Uniwersytet Mikołaja Kopernika
Wydział Matematyki i Informatyki

Klaudia Augustyńska
nr albumu: 265408
informatyka

Praca magisterska

Wykorzystanie Cloud Computing w aplikacjach mobilnych

Opiekun pracy dyplomowej
dr Błażej Zyglarski

Toruń 2018

Spis treści

Wstęp **5**

Problem zastosowania chmury obliczeniowej w aplikacjach mobilnych 6

Cel pracy 7

Opis rozdziałów 7

1. Wprowadzenie **9**

1.1. Charakterystyka Cloud Computingu 9

1.2. Ewolucja Cloud Computingu 12

1.3. Historia najnowsza. Współczesne wyzwania 15

1.3.1. Rozwój chmur publicznych 15

1.3.2. Konteneryzacja i DevOps 16

1.3.3. Mikrousługi i serverless 21

1.3.4. Bazy NoSQL i rozwiązania dla Big Data 25

1.3.5. CDN i Fog Computing 30

1.4. Wymagania stawiane współczesnym chmurom 30

1.5. Dokąd zmierza rozwój CC. 30

2. Porównanie różnych podejść **31**

2.1. Chmury publiczne 31

2.2. Chmury prywatne 31

3. Analiza wyboru platformy dla aplikacji mobilnej **33**

3.1. Wytumaczenie czemu to jest typowy projekt 33

3.2. Wymagania co trzeba wziąć pod uwagę 33

3.3. Werdykt, kandydatka nr 1, 2 ,3 33

4. Opis wdrożenia aplikacji zgodnie z kandydatką nr 1 **35**

Spis rysunków **37**

Spis tabel **39**

Wstęp

Cloud Computing to model stanowiący podstawę dla jednych z najprężniej rozwijających się technologii informatycznych obecnych czasów. Jego wykorzystanie odnosi się do praktycznie wszystkich dziedzin informatyki, poczynając od administrowania infrastrukturą komputerową, przez tworzenie systemów informatycznych, po wsparcie badań naukowych oraz pracy przeciętnych użytkowników komputerów.

Technologie oparte o Cloud Computing mają swoje podwaliny w starszych technologiach, sięgających lat 70. ubiegłego wieku. Chodzi zatem o produkt ewolucji technologicznej, nie zaś odrębną nową technologię. Dzisiejszy dynamiczny rozwój Cloud Computingu zawdzięczamy m.in. szybkiemu łączu internetowemu, upowszechnieniu komputerów PC oraz mocy obliczeniowej umożliwiającej wirtualizację na poziomie wykorzystania sztucznej inteligencji do inteligentnego zarządzania infrastrukturą komputerową (ang. *autonomic computing*). Złożenie tych czynników umożliwiło urzeczywistnienie idei po raz pierwszy wymienionej w 1961 r. przez Johna McCarthy'ego, który pisał, że zasoby komputerowe staną się użytecznością publiczną – podobnie jak prąd, który pobieramy z sieci elektrycznej, nie zaś z własnego generatora prądu.[1, 5] Stąd też analizując temat Cloud Computingu niejednokrotnie można spotkać się ze zwrotem „X jako usługa"(ang. X *as a service*), ponieważ praktycznie wszystko, co może być związane z wykorzystaniem komputerów, można dostarczać jako usługę. [1]

Za chmurami obliczeniowymi stoją olbrzymie centra danych, łączące w sieć nawet tysiące komputerów. Wirtualizacja wykorzystywana w Cloud Computingu pozwala na automatyczną konfigurację nowych serwerów dołączanych do sieci oraz odpowiednie zachowanie sieci w przypadku gdy jakaś jej część przestanie działać. Dzięki takim narzędziom można dowolnie rozbudować sieć, a więc uzyskać dowolną przestrzeń dyskową i moc obliczeniową. Dlatego model chmury obliczeniowej jest nieodzownie powiązany z technikami wirtualizacji oraz technikami pracy na systemach rozproszonych.

Duża i zarazem bardzo znacząca część technologii związanych z Cloud

Computingiem dotyczy programowania. Chodzi nie tylko o usługi usprawniające proces wytwórczy oprogramowania, ale również o technologie i wzorce wspierające skalowalność oraz obsługę dużej ilości danych. Chmura pozwala zautomatyzować wiele zadań, z drugiej strony jej ogromne możliwości stanowią duże wyzwanie, gdyż mogą oznaczać całkowitą zmianę podejścia do wytwarzania oprogramowania, wyboru technologii oraz wzorców architektonicznych. Wiele firm powoli podejmuje to wyzwanie przez np. stopniową rezygnację z serwerów firmowych na rzecz serwerów wirtualnych dostępnych w chmurze, wdrażanie strategii CI/CD czy tworzenie nowych projektów w architekturze mikrousług.

Biorąc pod uwagę w jak szybkim czasie pojawiło się wiele istotnych narzędzi szybko znajdujących zastosowanie w biznesie, istnieje duże zapotrzebowanie na opracowanie tego tematu całościowo.

Problem zastosowania chmury obliczeniowej w aplikacjach mobilnych

Chmura obliczeniowa stanowi ważną część tzw. *networked society*, czyli obrazu do którego współcześnie dąży technologia oraz sposób korzystania z niej przez społeczeństwo. Jest to paradygmat w którym korzystanie z elektronicznych asystentów czy IoT (ang. *Internet of Things*) stanowi niezbędny element rzeczywistości.[5] W tym nowym obrazie świata chmura stanowi spoiwo, natomiast komunikacja z człowiekiem w dużej mierze odbywa się przy pomocy aplikacji mobilnych. Można spodziewać się rosnącego zapotrzebowania na aplikacje mobilne, których głównym zadaniem jest skuteczna komunikacja z chmurą.

Podczas wyboru technologii, na etapie analizy dostępnych rozwiązań okazuje się, że istnieje bardzo wiele podejść do tematu, z których duża część powstała w ciągu kilku ostatnich lat i od momentu publikacji zdobyła natychmiastową popularność. Dodatkowo wszystkie materiały, które pozwoliłyby to wszystko zrozumieć, występują w języku angielskim, najczęściej w kontekście wybranego specjalistycznego zastosowania.

Powyższy problem przyczynia się do braku zrozumienia technologii chmurowych jako całości. Powoduje to niską świadomość jak w pełni wykorzystać możliwości chmury w kontekście tworzenia typowego projektu informatycznego, jakim jest aplikacja mobilna.

Cel pracy

Celem pracy było rozstrzygnięcie jak na obecny stan wiedzy podejść do problemu realizacji typowego projektu aplikacji mobilnej w oparciu o Cloud Computing. W rozumieniu niniejszej pracy, typowa aplikacja pozwala na wprowadzanie i otrzymywanie danych od serwera za pomocą udostępnionego API. W rezultacie miała zostać wybrana konkretna technologia, która pod względem teoretycznym najlepiej odpowiadałaby stawianym wymaganiom.

Wybrana technologia miała być zastosowana w praktyce do stworzenia aplikacji na system Android, komunikującej się z API udostępnianym przez chmurę. Aplikacja miała umożliwiać zapisywanie wydatków w sposób uwzględniający fakt, że wiele wydatków ludzie dokonują nie tylko z myślą o sobie samych. System miał pozwolić założyć konto w serwisie, a następnie połączyć konta w grupy odpowiadające gospodarstwu domowemu. Kategorie wydatków miały pozwolić na uzgodnienie, kto ile płaci w danej kategorii (np. wydatki na higienę po 50%). Serwer miał być potrzebny do wspomagania rozliczeń pomiędzy poszczególnymi osobami, dynamicznie naliczając kto ma ile do oddania. Ponadto miał wspomagać kontrolę własnych wydatków przez uwzględnienie, że koszty życia to nie tylko własne wydatki, ale również wydatki poniesione przez inne osoby prowadzące to samo gospodarstwo domowe. Ponieważ tradycyjne programy do zarządzania wydatkami nie posiadają wyżej wymienionych funkcjonalności, jak również zazwyczaj mogą działać bez serwera, to jest przykład aplikacji, która może nagle stać się popularna i powinna być w stanie wówczas obsłużyć większe obciążenie serwera.

Opis rozdziałów

Rozdział 1. – Wprowadzenie TODO. NAPISZĘ O TYCH ROZDZIAŁACH W CZASIE PRZESZŁYM, JAk RZECZYWIŚCIE BĘDĄ W CZASIE PRZESZŁYM

Rozdział 1.

Wprowadzenie

Niniejszy rozdział wyjaśnia w jaki sposób ewoluowały technologie, by dać się później poznać jako Cloud Computing, a także czego należy oczekiwać od współczesnej chmury. Na końcu rozdziału opisano dokąd zmierza rozwój przedstawionych technologii.

1.1. Charakterystyka Cloud Computingu

Cloud Computing to model zgodnie z którym wszelkie zasoby informatyczne (oprogramowanie, przestrzeń dyskowa, dostęp do bazy danych itp.) dostarczane są w formie usługi. Istotną cechą jest wysoka skalowalność udostępnianych rozwiązań. Po stronie klienta ma to wyglądać tak, jak gdyby posiadał dostęp do nieskończonej mocy obliczeniowej i niekończącej się przestrzeni dyskowej, natomiast po stronie usługodawcy podłączenie nowych serwerów w celu podtrzymania tej iluzji nie powinno stanowić problemu. [2]

Aby była możliwa tak wysoka elastyczność, fizyczne serwery są oddzielone warstwą abstrakcji, na której są widoczne jako pula zasobów takich jak przestrzeń dyskowa czy moc procesora. Każdy program czy serwer wirtualny działający w chmurze osadzany jest na wirtualnych zasobach; nie ma możliwości zdecydowania z którego konkretnego fizycznego zasobu chce się korzystać. Moc chmury obliczeniowej buduje się przez łączenie tanich, łatwo wymienialnych komponentów sprzętowych w potężne zasoby wirtualne. [1]

Poza rozwiązaniem problemu zarządzania ogromną ilością fizycznych serwerów, wirtualizacja zasobów pozwala na lepsze wykorzystanie sprzętu. W tradycyjnym modelu komputery muszą być przygotowane na wypadek gdyby zainstalowane na nich programy powodowały większe zużycie zasobów. Dzieje się tak nawet w przypadku komputerów PC – kupuje się specjalnie większe dyski i lepszy procesor, aby przydały się w przyszłości. W

ten sposób wykorzystuje się niewielką część możliwości pojedynczego komputera, ponieważ przez większość czasu potrzeba mu znacznie mniej zasobów niż fizycznie posiada. W przypadku wirtualnych zasobów, do fizycznej jednostki można dynamicznie przypisać zużycie powodowane przez wielu użytkowników, wiele wirtualnych systemów operacyjnych, w zgodzie z ustalonym algorytmem. Algorytm może definiować, że np. wszystkie komputery mają zostać obciążone po równo (round robin), czy że pojedynczy węzeł ma być wykorzystany w 100% [3]. Technikę tę nazywa się równoważeniem obciążenia (ang. *load balancing*).

Technika zrównoważonego obciążenia przynosi kilka ważnych korzyści będących istotnymi cechami chmur obliczeniowych. Dzięki niej usługi działające w chmurze mogą być uruchomione na różnych węzłach, w tylu instancjach, ile wymagane jest do prawidłowego obsłużenia ruchu. W przypadku awarii którejś z instancji użytkownicy usługi niczego nie odczuwają, gdyż zostają przekierowani na instancję co do działania której nie wykazano błędów. Wszystko to ułatwia tworzenie wysoce skalowalnego oprogramowania. Nie bez znaczenia jest także pozytywny wpływ na ekosystem, ponieważ chmury obliczeniowe oznaczają optymalne zużycie istniejącego sprzętu komputerowego, a więc nie trzeba produkować go więcej niż potrzeba ani niepotrzebnie zużywać energii elektrycznej.

Najbardziej spektakularne efekty wykorzystywania modelu chmury widać w przypadku dostawców posiadających największe centra danych na świecie, takich jak Amazon, Microsoft, Google czy IBM. Na kilkudziesięciu m^2 gromadzą zasoby komputerowe, których równocześnie mogą używać miliony osób na całym świecie. Na jednym fizycznym komputerze zasoby mogą być wykorzystywane przez wiele osób niewiedzących o sobie nawzajem (po angielsku tę właściwość określa się jako *multi-tenancy*). Chmury o takiej architekturze mają potencjał ucieleśnić ideę zgodnie z którą zasoby komputerowe mogą być dostarczane jako usługa użyteczności publicznej.

Wyżej wymieniony sposób myślenia o chmurze jest tym co wyraźnie odróżnia chmury publiczne od chmur prywatnych. Nic nie stoi na przeszkodzie, aby samemu stworzyć prywatną sieć opartą o model chmury. Wówczas formalnie jest to chmura, lecz nie zapewnia niektórych ważnych korzyści, głównie przez konieczność samodzielnego administrowania serwerami, mniejszą ilość potencjalnych użytkowników oraz mniejszy potencjał puli zasobów. Przykładowo, jeśli firma nadal musi martwić się o zarządzanie fizycznymi jednostkami komputerowymi, to nie będzie miała możliwości wychwalać modelu chmury za brak konieczności zatrudniania specjalistów zajmujących się

infrastrukturą komputerową.

Korzyści i wady

Do głównych korzyści wynikających ze stosowania modelu chmury należą:

- **rozwiązanie problemu skalowania aplikacji**
  Wraz z rozwojem Web 2.0, dostępnością szybkiego łącza internetowego, zwiększaniem ilości urządzeń podłączonych do Internetu oraz ilości transmitowanych danych, istnieje rosnące zapotrzebowanie na aplikacje będące w stanie obsłużyć duży ruch. Dzięki chmurom uruchomienie wielu instancji aplikacji na różnych węzłach, dodatkowo w odmiennych wersjach, sprowadza się do opisu - ile, w jakiej wersji, w jakich proporcjach.[4] Podobnie ułatwione jest korzystanie z baz NoSQL.

- **rozwój Big Data**
  Chmury są w stanie zapewnić odpowiednie środowisko do przechowywania oraz przetwarzania dużych zbiorów danych. Udostępnienie tego środowiska jako usługi znacząco redukuje koszt przechowywania i analizy tych zbiorów, co za tym idzie próg wejścia w ten temat staje się dużo niższy. Za tym idzie wydajniejsze odkrywanie wiedzy z danych, co może mieć wpływ m.in. na rozwój medycyny.

- **ochrona środowiska**
  W modelu chmury sprzęt komputerowy zostaje wydajniej wykorzystany, w związku z tym jest mniejsze zapotrzebowanie na nowy sprzęt. Również dzięki pozostawieniu kosztownych operacji po stronie chmury, urządzenia klienckie łączące się z chmurą nie muszą być regularnie wymieniane na silniejsze.

- **lepsza koncentracja na wybranym zadaniu**
  Model chmury zwalnia użytkowników z dodatkowych czynności przy realizacji danego zadania. Przykładowo, jeśli komuś jest potrzebny serwer, to z chmurą publiczną nie musi martwić się zakupem sprzętu, zapewnieniem odpowiedniego pomieszczenia czy dostępem do Internetu. Może się zamiast tego skupić na właściwym skonfigurowaniu systemu na serwerze wirtualnym.

- **płatność tylko za rzeczywiste zużycie**
  Chmura umożliwia rozliczanie na podstawie czasu używania proce-

sora, ilości przesłanych megabajtów czy ilości danych przechowywanych w chmurze. W szczególności zmniejsza to początkowy koszt wdrażania czegokolwiek w chmurze.

- **gwarancja jakości usługi**
  Każdy dostawca usług chmurowych udostępnia klientom SLA (ang. *Service-level aggrements*), w którym zobowiązuje się do utrzymania określonego poziomu niezawodności usług (np. dostępność w 99,9% przypadków) oraz przewidywanego zachowania w przypadku niedotrzymania zapewnień (np. obniżki cen).

Ostatnią korzyścią, a jednocześnie kontrowersją, jest bezpieczeństwo danych w chmurach publicznych. Wysyłanie niejednokrotnie wrażliwych danych w nie do końca określone miejsce budzi obawy. Według autorów specjalistycznych wydawnictw[1, 2] są niesłuszne – pozostają zgodni co do opinii, że główni dostawcy usług chmurowych są w stanie zapewnić najwyższy poziom bezpieczeństwa, a fizyczne przechowywanie danych na terenie firmy daje tylko ułudę bezpieczeństwa.

Do potencjalnych wad rozwiązań bazujących na chmurach obliczeniowych należą:

- **problem z przenoszeniem aplikacji z jednej chmury na drugą**
  Znaczna część usług oferowanych przez największych dostawców usług chmurowych jest dedykowana dla tworzonej przez nich chmury. Oznacza to, że jeśli użytkownik chce mieć możliwość zmiany dostawcy, powinien uwzględnić to przy wyborze narzędzi pracy. Więcej na ten temat można przeczytać w rozdziale 3.

- **problemy prawne**
  Istnieje ryzyko rozbieżności pomiędzy prawem do którego dostosowywał się dostawca usług chmurowych, a prawem chroniącym dane w kraju, na terenie którego chce się korzystać z tych usług.

1.2. Ewolucja Cloud Computingu

Cloud Computing nie stanowi odrębnej, nowej technologii – jest to produkt ewolucji technologii rozwijanych od blisko 50 lat. Prześledzenie historii pozwala na zrozumienie, które koncepcje rzeczywiście wprowadzają nową jakość, a nie są jedynie od dawna istniejącą technologią, tyle że ubraną w ładnie brzmiące słowo.

Historia ma początek około lat 70., gdy używano wielkich superkomputerów zwanych ***mainframe***. Można było z nich korzystać za pomocą terminali, określanych mianem „głupich" (ang. *dump terminal*), ponieważ nie posiadały procesora i mogły być używane jedynie do operacji I/O. Jako że z serwera mógł korzystać jeden terminal naraz, serwer ustawiał je w kolejkę i musiały długo czekać na obsłużenie.

W latach 70. terminale zaczęły być wyposażone w mikroprocesory i być określane mianem **„inteligentnych terminali"** (ang. *inteligent terminal*). Mogły już partycypować przy uruchamianiu programów, co skróciło czas obsługi i dało początek modelowi klient-serwer.

Dalszy rozwój umożliwił rozwój mikroprocesorów, co dało początek **komputerom PC**. Komputery te mogły działać samodzielnie i były znacznie tańsze od mainframe-a.

Wynaleziono również LAN (ang. *local area network*) i WAN (ang. *wide area network*), co umożliwiło łączenie komputerów PC w sieci bez konieczności łączenia z mainframe-m. Powstały sieci **P2P** (ang. *Peer-To-Peer*).

We wczesnych latach 80. wynaleziono **systemy rozproszone**. Ponieważ były w stanie przetwarzać dane równolegle, zachwiało to poglądem iż większą moc obliczeniową należy uzyskiwać przez wynajdowanie silniejszych procesorów. Systemy rozproszone wymagały wzmożonej ilości operacji komunikacji, lecz występowało to w parze z rozwojem LAN osiągającym przepustowość 100 Mbps oraz WAN osiągającym 64 kbps.

Następnie powstała koncepcja **klastrów komputerowych**. Klaster polegał na połączeniu komputerów tego samego typu (homogenicznych) w sieć LAN i wyłonieniu wśród nich zarządcy, który będzie zajmował się przydzielaniem zadań pozostałym węzłom. Gdyby któryś z węzłów miał awarię, to inne mogły przejąć jego zadanie. Dało to początek idei **puli zasobów**. W klastrze niezawodność osiągało się przez nadmiarowość zasobów.

Główną wadą klastrów była konieczność powierzenia zarządzania klastrem jednemu komputerowi. Stanowił on miejsce od niezawodności którego zależała niezawodność całej sieci (ang. *single point of failure*). Rozwiązaniem okazały się **gridy**, w których każdy węzeł posiadał równy priorytet. Klient mógł podłączyć się do dowolnego komputera w gridzie, a ponadto komputery te mogły być różnego typu (heterogeniczne). Wkrótce gridy przeniknęły ze świata naukowego do świata biznesu i zaczynały być łączone przez WAN.

Dotychczas gdy uruchamiało się program na wybranym węźle gridu, to znajdował się na tym węźle dopóki proces nie został zakończony. Stanowiło to problem na drodze do skalowania w czasie rzeczywistym, gdzie na skutek

zmienionych potrzeb byłoby dobrze mieć możliwość od nowa zaalokować zasoby bez zakłócania działającej usługi. Problem ten rozwiązała **wirtualizacja sprzętowa**, za pomocą której zadania były alokowane do maszyn wirtualnych. Technika ta umożliwiła ***utility computing***, czyli model w którym zasoby komputerowe mogą być dostarczane jako usługa, co stanowi serce chmur obliczeniowych. Prekursorem została firma Salesforce.com, która udostępnia oprogramowanie w formie usługi internetowej od 1999 r.

W 2001 r. firma IBM zbudowała pierwszy **autonomiczny system** (ang. *antonomic computing*) – system, który potrafi sobą zarządzać bez interwencji człowieka. Dzięki wykorzystaniu sztucznej inteligencji do działania, eliminuje się ryzyko związane z błędem ludzkim oraz złożoność związaną z koniecznością doglądania przez człowieka złożonego systemu. IBM wytyczył następujące cechy modelu:

- automatyczna konfiguracja – konfiguracja tworzy się automatycznie na podstawie zapotrzebowania,
- automatyczna naprawa błędów – system sam wykrywa błędy i reaguje na nie,
- automatyczna optymalizacja – system sam dba o optymalne użycie zasobów,
- automatyczna ochrona – system wykrywa próby ataków i zapobiega im.

Wirtualizacje w tym systemie działają zgodnie z „pętlą adaptacyjną": obserwuj, decyduj, zareaguj.[5]

Duże znaczenie miało także wynalezienie metody **SOA** (ang. service oriented architecture), czyli techniki wytwarzania oprogramowania przez wydzielanie niezależnie działających komponentów, składających się razem na większy system informatyczny. Ważna była również koncepcja **Web 2.0**, nazwana tak po raz pierwszy w 2002 r. roku. Model Web 2.0 spowodował zmiany w sposobie myślenia o Internecie, zgodnie z którymi nie powinien jedynie służyć do dostarczania statycznej treści, lecz może być użyteczny również do udostępniania treści tworzonej przez użytkowników Internetu.

Wymienione wyżej technologie zestawione razem dały podwaliny chmurom obliczeniowym.

Przyjmuje się, że termin *„cloud computing"* po raz pierwszy został użyty przez CEO firmy Google, Erica Schmidta, w trakcie konferencji w 2006 r. Tego samego roku Amazon użył tej nazwy publikując pionierską w swoim

gatunku usługę Elastic Compute Cloud (EC2), czyli możliwość wynajmu serwera wirtualnego w ich chmurze.

1.3. Historia najnowsza. Współczesne wyzwania

Odkąd Amazon uruchomił usługę EC2, nastąpił gwałtowny rozwój rozwiązań chmurowych. Praktycznie w tym samym czasie rozwinęły się technologie postrzegane w czasie pisania niniejszej pracy jako rewolucyjne. Na potrzeby opracowania wyróżniono następujące grupy zmian:

- rozwój chmur publicznych (sekcja 1.3.1),
- rozwój rozwiązań opartych o kontenery (sekcja 1.3.2),
- rozwój technik programistycznych wyznaczających najlepsze praktyki tworzenia aplikacji dla chmury (sekcja 1.3.3),
- rozwój technologii bazodanowych dla systemów rozproszonych oraz metod analizy danych (sekcja 1.3.4),
- rozwój technologii przyśpieszających dostęp do usług chmurowych (sekcja 1.3.5).

1.3.1. Rozwój chmur publicznych

W 2003 roku firma Citrix stworzyła Xen - system operacyjny przeznaczony tylko do wirtualizacji innych systemów operacyjnych. Niedługo potem Xen został udostępniony jako oprogramowanie otwartoźródłowe. W marcu 2006 r. roku Amazon na podstawie Xena opracował swój produkt EC2. W ciągu 18 miesięcy zaczęło z niego korzystać ponad pół miliona osób.[2]

W kolejnych latach powstały kolejne chmury publiczne oraz nastąpił rozwój rozwiązań typu PaaS (ang. *platform as a service*). W czerwcu 2007 r. powstał Heroku, będący pionierem w tej kategorii. Niecały rok później Google udostępnił usługę Google App Engine. W lutym 2010 r. Microsoft udostępnił Windows Azure (w kwietniu 2014 przemianowaną na Microsoft Azure [6]).

Do tego momentu powstały już 3 największe chmury publiczne: Amazon AWS, Google Cloud Platform, Microsoft Azure. Początkowo miały zróżnicowaną ofertę. W literaturze[2] z 2011 r. porównywanie tych platform sprowadzało się do zakresu oferowanych przez nie usług. Z biegiem czasu różnice

Typ usługi	Dostawca		
	AWS	GCP	Azure
serwery wirtualne	2006 – Amazon EC2 [1]	2013 – Google Compute Engine [31]	2010 – Azure Virtual Machines [28]
platforma aplikacji	2011 – AWS Elastic Beanstalk [29]	2008 – Google App Engine [31]	2010 – Azure Cloud Services [28]
bazy SQL	2009 – Amazon Relational Database Service [30]	2011 – Google Cloud SQL [24]	2010 – Azure SQL Database [28]
bazy klucz-wartość	2007 – Amazon SimpleDb (w 2012 Amazon DynamoDB) [32]	2008 – Google Bigtable [23] (jako część App Engine)	2010 – Azure Table Storage [28] (2017 – Azure Cosmos DB)
serverless	2014 – AWS Lambda [7]	sierpień 2018 – Google Cloud Functions [33]	2016 – Azure Functions [7]
Kubernetes	2018 – Amazon Elastic Container Service for Kubernetes [26]	2015 – Google Kubernetes Engine [25]	2017 – Azure Container Service [27]

Tablica 1.1: Porównanie historii udostępniania usług chmurowych wśród głównych dostawców

zatarły się – obecnie usługa publikowana przez jednego dostawcę po kilku miesiącach jest dostępna także u innych. Tabela 1.1 prezentuje porównanie tempa rozbudowy zakresu usług.

1.3.2. Konteneryzacja i DevOps

W tym samym czasie rozwijały się technologie przeznaczone do wystawiania własnych usług chmurowych. W 2010 r. powstała otwartoźródłowa platforma **OpenStack** służąca do świadczenia własnych usług typu IaaS. Rok później powstał **Cloud Foundry** – również otwartoźródłowy projekt, pozwalający na prowadzenie własnej platformy typu PaaS do tworzenia apli-

kacji w chmurze.

Cloud Foundry stanowił odpowiedź na główny problem chmur publicznych – tworzenie aplikacji na wybraną chmurę wiązało ją z mechanizmami specyficznymi dla tej chmury. Brakowało pomostu, który oddzieliłby warstwę tworzenia aplikacji w chmurze od wybranego usługodawcy, bez konieczności tworzenia serwerów wirtualnych na których trzeba instalować określone środowisko. [3]

Cloud Foundry jest niezależny od infrastruktury, może działać zarówno na OpenStacku jak i na serwerze wirtualnym w chmurze publicznej. Zapewnia możliwość wyboru dowolnych technologii do wytwarzania oprogramowania. Do działania używa kontenerów zgodnych ze standardem OCI (Open Container Initiative). [3]

Konteneryzacja to metoda wirtualizacji na poziomie systemu operacyjnego, gdzie na działającym systemie operacyjnym można uruchomić wiele odizolowanych od siebie instancji przestrzeni użytkownika. Każda taka instancja to kontener. Kontener zapewnia odpowiednie środowisko do działania dla umieszczonej w nim aplikacji, np. środowisko uruchomieniowe .NET Core w odpowiedniej wersji. [3, 5]

Cloud Foundry zapewnia orkiestrację kontenerów oraz obsługę procesu CI/CD (ciągłej integracji i ciągłego dostarczania, ang. *continuous integration / continuous delivery*).[3] Stanowi narzędzie pozwalające z powodzeniem wdrażać strategie DevOps.

DevOps to słowo po raz pierwszy użyte 2009 r. na konferencji w Belgii. Stanowi połączenie ze sobą słów *„development”* (rozwój) i *„operations”* (eksploatacja). Wskazuje to na zacieśnianie współpracy pomiędzy zespołami zajmującymi się rozwojem oprogramowania, a zespołami, które je wdrażają. Współpraca ta ma doprowadzać do automatyzacji procesów towarzyszących wytwarzaniu oprogramowania, co w efekcie minimalizuje czas potrzebny od wprowadzenia poprawki do wdrożenia na środowisko produkcyjne, zachowując przy tym niezawodność. Jako strategie DevOps określa się wszelkie technologie i narzędzia inżynierii oprogramowania, które dają taki efekt. [8]

Temat DevOps jest mocno powiązany z chmurami, ponieważ jego koncepcje znacząco redukują złożoność towarzyszącą tworzeniu wysoce skalowalnego oprogramowania. Jednocześnie wychodzą one naprzeciw wymaganiom społeczeństwa coraz bardziej przyzwyczajonego do częstych aktualizacji oprogramowania, szybkiej reakcji na błędy oraz braku przestojów towarzyszących wdrażaniu nowych wersji. W przypadku chmur obliczeniowych, w których jedna usługa internetowa może mieć np. kilkadziesiąt instancji na

różnych węzłach, taka automatyzacja ma kluczowe znaczenie. W związku z tym pojęcie DevOps obecnie wchodzi w kanon pojęć związanych z chmurą.

Istotny element DevOps stanowią wcześniej wspomniane kontenery. Idea kontenerów kiełkowała już kilka lat wcześniej, np. Solaris Containers z 2004 r. lub OpenVZ (2005 r.) i LXC (Linux Containers – 2008 r.), które w istocie do działania wykorzystywały UNIX-owy program chroot z 1982 r.[34] Jednak prawdziwe ożywienie spowodowało dopiero pojawienie się Dockera – sprawił iż tworzenie kontenerów oraz przenoszenie ich pomiędzy systemami stało się łatwe. Na nim bazuje kilka projektów intensywnie rozwijanych w czasie pisania niniejszej pracy.

Docker to system kontenerów, który powstał jako element platformy dotCloud, świadczącej usługi typu PaaS. Spopularyzował się w dość osobliwy sposób. Założyciel myślał, że tylko wygłosi mały referat na PyCon w 2013 r., lecz okazało się, że na konferencji było dużo osób, a jego referat był głównym elementem programu. Do końca roku Docker miał 100 mln pobrań, 3 miesiące później – 300 mln, a w czerwcu 2017 – 13 miliardów.[35] Z kolei platforma dotCloud nie okazała się sukcesem – w 2014 r. Docker Inc. sprzedał ją niemieckiej firmie cloudControl GmbH, która w grudniu 2015 r. zbankrutowała, a w marcu 2016 r. zamknęła usługę.[36]

Logo Dockera dobrze opisuje jego istotę. Przedstawia wieloryba transportującego kontenery. Miało to nawiązywać do tego, iż załadowanie statku trwa dłużej niż transport statkiem. Ma to swoją alegorię do programowania. W tradycyjnym podejściu, gdyby zechcieć uruchomić np. aplikację napisaną w ASP.NET, najpierw trzeba byłoby poświęcić kilka godzin na pobieranie i instalację narzędzi: Visual Studio z narzędziami sieci Web, serwer IIS oraz SQL Server LocalDB. Natomiast w podejściu Dockera, posiadając zbudowany obraz można by tę aplikację uruchomić od razu. Razem z krokami budowania obrazu byłoby to:

1. napisanie pliku Dockerfile z definicją zależności: system Windows, MSBuild, .NET Framework, serwer IIS, menadżer pakietów NuGET,

2. użycie CLI Dockera oraz pliku Dockerfile do zbudowania tzw. obrazu Dockera,

3. użycie obrazu Dockera do zbudowania i uruchomienia aplikacji w kontenerze.

Obrazy Dockera działają wszędzie, gdzie jest zainstalowany Docker – niezależnie od systemu operacyjnego czy środowisk jakie ma zainstalowane. Po-

nadto zależności takie jak np. system operacyjny Ubuntu występują w wersjach „odchudzonych" - przykładowo obraz Minimal Ubuntu 18.04 zajmuje jedynie 29 MB.[37]. Najlżejszym obrazem Linuxa jest dystrybucja Alpine wielkości niecałych 4 MB, stąd używana jest jako baza dla obrazów zapewnianych przez Docker.[11] Podejście Dockera stwarza następujące możliwości:

- Na jednej maszynie może jednocześnie działać wiele aplikacji wymagających różnych zależności, lub nawet różniących się jedynie wersjami, w izolacji od siebie oraz maszyny na której są uruchomione.

- W procesie wytwarzania oprogramowania gwarantuje użycie identycznych wersji zależności na środowisku deweloperskim, testowym oraz produkcyjnym. Eliminuje to problem, gdy na zgłoszony błąd słyszy się odpowiedź „u mnie działa".

- Obrazy Dockera są lekkie: współdzielą wiele zasobów systemu operacyjnego, dzielą między sobą inne obrazy Dockera.[38] Na jednej maszynie może działać wiele kontenerów Dockera, podczas gdy maszyn wirtualnych może być zaledwie kilka.[5]

Powyższe znacząco ułatwia tworzenie aplikacji „dla chmury", orkiestrację chmury oraz stosowanie strategii DevOps.

W 2014 r. Google udostępnił system **Kubernetes**, służący do orkiestracji kontenerów w chmurze. Wywodzi się z projektu Borg, będącego wewnętrznym narzędziem Google do obsługi 2 miliardów[10] kontenerów tygodniowo, dopracowywanym w tej firmie od ponad dekady.[39] Jako główny system kontenerów wykorzystuje Dockera.[4] Może zostać skonfigurowany zarówno na fizycznych serwerach jak i na serwerach wirtualnych (można też je mieszać).[9]

Na pierwszy rzut oka Kubernetes przypomina Cloud Foundry, jest to jednak zupełnie inne narzędzie. Tabela 1.2 prezentuje porównanie tych dwóch platform.

Do tego momentu kontenery jawią się jako antidotum na przywiązanie do wybranego dostawcy usług chmurowych. Aplikacje w kontenerze Dockera mogą używać szerokiego wachlarza technologii, nie tylko otwartoźródłowych jak np. MongoDB, Node.js, .NET Core. Jako zależność można podać np. OpenJDK czy obraz systemu Windows Server udostępniany przez Microsoft, następnie można do niego doinstalować inne zależności komendą `RUN`

Cloud Foundry	Kubernetes
powstał w 2011 r.	powstał w 2014. r.
wysoki poziom automatyzacji – rozpoznaje zależności aplikacji, sam buduje kontener i konfiguruje skalowanie aplikacji	więcej kontroli nad kontenerami, lecz także więcej pracy dla deweloperów. Dla większej automatyzacji należy sięgnąć po rozwiązania PaaS bazujące na Kubernetes, takie jak RedHat OpenShift [10]
minimalne wymagania – 40 GB HDD, 8 GB RAM, 4 rdzenie CPU	2 GB RAM, 2 rdzenie CPU. Na tyle lekki, że można go używać na Raspberry PI [40]
domyślnie używa własnego systemu kontenerów, ale pozwala na korzystanie z obrazów Dockera	stosuje obrazy Dockera jako główny system kontenerów, ale pozwala zastąpić go innym

Tablica 1.2: Główne różnice pomiędzy Cloud Foundry i Kubernetes

w Dockerfile. [11] Dostępność platformy Kubernetes wyznaczyła standard w zarządzaniu kontenerami na szeroką skalę.

Problemem, który w wielu przypadkach mógł podnosić barierę wejścia do stosowania takich rozwiązań, była konieczność samodzielnej instalacji Kubernetes lub podobnego narzędzia na serwerze wirtualnym w chmurze. Chmury publiczne wyszły naprzeciw tym oczekiwaniom i zaczęły udostępniać usługi typu **CaaS** (ang. *Containers-as-a-Service*), nazywane też KaaS (ang. *Kubernetes-as-a-Service*). Jako przykłady można wymienić:

- **Google Kubernetes Engine** (wcześniej: Google Container Engine) – usługa udostępniona w sierpniu 2015 r. [25] z gotowym do użycia Kubernetes. Można ją przetestować za darmo w ramach kredytu 300 USD do testowania Cloud Platform.[41]

- **AKS – Azure Kubernetes Service** (wcześniej: Azure Container Service) – początkowo od 2015 r. wspierała orkiestratory Mesosphere DC/OS oraz Docker Swarm, jednak na fali popularności Kubernetes firma Microsoft uznała go za standard i w 2017 r. opracowała wsparcie dla Kubernetes. W związku z tym nastąpiła także zmiana nazwy usługi. [27] Usługa jest darmowa[42], opłaty są naliczane za użyte maszyny wirtualne, które przez rok również w ograniczonym zakresie są bezpłatne.[43]

- **Amazon ECS – Amazon Elastic Container Service** – usługa dostępna od kwietnia 2015 r., jest to własna implementacja orkiestracji kontenerów Dockera stworzona przez Amazona. Pozwala na 2 typy rozliczeń: płatność za zużycie zasobów przez kontener lub płatność za przechowanie i uruchamianie aplikacji na EC2.[46] Nie występuje na liście usług dostępnych do wyprówowania za darmo.[47]

- **Amazon EKS – Amazon Elastic Container Service for Kubernetes** – usługa została uruchomiona w czerwcu 2018 r. Publikując usługę Amazon chwalił się, że na podstawie danych Cloud Native Computing Foundation (założonej wraz z powstaniem Kubernetes), 57% firm korzystających z Kubernetes robiło to na serwerach wirtualnych AWS. Płatność wynosi 0.20 USD za godzinę dla każdego klastra w usłudze a dodatkowo za zużycie zasobów AWS.[48] Nie występuje na liście usług dostępnych do wyprówowania za darmo.[47]

Dostępność tych usług ucieleśnia wizję tworzenia aplikacji dla chmury tak że:

- Do tworzenia aplikacji można wybrać dowolną technologię i wciąż koncentrować się na tworzeniu tej aplikacji, a nie zapewnianiu jej odpowiedniego środowiska na serwerze wirtualnym.

- Istnieje standardowy sposób tworzenia obrazu kontenera (Docker).

- Istnieje standardowy sposób orkiestracji kontenerów (Kubernetes). Wiedzę na temat korzystania z niego można zastosować do projektów działających na chmurach publicznych, prywatnych i hybrydowych.

1.3.3. Mikrousługi i serverless

Gwałtowny rozwój chmur obliczeniowych poskutkował również wytworzeniem nowych wzorców projektowych, wzorców architektonicznych oraz dobrych praktyk używanych podczas tworzenia aplikacji przeznaczonej dla chmury. W chmurze może działać praktycznie każdy program, wszakże odpowiednio przygotowana maszyna wirtualna wszystko przyjmie. Chcąc jednak maksymalnie wykorzystać atuty chmury, a jednocześnie zapewnić niezawodność i jak najniższy koszt eksploatacji, należy zapoznać się ze specyficznym sposobem tworzenia aplikacji dla chmury. Aplikację zgodną z tymi wytycznymi określa się jako ***cloud native***.

Jak zostało wspomniane wcześniej, platforma Heroku była pionierem jeśli chodzi o udostępnianie programistom platformy do tworzenia aplikacji w chmurze (2007 r.). Współzałożyciel Heroku, Adam Wiggins wraz z zespołem, w 2012 r. na podstawie doświadczeń zebranych w Heroku stworzyli metodykę „twelve-factor" (ang. ***The Twelve-Factor App***). Jest to zbiór podstawowych 12. reguł dotyczących programowania oraz pracy z aplikacjami przeznaczonymi dla chmury. Reguły te są często przywoływane w literaturze zajmującej się tym tematem.[3, 12] Można się z nimi zapoznać odwiedzając stronę `https://12factor.net` (w czasie pisania pracy strona była dostępna w 13 językach, w tym w języku polskim).

Osobne „dobre praktyki" powstały dla systemów rozproszonych używających kontenerów (w tym Kubernetes). Dzielą się na:

- Wzorce dla kontenerów występujących na jednym węźle (ang. *Single-node patterns*). Należą do nich wzorce: *Sidecar, Ambassador, Adapter.*
- Wzorce dla kontenerów występujących na różnych węzłach (ang. *Multi-node patterns*). Koncentrują się na właściwych metodach koordynacji działań pomiędzy różnymi maszynami. Są to wzorce np. komunikacji z użyciem zdarzeń czy kolejek. [9, 13]

Praktyki wyznaczone przez powyższe wzorce w dużej mierze opierają się na **mikrousługach**. Mikrousługi wywodzą się z **SOA**. Termin SOA (ang. *Service oriented architecture*) został pierwszy raz użyty przez analityka firmy Gartner podczas wykładu. Utworzył on ten termin, ponieważ zwrot „klient/serwer" tracił na swoim pierwotnym znaczeniu, gdyż zamiast nazywać tak aplikację dla serwera lub klienta, ludzie zaczęli tak nazywać fizyczne maszyny. Następnie inni analitycy w 1996 r. publikowali na ten temat raporty. Bardziej znaczące użycie terminu SOA należało do Microsoftu. W 2000 r. opisywał zbiór standardów do komunikacji komputerów przez Internet, gdzie przedstawił architekturę SOA jako ważną, choć nie niezbędną, dla Web Services. Wkrótce termin podchwyciły inne firmy, w tym takie jak IBM, Oracle, HP, publikując nowe koncepcje czy narzędzia powiązane z SOA. [14]

SOA to po polsku architektura zorientowana na usługi. Polega na podzieleniu systemu informatycznego na odrębne części, które mogą znajdować się na różnych maszynach i komunikować się ze sobą przez sieć. Główne koncepcje SOA to:

- usługi – rozumiane jako samodzielne komponenty realizujące daną funkcjonalność biznesową,

- wysoka interoperacyjność – łatwe łączenie ze sobą systemów różnych typów,

- luźne wiązania – jak najmniejsza liczba zależności. [14]

SOA to bardzo ogólne pojęcie, nie definiuje jakiej wielkości powinny być poszczególne usługi. Wraz z rozwojem technologii chmurowych, nastała potrzeba dostosowania paradygmatu SOA do wyzwań związanych ze skalowalnością, szybkością wytwarzania oprogramowania oraz potrzebą adaptacji nowych technologii dla istniejących rozbudowanych systemów. Odpowiedzią na ten problem okazały się mikrousługi.

Mikrousługi (ang. *microservices*) stanowią szczególny przypadek SOA. Ich główna cecha: mają być małe. Zamiast tworzyć jedną aplikację, która robi wszystko, system ma składać się z wielu małych, niezależnych od siebie komponentów, odpowiedzialnych za tylko jedną rzecz. [12]

Z jednej strony powinny grupować powiązane ze sobą funkcjonalności, z drugiej strony nie wolno doprowadzać do sytuacji, gdy serwis może stać się zbyt duży i trudny w utrzymaniu. Pomocne w wyobrażeniu jakiej wielkości powinna być przeciętna mikrousługa są następujące wskazówki:

- Zasada pojedynczej odpowiedzialności, wywodząca się z zasad SOLID, a zaadoptowana do usług. Polega na umieszczaniu razem elementów, które będą zmieniać się z tego samego powodu. [16]

- Kod mikrousługi powinien być na tyle mały, żeby dało się go przepisać w dwa tygodnie. [16]

- Mikrousługa powinna być na tyle mała, żeby do pracy z nią wystarczył jeden zespół programistów. [15]

- W przypadku gdy mikrousługa jest często używana i potrzebuje do działania wielu instancji, nie powinna jednocześnie powielać funkcjonalności rzadziej używanych i tym sposobem marnować zasobów. [15]

W odniesieniu do chmur, mikrousługi wprowadzają następujące udogodnienia: [12, 15, 16]

- **Efektywne skalowanie.** Gdy określona funkcjonalność systemu jest często używana, uruchamia się więcej instancji mikrousługi odpowiedzialnej za tę funkcjonalność. Nie trzeba zużywać więcej zasobów niż jest to potrzebne.

- **Lepsze zapanowanie nad kodem.** Do mniejszego programu łatwiej wprowadzać poprawki, w razie potrzeby przepisanie go do nowszych technologii nie powinno stanowić problemu. Każda mikrousługa może być napisana przy użyciu innych technologii. Większe zmiany w kodzie oddziałują tylko na daną mikrousługę, nie na cały system.

- **Szybsze wdrażanie.** Procesy ciągłej integracji i ciągłego dostarczania (CI/CD) szybciej działają, gdy mają do kopiowania, budowania i testowania mniejszą ilość kodu.

Popularność kontenerów, mikrousług oraz modelu *utility computing* (rozliczanie za rzeczywiste użycie zasobów komputerowych) doprowadziła do wytworzenia nowego paradygmatu tworzenia aplikacji w chmurze, znanego jako ***serverless*** (z ang. „bez serwera"). Serverless jest obecnie obiecującą nowością jeśli chodzi o przetwarzanie w chmurze. Nazwa ta zaczęła zdobywać popularność na początku 2016 r. [17]

Aplikacje tego typu zamiast być przez cały czas uruchomione na serwerze w oczekiwaniu na zapytania, mają być uruchamiane dopiero gdy przyjdzie określone zdarzenie (ang. *event-driven computation*). Zdarzeniem może być np. nowe zapytanie HTTP, nowy wpis w kolejce, wywołanie po upłynięciu interwału czasowego. Wówczas zostaje powołana do życia aplikacja przeznaczona tylko do obsługi danego zdarzenia i działa tylko na krótki czas obsługi zdarzenia (jest to maksymalnie około kilka minut). Aplikacje te to w istocie mikrousługi, tyle że posiadające tylko jedną funkcję – stąd nazywa się je po prostu funkcjami. Dla lepszego odróżnienia od mikrousług określa się je jako „nanousługi" (ang. *nanoservices*). [7, 17, 13]

Platforma w chmurze stosująca model serverless określana jest jako **FaaS** (ang. *Function-as-a-Service*). Pierwszym znaczącym dostarczycielem tej usługi był Amazon – już w 2014 r. uruchomił usługę AWS Lambda typu FaaS. W 2016 r. Microsoft udostępnił usługę Azure Functions. W Google Cloud Platform tego typu usługa została oficjalnie udostępniona w sierpniu 2018 (miesiąc przed wydaniem niniejszej pracy), choć wersja poglądowa była dostępna wcześniej.

Posługując się modelem serverless, usługodawca może stosować rozliczenie według ilości wywołań poszczególnych funkcji. Dzięki temu nie trzeba płacić za programy bezczynnie czekające na przychodzące zapytania.

Dla programisty użycie nanoserwisów oznacza mniej projektowania jeśli chodzi o optymalną architekturę mikrousług. Mikrousługi grupowały ze sobą niektóre elementy, natomiast w nanousługach każda funkcja stanowi

osobny byt. Według Maddie Stigler, autorki książki *„Beginning Serverless Computing”*, użycie architektury serverless oszczędza połowę czasu potrzebnego na stworzenie skalowalnych aplikacji od podstaw.[7]

Rozwijane są także inicjatywy niezwiązane z usługami FaaS konkretnych dostawców. W 2015 r. powstał projekt Serverless Framework,[49] służący do odseparowania logiki konkretnej usługi FaaS od logiki biznesowej funkcji. Ponadto zapewnia narzędzia wspierające proces tworzenia funkcji.

W 2017 r. powstał projekt Kubeless.[50] Jest to otwartoźródłowy framework dla Kubernetes, który ma działać w sposób podobny do sposobu wyznaczonego przez AWS Lambda, Azure Functions oraz Google Cloud Functions.[13]

Serverless Framework obsługuje wszystkie wyżej wymienione platformy, z Kubeless włącznie.[49]

1.3.4. Bazy NoSQL i rozwiązania dla Big Data

W literaturze rozróżnia się trzy rewolucje w rozwoju technologii bazodanowych. [18]

Za pierwsze „bazy danych” uznaje się wszelkie techniki zapisu i odczytu danych z nośników, na które pozwalała technika – kart perforowanych, taśm magnetycznych, magnetycznych dysków twardych. Przed pierwszą rewolucją, każda aplikacja posiadała swoją własną implementację dostępu do danych. Nie istniały żadne mechanizmy chroniące przed uszkodzeniem danych, przed równoczesnym dostępem wielu użytkowników, nie wspominając o optymalizacjach.

Pierwszą rewolucją było stworzenie **systemu bazodanowego** (**DBMS** - ang. *Database Management System*). Wczesne systemy nie opierały się na żadnych podstawach matematycznych. Były to tzw. bazy nawigacyjne, gdzie relacje uzyskiwało się definiując połączenia pomiędzy obiektami. Struktura projektu definiowała jakie będzie można tworzyć rodzaje zapytań do bazy. Przykłady tych wczesnych baz stanowią: IMS od firmy IBM (typ hierarchiczny) oraz IDMS będący implementacją standardu CODASTYL (typ sieciowy).

Drugą rewolucję zaczął Edgar Codd, będący matematykiem-programistą oraz wieloletnim pracownikiem firmy IBM. Był on pierwszą osobą, która zwróciła uwagę na wady istniejących rozwiązań wynikające z braku podstaw matematycznych, i w 1970 r. opublikował artykuł opisujący relacyjne bazy danych. Początkowo pozostał on bez odzewu. Dopiero w 1974 r. IBM zaczął prace nad prototypem zwanym Systemem R. Prototyp był gotowy w 1976 r.

Poza modelem opracowanym przez Codd'a, na potrzeby systemu stworzono język SQL do tworzenia zapytań do bazy danych oraz opracowano zasady obsługi równoległych zapytań. Jim Gray zdefiniował transakcje ACID.

Pierwszą komercyjną **relacyjną bazą danych** (**RDBMS** - *Relational Database Management System*) był Oracle. Założyciel Oracle, Larry Ellison był zaznajomiony zarówno z pracą Codd'a jak i Systemem R. Wierząc relacyjne bazy danych, w 1979 r. wydał pierwszą na świecie komercyjną relacyjną bazę danych obsługującą język SQL. [19]

W tym samym czasie (od roku 1977) pionierski stał się również projekt Ingres, stworzony przez Michaela Stonebraker'a. Początkowo jako język zapytań stosował QUEL, popularność bazy Oracle wymusiła jednak konieczność obsługi SQL. W połowie lat 80. na bazie Ingres powstał projekt Postgres (post-ingres) znany dzisiaj jako PostgreSQL.

Po sukcesie Oracle IBM opublikował komercyjną bazę SQL/DS w 1981 r. W późniejszych latach powstawały inne relacyjne bazy chętnie stosowane do dziś: Microsoft SQL Server, PostgreSQL, MySQL. Wszystkie te bazy co do zasady są do siebie podobne – opierają się o model Codd'a, transakcje ACID oraz język SQL. [19]

Potencjalna rewolucja miała szanse dokonać się po latach 80., gdy zaczął się popularyzować paradygmat programowania obiektowego. Istniejące języki zaczęły występować w wersji obiektowej, np. Object Pascal, powstały języki od początku obiektowe, np. Java. Niektórym programistom nie podobała się rozbieżność pomiędzy reprezentacją danych w obiekcie, a w relacyjnej bazie danych. W grudniu 1989 r. powstał manifest obiektowych baz danych (OODBMS, ang. *Object Oriented Database Management System*), w którym bazy relacyjne jawiły się jako relikt przeszłości a obiektowe jako ich następcy. Jednak na przestrzeni lat 90. okazało się, że bazy obiektowe nie zdobyły rynku, nawet w postaci nowych opcji w popularnych bazach jak np. Oracle. Według Guy'a Harrison'a, autora książki *„Next Generation Databases: NoSQL, NewSQL, and Big Data"*, założenia baz obiektowych koncentrowały się wokół korzyści dla programisty, ignorując korzyści dla biznesu (np. brak wsparcia dla języka SQL będącego powszechnie w użyciu). Dodatkowo dostępność narzędzi ORM (ang. *Object-relational mapping*) znacząco ułatwiła korzystanie z RDBMS w językach obiektowych.

Trzecia rewolucja

Trzecia rewolucja rozpoczęła się za sprawą coraz większego znaczenia systemów rozproszonych oraz konieczności analizy dużej ilości danych. Zaczęło

się od Google. W 1996 r. powstała wyszukiwarka Google, korzystająca z algorytmu PageRank. W 2005 r. Google był największą stroną internetową na świecie, stroną zbierającą i przetwarzającą dane o wszystkich innych stronach. Potrzebowano nowych narzędzi do rozwiązania problemu, z którym jako pierwsi mieli styczność. W kolejnych latach Google publikował szczegóły swoich rozwiązań, co miało duży wpływ na rozwój dalszych technologii:

- w 2003 r. – system plików **GFS** (Google File System) dla systemów rozproszonych, gdzie pula zasobów dyskowych jest widoczna jako jedno,
- w 2004 r. – model **MapReduce** pozwalający na przetwarzanie danych w systemach rozproszonych,
- w 2006 r. – baza danych **BigTable**.

Ponadto w 2005 r. Google opatentował Modular Data Center, czyli sposób na szybką rozbudowę centrum danych. Zamiast z osobna dodawać serwery, mogli dodawać od razu po tysiąc naraz.

Rozwiązania Google zainspirowały inne projekty. W 2004 r. powstał projekt Apache Nutch, który miał być otwartoźródłową wyszukiwarką internetową. Gdy Google opublikował GFS i MapReduce, dla Nutch to był sygnał jak rozwiązywać problemy skalowalności. W 2006 r. Yahoo! zainwestował w projekt, pomimo że był otwartoźródłowy, ponieważ chciał użyć go w swoim produkcie. Adaptację rozwiązań Google wyodrębniono jako osobny projekt. Tak w 2007 r. powstał **Apache Hadoop**. [20]

Hadoop to nie tylko projekt, ale podejście do skalowalnego przetwarzania danych. [20] Obecnie Hadoop to praktycznie synonim Big Data. [21] Termin **Big Data** (z ang. „duże dane") po raz pierwszy został użyty w 1999 r., jednak zaczął być powszechnie używany odkąd Google użył tego terminu w dokumencie opisującym MapReduce w 2004 r. Początkowo (w 2001 r.) opisywano Big Data za pomocą „trzech V": *volume, variety, velocity* (ilość, różnorodność, szybkość). W 2014 r. rozszerzono to do „pięciu V" o: *veracity, value* (wiarygodność, wartość).[22] Właściwości te wyznaczają obszar zainteresowań, gdzie zastosowanie mają takie narzędzia jak Hadoop:

- *volume* – ilość danych,
- *variety* – stopień różnorodności źródeł danych,
- *velocity* – szybkość z jaką są tworzone nowe dane oraz jak szybko należy je przetworzyć,

- *veracity* – na ile można ufać tym danym, czy źródło danych jest zanieczyszczone,
- *value* – czy dane mają jakiekolwiek znaczenie, czy analiza tych danych tylko tworzy koszty czy daje wartość.

Powyższe oznacza, że wcale nie trzeba mieć bardzo dużego zbioru danych, aby potrzebować Big Data. [20]

Wracając do Hadoopa, rok po jego wdrożeniu Yahoo! ogłosił, że ich klaster Hadoopa ma 5 petabajtów dysku i ponad 10 000 rdzeni CPU, co generowało im cały indeks wyszukiwarki. Podobnie Facebook skorzystał z Hadoopa w 2007 r. i już w 2008 r. mieli 2500 rdzeni CPU, a w 2012 r. 100 petabajtów dysku, co wyparło użycie rozwiązań Oracle w tej firmie.

Do głównych zalet Hadoopa zalicza się:

- tanie przechowywanie danych – jest to możliwe na zwykłych dyskach;
- skalowanie operacji I/O – ponieważ zamiast dodawać tylko dyski, dodaje się nowe komputery;
- niezadowność – jeśli zepsuje się jeden komputer, to jego rolę przejmie inny
- *„schema on read”* – struktura danych tworzy się podczas czytania źródła danych, w przeciwieństwie do tradycyjnego modelu relacyjnego, gdzie przy zapisie danych należy się zastosować do istniejącego schematu (ang. *schema on write*).

Klasyczny Hadoop posiadał ograniczenia jeśli chodzi o użycie innego modelu przetwarzania zadań niż MapReduce. W 2012 r. powstała wersja 2.0, gdzie doszła platforma YARN (ang. *Yet Another Resource Negotiator* lub rekursywnie *YARN Application Resource Negotiator*). YARN pozwala na użycie MapReduce jako jednego z możliwych frameworków do zastosowania, co pozwala np. na użycie SPARK-a pozbawionego niektórych niedogodności MapReduce (krótkotrwałość zadań, brak możliwości trzymania roboczego zbioru danych w pamięci). [20, 21]

Stworzenie BigTable oraz Hadoopa spowodowało powstanie wielu nowych systemów bazodanowych, które miały lepiej działać na systemach rozproszonych. Dzielą się na:

- **Bazy NoSQL** – bazy nierelacyjne. Zalicza się do nich wszelkie rozwiązania, które dzięki niespełnieniu niektórych reguł ACID mogą pozwolić na lepszą skalowalność i/lub szybkość przetwarzania.

- **Bazy NewSQL** – bazy posiadające możliwość skalowania, zachowujące właściwości ACID baz relacyjnych.

W 2005 r. Michael Stonebraker (założyciel Ingres i PostgreSQL) opracował bazę C-Store, postulującą **model kolumnowy**. Pomysł opiera się o tradycyjną relacyjną bazę danych, gdzie w tabeli zostają zamienione miejscami wiersze z kolumnami. Taka zamiana powoduje szybsze działanie zapytań agregujących, gdyż wszystkie potrzebne dane znajdują się obok siebie. Poza tym ponieważ obok siebie znajdują się dane bardzo podobnej postaci, można zastosować lepszą kompresję, nawet przez traktowanie każdej kolejnej danej jako zmiana względem poprzedniej.

W 2007 r. Amazon opracował bazę Dynamo, który stał się wzorcem dla baz w modelu **klucz-wartość**. Jedną z nich jest Apache Cassandra (2008 r.), baza typu ***wide-column***. W bazach tego typu w miejscu wartości jest dynamiczna liczba kolumn, co pozwala np. na przeszukiwanie po danej kolumnie lub aktualizacja tylko danej kolumny zamiast całego obiektu. Podobnym projektem jest Apache HBase (2008 r.).

Również w 2007 r. powstała pierwsza **baza dokumentowa**, która mogła być postrzegana jako alternatywa dla RDBMS. Od 2000 r. powstawały bazy XML, ale raczej mające na celu zarządzanie już istniejącymi dokumentami w tym formacie. XML rozpowszechniał się za sprawą popularności AJAX, później zaczął być zastępowany formatem JSON. Takie dane były już formą dokumentu, a stworzenie systemu do przechowywania ich było kwestią czasu. W 2005 r. powstała baza dokumentowa Apache CouchDB, na razie obsługująca XML, a w 2007 r. była to pierwsza znacząca baza dokumentów JSON. W 2009 r. powstała baza MongoDB, przechowująca dokumenty JSON (wewnętrznie zapisywane jako BSON – *Binary JSON*). Stanowi dziś najpopularniejszą bazę tego typu.

Kolejnym osiągnięciem z 2007 r. był artykuł Michaela Stonebrakera pt. „The end of an architectural era: (it's time for a complete rewrite)" („Koniec współczesnych architektur, czas przepisać je na nowo" – tłumaczenie własne). Opisał tam koncepcję bazy H-store typu ***in-memory***, która do transakcji nie potrzebuje wykonywać operacji I/O na dysku. Komercyjną implementacją tego pomysłu była baza VoltDB, utworzona w 2008 r. Jest to baza typu NewSQL.

Rok 2007 był przełomowy także dla modelu **baz grafowych** – powstała baza Neo4J. Bazy grafowe przypominają wczesne, nawigacyjne bazy danych, lecz operują na wyższym poziomie abstrakcji oraz pozwalają na przeszukiwanie grafu do zadanego poziomu. Są stworzone dla danych, gdzie najważ-

niejszą właściwością są relacje z innymi danymi, np. relacje w social media. Do odpytywania bazy grafowej używa się języków Cypher oraz Gremlin. Neo4J stanowi bazę czysto grafową, jednak na dzień dzisiejszy nie obsługuje systemów rozproszonych, gdyż łączenie wierzchołków grafu z różnych serwerów znacząco obniżyłoby jego wydajność. Do projektów przetwarzania grafów na systemach rozproszonych należą: Apache Giraph (2016 r.), GraphX (część Apache Spark, 2014 r.), Titan (2015 r.) – działający jako nakładka na HBase czy Cassandrę.

1.3.5. CDN i Fog Computing

1.4. Wymagania stawiane współczesnym chmurom

1.5. Dokąd zmierza rozwój CC.

Rozdział 2.

Porównanie różnych podejść

2.1. Chmury publiczne

2.2. Chmury prywatne

Rozdział 3.

Analiza wyboru platformy dla aplikacji mobilnej

3.1. Wytumaczenie czemu to jest typowy projekt

3.2. Wymagania co trzeba wziąć pod uwagę

3.3. Werdykt, kandydatka nr 1, 2 ,3

Rozdział 4.

Opis wdrożenia aplikacji zgodnie z kandydatką nr 1

Podsumowanie

Spis rysunków

Spis tablic

1.1. Porównanie historii udostępniania usług chmurowych wśród głównych dostawców 16
1.2. Główne różnice pomiędzy Cloud Foundry i Kubernetes . . . 20

Bibliografia

[1] Sandeep Bhowmik. *Cloud Computing*. 1st. New York, NY, USA: Cambridge University Press, 2017. ISBN: 978-1-316-63810-1.

[2] Jothy Rosenberg i Arthur Mateos. *Chmura obliczeniowa. Rozwiązania dla biznesu*. Gliwice: Helion, 2011. ISBN: 978-83-246-4167-3.

[3] Rick Farmer, Rahul Jain i David Wu. *Cloud Foundry for Developers*. 1st. Birmingham, UK: Packt Publishing Ltd., November 2017. ISBN: 978-1-78839-144-3.

[4] Kelsey Hightower, Brendan Burns i Joe Beda. *Kubernetes: Up and Running*. 1st. O'Reilly Media, Inc., 2017. ISBN: 978-1-491-93567-5.

[5] Nick Antonopoulos i Lee Gillam. *Cloud Computing. Principles, Systems and Applications*. 2nd. Cham, Switzerland: Springer International Publishing AG, 2017. ISBN: 978-3-319-54645-2.

[6] Florian Klaffenbach, Jan-Henrik Damaschke i Oliver Michalski. *Implementing Azure Solutions*. 1st. Birmingham, UK: Packt Publishing Ltd., May 2017. ISBN: 978-1-78646-785-0.

[7] Maddie Stigler. *Beginning Serverless Computing. Developing with Amazon Web Services, Microsoft Azure, and Google Cloud*. 1st. Apress, 2018. ISBN: 978-1-4842-3084-8.

[8] Joakim Verona. *Practical DevOps*. 1st. Birmingham, UK: Packt Publishing Ltd., 2016. ISBN: 978-1-78588-287-6.

[9] Gigi Sayfan. *Mastering Kubernetes*. 1st. Birmingham, UK: Packt Publishing Ltd., May 2017. ISBN: 978-1-78646-100-1.

[10] Jonathan Baier. *Getting Started with Kubernetes*. 2nd. Birmingham, UK: Packt Publishing Ltd., May 2017. ISBN: 978-1-78728-336-7.

[11] Russ McKendrick i Scott Gallagher. *Mastering Docker*. 2nd. Birmingham, UK: Packt Publishing Ltd., July 2017. ISBN: 978-1-78728-024-3.

[12] Gaurav Kumar Aroraa, Lalit Kale i Kanwar Manish. *Building Microservices with .NET Core*. 1st. Birmingham, UK: Packt Publishing Ltd., June 2017. ISBN: 978-1-78588-783-3.

[13] Brendan Burns. *Designing Distributed Systems*. 1st. Sebastopol, CA, USA: O'Reilly Media, Inc., January 2018. ISBN: 978-1-492-03177-2.

[14] Nicolai M. Josuttis. *SOA in Practice*. O'Reilly Media, Inc., August 2007. ISBN: 9780596529550.

[15] Susan J. Fowler. *Production-Ready Microservices*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2017. ISBN: 978-1-491-96597-9.

[16] Sam Newman. *Budowanie mikrousług. Wykorzystaj potencjał architektury usługi*. Gliwice: Helion S.A., 2015. ISBN: 978-83-283-1381-1.

[17] Sanjay Chaudhary, Gaurav Somani i Rajkumar Buyya. *Research Advances in Cloud Computing*. 1st. Singapore, Singapore: Springer Nature Singapore Pte Ltd., 2017. ISBN: 978-981-10-5026-8.

[18] Guy Harrison. *Next Generation Databases: NoSQL, NewSQL, and Big Data*. O'Reilly Media, Inc., January 2016. ISBN: 9781484213292.

[19] Jan L. Harrington. *Relational Database Design and Implementation*. 4th. Morgan Kaufmann, April 2016. ISBN: 9780128499023.

[20] Douglas Eadline. *Hadoop 2 Quick-Start Guide: Learn the Essentials of Big Data Computing in the Apache Hadoop 2 Ecosystem*. Addison-Wesley Professional, October 2015. ISBN: 9780134050119.

[21] Vijay Srinivas Agneeswaran. *Big Data Analytics Beyond Hadoop: Real-Time Applications with Storm, Spark, and More Hadoop Alternatives*. PH Professional Business, May 2014. ISBN: 9780133838268.

[22] Saurabh Gupta i Shilpi Saxena. *Practical Real-time Data Processing and Analytics*. Birmingham, UK: Packt Publishing Ltd., September 2017. ISBN: 9781787281202.

[23] *BigTable – Wikipedia*. URL: `https://pl.wikipedia.org/wiki/BigTable` (term. wiz. 09. 09. 2018).

[24] *Google Cloud SQL: your database in the cloud*. URL: `http://googlecode.blogspot.com/2011/10/google-cloud-sql-your-database-in-cloud.html` (term. wiz. 09. 09. 2018).

[25] *Google Container Engine is Generally Available*. URL: `https://cloudplatform.googleblog.com/2015/08/Google-Container-Engine-is-Generally-Available.html` (term. wiz. 11. 09. 2018).

[26] *Amazon EKS – Now Generally Available.* URL: https://aws.amazon.com/blogs/aws/amazon-eks-now-generally-available/ (term. wiz. 09. 09. 2018).

[27] *Introducing AKS.* URL: https://azure.microsoft.com/en-us/blog/introducing-azure-container-service-aks-managed-kubernetes-and-azure-container-registry-geo-replication/ (term. wiz. 09. 09. 2018).

[28] *Introducing Windows Azure.* URL: https://azure.microsoft.com/en-us/blog/introducing-windows-azure/ (term. wiz. 09. 09. 2018).

[29] *AWS Elastic Beanstalk – Wikipedia.* URL: https://en.wikipedia.org/wiki/AWS_Elastic_Beanstalk (term. wiz. 09. 09. 2018).

[30] *Amazon Relational Database Service.* URL: https://en.wikipedia.org/wiki/Amazon_Relational_Database_Service (term. wiz. 09. 09. 2018).

[31] *Google Cloud Platform – Wikipedia.* URL: https://en.wikipedia.org/wiki/Google_Cloud_Platform (term. wiz. 09. 09. 2018).

[32] *Amazon SimpleDB – Wikipedia.* URL: https://en.wikipedia.org/wiki/Amazon_SimpleDB (term. wiz. 09. 09. 2018).

[33] *Cloud Functions serverless platform is generally available.* URL: https://cloud.google.com/blog/products/gcp/cloud-functions-serverless-platform-is-generally-available (term. wiz. 12. 09. 2018).

[34] *Operating-system-level virtualization – Wikipedia.* URL: https://en.wikipedia.org/wiki/Operating-system-level_virtualization (term. wiz. 10. 09. 2018).

[35] *Docker's Tools of Mass Innovation: Explosive Growth From Open-Source Containers to Commercial Platform for Modernizing and Managing Apps.* URL: http://www.hostingadvice.com/blog/dockers-explosive-growth-from-open-source-containers-to-commercial-platform/ (term. wiz. 02. 02. 2018).

[36] *CloudControl – Wikipedia.* URL: https://en.wikipedia.org/wiki/CloudControl (term. wiz. 10. 09. 2018).

[37] *Minimal Ubuntu, on public clouds and Docker Hub.* URL: https://blog.ubuntu.com/2018/07/09/minimal-ubuntu-released (term. wiz. 10. 09. 2018).

[38] *How is Docker different from a virtual machine?* URL: https://stackoverflow.com/a/16048358/5194338 (term. wiz. 10. 09. 2018).

[39] *Borg: The Predecessor to Kubernetes*. URL: https://kubernetes.io/blog/2015/04/borg-predecessor-to-kubernetes/ (term. wiz. 10. 09. 2018).

[40] *Setup Kubernetes on a Raspberry Pi Cluster easily the official way*. URL: https://blog.hypriot.com/post/setup-kubernetes-raspberry-pi-cluster/ (term. wiz. 01. 02. 2018).

[41] *Wypróbuj bezpłatnie Cloud Platform*. URL: https://console.cloud.google.com/freetrial/signup/ (term. wiz. 11. 09. 2018).

[42] *Azure Kubernetes Service (AKS) pricing*. URL: https://azure.microsoft.com/en-us/pricing/details/kubernetes-service/ (term. wiz. 11. 09. 2018).

[43] *Utwórz bezpłatne konto platformy Azure już dzisiaj*. URL: https://azure.microsoft.com/pl-pl/free/ (term. wiz. 11. 09. 2018).

[44] *Amazon EC2 Container Service is Now Generally Available*. URL: https://aws.amazon.com/about-aws/whats-new/2015/04/amazon-ec2-container-service-is-now-generally-available/ (term. wiz. 11. 09. 2018).

[45] *Amazon Elastic Container Service*. URL: https://aws.amazon.com/ecs/ (term. wiz. 11. 09. 2018).

[46] *Amazon Elastic Container Service Pricing*. URL: https://aws.amazon.com/ecs/pricing/ (term. wiz. 11. 09. 2018).

[47] *AWS Free Tier*. URL: https://aws.amazon.com/free/ (term. wiz. 11. 09. 2018).

[48] *Amazon EKS Pricing*. URL: https://aws.amazon.com/eks/pricing/ (term. wiz. 11. 09. 2018).

[49] *Serverless Framework – Wikipedia*. URL: https://en.wikipedia.org/wiki/Serverless_Framework (term. wiz. 12. 09. 2018).

[50] *Wydania do Kubeless*. URL: https://github.com/kubeless/kubeless/releases?after=v0.2.1 (term. wiz. 12. 09. 2018).

[51] *The Object-Oriented Database System Manifesto*. URL: https://www.cs.cmu.edu/~clamen/OODBMS/Manifesto/ (term. wiz. 13. 09. 2018).