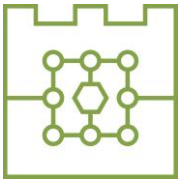
Politechnika Krakowska
im. Tadeusza Kościuszki

Wydział Informatyki i Telekomunikacji



Bartosz Szymański

Numer albumu: 148976

Praktyczny przykład implementacji bezpiecznego przeniesienia aplikacji na architekturę mikroserwisów w chmurze AWS

A practical example of an implementation of a monolithic application secure migration to AWS cloud microservices architecture

Praca magisterska
na kierunku Informatyka
specjalność Cyberbezpieczeństwo

Praca wykonana pod kierunkiem:
dr Barbara Borowik

Kraków, 2024

Spis treści

1. WSTĘP

1.1 Cel pracy

Celem niniejszej pracy dyplomowej jest zbadanie procesu migracji systemu monolitycznego do architektury mikroserwisowej opartej o usługi chmury obliczeniowej dostawcy Amazon Web Services (w skrócie AWS) pod kątem ulepszenia zabezpieczeń omawianego systemu. System objęty badaniem, to: „System do internetowego wspomagania pacjenta i lekarza", który został w całości zaprojektowany i zaimplementowany w formie monolitycznej przez autora pracy dyplomowej jako projekt inżynierski w celu ukończenia studiów inżynierskich pierwszego stopnia. Wybór systemu do przeprowadzenia analizy motywowany jest znajomością jego architektury, wpasowującą się doskonale w temat pracy oraz motywacja do jego dalszego rozwijania i ulepszania w zakresie cyberbezpieczeństwa.

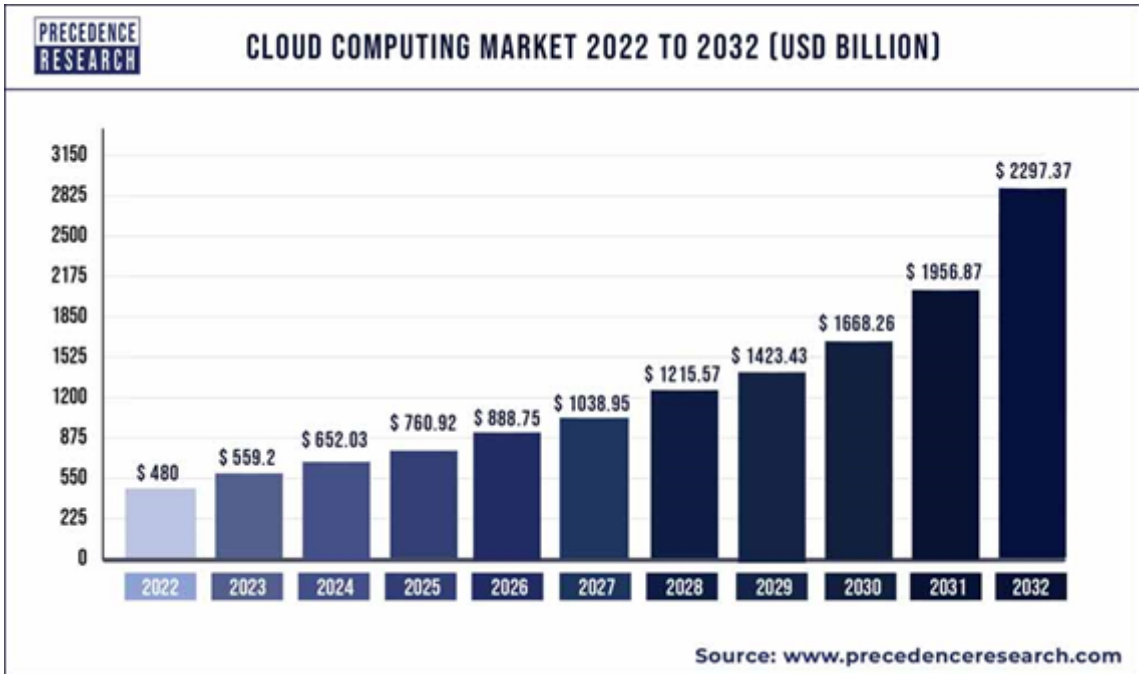
Analiza procesu migracji oraz rozwinięcia systemu zabezpieczeń systemu ma wykazać jak dobrze usługi chmurowe ochraniają swoich usługobiorców i ich oprogramowania przed popularnymi atakami hakerskimi oraz jak duży wpływ na bezpieczeństwo systemu ma jego architektura. Narzędzia używane przez autor pracy to przede wszystkim oprogramowanie wirtualizujące systemy operacyjne Docker, język programowania PHP, język programowania Java Script, zestaw narzędzi programistycznych Symfony oraz Vue.js, dodatkowo usługi AWS, takie jak: Elastic Compute Cloud (w skrócie EC2), Relational Databases (w skrócie RDS), Route 53, Code Pipeline, Code Build, Code Deploy, Elastic Cache, Simple Storage Service, Elastic Container Registry, Lambda. Również narzędzia do analizy ruchu sieciowego, takie jak: Wireshark.

Praca ta omówia aspekty takie jak: ewaluacja zastanej architektury aplikacji, planowanie architektury mikroserwisowej w chmurze AWS, ocena zagadnień bezpieczeństwa podczas migracji, implementacja i wdrażanie mikroserwisów w chmurze AWS, ocena i analiza wyników. Każdy wymieniony etap pracy jest szczegółowo omówiony, odzwierciedlając wiedzę i doświadczenie akademickie jak i zawodowe autora.

1.2 Motywacja

Motywacją do podjęcia tematu pracy są doświadczenia autora na tle zawodowym, które przyspieszyły naukę w zakresie obliczeń chmurowych. Jednak największym moty-

watorem do wykonania badania jest kontynuacja pracy akademickiej, dążenie do ciągłego rozwoju wyprodukowanego przez siebie oprogramowania i wdrażanie wiedzy pozyskanej w czasie studiów drugiego stopnia, tak by umiejętnie zabezpieczać oprogramowanie. Dodatkowym motywatorem jest również fakt, iż technologia chmurowa każdego roku zdobywa coraz większy udział na rynku usług hostingowych, a także staje się dzięki wzrastającej konkurencyjności przystępniejsza dla mniejszych firm lub prywatnych odbiorców. Serwis Precedence Research przewiduje, iż w nadchodzących ośmiu latach udział rynkowy technologii chmurowych zwiększy się około pięciokrotnie do wartości 2297 miliardów dolarów [1]. Patrz Rysunek 1.1.



Rys. 1.1. Wykres przedstawiający prognozę wzrostu udziału rynkowego technologii chmurowej

1.3 Założenia i spodziewane wyniki

Rezultatem niniejszej pracy dyplomowej jest rozebranie monolitycznego systemu na mikroserwisy z pojedynczą odpowiedzialnością logiczną. Zasada działa aplikacji nie ulega zmianie, dla użytkownika zewnętrznego wprowadzenie zmiany powinno odbyć się bez odczuwalnej różnicy. Samo rozczłonkowanie systemu odbywa jak najbardziej atomicznie się da przy rozsądnym nakładzie pracy pozwalającym ukończyć tę pracę w terminie jej obrony. Autor zakłada, iż poprawie ulegnie bezpieczeństwo danych przechowy-

wanych w bazie danych poprzez ukrycie bazy danych przed dostępem z zewnątrz przy wykorzystaniu wirtualnej sieci prywatnej, także większa granularność aplikacji poprawi bezpieczeństwo wdrożenia zmian poprzez ustystematyzowanie procesu jaki za tym stoi poprzez wykorzystanie technologii „Pipeline” (ciąg zautomatyzowanych procesów dostarczania nowego oprogramowania). Również autor odświeżył wersję zależności z jakich składa się projekt, tak by rozwiązać problemy luk bezpieczeństwa wynikających z użycia przestarzałych bibliotek. Dodatkowym poziomem ulepszenia zabezpieczeń aplikacji będzie przeszukanie kodu w pod kątem znalezienia luk logicznych lub twardo zakodowanych dostępów, takich jak hasła do bazy danych lub inne, z których aplikacja korzysta.

2. EWALUACJA ZASTANEJ ARCHITEKTURY APLIKACJI

Zgodnie z stanem aplikacji z dnia 11.06.2022r. projekt “System do internetowego wspomagania pacjenta i lekarza” oparty jest o kilka składowych jakie należy wyróżnić, a są nimi:

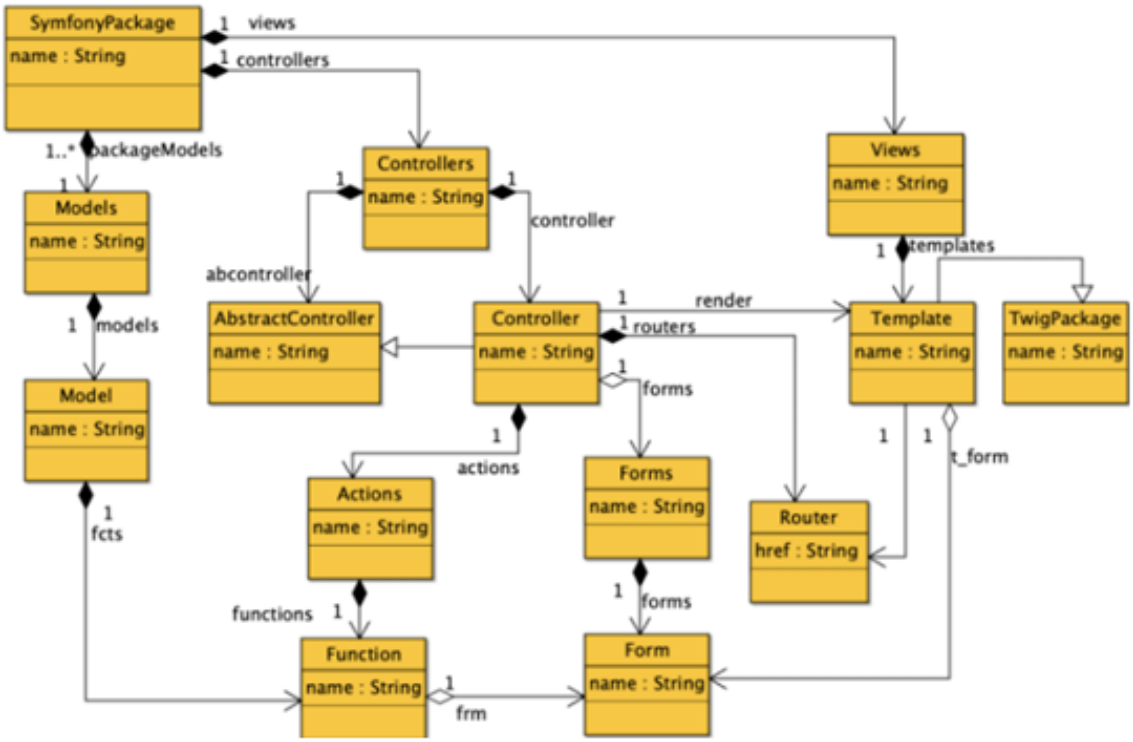
- baza danych oprata o silnik MySQL,
- trzon logiki biznesowej oparty o zestaw narzędzi Symfony w wersji 5.4,
- warstwa widoku aplikacji oparta o silnik Twig oraz Vue.js,
- silnik nginx jako narzędzie do przekazania zapytania klienckiego do logiki biznesowej.

Wyróżnione komponenty są ściśle ze sobą połączone. Logika biznesowa oraz baza danych są nierozerwalne, logika biznesowa posiada klasy encji, które bezpośrednio przekładają się na tabele w bazie danych. Logika biznesowa również decyduje o tym jaki widok jest obecnie wyświetlany, a warstwa widoku bazuje na tym, co dostarczył kontroler Symfony. Cały zestaw wykorzystuje narzędzie nginx do tego, by otrzymać zapytanie od klienta. W niniejszym rodziale zostaną przedstawione kolejno wyżej wymienione komponenty z szerszym opisem co do każdego z nich.

2.1 Logika biznesowa i baza danych

Symfony jako framework, który za zadanie ma ułatwić rozwój oprogramowania internetowego w projekcie z pracy inżynierskiej autora niniejszej pracy spełnił swoją rolę niejako przyczyniając się do skrócenia czasu potrzebnego na rzecz implementacji założeń diagramów użyć, odwzwierciedlenie domen każdego z aktorów systemu, a także diagramów przepływu danych. Narzędzia takie jak wbudowane kontrolery z dostępnym API do zapytań, czy formularze do walidacji danych wejściowych do systemu, router odpowiadający na zapytania, czy szablony wspierane przez silnik Twig to tylko niektóre z narzędzi jakie zostały wykorzystane przy okazji implementacji. Wyczerpujący schemat udogodnień oferowanych przez Symfony znajduje się na Rysunku 2.1.

W celu odzwierciedlenia domen aktorów, w projekcie została wykorzystana biblioteka Doctrine, wspierana przez Symfony. Samo Doctrine to narzędzie do manipulacji bazą danych przy wykorzystaniu obiektów PHP [3]. By zainicjować dane w systemie bez



Rys. 2.1. Schemat metamodelu Symfony [2]

konieczności manualnego wprowadzania ich autor wykorzystał dedykowaną ku temu bibliotekę wykorzystującą mechanizm fabryki danych testowych (ang. Fixtures Factory). Sam mechanizm fabryki danych testowych w procesie tworzenia oprogramowania umożliwia uruchomienie systemu z wypełnioną bazą danych, tak by móc przeprowadzać wymagane akcje i sprawdzić poprawność logiki biznesowej [4].

3. PLANOWANIE ARCHITEKTURY MIKROSERWISOWEJ W CHMURZE AWS

Niniejszy rozdział pokrywa zagadnienie planowania architektury mikroserwisowej oraz wykorzystanych usług potrzebnych do realizacji celu uruchomienia platformy. Nie istnieje mapa działań jakie należy wykonać by zgodnie z sztuką podzielić aplikację monolityczną na zbiór mikroserwisów. Istnieją natomiast wskazówki, którymi można się posługiwać by ułatwić doprowadzenie tego przedsięwzięcia do końca, a są nimi:

- identyfikacja mikroserwisów,
- dbanie o szczególną troskę w procesie ekstrakcji modułów, które są kandydatami na mikroserwisy,
- wprowadzenie modelu heksagonalnego aplikacji. [5]

3.1 Wybór odpowiednich usług do utrzymania aplikacji

W wyborze usług potrzebnych do użycia dla architektury mikroserwisowej, autor korzystał z własnych doświadczeń w pracy z platformą AWS. W związku z powyższym powody wyboru konkretnej usługi zostaną przedstawione w podrozdziale poświęconym na rzecz opisania motywacji doboru.

3.1.1 Amazon VPC - Virtual Private Cloud (ang. prywatna, wirtualna chmura)

Powody wyboru VPC dla celu realizacji architektury sieciowej:

- możliwość tworzenia wyizolowanych sieci w chmurze, co zapewnia wysoki standard bezpieczeństwa dla systemu, pozwala kontrolować ruch sieciowy przy pomocy reguł zapory sieciowej,
- możliwość kontrolowania adresacji IP, podsieci, tabel routingu i bram internetowych, co pozwala na dostosowanie sieci do indywidualnych potrzeb budowanego przez autora systemu. [6]

3.1.2 Amazon ECS - Elastic Container Service (ang. elastyczny serwis kontenerów)

Wybór ECS jako usługi pozwalającej na uruchomienie mikroserwisów jest uargumentowany niniejszymi powodami:

- ułatwione zarządzanie kontenerami poprzez wbudowane narzędzia do orkiestracji nimi,
- możliwość uruchomienia kontenerów bez konieczności zarządzania architekturą serwerową. [7]

3.1.3 Amazon RDS - Relational Database Service (ang. zarządzany serwis relacyjnych baz danych)

RDS jest idealnym wyborem do zarządzania bazami danych z następujących powodów:

- automatyzuje zadania typu tworzenie kopii zapasowych, aktualizacje oprogramowania oraz monitorowanie,
- posiada rozwiązania gwarantujące wysoką dostępność,
- oferuje szyfrowanie danych w spoczynku i w trakcie przesyłania oraz przy wykorzystaniu VPC do wyizolowania usługi zapewnia wysoki poziom bezpieczeństwa. [8]

3.1.4 Amazon MQ - Amazon Managed Message Broker Service (ang. zarządzany broker komunikatów dostarczany przez Amazon)

Z racji wykorzystania protokołu AMQP w projekcie migracji systemu, Amazon MQ jest odpowowiednim wyborem do zarządzania komunikacją między mikroserwisami z następujących powodów:

- umożliwia zautomatyzowanie konfiguracji, skalowania i zarządzania infrastrukturą brokerską,
- obsługuje popularne protokoły wiadomości, takie jak: AMQP, MQTT oraz STOMP, najbardziej kluczowym dla aplikacji systemu do internetowego wspomagania pacjenta i lekarza jest AMQP,
- posiada wbudowane mechanizmy redundancji i automatycznego przełączania awaryjnego,
- oferuje szyfrowanie danych oraz izolację środowiska w VPC. [9]

3.1.5 Amazon IAM - Identity and Access Management (ang. zarządzanie tożsamością i dostępem)

Najważniejszymi powodami do wyboru IAM jako usługi zarządzającej dostępem są:

- precyzyjne zarządzanie dostępem do wszystkich zasobów AWS,
- umożliwienie definiowania szczegółowych polityk uprawnień dla użytkowników, grup i ról,
- zapewnia wysoki poziom bezpieczeństwa poprzez funkcje takie jak dwuskładnikowe uwierzytelnianie (MFA), tymczasowe poświadczenia oraz możliwość monitorowania działań użytkowników,
- IAM udostępnia centralne zarządzanie tożsamościami i uprawnieniami, przez co upraszcza administrację i dodaje przejrzystości zarządzania dostępem. [10]

3.1.6 Amazon S3 - Simple Storage Service (ang. prosty magazyn danych)

Przechowywanie danych aplikacji to kluczowe zadanie, jakie zostało postawione usłudze S3, a jej wybór został podyktowany następującymi powodami:

- usługa ta automatycznie skaluje się w górę lub dół, przez co umożliwia praktycznie nieograniczone ilościowo składowanie danych,
- oferuje wysoką trwałość danych na poziomie bliskim 100%, a także wysoką dostępność,
- zapewnia szyfrowanie danych, zarządzanie dostępem na poziomie obiektu oraz integrację z AWS IAM,
- umożliwia optymalizację kosztów na podstawie częstotliwości dostępu do danych i wymagań dotyczących trwałości. [11]

3.1.7 Amazon CloudWatch

Usługa o nazwie CloudWatch, w szerokim spektrum usług jakie oferuje platforma AWS, dedykowana jest monitorowaniu i logowaniu akcji wykonanych w samych usługach, jak i w kodzie aplikacji. To są powody jakie zostały uwzględnione przy podjęciu decyzji o skorzystaniu z niej:

- umożliwia centralne zarządzanie i analizę danych,
- oferuje zaawansowane możliwości monitorowania aplikacji i infrastruktury, a także konfigurację alarmów, które powiadamiają o problemach w czasie rzeczywistym,

- na podstawie logów zbieranych przez CloudWatch, można zautomatyzować uruchamianie akcji skalowania zasobów, czy też funkcji Lambda. [12]

3.1.8 Amazon Route 53

Jako, iż system modernizowany przez autora niniejszej pracy dyplomowej jest aplikacją internetową, to należy zapewnić mu dostęp do domeny, która będzie propagowoać jego usługi na cały świat. Usługą, w portfolio AWS, która zapewnia takie rozwiązania jest Amazon Route 53. Poniżej przedstawione są najważniejsze powody, które zostały uwzględnione przy dokonaniu wyboru:

- wysoka dostępność i niezawodność gwarantowane przez rozproszenie usługi DNS,
- elastyczność i skalowalność,
- zintegrowane funkcje geolokalizacji,
- integracja z innymi, ważnymi z punktu widzenia systemu usługami, takimi jak: Elastic Load Balancing i CloudFront. [13]

3.1.9 Amazon Elastic Load Balancing

Oferowana przez firmę Amazon usługa Elastic Load Balancing (ang. elastyczne balansowanie obciążeniem) jest kluczowym elementem każdej aplikacji z uwagi na rozproszenie obciążenia systemu. Niżej przedstawiono wszystkie powody wykorzystania omawianej usługi:

- automatycznie rozkładanie ruchu sieciowego między wieloma zasobami,
- poprawa dostępności aplikacji poprzez monitorowanie stanu zasobów i automatyczne przekierowanie ruchu do zdrowych instancji,
- zarządzanie ruchem na podstawie zawartości, geolokalizacji i opóźnienia między żądanym zasobem,
- integracja z AWS Certificate Manager do zarządzania certyfikatami SSL/TLS, a także ochrona przez atakami DDoS i zarządzanie ruchem HTTP/2,
- integracja z usługą Auto Scaling (ang. automatyczne skalowanie) oraz CloudWatch. [14]

3.1.10 ACM - AWS Certificate Manager (ang. menadżer certyfikatów AWS)

Narzędzie ACM to natywne rozwiązanie chmury AWS oferujące szereg udogodnień istotnych z punktu widzenia cyberbezpieczeństwa, a są nimi:

- automatyzacja procesu tworzenia, wdrażania i odnawiania SSL/TLS,
- integracja z Elastic Load Balancing (ELB), Amazon CloudFront i API Gateway (ang. bramka API),
- bezpłatne utrzymanie certyfikatów SSL/TLS dla domen zarządzanych przez AWS,
- łatwość użycia poprzez uproszczenie procesu zarządzania certyfikatami dzięki intuicyjnemu interfejsowi. [15]

3.1.11 Amazon API Gateway (ang. Bramka API Amazon)

Usługa Amazon API Gateway została stworzona z myślą o maksymalnym uproszczeniu tworzenia struktury API i łączenia zasobów w jedną całość. Autor niniejszej pracy dyplomowej wykorzystał ją, po uprzednim zgłębieniu jej zalet, a są nimi:

- integracja z Amazon Elastic Load Balancing,
- zarządzanie ruchem poprzez mechanizm Throttiling Rules (ang. zasady tłumienia),
- łatwość tworzenia API i jego wdrożenia,
- monitoring operacji wykonywanych wewnątrz API. [16]

3.2 Opracowanie planu migracji

Migracja działającego systemu monolitycznego na architekturę mikroserwisów jest wyzwaniem. Wiąże się tym duży koszt wstępny jaki musi ponieść przedsiębiorstwo, by uruchomić szereg koniecznych procesów pozwalających przeprowadzić odpowiednio migrację. Samą migrację należy przeprowadzić w pełnej świadomości i po przeprowadzeniu uprzednio audytu opłacalności tego zabiegu. Zabieg ten wprowadza znaczne skomplikowanie do projektu i wymaga wykwalifikowanej kadry inżynierów, którzy będą w stanie zapewnić ciągłość działania aplikacji w fazach przejściowych, a przy okazji rozwijając nową funkcjonalność. W niniejszej pracy dyplomowej migracja zostanie przeprowadzona na systemie hobbystycznym, a co za tym idzie jego ewentualna przerwa w działaniu nie

zadziała destrukcyjnie na ewentualne przedsiębiorstwo, które mogłoby ponieść koszta niedziałającej platformy.

3.2.1 Domain-Driven Design

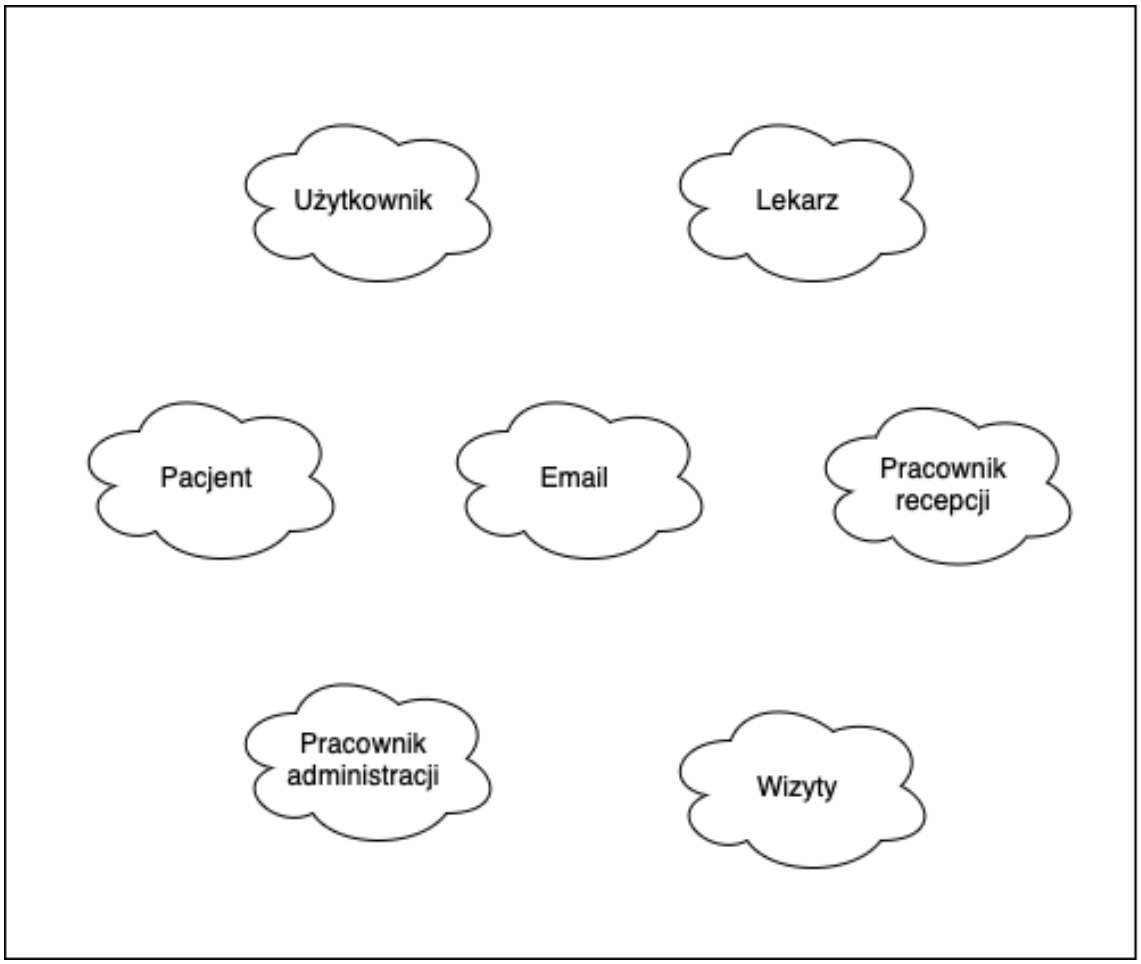
W skrócie DDD. Nazwa ta w języku polskim oznacza nacisk na prowadzenie rozwoju oprogramowania w kontekście jakiejś domeny. Jest to filozofia, która ma pomagać rozwiązywać problemy budowy oprogramowania dla skomplikowanych domen [17]. To podejście zakłada podział systemu na komponenty oraz ich zachowania, aby te wiernie odzwierciedlały rzeczywistość. Po wstępnej fazie planowania obszarów systemu następuje przejście do fazy implementacji.

Zadaniem autora w niniejszej pracy dyplomowej było wykorzystanie potencjału jaki został stworzony w jego pracy inżynierskiej. Mianowicie w omawianej pracy system został podzielony w naturalny sposób pośród czterech aktorów, jakimi są: pacjent, lekarz, pracownik recepcji oraz pracownik administracji. W tego podziału wyłania się domena podziału. Każdy aktor ma swój zestaw czynności jakie w swojej domenie może wykonywać. Tak na przykład każdy użytkownik ma prawa do edycji swojego konta, usunięcia go, oraz podglądu danych jakie się w nim znajdują. Przyglądając się funkcjonalności wizyt wyłania się osobna zależność, która jest możliwa do wydzielenia z całości systemu, a dostęp mają do niej aktorzy tacy jak: pacjent, lekarz i pracownik recepcji. Każdy z aktorów ma też swój indywidualny zestaw danych, który jest niezależny od innego z aktorów, co kwalifikuje ten fakt na uwzględnienie go jako podział na cztery dodatkowe domeny. Osobną zależnością jest również sama wysyłka maili, która z punktu widzenia systemu jest istotna, ale bardzo mała porównując zakres czynności jakie wykonuje.

Dalsze działania wynikające z planowania będą bazować bezpośrednio na wytyczonym powyżej podziale. Sam podział nie jest nowy w kontekście całego systemu. Autor zgodnie z sztuką zbudował projekt, tak by w przyszłości móc go rozwinąć właśnie poprzez modyfikację w kierunku mikroserwisów.

3.2.2 Zakres migracji

Według autora książki “Monolith to Microservices“ należy zrozumieć w jakim zakresie szczegółowa ma być domena, którą próbuje się dzielić na mniejsze części, tak by było to rozsądne [18]. Patrz Rysunek 3.1.



Rys. 3.1. Zakres domenowy systemu do internetowego wspomagania pacjenta i lekarza

3.2.3 Podejście Event Storming

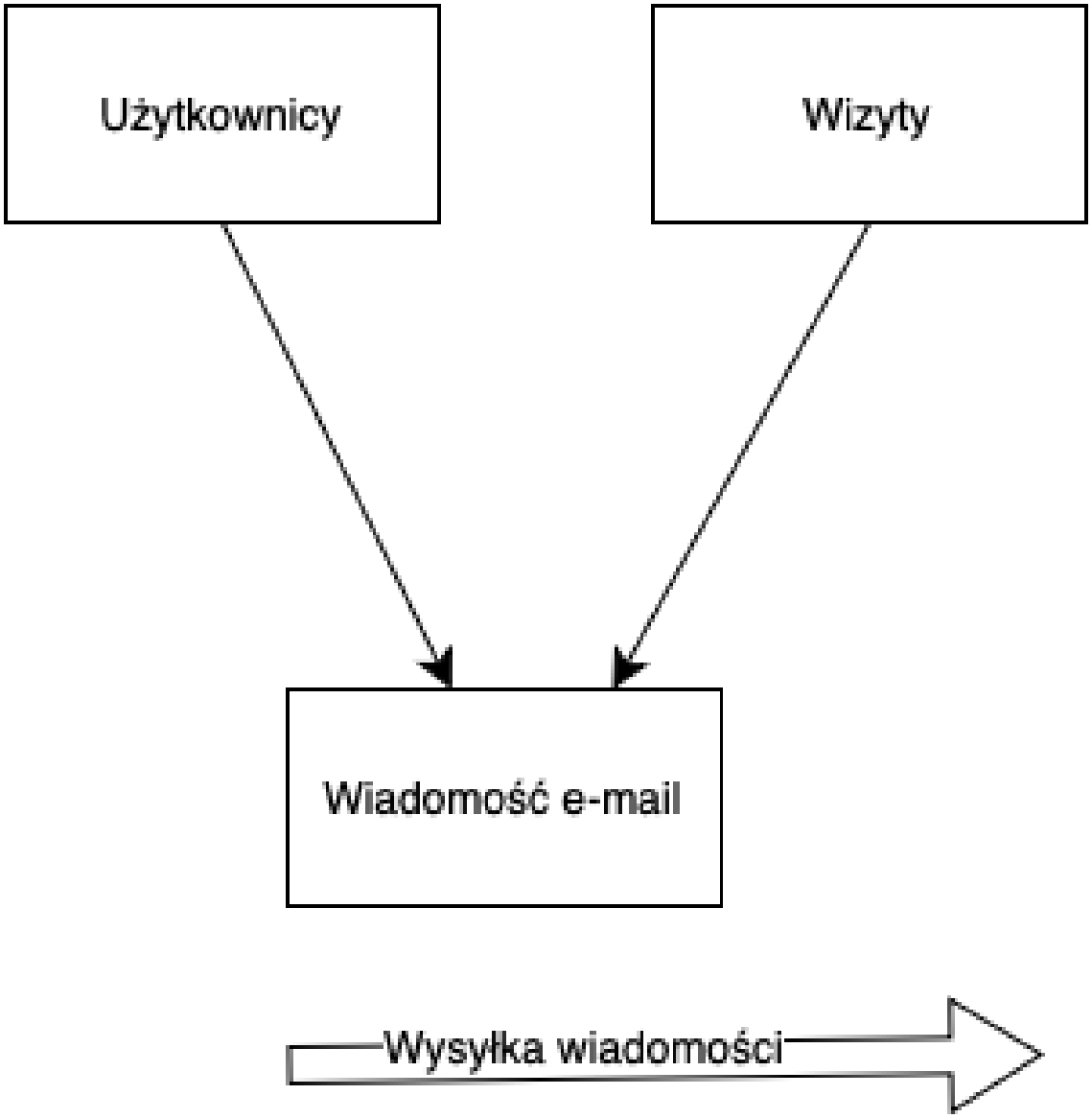
Podejście to polega na zgrupowaniu potencjalnych domen systemu za pośrednictwem akcji jakie wykonują. Tak na przykład w systemie do internetowego wspomagania pacjenta i lekarza da się pogrupować czynności wykonywane w systemie pod kątem wysyłki wiadomości e-mail. Na (Wstaw referencję obrazka) widoczny jest schemat pokazujący, że poszczególne domeny, takie jak: użytkownicy, wizyty są zgrupowane do akcji wysyłania wiadomości drogą mailową. Całość potencjału omawianego podejścia została przedstawiona na Rysunku 3.2.

3.2.4 Przebieg migracji

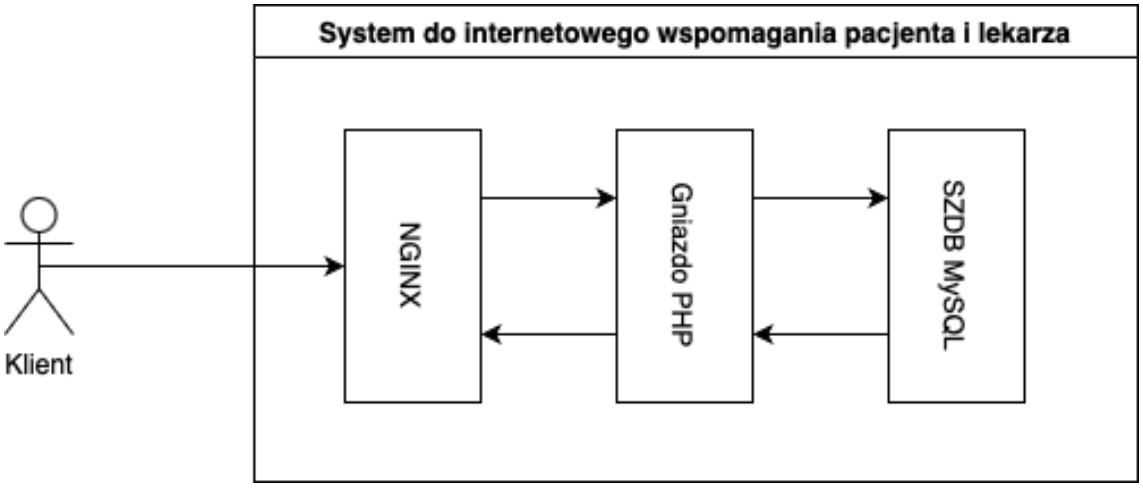
Po fazie planowania należy przejść do fazy, w której mając schemat podzielona zostanie całkowita migracja na fazy, tak by utrzymać ciągłość działania systemu. System uprzednio uruchomiony został w zwirtualizowanym środowisku Vagrant, gdzie miał zain-

stalowanie, które było proxy (ang. pośrednikiem) zapytań - nginx. W tym samym środowisku uruchomiony był również system do zarządzania relacyjną bazą danych - MySQL. Dodatkowo uruchomiona była tam również logika biznesowa przy pomocy gniazda PHP. Schematyczny obraz fazy pierwszej znajduje się na Rysunku 3.3.

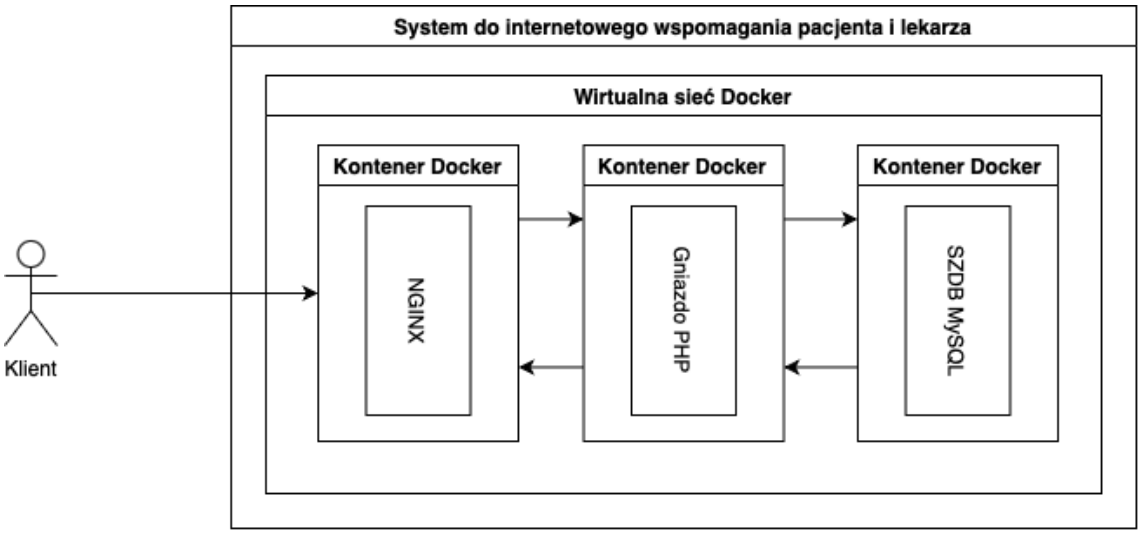
Wspieranym oprogramowaniem wirtualizującym na środowisku lokalnym i w samym AWS jest Docker. W drugiej fazie migracji wykonano opisanie bazy danych, pośrednika zapytań nginx i logiki biznesowej w osobnych, niezależnych kontenerach, skomunikowanych ze sobą wirtualną siecią. Omawiana zmiana widoczna na Rysunku 3.4.



Rys. 3.2. Wysyłka wiadomości jest logicznie połączona dla tego modelu domeny, więc jej dalsza ekstrakcja może być trudna



Rys. 3.3. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w pierwszej fazie



Rys. 3.4. Schemat architektury systemu do internetowego wspomagania pacjenta i lekarza w drugiej fazie

Bibliografia

[1] “Cloud computing market,” 2023, data dostępu: 20 Styczeń 2024.

[2] M. Rahmouni, C. Talbi, and S. Ziti, “Model-driven architecture; generating models from symfony,” 2023, data dostępu: 6 Kwiecień 2024.

[3] F. Potencier, *Symfony 5: The Fast Track*, 2020, data dostępu: 6 Kwiecień 2024.

[4] L. da Silva and P. Vilain, “Execution and code reuse between test classes.” Los Alamitos, CA, USA: IEEE Computer Society, Czerwiec 2016, pp. 99–106, data dostępu: 11 Kwiecień 2024. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SERA.2016.7516134

[5] R. Rocha, P. A. Simoes, and J. C. Purificação, *Java EE 8 Design Patterns and Best Practices*, 2018, data dostępu: 1 Sierpień 2024.

[6] “What is amazon vpc? - amazon virtual private cloud,” https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html, data dostępu: 1 Sierpień 2024.

[7] “What is amazon elastic container service? - amazon elastic container service,” https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html#welcome-terminology, data dostępu: 1 Sierpień 2024.

[8] “What is amazon relational database service (amazon rds)? - amazon relational database service,” https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html, data dostępu: 1 Sierpień 2024.

[9] “Amazon mq features | managed message broker service | amazon web services,” https://aws.amazon.com/amazon-mq/features/, data dostępu: 1 Sierpień 2024.

[10] “Amazon mq features | managed message broker service | amazon web services,” https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html, data dostępu: 1 Sierpień 2024.

[11] “What is amazon s3? - amazon simple storage service,” https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html, data dostępu: 1 Sierpień 2024.

[12] “Apm tool - amazon cloudwatch - aws,” https://aws.amazon.com/cloudwatch/, data dostępu: 1 Sierpień 2024.

[13] “Welcome - amazon route 53,” https://docs.aws.amazon.com/Route53/latest/APIReference/Welcome.html, data dostępu: 3 Sierpień 2024.

[14] “Welcome - elastic load balancing,” https://docs.aws.amazon.com/elasticloadbalancing/latest/APIReference/Welcome.html, data dostępu: 3 Sierpień 2024.

[15] “Aws certificate manager features - amazon web services (aws),” https://aws.amazon.com/certificate-manager/features/?nc=sn&loc=2, data dostępu: 3 Sierpień 2024.

[16] “Amazon api gateway features | api management | amazon web services,” https://aws.amazon.com/api-gateway/features/, data dostępu: 3 Sierpień 2024.

[17] S. Millett and N. Tune, *Patterns, Principles, and Practices of Domain-Driven Design*, 2015, data dostępu: 11 Sierpień 2024.

[18] S. Newman, *Monolith to Microservices*, 2019, data dostępu: 11 Sierpień 2024.

Spis rysunków