzędziu Elasticsearch jest JSON. Narzędzia te oferują możliwości budowania wizualizacji i kokpitów administracyjnych. Podstawowym wymiarem analizy logów i monitoringu jest wymiar czasu. Narzędzia ta oferują przeszukiwanie w wyspecyfikowanym interwale czasu oraz dostarczają zagregowane informacje o ilości logów w danym przedziale. Bazy danych zorientowane na wymiar czasu (ang. *time-series)* dobrze spisują się w zastosowaniach monitoringowych danego parametru na przykład wartości odczytu temperatury z sensorów.

3.2. Charakterystyka wykorzystanych narzędzi

Efektywne osiąganie rezultatów pracy wymaga odpowiednich narzędzi. Różnorodność i rozmaitość dostępnych rozwiązań wymaga refleksji nad doborem zestawu technologii do pracy nad danym problemem. Przy doborze narzędzi należy rozważyć charakterystykę i dziedzinę problemu do rozwiązania. Istotna z punktu widzenia prowadzenia projektu jest także dostępność na rynku specjalistów w danej technologii i rzetelnej dokumentacji. Bardzo ważna jest także popularność wykorzystanego narzędzia co przekłada się na szybkie wyszukiwanie rozwiązań na popularnie napotykane problemy. Zalecane jest korzystanie z nowożytnych narzędzi informatycznych, których rozwój jest deklarowany i zapewniony w klauzuli zwanej LTS (ang. *long term support*). W osiągnięciu rezultatów pracy wykorzystano popularne i rekomendowane technologie, które są perspektywiczne z punktu widzenia budowy przyszłych systemów informatycznych.

3.2.1 Narzędzia deweloperskie oraz administracyjne dla usług chmurowych AWS

Budowa natywnych systemów chmurowych wymaga umiejętności posługiwania się narzędziami do obsługi i zarządzania usługami chmurowymi. Większość zadań administracyjnych może zostać wykonana w przeglądarkowej konsoli po uprzednim zalogowaniu. Konsola oferuje dostęp do wykorzystywanych zasobów i ich administrację z każdego miejsca na świecie z dostępem do internetu. Udostępnia ona prezentację graficzną i kokpity administracyjne dla usług udostępnionych przez AWS. Wygodnie można nawigować się pomiędzy dostępnymi regionami. Jest to bardzo wygodne narzędzie, którego użyteczność wzrasta wraz ze zdobywaniem doświadczenia w korzystaniu. Znajomość aspektów technologii informatyki pomaga w obsłudze tego narzędzia. Udostępniona przez producenta dokumentacja posiada liczne przykłady jak wykonać daną czynność administracyjną za pomocą konsoli. Jednakże narzędzie w postaci konsoli administracyjnej najlepiej sprawdza się w czynnościach administracji systemu. Przy budowie systemu narzędzie to jest niepraktycznie ze względu na dużą ilość wprowadzanych zmian wymagających wdrożenia i testowania.

Przy budowie systemów w oparciu o usługi chmurowe najlepiej sprawdza się skorzystanie z dostępnego SDK (ang. *Software Development Kit*) w danym języku programowania. W przypadku języka Python oferowana biblioteka nosi nazwę *boto3*. Oprogramowanie to dostarcza API do komunikacji ze wszystkimi dostępnymi usługami AWS. Dodatkowo możliwe jest napisanie testów jednostkowych w oparciu o biblioteki SDK, które umożliwiają stworzenie imitacji środowiska usług chmurowych (ang. *mocks*). Wykorzystywanie SDK jest zalecane przy implementacji logiki biznesowej funkcji AWS Lambda czy też programów uruchomionych w kontenerach usługi AWS ECS lub AWS EC2. Zasadność wykorzystania SDK to:

- dostęp do zasobów AWS poprzez API;
- machizmy wykładniczego ponownego zapytania (ang. *exponential backoff retry*) [38];
- współdzielenie sesji klienta w celu poprawy wydajności;
- dwa rodzaje klientów do komunikacji z zasobami oferujące uproszczony i rozszerzony dostęp;
- paginacja zwracanych wyników;
- dostarczone mechanizmy do autentykacji wykorzystujące AWS Sig4;
- rzetelna dokumentacja.

Alternatywą do konsoli oraz SDK jest wykorzystanie komend wiersza poleceń zwanych AWS CLI. Narzędzie to jest pomocne przy pisaniu skryptów administracyjnych lub automatyzacji procesów. Aby móc skorzystać z komend wiersza poleceń należy zainstalować oprogramowanie w rekomendowanej i rozwijanej drugiej wersji. Następnie należy utworzyć użytkownika w usłudze AWS IAM i pobrać dane uwierzytelniające. Do uwierzytelnienia wyróżnia się klucz dostępu (ang. *access key*), sekretny klucz (ang. *secret access key*) oraz opcjonalnie token sesji (ang. *session token*). Ponadto należy wyróżnić region dla którego będą wykonywane komendy oraz format zwracanych danych na przykład JSON. Listing 2 pokazuje proces konfiguracji domyślnego użytkownika wiersza poleceń AWS.

Listing 2. Konfiguracja profilu AWS CLI. Źródło: [39]

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

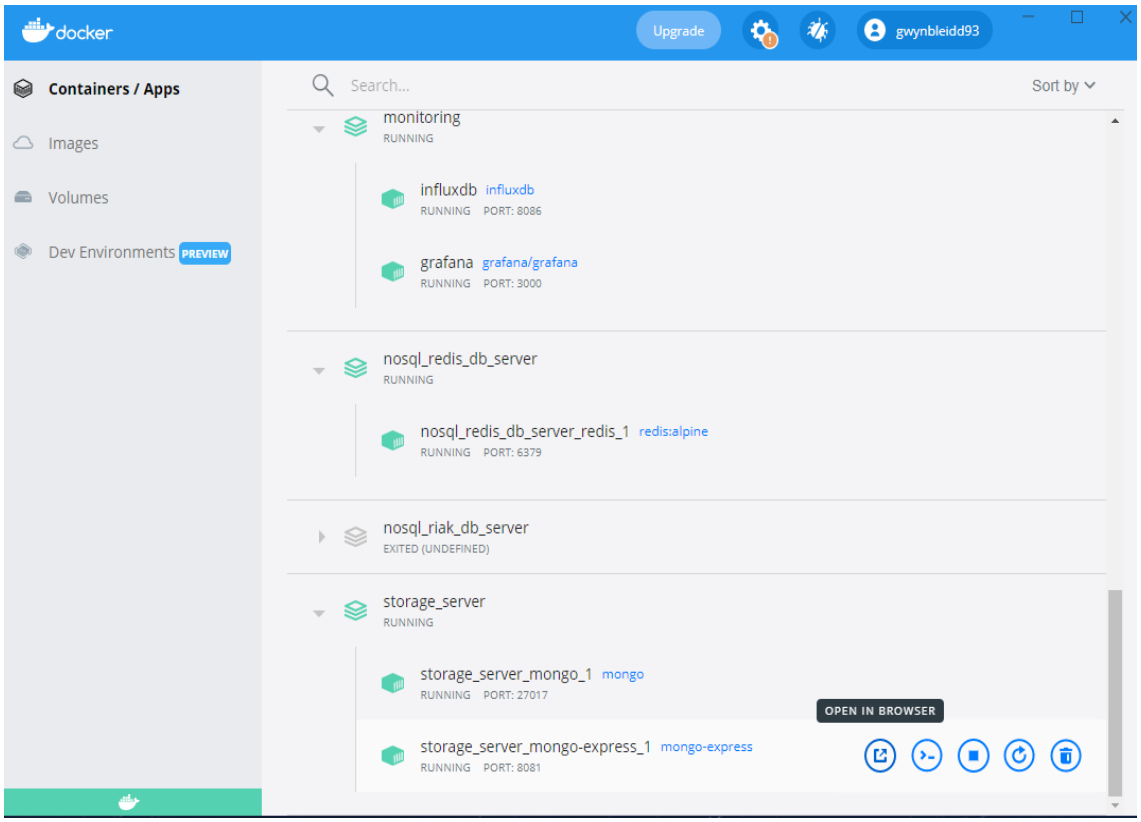
Pomocnymi usługami zorientowanymi na deweloperów są usługi takie jak AWS CodeCommit do wersjonowania plików czy też AWS CodePipeline do zdefiniowania automatycznego potoku wdrożenia nowej wersji oprogramowania. Usługi takie jak AWS CodeGuru pozwalają przeanalizować kod pod względem optymalizacji i stosowania najlepszych praktyk. Istotnym narzędziem do ponownego wykorzystania kodu jest AWS Lambda Layers. Pozwala to zdefiniować paczkę kodu do ponownego użytku, która może być wykorzystana przez wiele funkcji.

3.2.2 Narzędzia ekosystemu Docker

W celu tworzenia i uruchomiania skonteneryzowanych aplikacji wymagana jest konfiguracja wymaganych narzędzi ekosystemu Docker. Składa się na to instalacja środowiska, które zapewni uruchomienie procesu Docker Deamon na danym systemie operacyjnym. W realizacji pracy wykorzystano zasoby komputerowe wykorzystujące system operacyjny Windows. Producent oprogramowania Docker udostępnia aplikację nazwaną Docker Desktop for Windows, która oferuje zainstalowanie środowiska aplikacji kompatybilnego z systemem plików Windows, udostępnia dostęp do graficznego interfejsu użytkownika oraz dostęp do komend Dockera za pośrednictwem wiersza poleceń. Główny proces zarządzający, to jest Docker Deamon, jest uruchomiony dopóki aplikacja działa w tle. W aplikacji możliwy jest podgląd uruchomionych kontenerów, dostępnych obrazów czy też podpiętych wolumenów na dane. Dostępne opcje oferują wyświetlenie w przeglądarce skonteneryzowanych aplikacji, jeśli te udostępniają publiczne porty, na których te aplikacje działają. Jest to wygodne, aby przełączyć się na widok kokpitu administracyjnego danej aplikacji na przykład bazy danych MongoDB. Interfejs użytkownika aplikacji Docker Desktop został ukazany na Rysunku 10.

Aby skorzystać z gotowych, sprawdzonych obrazów należy przeszukać repozytoria obrazów kontenerów. Oficjalne repozytorium nazywa się Docker Hub. Po zalogowaniu się do domeny Dockera istnieje możliwość pobierania oraz załadowania swoich obrazów do repozytorium. Zaleca się korzystanie ze zweryfikowanych obrazów, gdyż często są one solidnie przetestowane pod kątem bezpieczeństwa i wydajności. Dobrą praktyką jest wskazanie konkretnej wersji obrazu do pobrania. Domyślnie będzie pobierany najnowszy obraz o nazwie tagu *latest*. Wystawia to aplikację na ryzyko

zmiany funkcjonalności w przypadku pojawiania się nowej wersji, która cechuje się znacznymi zmianami.



Rysunek 10. Kokpit aplikacji Docker Desktop. Źródło: opracowanie własne

Z ekosystemem Dockera bardzo wygodnie się pracuje za pośrednictwem wiersza poleceń. Dokumentacja komend jest rzetelna co ułatwia wyszukiwanie opcji konfiguracyjnych. Popularne komendy związane z pracą z kontenerami dotyczą budowania obrazu (*docker build*), uruchamianie instancji kontenera (*docker run*), sprawdzanie listy aktywnych kontenerów (*docker ps*), sprawdzenie lokalnych obrazów (*docker images)* czy też zatrzymanie instancji kontenera (*docker stop*).

Przy definiowaniu własnych obrazów kontenera wykorzystano pliki Dockerfile. W przypadku zbioru współpracujących kontenerów użyto funkcjonalności oferowane przez Docker Compose. Pozwala to na definicję wielu kontenerów w jednym pliku i umożliwia to ułatwioną integrację. Przypisanie wspólnej sieci oraz integrowanie aplikacji jest możliwe poprzez odniesienia do logicznych nazw zawartych w pliku definicji Docker Compose. Aby udostępnić sekretne wartości, na przykład hasła do baz danych, wykorzystano plik konfiguracyjny. Jest on wczytany do zmiennych środowiskowych adekwatnych kontenerów.

W celu uzyskania trwałości danych po ponownym uruchomieniu aplikacji wykorzystano mechanizm Docker Volumes. Pozwala to zapisywać dane generowane przez instancje kontenera do przestrzeni dyskowej hosta, a następnie wczytać te dane przy ponownym uruchomieniu. Jest to szczególnie istotne przy kontenerach bazodanowych, które przechowują dane i stan aplikacji.

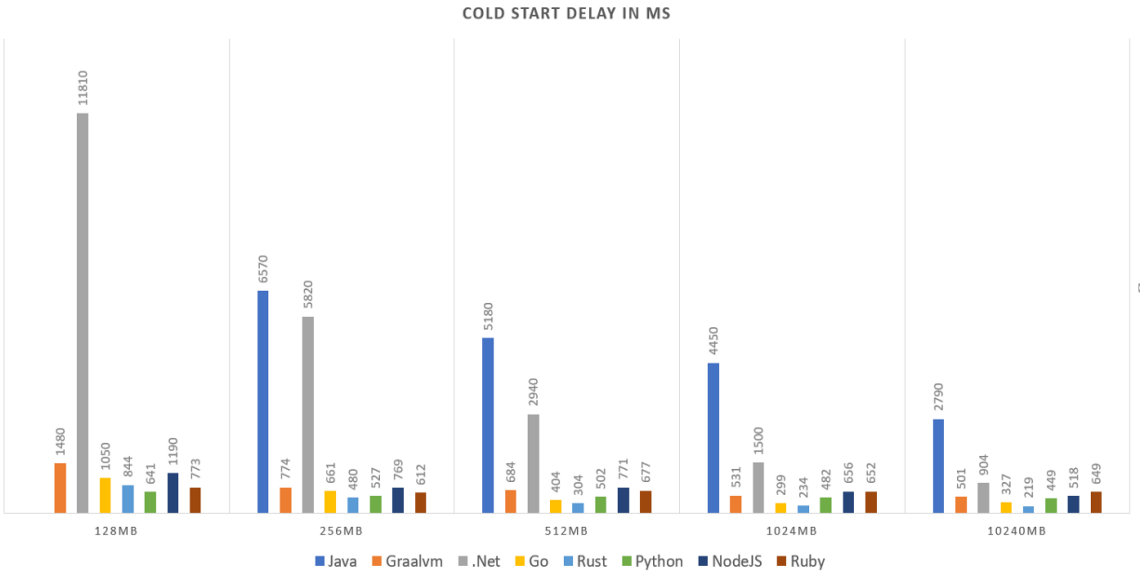
3.2.3 Język programowania Python

Podstawowym narzędziem do budowy systemów informatycznych jest język programowania. Pozwala on na zapisanie reguł biznesowych za pomocą algorytmu co umożliwia automatyzację procesów. Tak jak w przypadku innych narzędzi informatycznych, rodzina języków programowania jest szeroka i posiada różne charakterystyki i paradygmaty. Przy realizacji pracy wykorzystano język programowania Python w wersji 3.8. Motywacją do wyboru tego języka programowania jest:

- perspektywiczność i popularność na rynku pracy;
- wsparcie w usługach chmurowych AWS takich jak AWS Lambda;
- doświadczenie w posługiwaniu się tym językiem;
- wygoda tworzenia zarówno prostych skryptów jak i złożonych, modularnych aplikacji;
- wydajność oraz szybki czas uruchomienia programu;
- zwięzła i prosta semantyka języka zwiększająca czytelność programów;
- otwarte oprogramowanie;
- filozofia twórców języka preferująca proste rozwiązania nad skomplikowanymi.

Język programowania Python jest językiem wysokiego poziomu, który może zostać wykorzystany uniwersalnie w różnych dziedzinach. Według statystyk język Python stał się najczęściej wykorzystywanym językiem programowania w latach 2021/2022 [40]. Do tego stanu rzeczy przyczynił się dynamiczny rozwój dziedzin takich jak uczenie maszynowe czy analiza dużych zbiorów danych, w których to język Python umożliwia znaczne wsparcie dzięki wyspecjalizowanym bibliotekom.

Zaletą tego języka jest swoboda użycia. Nie wymusza on konkretnego stylu programowania. W rezultacie jest on przydatny do pisania krótkich skryptów automatyzujących codzienne czynności czy też implementowania dużych systemów. Oferuje on możliwość programowania strukturalnego, obiektowego lub funkcyjnego. Język Python jest językiem interpretowanym, co skutkuje szybszym uruchomianiem programu. Różnica w czasie uruchomienia jest wyraźnie widoczna przy analizie zimnych startów w usługach typu FaaS w porównaniu do języka wymagającego kompilacji takiego jak Java [41] co ukazuje Rysunek 11.



Rysunek 11. Wpływ wybranego języka programowania na zimny start funkcji. Źródło: [41]

Język Python wykorzystuje dynamiczne typowanie zmiennych. Wspiera popularne typy liczbowe całkowite i zmiennoprzecinkowe oraz struktury danych takie jak listy, słowniki, sety, krotki. Zagnieżdżenie instrukcji programu osiąga się poprzez wcięcia odpowiadające tabulacjom. Powstałe standardy takie jak PEP8 wymuszają utrzymanie schludnego kodu na przykład poprzez rekomendowaną maksymalną długość pojedynczej linii kodu. W rezultacie kod jest łatwy w czytaniu i utrzymaniu. Jest to pożądana cecha w związku z empirycznym truizmem mówiącym, że napisany kod czyta się kilkakrotnie razy częściej niż się go edytuje.

W realizacji pracy język ten został wykorzystany do implementacji logiki biznesowej w funkcjach AWS Lambda oraz w aplikacjach skonteneryzowanych. Do wystawienia API w systemie skonteneryzowanym wykorzystano bibliotekę Flask, która udostępnia możliwość tworzenia punktów końcowych systemu typu REST. Ponadto język ten został wykorzystany do napisania skryptów automatyzujących procesy pomocnicze takie jak generowanie danych testowych do systemu.

Do efektywnej pracy skorzystano ze zintegrowanego środowiska programistycznego PyCharm. Jest to produkt firmy JetBrains, czyli twórców IntelliJ IDEA. Program PyCharm jest zoptymalizowany pod język Python. Praca z IDE ułatwia zarządzanie projektem, wyszukiwanie w kodzie, refaktoring czy też testowanie.

3.2.4 Silniki bazodanowe

Systemy informatyczne przetrzymują stan i dane w nieulotnych instancjach baz danych. Ciągłość dostępu, trwałość i spójność danych oraz czasu dostępu to kluczowe aspekty stawiane przed silnikami bazodanowymi. Rodzaje baz danych można scharakteryzować za pomocą tak zwanej teorii CAP (ang. *Consistency, Availability, Partition Tolerance*) [42]. Teoria ta prezentuje 3 pożądane cechy baz danych takie jak spójność danych, wysoka dostępność oraz odporność na awarie poprzez zastosowanie partycji. W praktyce silniki bazodanowe mogą posiadać maksymalnie dwie z trzech wymienionych cech, gdyż optymalizacje pod względem wysokiej dostępności przypłaca się mniejszą spójnością danych i odwrotnie. Sprawdzone i najbardziej popularne na rynku relacyjne bazy danych wpisują się w typ baz danych CA (ang. *Consistency, Availablility*). Natomiast bazy danych noSQL takie jak AWS DynamoDB czy MongoDB są typem baz danych AP (ang. *Availability, Partition Tolerance*).

Bazy danych typu noSQL zyskały popularność ze względów wydajnościowych i potrzeby lepszej tolerancji awarii. Impulsem do rozwoju różnych rodzajów baz noSQL była pilna potrzeba rozwiązania problemów wydajnościowych w związku z dużą ilością danych do przetwarzania przez firmy takie jak Google, Facebook czy Twitter w pierwszym dziesięcioleciu XXI w. [43].

W realizacji pracy skorzystano z silników bazodanowych noSQL takich jak:

- AWS DynamoDB;
- MongoDB;
- RedisDB;
- InfluxDB;
- AWS Timestream.

Bazy danych takie jak AWS DynamoDB oraz RedisDB są bazami typu klucz-wartość. Pod unikalnym, haszowanym kluczem wyszukiwania znajduje się rekord w postaci danej struktury danych. Schemat rekordu i zbiór atrybutów może różnić się pomiędzy kluczami. Bazy te są zoptymalizowane pod względem wyszukiwania i skalowania. Umożliwiają dostęp do rekordu danych w czasie liczącym kilka milisekund po zastosowaniu akceleratorów lub kilkadziesiąt milisekund w standardowym zapytaniu po kluczu. Tego typu silniki bazodanowe z powodzeniem można stosować w aplikacjach czasu rzeczywistego takie jak gry online lub jako pamięć podręczna. Możliwa jest łatwa replikacja na inne partycje przez co zwiększona jest trwałość i niezawodność danych. W bazie danych DynamoDB osiąga się to poprzez skonfigurowanie globalnej tabeli pomiędzy regionami.

Baza danych MongoDB to dokumentowa nierelacyjna baza danych. Przechowuje rekordy w postaci dokumentów w formacie JSON pod wygenerowanym identyfikatorem. Umożliwia stworzenie rozproszonej struktury przez co zwiększa się tolerancja na awarie. MongoDB oferuje deklaratywny język zapytań oraz narzędzia administracyjne takie jak Mongo Express czy komendy wiersza poleceń. Warto zaznaczyć, że domyślnie serwer MongoDB nie wymaga stworzenia użytkownika i hasła co wymaga szczególnej uwagi, aby nie udostępnić publicznie danych znajdujących się w niezabezpieczonej bazie danych.

Bazy danych InfluxDB oraz AWS Timestream to bazy nierelacyjne typu *time-series*. Bazy te są zorientowane na śledzenie zmiany wartości danego aspektu względem wymiaru czasu. Użycie tych silników bazodanowych dobrze się wpisuje w zastosowaniach telemetrycznych. Przykładem może być śledzenie zmiany temperatury na stacji meteorologicznej z danego czujnika. Bazy danych tego typu nie wpisują się całkowicie w ani w model CA, ani w AP. W konkretnych aspektach takich jak tworzenie klastrów czy obsługa zapisów bazy te korzystają po części z obu tych podejść [44].

4. Porównanie procesu budowy systemu chmurowego i skonteneryzowanego

Nauka nowych technologii oraz sprawdzanie ich możliwości jest efektywne w podejściu prototypowania (ang. *prove of concept*). W tym sposobie należy określić główne problemy, które należy rozwiązać, a następnie rozpocząć eksplorację dostępnych możliwości do osiągnięcia celu. Aby wyszukać wartościowe źródła wiedzy technicznej można wykorzystać zasoby Internetu oraz dokonać oceny jakości źródeł. Najlepiej użyć oficjalnej dokumentacji danego produktu, artykuły naukowe lub rekomendowane blogi technologiczne. Przegląd materiałów pozwala w sposób wyedukowany określić kierunek rozwoju prototypu.

W celu zdobycia wiedzy i poszerzenia kompetencji praktycznych w dziedzinie problemowej niniejszej pracy, zbudowane zostały dwa prototypy systemów informatycznych. Realizują one bardzo zbliżony zakres funkcjonalności biznesowych, jednakże zostały wykonane w różnych podejściach technologicznych. Pierwszy prototyp wykorzystuje usługi chmurowe w ekosystemie AWS. Natomiast drugie rozwiązanie opiera się na wykorzystaniu konteneryzacji z użyciem platformy Docker.

4.1 Opis wymagań biznesowych systemu

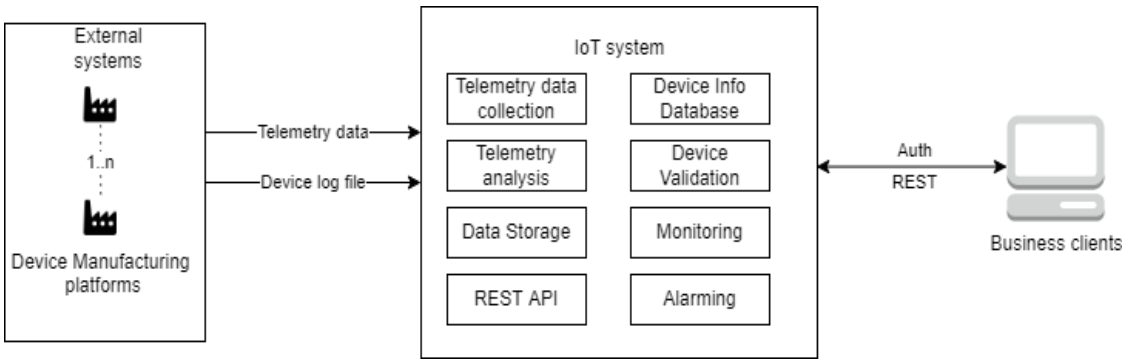
Główną funkcjonalnością zbudowanych prototypów jest zbieranie i przetwarzanie danych telemetrycznych z czujników podczas procesu wytwórczego danych urządzeń elektronicznych. Produkcja urządzeń jest oddelegowana do zewnętrznego dostawcy co wymusza integrację pomiędzy niezależnymi systemami. Uruchomienie nowego rozwiązania ma zapewnić realizację następujących celów biznesowych:

- monitoring procesu wytwórczego;
- inwentaryzacja urządzeń;
- wykrywanie anomalii w odczytach czujników;
- reakcja w czasie rzeczywistym na niebezpieczne wartości odczytów;
- kontrola jakości;
- generowanie raportów;
- aktualizacja oprogramowania układowego;
- budowanie wiedzy o produkcie i planowanie strategiczne.

Wymagania niefunkcjonalne stawiane przed systemem to:

- skalowalność;
- wysoka wydajność;
- wysoka dostępność;
- odporność na awarie;

- oszczędność;
- dostęp do kokpitu administracyjnego;
- ciągłość dostarczania oprogramowania i integracji.



Rysunek 12. Wysokopoziomowa architektura systemu. Źródło: opracowanie własne

Architektura i wymagania biznesowe na wysokim poziomie abstrakcji zostały pokazane na Rysunku 12. Realizacja tych wymagań została osiągnięta poprzez dobór odpowiednich usług, metodyk, technik architektonicznych oraz implementacji reguł biznesowych z wykorzystaniem języka programowania.

4.2 Budowa generatora danych telemetrycznych

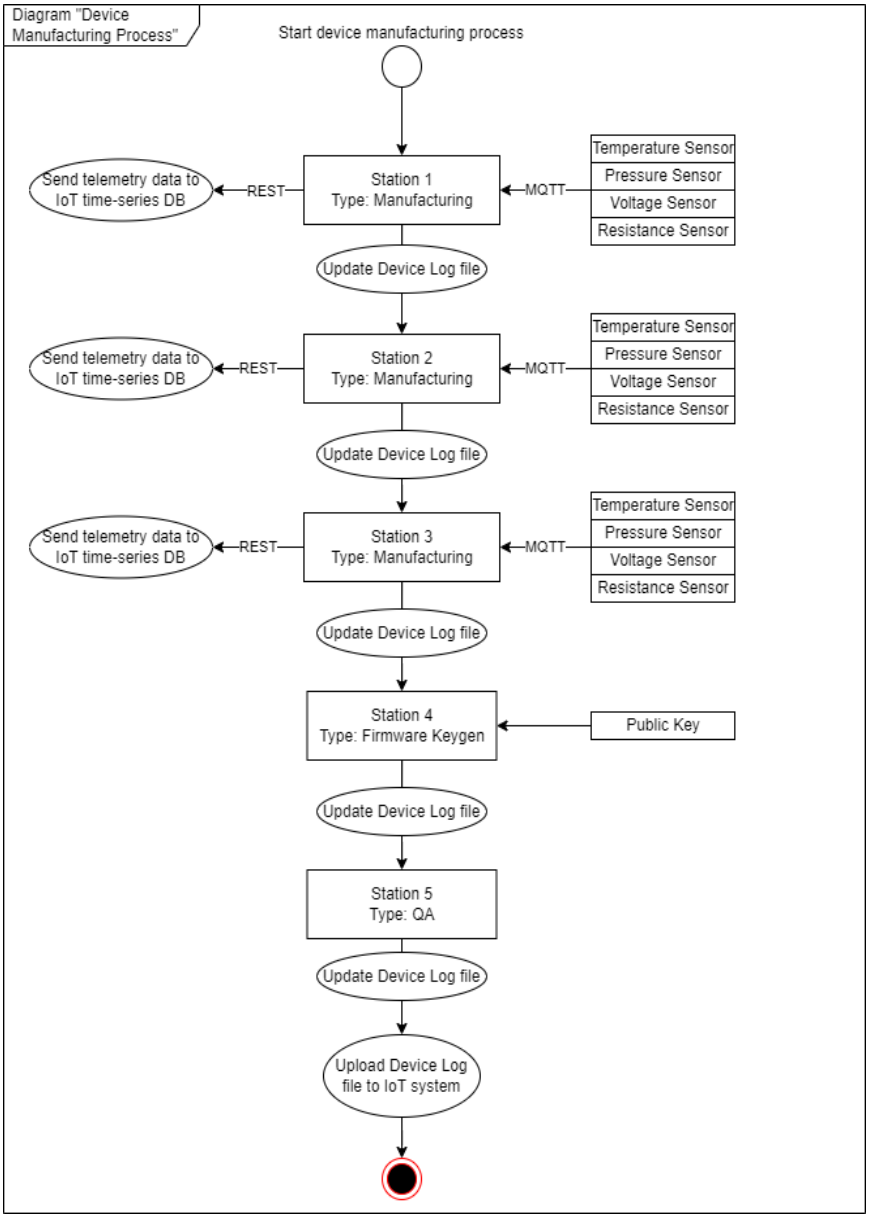
Zbudowany system zarządza i monitoruje procesem wytwarzania urządzeń elektronicznych. W przyjętej architekturze proces produkcji odbywa się w zewnętrznym systemie. Ten z kolei jest zintegrowany z główną platformą IoT. Zadaniem, które realizuje zewnętrzny komponent zwany *Device Manufacturing Platform*, jest generacja danych telemetrycznych co symuluje odczyt pomierzonych wartości z czujników umieszczonych na urządzeniu i taśmie produkcyjnej. Ponadto wytwarzany jest plik logu produkcyjnego w formacie JSON, który przechowuje wszystkie dane związane z procesem wytwarzania urządzenia.

Wyróżnione zostały 4 wartości monitorowane przez czujniki. Są to odczyty temperatury, ciśnienia atmosferycznego, napięcia i oporu elektrycznego. Proces składania urządzenia odbywa się na trzech stacjach roboczych zwanych *Manufacturing Station*. Na tym etapie produkcji dokonywane są odczyty z czujników w równym interwale czasu wynoszącym jedną sekundę. Dane telemetryczne wysyłane są za pośrednictwem protokołu MQTT do brokera, a następnie zapisywane w logu produkcyjnym oraz załadowane do instancji bazy danych typu *time-series* w celu wizualizacji i monitoringu.

Kolejnym etapem po złożeniu urządzenia na stacjach roboczych jest wygenerowanie symetrycznego klucza kryptograficznego, który następnie zostanie zaszyfrowany z wykorzystaniem klucza publicznego stosując techniki szyfrowania asymetrycznego. Unikalny klucz dla danego urządzenia zostaje zapisany w logu produkcyjnym. Celem tego zabiegu jest umożliwienie funkcjonalności zdalnego zaktualizowania oprogramowania układowego. Wgranie oprogramowania

będzie wymagało odszyfrowania plików z użyciem klucza symetrycznego co ma zapobiec nieuprawnionym próbom modyfikacji oprogramowania.

Ostatnim etapem jest kontrola jakości wytworzonego urządzenia. W rezultacie dokonywana jest walidacja wartości odczytów czujników pod kątem zmieszczenia się w zdefiniowanych dopuszczalnych wartościach. Prawdopodobieństwo uzyskania wartości odstających ustalono w postaci parametru i domyślnie wynosi 1% spośród dokonanych pomiarów. W tym celu wykorzystana została funkcja randomizacji. Ponadto weryfikowana jest poprawność wygenerowanego klucza. Wpis z wynikiem kontroli jest zapisywany w logu produkcyjnym. W tym momencie proces produkcji urządzenia jest ukończony. Nowy utwór schodzi z taśmy produkcyjnej, a plik logu produkcyjnego zostaje zapisywany w systemie klienta. Rysunek 13 ukazuje schemat blokowy procesu wytwarzania urządzenia.



Rysunek 13. Diagram procesu wytwórczego urządzenia. Źródło: opracowanie własne

Komponent *Device Manufacturing Platfrom* został zaimplementowany w języku Python. Jest on gotowy do użycia zarówno dla systemu chmurowego i skonteneryzowanego, ze względu na napisanie kodu umożliwiającego połącznie z obydwoma platformami. Wybór konkretnej integracji w danym momencie jest sterowane ustawieniem zmiennej środowiskowej. W przypadku integracji z platformą chmurową wykonana jest integracja z usługą AWS S3, aby załadować plik produkcyjny urządzenia. Dane telemetryczne są ładowane do usługi AWS Timestream. W rozwiązaniu używającym skonteneryzowane komponenty, plik produkcyjny jest zapisywany w bazie dokumentowej MongoDB, a wartości pomiarowe trafiają do bazy danych typu *time-series* o nazwie InfluxDB.

Do umożliwienia komunikacji z użyciem protokołu MQTT posłużył broker połączeń RabbitMQ. Do wygodnego wdrożenia aplikacji oraz brokera MQTT zdefiniowano plik Docker Compose. Wykorzystano funkcjonalności Docker Volumes w celu przechowywania i dostępu do wygenerowanych logów produkcyjnych urządzeń, aby mieć do nich dostęp po ponownym uruchomieniu aplikacji.

Komponent ten może być skalowalny z użyciem wbudowanej opcji skalowania serwisów Dockera co ukazuje Listing 3. Może to symulować integrację wielu niezależnych systemów zewnętrznych różnych producentów. Ponadto umożliwia to wygenerowanie większego ruchu sieciowego oraz ilości przesyłanych danych.

Listing 3. Konfiguracja skalowania procesu generacji danych telemetrycznych. Źródło: opracowanie własne

```
$ docker-compose build
$ docker-compose up --scale log-gen-service=10
```

W logu produkcyjnym urządzenia zapisywany jest unikalny identyfikator urządzenia, nazwa producenta, klucz kryptograficzny oraz status kontroli jakości. Ponadto zawarte są wartości odczytów z czujników na stacjach roboczych z uwzględnieniem czasu pomiaru, jednostką pomiarową i identyfikatorem czujnika.

Wysyłane dane telemetryczne charakteryzują się zwięzłym formatem, aby umożliwić szybką transmisję danych. Modele oraz przykładowe wiadomości komunikatów telemetrii wraz z plikiem logu produkcyjnego zawarte są w Dodatku B.

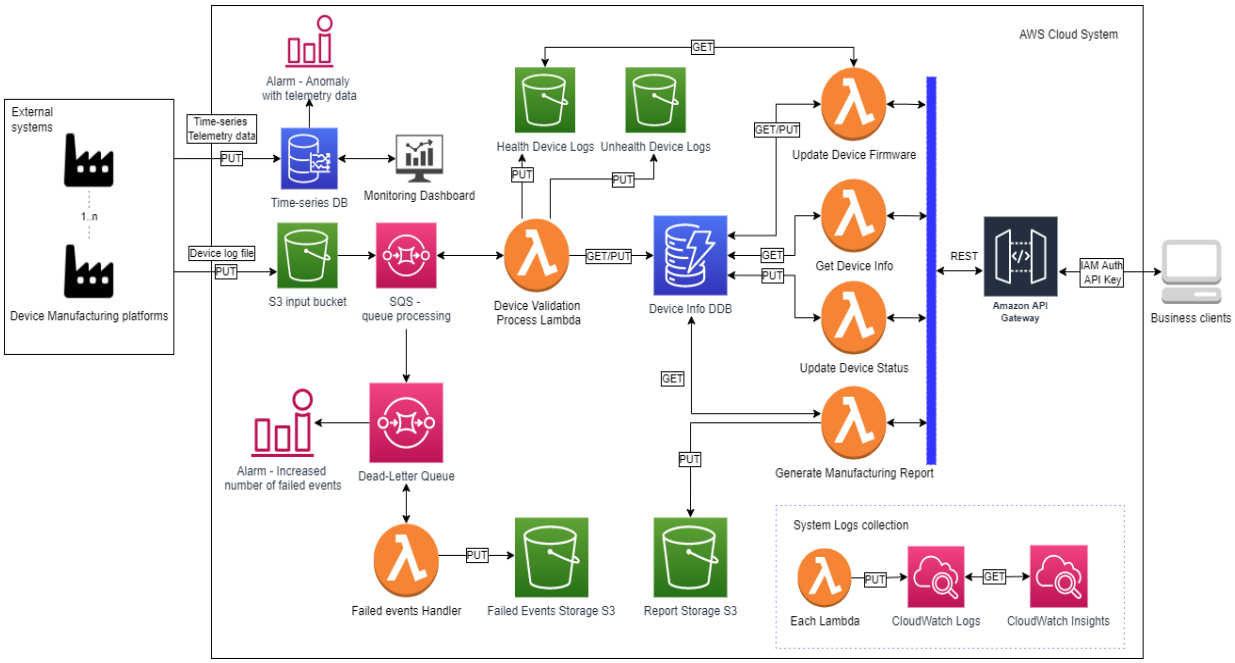
4.3 Budowa chmurowego systemu IoT z wykorzystaniem usług AWS

Prototyp systemu chmurowego został zbudowany w oparciu o natywne serwisy oferowane przez AWS z dziedziny usług *Serverless*. Do implementacji reguł biznesowych w funkcjach Lambda posłużył język Python w wersji 3.8. Realizacja wymagań biznesowych została osiągnięta poprzez odpowiedni dobór usług wraz z ich konfiguracją i wzajemną integracją. Pozwoliło to uzyskać wydajny, skalowalny i wysoko dostępny system z możliwością monitoringu. Ponadto możliwe jest wykonywanie zadań analitycznych na zgromadzonych logach aplikacyjnych w celu tworzenia raportów. Rozwój i utrzymanie systemu jest wsparte poprzez zdefiniowanie procesu CI/CD co ułatwia wdrożenie nowych zmian i potencjalną równoległą pracę wielu programistów. Kluczowymi elementami wchodzącymi w skład architektury dostarczania nowej wersji oprogramowania są

repozytorium kodu źródłowego, skrypty przygotowujące paczki wdrożeniowe funkcji Lambda, mechanizmy interpretujące skrypt definiujące infrastrukturę oraz automatyczny potok wdrożenia.

4.3.1 Architektura prototypu systemu chmurowego

Po zdefiniowaniu wymagań biznesowych systemu należało zaprojektować architekturę, która pozwoliła zrealizować stawiane cele. Składa się na to zdefiniowanie rodzajów usług, zależności pomiędzy komponentami, sposobów integracji oraz wymiany komunikatów. Należało mieć również na uwadze zasady dobrych praktyk takie jak pojedyncza odpowiedzialność komponentów, zdefiniowanie interfejsów oraz unikanie niepotrzebnych zależności, aby powstające rozwiązanie cechowało się odpowiednią jakością. Budowanie systemu z użyciem usług AWS oferuje znaczące ułatwienie w integracji usług poprzez wykorzystywanie modelu procesowania w oparciu o zdarzenia. Stąd integracja usług sprowadza się do konfiguracji zawartej w skrypcie CloudFormation, a przepływy biznesowe do potoku zdarzeń pomiędzy elementami systemu. Wykonany diagram architektury został stworzony w oparciu o analizę wymagań i zdobytą wiedzę na temat technik architektonicznych w ekosystemie AWS co przedstawia Rysunek 14.



Rysunek 14. Diagram architektury systemu chmurowego AWS. Źródło: opracowanie własne

Na architekturę systemu składają się następujące usługi AWS: S3, SQS, Lambda, CloudWatch, Dynamo DB, API Gateway, Timestream DB, IAM. Głównym regionem wdrożenia infrastruktury jest Irlandia (eu-west-1). Do przechowywania logów produkcyjnych urządzeń generowanych przez zewnętrzny system wykorzystano serwis S3. Wykonanie reguł logiki biznesowej zostało zaimplementowane w funkcjach Lambda. Aby uzyskać efektywne skalowanie, regulację procesowania oraz obsługiwanie błędnych wiadomości wykorzystano usługę SQS. Informacje o urządzeniach zostały przechowywane w nierelacyjnej bazie danych Dynamo DB. Komunikacja z platformą jest możliwa za pośrednictwem wystawionych punktów końcowych z użyciem fasady API Gateway. Przechowywanie logów aplikacyjnych, monitoring, definicja alarmów oraz wykonywanie zapytań do generacji raportów uzyskano z wykorzystaniem usługi CloudWatch. Autentykacja i

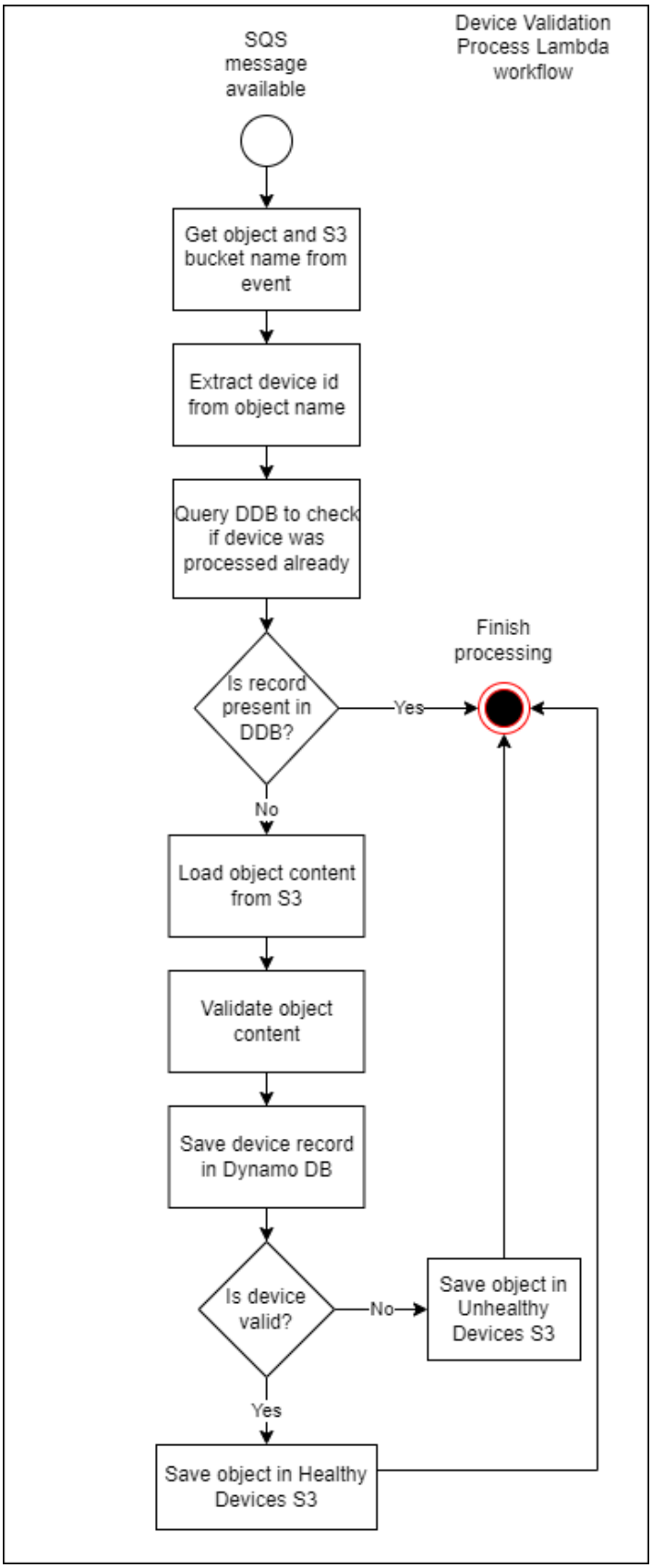
autoryzacja wraz z ziarnistymi pozwoleniami została dostarczona poprzez usługę IAM. Dane telemetryczne zostały przechowane i wizualizowane w bazie danych typu *time-series* o nazwie Timestream DB.

Integracja z zewnętrznym systemem została wykonana poprzez stworzenie dedykowanego technicznego użytkownika wykorzystując usługę IAM Users. Użytkownik ten ma ziarniste pozwolenia do załadowania danych do wyspecyfikowanych zasobów. Tymi zasobami są S3 *Input Bucket* oraz *Time-series DB.* Danymi do autentykacji są klucze AWS CLI. Zewnętrzny system wykorzystując przekazane poświadczenia oraz bibliotekę AWS SDK o nazwie *boto3* jest w stanie wywołać poprawnie API do usług S3 i Timestream. Fakt załadowania danych z zewnętrznego systemu do chmurowej platformy IoT rozpoczyna procesowanie zgodnie ze zdefiniowaną obsługą zdarzeń.

Klienci biznesowi mogą użyć dostępnych API do komunikacji z systemem. W tym celu muszą dokonać autentykacji w oparciu o podanie specjalnego nagłówka żądania http zwanego *AWS Signature version 4* [48] oraz klucza API. Definicja interfejsów jest udokumentowana w postaci pliku *swagger* wchodzącego w standard OpenAPI. Testowanie punktów końcowych zostało wykonane przy użyciu programu Postman.

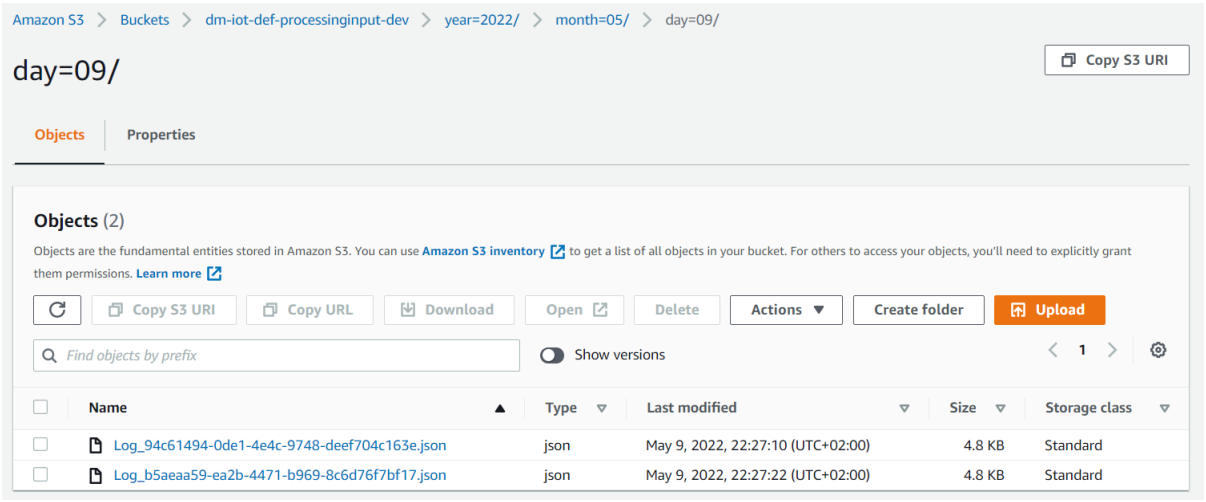
4.3.2 Procesowanie pliku produkcyjnego urządzenia

Jedną z kluczowych funkcjonalności systemu jest procesowanie pliku logu produkcyjnego urządzenia. Proces ten ma zapewnić dodanie urządzenia do bazy danych w celach inwentaryzacyjnych. Wpis w bazie danych dla danego urządzania jest niezbędny do przeprowadzania dalszych operacji. Ponadto ponownie jest przeprowadzana walidacja odczytów czujników w procesie produkcji, niezależnie od kontroli wykonanej przez producenta. Plik logu produkcyjnego jest wymagany także w zamodelowanej funkcjonalności zdalnego zaktualizowania oprogramowania układowego, gdyż przechowuje on zaszyfrowany klucz symetryczny. Rysunek 15 przedstawia diagram zawierający logikę biznesową funkcji odpowiedzialnej za procesowanie pliku logu urządzenia.

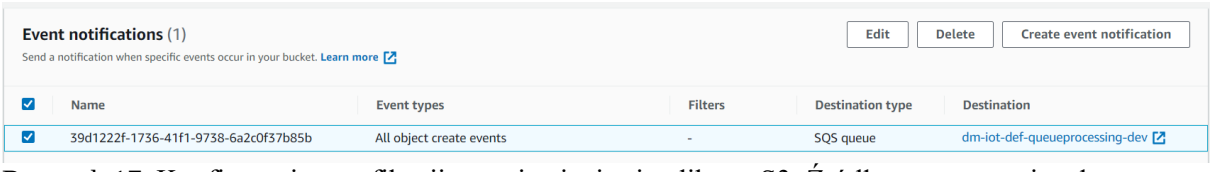


Rysunek 15. Diagram logiki biznesowej funkcji AWS Lambda *Device Validation Process*. Źródło: opracowanie własne

Zewnętrzny system dokonuje załadowania gotowego pliku do S3 *Input bucket* co ukazuje Rysunek 16 dla przykładowych urządzeń. Konfiguracja pojemnika S3 umożliwia wygenerowanie notyfikacji w momencie zdarzenia załadowania pliku. Przykład tej konfiguracji został ukazany na Rysunku 17. Komunikat ten trafia do odbiorcy, którym jest kolejka SQS. Opcje te nazywają się *S3 Event Notification.* Następnie wiadomości z kolejki są pobierane przez funkcję Lambda, która dokonuje właściwego procesowania.

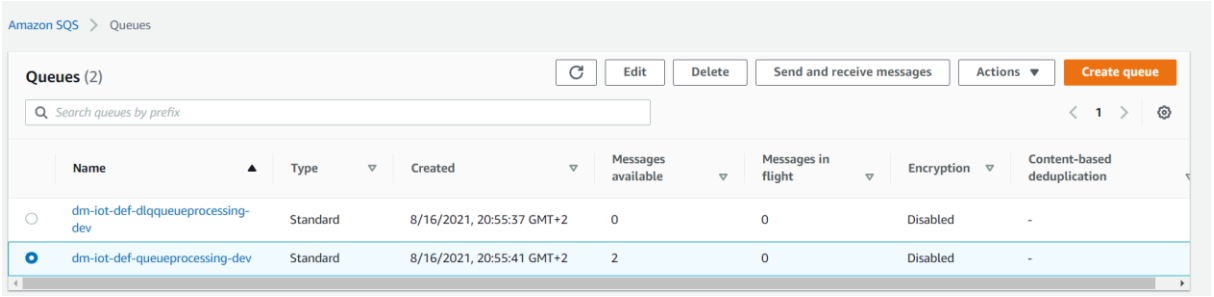


Rysunek 16. Pliki logów produkcyjnych urządzeń w pojemniku S3. Źródło: opracowanie własne



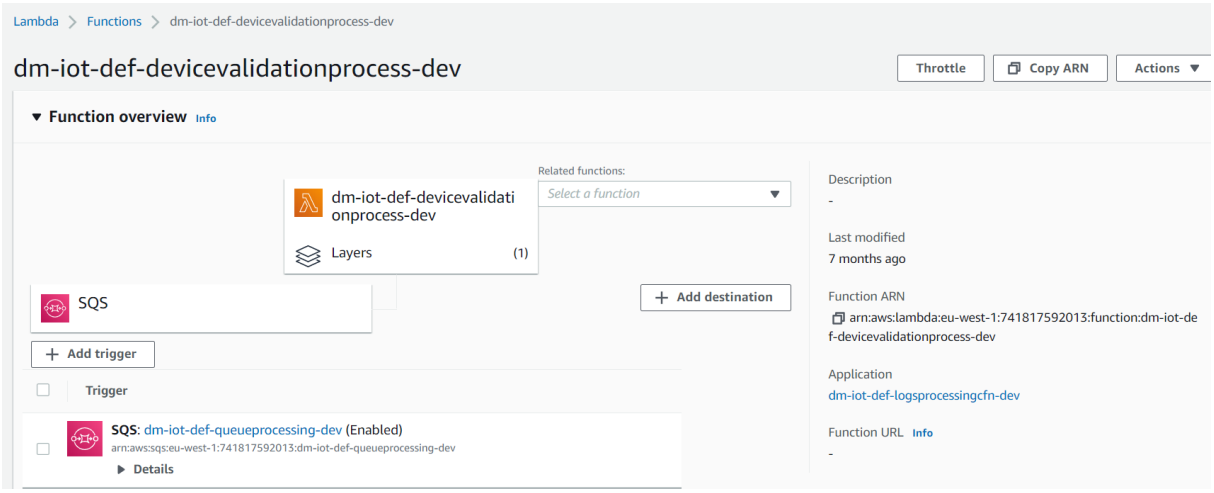
Rysunek 17. Konfiguracja notyfikacji na pojawienie się pliku w S3. Źródło: opracowanie własne

Wykorzystanie pośredniczącej kolejki pomiędzy pojemnikiem S3, a funkcją Lambda ma kilka istotnych powodów. W ten sposób proces jest bardziej efektywnie skalowany poprzez uruchomienie odpowiedniej liczby instancji funkcji Lambda w zależności od liczby wiadomości oczekujących w kolejce procesowania. Użycie SQS pozwala odrzucać nieprawidłowe wiadomości (ang. *poison pill messages*) w momencie gdy dany komunikat nie jest w stanie zostać przeprocesowany poprawnie przez zadaną ilość razy. Wtedy taka wiadomość jest przekierowywana do specjalnej kolejki oczekującej na nieprawidłowe wiadomości (ang. *dead letter queue*). Ponadto kolejka pozwala na ponowienie próby procesowania w przypadku chwilowej niedostępności mocy obliczeniowej funkcji Lambda (ang. *throttling*). Kolejną właściwością jest możliwość zarządzania oknem czasowym procesowania. W specjalnych przypadkach takich jak przerwa techniczna możliwe jest odłączenie wyzwalacza lambdy od kolejki, która w dalszym ciągu gromadzi wiadomości do procesowania. Uzyskanie tego efektu nie byłoby możliwe przy bezpośredniej integracji pomiędzy S3 i funkcją Lambda. Lista wykorzystanych kolejek w zbudowanym prototypie została ukazana na Rysunku 18.



Rysunek 18. Lista wykorzystanych kolejek SQS. Źródło: opracowanie własne

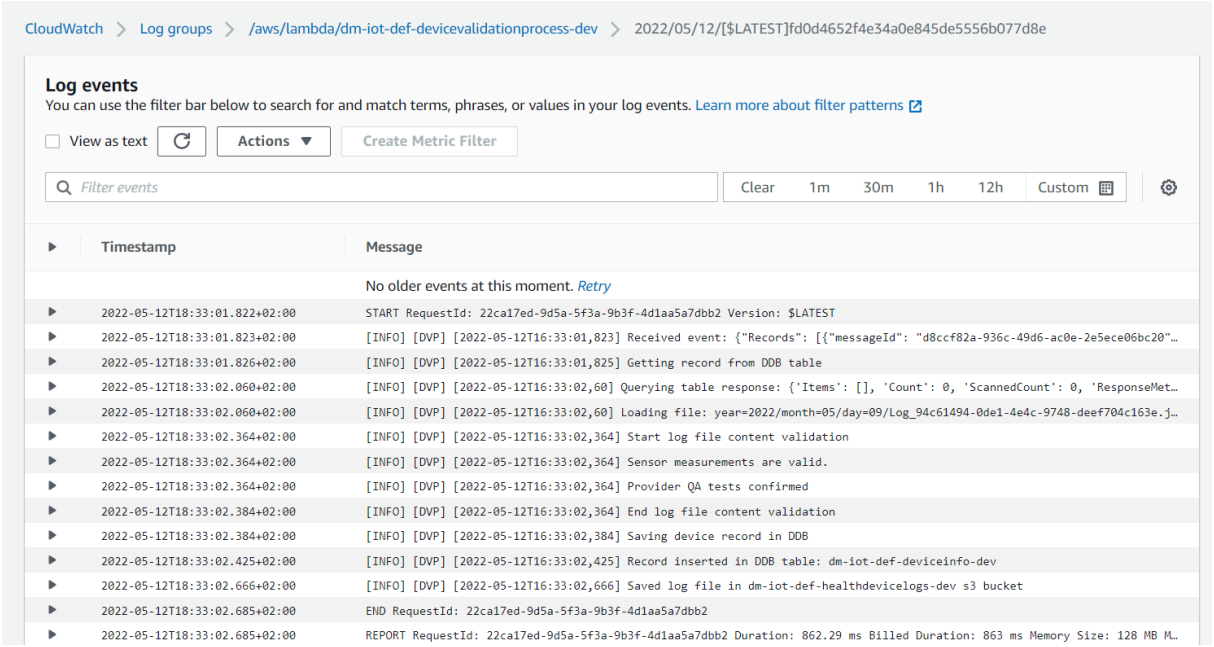
Notyfikacje wysyłane przez usługę S3 są typu *push*, natomiast komunikaty przechowywane w SQS są typu *pull*. Oznacza to, że serwis S3 po wgraniu pliku zawsze wyśle od razu komunikat i umieści go w miejscu docelowym. Natomiast wiadomości z SQS muszą zostać wczytane przez konkretnego konsumenta. Dlatego kolejka oferuje dłuższy czas oczekiwania na dostępność mocy obliczeniowej i większą tolerancję na błędy. Maksymalny czas przechowywania komunikatów w kolejce SQS to 14 dni. Z danej kolejki wiadomości pobierane się tylko przez jednego konsumenta. Oznacza to, że nie jest możliwe aby dwie różne funkcje Lambda były podpięte do tej samej kolejki SQS. Aby zrealizować taką architekturę gdzie jeden producent ma wielu niezależnych konsumentów (ang. *fanout*) należy użyć usługi SNS.



Rysunek 19. Funkcja Lambda *Device Validation Process* z wyzwalaczem jako kolejka SQS. Źródło: opracowanie własne

Funkcja Lambda nazwana *Device Validation Process* wczytuje aktywnie wiadomości z kolejki SQS w momencie zdefiniowania wyzwalacza (ang. *trigger*) co zostało pokazane na Rysunku 19. W konfiguracji funkcji można zdefiniować ilość wiadomości do pobrania co umożliwia procesowanie seryjne (ang. *batch processing*). Parametrami wejściowymi do funkcji Lambda są obiekty nazwane *event* oraz *context*. W tych obiektach dostępne są informacje takie jak identyfikator żądania, czas pojawienia się wiadomości, zawartość ciała notyfikacji, nagłówki. W przypadku komunikatu wygenerowanego przez S3 dostępna jest informacja o nazwie pojemnika S3 oraz pliku, który wywołał procesowanie. Umożliwia to pobranie i wczytanie pliku w procesie funkcji Lambda. Atrybuty procesowania są zapisywane w obiekcie pomocniczej klasy na dane (ang. *data transfer object*).

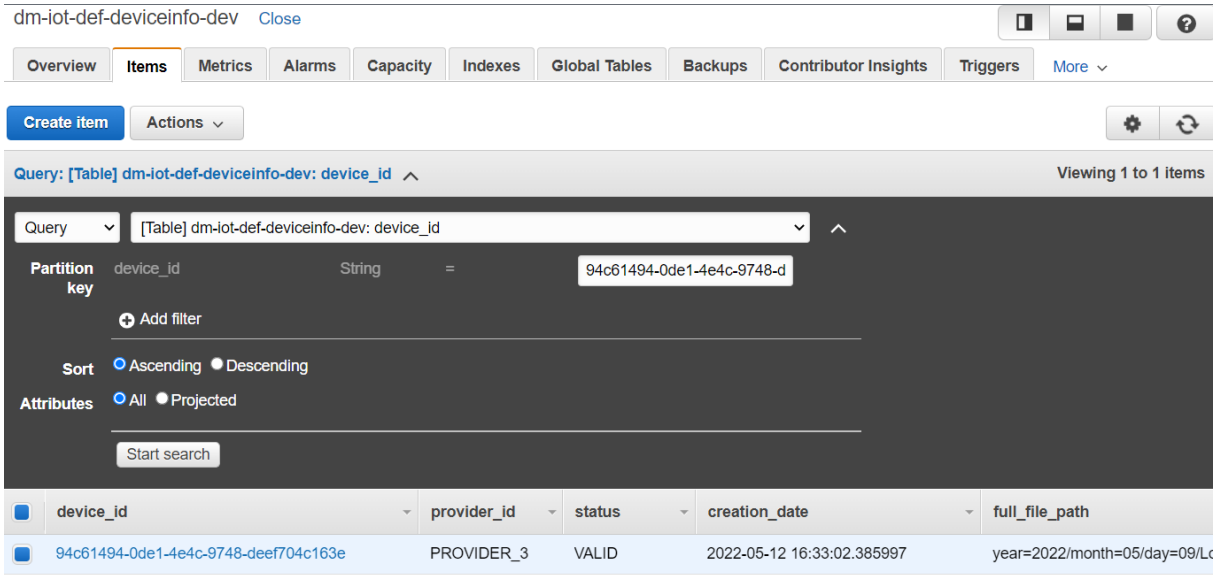
Na podstawie nazwy pliku zostaje wyekstrahowany identyfikator urządzenia. Następnie odpytywana jest baza danych Dynamo DB o nazwie *Device Info*, aby sprawdzić czy dane urządzenie zostało już przeprocesowane. Jak pokazano na Rysunku 21 kluczem głównym tej bazy danych jest identyfikator urządzenia (ang. *device id*). Procesowanie zostanie zakończone, jeśli rekord w bazie danych już istnieje. W przeciwnym wypadku plik z pojemnika S3 jest wczytywany do procesu funkcji. Wykonanie etapów funkcji można śledzić w logach aplikacyjnych co zostało przedstawione na Rysunku 20.



Rysunek 20. Logi aplikacyjne wygenerowane przez funkcję *Device Validation Process* zapisane w AWS CloudWatch . Źródło: opracowanie własne

Plik logu produkcyjnego urządzania jest w formacie JSON. Język Python umożliwia bardzo intuicyjną nawigację po tym formacie w kontraście do języka Java w wersji 8, gdyż jest on rzutowany na typ słownika z użyciem wbudowanej biblioteki. Kolejnym krokiem jest walidacja zawartości pliku, w szczególności dokonywane jest sprawdzenie czy nie wystąpiły odczyty pomiarów przekraczające dopuszczalne normy. Status procesowania wraz ze wszystkimi danymi posiadanymi na temat urządzenia jest zapisywany w bazie danych. Rysunek 22 przedstawia sposób wyszukania zapisanego urządzenia w interfejsie bazy danych na podstawie klucza głównego. Model rekordu bazy danych jest dostępny w dodatku C.

Rysunek 21. Podgląd bazy danych *Device Info* Dynamo DB . Źródło: opracowanie własne



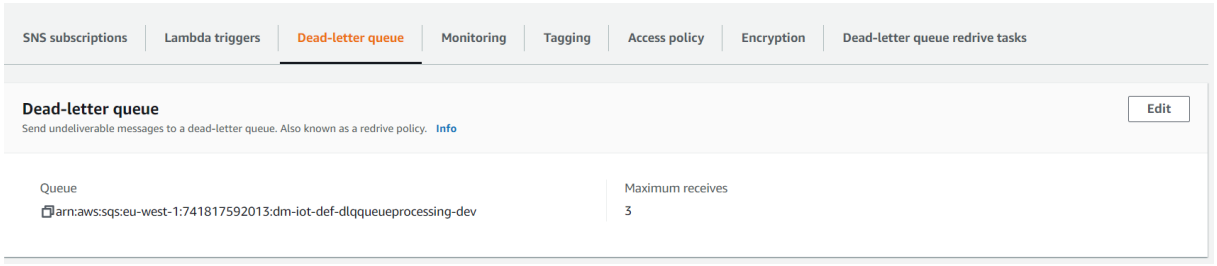
Rysunek 22. Wyszukiwanie urządzenia w bazie danych Dynamo DB. Źródło: opracowanie własne

Na tym etapie dokonywana jest segregacja urządzeń pod względem kontroli jakości. Pliki poprawnych urządzeń trafiają do pojemnika S3 o nazwie *Health Device Logs.*, a niepoprawne pliki trafią do *Unhealth Device Logs* S3. Podział ten ułatwia zadania administracyjne takie jak reklamacja wadliwych urządzeń do producentów. Obiekty w pojemnikach S3 są agregowane w prefiksach ze względu na datę załadowania obiektów w specjalnym formacie zwanym *hive-partitioning* [49] co ukazano dla przykładu w Listingu 4. Pozwala to wygodnie zarządzać obiektami z uwzględnieniem wymiaru czasu oraz zwiększa skalowalność użycia API usługi S3 zgodnie z zasadą zdefiniowaną przez AWS, że maksymalne limity zapytań odnoszą się do konkretnego prefiksu [50]. Ponadto ten format umożliwia analizę danych w pojemniku S3 z wykorzystaniem usługi AWS Athena poprzez zapytania podobne do języka SQL. Zapisanie pliku w jednym z dwóch pojemników S3 kończy procesowanie logu urządzenia.

Listing 4. Struktura prefiksów S3 używając *hive-partitions*. Źródło: opracowanie własne

```
s3://input_bucket/year=2021/month=12/day=05/ Log_1.json
s3://input_bucket/year=2021/month=12/day=11/ Log_2.json
s3://input_bucket/year=2022/month=01/day=10/ Log_3.json
s3://input_bucket/year=2022/month=01/day=10/ Log_4.json
s3://input_bucket/year=2022/month=02/day=01/ Log_5.json
```

W przypadku wystąpienia błędu odpowiednia informacja jest zapisywana w logach aplikacyjnych, które umożliwiają zidentyfikowanie problemu. Ponadto dokonywana jest ponowna próba przeprocesowania danego pliku. Jeżeli plik będzie zawierał krytyczny błąd uniemożliwiający poprawne procesowanie przez zdefiniowaną liczbę ponownych procesowań to trafi on do specjalnej kolejki zwanej *dead letter queue*. Konfigurację tej kolejki ukazuje Rysunek 23. Wiadomości z tej kolejki są obsługiwane przez dedykowaną funkcję Lambda nazwaną *Failed Events Handler*. Dokonuje ona zapisu pliku do specjalnego pojemnika S3 przechowującego wadliwe wiadomości. Umożliwia to analizę tych danych w dowolnym czasie w odrębnym procesie.



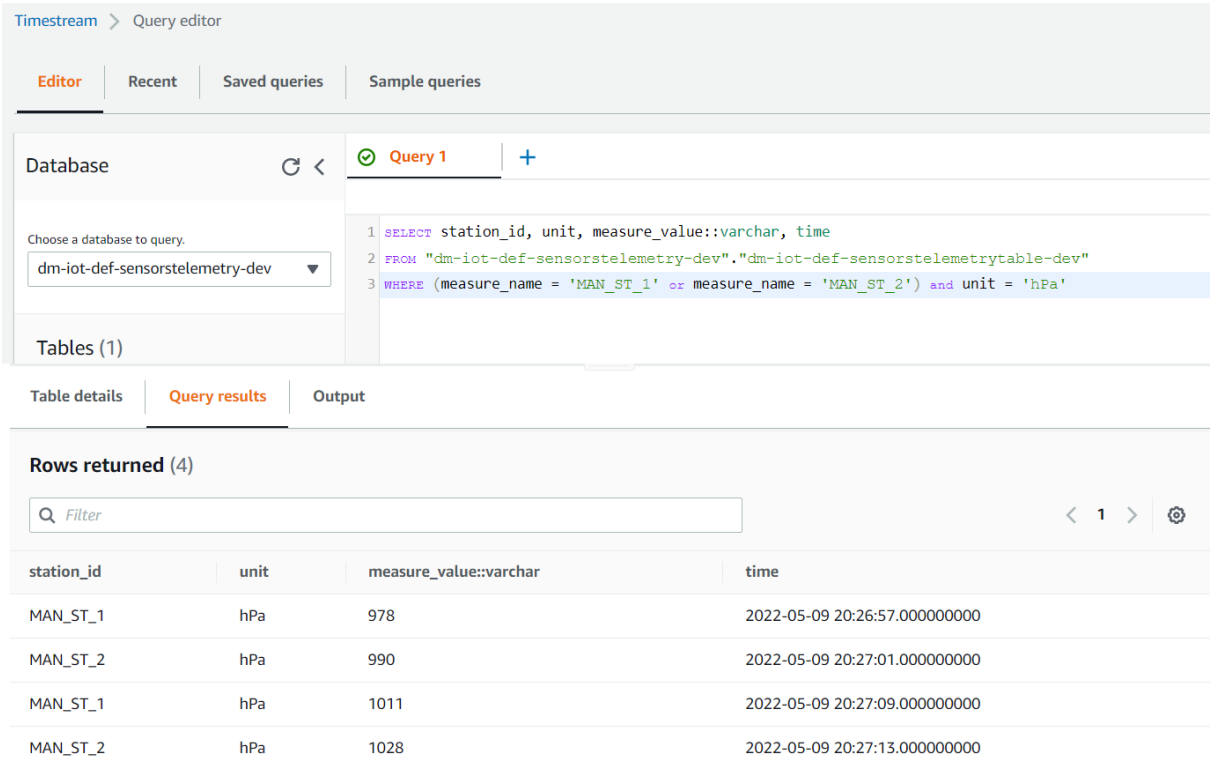
Rysunek 23. Konfiguracja kolejki *Dead Letter Queue* SQS. Źródło: opracowanie własne

4.3.3 Procesowanie danych telemetrycznych

Zewnętrzne systemy producentów urządzeń poza plikiem logu produkcyjnego dostarczają do platformy IoT dane telemetryczne. Są one ładowane w czasie rzeczywistym z dokładnością do opóźnienia związanego z transmisją w sieci. Zapis odczytów telemetrycznych odbywa się za pośrednictwem AWS SDK dla języka Python oraz instancji klienta usługi AWS Timestream. Odczyty wysyłane są do chmurowej platformy IoT w momencie odebrania wiadomości protokołu MQTT w funkcji *callback*, która nasłuchuje nadchodzących wiadomości.

Schemat komunikatu integracyjnego z bazą danych Timestream wymaga specyfikacji nazwy bazy danych, nazwy tabeli oraz rekordów do zapisu. Każdy rekord składa się z nazwy pomiaru, odczytanej wartości, typu danych, jednostki czasu oraz wartości czasu w formacie uniksowym (ang. *unix epoch*) [51]. Ponadto istnieje możliwość wyspecyfikowania dodatkowych atrybutów zwanych wymiarami, które są powiązane z odczytanym pomiarem.

W zamodelowanej strukturze danych Timestream DB nazwa pomiaru odnosi się do identyfikatora stacji pomiarowej. Wartości pomiarów to dane przesłane przez czujniki. Jednostką czasu są sekundy, a typ danych określony został jako VARCHAR. Wymiarami towarzyszącymi pomiarowi są identyfikator czujnika, jednostka pomiaru (volt, stopnie Celsjusza, hektopaskale, Ohm), stempel czasowy w formacie ISO oraz unikalny identyfikator odczytu.



Rysunek 24. Zapytanie do bazy danych Timestream na temat pomiarów ciśnienia na stacjach roboczych. Źródło: opracowanie własne

Baza danych Timestream umożliwia przeszukiwanie zgromadzonych danych z wykorzystaniem języka zbliżonego do SQL. Zapytanie na danych zgormadzonych danych dotyczące pomiaru ciśnienia pokazuje Rysunek 24. W konsoli AWS dostępne są przykładowe zapytania oraz materiały szkoleniowe, które pomagają w nauce konstrukcji składni do podstawowych zastosowań. Zdefiniowanie zapytania można zapisać i uruchamiać w zadanych interwałach czasu lub o określonej porze w celu generowania raportów.

Istotną cechą tej bazy danych jest rozbudowany kokpit do monitorowania charakterystyk ładowania danych do bazy oraz pracy silnika zapytań. Umożliwia to śledzenie procesu odczytu danych i wykrywanie anomalni w procesie odbioru telemetrii, co można zautomatyzować poprzez podpięcie odpowiednio zdefiniowanego alarmu.

4.3.4 Komunikacja z systemem poprzez REST API

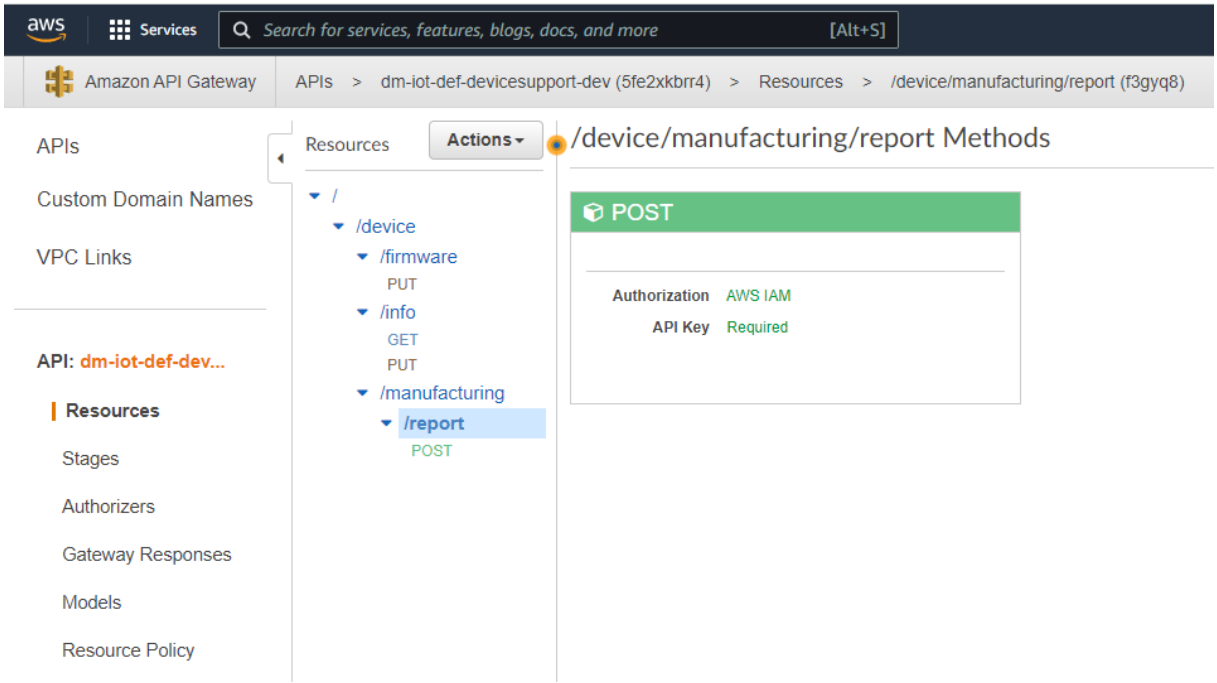
Interakcja z systemem IoT jest możliwa dzięki wykorzystaniu usługi AWS API Gateway. Oferuje ona możliwość wystawienia interfejsu REST API. W zbudowanym prototypie wszystkie punkty końcowe są obsługiwane przez funkcje Lambda, które realizują logikę biznesową i zwracają odpowiedni komunikat odpowiedzi. Zaimplementowane funkcjonalności umożliwiają:

- pobranie informacji o urządzeniu;
- zaktualizowanie danych urządzenia;
- rozpoczęcie procesu aktualizacji oprogramowania układowego;
- wygenerowanie raportu na temat produkcji urządzeń.

Definicja interfejsów została zawarta w standardzie Open API jako plik *swagger.yaml*. Jest to spopularyzowany format, który może zostać bezpośrednio zaimportowany przez usługę API Gateway. Ponadto jest dostępna opcja eksportu już zdefiniowanych zasobów w postaci tego pliku. Ten z kolei może zostać ponownie zaimportowany w innej instancji API Gateway. Ułatwia to proces potencjalnej migracji. Informacje zawarte w pliku *swagger* to:

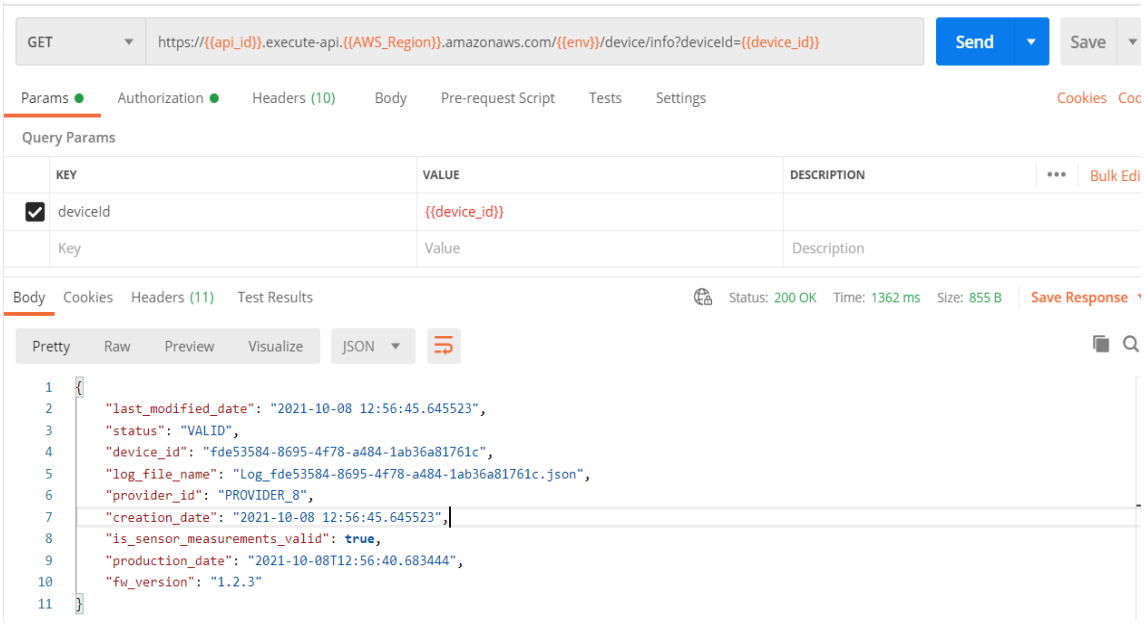
- adresy URL punktów końcowych;
- wykorzystana metoda protokołu HTTP;
- podsumowanie i szczegółowy opis biznesowy API;
- modele obiektów żądania i odpowiedzi;
- walidacje parametrów wejściowych na podstawie enumeracji czy wyrażeń regularnych;
- wyspecyfikowanie atrybutów obowiązkowych i opcjonalnych;
- definicja możliwych odpowiedzi powiązanych z kodem statusu;
- mechanizmy autentykacji z wyróżnieniem nagłówka zawierającego klucz;
- specjalne adnotacje AWS umożliwiające integrację z funkcjami Lambda.

Autentykacja opiera się na usłudze AWS IAM. Tylko użytkownicy lub role, które zostały zawarte w definicji polityki zasobu API Gateway, mogą wywołać dane API. Ponadto w nagłówku zapytania należy dołączyć klucz API. Służy on jako dodatkowy parametr poświadczający oraz reguluje on limity dostępu do danego zasobu. Każdy klucz API jest powiązany z planem użytkowania, który określa maksymalną liczbę wywołania punktu końcowego w danym okresie czasu oraz liczbę równoległych zapytań w ciągu sekundy. Podgląd listy zasobów oraz sposobu ich autentykacji został przedstawiony na Rysunku 25.



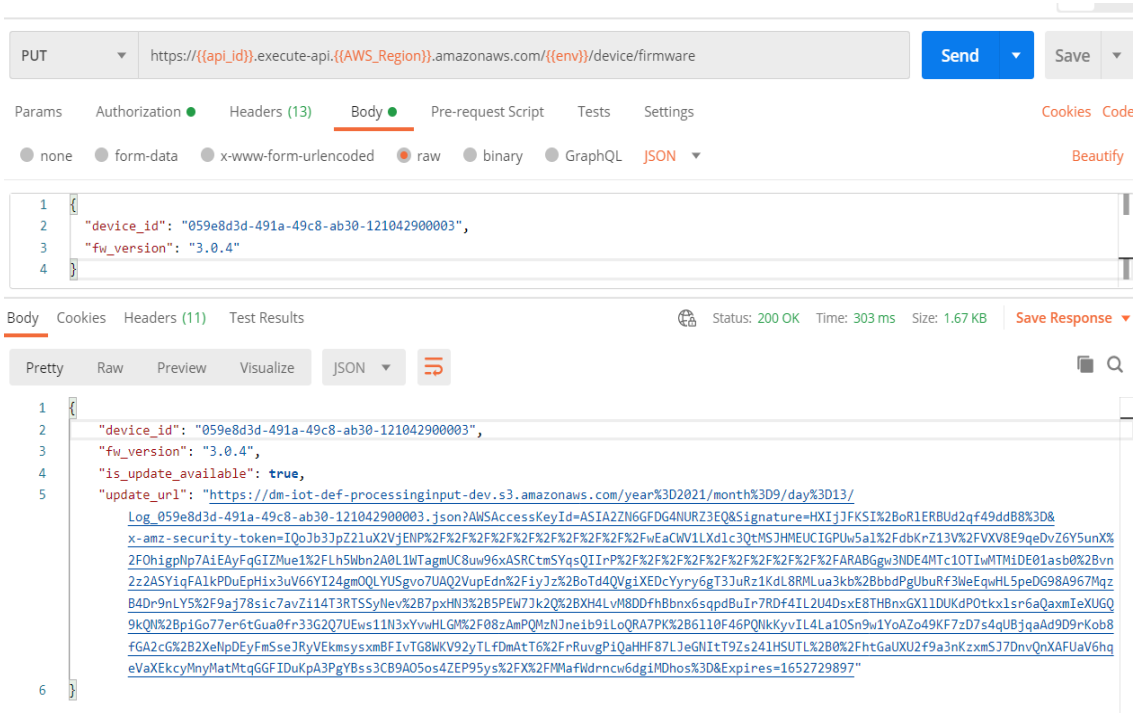
Rysunek 25. Podgląd zdefiniowanych zasobów API Gateway. Źródło: opracowanie własne

Funkcjonalności pobierania i aktualizacji informacji o urządzeniu mogą mieć szerokie zastosowania takie jak integracja z komponentami wizualizującymi typu *Front-End*. Funkcje Lambda realizujące logikę biznesową dokonują pobrania atrybutów wejściowych z obiektu zdarzenia oraz ich walidacji. Następnie wykonane jest zapytanie do bazy danych Dynamo DB o nazwie *Device Info*. Odpowiednia informacja jest zwracana do klienta w zależności czy rekord został odnaleziony w tabeli. Rysunek 26 ukazuje komunikat odpowiedzi dla znalezionego urządzenia widoczny w programie Postman.



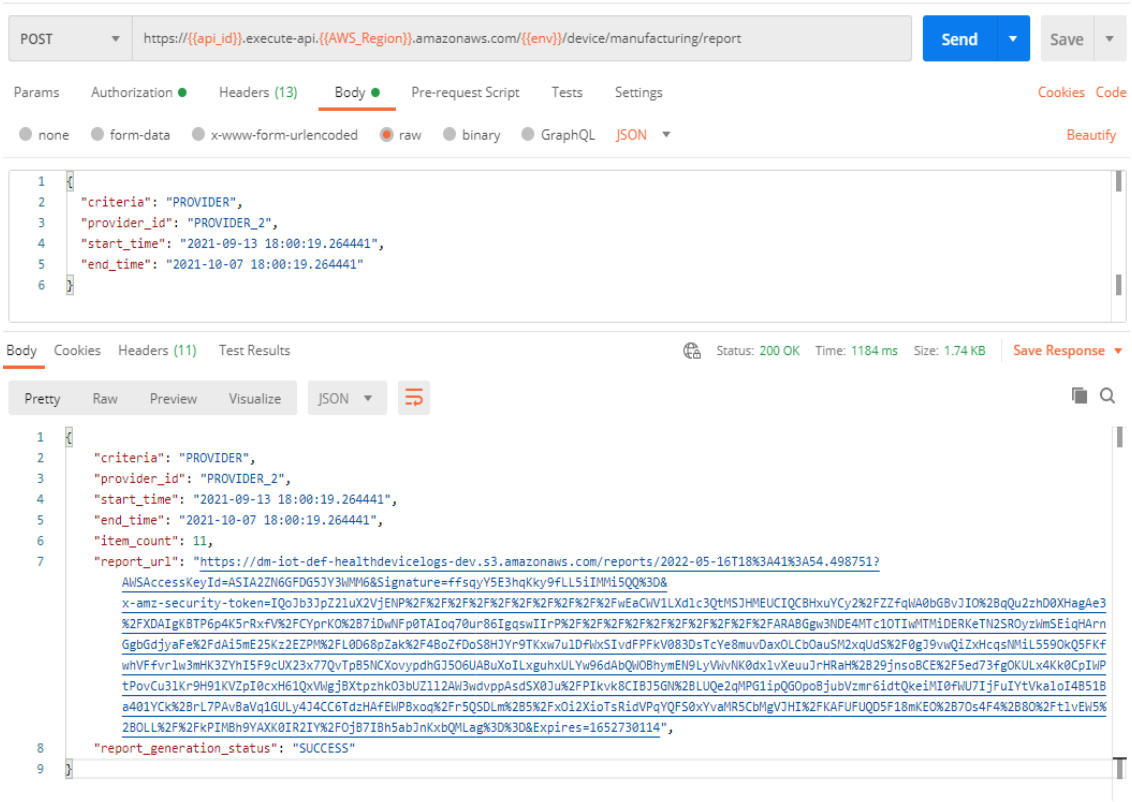
Rysunek 26. Wywołanie *Get Device Info* API z użyciem programu Postman. Źródło: opracowanie własne

Funkcjonalność oferująca zaktualizowanie oprogramowania układowego (ang. *firmware*) polega na sprawdzeniu czy dostępna jest nowsza wersja oprogramowania. Proces zostanie rozpoczęty gdy urządzenie spełnia warunki początkowe takie jak poprawny rekord w bazie danych, poprawny plik logu produkcyjnego pojemniku S3 oraz obecnie wgrana wersja oprogramowania jest niższa niż najnowsza wersja. Stąd ten punkt końcowy może zwrócić kod statusu 404 i komunikat o brakującym rekordzie lub pliku w spodziewanym miejscu. W szczęśliwej ścieżce wywołania zwrócony zostanie link do nowej wersji oprogramowania co pokazuje Rysunek 27 lub informacja o tym, że aktualizacja nie jest potrzebna.



Rysunek 27. Odpowiedź *Device Firmware* API dla przypadku znalezienia nowszej wersji oprogramowania. Źródło: opracowanie własne

Automatyzacja funkcjonalności generowania raportów jest bardzo użyteczna z perspektywy zespołów analitycznych lub kadry zarządzającej. Stąd powstało API, które na podstawie parametrów wejściowych generuje plik raportu i zapisuje go w pojemniku S3. W zaimplementowanym punkcie końcowym należy zdefiniować temat raportu. Dostępne tematy to podsumowanie produkcji urządzeń ze względu na producenta lub status kontroli jakości. Następnie należy określić rozpatrywany zakres czasowy. Po wysłaniu żądania dokonywana jest walidacja. Jeśli wszystkie warunki zostaną spełnione to wygenerowany raport jest gotowy do pobrania w linku załączonym w odpowiedzi. Odpowiedź z API wraz z wygenerowanym linkiem został ukazany na Rysunku 28. Dostarczony link wykonany został z użyciem funkcjonalności S3 *Pre-signed URLs* [52]. Pozwala to na pobranie obiektu tylko zautoryzowanym encjom, które mają dostęp do adresu URL. Ponadto link wygaśnie po określonym czasie. Domyślnie jest to okres jednej godziny.



Rysunek 28. Generacja raportu na temat produkcji urządzeń danego producenta z użyciem *Generate Report* API. Źródło: opracowanie własne

4.3.5 Monitoring, alarmy i analiza logów aplikacyjnych

Do zarządzania dziedziną problemową jaką jest szeroko pojęty monitoring wykorzystano usługę AWS CloudWatch. Oferuje ona wachlarz funkcjonalności, które pozwalają stworzyć kompleksowe narzędzie do monitorowania platformy. W zaimplementowanym prototypie usługa CloudWatch została użyta w następujących zastosowaniach:

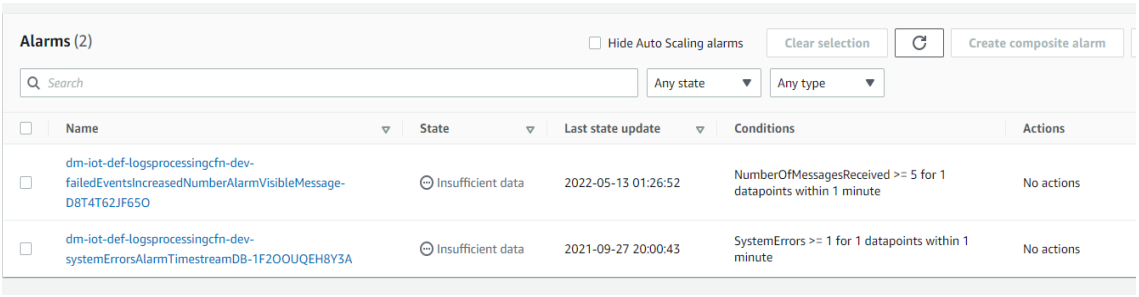
- przechowywanie logów aplikacyjnych z funkcji Lambda i API Gateway przy użyciu AWS CloudWatch Log Groups;
- bardzo szybkie przeszukiwanie logów aplikacyjnych z wykorzystaniem języka zapytań AWS CloudWatch Insights;
- wywołanie alarmów w reakcji na zdefiniowane zdarzenia w oparciu o AWS CloudWatch Alarms;
- monitorowanie charakterystyk usług takich jak ilość błędów czy wydajność z użyciem AWS CloudWatch Metrics.

Domyślnie logi aplikacyjne są przechowywane w nieskończoność. W celu obniżenia kosztów można zdefiniować strategię przechowywania logów, w której wpisy starsze niż określona wartość czasu od momentu powstania danych, zostaną zarchiwizowane w pojemniku S3.

Funkcje Lambda współdzielą klasę loggera w bibliotece, która została stworzona jako AWS Layers. Pozwoliło to uzyskać standaryzację sposobu logowania. Schemat logu przyjmuje postać ustrukturyzowanego tekstu, gdzie poszczególne parametry znajdują się w ustalonej kolejności w

nawiasach kwadratowych. Zdefiniowane parametry to poziom logowania, nazwa funkcji, stempel czasowy oraz docelowa wiadomość. Najprostszy raport o błędach można wygenerować wyszukując wpisy o poziomie logowania równym *ERROR*.

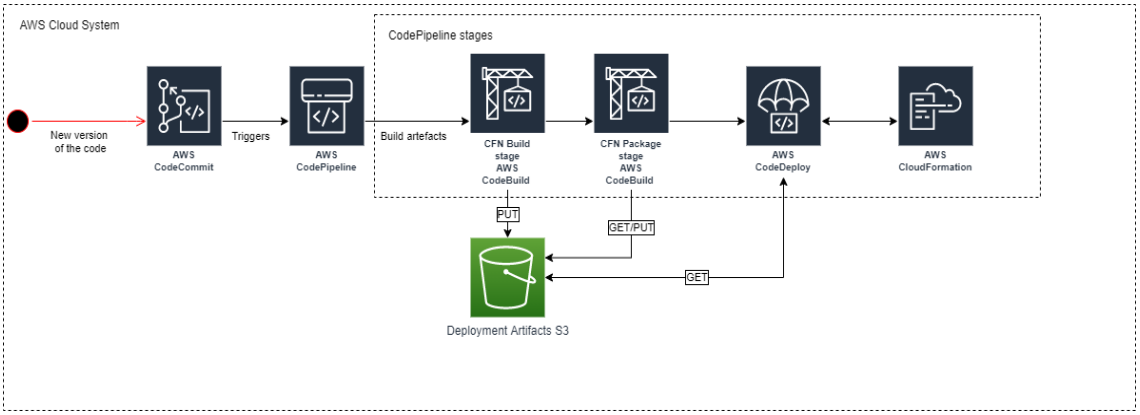
Utworzenie alarmów jest istotne z punktu widzenia automatyzacji procesu reakcji na niebezpiecznie zdarzenia z perspektywy funkcjonowania systemu. Stworzone alarmy dotyczą sytuacji, gdy zwiększona liczba wiadomości wejściowych nie jest w stanie się prawidłowo przeprocesować. W ten sposób monitorowany jest proces odbioru danych telemetrycznych w bazie Timestream oraz liczba wiadomości przekazanych do kolejki *Dead Letter Queue* SQS. Rysunek 29 przedstawia zbiór zdefiniowanych alarmów w zbudowanym prototypie wraz z warunkami wymaganymi do uruchomienia akcji. Po spełnieniu warunku uruchomienia alarmu zostanie zmieniony status. Reakcją na powstały alarm mogą być akcje takie jak wysłanie emaila lub odcięcie integracji pomiędzy usługami w celu uniknięcia propagacji wadliwych danych wewnątrz systemu. Może to przyczynić się do oszczędności mocy obliczeniowej związanej z nieprocesowaniem niepoprawnych komunikatów.



Rysunek 29. Podsumowanie warunków zdefiniowanych alarmów. Źródło: opracowanie własne

4.3.6 Potok wdrożenia CI/CD

W celu sprawnego dostarczania nowej wersji systemu należało zbudować proces wdrożenia. Punktem startowym jest wypchnięcie zmiany kodu źródłowego do usługi wersjonowania plików jaką jest AWS CodeCommit. To zdarzenie w sposób automatyczny uruchamia kolejne fazy potoku wdrożenia zdefiniowanego w usłudze AWS CodePipeline. Rysunek 30 przedstawia architekturę zaimplementowanego procesu CICD w prototypie systemu chmurowego.

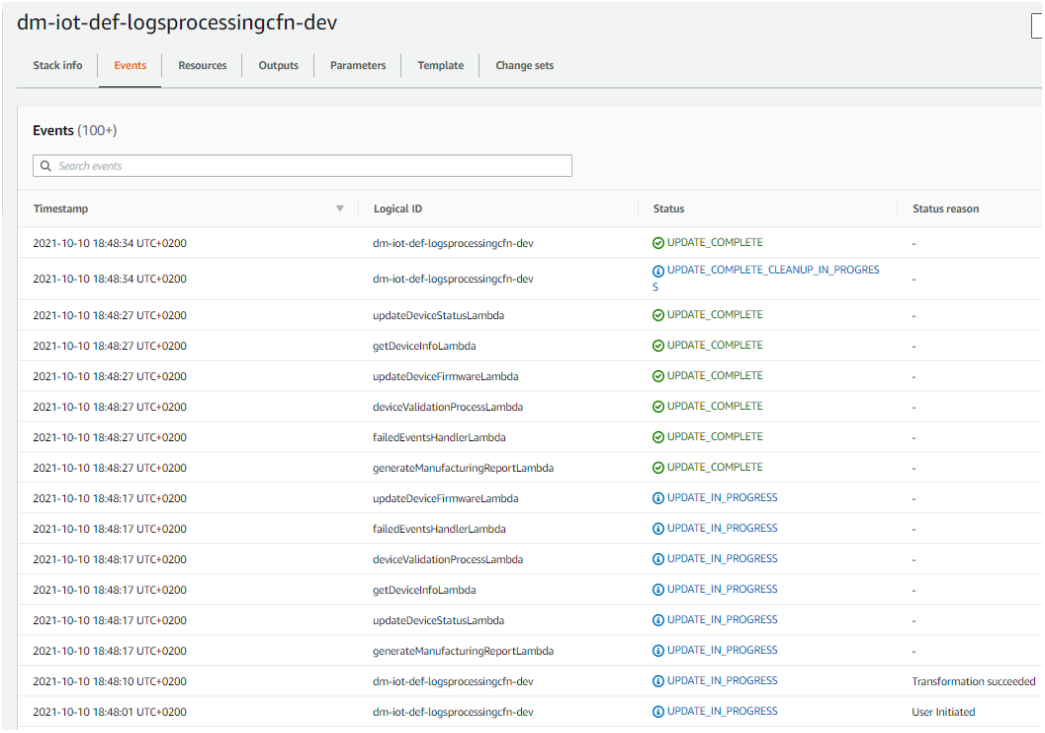


Rysunek 30. Architektura procesu CI/CD systemu chmurowego. Źródło: opracowanie własne

Pierwszą fazą potoku jest pobranie plików kodu źródłowego z repozytorium, zainstalowanie dodatkowych zależności dla funkcji Lambda oraz zbudowanie współdzielonej biblioteki umieszczonej w Lambda Layers. Czynności te odbywają się w zdefiniowanym zadaniu usługi AWS CodeBuild. Nawigacja po folderach funkcji Lambda jest wykonana w oparciu o ustaloną strukturę projektu. Zbudowane artefakty w postaci kompletnych, spakowanych paczek funkcji Lambda są załadowane do specjalnego pojemnika S3 o nazwie *Deployment Artifacts.* Ponadto wgrany zostanie plik swagger.yaml, który następnie zostanie zaimportowany przez API Gateway.

W kolejnym etapie potoku dokonywane jest wczytanie skryptu AWS CloudFormation i rozwiązanie użytych zmiennych dynamicznych. Skrypt ten zawiera definicję wszystkich zasobów chmurowych w myśl zasady *Infrastructure as a Code.* W rezultacie zaktualizowany skrypt zostanie umieszczony w pojemniku wdrożeniowym S3.

Następnym krokiem jest uruchomienie wdrożenia nowej wersji. Dokonywane jest porównanie obecnego stanu infrastruktury zapisanego w AWS CloudFormation z nową definicją. Zmiany są wypisane jako *Change Set.* Sposób wdrożenia zasobów zależy od typu usługi. Funkcje Lambda i wgranie definicji API Gateway przeładowywane są zawsze, natomiast zasoby takie jak Dynamo DB czy też S3 zostaną dołączone do listy zmian tylko wtedy, gdy została wykryta zmiana w konfiguracji. Po zakończeniu wgrania zmian proces się kończy i nowe funkcjonalności są gotowe do użycia. W przypadku niepowodzenia wdrożenia w wyniku błędu, usługa AWS CloudFormation dokona próby przywrócenia poprzedniego stanu systemu. W krytycznych przypadkach nie jest to możliwe na przykład gdy zostały wprowadzone zmiany manualnie, które rozsynchronizowały stan faktyczny z zawartością skryptu CloudFormation. Informacje o przebiegu wdrożenia można śledzić w kokpicie zdarzeń potoku co ukazuje Rysunek 31.



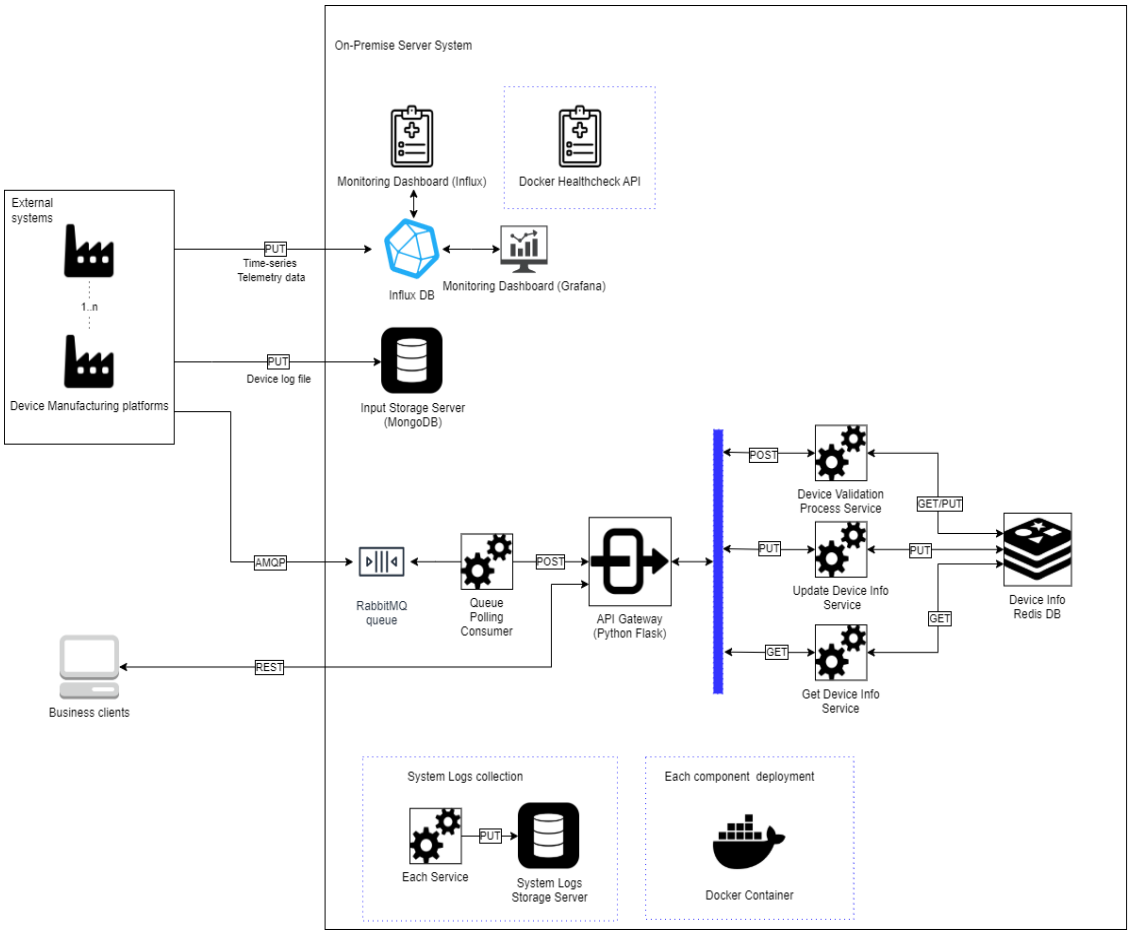
Rysunek 31. Zdarzenia procesu wdrożenia nowej wersji systemu chmurowego w usłudze AWS CloudFormation. Źródło: opracowanie własne

4.4 Budowa skonteneryzowanego systemu IoT

Technologia konteneryzacji stanowi znaczny udział w rozwiązaniach stosowanych na rynku pracy. Znajduje ona powszechne zastosowania również w usługach chmurowych. Usługi typu Serverless oferują korzystanie z narządzi zbudowanych w oparciu o konteneryzację bez potrzeby zarządzania kontenerami. Jednakże wiedza praktyczna na temat charakterystyk korzystania ze skonteneryzowanych aplikacji znacznie zwiększa świadomość i umiejętność rozwiązywania problemów również w systemach chmurowych. Motywacja zwiększenia umiejętności w zakresie platformy Docker przyczyniła się do budowy systemu IoT opartego na powiązanych, skonteneryzowanych aplikacjach. Funkcjonalności budowanego prototypu miały za zadanie odwzorować możliwości i cele systemu chmurowego. W związku z tym wykonana jest integracja z komponentem *Device Manufacturing Platform*, który generuje dane wejściowe do procesowania. Do budowy systemu wykorzystano zarówno gotowe obrazy dostępne w repozytorium DockerHub oraz wzbogacane obrazy o własną logikę biznesową zdefiniowaną w plikach Dockerfile. Skorzystano z narzędzi typu Docker Compose, Docker Desktop oraz języka programowania Python.

4.4.1 Architektura prototypu systemu skonteneryzowanego

Założeniem budowy prototypu wykorzystującego platformę Docker było zrealizowanie przedstawionych celów biznesowych poprzez implementację logiki biznesowej, ale także zapewnienie wysokiej skalowalności i odporności na awarie. Stąd przy projektowaniu systemu jasno zostały zdefiniowane granice i odpowiedzialności poszczególnych aplikacji. Ma to na celu umożliwienie niezależnego skalowania luźno powiązanych komponentów. Inspiracją do tworzenia architektury prototypu była próba zrozumienia jak analogiczna funkcjonalność jest zbudowana w usługach chmurowych. Stąd diagram architektury systemu skonteneryzowanego przypomina topologicznie architekturę systemu chmurowego. Rysunek 32 przedstawia wykonany diagram zaprojektowanej architektury prototypu systemu zbudowanego z użyciem platformy Docker.



Rysunek 32. Diagram architektury systemu skonteneryzowanego z użyciem platformy Docker. Źródło: opracowanie własne

Platforma IoT jest zintegrowana z zewnętrznymi systemami produkującymi urządzenia oraz z klientami biznesowymi. Komponent producentów urządzeń przesyła dane z wykorzystaniem zdefiniowanych natywnych REST API dla narzędzi MongoDB oraz InfluxDB. Informacja o dostarczeniu pliku logu produkcyjnego danego urządzenia jest dostarczona za pośrednictwem protokołu AMQP do aplikacji kolejki RabbitMQ. Pojawienie się komunikatu na kolejce jest wyzwalaczem do dalszego procesowania pliku wewnątrz systemu. Natomiast klienci biznesowi mogą korzystać z funkcjonalności platformy poprzez używanie wystawionych API poprzez serwer aplikacyjny napisany z użyciem narzędzia Python Flask.

Wzajemna komunikacja wewnętrzna pomiędzy kontenerami jest możliwa poprzez zdefiniowanie wspólnej sieci. Definicja wielu aplikacji przy użyciu Docker Compose pozwala na odnoszenie się po nazwach logicznych do innych kontenerów co znacznie ułatwia integracje. W przypadku interakcji zewnętrznej z platformą IoT wymagane jest wystawienie portów, na których nasłuchuje zaimplementowana aplikacja. Umieszczenie całej infrastruktury w plikach Dockerfile oraz Docker Compose umożliwia łatwą migrację, ponowne użycie pojedynczych aplikacji oraz skalowanie wybranych kontenerów. Ponadto umożliwia to szybką replikację lub odbudowę systemu w przypadku wystąpienia awarii.

Dane aplikacyjne są przechowywane w użyciem funkcjonalności Docker Volumes. Dokumenty bazy danych MongoDB, pliki logów produkcyjnych urządzeń czy rekordy baz danych InfluxDB i

RedisDB są dostępne po ponownym uruchomieniu aplikacji. Logi aplikacyjne zapisywane są z użyciem natywnej funkcjonalności Dockera i dostępne są w zdefiniowanym katalogu hosta.

Do administracji systemu służy aplikacja Docker Desktop oraz komendy wiersza poleceń. Do typowych zadań administracyjnych podczas budowy prototypu zalicza się:

- budowanie obrazów kontenerów;
- uruchamianie instancji kontenerów;
- zatrzymywanie instancji kontenerów;
- sprawdzenie listy uruchomionych instancji kontenerów;
- sprawdzenie listy dostępnych obrazów;
- sprawdzanie szczegółów instancji kontenera z wyszczególnieniem przypisanej sieci;
- konfiguracja sieci;
- czyszczenie nieużywanych zasobów takich jak obrazy, sieci, zatrzymane instancje kontenerów;
- nawigacja po logach aplikacyjnych;
- logowanie się do wnętrza instancji kontenera;
- skalowanie liczby instancji kontenera;
- nawigacja po kokpitach administracyjnych aplikacji;
- sprawdzenie stanu zdrowia kontenerów;
- zarządzanie wolumenami danych.

Uruchomienie platformy po raz pierwszy wymaga pobrania i budowy obrazów zdefiniowanych w plikach Docker Compose, a następnie ich uruchomienia. W celu minimalnego czasu wdrożenia zalecane jest powoływanie do działania składowych systemów w ustalonej kolejności zgodnie z kierunkiem zależności aplikacji. Nie jest to jednak warunek obowiązkowy, gdyż aplikacje są luźno powiązane i mogą zostać uruchomione niezależnie od siebie.

4.4.2 Porównanie funkcjonalności systemu chmurowego i skonteneryzowanego

Skonfigurowanie integracji pomiędzy generatorem danych i systemem złożonym z aplikacji Docker sprowadza się do ustawienia zmiennej środowiskowej wskazującej na system *on-premise* zamiast na platformę chmurową. Ze względu na użycie różnorodnych narzędzi w aplikacjach kontenerowych takich jak MongoDB, RabbitMQ, InfluxDB to każdą aplikację należało integrować dedykowaną instancją klienta, zapoznać się z dostępnymi API oraz dokonać połączenia uwzględniając autoryzację. W przypadku systemu chmurowego za połączenia z usługami odpowiadało SDK boto3 oraz klucze użytkownika technicznego stworzonego w obszarze IAM.

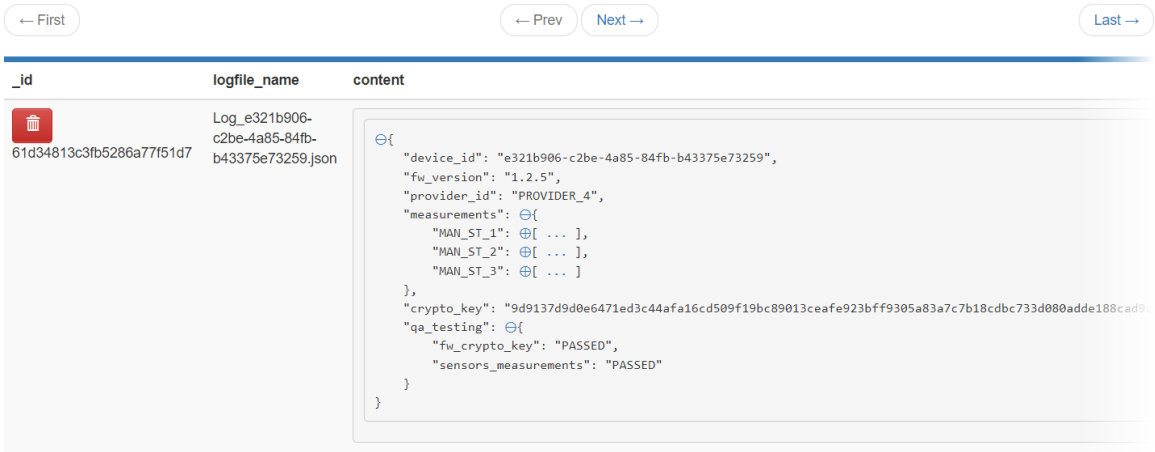
Zarządzanie infrastrukturą i wdrażanie nowej wersji oprogramowania w systemie chmurowym polegało na procesie CI/CD opartym na natywnych usługach AWS z wyszczególnieniem CloudFormation. W systemie skonteneryzowanym definicja komponentów została zapisana w plikach Dockerfile oraz Docker Compose. Wprowadzenie zmian wymagało przebudowy obrazów kontenerów i ponowne ich uruchomienie. Zostało to zautomatyzowane skryptami napisanymi w powłoce bash oraz

języku Python. Potencjalnym usprawnieniem tego procesu byłoby skorzystanie z narzędzia zarządcy wdrożenia takiego jak Jenkins co automatycznie by uruchamiało odpowiednie skrypty po wykryciu zmiany w repozytorium kodu. Rysunek 33 przedstawia status uruchomionych kontenerów po wdrożeniu systemu.

```
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS          PORTS                      NAMES
a4bf10112904   python-docker      "python3 -m flask ru…"   6 minutes ago   Up 6 minutes    0.0.0.0:5000->5000/tcp     api_gateway_web_server_1
61713c0263a6   influxdb           "/entrypoint.sh infl…"   4 months ago    Up 20 minutes   0.0.0.0:8086->8086/tcp     influxdb
cd3131942ed3   grafana/grafana    "/run.sh"                4 months ago    Up 20 minutes   0.0.0.0:3000->3000/tcp     grafana
f505c053777d   redis:alpine       "docker-entrypoint.s…"   4 months ago    Up 20 minutes   0.0.0.0:6379->6379/tcp     nosql_redis_db_server_redis_1
076fb350e329   mongo              "docker-entrypoint.s…"   4 months ago    Up 20 minutes   0.0.0.0:27017->27017/tcp   storage_server_mongo_1
36de75a20cb8   mongo-express      "tini -- /docker-ent…"   4 months ago    Up 20 minutes   0.0.0.0:8081->8081/tcp     storage_server_mongo-express_1
```

Rysunek 33. Lista uruchomionych instancji kontenerów prototypu systemu IoT . Źródło: opracowanie własne

Do zrealizowania funkcjonalności przechowywania plików logów produkcyjnych urządzeń wykorzystano serwer bazy danych MongoDB. Jest to nierelacyjna dokumentowa baza danych wyspecjalizowana w przechowywaniu obiektów w formacie JSON. Analogiczna funkcjonalność w systemie chmurowym została zrealizowana w użyciem pojemnika AWS S3. Charakterystyki bazy danych MongoDB pozwalają na skalowanie i szybki dostęp do danych. Jednakże przewagą usługi S3 okazała się łatwość skonfigurowania wyzwalacza w momencie pojawienia się pliku co rozpoczyna procesowanie. Narzędzie MongoDB posiada funkcję do notyfikowania o zmianach nazwaną *Change Streams*, jednakże konfiguracja tego rozwiązania wymaga specjalnych, nietrywialnych zabiegów takich jak stworzenie repliki bazy danych oraz odpowiednie podpięcie sieci. Trudnością okazało się zautomatyzowanie tego procesu w plikach Docker Compose. W rezultacie notyfikacja o załadowaniu pliku jest wysyłana do kolejki RabbitMQ w momencie otrzymania potwierdzenia o poprawnym umieszczeniu rekordu w bazie danych MongoDB. Dostęp do logów urządzeń możliwy jest poprzez kokpit administracyjny dostarczony przez narzędzie MongoDB Express lub z użyciem języka zapytań wiersza poleceń. Rysunek 34 przedstawia wizualizację danych o urządzeniu w bazie danych MongoDB.



Rysunek 34. Podgląd rekordu urządzenia w kontenerze MongoDB Express. Źródło: opracowanie własne

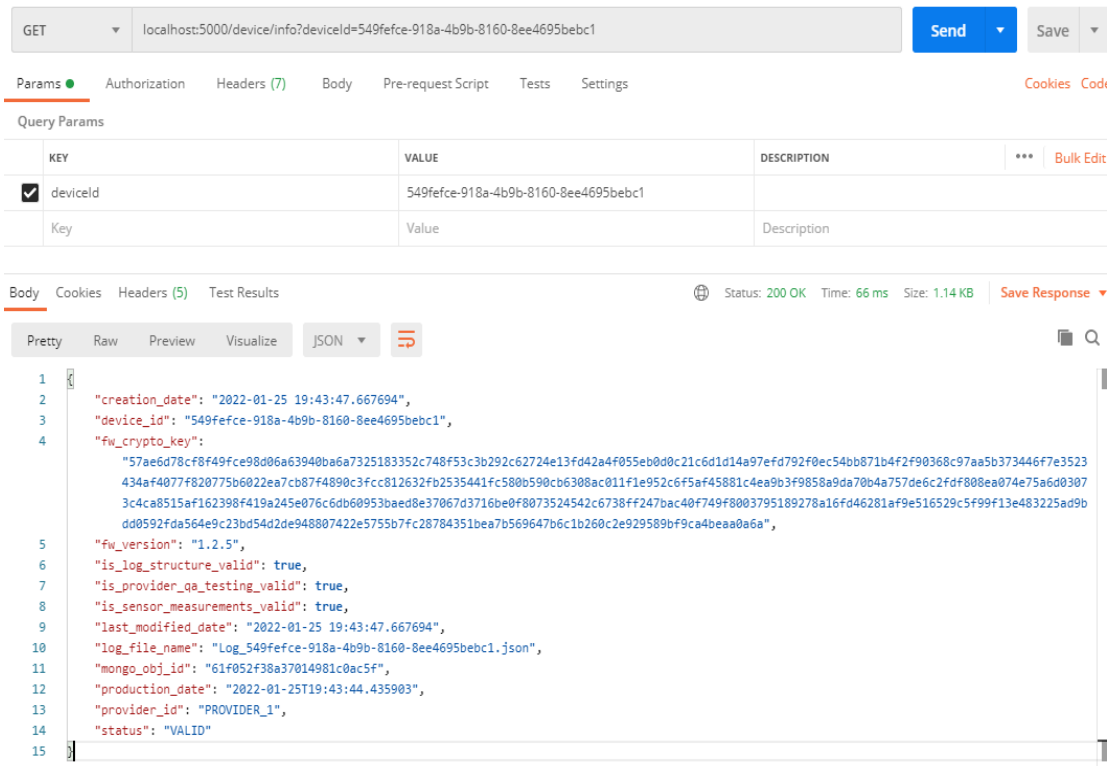
Zastosowanie kolejki procesowania zostało zrealizowane dzięki kontenerowi RabbitMQ. Odbiera ona komunikaty z zewnętrznego systemu poprzez protokół AMQP. Pozwala to osiągnąć zrównoważone procesowanie danych w oparciu o dostępne moce obliczeniowe oraz umożliwia

ponowienie obsługi komunikatu w przypadku wystąpienia błędu. Jest analogiczne zastosowanie jak użycie kolejki SQS w systemie chmurowym.

Wiadomości z kolejki RabbitMQ są odczytywane przez aplikację nazwaną *Queue Polling Consumer*, która dokonuje wczytania i walidacji komunikatu, a następnie wywołuje API usługi *Device Validation Processing Service.* Komunikat z kolejki zawiera informacje o nazwie pliku urządzania oraz identyfikatorze rekordu w bazie danych MongoDB. Udostępnione punkty końcowe są zaimplementowane z użyciem kontenera zawierającego serwer aplikacyjny Python Flask. Instancja tego kontenera nasłuchuje zapytań na wystawionym porcie o numerze 5000. Funkcja odpowiedzialna za obsługę przeprocesowania pliku urządzenia posiada logikę biznesową taką jak funkcja Lambda *Device Validation Processing* w platformie chmurowej IoT. Po przeprowadzonej niezależnej walidacji rekord zostanie zapisany w bazie danych RedisDB o nazwie *Device Info.*

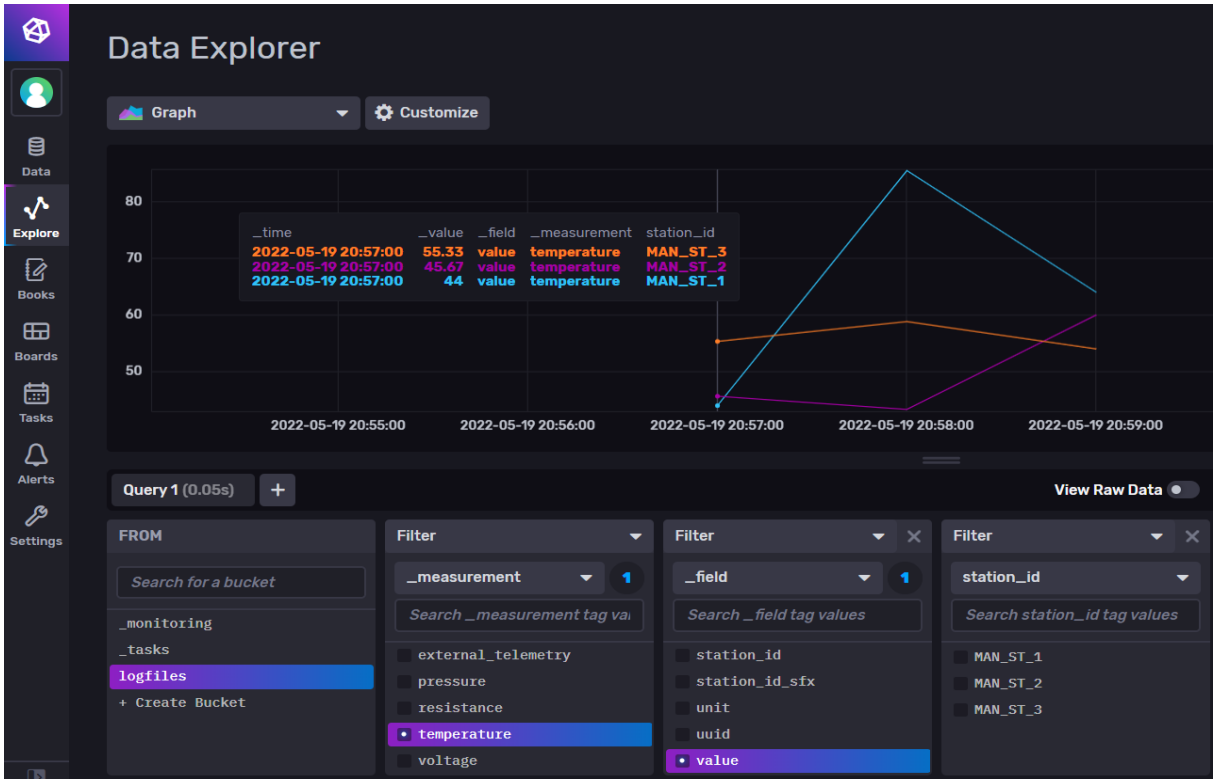
Użycie bazy danych Redis uzasadnione było podobną charakterystyką do usługi AWS DynamoDB. Obydwie bazy są typu klucz – wartość oraz posiadają elastyczną strukturę rekordu włączając w to atrybuty złożone. Instancja kontenera zawierającego serwer bazy danych RedisDB domyślnie nasłuchuje na porcie o numerze 6379.

Kontener obsługujący zapytania REST API udostępnia także funkcje pobrania oraz aktualizacji informacji o urządzeniu. Instancja serwera Python Flask w systemie skonteneryzowanym zrealizuje odpowiedzialności usługi AWS API Gateway w platformie chmurowej. Możliwości kontenera symulującego fasadę REST API są bardzo ograniczone w stosunku do jej odpowiednika w systemie chmurowym. W szczególności dotyczy to rozbudowanych mechanizmów autentykacji, logowania oraz kokpitu administracyjnego. Interakcja klientów biznesowych z wystawionymi API jest możliwa z użyciem narzędzia Postman co ukazano na Rysunku 35 dla pobrania danych urządzenia.



Rysunek 35. Wywołanie API *Get Device Info* obsłużonego przez kontener Python Flask . Źródło: opracowanie własne

Dane telemetrycznie w systemie *on-premise* są zapisywane w bazie danych *time-series* o nazwie InfluxDB. Podobnie jak w systemie chmurowym, wysłanie danych odbywa się w czasie prawie rzeczywistym w momencie odebrania komunikatu z czujnika. Zgromadzone dane służą do wykonywania analiz i wizualizacji z wyszczególnieniem wymiaru czasu. Użyta instancja kontenera pozwala zalogować się do kokpitu administracyjnego bazy danych Influx i dokonywać eksploracji danych. Rysunek 36 przedstawia wizualizację zmiany odczytów temperatury na wszystkich trzech stacjach pomiarowych. Przedstawienie danych może być wykonane w postaci grafu, histogramu, heatmapy czy tabeli. Analogiczną funkcjonalność w systemie chmurowym spełnia usługa AWS Timestream. Z doświadczeń w korzystaniu z obu narzędzi można sformułować wniosek, że narzędzie InfluxDB posiada większe możliwości budowania kokpitów administracyjnych natomiast usługa AWS Timestream ma bardziej rozbudowany język zapytań i jest z dziedziny Serverless co zapewnia wysoką skalowalność.



Rysunek 36. Kokpit administracyjny InfluxDB przedstawiający wizualizację odczytów zmiany temperatury na stacjach roboczych. Źródło: opracowanie własne

Monitorowanie platformy i alarmowanie o błędach zostało oparte na natywnych możliwościach zbierania logów z instancji kontenerów oraz korzystając z wewnętrznych API Dockera. Stąd możliwe jest śledzenie logów aplikacyjnych i analizowanie błędów systemu na tej podstawie. Jednakże w porównaniu z zastosowaną usługą AWS CloudWatch w platformie chmurowej jest to rozwiązanie dość ograniczone i niewygodne. Przede wszystkim brakuje wbudowanego silnika do przeszukiwania plików logów co umożliwiało by szybkie sprawdzenia lub analizy. Usprawnieniem obecnej implementacji byłoby stworzenie dedykowanego kontenera świadczącego usługę zbierania logów aplikacyjnych z możliwością ich przeszukiwania. Kandydatem do realizacji tego zadania jest narzędzie Elasticsearch z wbudowanym kokpitem do eksploracji danych zwanym Kibana.

4.5 Opis rezultatów prac wykonanych prototypów

Wykonanie prac implementacyjnych zbudowanych prototypów okazało się doskonałym sposobem do eksploracji możliwych rozwiązań technologicznych i zwiększenia umiejętności praktycznych w badanej dziedzinie problemowej. Powstałe prototypy realizują główne zakładane cele biznesowe. Jednakże część funkcjonalności musiała zostać przeprojektowana lub ograniczona ze względu na napotkane trudności techniczne. Wykonane systemy są dowodem, że obecnie dostępne narzędzia i usługi pozwalają na budowę średnio-zaawansowanych platform w danej domenie problemowej w stosunkowo krótkim czasie i niewysokich nakładach kapitałowych. Wykorzystanie gotowych rozwiązań pozwala łatwo wdrożyć zaawansowane koncepty technologiczne w swoją implementację takie jak wysoka skalowalność, replikacje danych, odporność na awarie z możliwością szybkiej odbudowy systemu.

Możliwości wykonanych prototypów w obu technologiach obejmują:

- równoległe procesowanie dużych ilości danych w postaci komunikatów telemetrycznych;
- przechowywanie i dostęp do zgromadzonych danych w silnikach bazodanowych;
- wizualizacje i kokpity do śledzenia procesowania w czasie rzeczywistym;
- wykonywanie raportów i zapytań analitycznych poprzez wbudowane narzędzia oraz API;
- łatwe wparcie integracji dla nowych klientów w postaci fabryk producentów urządzeń;
- ułatwiona migracja, replikacja lub odbudowa systemu po katastrofie poprzez zdefiniowanie systemów jako IaaC;
- równoległa praca wielu programistów poprzez stosowanie narzędzi do wersjonowania i automatycznego wdrożenia kodu źródłowego.

Poszczególne prototypy charakteryzują się innym zbiorem powstałych zalet i wad ze względu na wykorzystaną technologię. Obiektywnie można stwierdzić, że wykonany prototyp systemu chmurowego jest bardziej doskonały niż jego odpowiednik korzystający z technologii konteneryzacji. Jest to spowodowane wykorzystaniem gotowych i sprawdzonych rozwiązań udostępnionych przez dostawcę usług chmurowych względem własnej implementacji i integracji rozwiązań kontenerowych, które niepozostały wolne od błędów lub niedogodności. Główną zaletą obu prototypów jest wykorzystanie nowoczesnej technologii co pozwoliło na eksplorację nowych możliwości pod względem dydaktycznym oraz pozwoli na długoterminowy rozwój i dostęp do specjalistów na rynku z perspektywy zarządzania projektem. Kolejną zaletą systemów jest wysoka wydajność poprzez wykorzystanie przetwarzania równoległego opartego na zdarzeniach i luźne powiązania architektoniczne komponentów z użyciem kolejek. Platforma chmurowa gwarantuje również niskie koszty utrzymania ze względu na użycie modelu *Pay as you go* dla usług bezserwerowych. Ten aspekt jest z kolei wadą systemu skonteneryzowanego, który wymaga kosztów zorganizowania infrastruktury sprzętowej do wdrożenia. Zaletami obu prototypów są dość rozbudowane kokpity administracyjne i możliwość tworzenia wizualizacji dla śledzenia danych. W przypadku danych telemetrycznych stwierdza się, że system skonteneryzowany oferuje lepsze wizualizacje z użyciem bazy danych InfluxDB niż analogiczne rozwiązanie w systemie chmurowym z użyciem AWS Timestream.

Do głównych wad powstałych rozwiązań, które powinny zostać uwzględnione do ulepszenia w dalszym rozwoju jest brak rozbudowanego procesu CI/CD w systemie wykorzystującym Dockera.

Prace wdrożeniowe mogłyby zostać bardziej zautomatyzowane i sformalizowane z użyciem technologii Jenkins. Pozwoliło by to na lepsze administrowanie przebudową kontenerów. Obszarem do rozwoju jest również obsługa i monitorowanie logów aplikacyjnych. W obecnym rozwiązaniu brakuje możliwości do przeszukiwania logów z użyciem języka zapytań. Kolejnym aspektem do ulepszenia jest zdefiniowanie procesu w reakcji na powstałe alarmy. Obecnie administrator systemu jest notyfikowany o powstałym alarmie w poziomu kokpitu jednakże zalecane jest zautomatyzowanie czynności potrzebnych do obsługi alarmów. Błędne komunikaty, które powodują wywołanie alarmu są zapisywane w systemie i mogą posłużyć jako wyzwalacz do procesu obsługi wadliwych wiadomości. Potencjalnym ulepszeniem systemu chmurowego byłoby wykorzystanie narzędzia Terraform zamiast usługi AWS CloudFormation, aby przygotować rozwiązanie do wsparcia innych dostawców usług chmurowych.

Na nakłady pracy wymagane do wykonania omawianych prototypów składają się aktywności związane z projektowaniem, nauką nowych technologii, implementacją logiki biznesowej, integracją usług, testowaniem i naprawą błędów, które sumarycznie trwały około 12 miesięcy. Proces ten wymagał przeglądu literatury i dokumentacji technicznej, eksperymentowania i rewizji przyjętych założeń. Wymagającą częścią projektu było zdefiniowanie całego ekosystemu prototypów w plikach wdrożeniowych *Infrastrucutre as a Code*. Liczność stosowanych narzędzi wymagała dokonania wielu integracji co zwiększyło złożoność prac. Powstałe prototypy mają duże możliwości do rozbudowy lub adaptacji do nowych zastosowań. Jest to bardzo istotne z perspektywy ponownego wykorzystania kodu. Rezultaty wykonanych prac implementacyjnych są zadawalające, gdyż stawiane cele biznesowe projektowanych systemów zostały spełnione wraz z zakładanym aspektem dydaktycznym dotyczącym poszerzenia wiedzy i umiejętności praktycznych w omawianej dziedzinie technologii chmurowej i konteneryzacji.

Podsumowanie

Rozwój technologii usług chmurowych oraz konteneryzacji znacząco usprawnił proces budowy skalowalnych i wydajnych systemów informatycznych co zostało potwierdzone przy implementacji dwóch prototypów systemów IoT. Skorzystanie z narzędzi oferowanych przez AWS pozwoliło skupić się na budowie logiki systemu zamiast na żmudnej konfiguracji infrastruktury. Sprawne poruszanie się po ekosystemie AWS wymaga wprawy i doświadczenia. Stąd dla początkującego użytkownika budowa systemu w oparciu o usługi chmurowe może okazać się trudnym zadaniem. Jednakże implementacja platformy wykorzystującej skonteneryzowane aplikacje okazuje się wyzwaniem jeszcze większym ze względu na konieczność uwzględniania wszelkich konfiguracji oraz samodzielnej implementacji niektórych funkcjonalności, które w usługach chmurowych są domyślnie wbudowane. Ewolucja technologiczna konteneryzacji przyczyniła się do znacznego rozwoju rozwiązań chmurowych i to te rozwiązania najszybciej zyskują na popularności przy tworzeniu nowych systemów.

W wyniku realizacji niniejszej pracy została dokonana obszerna analiza charakterystyk dziedziny IoT. Ponadto dokonano przeglądu możliwości i ograniczeń rozwiązań chmurowych oraz platformy Docker. Skutkowało to wykonaniem dwóch średnio-zaawansowanych prototypów systemów informatycznych, które realizowały stawiane cele biznesowe wraz możliwością skalowania, wysokiej dostępności oraz odporności na awarie. Miało to na celu zdobycie praktycznych umiejętności w tych bardzo perspektywicznych obszarach technologii informatyki. Wiedza na temat konteneryzacji znacznie zwiększa zrozumienie mechanizmów odpowiedzialnych za działanie usług chmurowych.

Zbudowane systemy mają potencjał do dalszej rozbudowy i usprawnień. Dotyczy to dodania nowych funkcjonalności biznesowych poprzez implementację dedykowanych API oraz poprawę przetwarzania logów aplikacyjnych i alarmów, szczególnie w systemie skonteneryzowanym. Platformy zostały zaprojektowane w myślą o łatwej rozszerzalności i dodawaniu nowych komponentów poprzez luźne powiązania aplikacji. Liczne kokpity administracyjne pozwalają monitorować proces produkcji urządzeń i dokonywać analiz danych telemetrycznych.

Wykaz rysunków

Wykaz tabel

Wykaz listingów

Prace cytowane

[1]. Barrie Sosinsky. „Cloud Computing Biblie”. 2010. ISBN: 9781118023990

[2]. Marcin Sikorski. „Internet rzeczy”. 2020. ISBN: 9788301208400

[3]. Robert C. Martin. „Czysta architektura. Struktura i design oprogramowania. Przewodnik dla profesjonalistów”. 2018 . ISBN: 9788328342262

[4]. Robert C. Martin. „Czysty kod. Podręcznik dobrego programisty”. 2015. ISBN: 9788328313996

[5]. Gamma Erich , Helm Richard , Johnson Ralph , Vlissides John. „Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku”. 2021. ISBN: 9788328386099

[6]. „Difference between AMQP vs MQTT”. URL: https://www.educba.com/amqp-vs-mqtt/ (02.2022)

[7]. „Cloud Services Comparison 2022 – market share, main differences”. URL: https://www.future-processing.com/blog/cloud-services-comparison-market-share-main-differences/ (03.2022)

[8]. „Serverless on AWS. Build and run applications without thinking about servers”. URL: https://aws.amazon.com/serverless/. (03.2022)

[9]. „The Basics Of Cloud Computing”. URL: https://uniserveit.com/blog/the-basics-of-cloud-computing (03.2022)

[10]. „Use containers to Build, Share and Run your applications” . URL: https://www.docker.com/resources/what-container . (03.2022)

[11]. „Skalowanie aplikacji — jak efektywnie zarządzać danymi”. URL: https://studiosoftware.pl/blog/skalowanie-aplikacji-jak-efektywnie-zarzadzac-danymi (03.2022)

[12]. Peter Mell, Timothy Grance. „The NIST Definition of Cloud Computing”. NIST Special publication 800-145. 2012.

[13]. „Insecure Transportation Security Protocol Supported (TLS 1.0)”. URL: https://www.invicti.com/web-vulnerability-scanner/vulnerabilities/insecure-transportation-security-protocol-supported-tls-10/ (03.2022)

[14]. „Guide to MQTT” . URL: https://cedalo.com/guide-to-mqtt/ (03.2022)

[15]. „What is the internet of things (IoT)?”. URL: https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT (03.2022)

[16]. „Migracja aplikacji do chmury – z głową!” . URL: https://www2.deloitte.com/pl/pl/pages/risk/articles/migracja-aplikacji-do-chmury.html (03.2022)

[17]. „Korzyści z migracji firmy do chmury”. URL: https://www.jcommerce.pl/jpro/artykuly/korzysci-z-przeniesienia-firmy-do-chmury (03.2022)

[18]. „Reliability and high availability in cloud computing environments: a reference roadmap”. URL: https://hcis-journal.springeropen.com/articles/10.1186/s13673-018-0143-8 (03.2022)

[19]. „Availability Regions and Zones for AWS, Azure & GCP”. URL: https://www.bmc.com/blogs/cloud-availability-regions-zones/ (03.2022)

[20]. „AWS Service Level Agreements (SLAs)” . URL: https://aws.amazon.com/legal/service-level-agreements/?aws-sla-cards.sort-by=item.additionalFields.serviceNameLower&aws-sla-cards.sort-order=asc&awsf.tech-category-filter=*all (03.2022)

[21]. „Terraform Official Documentation”. URL: https://www.terraform.io/ (03.2022)

[22]. „AWS Pricing: 5 models & pricing for 10 popular AWS services”. URL: https://spot.io/resources/aws-pricing-5-models-pricing-for-10-popular-aws-services/ (03.2022)

[23]. „Shared Responsibility Model”. URL: https://aws.amazon.com/compliance/shared-responsibility-model/ (03.2022)

[24]. „Azure for US Government”. URL: https://azure.microsoft.com/en-us/global-infrastructure/government/#why-azure (03.2022)

[25]. „Security best practices in IAM”. URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html (03.2022)

[26]. „Compare AWS Support Plans”. URL: https://aws.amazon.com/premiumsupport/plans/ (03.2022)

[27]. „Official Docker documentation”. URL: https://www.docker.com/resources/what-container/ (04.2022)

[28]. „Best practices writing Dockerfile”. URL: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/ (04.2022)

[29]. „Docker security overview”. URL: https://docs.docker.com/engine/security/ (04.2022)

[30]. „Docker security best practices”. URL: https://blog.gitguardian.com/how-to-improve-your-docker-containers-security-cheat-sheet/ (04.2022)

[31]. „Advantages of Event Driven Architecture”. URL: https://developer.ibm.com/articles/advantages-of-an-event-driven-architecture/ (04.2022)

[32]. „5 reasons to adopt CICD”. URL: https://fullscale.io/blog/cicd-reasons/ (04.2022)

[33]. „Wzorce projektowe”. URL: https://refactoring.guru/pl/design-patterns (04.2022)

[34]. „RTO i RPO - kluczowe parametry disaster recovery”. URL: https://www.beyond.pl/baza-wiedzy/poradniki/rto-rpo-disaster-recovery/ (04.2022)

[35]. „Dokumentacja Elasticsearch”. URL: https://www.elastic.co/what-is/elasticsearch (04.2022)

[36]. „Failure Injection Testing” URL: https://netflixtechblog.com/fit-failure-injection-testing-35d8e2a9bb2 (04.2022)

[37]. „AWS XRay with API Gateway”. URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-using-xray-maps.html (04.2022)

[38]. „Developer Guide - Boto3 retries”. URL: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/retries.html (04.2022)

[39]. „AWS CLI configuration basics”. URL: https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html (04.2022)

[40]. „Język programowania Python”. URL: https://jaki-jezyk-programowania.pl/technologie/python/ (04.2022)

[41]. „AWS Lambda cold starts comparison”. URL: https://filia-aleks.medium.com/aws-lambda-battle-2021-performance-comparison-for-all-languages-c1b441005fd1 (04.2022)

[42]. „CAP theorem”. URL: https://www.bmc.com/blogs/cap-theorem/ (04.2022)

[43]. „NoSQL databases brief history”. URL: https://www.dataversity.net/a-brief-history-of-non-relational-databases/ (04.2022)

[44]. „InfluxDB in CAP spectrum”. URL: https://www.influxdata.com/blog/influxdb-clustering-design-neither-strictly-cp-or-ap/ (04.2022)

[45]. „Best Jenkins Alternatives”. URL: https://www.guru99.com/jenkins-alternative.html (05.2022)

[46]. „AWS CodePipeline”. URL: https://aws.amazon.com/codepipeline/ (05.2022)

[47]. „Azure Pipelines”. URL: https://azure.microsoft.com/pl-pl/services/devops/pipelines/ (05.2022)

[48]. „AWS Signature version 4”. URL: https://docs.aws.amazon.com/general/latest/gr/signature-version-4.html (05.2022)

[49]. „Hive partitions in AWS”. URL: https://docs.aws.amazon.com/athena/latest/ug/partitions.html (05.2022)

[50]. „AWS S3 Performance Best Practices”. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/optimizing-performance.html (05.2022)

[51]. „Unix epoch time”. URL: https://kb.narrative.io/what-is-unix-time (05.2022)

[52]. „Sharing objects using presigned URLs”. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/ShareObjectPreSignedURL.html (05.2022)

Dodatki

Dodatek A: Słownik użytej terminologii i skrótów

API – (ang. *Application Programming Interface*) zdefiniowany interfejs programistyczny udostępniający zasoby

AQMP – (ang. *Advanced Message Queue Protocol*) rozbudowany protokół komunikacyjny warstwy aplikacji

AWS – (ang. *Amazon Web Services*) – dostawca usług chmurowych, lider na rynku

CI/CD – (ang. *Continuous integration Continuous Delivery*) metodologia dostarczania i wdrażania oprogramowania w sposób ciągły

CLI – (ang. *Command Line Interface*) interfejs wiersza poleceń

DoS – (ang. *Denial of Service*) metoda ataku na system skutkująca utratą dostępności usługi

IaaC – (ang. *Infrastructure as a Service*) typ usług chmurowych oferujących wynajęcie infrastruktury

IDE – (ang. *Integrated Development Environment*) zintegrowane środowisko programistyczne

Internet – globalna sieć komputerowa

IoT – (ang. *Internet of Things*) internet rzeczy – system integrujący wiele rodzajów urządzeń, aby wykreować większą wartość poprzez współpracę tych urządzeń

KPI – (ang. *Key Performance Indicator*) kluczowy wskaźnik funkcjonowania systemu służący do monitoringu

MQTT – (ang. *Message Queuing Telemetry Transport*) lekki protokół komunikacyjny dla danych telemetrycznych

NIST – (ang. *National Institute of Standards and Technology*) instytut standaryzujący zagadnienia technologiczne

PaaS – (ang. *Platform as a Service*) typ usług chmurowych oferujących skorzystanie z konfigurowalnych platform

PoC – (ang. *Prove of Concept*) prototyp dowodzący słuszność wybranego podejścia

RPO – (ang. *Recovery Point Objective*) – wskaźnik definiujący punkt w czasie, do którego będzie przywrócona struktura danych systemu po awarii

RTO – (ang. *Recovery Time Objective*) – wskaźnik definiujący wymagany czas przywrócenia funkcjonowania systemu po awarii

SaaS – (ang. *Software as a Service*) typ usług chmurowych oferujących gotowe produkty oprogramowania

SDK – (ang. *Software Development Kit*) zbiór zdefiniowanych bibliotek programistycznych umożliwiające korzystanie z danej platformy

Serverless – określenie grupy usług nie wymagających zarządzania serwerami na których się znajdują

SLA – (ang. *Service Level Agreement*) umowa pomiędzy dostawcą usług a klientem dotyczące dostępnością i jakością usług

On-Premise – system wdrożony na własnej infrastrukturze przedsiębiorstwa, wymagający pełnej administracji i utrzymania

Dodatek B: Modele danych generowane przez *Device Manufacturing Platform*

Przykłady komunikatów MQTT z pomiaru telemetrycznego


```
{
        "uuid": "11be2e19-67e4-442b-83eb-8ba70747a712",
        "sensor_id": "RES_2",
        "station_id": "MAN_ST_2",
        "id_sfx": "2",
        "timestamp": 1652128033,
        "value": 102,
        "unit": "Ohm",
        "iso_datetime": "2022-05-09T20:27:13.571307"
}
```

```
{
        "uuid": "2830c23c-9731-4884-87e2-48b2d70abda2",
        "sensor_id": "TEMP_2",
        "station_id": "MAN_ST_2",
        "id_sfx": "2",
        "timestamp": 1652128033,
        "value": 28,
        "unit": "C",
        "iso_datetime": "2022-05-09T20:27:13.269520"
}
```

```
{
        "uuid": "df339388-baae-40af-975e-607c38e96c0c",
        "sensor_id": "VOLT_2",
        "station_id": "MAN_ST_2",
        "id_sfx": "2",
        "timestamp": 1652128033,
        "value": 336,
        "unit": "mV",
        "iso_datetime": "2022-05-09T20:27:13.470710"
}
```

```
{
        "uuid": "87d35d80-ef8e-4431-9c44-006d08e040b9",
        "sensor_id": "PRES_2",
        "station_id": "MAN_ST_2",
        "id_sfx": "2",
        "timestamp": 1652128033,
        "value": 1028,
        "unit": "hPa",
        "iso_datetime": "2022-05-09T20:27:13.370103"
}
```

Przykład pliku logu produkcyjnego urządzenia


```
{
 "device_id": "e81f48bb-9b4d-48f1-8598-02debd6bec13",
 "provider_id": "PROVIDER_1",
 "measurements": {
  "MAN_ST_1": [{
     "uuid": "e8c6bb80-5bae-46ff-981c-b0e0d74218cb",
     "sensor_id": "TEMP_1",
     "station_id": "MAN_ST_1",
     "id_sfx": "1",
     "timestamp": 1630744655.85491,
     "value": 84,
     "unit": "C",
     "iso_datetime": "2021-09-04T08:37:35.854913"
    },
    {
     "uuid": "d7365b2a-b6e4-477b-b16e-51b0dfe1d80d",
     "sensor_id": "PRES_1",
     "station_id": "MAN_ST_1",
     "id_sfx": "1",
     "timestamp": 1630744655.9557548,
     "value": 983,
     "unit": "hPa",
     "iso_datetime": "2021-09-04T08:37:35.955756"
    },
    {
     "uuid": "79bbd470-9b92-4738-96c5-40c3893bbf4d",
     "sensor_id": "VOLT_1",
     "station_id": "MAN_ST_1",
     "id_sfx": "1",
     "timestamp": 1630744656.0561564,
     "value": 349,
     "unit": "mV",
     "iso_datetime": "2021-09-04T08:37:36.056158"
    },
    {
     "uuid": "e4b1181d-d185-4d7b-a329-3e2de804ba98",
     "sensor_id": "RES_1",
     "station_id": "MAN_ST_1",
     "id_sfx": "1",
     "timestamp": 1630744656.156608,
     "value": 97,
     "unit": "Ohm",
     "iso_datetime": "2021-09-04T08:37:36.156610"
    }
  ],

  "crypto_key":
"9f26503369324b04688d232a4856f723e306c07a05782ded22ab10dc3ab390c69a354a3ae1e8a82406feb0606f3ddf
a3c685e066aa107bab20286f3b6dde063fee2d97729c08cc5f44847552f4a8c87a18e81d8c4bdc6a85ea66865aaace9
253c88ccc1bbf5a1690d18ec07a9461e05b872099e7a6242011abd3a3f82eb75a9f82290651664da5c5dbf7d8d216c
d8dd28374c58936bae32159e0e209b6221f6fccdca1f6af5038e748d47c1596396c01e45b03952f563b07801768f90
393bb2dc2438ad8d6ea785d86d2f33b3ba32c16f2f179c70223e99993ff746c534872af3decd6ac2d96b61e435a136d
72debc440cd476aa2270acc5a0902ab2f82ee8f7",
  "qa_testing": {
    "fw_crypto_key": "PASSED",
    "sensors_measurements": "PASSED"
  }
 }
}
```

Dodatek C: Model danych rekordu *Device Info Dynamo DB*

Model rekordu

```
{
  "creation_date": ISO Datetime
  "device_id": uuid4 String, Partition Key of Dynamo DB
  "full_file_path": String
  "fw_crypto_key": String
  "fw_version": String
  "input_bucket_name": String
  "is_log_structure_valid": Boolean
  "is_provider_qa_testing_valid": Boolean
  "is_sensor_measurements_valid": Boolean
  "last_modified_date": ISO Datetime
  "log_file_name": String
  "log_file_version_id": String
  "production_date": ISO Datetime
  "provider_id": String
  "status": String – [VALID | INVALID]
}
```

Przykład rekordu


```
{
  "creation_date": "2021-10-08 12:56:45.631529",
  "device_id": "034bc787-5e6b-4cb9-93cf-6d879ebbf453",
  "full_file_path": "year=2021/month=10/day=08/Log_034bc787-5e6b-4cb9-93cf-6d879ebbf453.json",
  "fw_crypto_key":
"4da27811320718fcc233a95ca1940942e5020ccafcdebe7e34dede6110757403f4d3f97a9e1540bc3ab621fa552899a58
2dddf7551c9a66f859372908eda30b5912e98e94617111fb0e7f49eebe03136f5e4f0e499e1bcc4ac45482f9cd658ad0db
84a953f62d639afd1632c411f8d521e5d5c7ceab4f405ac654e13876ff1a08c9305ae7018dda05740324e296ad9930c7b5
1c4385e3d8777f1de22521aa31233d710389f945919715f382105becda06e27cb2f789d57d3e9ac9fa14540309f3c1ec99
797aff26fcf9efb05c8c8e5c5955fe657ba6a8927276d0e1b3b8f14736ccf7fc4cbf69a935ef6e21b9e81ab21f905ad59251
c326f63079b5745bb4621",
  "fw_version": "1.2.5",
  "input_bucket_name": "dm-iot-def-processinginput-dev",
  "is_log_structure_valid": true,
  "is_provider_qa_testing_valid": true,
  "is_sensor_measurements_valid": true,
  "last_modified_date": "2021-10-08 12:56:45.631529",
  "log_file_name": "Log_034bc787-5e6b-4cb9-93cf-6d879ebbf453.json",
  "log_file_version_id": "vCqtRq_swFZk63nznK2Fertg2MvhTRfN",
  "production_date": "2021-10-08T12:56:40.760468",
  "provider_id": "PROVIDER_2",
  "status": "VALID"
}
```