

UNIWERSYTET EKONOMICZNY WE WROCŁAWIU
WYDZIAŁ INŻYNIERYJNO-EKONOMICZNY

Daniel Aniszkiewicz

**PODEJŚCIE SERVERLESS W TWORZENIU
APLIKACJI INTERNETOWYCH Z
WYKORZYSTANIEM TECHNOLOGII AMAZON WEB
SERVICES**

Praca magisterska

Promotor:
dr Marcin Hernes

**Katedra
Rachunkowości,
Controllingu,
Informatyki i Metod
Ilościowych**

Pracę akceptuję i wnioskuję o jej dopuszczenie
do dalszych etapów postępowania

.....
podpis promotora

Wrocław 2019

Spis treści

ABSTRACT.....	3
WSTĘP.....	3
1. EWOLUCJA W TWORZENIU APLIKACJI INTERNETOWYCH.....	4
1.1 . APLIKACJE JAKO MONOLIT.....	4
1.2. APLIKACJE OPARTE O MIKROSERWISY.....	10
1.3. APLIKACJE OPARTE O FUNKCJE (FAAS).....	21
2. POJĘCIE SERVERLESS.....	27
2.1. WSTĘP.....	27
2.2. ZASTOSOWANIA ARCHITEKTURY SERVERLESS.....	31
2.3. PRZYKŁADY ROZWIĄZAŃ OFERUJĄCYCH PODEJŚCIE SERVERLESS.....	34
2.3.1. <i>A Cloud Guru</i>	34
2.3.2. <i>Custom Ink</i>	35
2.3.3. <i>SundaySky</i>	36
3. AMAZON WEB SERVICES.....	38
3.1. CHMURA OBLICZENIOWA AMAZON WEB SERVICES.....	38
3.2. BEZPIECZEŃSTWO.....	42
3.3 . WYBRANE USŁUGI AWS.....	44
4. PROJEKT APLIKACJI WIRTUALNEGO RYNKU SPRZEDAŻY TORTÓW.....	52
4.1. WYKORZYSTANE TECHNOLOGIE.....	52
4.1.1. <i>Część Wizualna</i>	52
4.1.2. <i>Część serwerowa</i>	54
4.2. DOSTĘP DO APLIKACJI.....	61
4.3. OPIS FUNKCJONALNOŚCI APLIKACJI.....	63
ZAKOŃCZENIE.....	69
LITERATURA.....	70
SPIS ILUSTRACJI:.....	73
SPIS TABEL:.....	74

Abstract

Serverless approach in creating web applications.

The purpose of this master thesis is to show a new approach in creating modern, stable and robust web applications based on the Serverless approach. In the aforementioned master thesis, the focus was on the evolution of creating applications, first as a monolith with all pros and cons, in relation to this approach. Second as a microservices approach, as a new quality opposite to monolith applications, third, Function as a Service (FaaS) approach. In the second chapter, the whole theory behind Serverless is briefly described, all positive and negatives aspects. In the third chapter, introduced cloud computing company Amazon Web Services (AWS), starting from what cloud computing is, and finishing on chosen AWS services. In the last fourth chapter practical project, based on previously described AWS services, to create a web application based on Serverless architecture.

Wstęp

Rozwiązanie Serverless jest stosowane przez wiele firm na całym świecie, poprzez coraz większą popularyzację chmury obliczeniowej, w celu rozwiązywania problemów implementacyjnych oprogramowania, jak i wdrażania logiki biznesowej przedsiębiorstw. Podejście Serverless zapewnia niezawodność wykonywanych działań, skalowalność rozwiązań, stabilność oraz niskie koszty przetwarzania danych, poprzez stosowanie opłat tylko za wykorzystane zasoby, a nie za gotowość usług.

Celem pracy jest opracowanie projektu aplikacji z wykorzystaniem podejścia Serverless w technologii Amazon Web Services. Jako przykład przyjęto wirtualny rynek sprzedaży tortów.

W pierwszym rozdziale skupiono uwagę na analizie ewolucji w podejściu do tworzenia aplikacji internetowych. Scharakteryzowano architekturę monolityczną. Następnie przedstawiono podejście architektury mikroserwisów. W ostatnim punkcie rozdziału pierwszego, poświęcono uwagę podejściu Function As A Service (FaaS), czyli funkcji jako serwis. W rozdziale drugim, przedstawiono istotę koncepcji Serverless, zarówno pod względem korzyści, zagrożeń, szans oraz minusów związanych z tą koncepcją. W rozdziale trzecim, skupiono uwagę na właściwościach chmurze obliczeniowej Amazon Web Services

(AWS). Przedstawiono w jaki sposób ona funkcjonuje, w jaki sposób zapewniane jest bezpieczeństwo danych oraz opisano najpopularniejsze usługi AWS. W ostatnim rozdziale opracowano projekt praktyczny, w postaci aplikacji internetowej do sprzedaży tortów, opartej o architekturę Serverless. Stworzono interfejs użytkownika oraz połączono usługi AWS takie, jak: Amazon Simple Storage Service (S3), Amazon Simple Email Service (SES), Amazon Cognito, Amazon DynamoDB i Amazon AppSync.

W pracy zastosowano takie metody badawcze, jak analiza literatury przedmiotu, studium przypadku i metodę eksperymentalną.

1. Ewolucja w tworzeniu aplikacji internetowych

1.1. Aplikacje jako monolit

W ciągu ostatnich kilkunastu lat aplikacje internetowe zaczęły odgrywać coraz ważniejszą rolę w życiu ludzi. Niezależnie od tego, czy są to duże aplikacje, takie jak sieci społecznościowe, średnie, takie jak strony bankowości elektronicznej, czy też mniejsze, takie jak systemy księgowości online lub narzędzia do zarządzania projektami dla małych przedsiębiorstw, zależność od tych usług wyraźnie rośnie. Wraz z pojawieniem się sieci internetowej wykorzystywanej jako platforma dla aplikacji, rozmyła się granica między pojęciami strony internetowej i aplikacji internetowej. Strony internetowe były zazwyczaj oparte na treści - istniały po to, aby przekazywać informacje, często z pewną interaktywnością. Wiele frameworków webowych, powstało, aby przyspieszyć tworzenie aplikacji internetowych w różnych technologiach. Naturalnym trendem dla nowych firm z branży high-tech, takich jak Amazon czy Netflix, jest budowanie nowego oprogramowania z wykorzystaniem mikroserwisów. Uzyskują one dużą przewagę oprogramowania zorientowanego na usługi, aby rozszerzyć zakres swoich nowych produktów. Problem polega na tym, że nie wszystkie firmy mogą planować swoje produkty z góry. Zamiast planować wszystkie rozwiązania z wyprzedzeniem, firmy te, budują oprogramowanie w oparciu o zdobywaniu doświadczenia budując małe rzeczy, które się z czasem stają się ważnymi komponentami w projektach. Nierzadko zdarza się, że firmy posiadają dwa lub więcej dużych komponentów oprogramowania: stronę internetową skierowaną do użytkownika i

panel do administracji wewnętrznej narzędziami. Jest to zazwyczaj znane jako monolityczna architektura oprogramowania.¹

Słowo "monolit" było pierwotnie używane przez starożytnych Greków do opisanie pojedynczego, górskiego bloku kamiennego. Chociaż słowo to jest dziś używane szerzej, idea pozostaje ta sama - monolityczny produkt programowy jest pojedynczą, niepodzielną jednostką, która generalnie rozrasta się do dużych rozmiarów.² Monolit jest pojedynczym kodem źródłowym z pojedynczą aplikacją, która wykonuje wszystkie funkcje tego systemu. Aplikacja monolityczna jest aplikacją o wysokim stopniu złożoności, która wykonuje całą grupę zadań w celu wdrożenia całego przypadku użytkowego. Nie składa się z żadnych jednostek logicznych, które można zidentyfikować. Pełni rolę realizatora całych funkcji, nie zaś tylko konkretnych zadań wewnątrz tych funkcji. Aplikacje monolityczne są konstruowane bez modułowości. Może być skalowalna, ale skala jest osiągana poprzez powielanie całego systemu na innych maszynach, a nie poszczególnych jego komponentów. Chociaż może posiadać API w celu integracji z innymi systemami, wewnętrzne komponenty systemu komunikują się ze sobą poprzez wywołania metod lub funkcji. Monolit oznacza skomponowanie wszystkiego w jednym kawałku. Komponentami wewnętrznymi mogą być:

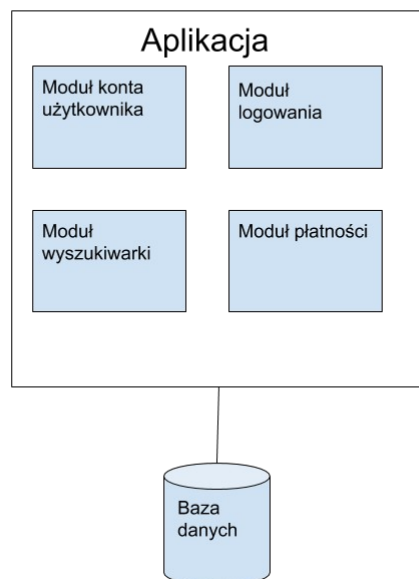
- Autoryzacja - odpowiedzialna za autoryzację użytkownika.
- Prezentacja - odpowiedzialna za obsługę żądań HTTP i odpowiadanie za pomocą HTML lub JSON/XML (dla API usług internetowych).
- Logika biznesowa - logika biznesowa aplikacji.
- Warstwa bazy danych - obiekty dostępu do danych odpowiedzialne za dostęp do bazy danych.
- Integracja aplikacji - integracja z innymi usługami (np. poprzez komunikatory lub REST API). Albo integracja z innymi źródłami danych.
- Moduł powiadamiania - odpowiedzialny za wysyłanie powiadomień e-mail w razie potrzeby.³

¹ Chris Northwood, "The Full Stack Developer Your Essential Guide to the Everyday Skills Expected of a Modern Full Stack Web Developer", Apress, 2018 Manchester, s. 82

² <https://www.thorntech.com/2017/12/microservices-vs-monoliths-whats-right-architecture-software/>

³ Alexandru Jecan, Java 9 Modularity Revealed Project Jigsaw and Scalable Java Applications, Apress, 2017 Munich, s. 22

Monolit



Rysunek 1. Schemat aplikacji monolitycznej. Opracowanie własne.

Rozważono przykład aplikacji e-commerce przedstawionej na rysunku nr 1, która autoryzuje klienta, przyjmuje zamówienie, sprawdza zapasy produktów, autoryzuje płatności i wysyła zamówione produkty. Aplikacja ta składa się z kilku komponentów, w tym interfejsu użytkownika sklepu internetowego dla klientów (widok strony internetowej sklepu) wraz z kilkoma usługami typu backend do sprawdzania zapasów produktów, autoryzacji i pobierania opłat za płatności i zamówień wysyłkowych.

Główne zalety zastosowania architektury monolitycznej wynikają z jej prostoty. Łatwo je opracować, wdrożyć i skalować. W ramach architektury monolitycznej, łatwiej jest również, w stosunku do architektury mikroserwisów, przetestować aplikację ze względu na czytelność kodu zawartego w jednej strukturze. Żądania bazodanowe mają jedną bazę danych⁴. Główną zaletą architektury monolitycznej jest to, że większość aplikacji ma zazwyczaj wiele aspektów o charakterze przekrojowym, takich jak logowanie, ograniczanie prędkości i funkcje bezpieczeństwa, takie jak ścieżki audytowe i ochrona przed atakami typu DOS. Gdy wszystko realizowane jest tę samą aplikacją, łatwo jest podłączyć komponenty do tych przekrojowych rozwiązań. Kolejną kwestią jest mniejsza ilość kosztów operacyjnych, tylko dla jednej aplikacji należy skonfigurować logowanie, monitorowanie i testowanie. Jest

⁴ Martin Kleppmann, "Designing Data-Intensive Applications", O'Reilly Media, Inc, 2017

to również na ogół mniej skomplikowane w odniesieniu do wdrożenia. Może to również przynieść korzyści w zakresie wydajności, ponieważ dostęp do współdzielonej pamięci jest szybszy niż komunikacja między procesami. Monolity to wygodny sposób na rozpoczęcie nowego projektu oprogramowania z minimalnym zakresem pracy przy konfiguracji na serwerze lub w środowisku chmury. Podczas gdy złożoność może rosnąć w czasie, odpowiednie zarządzanie bazą kodu może pomóc w utrzymaniu produktywności przez cały okres życia monolitycznej aplikacji. W początkowych fazach projektu podejście oparte na architekturze monolitu działa dobrze i w zasadzie większość dużych i udanych aplikacji, które istnieją obecnie, została uruchomiona jako monolit. Architektura monolityczna działała z powodzeniem przez wiele dziesięcioleci. Wiele dużych aplikacji korporacyjnych dużych firm nadal jest wdrażanych jako monolit. Jednak wraz ze zmieniającym się rynkiem i pojawieniem się nowych technologii nastąpiła zmiana paradygmatu w sposobie działania branży IT. Istnieją pewne poważne problemy z architekturą monolityczną, które większość firm stara się rozwiązać.⁵

Gdy aplikacja staje się duża, a zespół powiększa się, podejście to ma wiele wad, które stają się coraz bardziej znaczące. Duża monolityczna baza kodu odstrasza programistów, zwłaszcza tych, którzy są nowi w zespole. Aplikacja może być trudna do zrozumienia i modyfikacji. W rezultacie, spada dynamika rozwoju. Ponadto nie ma sztywnych granic pomiędzy modułami i modularność zanika wraz z upływem czasu. Ponieważ może być trudno zrozumieć, jak prawidłowo zaimplementować zmianę, jakość kodu spada z upływem czasu. Kolejnym istotnym problemem jest fakt przeciążania narzędzi programistycznych. Im większy jest kod źródłowy aplikacji, tym wolniej ładują się narzędzia do tworzenia kodu źródłowego, co potem przekłada się na spadek produktywności programistów w zespole. Wraz ze wzrostem aplikacji, często zdarza się przeciążanie serwera, na którym znajduje się aplikacja - im większa aplikacja tym dłużej trwa uruchomienie. Ma to ogromny wpływ na produktywność programistów z powodu marnowanego czasu oczekiwania na uruchomienie serwera. Ma to również wpływ na wdrożenie nowych funkcjonalności.

Nieustanne wdrażanie jest trudne - duża monolityczna aplikacja jest również przeszkodą dla częstych wdrożeń. Aby zaktualizować jeden komponent, należy przeinstalować całą aplikację. Przerwie to realizację zadań w tle, bez względu na to, czy zmiana wpłynie na nie. Istnieje również szansa, że komponenty, które nie zostały zaktualizowane, nie uruchomią się poprawnie. W rezultacie wzrasta ryzyko związane z opóźnieniami, co zniechęca do częstych aktualizacji. Jest to szczególnie problem dla programistów interfejsów użytkownika,

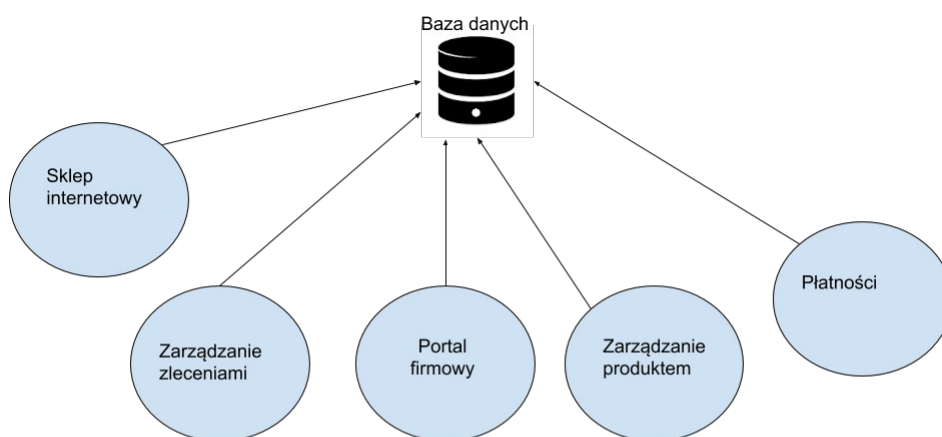
⁵ <https://www.webdesignerdepot.com/2018/05/monolith-vs-microservices-which-is-the-best-option-for-you/>

ponieważ zazwyczaj muszą oni szybko i często dokonywać napraw. Kolejnym istotnym problemem jest skalowanie. Skalowanie aplikacji może być trudne - monolityczna architektura może skalować się tylko w jednym wymiarze, wraz ze wzrostem wolumenu transakcji, uruchamiając więcej kopii aplikacji. Niektóre chmury mogą nawet dynamicznie dostosowywać liczbę instancji w zależności od obciążenia. Ale z drugiej strony, ta architektura nie może skalować się wraz ze wzrostem wolumenu danych. Każda kopia instancji aplikacji będzie miała dostęp do wszystkich danych, co zmniejsza efektywność buforowania i zwiększa zużycie pamięci oraz ruch wewnątrz jak i na zewnątrz. Ponadto, różne komponenty aplikacji mają różne wymagania dotyczące zasobów - jeden z nich może wymagać dużego nakładu pracy procesora, podczas gdy inny może wymagać dużej ilości pamięci. Z monolityczną architekturą nie możemy skalować każdego komponentu niezależnie. Gdy aplikacja osiągnie określoną wielkość, najczęściej dokonywany jest podział organizacji zespołów inżynierskich na podzespoły, które skupiają się na określonych obszarach funkcjonalnych. Na przykład, może być zespół od interfejsu użytkownika, zespół księgowy, zespół inwentaryzacyjny, itp. Problem z monolityczną aplikacją polega na tym, że uniemożliwia ona zespołom samodzielną pracę. Zespoły muszą koordynować swoje działania rozwojowe i przesunięcia. Dużo trudniej jest zespołowi dokonać zmiany i aktualizacji produkcji. Innym problemem w przypadku zastosowań aplikacji monolitycznych jest niezawodność. Błąd w dowolnym module (np. wyciek pamięci) może potencjalnie doprowadzić do załamania całego procesu. Ponadto, wszystkie instancje aplikacji są identyczne, błąd ten będzie miał wpływ na dostępność całej aplikacji. Wymaga się długoterminowego zaangażowania w dobór technologii - monolityczna architektura zmusza do wykorzystania zestawu technologii (a w niektórych przypadkach z konkretną wersją tej technologii), która została wybrana na początku rozwoju, przez długi okres. W przypadku aplikacji monolitycznej, może być trudno stopniowo wdrażać nowszą technologię. Ponadto, jeśli aplikacja korzysta z platformy, która staje się przestarzała, wówczas stopniowa migracja aplikacji do nowszej i lepszej platformy może stanowić wyzwanie. Aby zaadoptować nowszą platformę, trzeba przepisać całą aplikację, co jest ryzykownym przedsięwzięciem.⁶ Aplikacja monolityczna składa się zazwyczaj z wielu warstw: warstwy prezentacji, warstwy biznesowej, warstwy dostępu do danych i bazy danych. Z tym systemem warstw, korzystanie z wielu technologii dla jednej warstwy jest trudne. Czasami możliwe jest zastosowanie wielu technologii. Ale posiadanie takich ograniczeń jest zdecydowanie wadą dla każdego programisty. Jeśli pojawi się biblioteka, która może łatwiej rozwiązać konkretny problem,

⁶ <https://microservices.io/patterns/monolithic.html>

może nie być ona kompatybilna z technologią wykorzystywaną na tej warstwie. Łatwiej byłoby podzielić warstwy na różne mikroserwisy i mieć swobodę użytkowania. Czasami zmuszanie do dalszego korzystania z istniejącej technologii jest błędem, ponieważ technologia może być już stara.⁷

Rozważono kolejny przykład aplikacji monolitycznej:



Rysunek 2 Aplikacja do sprzedaży detalicznej online opracowana w oparciu o architekturę monolityczną. Opracowanie własne

Cała aplikacja detaliczna przedstawiona na rysunku nr 2 jest zbiorem kilku komponentów, takich jak zarządzanie zamówieniami, płatnościami, zarządzanie produktem itp. Każdy z tych komponentów oferuje szeroki zakres funkcjonalności biznesowych. Dodawanie lub modyfikowanie funkcjonalności komponentu było niezwykle kosztowne ze względu na jego monolityczny charakter. Ponadto, aby ułatwić spełnienie ogólnych wymagań biznesowych, komponenty te musiały się ze sobą komunikować. Komunikacja pomiędzy tymi komponentami była często budowana w oparciu o autorskie protokoły i standardy oraz opierała się na stylu komunikacji punkt-punkt. W związku z tym modyfikacja lub wymiana danego komponentu również była dość skomplikowana. Na przykład, jeśli przedsiębiorstwo handlu detalicznego chciałoby przełączyć się na nowy system zarządzania zamówieniami

⁷ Alexandru Jecan, "Java 9 Modularity Revealed Project Jigsaw and Scalable Java Applications", Apress, 2017 Munich, s. 13

przy jednoczesnym zachowaniu reszty, wymagałoby to dość dużych zmian w innych istniejących komponentach. W celu rozwiązania tego typu problemów stworzono mikroserwisy. Zamiast pojedynczej jednostki monolitycznej, aplikacje zbudowane przy użyciu mikroserwisów składają się z luźno połączonych, autonomicznych usług. Budując usługi, które robią jedną rzecz bez zarzutu, można zapobiec zastoju i entropii dużych zastosowań. Nawet w istniejących aplikacjach, można stopniowo wyodrębniać funkcje w niezależne usługi, aby uczynić cały system bardziej podatnym na utrzymanie.⁸

1.2. Aplikacje oparte o mikroserwisy

Zespoły składające się z programistów starają się tworzyć skuteczne i szybkie rozwiązania złożonych problemów. Pierwszym problemem, który zazwyczaj starają się rozwiązać jest, zebranie wszelkich wymagań od klienta. Jeżeli problem, który mają rozwiązać jest dobrze zrozumiany, rozwiązania problemów zaproponowane i zatwierdzone przez klienta, wdrożone do użytku, to projekt można uznać za udany. W dalszym etapie projekt stale się rozwija, problemy są na bieżąco rozwiązywane, budowane są nowe funkcje, a cała infrastruktura jest dostępna i działa bez zarzutu. Jednak wtedy rodzi się problem, ponieważ nawet najbardziej zdyscyplinowane zespoły mogą mieć problemy z utrzymaniem szybkiego tempa i elastyczności w obliczu rosnącego zapotrzebowania. W najgorszym przypadku, z początku stabilny projekt, staje się uciążliwy zarówno dla klienta, jak i zespołu programistów go utrzymujących. Zamiast w sposób zrównoważony dostarczać więcej wartości dla klientów, klienci są zmęczeni przestojami, zaniepokojeni długi czasem wdrażania projektu dla użytkowników końcowych, jak i coraz dłuższym czasem, poświęconym, aby dostarczać nowe funkcjonalności lub korygować istniejące awarie. Ani klienci, ani programiści nie są zadowoleni.⁹

Na wczesnym etapie, gdy projekt jest mało skomplikowany, a liczba programistów tworzących projekt jest niewielka, programiści zazwyczaj dzielą swoją pracę, przyczyniając się do tworzenia i utrzymywania kodu źródłowego. Wraz z rozwojem projektu, aby nadążyć zarówno nad tempem rozwoju oraz z dodawaniem nowych funkcjonalności, zatrudnianych jest coraz większa ilość programistów. Można wtedy zauważyć trzy istotne kwestie:

⁸ MORGAN BRUCE, PAULO A. PEREIRA, "Microservices in Action", Manning Publications Co., 2019 Nowy Jork, s. 4-5.

⁹ MORGAN BRUCE, PAULO A. PEREIRA, "Microservices in Action", Manning Publications Co., 2019 Nowy Jork, s. 3-4.

- **Następuje wzrost obciążenia operacyjnego.** Praca operacyjna to, ogólnie rzecz biorąc, praca związana z uruchomieniem i utrzymaniem działającej aplikacji. Zazwyczaj prowadzi to do zatrudniania inżynierów operacyjnych (administratorów systemów, inżynierów TechOps i tzw. inżynierów "DevOps"), którzy przejmują większość zadań operacyjnych, takich jak te związane ze sprzętem, monitoringiem jak i pracą pod telefonem w razie wystąpienia awarii.
- **Wzrost złożoności aplikacji.** Drugą rzeczą, która się dzieje, jest wynik prostej dedukcji - dodawanie nowych funkcji do aplikacji zwiększa zarówno liczbę linii kodu w aplikacji, jak i złożoność samej aplikacji. Z tego powodu coraz większą ilość pracy oraz czasu należy poświęcić na rozwijaniu aplikacji, coraz więcej rzeczy wewnątrz samej aplikacji posiada coraz więcej zależności. Implementacja nowych funkcjonalności musi być konsultowana przez cały zespół, mając na uwadze istniejące już zależności np. pomiędzy obiektami wewnątrz aplikacji.
- **Skalowanie aplikacji.** Trzecią z kwestii jest niezbędne horyzontalne i/lub wertykalne skalowanie aplikacji. Wzrost natężenia ruchu stawia znaczne wymagania w zakresie skalowalności i wydajności aplikacji, wymagając, aby aplikacja była obsługiwana przez większą liczbę serwerów. Dodaje się więcej serwerów, kopia aplikacji jest instalowana na każdym serwerze, a narzędzia służące do równoważonego obciążenia są instalowane w taki sposób, aby żądania użytkowników były rozprowadzane odpowiednio wśród serwerów, na których znajduje się aplikacja. Złożoność aplikacji stale rośnie, oraz setki (jeśli nie tysiące) testów muszą być napisane w celu zapewnienia, że każda wprowadzona zmiana (nawet zmiana jednej lub dwóch linii) nie narusza integralności istniejących tysięcy na tysiącach linii kodu. Rozwój i wdrożenie stają się istotnym problemem, testowanie staje się obciążeniem i przeszkodą we wdrożeniu nawet najistotniejszych poprawek, a dług technologiczny użytych technologii w projekcie stale rośnie.¹⁰ Aby zaradzić tym problemom, często popularnym rozwiązaniem jest użycie architektury mikroserwisów.

Aplikacja w oparciu o mikroserwisy to zbiór usług autonomicznych, z których każda z nich wykonuje jedno zadanie. Usługi te współpracują ze sobą, aby wykonywać bardziej skomplikowane operacje. Zamiast pojedynczego, złożonego systemu, można zbudować i

¹⁰ Susan J. Fowler, "Production-Ready Microservices", O'Reilly Media, Inc, 2017 Sebastopol, s. 18-21.

zarządzać zbiorem stosunkowo prostych usług, które mogą oddziaływać na siebie w złożony sposób. Usługi te współpracują ze sobą poprzez protokoły komunikacji technologiczno-agnostycznej, albo punkt-punkt, albo asynchronicznie. Klasyczna praktyka inżynierii oprogramowania wymaga wysokiej spójności i luźnego sprzężenia jako pożądaných właściwości dobrze zaprojektowanego systemu. System, który ma te właściwości, będzie łatwiejszy w utrzymaniu i bardziej elastyczny w obliczu zmian.¹¹ Jedną z kluczowych koncepcji architektury mikroserwisów jest to, że dana usługa musi być zaprojektowana w oparciu o możliwości biznesowe, tak aby dana usługa służyła konkretnym potrzebom i miała dobrze zdefiniowany zestaw obowiązków. Posiadanie autonomicznych usług jest najważniejszą siłą napędową realizacji architektury mikroserwisów. Mikroserwisy są opracowywane, wdrażane i skalowane jako niezależne obiekty. W przeciwieństwie do usług internetowych takich jak monolityczne aplikacje, usługi nie mają tego samego czasu wykonywania zadań. Raczej są one wdrażane jako oddzielne operacje.¹² Mówi się, iż wyróżnia się sześć głównych cech mikroserwisów, są to kolejno:

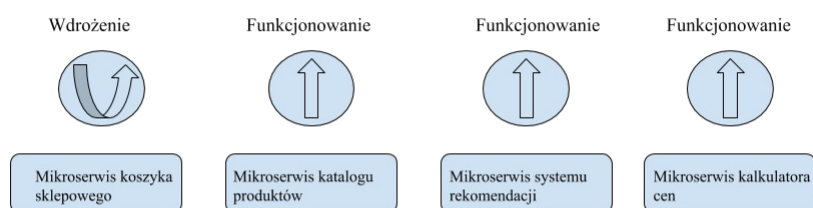
- 1) **Mikroserwis jest odpowiedzialny za jedną funkcję.** W przypadku mikroserwisów, stosujemy zasadę pojedynczej odpowiedzialności na poziomie usług.¹³ Ponadto, należy dążyć do tego, aby mikroserwis pełni realizował swoje możliwości, tak aby tylko jeden mikroserwis musiał się zmienić po zmianie właściwości. Istnieją dwa rodzaje zależności w architekturze mikroserwisów:
- 2) **Zależność biznesowa,** czyli to, co system robi, co przyczynia się do osiągnięcia celu systemu, np. śledzenie wózków zakupowych użytkowników lub obliczanie cen. A dobrym sposobem na wyodrębnienie odrębnych zależności biznesowych systemu jest wykorzystanie projektowania opartego na podejściu DDD (z ang. Domain Driven Design), czyli do tworzenia oprogramowania kładące nacisk na takie definiowanie obiektów i komponentów systemu oraz ich zachowań, aby wiernie odzwierciedlały rzeczywistość.

¹¹ MORGAN BRUCE, PAULO A. PEREIRA, "Microservices in Action", Manning Publications Co., 2019 Nowy Jork, s. 6.

¹² Kasun Indrasiri, Prabath Siriwardena, "Microservices for the Enterprise", Apress, 2018 San Jose, s. 8-9

¹³ <http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod>

- 3) **Zależności techniczne** to takie, które kilka innych mikroserwisów musi wykorzystać - na przykład do integracji z jakimś systemem dostawcy zewnętrznego. Zależności techniczne nie są głównymi czynnikami wpływającymi na podział systemu na mikroserwisy; są one identyfikowane tylko wtedy, gdy znajdzie się takich kilka mikroserwisów biznesowych, które wymagają takie same zależności technologiczne.
- 4) **Mikroserwis może być wdrażany indywidualnie.** Mikroserwis jako usługa, powinna być wdrażana do użytku indywidualnie. Gdy mikroserwis w jakikolwiek sposób ulega modyfikacji, zespół, który odpowiadający za wdrożenie go do ponownego użytku, powinien mieć możliwość wdrożenia go bez konieczności ponownego wdrażania, lub też modyfikacjach jakichkolwiek innych części systemu. Pozostałe mikroserwisy w systemie powinny nadal funkcjonować i być aktywne oraz działać podczas wdrażania zmienionego na nową wersję.¹⁴



Rysunek 3 Rysunek przedstawia działanie mikroserwisów, w trakcie wdrażania mikroserwisu koszyka sklepowego.

Przykładowo, kiedy dokonywana jest zmiana w mikroserwisie koszyka zakupowego, przy zastosowaniu architektury mikroserwisów, zespół powinien wdrożyć tylko ten mikroserwis, jak pokazano na rysunku 3. Tymczasem mikroserwis kalkulatora cen,

¹⁴ CHRISTIAN HORSDAL GAMMELGAARD, "Microservices in .NET Core", Manning Publications Co, 2017, s. 27-28

mikroserwis systemu rekomendacji, mikroserwis katalogu produktów powinny kontynuować pracę i obsługę poleceń użytkowników.

Możliwość wdrożenia każdej mikrousługi indywidualnie jest ważna, ponieważ istnieje wiele mikroserwisów, a każdy z nich może współpracować z kilkoma innymi. Jednocześnie, prace rozwojowe są wykonywane na niektórych lub we wszystkich mikroserwisach równolegle. Gdyby trzeba było wdrożyć wszystkie na raz, zarządzanie wdrożeniami szybko stałoby się nieefektywne, co zazwyczaj prowadzi do rzadkich i dużych, ryzykownych wdrożeń. Należy tego zdecydowanie unikać. Programiści preferują możliwość częstego wdrażania małych zmian w każdym mikroserwisie, co skutkuje małymi i mało ryzykownymi wdrożeniami.

Aby móc wdrożyć pojedynczy mikroserwis, podczas gdy reszta systemu będzie kontynuowana, proces budowy musi być zorganizowany w następujący sposób:

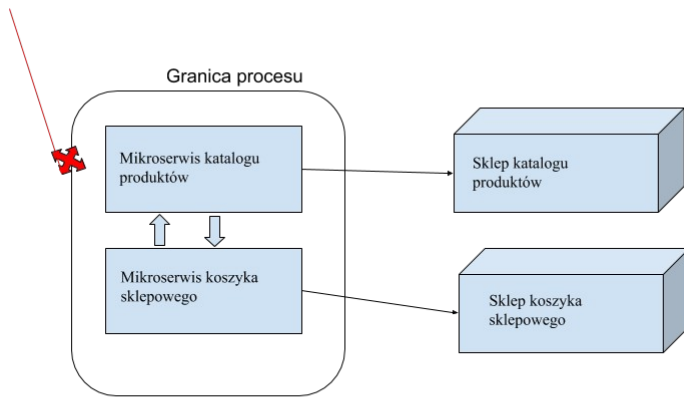
- Każdy mikroserwis musi być spakowany w osobne obiekty lub paczki.
- Proces wdrażania musi być również skonfigurowany w taki sposób, aby wspierał indywidualne wdrażanie mikroserwisu, podczas gdy inne mikroserwisy nadal działają. Na przykład, można skorzystać z procesu wdrażania, w którym mikroserwis jest wdrażany na jednym serwerze jednocześnie, aby zredukować czas przestoju.

Fakt, że zespół wdraża mikroserwisy indywidualnie wpływa na sposób, w jaki oddziałują one na siebie. Zmiany w interfejsie mikroserwisu zwykle muszą być wstecznie kompatybilne, więc pozostałe istniejące mikrousługi mogą w dalszym ciągu współpracować z nową wersją tak samo jak ze starą. Ponadto sposób, w jaki mikroserwisy oddziałują na siebie, musi być solidny w tym sensie, iż każdy mikroserwis musi oczekiwać, że inne usługi raz na jakiś czas zawiodą i musi działać jak najlepiej. Awaria jednego mikroserwisu, na przykład z powodu przestoju podczas wdrażania, nie może skutkować awarią innych mikroserwisów, a jedynie zmniejszoną funkcjonalnością lub nieco dłuższym czasem przetwarzania.¹⁵

- **Mikroserwis składa się z jednego lub więcej procesów.** Mikroserwis musi działać w oddzielnym procesie, jeśli ma pozostać jak najbardziej niezależny od innych mikroserwisów w tym samym systemie. To samo dotyczy sytuacji, gdy mikroserwis ma być wdrażany indywidualnie.

¹⁵ Thomas Hunter II, "Advanced Microservices. A Hands-on Approach to Microservice Infrastructure and Tooling", Apress, 2017 California, s. 2-4

Problematyczna granica procesu. Mikroserwisy powinny działać w oddzielnych procesach, aby uniknąć sprzężeń.



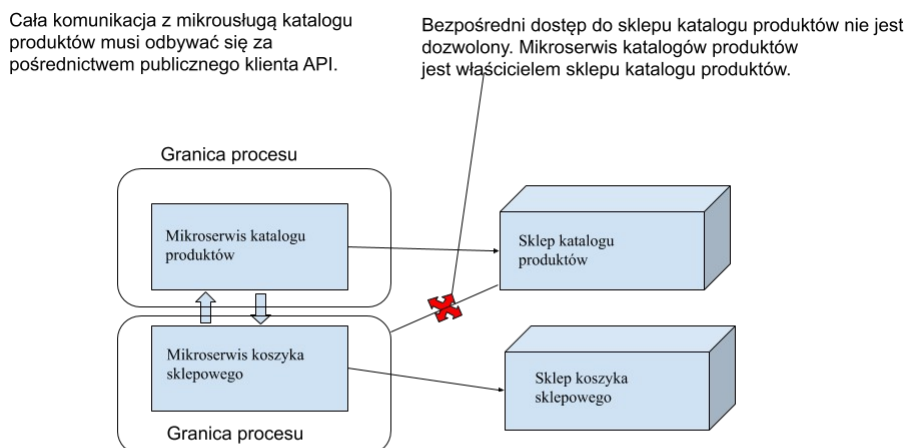
Rysunek 4 Uruchomienie więcej niż jednego mikroserwisu w ramach procesu. Opracowanie własne

Rozważono jeszcze raz mikroserwis koszyka sklepowego. Jeśli przebiegał on w tym samym procesie co mikroserwis w katalogu produktów, jak pokazano na rysunku 4, kod źródłowy koszyka sklepowego może wywołać efekt uboczny w katalogu produktów. Oznaczałoby to ściśle, niepożądane sprzężenie pomiędzy mikroserwisem koszyka sklepowego a mikroserwisem katalogu produktów - jeden z nich może powodować przestoje lub błędy w drugiej. Teraz należy rozważyć wdrożenie nowej wersji mikroserwisu koszyka sklepowego. Trzeba będzie ponownie uruchomić mikroserwis koszyka albo trzeba będzie ponownie uruchomić mikroserwis katalogu produktów. Może też być potrzeba stworzenia pewnego rodzaju dynamicznego ładowania kodu umożliwiające przełączanie kodu koszyka w działającym procesie. Pierwsza opcja dotyczy bezpośrednio mikroserwisów, które mogą być wdrażane indywidualnie. Druga opcja jest złożona i w minimalnym stopniu wystawia mikroserwis katalogu produktów na zagrożenie braku dostępu w związku z wdrożeniem mikroserwisu koszyka sklepowego.¹⁶

- **Mikroserwis posiada własny magazyn danych.** Mikroserwis jest odpowiedzialny za przechowywanie danych, w którym zapisuje niezbędne informacje. Jest to kolejna

¹⁶ CHRISTIAN HORSDAL GAMMELGAARD, "Microservices in .NET Core", Manning Publications Co, 2017, s. 30

konsekwencją tego, że zakres mikroservisu jest w pełni funkcjonalny. Większość możliwości biznesowych wymaga przechowywania danych.



Rysunek 5. Jeden mikroservis nie może uzyskać dostępu do danych przechowywanych przez inną usługę. Opracowanie własne

Przykładem może być mikroservis katalogu produktów, który potrzebuje pewnych informacji na temat każdego produktu. Aby katalog produktów pozostawał swobodnie sprzężony z innymi mikroservisami, dane przechowywane w bazie danych zawierającej informacje o produktach są w całości własnością mikroservisów. Mikroservis katalogu produktów decyduje o tym, w jaki sposób i kiedy informacje o produktach są przechowywane. Jak pokazano na rysunku 5, inne mikroservisy, takie jak koszyk sklepowy, mogą uzyskać dostęp do informacji o produkcie tylko poprzez interfejs do katalogu produktów i nigdy bezpośrednio z danych sklepu katalogu produktów.¹⁷

- **Niewielki zespół może obsługiwać kilka mikroservisów.** Utrzymanie mikroservisu oznacza tutaj zajmowanie się wszystkimi aspektami utrzymania go w dobrym stanie i przydatności do użytku: rozwijanie nowej funkcjonalności, wypuszczanie nowych mikrosług z tych, które stały się zbyt duże, uruchamianie ich w środowisku

¹⁷ Chander Dhall, "Scalability Patterns Best Practices for Designing High Volume Websites", Apress, 2018 Texas, s. 33

produkcyjnym dla użytkowników, monitorowanie go, testowanie, naprawianie błędów i wszystko inne co jest wymagane.

- **Mikroserwis jest wymienialny.** Aby mikroserwis mógł zostać wymieniony, musi on zostać przepisany od zera w rozsądnych ramach czasowych. Innymi słowy, zespół utrzymujący mikroserwis powinien być w stanie zastąpić obecną implementację zupełnie nową implementacją i wykonać to w normalnym tempie ich pracy. Ta cecha jest kolejnym ograniczeniem dla rozmiaru mikroserwisu: jeżeli mikroserwis rozrasta się zbyt szybko, jego wymiana będzie kosztowna; lecz jeśli jest mały, napisanie go na nowo jest mniej skomplikowane. Nie jest to również sytuacja pożądana, ale zmiany w wymaganiach w czasie mogą powodować powstanie kodu źródłowego, który ma sens raczej zastępować niż utrzymywać. Jeżeli mikroserwis nie jest w pełni rozbudowany, a także złożoność mikroserwisu może zostać szybko zrozumiana przez zespół programistów, a zespół posiada zasoby oraz czas, aby w krótkim okresie można było przepisać mikroserwis, np. ze względu na zmianę założeń, czy też zmiany technologii. Zespół może wtedy stworzyć na nowo w oparciu o wszelką wiedzę uzyskaną w trakcie pisania istniejącego wdrożenia, zachowując przy tym wszelkie nowe wymagania.¹⁸

Powyższe wytyczne cech charakterystycznych powinna pomóc w rozpoznaniu dobrze uformowanego mikroserwisu, jak i przy wdrożeniu kolejnych.

W architekturze monolitycznej aplikacja przechowuje dane w pojedynczych i scentralizowanych logicznych bazach danych w celu wdrożenia różnych funkcjonalności/zdolności aplikacji. W architekturze mikroserwisów, funkcjonalności są rozproszone na wiele mikroserwisów. Jeśli programiści będą korzystać z tej samej centralnej bazy danych, mikroserwisy nie będą dłużej niezależne od siebie (np. jeśli schemat bazy danych jest zmieniany przez jeden mikroserwis, który zakłóci pracę kilku innych usług). W związku z tym, każdy mikroserwis musi posiadać własną bazę danych i schemat bazy danych. Każdy mikroserwis może mieć prywatną bazę danych, aby przechowywać dane, które wymagają implementacji oferowanej przez niego funkcjonalności biznesowej. Dany mikroserwis może jedynie uzyskać dostęp do dedykowanej prywatnej bazy danych, a nie do baz danych innych mikroserwisów.

¹⁸ CHRISTIAN HORSDAL GAMMELGAARD, "Microservices in .NET Core", Manning Publications Co, 2017, s. 32-33

W niektórych scenariuszach biznesowych, być może trzeba będzie zaktualizować kilka kolumn które stanowią atrybuty jakiegoś obiektu danych dla istniejącej bazy danych. W takich scenariuszach należy zaktualizować bazy danych innych mikroserwisów poprzez odpowiedniego klienta API usługi dla każdego mikroserwisu (Mikroserwy nie mają dostępu do bazy danych bezpośrednio). Zdecentralizowane zarządzanie danymi zapewnia w pełni izolowane mikroserwisy oraz wolność wyboru odmiennych technik zarządzania danymi (Systemy bazy danych SQL lub NoSQL itp., różne systemy zarządzania bazami danych dla każdej usługi).¹⁹

Architektura mikroserwisów daje impuls do posiadania własnych baz danych, i te bazy danych nie powinny być współdzielone z żadną inną usługą. W związku z tym dany mikroserwis będzie miał wyizolowany magazyn danych lub użyje wyizolowanej usługi (np. od dostawcy chmury). Posiadanie osobnej bazy danych dla każdego mikroserwisu daje dużo swobody, jeśli chodzi o autonomię mikroserwisów. Na przykład, twórcy mikroserwisów mogą modyfikować kształt bazy danych zgodnie z wymaganiami biznesowymi, bez obaw o zewnętrznych użytkowników bazy danych. Nie istnieje podmiot z zewnętrznych aplikacji, który mógłby uzyskać bezpośredni dostęp do bazy danych. Daje to również twórcom mikroserwisów swobodę w dostępie do bazy danych, aby wybrać dowolną technologię, która ma być stosowana jako baza danych mikroserwisów, pozwala to również na eksperymentowanie z różnymi technologiami. Różne mikroserwisy mogą wykorzystywać różne technologie przechowywania danych, takie jak RDBMS, NoSQL lub inne usługi w chmurze. Jednak posiadanie bazy danych dla każdej usługi wprowadza nowy zestaw wyzwań. Kiedy przychodzi do realizacji każdego scenariusza biznesowego, dzielenia się danymi pomiędzy mikroserwisami, a realizacja transakcji w ramach i poza granicami usług staje się dość trudne.²⁰

Jednym z głównych powodów popularności architektury mikroserwisów są korzyści, jakie zapewnia ona w porównaniu z konwencjonalną architekturą oprogramowania.

- **Dynamiczny i szybki rozwój funkcji biznesowych.** Architektura mikroserwisów sprzyja niezależnemu rozwojowi usług, który podąża za szybkim rozwojem funkcjonalności biznesowych. W przypadku tradycyjnej architektury, przekształcając funkcjonalność biznesową w oprogramowanie gotowe do użytku przez użytkowników

¹⁹ Kasun Indrasiri, Prabath Siriwardena, "Microservices for the Enterprise", Apress, 2018 San Jose , s.29

²⁰ Kasun Indrasiri, Prabath Siriwardena, "Microservices for the Enterprise", Apress, 2018 San Jose, s. 127

funkcjonalność aplikacji wymaga wiele czasu poświęconego na ich stworzenie, głównie ze względu na wielkość systemu i zależności. Z niezależnym rozwojem usług, należy skupić się głównie na interfejsie użytkownika oraz na funkcjonalności usługi (a nie na funkcjonalności całego systemu, który jest znacznie bardziej złożony), jak wszystkie inne usługi tylko komunikują się ze sobą przez połączenia sieciowe poprzez interfejsy serwisowe.²¹

- **Wymienialność.** Ze względu na swoją autonomiczną naturę, mikroserwisy są również zastępowalne. Ponieważ buduje się usługi jako niezależny byt, który komunikuje się poprzez połączenia sieciowe i zdefiniowany klient API, można łatwo zastąpić tę funkcjonalność inną lepszą implementacją. Skupienie się na konkretnej funkcjonalności, posiadanie ograniczonego zasobu i rozmiaru oraz wdrożenie jej w niezależnym czasie pracy znacznie ułatwia zbudowanie usługi zastępczej.²²
- **Izolacja awarii i przewidywalność.** Wymienność pomaga nam również osiągnąć izolację i przewidywanie awarii. Jak już wspomniano, aplikacja oparta na mikroserwisach nie może ulegać niedziałaniu się jak konwencjonalna aplikacja monolityczna z powodu awarii jakiegokolwiek elementu lub usługi. Posiadanie właściwych właściwości obserwacji pomagają nam również zidentyfikować lub przewidzieć potencjalne awarie.²³
- **Elastyczne wdrażanie i skalowalność.** Łatwość wdrażania i skalowania może być najważniejszą zaletą mikroserwisów. Dzięki nowoczesnym, opartym na chmurze, natywnym infrastrukturom kontenerowym, istnieje możliwość łatwego wdrożenia usługi i jej dynamicznego skalowania. Aby ułatwić wdrożenie i skalowalność zaleca się używanie technologii chmurowych, jak i wirtualizacji danych.²⁴
- **Dostosowanie do struktury organizacyjnej.** Ponieważ mikroserwisy są zorientowane na możliwości biznesowe, mogą być dobrze dopasowane do struktury

²¹ Moises Macero, "Learn Microservices with Spring Boot A Practical Approach to RESTful Services using RabbitMQ, Eureka, Ribbon, Zuul and Cucumber", 2017 Nowy Jork, s. 13

²² Moises Macero, "Learn Microservices with Spring Boot A Practical Approach to RESTful Services using RabbitMQ, Eureka, Ribbon, Zuul and Cucumber", 2017 Nowy Jork, s. 14

²³ Moises Macero, "Learn Microservices with Spring Boot A Practical Approach to RESTful Services using RabbitMQ, Eureka, Ribbon, Zuul and Cucumber", 2017 Nowy Jork, s. 15

²⁴ Lee Atchison, "Architecting for Scale High Availability for Your Growing Applications", O'Reilly Media, Inc, 2016, s.5

organizacyjnej/zespołowej. Często dana organizacja jest zorganizowana w taki sposób, że zapewnia możliwości biznesowe. W związku z tym, własność każdej usługi może być łatwo przekazywana zespołom, które posiadają funkcjonalność biznesową. Biorąc pod uwagę, że mikroserwis skupia się na konkretnej funkcjonalności biznesowej, można wybrać relatywnie małą usługę. Ma to pozytywny wpływ na zespół rozwojowy, ponieważ zakres danej usługi jest prosty i dobrze zdefiniowany. W ten sposób zespół może w pełni utrzymywać usługę przez cały cykl życia.²⁵

Większość zobowiązań wynikających z architektury mikrousług wynika przede wszystkim z rozprzestrzenianie się usług, z którymi programiści muszą sobie poradzić. Dobrym przykładem będzie złożoność komunikacji między serwisowej. Mogłaby stanowić większe wyzwanie niż wdrożenie rzeczywistych usług. Programiści muszą poświęcić sporo czasu na wspólne wykonywanie mikroserwisów, aby stworzyć złożoną funkcjonalność biznesową. Posiadanie szeregu usług przekazywanych za pośrednictwem sieci komplikuje również aspekt zarządzania i możliwości obserwacji usług. Jeśli nie posiada się odpowiednich narzędzi zarządzania i obserwacji, ograniczona będzie identyfikacja zależności od usług i wykrywanie awarii. Na przykład, zarządzanie cyklem życia usługi, testowanie, wyszukiwanie, monitorowanie, jakość usług i różne inne możliwości zarządzania usługami stają się bardziej skomplikowane wraz z architekturą mikroserwisów. Czynnikiem sukcesu w wdrażaniu, jak i skalowaniu usług związanych mikroserwisami jest w dużej mierze uzależnione od systemów wirtualizacji, czyli izolowaniu zasobów, z pominięciem konieczności tworzenia wirtualnego systemu operacyjnego. Jeśli nie posiada się takich systemów w istniejącej infrastrukturze, trzeba będzie zainwestować w nią czas i energię. Ostatecznie, skuteczna architektura mikroserwisów zależy również od zespołów składających się z specjalistów. Tworzenie i utrzymywanie usług, troska o to, aby usługi były małe i hermetyczne, nieposiadające centralnego punktu do integracji usług itp. wymaga zmiany w kulturze inżynierskiej. Ponieważ architektura mikroserwisów promuje lokalizowanie danych do danej usługi, rozproszone zarządzanie danymi będzie dość trudne. To samo dotyczy transakcji rozproszonych. Wdrożenie ograniczeń transakcji, które obejmują wiele mikroserwisów, będzie dość trudne.²⁶

²⁵ Chander Dhall, "Scalability Patterns Best Practices for Designing High Volume Websites", Apress, 2018, s. 35

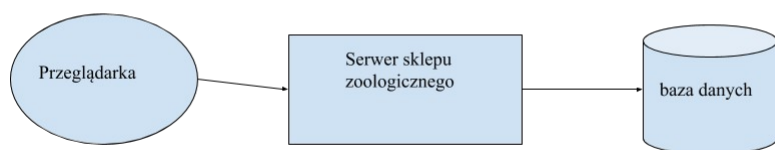
²⁶ Kasun Indrasiri Prabath Siriwardena, "Microservices for the Enterprise Designing, Developing, and Deploying", Apress, 2018 San Jose, s. 11-14

1.3. Aplikacje oparte o funkcje (FaaS)

Model FaaS (ang. Function as a Service, pol. Funkcja jako usługa) umożliwia programistom pisanie własnej logiki, tworzenie zestawów kodu i wykonywanie ich w określonym kontekście. Funkcje Serverless wymagają wykonania wywołań zdarzeń. Główne platformy Serverless obsługują różne języki i czasy działania w modelu FaaS. Koncepcja FaaS to podejście typu Serverless do przetwarzania danych oraz rodzaj architektury, co oznacza, że programista nie musi brać pod uwagę operacji serwerowych, ponieważ są one przechowywane na zewnątrz. Jest to zazwyczaj wykorzystywane przy tworzeniu mikroservisów, takich jak aplikacje internetowe, procesory danych, chatboty i automatyka IT. FaaS zapewnia programistom możliwość uruchamiania pojedynczej funkcji, części logiki lub aplikacji. Podczas wywoływania funkcji, zdalnie są uruchamiane są serwery obliczeniowe w celu wykonania zamierzonej akcji. W przeciwieństwie do innych modeli cloud computing, które działają na co najmniej jednym serwerze przez cały czas, FaaS działa tylko wtedy, gdy funkcja jest wykonywana, a następnie wyłącza się. Ważne jest, aby zauważyć, że dzięki modelowi chmury obliczeniowej FaaS można nadal budować architekturę opartą na mikroservisach. Należy pamiętać, że koncepcja mikroservisów koncentruje się na budowaniu małych usług, z ograniczoną odpowiedzialnością, wykorzystując interfejs oparty na protokole HTTP do komunikacji. Pojawiające się platformy przetwarzania w chmurze, takie jak FaaS, dotyczą tak naprawdę alternatywnych mechanizmów infrastruktury do wdrażania mikroservisów.²⁷

Jako przykład rozważono stronę sklepu zoologicznego online, w trójwarstwowym systemie zorientowanym na klienta z logiką po stronie serwera. Architektura będzie wyglądać jak na rys. 6. Przykładowo jest ona zaimplementowana w Ruby po stronie serwera, z komponentem w języku HTML + Javascript jako interfejs klienta.

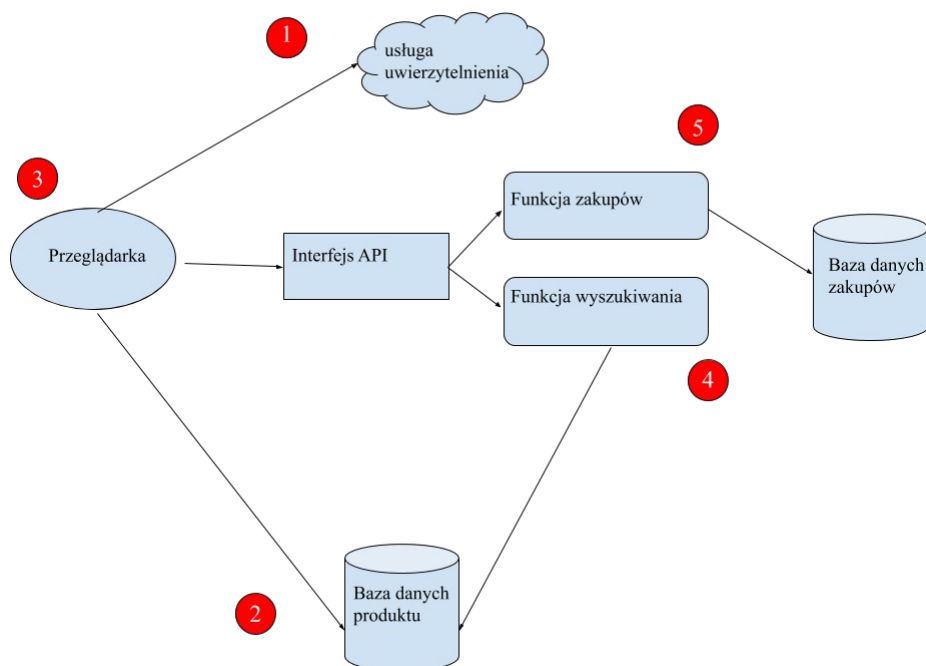
²⁷ Thurupathan Vijayakumar, " Practical API Architecture and Development with Azure and AWS Design and Implementation of APIs for the Cloud", Apress, s. 133-134



Rysunek 6 Aplikacja sklepu zoologicznego online opracowana w oparciu o architekturę monolityczną. Opracowanie własne

Dzięki tej architekturze aplikacja klienta może być nieprzemysłana, z dużą częścią logiki w systemie - uwierzytelnianie, nawigacją po stronach, wyszukiwanie, operacje - realizowane przez aplikację serwerową.

W przypadku architektury FaaS może to wyglądać bardziej tak jak na rysunku 7.



Rysunek 7 Aplikacja sklepu zoologicznego online opracowana w oparciu o architekturę Function as a Service. Opracowanie własne

Jest to widok znacznie uproszczony, ale nawet tutaj można zauważyć wiele istotnych zmian, między innymi:

1. Usunięto logikę uwierzytelniania w oryginalnej aplikacji.

Następnie zastąpiono ją zewnętrzną logiką. Korzystając z innego przykładu odrębnej funkcji, pozwolono klientowi na bezpośredni dostęp do podzbioru bazy danych (w przypadku produktów), która sama w sobie jest w pełni obsługiwana przez stronę trzecią (np. Zewnętrzna baza danych.) Dzięki temu posiada ona inny profil bezpieczeństwa dla klienta uzyskującego dostęp do bazy danych w ten sposób niż dla zasobów serwera, które uzyskują dostęp do bazy danych. Te dwa poprzednie punkty sugerują bardzo ważną trzecią: część logiki, która znajdowała się na serwerze sklepu zoologicznego jest teraz w kliencie - np.

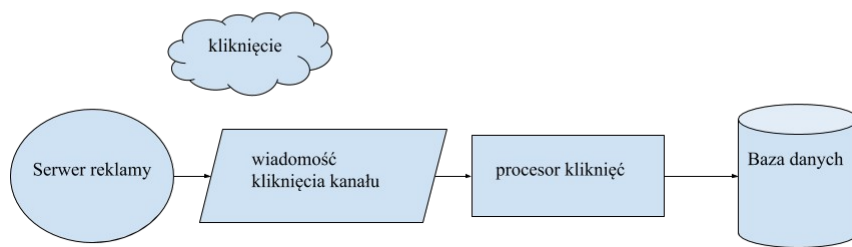
2. śledzenie sesji użytkownika,
3. zrozumienie struktury UX aplikacji,
4. odczytywanie z bazy danych i tłumaczenie tego na widok do wykorzystania itp.

Można również zachować niektóre funkcje związane z UX (User Experience - doświadczenie użytkownika) na serwerze, jeśli, na przykład, jest wymagający obliczeniowo lub wymaga dostępu do znacznych ilości danych. W sklepie zoologicznym przykładem jest "wyszukiwanie". Zamiast zawsze działającego serwera, jak to miało miejsce w oryginalnej

architekturze, można zamiast tego zaimplementować funkcję FaaS, która odpowiada na żądania HTTP poprzez Interfejs API. Zarówno klient jak i funkcja "szukaj" serwera czytają dane produktu z tej samej bazy danych. Jeśli zdecydowano się na użycie usługi AWS Lambda dla platformy FaaS, można przenieść kod wyszukiwania z oryginalnego serwera sklepu zoologicznego do nowej funkcji „wyszukiwarka” bez konieczności całkowitego przepisywania, ponieważ Lambda obsługuje Ruby i Javascript - oryginalne języki implementacji poprzedniego projektu. Wreszcie, można zastąpić funkcję "zakupu" inną oddzielną funkcją FaaS, wybierając utrzymanie jej po stronie serwera ze względów bezpieczeństwa, a nie ponowne jej wdrożenie w kliencie. Jest on również obsługiwany przez interfejs API. Podział różnych wymagań logicznych na osobno rozmieszczone komponenty jest bardzo powszechnym podejściem w przypadku korzystania z FaaS. Ten przykład pokazuje kolejny bardzo ważny punkt dotyczący architektury Serverless oraz FaaS. W wersji oryginalnej cały przepływ, kontrola i bezpieczeństwo były zarządzane przez centralną aplikację serwerową. Zamiast tego można zobaczyć pierwszeństwo układu logicznego przed orkiestracją, przy czym każdy komponent odgrywa bardziej świadomą architektonicznie rolę - pomysł ten jest również powszechny w podejściu mikroserwisowym.²⁸

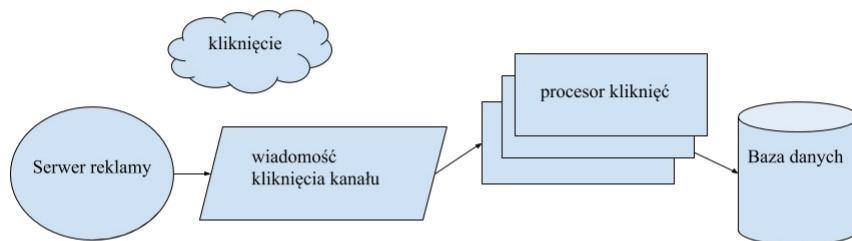
Innym przykładem jest usługa przetwarzania danych przez backend. Założono, że zespół musi napisać aplikację zorientowaną na użytkownika, która musi szybko reagować na żądania interfejsu użytkownika, a po drugie, musi przechwytywać wszystkie rodzaje aktywności użytkownika, w celu dalszego przetwarzania. W przypadku reklamy online, kiedy użytkownik kliknie na reklamę, należy bardzo szybko przekierować go do celu tej reklamy. W tym samym czasie, trzeba zarejestrować, kliknięcie, aby można obciążyć reklamodawcę. Tradycyjnie, architektura może wyglądać jak na rysunku Serwer reklamy synchronicznie odpowiada użytkownikowi oraz wysyła "wiadomość kliknięcia" na kanał. Wiadomość ta jest następnie asynchronicznie przetwarzana przez aplikację "procesor kliknięć", który aktualizuje bazę danych, np. w celu zmniejszenia budżetu reklamodawcy.

²⁸ <https://martinfowler.com/articles/serverless.html#unpacking-faas>



Rysunek 8 Schemat systemu reklamy online w podejściu tradycyjnym. Opracowanie własne.

W podejściu FaaS wygląda to następująco:



Rysunek 9 Schemat systemu reklamy online w podejściu FaaS. Opracowanie własne

Zmiana architektury jest tu znacznie mniejsza w porównaniu z pierwszym przykładem - dlatego asynchroniczne przetwarzanie wiadomości jest bardzo popularnym przypadkiem zastosowania technologii Serverless. Zastąpiono aplikację przeznaczoną dla konsumentów wiadomości funkcją FaaS. Funkcja ta działa w ramach kontekstu wywołanego przez zdarzenie, który zapewnia dostawca. Zauważono, że dostawca platformy chmury dostarcza zarówno brokera komunikatów, jak i środowisko FaaS - oba systemy są ze sobą ściśle powiązane. Środowisko FaaS może również przetwarzać równolegle kilka komunikatów poprzez inicjowanie wielu kopii kodu funkcji. W zależności od tego, jak napisano oryginalny proces, może to być nowa koncepcja, którą należy rozważyć.²⁹ Zasadniczo FaaS polega na uruchamianiu kodu backendu bez zarządzania własnymi systemami serwerowymi lub własnymi aplikacjami serwerowymi. Ta druga część - aplikacje serwerowe o długiej żywotności - jest kluczową różnicą w porównaniu z innymi nowoczesnymi trendami architektonicznymi, takimi jak kontenery i PaaS (Platform as a Service). Jeśli wróci się do przykładu przetwarzania kliknięć z wcześniejszego etapu, FaaS zastępuje serwer procesora kliknięć (być może fizyczną maszynę, ale na pewno konkretną aplikację) czymś, co nie wymaga serwera, ani aplikacji działającej cały czas. FaaS nie wymaga pisania kodu do konkretnego frameworka lub biblioteki. Funkcje FaaS są zwykłymi aplikacjami, jeśli chodzi o język i środowisko. Na przykład, funkcje AWS Lambda mogą być zaimplementowane w językach Javascript, Python, Go, dowolnym języku JVM (Java, Clojure, Scala, itp.), lub dowolnym języku np .NET. Jednak funkcja Lambda może również wykonać inny proces, który jest dołączony do swojego manifestu wdrożeniowego. Wdrożenie różni się bardzo od tradycyjnych systemów, ponieważ nie ma aplikacji serwerowych do samodzielnego uruchomienia.

W środowisku FaaS przesyłamy kod funkcji do dostawcy FaaS, a dostawca wykonuje zadania niezbędne do zapewnienia zasobów, inicjowania maszyn wirtualnych, zarządzania procesami itp. Skalowanie horyzontalne jest całkowicie automatyczne, elastyczne i zarządzane przez dostawcę. Jeśli system musi przetwarzać równolegle 100 zapytań, dostawca poradzi sobie z tym bez dodatkowej konfiguracji ze strony programisty. Kontenery obliczeniowe realizujące funkcje są krótkotrwałe, a dostawca FaaS tworzy i niszczy je, kierując się wyłącznie potrzebami operacyjnymi. Co najważniejsze, w przypadku FaaS dostawca zajmuje się dostarczaniem wszystkich podstawowych zasobów i alokacją - w ogóle nie jest wymagane przez użytkownika zarządzanie klastrem lub maszynami wirtualnymi.³⁰

²⁹ <https://martinfowler.com/articles/serverless.html#Message-driven-applications>

³⁰ <https://martinfowler.com/articles/serverless.html#State>

2. Pojęcie Serverless

2.1. Wstęp

Operacje obliczeniowe bez użycia serwera, w skrócie Serverless, jest to model wykonywania, w którym dostawca chmury (Amazon Web Services, Microsoft Azure lub Google Cloud Platform) jest odpowiedzialny za wykonanie fragmentu kodu programowania przez dynamiczne przydzielanie zasobów.³¹ Architektura Serverless może zmniejszyć złożoność operacyjną uruchomionych aplikacji. Możliwe jest budowanie zarówno usług opartych na zdarzeniach, jak i usług synchronicznych dla urządzeń mobilnych, internetowych, analitycznych, logiki biznesowej CDN i Internetu Rzeczy (IoT) bez konieczności zarządzania infrastrukturą serwerową. Architektury te mogą zmniejszyć koszty, ponieważ nie trzeba zarządzać niewykorzystanymi serwerami ani płacić za nie w pełni wykorzystane serwery, ani też zapewniać nadmiarowej infrastruktury w celu wdrożenia wysokiej dostępności.³² Na przykład, usługi pamięci masowej mogą działać jak statyczne strony internetowe, usuwając potrzebę korzystania z serwerów internetowych, a usługi zdarzeń mogą przechowywać kod za programistę. To nie tylko usuwa obciążenie operacyjne związane z zarządzaniem tymi serwerami, ale również może obniżyć koszty transakcyjne, ponieważ usługi zarządzane działają w skali chmury.³³

Funkcjonowanie aplikacji informatycznych wiąże się, między innymi, z ryzykiem awarii i braku dostępności tych aplikacji. Im większa jest liczba różnych typów systemów lub komponentów i za zarządzanie nimi odpowiedzialne są różne zespoły, tym większa jest podatność na pojawiające się problemy. Zamiast samodzielnie zarządzać systemami, można je zlecać na zewnątrz w formie outsourcingu. Dzięki technologii Serverless znacznie zmniejsza się liczbę różnych technologii, za których bezpośrednie działanie odpowiadają zespoły programistów. Samodzielnie zarządza się serwerami, którymi zarządza zespół

³¹ <https://serverless-stack.com/chapters/what-is-serverless.html>

³² Amazon Web Services, "Architecting for the Cloud. AWS Best Practices", Amazon Web Services Inc, 2018, s. 19.

³³ Amazon Web Services, "AWS Well-Architected Framework", Amazon Web Service Inc., 2018, s. 25

administratorów, gdyż wtedy jest większa pewność, że będą mogli radzić sobie z awariami, gdy do nich dojdzie.³⁴

Firma Amazon Web Services opracowała cztery cechy, które klasyfikują architekturę Serverless:

- **Brak serwerów do zarządzania lub udostępniania.** Programista nie zarządza serwerami fizycznymi, maszynami wirtualnymi ani kontenerami. Chociaż istnieją one fizyczne, ale są zarządzane przez dostawcę chmury i niedostępne dla programistów.³⁵
 - **Wycenione według zużycia (a nie pojemności).** Jeżeli ktoś nie korzysta z usługi stworzonej przez programistę, to za nią nie płaci.³⁶ Dużą korzyścią, jaką przynosi dla tego obszaru podejście Serverless, jest to, że nie ma potrzeby planowania, ani przydzielania ani udostępniania zasobów. Pozwala się, aby usługa zapewniała dokładnie taką moc, jakiej potrzeba w dowolnym momencie. Jeśli nie ma żadnego zużycia, nie potrzeba żadnych zasobów obliczeniowych i nie będzie trzeba za nie płacić. Jeśli jest tylko 1 GB danych, nie potrzeba pojemności do przechowywania 100 GB. Jest zapewnienie, że usługa zwiększy się dokładnie wtedy, gdy jej potrzebujemy, na żądanie.³⁷
 - **Skalowanie z użyciem.** W systemach Serverless programiści są przyzwyczajeni do skalowania usług zarówno horyzontalnie jak i wertykalnie, aby zaspokoić popyt na dane usługi. W przeszłości ta praca była wykonywana ręcznie, dopóki dostawcy chmury, nie zaczęli oferować usługi automatycznego skalowania.
- Obecnie usługi i aplikacje typu Serverless mają wbudowane automatycznie skalowanie. Gdy przychodzą żądania do aplikacji programisty, chmura obliczeniowa skaluje daną aplikację lub usługę, aby zaspokoić popyt. Serverless wykonuje tą pracę za programistę.³⁸
- **Wbudowana dostępność i odporność na błędy.** Odpowiedzialność za dostępność oraz odporność na błędy spoczywa na dostawcę chmury.³⁹

³⁴ John Chapin, Michael Roberts, "What Is Serverless?", O'Reilly Media, Inc., 2017, s. 20

³⁵ Tom McLaughlin, "Serverless Devops", 2018, s. 6

³⁶ Tom McLaughlin, "Serverless Devops", 2018, s. 6

³⁷ John Chapin, Michael Roberts, "What is serverless", O'Reilly Media, Inc., 2017, s. 17

³⁸ Tom McLaughlin, "Serverless Devops", 2018, s. 7

³⁹ Tom McLaughlin, "Serverless Devops", 2018, s. 7

Wdrażanie aplikacji Serverless, jak i częściowo Serverless, umożliwia: obniżenie kosztów i przyspieszenie czasu wprowadzenia na rynek. Ale w kontekście tworzonej aplikacji należy dokładnie rozważyć drogę do architektury Serverless.⁴⁰

Do zalet należą między innymi:

- **Szybszy czas wykonywania.** W przeciwieństwie do standardowych instancji w chmurze, które generalnie potrzebują do kilku minut, aby zostały uruchomione, z kolei dla funkcji Serverless potrzeba zaledwie kilku sekund.⁴¹
- **Kompatybilność mikroserwisów.** Ponieważ funkcje obliczeniowe Serverless są niewielkie, składają się z niezależnych kawałków kodu, które są przeznaczone do wykonywania bardzo specyficznego zestawu zadań lub działań, mogą one być wykorzystywane jako nośnik dostaw dla mikroserwisów. Jest to ogromna zaleta w porównaniu do serwera monolitycznego aplikacji w chmurze, które nie skalują się tak skutecznie.
- **Wsparcie popularnych języków programowania.** Serverless platformy obliczeniowe, obsługują dziś wiele różnych języków programowania, takich jak Java, Node.js, Python, a nawet C#. Funkcje Azure pozwalają na korzystanie ze skryptów F#, PHP, Bash, Batch i PowerShell oraz kilku innych.⁴²
- **Aplikacje oparte na zdarzeniach.** Funkcje Serverless są bardzo dobrym wyborem do projektowania i uruchamiania aplikacji opartych na zdarzeniach, które reagują na określone zdarzenia i biorą niektóre z nich działania przeciwko nim. Na przykład, operacja przesyłania obrazu do pamięci masowej w chmurze wyzwała funkcję, która tworzy skojarzone miniaturki obrazów dla tej samej funkcji.⁴³

Serverless to inny sposób budowania i obsługi systemów operacyjnych i podobnie jak w przypadku większości rozwiązań alternatywnych, istnieją zarówno zalety jak i ograniczenia. Serverless jest wciąż nową platformą. Przykładowo AWS Lambda (opisana szerzej

⁴⁰ PETER SBARSKI, "Serverless Architectures on AWS", Manning Publications Co, 2017, s.12

⁴¹ Yohan Wadia, Udit Gupta, Mastering AWS Lambda, Packt Publishing, 2017, s. 21

⁴² Yohan Wadia, Udit Gupta, Mastering AWS Lambda, Packt Publishing, 2017, s. 21

⁴³ Yohan Wadia, Udit Gupta, Mastering AWS Lambda, Packt Publishing, 2017, s. 21

następnym rozdziale), jest najbardziej dojrzałą platformą, a jej pierwsza, bardzo ograniczona wersja została uruchomiona dopiero pod koniec 2014 roku.⁴⁴

Podejście Serverless posiada jednak pewne ograniczenia w następujących obszarach:⁴⁵

- **Decentralizacja.** Przejście od monolitycznego podejścia do bardziej zdecentralizowanego podejścia Serverless nie zmniejsza złożoności systemu bazowego. Rozproszona natura rozwiązania może wprowadzić własne wyzwania ze względu na konieczność wykonywania połączeń zdalnych, a nie procesowych oraz konieczność obsługi awarii i opóźnień w sieci.⁴⁶
- **Sterowniki decyzyjne.** Serverless może nie być odpowiednie dla aplikacji wrażliwych na opóźnienia lub oprogramowania z określonymi umowami o poziomie usług (SLA). Zablokowanie dostawcy może być problemem dla klientów korporacyjnych i rządowych, a decentralizacja usług może być wyzwaniem.
- **Blokada dostawcy** Blokowanie dostawcy to kolejny problem. Jeśli programista zdecyduje się użyć interfejsów API i usług innych firm, w tym AWS, istnieje szansa, że architektura będzie silnie sprzężona z używaną platformą. Konsekwencje blokady dostawców i ryzyko korzystania z usług innych firm - w tym rentowność firmy, suwerenność danych i prywatność, koszty, wsparcie, dokumentacja i dostępny zestaw funkcji - muszą być dokładnie przemyślane.⁴⁷

Nawet jeśli uda się łatwo przenieść jedną część ekosystemu, może to oznaczać, że inny komponent architektury będzie miał na programistę większy wpływ. Na przykład, założmy, że używane jest AWS Lambda do reagowania na zdarzenia w systemie AWS Kinesis. Różnice pomiędzy AWS Lambda, Google Cloud Functions i Microsoft Azure Functions mogą być stosunkowo niewielkie, ale nadal programista nie będzie mógł wdrożyć dwóch ostatnich implementacji dostawcy bezpośrednio do strumienia AWS Kinesis. Oznacza to, że przenoszenie kodu z jednego rozwiązania do drugiego nie będzie możliwe bez przenoszenia innych elementów infrastruktury.⁴⁸

⁴⁴ John Chapin, Michael Roberts, "What Is Serverless?", O'Reilly Media, Inc., 2017, s. 20

⁴⁵ John Chapin, Michael Roberts, "What Is Serverless?", O'Reilly Media, Inc., 2017, s. 20

⁴⁶ PETER SBARSKI, "Serverless Architectures on AWS", Manning Publications Co, 2017, s.14

⁴⁷ Yohan Wadia, Udita Gupta, Mastering AWS Lambda, Packt Publishing, 2017, s. 21

⁴⁸ <https://martinfowler.com/articles/serverless.html>

Istotną rzeczą jest fakt, jeżeli pojedynczy dostawca zmienia coś w swoich usługach, robi to dodatkową pracę dla programistów do wdrożenia. Z tego powodu niektórzy programiści przyjmują podejście "multicloud", rozwijając i obsługując aplikacje w sposób, który jest agnostyczny w stosunku do faktycznego dostawcy chmury, który jest używany. Często jest to nawet bardziej kosztowne niż podejście oparte na pojedynczej chmurze, więc podczas gdy blokada dostawcy jest uzasadnionym problemem, nadal zaleca się wybranie dostawcy, który najlepiej spełnia wymagania użytkownika i wykorzystanie jego możliwości w jak największym stopniu.⁴⁹

- **Brak narzędzi.** Mimo, że podejście Serverless do przetwarzania danych jest aktualnie popularnym tematem, jednakże wiele nieszablonowych narzędzi do zarządzania, było zaprojektowane z myślą o złożonych aplikacjach, a nie dla prostych funkcjach, które będą wykonywane w czasie kilku sekund.⁵⁰
- **Testy lokalne.** Trudności w lokalnym testowaniu są jednym z największych ograniczeń architektur Serverless aplikacji. W świecie Serverless, deweloperzy często posiadają lokalne odpowiedniki komponentów aplikacji (takich jak bazy danych lub kolejki wiadomości), które mogą być zintegrowane do testowania w taki sam sposób, w jaki aplikacja może być wdrażana w produkcji. Aplikacje Serverless mogą oczywiście polegać na testach jednostkowych, ale bardziej realistyczna integracja lub testowanie końcowe jest znacznie trudniejsze. Trudności w lokalnym testowaniu aplikacji Serverless można sklasyfikować na dwa sposoby. Po pierwsze, ponieważ znaczna część infrastruktury jest wyabstrahowana wewnątrz platformy, może być trudno połączyć komponenty aplikacji w realistyczny sposób, włączając w to obsługę błędów produkcyjnych, logowanie, wydajność i charakterystykę skalowania. Po drugie, aplikacje Serverless są z natury rozproszone i składają się z wielu oddzielnych elementów, więc proste zarządzanie niezliczonymi funkcjami i komponentami jest trudne, nawet lokalnie. Zamiast próbować przeprowadzać testy integracyjne lokalnie, rozwiązaniem, jest wykonywanie ich zdalnie. Wykorzystuje to bezpośrednio platformę Serverless, choć to również ma swoje ograniczenia.⁵¹

2.2. Zastosowania architektury Serverless

⁴⁹ <https://martinfowler.com/articles/serverless.html>

⁵⁰ Yohan Wadia, Udit Gupta, Mastering AWS Lambda, Packt Publishing, 2017, s. 21

⁵¹ John Chapin, Michael Roberts, "What is serverless", O'Reilly Media, Inc., 2017, s. 22

Architektura Serverless pozwala programistom skupić się na procesie wytwarzania oprogramowania, a nie na infrastrukturze serwerowej. Skalowalność i wysoka dostępność są łatwiejsze do osiągnięcia, a cena jest często bardziej sprawiedliwa, ponieważ programiści płacą tylko za to, z czego sami skorzystają. Co ważne, bez serwera, możliwe jest zmniejszenie złożoności systemu, minimalizując liczbę warstw abstrakcji w projekcie oraz ilość potrzebnego kodu.⁵²

Aplikacje internetowe i strony internetowe - Eliminacja serwerów umożliwia tworzenie aplikacji internetowych, które nie kosztują prawie nic, gdy nie ma ruchu, jednocześnie są skalowalne w celu przenoszenia obciążeń szczytowych, nawet tych nieoczekiwanych.

Backend dla aplikacji mobilnych które są Serverless oferują sposób na programistów, którzy skupiają się na rozwoju klienta w celu łatwego tworzenia bezpiecznych rozwiązań, wysoce dostępne i doskonale skalowalne backendy, bez stawiania się ekspertami. w projektowaniu systemów rozproszonych.

Przetwarzanie multimediów i logów - podejście Serverless oferuje przetwarzanie równoległe, ułatwiające przetwarzanie dużych obciążeń obliczeniowych bez konieczności budowania systemów wielowątkowych lub ręcznego skalowania zasobów obliczeniowych.

Automatyka IT - funkcje Serverless mogą być dołączane do alarmów i funkcji monitorowania, które w razie potrzeby można dostosować do własnych potrzeb.

Zadania typu CRON i inne - wymagania dotyczące infrastruktury informatycznej są znacznie prostsze do wdrożenia poprzez usunięcie wymogu posiadania i utrzymywania serwerów, zwłaszcza gdy te miejsca, gdzie znajdują się fizycznie serwery, ulegają częstej zmianie lokalizacji, przez to wymagana jest ogromna ilość konfiguracji.

IoT backendy- Możliwość sprowadzenia dowolnego kodu, w tym bibliotek natywnych, upraszcza proces tworzenia systemów opartych na chmurze, które mogą wdrożyć algorytmy specyficzne dla danego urządzenia.

Czatboty (w tym asystenci z obsługą głosową) i inne systemy oparte na usługach internetowych - Podejścia Serverless są idealnym rozwiązaniem dla każdego systemu opartego na webhook, jak chatbot. Ich zdolność do działania (np. uruchomienie kodu) tylko wtedy, gdy jest to konieczne (np. gdy użytkownik prosi o informacje z chatbota) sprawia, że są one proste i łatwe w obsłudze. Zazwyczaj w przypadku tych architektur podejście oparte na niższych kosztach. Na przykład większość Alexa Skills for Amazon Echo jest wdrażana z wykorzystaniem AWS Lambda.

⁵² PETER SBARSKI, "Serverless Architectures on AWS", Manning Publications Co, 2017, s.14

Serverless można wykorzystać również w przypadku strumieniowego przesyłania danych w czasie zbliżonym do rzeczywistego oraz innych procesów strumieniowego przesyłania danych w czasie rzeczywistym.

Rozwiązania Serverless oferują elastyczność w zakresie skalowania w górę i w dół za pomocą przepływu danych, umożliwiając im dostosowanie się do wymagań w zakresie przepustowości bez potrzeby złożoności budowania skalowalnego systemu obliczeniowego dla każdego z nich osobno. W połączeniu z technologią taką jak Amazon Kinesis, AWS Lambda może zaoferować szybkie przetwarzanie rekordów dla strumienia kliknięć analizy, wyzwalacze danych NoSQL, informacje o obrocie towarowym i wiele więcej⁵³

Zadania takie jak konfiguracja i zarządzanie serwerem, naprawa i konserwacja są realizowane przez dostawcę, co pozwala zaoszczędzić czas i pieniądze. Dostawcy chmury obliczeniowej dbają o stan fizyczny swojej floty serwerów, które zasilają chmurę obliczeniową. Jeżeli programista nie ma określonych wymagań do zarządzania lub modyfikowania zasobów obliczeniowych, to posiadanie dostawcy chmury jest dobrym rozwiązaniem. Programista jest odpowiedzialny tylko za własny kod, pozostawiając zadania operacyjne i administracyjne w rękach dostawcy.⁵⁴

Bezstanowość i skalowalność obliczeń można wykorzystać do rozwiązywania problemów, które przynoszą korzyści z przetwarzania danych równolegle. Aplikacje typu CRUD (stwórz, odczyt, Aktualizacja usuń), e-commerce, systemy zapleczone, złożone aplikacje internetowe oraz wszelkiego rodzaju oprogramowanie mobilne i stacjonarne można szybko zbudować przy użyciu architektury Serverless. Zadania, które dawniej zajmowały tygodnie, można wykonać w ciągu kilku dni lub godzin, o ile wybrano odpowiednią kombinację technologii. Podejście Serverless może działać wyjątkowo dobrze dla startupów, które chcą wprowadzać innowacje i szybko się rozwijać.⁵⁵

Tradycyjna architektura wymaga serwerów, które niekoniecznie muszą działać z pełną wydajnością przez cały czas. Skalowanie, nawet w systemach zautomatyzowanych, wiąże się z nowym serwerem, którego zużycie jest marnowane, dopóki nie nastąpi tymczasowy wzrost ruchu lub nowych danych. Systemy oparte na Serverless są bardziej

⁵³ <https://d0.awsstatic.com/whitepapers/optimizing-enterprise-economics-serverless-architectures.pdf>

⁵⁴ PETER SBARSKI, "Serverless Architectures on AWS", Manning Publications Co, 2017, s.14

⁵⁵ PETER SBARSKI, "Serverless Architectures on AWS", Manning Publications Co, 2017, s.14

wydajne pod względem skalowania i są opłacalne, zwłaszcza gdy obciążenia szczytowe są nierównomierne lub nieoczekiwane.⁵⁶

Architektura Serverless zapewnia możliwość zmniejszenia części złożoności i kodu w porównaniu z bardziej tradycyjnymi systemami. Nie ma potrzeby posiadania wielowarstwowego systemu zaplecza, zwłaszcza jeśli pozwala się wykonywać więcej pracy po stronie interfejsu użytkownika i bezpośrednio rozmawiać z usługami chmury obliczeniowej oraz z bazą danych.⁵⁷

Brak konieczności płacenia za stan bezczynności serwerów. Nie ma potrzeby wstępnego zapewniania ani nadmiernego udostępniania zasobów dla takich zadań, jak obliczenia i przechowywanie. Nie ma opłaty, jeżeli kod programowania nie jest w trakcie wykonywania w chmurze obliczeniowej.⁵⁸

2.3. Przykłady rozwiązań oferujących podejście Serverless

2.3.1. A Cloud Guru

A Cloud Guru (<https://acloud.guru>) to internetowa platforma edukacyjna dla architektów rozwiązań, administratorów systemów i programistów, którzy wykorzystują usługi Amazon Web Services. Podstawowe funkcje platformy obejmują (strumieniowe) kursy wideo, egzaminy praktyczne i quizy oraz fora dyskusyjne w czasie rzeczywistym. Cloud Guru to także platforma e-commerce, która umożliwia studentom kupowanie kursów i uczestniczenie w nich w wolnym czasie. Instruktorzy, którzy tworzą kursy dla A Cloud Guru, mogą przysyłać filmy bezpośrednio do usługi S3 (Simple Storage Service), które są natychmiast transkodowane do wielu różnych formatów (1080p, 720p, HLS, WebM itd.) i są udostępniane uczniom do oglądania. Platforma Cloud Guru wykorzystuje Firebase jako podstawową bazę danych skierowaną do klienta, która umożliwia klientom odbieranie aktualizacji w czasie zbliżonym do rzeczywistego bez odświeżania lub odpytywania (Firebase używa gniazd sieciowych do przekazywania aktualizacji do wszystkich podłączonych urządzeń w tym samym czasie).⁵⁹

⁵⁶ PETER SBARSKI, "Serverless Architectures on AWS", Manning Publications Co, 2017, s.14

⁵⁷ PETER SBARSKI, "Serverless Architectures on AWS", Manning Publications Co, 2017, s.15

⁵⁸ Amazon Web Services, "Serverless Architectures with AWS Lambda Overview and Best Practices", Amazon Web Services, Inc., 2017, s. 1

⁵⁹ PETER SBARSKI, "Serverless Architectures on AWS", Manning Publications Co, 2017, s.20

Usługa Auth0 służy do zapewnienia możliwości rejestracji i uwierzytelniania. Tworzy tokeny delegacji, które pozwalają interfejsowi użytkownika bezpośrednio i bezpiecznie komunikować się z innymi usługami, takimi jak Firebase.

Wykładowcy, którzy tworzą zawartość platformy, mogą przysyłać pliki (zazwyczaj filmy, ale mogą to być inne typy) bezpośrednio do usługi Simple Storage Service w skrócie S3 (służącym do hostowania plików na serwerze) za pośrednictwem przeglądarki. Aby to zadziałało, aplikacja internetowa wywołuje funkcję Lambda (za pośrednictwem bramki API), aby najpierw zażądać niezbędnych poświadczeń przesyłania. Po pobraniu poświadczeń aplikacja internetowa klienta rozpoczyna przesyłanie plików do S3 przez HTTP. Wszystko to dzieje się za kulisami i jest ukryte dla użytkownika.

Po przesłaniu pliku do S3 automatycznie uruchamia łańcuch zdarzeń (potok sterowany zdarzeniami), który transkoduje wideo, zapisuje nowe pliki w innym S3, aktualizuje bazę danych i natychmiast udostępnia transkodowane wideo innym użytkownikom.⁶⁰

2.3.2. Custom Ink

Custom Ink umożliwia projektowanie i zamawianie niestandardowych koszul i sprzętu dla swoich klubów, firm, organizacji charytatywnych, uroczystości rodzinnych i nie tylko. Custom Ink zaczął wykorzystywać AWS Lambda do zadań związanych z automatyzacją serwerów, w tym uruchamiania i zatrzymywania serwerów o znaczeniu niekrytycznym oraz zarządzania kopiami zapasowymi. Te zadania automatyzacji zadziałały dobrze w produkcji, a firma zdecydowała się na migrację swojego serwisu i grafiki do AWS Lambda. Usługa ta pozwala klientom na zastosowanie wielu efektów do ich własnych dzieł sztuki lub dowolnego elementu klipartów w bibliotece Custom Ink.

Migracja zajęła trzy miesiące, zmniejszyła ich koszty o 90% i zwiększyła niezawodność usług, przy wskaźniku błędu 0,005 dla 2000 zapytań na minutę. Custom Ink korzysta z automatycznego skalowania AWS Lambda, dzięki czemu nieprawidłowe zgłoszenia nie wpływają na ogólny stan usług w większym stopniu niż gdyby zarządzały nimi niezależnie. Architektura Serverless pozwala na obsługę ruchu w godzinach szczytu bez ostrzeżeń i pozwala klientom wygenerować klipart o wysokim wskaźniku DPI i odzyskiwać dane z nieprawidłowego przesłania danych przez użytkowników bez konieczności ręcznej interwencji. Dzięki zastosowaniu podejścia Serverless, Custom Ink był w stanie zredukować

⁶⁰ PETER SBARSKI, "Serverless Architectures on AWS", Manning Publications Co, 2017, s. 20-21

koszt platformy oraz zwiększyć jej skalowalność i dostępność. Zespół operacyjny może teraz skupić się na innych obszarach, ponieważ nie musi już permanentnie zarządzać usługą.⁶¹

2.3.3. SundaySky

SundaySky zapewnia kompleksową platformę wideo, która przekształca relacje z klientami poprzez spersonalizowane wideo. SundaySky zastosował podejście Serverless z AWS, przekształcając dużą część monolitycznej platformy do generowania i dostarczania wideo w osobne, bardziej zarządzalne mikrousługi. SundaySky zbudował nową platformę, która umożliwiła prosty proces tworzenia złożonych, niestandardowych integracji biznesowych, które obsługują spersonalizowane, dynamiczne filmy wideo, które są unikalne dla każdego z ich klientów. Zdecydowano się na zastosowanie AWS Lambda i Amazon API Gateway w całej platformie renderowania i strumieniowania wideo ze względu na korzyści wynikające ze zwiększonej zwinności, niskich kosztów operacyjnych i obniżonych kosztów. SundaySky wykorzystuje funkcję Lambda do enkapsulacji logiki biznesowej specyficznej dla klienta, wykorzystywanej do wygenerowania dynamicznego klipu wideo, do dostarczenia do systemu interfejsu API specyficznego dla klienta oraz do tworzenia oddzielnych i skalowalnych usług typu backend. Upraszczając i izolując każdą część architektury, SundaySky ma teraz znacznie prostszy ciągły przepływ wdrożenia, który prowadzi do szybszego testowania i wdrażania nowych funkcji. SundaySky obniżył koszt uruchomienia specyficznej dla klienta logiki biznesowej o 60% w porównaniu z ich monolityczną starą platformą.⁶²

2.3.4. Haufe Group

Haufe Group – z markami Haufe, Haufe Akademie i Lexware – jest uważana za pioniera cyfrowej transformacji w Niemczech i jednego z wiodących niemieckich dostawców rozwiązań i usług cyfrowego miejsca pracy, a także oferujących programy edukacyjne i szkoleniowe. Dział inżynierii treści i rozwoju (CED) firmy Haufe jest odpowiedzialny za wewnętrzne aplikacje i produkty dla grup docelowych (np. średnie i duże firmy i konsultanci, warstwy).

Aby sprostać wyzwaniu, jakim jest przeprojektowanie jednego z tych systemów i rozbicie monolitu na mikrousługi, CED wybrała Serverless podejście do przetwarzania

⁶¹ <https://aws.amazon.com/lambda/resources/customer-testimonials/>

⁶² <https://aws.amazon.com/lambda/resources/customer-testimonials/>

danych. Dla Haufe ważne jest posiadanie skalowalnego sposobu importu i eksportu dokumentów – przynajmniej w ciągu nocy, kiedy wszystkie produkty poszukują nowych treści. Te systemy wykorzystują AWS Lambda, Step Functions i API Gateway do pobierania plików z S3. Pliki są sprawdzane i ulepszone, a następnie zapisywane w relacyjnej bazie danych opartej na Aurorze.

Dzięki AWS Lambda, Haufe jest w stanie osiągnąć wskaźniki importu dokumentów, których wcześniej nie było w stanie osiągnąć, jak w przypadku importu 500.000 dokumentów w czasie krótszym niż sześć godzin. Architektura Serverless oparta na mikroservisach ułatwia również rozbudowę i zmianę systemu importu bez przerywania przepływu pracy, pozwalając im obsługiwać usługę importu i eksportu, która jest dostępna 24 godziny na dobę przez 7 dni w tygodniu i płacą za nią tylko wtedy, gdy z niej korzystają. Jest to ważny wymóg, ponieważ import i eksport odbywają się w nieregularnych odstępach czasu. Elastyczność w zakresie przepływów pracy związanych z importem pozwala na szybką zmianę schematu importowanych plików. Wybierając to podejście, CED jest w stanie zwiększyć elastyczność programistów, przy jednoczesnym zmniejszeniu kosztów i konserwacji.⁶³

Podsumowując chmura obliczeniowa była i nadal jest czynnikiem zmieniającym infrastrukturę IT i rozwój oprogramowania. Twórcy oprogramowania muszą zastanowić się nad sposobami maksymalizacji wykorzystania platform chmury w celu uzyskania przewagi konkurencyjnej. Architektura Serverless to innowacyjna technologia dla programistów i organizacji, którą należy badać i wdrażać. Technologia ta będzie szybko się rozwijała, ponieważ twórcy oprogramowania korzystają z usług obliczeniowych, takich jak AWS Lambda. A w wielu przypadkach aplikacje Serverless będą tańsze w uruchamianiu i szybsze w implementacji. Istnieje również potrzeba zmniejszenia złożoności i kosztów związanych z uruchomieniem struktury infrastruktury i rozwojem tradycyjnych systemów oprogramowania. Redukcja kosztów i czasu poświęcanego na utrzymanie infrastruktury oraz korzyści płynące ze skalowalności to powody, dla których organizacje i programiści powinni rozważyć architekturę Serverless.⁶⁴ Jednak technologia Serverless nie jest idealnym rozwiązaniem dla wszystkich przypadków i należy dokładnie przeanalizować, kiedy jest to uzasadnione. Krótkotrwałe, oparte na zdarzeniach (ang. Events) przetwarzanie danych napędza już na wstępnym etapie przyjmowanie innowacyjnych rozwiązań, a w przypadku

⁶³ <https://aws.amazon.com/lambda/resources/customer-testimonials/>

⁶⁴ PETER SBARSKI, "Serverless Architectures on AWS", Manning Publications Co, 2017, s.38

przedsiębiorstw, które oczekują wysokiego tempa zmian i nieprzewidywalnych potrzeb w zakresie przepustowości i infrastruktury, pojawiają się one coraz częściej.⁶⁵

3. Amazon Web Services

3.1. Chmura obliczeniowa Amazon Web Services

Chmura obliczeniowa (z ang. cloud computing) to dostarczanie na żądanie zasobów technologii informatycznych i oprogramowania za pośrednictwem Internetu na żądanie. Niezależnie od tego, czy uruchamiane są aplikacje, które udostępniają zdjęcia milionom użytkowników mobilnych, czy też dostarczane usługi, które wspierają kluczowe operacje dla danej firmy, usługi te są dostępne na żądanie. Chmura zapewnia szybki dostęp do elastycznych i niedrogich zasobów IT. Dzięki chmurze obliczeniowej nie trzeba dokonywać dużych wstępnych inwestycji w sprzęt komputerowy i poświęcać dużo czasu na zarządzanie tym sprzętem. Zamiast tego, można dostarczyć dokładnie taki typ i wielkość zasobów obliczeniowych, jakich potrzeba, aby prowadzić dział IT. Dzięki chmurze obliczeniowej można uzyskać dostęp do dowolnej liczby zasobów, niemal natychmiast i płacić tylko za to, czego się używa. W najprostszej formie chmura obliczeniowa zapewnia łatwy dostęp do serwerów, pamięci magazynowych, baz danych i szerokiego zestawu usług aplikacyjnych przez Internet. Dostawcy usług przetwarzania w chmurze, tacy jak Amazon Web Services, posiadają i utrzymują podłączony do sieci sprzęt wymagany do świadczenia tych usług aplikacyjnych, podczas gdy firmy korzystające z chmury wykorzystują to, czego potrzebują do wykonywania swoich zadań. AWS zapewnia dostarczanie zasobów IT na żądanie za pośrednictwem Internetu na bezpiecznej platformie usług w chmurze, oferując moc obliczeniową, pamięć masową, bazy danych, dostarczanie treści i inne funkcje pomagające przedsiębiorstwom rozwijać się i skalować. Korzystanie z zasobów AWS zamiast z zasobów firmy jest jak zakup energii elektrycznej od firmy energetycznej zamiast uruchamiania własnego generatora i zapewnia kluczowe zalety chmury obliczeniowej: Wydajność dokładnie odpowiada zapotrzebowaniu, płaci się tylko za to, z czego się korzysta, korzyści skali skutkują niższymi kosztami, a także jest świadczona przez dostawcę posiadającego doświadczenie w prowadzeniu dużych sieci.⁶⁶

⁶⁵ Sarah Allen, Chris Aniszczuk, Chad Arimura, "CNCF WG-Serverless Whitepaper", The Cloud Native Computing Foundation, 2019, s. 34

⁶⁶ Joe Baron, Hisham Baz, Tim Bixler, Biff Gaut, Kevin E. Kelly, Sean Senior, John Stamper, "Certified Solutions Architect Official", John Wiley & Sons, Inc., 2017 Indianapolis, s. 45



Rysunek 10 Logo Amazon Web Services. Źródło: https://en.wikipedia.org/wiki/Amazon_Web_Services

AWS, którego logotyp został przedstawiony na rysunku 10 obsługuje ponad milion aktywnych klientów w ponad 190 krajach i stale rozbudowuje swoją globalną infrastrukturę, aby pomóc organizacjom osiągnąć mniejsze opóźnienia i większą przepustowość w zaspokajaniu ich potrzeb biznesowych. AWS zapewnia wysoce dostępną platformę technologiczną z wieloma lokalizacjami na całym świecie. Lokalizacje te składają się z regionów i stref dostępności. Każdy region jest oddzielnym obszarem geograficznym. Każdy region posiada wiele odizolowanych lokalizacji znanych jako dostępność. Strefy. AWS umożliwia lokowanie zasobów i danych w wielu lokalizacjach. Zasoby nie są replikowane we wszystkich regionach, chyba że organizacje się na to zdecydują. Każdy region jest całkowicie niezależny i zaprojektowany w taki sposób, aby był całkowicie odizolowany od innych regionów. Osiąga się w ten sposób największą możliwą tolerancję na błędy i stabilność. Każda Strefa dostępności jest również izolowana oraz strefy dostępności w danym regionie są połączone poprzez łącza o niskim opóźnieniu. Strefy dostępności są fizycznie oddzielone w obrębie typowej metropolii i znajdują się na obszarach o niższym stopniu zagrożenia powodziowego (specyficzna kategoryzacja stref powodziowych różni się w zależności od regionu). Oprócz korzystania z oddzielnych, nieprzerywalnych źródeł zasilania (UPS) i rezerwowych generatorów na miejscu, każdy z nich jest zasilany za pośrednictwem różnych sieci z niezależnych źródeł w celu dalszego ograniczenia liczby pojedynczych punktów awarii. Umieszczając zasoby w oddzielnych Strefach Dostępności, można chronić witrynę internetową lub aplikację przed zakłóceniami w świadczeniu usług w jednej konkretnej

lokalizacji.⁶⁷ Jako klient AWS, można wybierać pomiędzy różnymi centrami danych. Centra danych AWS są dystrybuowane na całym świecie. Na przykład, możesz uruchomić maszynę wirtualną w Japonii w dokładnie taki sam sposób, jak w Irlandii. Umożliwia to obsługę klientów na całym świecie dzięki globalnej infrastrukturze. Mapa na rysunku 11 przedstawia centra danych AWS. Dostęp jest ograniczony do niektórych z nich: niektóre centra danych są dostępne tylko dla amerykańskich organizacji rządowych, a specjalne warunki dotyczą centrów danych w Chinach.⁶⁸ W tabeli 1 wymieniono regiony udostępnione przez AWS:

Tabela 1 Lista regionów centrów danych chmury obliczeniowej Amazon Web Services. Tabela składa się z identyfikatora regionu, który jest niepowtarzalnym i zgodnym identyfikatorem dla strefy dostępności, oraz nazwa regionu.

Identyfikator regionu	Nazwa
us-east-1	US East (N. Virginia)
us-east-2	US East (Ohio)
us-west-1	US West (N. California)
us-west-2	US West (Oregon)
ca-central-1	Canada (Central)
eu-central-1	EU (Frankfurt)

⁶⁷ Mike Ryan & Federico Lucifredi, "AWS System Administration Best Practices for Sysadmins in the Amazon Cloud", O'Reilly Media, Inc., 2018, s. 13

⁶⁸ MICHAEL WITTIG ANDREAS WITTIG, "Amazon Web Services in Action, Second Edition", Manning Publications Co., 2018, s. 7-8

eu-west-1	EU (Ireland)
eu-west-2	EU (London)
eu-west-3	EU (Paris)
eu-north-1	EU (Stockholm)
ap-east-1	Asia Pacific (Hong Kong)
ap-northeast-1	Asia Pacific (Tokyo)
ap-northeast-2	Asia Pacific (Seoul)
ap-northeast-3	Asia Pacific (Osaka-Local)
ap-southeast-1	Asia Pacific (Singapore)
ap-southeast-2	Asia Pacific (Sydney)
ap-south-1	Asia Pacific (Mumbai)
sa-east-1	South America (São Paulo)

Kolorem pomarańczowym na rys. 11 oznaczono lokalizacje, w których powstaną w najbliższych latach kolejne centra danych. Są to kolejno:

- Milan
- Bahrajn
- Dżakarta
- Cape Town⁶⁹



Rysunek 11 Mapa centrów danych chmury obliczeniowej Amazon Web Services. Źródło: <https://aws.amazon.com/about-aws/global-infrastructure/>

⁶⁹ <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html>

3.2. Bezpieczeństwo

Bezpieczeństwo informacji w chmurze obliczeniowej Amazon Web Services ma kluczowe znaczenie dla organizacji wykonujących zadania o znaczeniu krytycznym. Bezpieczeństwo jest podstawowym wymogiem funkcjonalnym, który chroni informacje o znaczeniu krytycznym przed przypadkową lub celową kradzieżą, wyciekiem, naruszeniem integralności i usunięciem. Pomoc w ochronie poufności, integralności i dostępności systemów i danych jest dla AWS sprawą najwyższej wagi, podobnie jak utrzymanie zaufania klientów oraz ich pewności siebie.⁷⁰

Chmura obliczeniowa jest środowiskiem współodpowiedzialności, co oznacza, że odpowiedzialność jest dzielona pomiędzy klienta i AWS. AWS odpowiada za następujące kwestie:

- Ochrona sieci poprzez zautomatyzowane systemy monitorowania i zapewnienie niezawodnego dostępu do Internetu, aby zapobiec atakom typu Distributed Denial of Service (DDoS).
- Przeprowadzanie kontroli pracowników, którzy mają dostęp do obszarów wrażliwych.
- Likwidacja urządzeń pamięci masowej poprzez ich fizyczne zniszczenie po zakończeniu eksploatacji.
- Zapewnienie fizycznego i środowiskowego bezpieczeństwa centrów danych, w tym ochrony przeciwpożarowej, i bezpieczeństwa pracowników.

Z kolei obowiązki klienta:

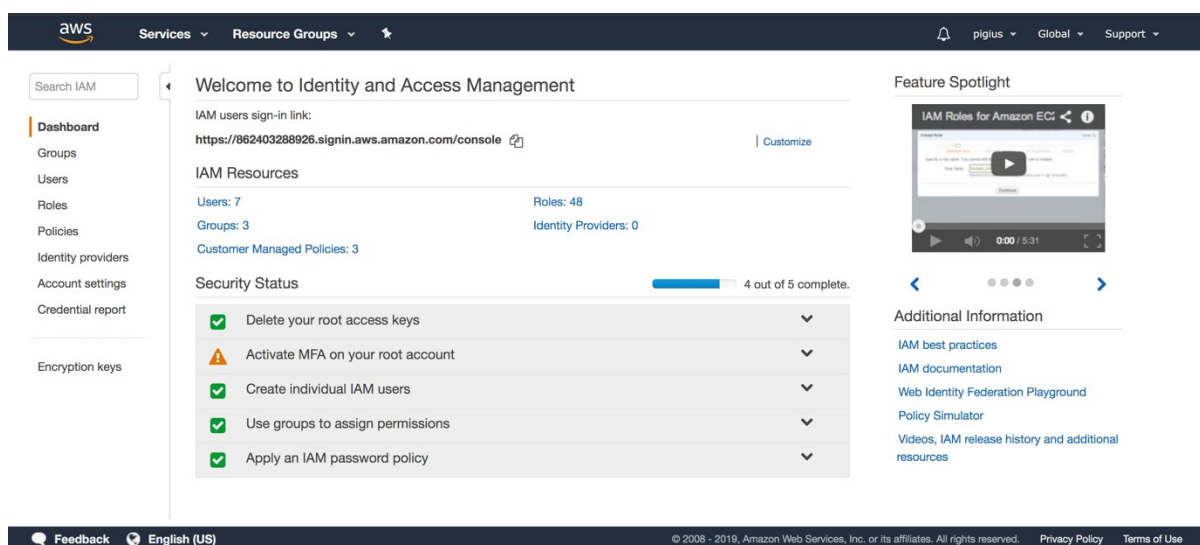
- Wdrożenie zarządzania dostępem, które ogranicza dostęp do zasobów AWS takich jak magazynowanie danych (np. Amazon Simple Storage Service) do minimum, przy użyciu narzędzia AWS IAM (ang. Identity and Access Management - Zarządzanie tożsamością i dostępem).
- Szyfrowanie ruchu sieciowego w celu uniemożliwienia intruzom odczytywania lub manipulowania danymi (np. przy użyciu protokołu HTTPS).

⁷⁰ Uchit Vyas, "Mastering AWS Development", 2015, Packt Publishing Ltd, s. 24

- Konfiguracja zapory sieci wirtualnej, która kontroluje ruch przychodzący i wychodzący za pomocą grup bezpieczeństwa i punktów dostępu ACL.
- Szyfrowanie danych w trybie spoczynku. Na przykład włączone szyfrowanie danych dla bazy danych lub innych systemów pamięci masowej.
- Zarządzanie poprawkami do systemu operacyjnego i dodatkowego oprogramowania na maszynach wirtualnych.

Bezpieczeństwo wiąże się z interakcją pomiędzy AWS, a klientem. W przypadku korzystania z systemu AWS przez można osiągnąć wysokie standardy bezpieczeństwa w chmurze.⁷¹

AWS Identity and Access Management (IAM) jest usługą umożliwiającą zarządzanie użytkownikami, grupami i uprawnieniami użytkowników w ramach infrastruktury AWS. Pozwala to na centralną kontrolę użytkowników, grup, dostępu użytkowników i certyfikatów bezpieczeństwa. W związku z dużą ilością usług oferowanych przez AWS istnieje potrzeba bezpiecznego dostępu do tych usług przez upoważnionych użytkowników. IAM definiuje koncepcje, konstrukcje i usługi służące osiągnięciu tego celu. Wygląd interfejsu użytkownika IAM, został przedstawiony na rysunku 12.



Rysunek 12 Widok interfejsu graficznego usługi IAM w Amazon Web Services. Opracowanie własne.

IAM rozwiązuje następujące problemy:

- Ustalanie zakresu odpowiedzialności: Zapewnia to dostęp i wymagane zezwolenia tylko do usług, których potrzebuje użytkownik. Na przykład, aplikacja internetowa

⁷¹ <https://aws.amazon.com/compliance/>

potrzebuje zapisywanie uprawnień do konkretnego pojemnika w obrębie usługi S3, zamiast przypisywania zapisu na pozwolenie na całą usługę S3.

- Dystrybucja uprawnień: Ułatwia to dystrybucję i rotację danych uwierzytelniających do użytkowników, usług AWS, instancji oraz aplikacji w bezpieczny sposób.
- Zarządzanie dostępem dla użytkowników zrzeszonych: Użytkownicy zrzeszeni to użytkownicy, którzy są zarządzani poza IAM. Zazwyczaj są to użytkownicy w spisie firmowym. IAM umożliwia przyznanie dostępu do zasobów AWS federalnym użytkownikom; jest to osiągnięte poprzez przyznanie tymczasowych certyfikatów bezpieczeństwa zrzeszonemu użytkownikowi.

AWS IAM posiada również role. Rola jest zestawem pozwoleń, które dają dostęp do zasobów AWS. Role nie są związane z żadnym użytkownikiem lub grupą, lecz są zakładane przez zaufaną jednostkę, która może być użytkownikiem IAM, aplikacją lub usługą AWS, taką jak S3. Różnica między użytkownikiem IAM a rolą polega na tym, że rola nie może uzyskać dostępu do zasobów AWS bezpośrednio sugerując, że nie posiadają oni żadnych danych uwierzytelniających. Właściwość ta jest bardzo przydatna, gdy zaufane usługi AWS, takie jak S3, pełnią swoją rolę; nie ma potrzeby dostarczania danych uwierzytelniających do usługi S3. Rozwiązuje to bardzo ważny problem, jakim jest dystrybucja uprawnień oraz rotacja plus nieposiadanie danych uwierzytelniających przechowywanych jako czysty tekst lub w formie zaszyfrowanej.⁷²

3.3. Wybrane usługi AWS

AWS oferuje ponad 200 usług w następujących kategoriach:

- Analityka,
- Rozwiązania na desktopy oraz aplikacje streamingowe,
- Usługi medialne
- Integracja aplikacji
- Narzędzia programistyczne
- Migracja
- AR (ang. Advanced Reality, rozszerzona rzeczywistość) i VR (ang. virtual reality, wirtualna rzeczywistość)
- Rozwój gier

⁷² https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html

- Usługi telefonii komórkowej
- Wydajność biznesowa
- Internet Wszechrzeczy (IoT)
- Tworzenie sieci i dostarczanie treści
- Obliczenia (ang. computing)
- Uczenie Maszynowe (ang. Machine Learning)
- Ochrona,
- Uwierzytelnienie
- Zaangażowanie klienta
- Narzędzia zarządzania
- Przechowywanie danych
- Bazy danych⁷³

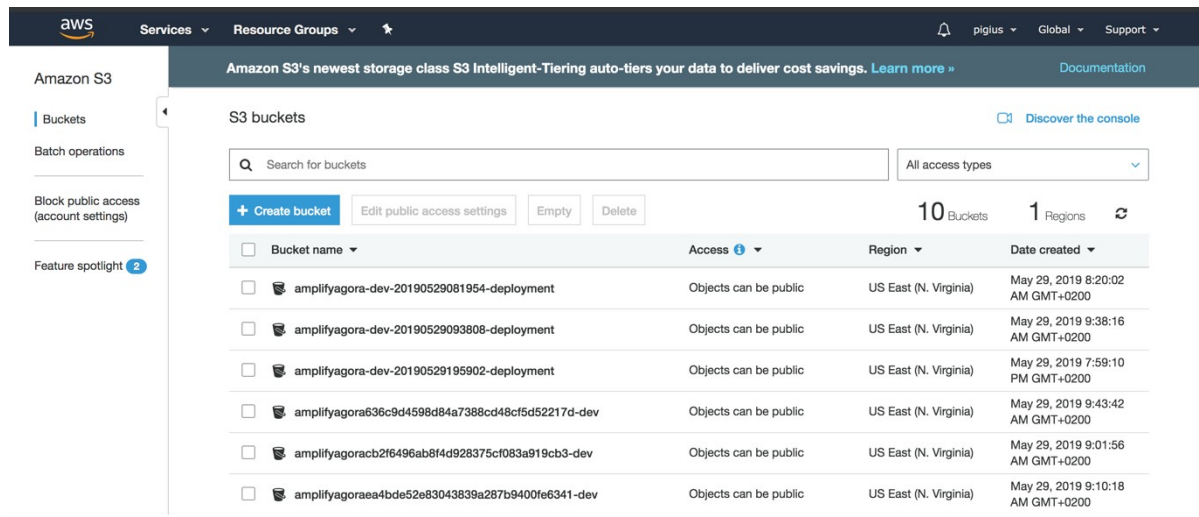
Wybrane usługi, które będą wykorzystane w projekcie praktycznym to:

- Amazon Simple Storage Service
- Amazon Cognito
- Amazon Lambda
- Amazon DynamoDB
- AWS AppSync

Amazon Simple Storage Service (Amazon S3) - zapewnia deweloperom i zespołom IT trwałą i skalowalną przestrzeń dyskową obiektów, która obsługuje praktycznie nieograniczone ilości danych i dużą liczbę jednoczesnych użytkowników. Organizacje mogą przechowywać dowolną ilość obiektów dowolnego typu, takich jak strony HTML, pliki kodu źródłowego, pliki graficzne, zaszyfrowane dane i uzyskiwać do nich dostęp za pomocą protokołów HTTP. Amazon S3 zapewnia ekonomiczne przechowywanie obiektów dla wielu różnych przypadków użycia, w tym tworzenia kopii zapasowych i odzyskiwania danych, archiwum danych, analizy dużych danych, odzyskiwania danych po awarii, aplikacji w chmurze i dystrybucji treści. Idea stojąca za S3 jest prosta. Używa się go do przechowywania obiektów dowolnego rodzaju w wybranej strukturze katalogów, nie obawiając się o sposób przechowywania obiektów. Celem jest przechowywanie obiektów bez uwzględniania podstawowej infrastruktury, tak aby można skupić się na potrzebach aplikacji z pozornie

⁷³ <https://aws.amazon.com/>

nieskończoną ilością obiektów na dysku twardym. S3 skupia się na obiektach, które mogą być dowolnego typu, a nie na strukturze katalogów składającej się z folderów zwanych pojemnikami (ang. buckets) przechowujących pliki. Oczywiście, nadal można organizować dane do folderów, a S3 może przechowywać pliki, jednak chodzi o to, że S3 nie dba o to, jakiego rodzaju pliki są przechowywane. Dany pojemnik automatycznie rozrasta się i kurczy, aby zmieściły się w nim dane, które się w nim umieszczają. Nie istnieją żadne granice dotyczące ilości oraz pojemności danych w przypadku S3. Nawet jeśli podczas pracy z pojemnikami S3 w jednym regionie, Amazon przechowuje dane w wielu lokalizacjach w wielu lokalizacjach w regionie. Automatyczna replikacja danych zmniejsza ryzyko ich utraty w wyniku dowolnej liczby klęsk żywiołowych lub innych rodzajów katastrof. Żadna technologia pamięci masowej nie jest niezawodna, przez to, dane w aplikacjach klientów mogą ulec zniszczeniu w przypadkach takich, jak klęski żywiołowe. W związku z tym, aby chronić dane, należy postępować zgodnie z najlepszymi praktykami. S3 posiada takie rozwiązanie jak wersjonowanie (ang. versioning) które zapewnia zabezpieczenie, dzięki któremu można odzyskać poprzednią wersję obiektu, gdy użytkownik usunie go lub aplikacja uszkodzi go w jakiś sposób.⁷⁴ Na rysunku 13 przedstawiono wygląd usługi S3 w interfejsie graficznym użytkownika AWS

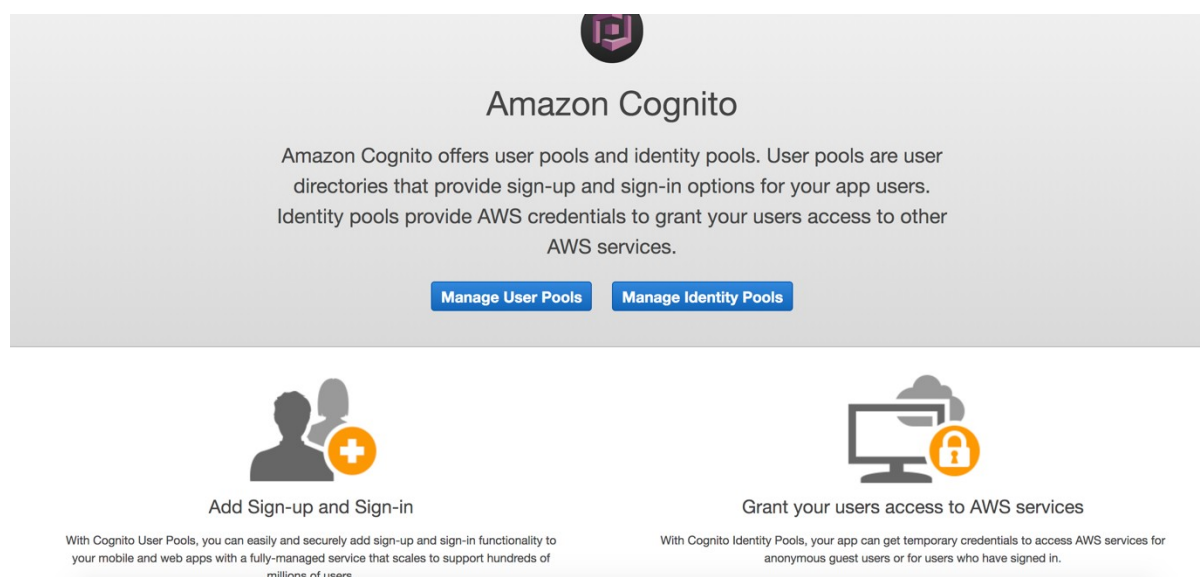


Rysunek 13 Wygląd usługi S3 w AWS. Opracowanie własne

Amazon Cognito zapewnia usługi identyfikacji i synchronizacji dla aplikacji mobilnych i internetowych. Upraszcza zadanie uwierzytelniania użytkowników oraz przechowywania, zarządzania i synchronizacji ich danych na wielu urządzeniach, platformach i aplikacjach.

⁷⁴ John Paul Mueller, "AWS For Admins", John Wiley & Sons, Inc, 2017 Hoboken, s. 146-147

Dostarcza tymczasowych, ograniczonych danych uwierzytniających zarówno uwierzytelnionym, jak i niewierzytelnionym użytkownikom bez konieczności zarządzania infrastrukturą typu back-end. Amazon Cognito współpracuje ze znanymi dostawcami tożsamości, takimi jak Google, Facebook i Amazon, aby uwierzytelniać użytkowników końcowych aplikacji mobilnych i internetowych. Można skorzystać z funkcji identyfikacji i autoryzacji zapewnianych przez te usługi zamiast budować i utrzymywać swoje własne. Dana aplikacja uwierzytelnia się z jednym z tych dostawców tożsamości za pomocą zestawu SDK dostawcy. Po uwierzytelnieniu użytkownika końcowego z dostawcą, zwrócony od dostawcy token OAuth lub OpenID Connect jest przekazywany przez daną aplikację do Amazon Cognito, która zwraca nowy identyfikator Amazon Cognito ID dla użytkownika oraz zestaw tymczasowych, ograniczonych danych uwierzytniających AWS. Ponadto, Amazon Cognito umożliwia synchronizację danych w urządzeniach użytkownika, tak aby ich działanie aplikacji pozostawało spójne podczas przełączania między urządzeniami lub aktualizacji do nowego urządzenia. Aplikacja może zapisywać dane lokalnie na urządzeniach użytkowników, umożliwiając działanie aplikacji nawet wtedy, gdy urządzenia są wyłączone, a następnie automatycznie synchronizować dane po powrocie urządzenia do sieci. Dzięki Amazon Cognito można skupić się na tworzeniu wspaniałych aplikacji, zamiast martwić się o budowanie, zabezpieczanie i skalowanie rozwiązania do zarządzania użytkownikami, uwierzytelniania i synchronizacji między platformami i urządzeniami.⁷⁵ Wygląd interfejsu graficznego usługi Cognito przedstawiono na rysunku 14.

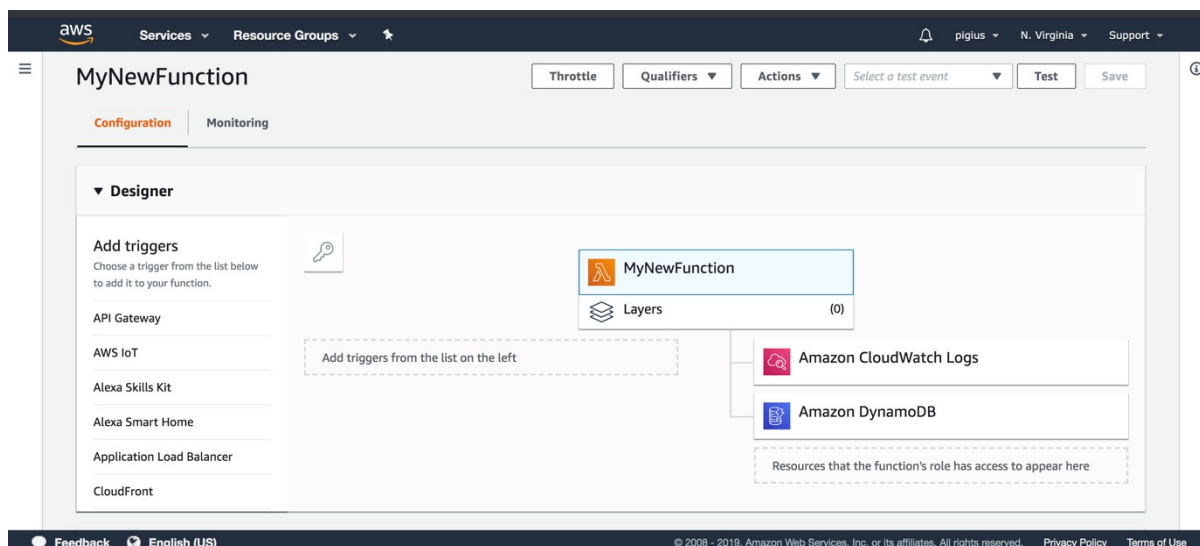


Rysunek 14 Wygląd usługi Cognito w AWS. Opracowanie własne.

⁷⁵ <https://aws.amazon.com/cognito/>

AWS Lambda jest flagową usługą chmury obliczeniowej Amazon Web Services. Jest usługą obliczeniową, która uruchamia kod źródłowy w odpowiedzi na zdarzenia (ang. events) i automatycznie zarządza zasobami obliczeniowymi za programistę. Można użyć AWS Lambda do rozszerzenia innych usług AWS z własną logiką lub stworzyć własne usługi typu back-end, które działają w oparciu o skalowalność, wydajność i bezpieczeństwo AWS. AWS Lambda może automatycznie uruchamiać kod w odpowiedzi na wiele zdarzeń, takich jak żądania HTTP poprzez usługi, modyfikacje obiektów w pojemnikach Amazon S3, aktualizacje tabel w bazie danych Amazon DynamoDB oraz przejścia stanów między usługami. Lambda uruchamia kod na infrastrukturze obliczeniowej o wysokiej dostępności i wykonuje wszystkie czynności administracyjne związane z zasobami obliczeniowymi, w tym konserwację serwerów i systemów operacyjnych, udostępnianie pojemności i automatyczne skalowanie, wdrażanie poprawek kodu i zabezpieczeń oraz monitorowanie kodu i logowanie. Wszystko, co musi zrobić programista, to dostarczyć kod. Kod, który uruchamia się na AWS Lambda jest nazywany "funkcją Lambda". Po utworzeniu funkcji lambda jest ona zawsze gotowa do uruchomienia, gdy tylko zostanie uruchomiona, podobnie jak formuła w arkuszu kalkulacyjnym. Każda funkcja zawiera kod źródłowy, jak również niektóre powiązane informacje konfiguracyjne, w tym nazwę funkcji i wymagania dotyczące zasobów. Funkcje Lambda są "bezstanowe", bez związku z podstawową infrastrukturą, dzięki czemu Lambda może szybko uruchomić tyle kopii funkcji, ile potrzeba, aby skalować do tempa przychodzących zdarzeń.⁷⁶ Wygląd interfejsu graficznego usługi Lambda przedstawiono na rysunku 15.

⁷⁶ <https://aws.amazon.com/lambda/features/>



Rysunek 15 Wygląd usługi Lambda w AWS. Opracowanie własne.

Amazon DynamoDB jest to szybka i elastyczna usługa bazy danych NoSQL dla dowolnej skali. Amazon DynamoDB jest bazą typu „Klucz–Wartość” i bazą danych dokumentów, która zapewnia jednocyfrową, milisekundową wydajność w dowolnej skali. Jest to w pełni zarządzana, wieloregionowa, wielozakresowa baza danych z wbudowanymi funkcjami bezpieczeństwa, tworzenia kopii zapasowych i przywracania danych oraz pamięci podręcznej dla aplikacji na skalę internetową. DynamoDB może obsłużyć ponad 10 trylionów zapytań dziennie i może obsłużyć ponad 20 milionów zapytań na sekundę. Wiele z najszybciej rozwijających się firm na świecie, takich jak Tinder, Airbnb i Redfin, a także przedsiębiorstw takich jak Samsung, Toyota i Capital One, opiera się na skali i wydajności DynamoDB, aby wspierać ich kluczowe dla ich misji zadania. Wygląd interfejsu graficznego usługi Lambda przedstawiono na rysunku 16.

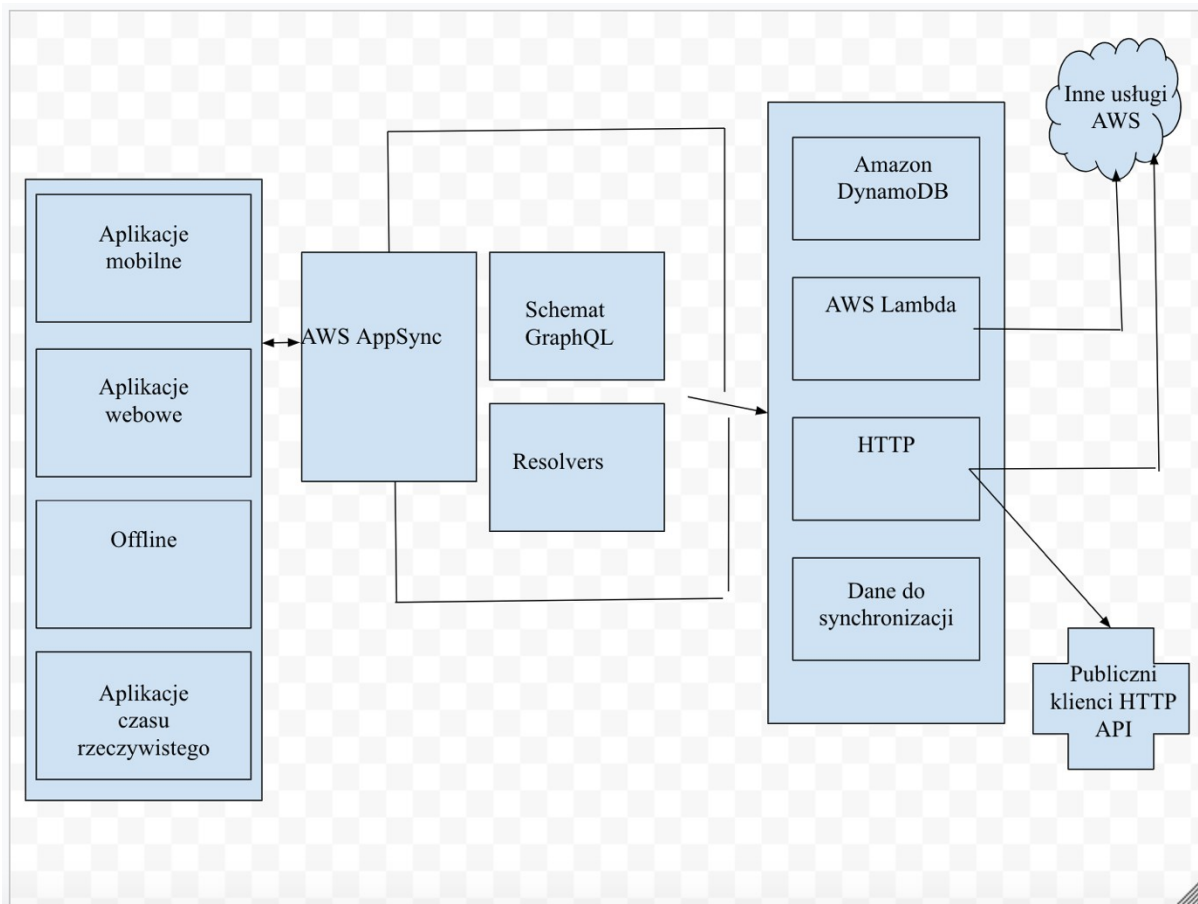
Name	Status	Partition key	Sort key	Indexes	Total read capacity	Total write capacity
Market-66ullicsi5c4lgq2cdit72oute-d	Active	id (String)	-	0	On-Demand	On-Demand
Market-hybxrwu5mzbjrfzyan44vqu4i	Active	id (String)	-	0	On-Demand	On-Demand
Order-66ullicsi5c4lgq2cdit72oute-de	Active	id (String)	-	1	On-Demand	On-Demand
Order-hybxrwu5mzbjrfzyan44vqu4u	Active	id (String)	-	1	On-Demand	On-Demand
Product-66ullicsi5c4lgq2cdit72oute-i	Active	id (String)	-	1	On-Demand	On-Demand
Product-hybxrwu5mzbjrfzyan44vqu4u	Active	id (String)	-	1	On-Demand	On-Demand
User-66ullicsi5c4lgq2cdit72oute-dev	Active	id (String)	-	0	On-Demand	On-Demand
User-hybxrwu5mzbjrfzyan44vqu4u-i	Active	id (String)	-	0	On-Demand	On-Demand

Rysunek 16 Wygląd usługi DynamoDB w AWS. Opracowanie własne.

AWS AppSync upraszcza tworzenie aplikacji, pozwalając na stworzenie elastycznego API w celu bezpiecznego dostępu, modyfikowania i łączenia danych z jednego lub więcej źródeł danych. AppSync jest usługą zarządzaną, która wykorzystuje GraphQL, aby ułatwić aplikacjom uzyskanie dokładnie takich danych, jakich potrzebują. GraphQL to język danych umożliwiający aplikacjom klienckim pobieranie, zmianę i subskrypcję danych z serwerów. W zapytaniu GraphQL klient określa, w jaki sposób dane mają być ustrukturyzowane, gdy są zwracane przez serwer. Umożliwia to klientowi wyszukiwanie tylko tych danych, które są mu potrzebne, w formacie, w jakim są one potrzebne. Dzięki AppSync można budować skalowalne aplikacje, w tym wymagające aktualizacji w czasie rzeczywistym, na wielu źródłach danych, takich jak bazy danych NoSQL, relacyjne bazy danych, interfejsy API HTTP oraz własne źródła danych z AWS Lambda. W przypadku aplikacji mobilnych i internetowych, AppSync dodatkowo zapewnia lokalny dostęp do danych, gdy urządzenia są wyłączone, oraz synchronizację danych z konfigurowalnym rozwiązywaniem konfliktów, gdy są one ponownie włączone. AWS AppSync umożliwia subskrypcje w czasie rzeczywistym na miliony urządzeń, jak również dostęp do danych aplikacji w trybie offline. Kiedy urządzenie offline łączy się ponownie, AWS AppSync synchronizuje tylko aktualizacje, które miały miejsce po odłączeniu urządzenia, a nie całą bazę danych. AWS AppSync oferuje konfigurowalne dla użytkownika rozwiązywanie konfliktów po stronie serwera, które powoduje, że zarządzanie konfliktami danych nie jest konieczne.⁷⁷

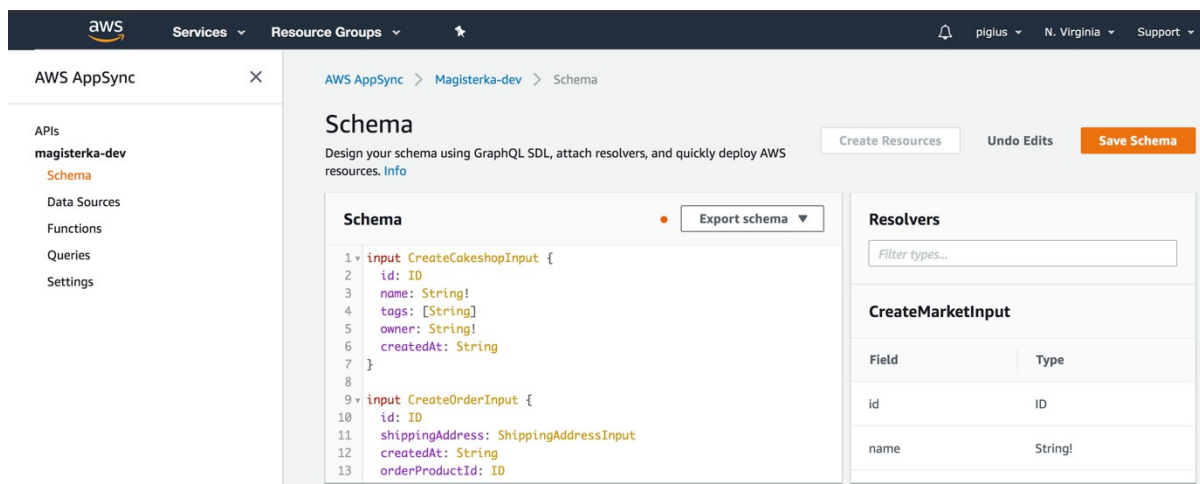
⁷⁷ <https://aws.amazon.com/appsync/>

Wizualny schemat działania AppSync przedstawiono na rysunku 17. Aplikacje mobilne, webowe, czasu rzeczywistego, jak i aplikacje działające offline komunikują się z AWS AppSync. AppSync zapewnia bezpieczny dostęp i łączy dane z baz danych, interfejsów API i innych systemów backendowych. Następnie automatycznie poprzez schemat aplikacji zapisany w GraphQL oraz resolvers. Resolver to funkcja lub metoda, zwracająca wartość dla typu lub pola, zdefiniowanych w schemacie opisującym API serwera. Następnie AppSync łączy się z bazą danych DynamoDB, funkcjami lambda, które następnie mogą łączyć się z innymi usługami AWS, oraz z publicznymi klientami API.



Rysunek 17 Schemat działania usługi AppSync w AWS. Opracowanie własne.

Wygląd interfejsu graficznego usługi Appsync przedstawiono na rysunku 18.



Rysunek 18 Wygląd usługi AppSync w AWS. Opracowanie własne.

4. Projekt aplikacji wirtualnego rynku sprzedaży tortów

4.1. Wykorzystane technologie

Opracowany projekt umożliwia stworzenie wirtualnego rynku sprzedaży tortów. Właściciele cukierni mogą tworzyć własny sklep i w nim wystawiać swoje produkty, a użytkownicy mogą zarówno kupować gotowe produkty w postaci tortów, jak i wirtualne, w postaci przepisów na torty. Aplikacja wykorzystuje nowoczesne rozwiązania, służące do budowania niezawodnych oraz skalowalnych rozwiązań, bez względu na ruch w aplikacji, które działają stabilnie, i zapewniają ciągłość ich żywotności.

Aplikację podzielono na część wizualną oraz część serwerową. W części serwerowej użyto gotowych usług chmury obliczeniowej Amazon Web Services, aby aplikacja była napisana w architekturze Serverless. Z kolei część wizualna służy do dostarczenia interfejsu wizualnego dla użytkownika oraz aby komunikować się z częścią serwerową, przez co można zarówno wysyłać i pobierać dane, a na końcu wyświetlać te dane użytkownikowi.

4.1.1. Część Wizualna



Rysunek 19 Logo frameworku ReactJS, który jest wykorzystywany do zbudowania interfejsu użytkownika. Logo pobrane z <http://www.danielledeveloper.com/reactjs-lets-talk-about-it/>

Aplikacja jest napisana z wykorzystaniem frameworku ReactJS, który odpowiada za interfejs użytkownika. Logo frameworku zostało przedstawione na rysunku 19. Jest to technologia SPA (Single Page Application), dzięki której użytkownik postrzega działanie aplikacji w czasie rzeczywistym, ponieważ strona nie przeładowuje się jak ma się to w przypadku stron statycznych. Poszczególne elementy, takie jak strona główna, strona wyszukiwarki są w aplikacji zdefiniowane jako komponenty, które są niezależne od siebie, i przesyłane są tam tylko potrzebne dane, które potem są przetwarzane i wyświetlane wewnątrz poszczególnego komponentu. Posłużono się przykładem komponentu to wyświetlania danych cukierni w tabeli. Na rysunku numer 20 przedstawiono przykładowy kod implementacji wyświetlania danych o cukierniach w tabeli. Znajduje się tam kilka komponentów, odpowiedzialnych za zbudowanie takiej tabeli. Istotną kwestią jest fakt, iż przekazujemy tylko potrzebne dane, które komponent potrzebuje do zbudowania tabeli. W tym przypadku jest to ``userProgressTableData``, czyli dane o cukierniach.

```

<Col md="6" sm="12" xs="12">
  <Card> Ilkwon Sim, a year ago • Init commit
    <CardHeader>Cukiernie</CardHeader>
    <CardBody>
      <UserProgressTable
        headers=[
          <MdPersonPin size={25} />,
          'Nazwa cukierni',
          'Ilość produktów',
        ]
        usersData={userProgressTableData}
      />
    </CardBody>
  </Card>
</Col>

```

Rysunek 20 Przykład implementacji napisany w ReactJS do wyświetlania danych o cukierni, wyświetlanych w tabeli. Opracowanie własne.

4.1.2. Część serwerowa

Część serwerowa używa usług Amazon Web Services, potrzebnych do stworzenia tej aplikacji. W celach implementacyjnych oraz przyspieszenia pracy użyto framework Amazon Amplify Framework⁷⁸. Jest to interfejs linii poleceń Amplify (CLI) który jest zintegrowanym łańcuchem narzędzi do tworzenia, integrowania i zarządzania usługami w chmurze AWS dla aplikacji. Logo frameworku można zobaczyć na rysunku 21. Dzięki Amplify framework, programista może przyspieszyć pracę związaną z implementacją podejścia Serverless z chmurą Amazon Web Services, zarówno przy tworzeniu aplikacji webowych, jak i aplikacji mobilnych. Główną zaletą jest fakt, iż zamiast ręcznie ustawiać wszystko w chmurze obliczeniowej Amazon Web Services, programista może zrobić to poprzez wpisanie odpowiednich poleceń w interfejsie linii poleceń, a cała reszta zostanie wykonana poprzez wbudowane metody Amplify.

⁷⁸ <https://aws-amplify.github.io/docs/>



Rysunek 21 Logo frameworku Amplify Framework wykorzystywanego w projekcie. Źródło: <https://aws-amplify.github.io/>

Rysunek 22 przedstawia wygląd interfejsu linii poleceń Amplify (CLI). Przedstawia on wszystkie wbudowane metody służące do wdrażania i komunikacji z usługami Amazon Web Services. Są to między innymi usługi analityczne, klientów API, autentyfikacji, funkcji FaaS, hostingowe, interakcji, powiadomień, powierzchni dla danych, jaki i rozszerzonej rzeczywistości. Ponadto przedstawione są wbudowane metody, takie jak konfiguracja projektu pod Amplify, status obecnych użytych funkcji, możliwość dodawania i usuwania, a ponadto publikowania gotowych rozwiązań do chmury obliczeniowej.

```
→ magisterka git:(master) ✗ amplify

Category
-----
analytics
api
auth
function
hosting
interactions
notifications
storage
xr

AMPLIFY FRAMEWORK

amplify <command> <subcommand>

init          Initializes a new project, sets up deployment resources in the cloud, and makes your project ready for Amplify.
configure     Configures the attributes of your project for amplify-cli, such as switching front-end framework and adding/removing cloud-provider plugins.
push          Provisions cloud resources with the latest local developments.
publish       Executes amplify push, and then builds and publishes client-side application for hosting.
serve         Executes amplify push, and then executes the project's start command to test run the client-side application locally.
status        Shows the state of local resources not yet pushed to the cloud (Create/Update/Delete).
delete        Deletes all of the resources tied to the project from the cloud.
              <category> add      Adds a resource for an Amplify category in your local backend.
              <category> update  Update resource for an Amplify category in your local backend.
              <category> push    Provisions all cloud resources in a category with the latest local developments.
              <category> remove  Removes a resource for an Amplify category in your local backend.
              <category>        Displays subcommands of the specified Amplify category.
codegen       Generates GraphQL statements(queries, mutations and subscriptions) and type annotations.
env           Displays and manages environment related information for your Amplify project.
```

Rysunek 22 Wygląd Interfejsu linii poleceń Amplify. Opracowanie własne.

Na potrzeby projektu użyto tylko kilka funkcji Amplify, między innymi Hosting, który odpowiada za stworzenie po stronie chmury, a dokładnie Amazon Simple Storage Service (S3) w celu stworzenia powierzchni na zdjęcia użytkowników. Dzięki użyciu Amplify od razu programista dostaje link do rzeczywistego kontenera danych który jest udostępniony na zewnątrz, tak jak widać na rysunku 23, który pokazuje zastosowanie Amplify framework w projekcie Tortowo. Ponadto wykorzystany jest klient API oraz funkcje do autentyfikacji użytkowników.

```
env Displays and manages environment related information for your Amplify project.
→ magisterka git:(master) ✕ amplify status

Current Environment: dev

| Category | Resource name | Operation | Provider plugin |
| ----- | ----- | ----- | ----- |
| Hosting | S3AndCloudFront | No Change | awscloudformation |
| Auth | magisterka9d5f702d | No Change | awscloudformation |
| Api | magisterka | No Change | awscloudformation |

GraphQL endpoint: https://zbcv1vmecbhnxcws5hff76pala.appsync-api.us-east-1.amazonaws.com/graphql
Hosting endpoint: http://magisterka-20190528174413-hostingbucket-dev.s3-website-us-east-1.amazonaws.com

→ magisterka git:(master) ✕
```

Rysunek 23 Wygląd Amplify framework na przykładzie projektu Tortowo. Opracowanie własne

Wewnątrz aplikacji użyto następujących usług Amazon Web Services:

- Amazon Cognito,
- Amazon AppSync,
- Amazon Simple Email Service, (SES),
- Amazon Simple Storage Service (S3),
- AWS Lambda

Do zbudowania procesu autentyfikacji użytkowników narzędzia Amazon Cognito.. Dzięki temu można zarówno logować się do aplikacji, jak i rejestrować, a także przypomnieć zapomniane hasło. Całość procesu odbywa się po stronie chmury obliczeniowej, co jest w pełni bezpieczne i odporne na błędy. Cognito zostało wdrożone przy pomocy narzędzia Amplify. Wstępna konfiguracja wygląda tak jak na rysunku numer 24. Należy zaimportować wewnątrz aplikacji interfejsu użytkownika potrzebne moduły, a następnie uzupełnić potrzebne dane.


```

You, a few seconds ago | 1 author (You)
import Amplify from 'aws-amplify';

Amplify.configure({
  Auth: {
    // REQUIRED only for Federated Authentication - Amazon Cognito Identity Pool ID
    identityPoolId: 'XX-XXXX-X:XXXXXXXX-XXXX-1234-abcd-1234567890ab',

    // REQUIRED - Amazon Cognito Region
    region: 'XX-XXXX-X',

    // OPTIONAL - Amazon Cognito Federated Identity Pool Region
    // Required only if it's different from Amazon Cognito Region
    identityPoolRegion: 'XX-XXXX-X',

    // OPTIONAL - Amazon Cognito User Pool ID
    userPoolId: 'XX-XXXX-X_abcd1234',

    // OPTIONAL - Amazon Cognito Web Client ID (26-char alphanumeric string)
    userPoolWebClientId: 'a1b2c3d4e5f6g7h8i9j0k1l2m3',

    // OPTIONAL - Enforce user authentication prior to accessing AWS resources or not
    mandatorySignIn: false,

    // OPTIONAL - Configuration for cookie storage
    // Note: if the secure flag is set to true, then the cookie transmission requires a secure protocol
    cookieStorage: {
      // REQUIRED - Cookie domain (only required if cookieStorage is provided)
      domain: '.yourdomain.com',
      // OPTIONAL - Cookie path
      path: '/',
      // OPTIONAL - Cookie expiration in days
      expires: 365,
      // OPTIONAL - Cookie secure flag
      // Either true or false, indicating if the cookie transmission requires a secure protocol (https).
      secure: true
    },

    // OPTIONAL - customized storage object
    storage: new MyStorage(),

    // OPTIONAL - Manually set the authentication flow type. Default is 'USER_SRP_AUTH'
    authenticationFlowType: 'USER_PASSWORD_AUTH'
  }
});

```

Rysunek 24 Konfiguracja Amazon Cognito, przy pomocy Amplify Framework. Opracowanie własne.

AWS AppSync upraszcza tworzenie aplikacji, pozwalając na stworzenie elastycznego API w celu bezpiecznego dostępu, manipulowania i łączenia danych z jednego lub więcej źródeł danych. AppSync jest usługą zarządzaną, która wykorzystuje GraphQL, aby ułatwić aplikacjom uzyskanie dokładnie takich danych, jakich w danej chwili potrzebują. Z tego też powodu, w projekcie wykorzystano tę usługę. Łączy ona w sobie zarówno usługi bazy danych, klienta API, a także funkcje AWS Lambda. Schemat GraphQL obiektów w aplikacji wygląda następująco jak na rysunku 25.

```

type Cakeshop @model @searchable {
  id: ID!
  name: String!
  products: [Product]
  @connection(name: "CakeshopProducts", sortField: "createdAt")
  owner: String!
  createdAt: String
}

type Product @model @auth(rules: [{ allow: owner, identityField: "sub" }]) {
  id: ID!
  description: String!
  cakeshop: Cakeshop @connection(name: "CakeshopProducts")
  file: S3Object!
  price: Float!
  shipped: Boolean!
  owner: String
  createdAt: String
}

type S3Object {
  bucket: String!
  region: String!
  key: String!
}

```

Rysunek 25 Schemat GraphQL dla usługi AWS Appsync w aplikacji Tortowo. Opracowanie własne

W aplikacji możemy wyróżnić:

1. „CakeShop” czyli cukiernia z atrybutami „id” jako identyfikator cukierni, nazwę, asocjacje do produktów, asocjacje do właściciela, a także datę stworzenia
2. „Product” jako produkt wewnątrz cukierni np. tort. Atrybutami są identyfikator, opis, asocjacje do „CakeShop”, cenę, informację czy produkt został wysłany, asocjacje do właściciela produktu, datę stworzenia, a także zdjęcie produktu jak „S3Object” który jest przetrzymywany w usłudze Amazon Simple Storage Service (S3).
3. „S3Object” składający się z nazwy pojemnika, regionu, oraz klucza identyfikującego w usłudze Amazon Simple Storage Service (S3).

Ponadto w aplikacji możemy wyróżnić dodatkowe schematy obiekty przedstawione na rysunku 26.

```

type User
  @model(
    queries: { get: "getUser" }
    mutations: { create: "registerUser", update: "updateUser" }
    subscriptions: null
  ) {
    id: ID!
    username: String!
    email: String!
    registered: Boolean
    orders: [Order] @connection(name: "UserOrders", sortField: "createdAt")
  }

type Order
  @model(
    queries: null
    mutations: { create: "createOrder" }
    subscriptions: null
  ) {
    id: ID!
    product: Product @connection
    user: User @connection(name: "UserOrders")
    shippingAddress: ShippingAddress
    createdAt: String
  }

type ShippingAddress {
  city: String!
  country: String!
  address_line1: String!
  address_state: String!
  address_zip: String!
}

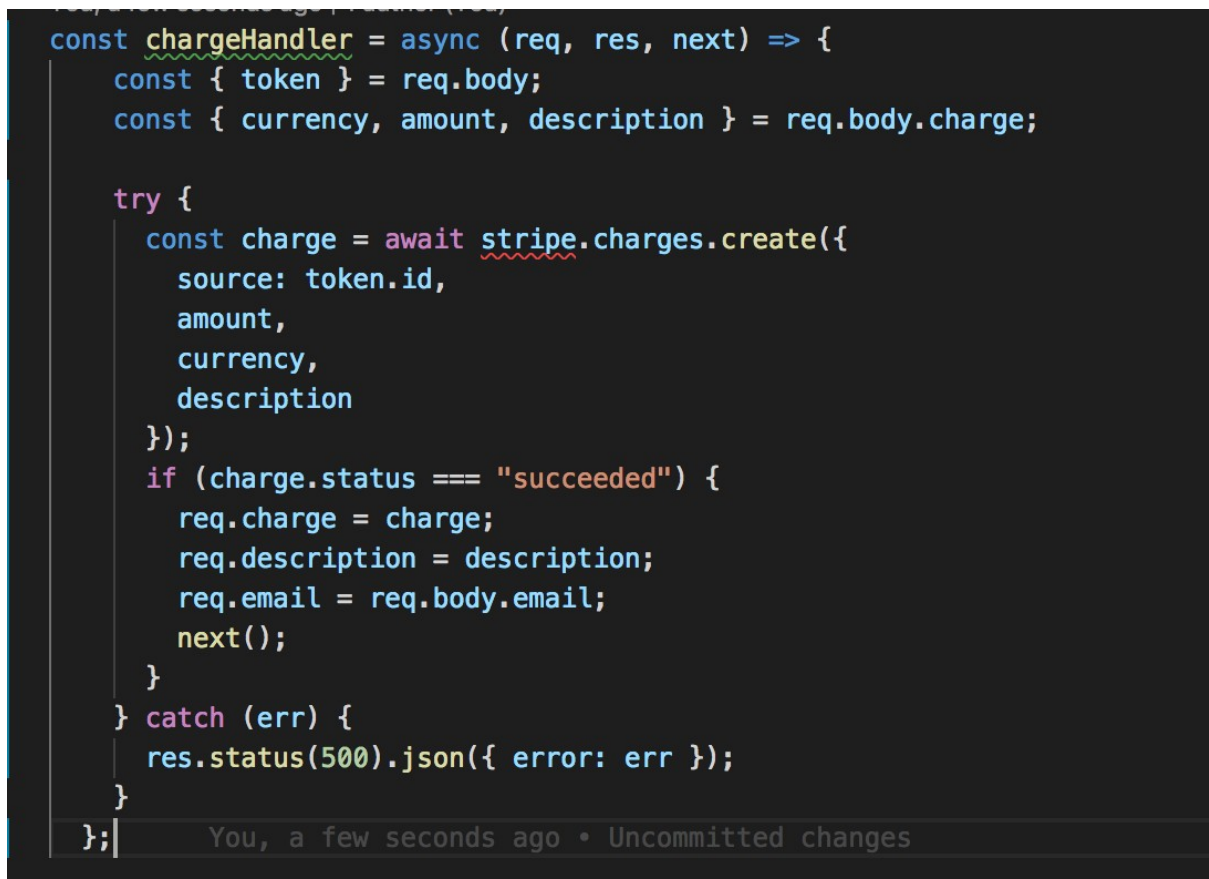
```

Rysunek 26 Schemat GraphQL dla modeli użytkownika, zamówienia oraz Adresu wysyłki. Opracowanie własne.

4. „User” czyli użytkownik, posiada identyfikator, nazwę użytkownika, email, informację, czy jest właścicielem oraz asocjacje do zamówień.
5. „Order” czyli zamówienie – posiada identyfikator, asocjacje do produktu, który jest w zamówieniu, również asocjacje do użytkownika, który ma dane zamówienia, datę stworzenia, oraz asocjacje do adresu wysyłki
6. „Shipping Address” – składa się z nazwy miasta, państwa, adresu wysyłki, województwa oraz kodu pocztowego.

Użyto również funkcji oferowanych przez AWS Lambda do przetworzenia obsługi płatności poprzez operatora płatności Stripe. Dzięki takiemu podejściu zarówno programiści, jak i użytkownicy nie muszą się martwić o tak istotną kwestię jak obsługa płatności. Nawet przy dużej liczbie dokonywanych jednocześnie zapłat za produkty, nie będzie problemu z

wykonaniem tego zadania. AWS Lambda automatycznie skaluje kod programowania, przedstawiony na rysunku nr 27. Funkcja ta, ma za zadanie asynchronicznie wysłać żądania do klienta płatności, wraz z danymi odnośnie zamówienia, takich jak cena za produkt, waluta, a także opis zamówienia. Jeżeli status zapłaty będzie pozytywny, klient dostaje informację zwrotną. W przeciwnym razie, przy niepowodzeniu, również jest o tym informowany.



```
const chargeHandler = async (req, res, next) => {
  const { token } = req.body;
  const { currency, amount, description } = req.body.charge;

  try {
    const charge = await stripe.charges.create({
      source: token.id,
      amount,
      currency,
      description
    });
    if (charge.status === "succeeded") {
      req.charge = charge;
      req.description = description;
      req.email = req.body.email;
      next();
    }
  } catch (err) {
    res.status(500).json({ error: err });
  }
};
```

Rysunek 27 Funkcja Lambda wykorzystana do obsługi płatności. Opracowanie własne.

Usługę Amazon Simple Email Service (Amazon SES) użyto w projekcie w celu wysyłania maili zarówno potwierdzeń zamówień, jak i informacji o nowych produktach dla użytkowników w formie powiadomień. Usługa jest bardzo prosta w użyciu, dodatkowo pierwsze 62 tysiące wysłanych emaili w miesiącu, są za darmo. Przykład implementacji wysyłania maila w przypadku złożenia nowego zamówienia, przedstawiono na rysunku nr 28.

```

Unsaved changes (cannot determine recent change of authors)
const emailHandler = (req, res) => {
  const {
    charge,
    description,
    email: { shipped, customerEmail, ownerEmail }
  } = req;

  ses.sendEmail(
    {
      Source: config.adminEmail,
      ReturnPath: config.adminEmail,
      Destination: {
        ToAddresses: [config.adminEmail]
      },
      Message: {
        Subject: [
          Data: "Szczegóły zamówienia"
        ],
        Body: {
          Html: {
            Charset: "UTF-8",
            Data: `
<h3>Tytuł wiadomosci!</h3>

<p style="font-style: italic; color: grey;">
</p>
`
          }
        }
      },
    },
    (err, data) => {
      if (err) {
        return res.status(500).json({ error: err });
      }
      res.json({
        message: "Zamówienie złożone!",
        charge,
        data
      });
    }
  );
};

```

Rysunek 28 Implementacja wysyłania maili przy pomocy usługi Amazon Simple Email Service. Opracowanie własne

4.2. Dostęp do aplikacji

Dostęp do strony dla użytkownika jest możliwy tylko i wyłącznie poprzez uwierzytelnienie użytkownika. Użytkownik musi posiadać własne konto w portalu, bez tego nie jest możliwe korzystanie z niego. Uwierzytelnienie oraz autentyfikacja są oparte o usługę Amazon Cognito, która zapewnia procesy logowania oraz rejestracji wewnątrz aplikacji. Dzięki temu narzędziu, można zdefiniować role i przyporządkować użytkowników do różnych ról, tak aby użytkownicy aplikacji mieli dostęp tylko do zasobów autoryzowanych dla każdego użytkownika.

Strona rejestracji została przedstawiona na rysunku 29

The image shows a registration form titled "Stwórz nowe konto w Tortowo". It contains the following fields and elements:

- Login(wymagany)**: A text input field with the placeholder "login".
- Hasło(wymagane)**: A text input field with the placeholder "hasło".
- Email(wymagany)**: A text input field with the placeholder "Email".
- Numer telefonu (wymagany)**: A dropdown menu showing "+48" and a text input field with the placeholder "Numer telefonu".
- At the bottom, there is a link "Masz już konto? Zaloguj się" and a blue button labeled "ZAŁÓŻ KONTO".

Rysunek 29 Strona rejestracji użytkownika. Opracowanie własne

Składa się ona z formularza, w którym należy uzupełnić wszystkie pola, które są wymagane, aby zostać zarejestrowany wewnątrz aplikacji. Są to kolejno:

1. login
2. hasło
3. email
4. telefon komórkowy wraz z prefiksem

Następnie na podany email przedtem email zostanie wysłany kod weryfikacyjny który trzeba wpisać w następnym etapie.

Strona logowania wygląda, tak jak jest to przedstawione na rysunku 30. Do aplikacji można się zalogować poprzez login lub email, oraz podanie hasła. Jeżeli użytkownik nie pamięta hasła, może on zrestartować hasło. Dzięki czemu otrzyma na email, na którym jest zarejestrowane konto link, który umożliwi stworzenie nowego hasła. Jeżeli dana osoba nie ma konta w aplikacji, jest przycisk do przekierowania do strony tworzenia nowego konta w systemie. Jeżeli proces uwierzytelnienia przejdzie poprawnie, użytkownik jest automatycznie przekierowany do strony głównej aplikacji.

Rysunek 30 Ekran logowania do aplikacji. Opracowanie własne

4.3. Opis funkcjonalności aplikacji

Po zalogowaniu użytkownika na stronę, ujrzy on stronę główną tak, jak jest to przedstawione na rysunku nr 31.

NOWE TORTY	
Tort owocowy Tort z owoców sezonowych...	88 PLN
Tort malinowy Pyszny tort z malin...	60 PLN
Tort waniliowy Tort waniliowy...	120 PLN
Tort czekoladowy Tort czekoladowy...	75 PLN
Przepis na tort wege Do zrobienia w jeden wieczór...	15 PLN
Tort mięsny Nowość w naszym sklepie...	89.99 PLN

CUKIERNIE			
	Nazwa Cukierni	Ilość Produktów	
	Czysta słodycz	13	Sprawdź%
	Cukiernia u Zuzi	10	Sprawdź%
	Słodko i pysznie	7	Sprawdź%
	Delicious	3	Sprawdź%
	U Anieli	6	Sprawdź%
	Domowo	20	Sprawdź%

Rysunek 31 Wygląd strony głównej po zalogowaniu się do aplikacji. Opracowanie własne

Strona główna składa się z następujących elementów:

1. Wyszukiwarki, która pozwala na wyszukiwanie wewnątrz aplikacji takich danych jak produkty, cukiernie, ceny, opisy produktów, i wiele więcej. Wyszukiwarka jest oparta o rozwiązanie AppSync. Pozwala to na bezpieczne pobieranie danych z zasobów

tworzonej aplikacji i wyszukiwanie, analiza i wizualizacja tych danych w czasie rzeczywistym. Usługa jest w pełni skalowana, więc w przypadku ogromnej liczby rekordów do przetworzenia, usługa ta, nie będzie miała żadnych problemów.

2. Powiadomień – użytkownik dostaje powiadomienia po każdym wydarzeniu związanym bezpośrednio z jego osobą. Są to między innymi powiadomienia o nowym zamówieniu, pojawieniu się nowego produktu w jego cukierni itd. Do stworzenia powiadomień wewnątrz aplikacji użyto usługi Amazon Simple Notification Service. Powiadomienia dotyczą:

- Profilu użytkownika – profil użytkownika, gdzie użytkownik może zarówno zobaczyć swoje dane takie jak imię, nazwisko, adres email, ilość produktów, jak i zmienić hasło, lub też usunąć konto. W celu lepszego personalizowania użytkowników wewnątrz aplikacji, jest dodana funkcja dodawania własnego zdjęcia jako awatar.
- Nagłówków – ich celem jest zachęcenie użytkowników do platformy. Wyświetlane są dane o ilości zamówień, produktów oraz cukierni.
- Listy cukierni. Wyświetlone są cukiernie jako lista, składająca się z awataru właściciela, jego aktywności w portalu (czy jest obecnie online), nazwy cukierni, ilości produktów w jego cukierni, jak i linku do zobaczenia dokładniejszego opisu cukierni.
- Listy nowych tortów. Wyświetlone są nowe produkty wraz ze zdjęciem, tytułem, opisem jak i z ceną za pojedynczy produkt.

Zarówno lista cukierni jak i lista nowych tortów jest pobierana z klienta API, stworzonego przy pomocy usługi Amazon Web Services AppSync. Dzięki takiemu zastosowaniu, nawet przy milionach ilości produktów oraz cukierni, dane będą dostarczane natychmiastowo w czasie rzeczywistym.

W celu dodania nowej cukierni, należy wypełnić formularz o potrzebne dane. Są to kolejno:

- Nazwa cukierni,
- Opis cukierni,
- imię i nazwisko właściciela, w celu potwierdzeni autentyczności cukierni,
- email właściciela.

Wygląd formularza przedstawiono na rysunku 32.

DODAJ NOWĄ CUKIERNIE

Nazwa cukierni

nazwa cukierni

Opis cukierni

opis Cukierni

Imię i nazwisko
właściciela

Imię i nazwisko właściciela

email

email

☐ Akceptuję regulamin

Dodaj cukiernie

Rysunek 32 Wygląd formularza tworzenia nowej cukierni. Opracowanie własne.

Dodatkową informacją, jaka jest zapisywana w systemie to automatycznie przypisywanie cukierni, którą tworzy dany użytkownik do jego konta.

Dodawanie nowego produktu, składa się z formularza, który należy uzupełnić o wymagane pola, tak jak to przedstawione na rysunku 33.

DODAJ NOWY PRODUKT

Nazwa produktu

Tort Malinowy

Opis produktu

Przepyszny tort malinowy

Cena

150

Kategoria

Załącz zdjęcie

Choose file

60296009_2448225488521152_8613274414949072896_n.png

Pliki tylko z rozszerzeniem jpg/png

Wybierz typ produktu (wymagane)

☒ Gotowy produkt

☐ Przepis na produkt

☒ Akceptuję regulamin

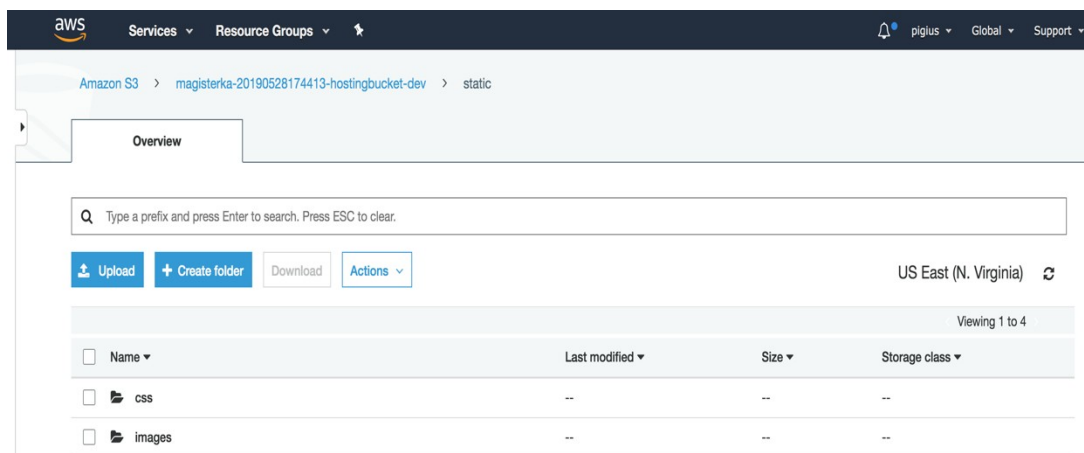
Dodaj produkt

Rysunek 33 Widok formularza służącego do tworzenia nowego produktu w sklepie. Opracowanie własne.

Formularz składa się z następujących pól:

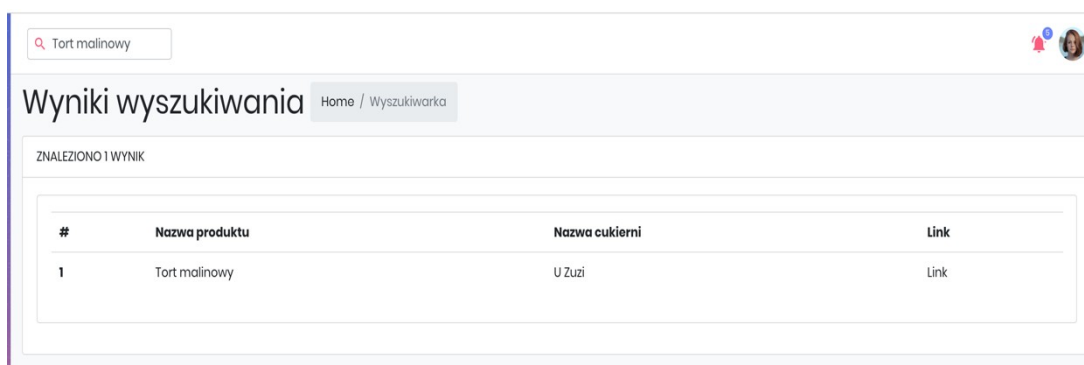
1. Nazwa produktu: wymagana nazwa produktu w celu identyfikacji go po tytule.
2. Opis produktu: Opis produktu służący do zachęcenia potencjalnego klienta do zakupu.
3. Cena: Cena za produkt jako liczba.
4. Kategoria: Dany produkt można przypisać do kategorii, w celu lepszego pozycjonowania go i lepszej możliwości wyszukania go wewnątrz aplikacji.
5. Typ produktu: Możliwość wyboru jednej opcji; Produkt fizyczny (np. tort), lub też produkt wirtualny w postaci przepisu do stworzenia produktu.
6. Akceptacja regulaminu
7. Dodanie produktu do bazy danych produktów
8. Zdjęcia: Zaimplementowano możliwość dodawania zdjęć produktu, w celu lepszej wizualizacji produktu. Zdjęcia są bezpośrednio przesyłane do usługi Amazon Simple Storage Service (Amazon S3). Dzięki zastosowaniu takiego rozwiązania,

użytkownicy nie muszą się martwić o rozmiar zdjęć, ich ilość a także o ich zaginięcie. Amazon S3 został zaprojektowany dla 99,999999999% długowieczności danych, ponieważ automatycznie tworzy i przechowuje kopie wszystkich obiektów S3 w wielu systemach. Miejsce, gdzie znajdują się zdjęcia zostały przekazane na rys. 34.



Rysunek 34 Miejsce, gdzie znajdują się zdjęcia produktów w usłudze Amazon S3. Opracowanie własne

W celu wyszukiwania informacji wewnątrz aplikacji zaimplementowano wyszukiwarkę. Wyszukiwarka wyszukuje informacji po słowach kluczowych i wyświetla wyniki w formie tabeli. Na rysunku nr 35, użytkownik wyszukał słowa kluczowego „tort malinowy” i został znaleziony jeden wynik, „tort malinowy”, który można nabyć w cukierni „U Zuzi”, ponadto dodano link do bezpośredniego przejścia do produktu.



Rysunek 35 Widok wyników z wyszukiwarki. Opracowanie własne.

Składanie zamówień wygląda następująco:

Użytkownik wybiera żądany produkt i następuje przekierowanie do osobnej strony z tworzeniem zamówienia. Po prawej stronie widoczne są dane wybranego produktu, a także

informacje czy jest aktualnie dostępny, jak i informację z jakiej cukierni pochodzi. W celu złożenia zamówienia użytkownik musi podać dane do wysyłki, czyli miasto, adres zamieszkania, kod pocztowy oraz województwo. Takich danych jak imię i nazwisko, email nie musi podawać, zostaną one automatycznie pobrane z bazy danych (profil użytkownika). Część wizualna została przedstawiona na rysunku numer 36. Po naciśnięciu przycisku potwierdź i zapłać, pojawi się okno modalne w celu dopełnienia płatności poprzez operatora płatności Stripe.

Twoje zamówienie Home / Tworzenie Zamówienia

DANE DO DOSTAWY

Miasto

Adres zamieszkania

Kod pocztowy

Województwo

☐ Akceptuję regulamin

Potwierdź i zapłać

WYBRANY PRODUKT

Nazwa produktu

Tort czekoladowy ✓

Produkt dostępny.

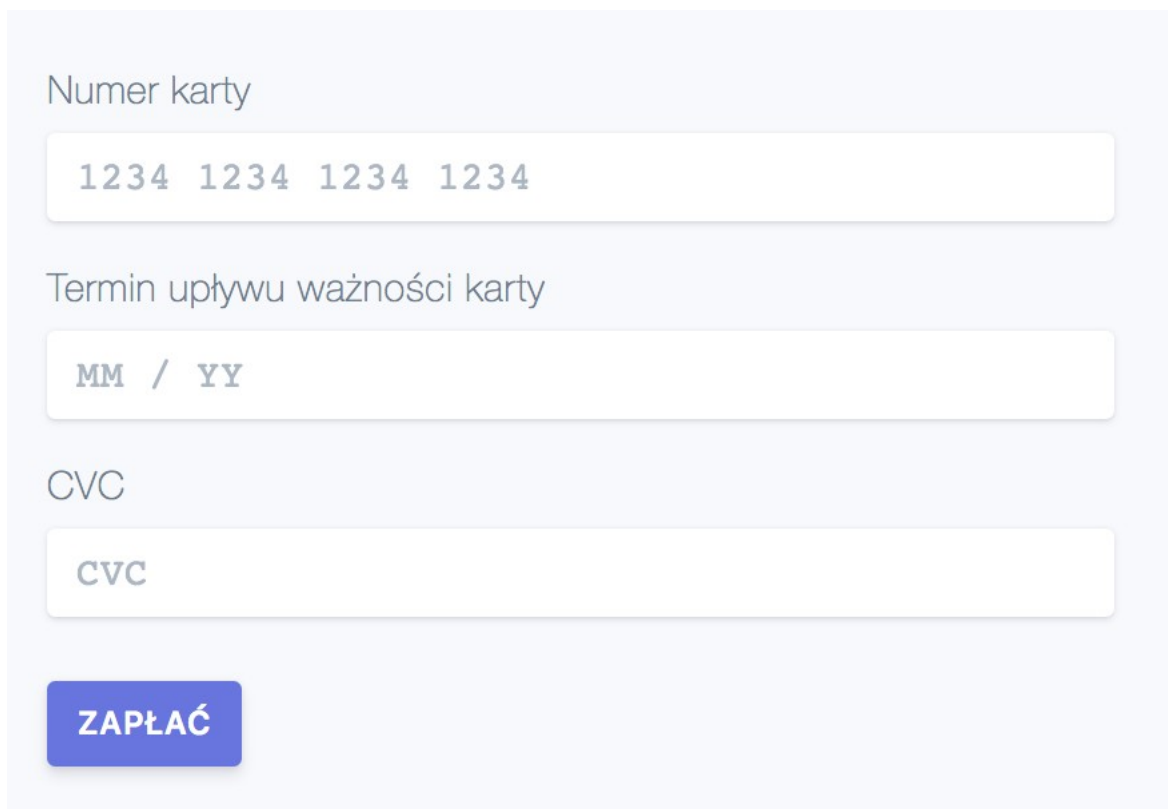
Cukiernia

U Zuzi

Wybrana cukiernia

Rysunek 36 Widok składania zamówienia. Opracowanie własne

Po uzupełnieniu danych karty kredytowej i wciśnięciu przycisku zapłać, środki z konta zostaną pobrane, a użytkownik dostanie maila odnośnie złożenia zamówienia oraz pobrania środków pieniężnych z konta. Wygląd okienka modalnego zostało przedstawione na rysunku numer 37.



Numer karty

1234 1234 1234 1234

Termin upływu ważności karty

MM / YY

CVC

CVC

ZAPŁAĆ

Rysunek 37 Widok modala, który odpowiada za płatność kartą, przez operatora płatności Stripe. Opracowanie własne

Stworzony projekt aplikacji jest w pełni oparty o architekturę Serverless. Dzięki użyciu gotowych, w pełni skalowalnych usług chmury obliczeniowej Amazon Web Services, zbudowano długotrwały projekt, odporny na przestoje, duży ruch użytkowników, a także w dłuższej perspektywie – umożliwiający zmniejszenie kosztów utrzymania aplikacji.

Zakończenie

W podejściu typu Serverless do przetwarzania danych programista nie musi brać pod uwagę operacji serwerowych, ponieważ są one przechowywane na zewnątrz u dostawcy chmury obliczeniowej. Jedyne co jest wymagane, to dostarczenie kodu źródłowego. Pozostałe zadania związane z dostarczeniem usługi realizowane są po stronie dostawcy chmury obliczeniowej. Dzięki podejściu Serverless można zbudować w krótkim czasie działające oraz w pełni skalowalne, niezależnie od ruchu aplikacje internetowe. Z biznesowej perspektywy można zauważyć fakt, iż koszt poniesiony przez aplikację typu Serverless opiera się na liczbie wykonanych funkcji, mierzonej w milisekundach zamiast w godzinach. Pozwala to również na zredukowanie kosztów operacyjnych, związanych np. z infrastrukturą

serwerową. Z perspektywy programistów jest to zmniejszona odpowiedzialność programistów za zaplecze infrastrukturalne, łatwiejsze zarządzanie operacyjne, szybsze wdrożenia, automatycznie skalowanie rozwiązań, jak i wspieranie innowacji. Można zauważyć również wady podejścia Serverless. Z perspektywy biznesu są to kolejno zmniejszona kontrola całkowita nad chmurą obliczeniową, Blokada dostawcy chmury obliczeniowej wymaga większego zaufania do zewnętrznego dostawcy. Również ważną wadą jest ryzyko związane z usuwaniem skutków awarii, a także fakt, iż koszt jest nieprzewidywalny, ponieważ liczba wykonań danej usługi nie jest wstępnie zdefiniowana. Natomiast wady z pozycji programisty to niedojrzała technologia która skutkuje rozproszeniem elementów, niejasno określonymi najlepszymi praktykami, złożoność architektoniczna. Istotnymi wadami są także znaczące ograniczenia w testowaniu na środowisku lokalnym, a także brak narzędzi operacyjnych. Ponadto przy podejściu Serverless czas realizacji pojedynczej funkcji, usługi jest zawsze ograniczony.

Dalsze prace badawcze mogą dotyczyć, na przykład badań ilościowych dotyczących efektywności funkcjonowania aplikacji na grupie programistów wdrażających architekturę Serverless.

Literatura

1. Alexandru Jecan, Java 9 Modularity Revealed Project Jigsaw and Scalable Java Applications, Apress, 2017 Munich, s. 13, 22
2. Amazon Web Services, "AWS Well - Architected Framework", Amazon Web Service Inc., 2018, s. 25
3. Amazon Web Services, "Architecting for the Cloud. AWS Best Practices", Amazon Web Services Inc, 2018, s. 19.
4. Amazon Web Services, "Serverless Architectures with AWS Lambda Overview and Best Practices", Amazon Web Services, Inc., 2017, s. 1
5. CHRISTIAN HORSDAL GAMMELGAARD, "Microservices in .NET Core", Manning Publications Co, 2017, s. 27 - 28, 30, 32 - 33.
6. Chander Dhall, "Scalability Patterns Best Practices for Designing High Volume Websites", Apress, 2018 Texas, s. 33, 35

7. Chris Northwood, "The Full Stack Developer Your Essential Guide to the Everyday Skills Expected of a Modern Full Stack Web Developer", Apress, 2018 Manchester, s. 82
8. Joe Baron, Hisham Baz, Tim Bixler, Biff Gaut, Kevin E.Kelly, Sean Senior, John Stamper, "Certified Solutions Architect Official", John Wiley & Sons, Inc., 2017 Indianapolis, s. 45
9. John Chapin, Michael Roberts, "What Is Serverless ?", O'Reilly Media, Inc., 2017, s. 17, 20, 22
10. John Paul Mueller, "AWS For Admins", John Wiley & Sons, Inc, 2017 Hoboken, s. 146 - 147
11. Kasun Indrasiri Prabath Siriwardena, "Microservices for the Enterprise Designing, Developing, and Deploying", Apress, 2018 San Jose, s. 11 - 14
12. Kasun Indrasiri, Prabath Siriwardena, "Microservices for the Enterprise", Apress, 2018 San Jose, s. 8 - 9, 29, 127
13. Lee Atchison, "Architecting for Scale High Availability for Your Growing Applications", O'Reilly Media, Inc, 2016, s. 5
14. Martin Kleppmann, "Designing Data - Intensive Applications", O'Reilly Media, Inc, 2017
15. MICHAEL WITTIG ANDREAS WITTIG, "Amazon Web Services in Action, Second Edition", Manning Publications Co., 2018, s. 7 - 8
16. MORGAN BRUCE, PAULO A. PEREIRA, "Microservices in Action", Manning Publications Co., 2019 Nowy Jork, s. 3 - 6.
17. Mike Ryan & Federico Lucifredi, "AWS System Administration Best Practices for Sysadmins in the Amazon Cloud", O'Reilly Media, Inc., 2018, s. 13
18. Moises Macero, "Learn Microservices with Spring Boot A Practical Approach to RESTful Services using RabbitMQ, Eureka, Ribbon, Zuul and Cucumber", 2017 Nowy Jork, s. 13 - 15
19. PETER SBARSKI, "Serverless Architectures on AWS", Manning Publications Co, 2017, s. 12, 14 - 15, 20 - 21, 38
20. Sarah Allen, Chris Aniszczyk, Chad Arimura, "CNCF WG - Serverless Whitepaper", The Cloud Native Computing Foundation, 2019, s. 34
21. Susan J.Fowler, "Production - Ready Microservices", O'Reilly Media, Inc, 2017 Sebastopol, s. 18 - 21.

22. Thomas Hunter II, "Advanced Microservices. A Hands - on Approach to Microservice Infrastructure and Tooling", Apress, 2017 California, s. 2 - 4
23. Thurupathan Vijayakumar, "Practical API Architecture and Development with Azure and AWS Design and Implementation of APIs for the Cloud", Apress, s. 133 - 134
24. Tom McLaughlin, "Serverless Devops", 2018, s. 6 - 7
25. Uchit Vyas, "Mastering AWS Development", 2015, Packt Publishing Ltd, s. 24
26. Yohan Wadia, Udit Gupta, Mastering AWS Lambda, Packt Publishing, 2017, s. 21
27. <http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod>
28. <https://aws-amplify.github.io/docs/>
29. <https://aws.amazon.com/appsync/>
30. <https://aws.amazon.com/cognito/>
31. <https://aws.amazon.com/compliance/>
32. <https://aws.amazon.com/lambda/features/>
33. <https://aws.amazon.com/lambda/resources/customer-testimonials/>
34. <https://aws.amazon.com/lambda/resources/customer-testimonials/>
35. <https://aws.amazon.com/lambda/resources/customer-testimonials/>
36. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html
37. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html>
38. <https://martinfowler.com/articles/serverless.html>
39. <https://martinfowler.com/articles/serverless.html>
40. <https://martinfowler.com/articles/serverless.html#Message-driven-applications>
41. <https://martinfowler.com/articles/serverless.html#State>
42. <https://martinfowler.com/articles/serverless.html#unpacking-faas>
43. <https://microservices.io/patterns/monolithic.html>
44. <https://serverless-stack.com/chapters/what-is-serverless.html>
45. <https://www.webdesignerdepot.com/2018/05/monolith-vs-microservices-which-is-the-best-option-for-you/>
46. <https://d0.awsstatic.com/whitepapers/optimizing-enterprise-economics-serverless-architectures.pdf>
47. <https://www.thorntech.com/2017/12/microservices-vs-monoliths-whats-right-architecture-software/>

Spis ilustracji:

Rysunek 1. Schemat aplikacji monolitycznej. Opracowanie własne.....	6
Rysunek 2 Aplikacja do sprzedaży detalicznej online opracowana w oparciu o architekturę monolityczną. Opracowanie własne.....	9
Rysunek 3 Rysunek przedstawia działanie mikroservisów, w trakcie wdrażania mikroservisów koszyka sklepowego.....	13
Rysunek 4 Uruchomienie więcej niż jednego mikroservisów w ramach procesu. Opracowanie własne.....	15
Rysunek 5. Jeden mikroservis nie może uzyskać dostępu do danych przechowywanych przez inną usługę. Opracowanie własne.....	16
Rysunek 6 Aplikacja sklepu zoologicznego online opracowana w oparciu o architekturę monolityczną. Opracowanie własne.....	22
Rysunek 7 Aplikacja sklepu zoologicznego online opracowana w oparciu o architekturę Function as a Service. Opracowanie własne.....	23
Rysunek 8 Schemat systemu reklamy online w podejściu tradycyjnym. Opracowanie własne.	25
Rysunek 9 Schemat systemu reklamy online w podejściu FaaS. Opracowanie własne.....	25
Rysunek 10 Logo Amazon Web Services. Źródło: https://en.wikipedia.org/wiki/Amazon_Web_Services	39
Rysunek 11 Mapa centrów danych chmury obliczeniowej Amazon Web Services. Źródło: https://aws.amazon.com/about-aws/global-infrastructure/	41
Rysunek 12 Widok interfejsu graficznego usługi IAM w Amazon Web Services. Opracowanie własne.....	43
Rysunek 13 Wygląd usługi S3 w AWS. Opracowanie własne.....	46
Rysunek 14 Wygląd usługi Cognito w AWS. Opracowanie własne.....	47
Rysunek 15 Wygląd usługi Lambda w AWS. Opracowanie własne.....	49
Rysunek 16 Wygląd usługi DynamoDB w AWS. Opracowanie własne.....	50
Rysunek 17 Schemat działania usługi AppSync w AWS. Opracowanie własne.....	51
Rysunek 18 Wygląd usługi AppSync w AWS. Opracowanie własne.....	52
Rysunek 19 Logo frameworku ReactJS, który jest wykorzystywany do zbudowania interfejsu użytkownika. Logo pobrane z http://www.danielledeveloper.com/reactjs-lets-talk-about-it/ . 53	53
Rysunek 20 Przykład implementacji napisany w ReactJS do wyświetlania danych o cukierni, wyświetlanych w tabeli. Opracowanie własne.....	54
Rysunek 21 Logo frameworku Amplify Framework wykorzystywanego w projekcie. Źródło: https://aws-amplify.github.io/	55
Rysunek 22 Wygląd Interfejsu linii poleceń Amplify. Opracowanie własne.....	55
Rysunek 23 Wygląd Amplify framework na przykładzie projektu Tortowo. Opracowanie własne.....	56
Rysunek 24 Konfiguracja Amazon Cognito, przy pomocy Amplify Framework. Opracowanie własne.....	57
Rysunek 25 Schemat GraphQL dla usługi AWS Appsync w aplikacji Tortowo. Opracowanie własne.....	58
Rysunek 26 Schemat GraphQL dla modeli użytkownika, zamówienia oraz Adresu wysyłki. Opracowanie własne.....	59
Rysunek 27 Funkcja Lambda wykorzystana do obsługi płatności. Opracowanie własne.....	60
Rysunek 28 Implementacja wysyłania maili przy pomocy usługi Amazon Simple Email Service. Opracowanie własne.....	61
Rysunek 29 Strona rejestracji użytkownika. Opracowanie własne.....	62
Rysunek 30 Ekran logowania do aplikacji. Opracowanie własne.....	63

Rysunek 31 Wygląd strony głównej po zalogowaniu się do aplikacji. Opracowanie własne.	63
Rysunek 32 Wygląd formularza tworzenia nowej cukierni. Opracowanie własne.....	65
Rysunek 33 Widok formularza służącego do tworzenia nowego produktu w sklepie. Opracowanie własne.....	66
Rysunek 34 Miejsce, gdzie znajdują się zdjęcia produktów w usłudze Amazon S3. Opracowanie własne.....	67
Rysunek 35 Widok wyników z wyszukiwarki. Opracowanie własne.....	67
Rysunek 36 Widok składania zamówienia. Opracowanie własne.....	68
Rysunek 37 Widok modala, który odpowiada za płatność kartą, przez operatora płatności Stripe. Opracowanie własne.....	69

Spis tabel:

Tabela 1 Lista regionów centrów danych chmury obliczeniowej Amazon Web Services. Tabela składa się z identyfikatora regionu, który jest niepowtarzalnym i zgodnym identyfikatorem dla strefy dostępności, oraz nazwa regionu.....	40
---	----