O'REILLY®

# Mikrousługi oparte na zdarzeniach

## Wykorzystywanie danych w organizacji na dużą skalę

Helion

Adam Bellemare

Tytuł oryginału: Building Event-Driven Microservices: Leveraging Organizational Data at Scale

Tłumaczenie: Lech Lachowski

ISBN: 978-83-283-7439-3



Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
*http://helion.pl/user/opinie/miopzd*
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

# Spis treści

ROZDZIAŁ 1.

# Dlaczego mikrousługi oparte na zdarzeniach?

*Przekazem jest medium*
— Marshall McLuhan

McLuhan twierdzi, że to nie *zawartość* mediów, ale raczej zaangażowanie w ich środek przekazu ma wpływ na ludzkość i wprowadza fundamentalne zmiany w społeczeństwie. Gazety, radio, telewizja, internet, komunikatory i media społecznościowe zmieniły interakcje międzyludzkie i struktury społeczne dzięki naszemu zbiorowemu zaangażowaniu.

To samo dotyczy architektur systemów komputerowych. Wystarczy przyjrzeć się historii wynalazków z dziedziny informatyki, aby zauważyć, że komunikacja sieciowa, relacyjne bazy danych, rozwiązania Big Data i chmura obliczeniowa znacząco zmieniły sposób budowania architektur i wykonywania pracy. Każdy z tych wynalazków zmienił nie tylko metody wykorzystania technologii w różnych projektach oprogramowania, ale także sposób komunikowania się między sobą organizacji, zespołów i ludzi. Od scentralizowanych komputerów typu mainframe po rozproszone aplikacje mobilne — każde nowe medium fundamentalnie zmieniło stosunek ludzi do informatyki.

Nowoczesna technologia w zasadniczy sposób zmieniła medium asynchronicznie generowanych i konsumowanych zdarzeń. Zdarzenia mogą być teraz zachowywane w nieskończoność na bardzo dużą skalę i wykorzystywane przez dowolną usługę niezbędną liczbę razy. Zasoby obliczeniowe można łatwo pozyskiwać i zwalniać na żądanie, co umożliwia bezproblemowe tworzenie mikrousług i zarządzanie nimi. Mikrousługi mogą przechowywać swoje dane i zarządzać nimi według własnych potrzeb, i to na skalę, która wcześniej była nieosiągalna z uwagi na ograniczenia rozwiązań przetwarzania wsadowego Big Data. Te udoskonalenia skromnego i prostego medium opartego na zdarzeniach mają daleko idące skutki, które nie tylko spowodowały zmiany w architekturach komputerów, ale także całkowicie zmieniły sposób tworzenia systemów i prowadzenia działalności gospodarczej przez organizacje, zespoły i poszczególne jednostki.

## Czym są mikrousługi oparte na zdarzeniach?

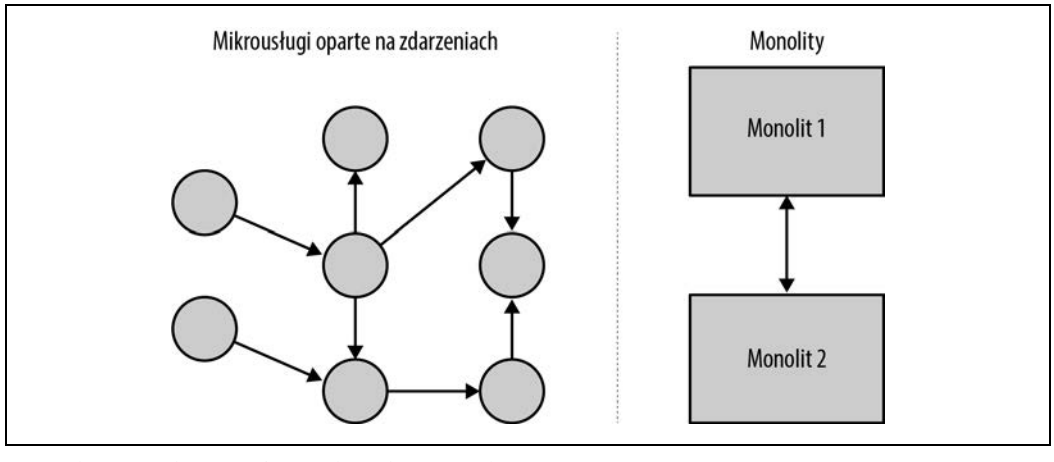
Mikrousługi i architektury mikrousługowe istnieją od lat w mnogich formach, pod wieloma odmiennymi nazwami. Architektury usługowe (ang. *service-oriented architecture* — SOA) często składają się z wielu mikrousług, które komunikują się synchronicznie bezpośrednio ze sobą. Architektury opierające się na przekazywaniu komunikatów wykorzystują konsumowalne zdarzenia do asynchro-

nicznej komunikacji między systemami. Komunikacja oparta na zdarzeniach z pewnością nie jest nowością, ale potrzeba obsługi rozległych zbiorów danych (Big Data) na dużą skalę i w czasie rzeczywistym **jest** nowa i wymusza odejście od starych stylów architektonicznych.

W nowoczesnej architekturze mikrousług opartych na zdarzeniach systemy komunikują się poprzez wysyłanie i konsumowanie zdarzeń. Zdarzenia te nie są niszczone podczas konsumpcji — jak ma to miejsce w systemach opartych na przekazywaniu komunikatów — tylko pozostają łatwo dostępne do odczytu przez innych konsumentów, gdy zachodzi taka potrzeba. Jest to ważne rozróżnienie, ponieważ umożliwia stosowanie naprawdę wszechstronnych wzorców, które zostaną omówione w tej książce.

Same usługi są niewielkie i mają określone przeznaczenie — mają dopomóc w realizacji niezbędnych celów biznesowych organizacji. Zwykle „niewielkie" usługi to z definicji takie, których napisanie zajmuje nie więcej niż dwa tygodnie. Inna definicja stwierdza, że usługa powinna być w stanie (koncepcyjnie) „zmieścić się w głowie" programisty. Te usługi konsumują zdarzenia z wejściowych strumieni zdarzeń, stosują do nich określoną logikę biznesową i mogą emitować własne zdarzenia wyjściowe, dostarczać dane w celu zapewnienia dostępu w schemacie żądanie-odpowiedź, komunikować się z zewnętrznym interfejsem API lub wykonywać inne wymagane czynności. Jak przekonasz się w trakcie dalszej lektury, usługi te mogą być stanowe lub bezstanowe, złożone lub proste, i mogą być implementowane jako długo działające, samodzielne aplikacje, lub wykonywane jako funkcje przy użyciu rozwiązania FaaS (ang. *Functions-as-a-Service*) — funkcja jako usługa.

Ta kombinacja strumieni zdarzeń i mikrousług formuje graf spójny aktywności w organizacji biznesowej. Podobną strukturę grafową mają tradycyjne architektury komputerowe, złożone z monolitów i komunikacji międzymonolitycznej. Oba te grafy zostały pokazane na rysunku 1.1.




*Rysunek 1.1. Struktury grafowe mikrousług i monolitów*

Określenie, w jaki sposób sprawić, żeby taka struktura grafu zapewniała wydajne działała, wymaga przyjrzenia się dwóm głównym komponentom: węzłom i połączeniom. W tym rozdziale omówimy po kolei oba te komponenty.

# Wprowadzenie do projektowania dziedzinowego i kontekstów ograniczonych

Pojęcie projektowania dziedzinowego (ang. *Domain-Driven Design* — DDD) — ukute przez Erica Evansa w książce *Domain-Driven Design. Zapanuj nad złożonym systemem informatycznym* (wyd. Helion) — wprowadza kilka koncepcji niezbędnych do tworzenia mikrousług opartych na zdarzeniach. Biorąc pod uwagę bogactwo łatwo dostępnych poświęconych temu tematowi artykułów (*https://oreil.ly/zAXqd*), książek (*DDD dla architektów oprogramowania* Vaughna Vernona, wyd. Helion) i wyróżnionych blogów, opiszę te kwestie w skrócie.

Podstawę projektowania dziedzinowego stanowią następujące koncepcje:

**Dziedzina**

Przestrzeń problemów, którymi zajmuje się dana firma, dostarczająca dla nich rozwiązania. Obejmuje to wszelakie kwestie, z jakimi firma ma na co dzień do czynienia, w tym reguły, procesy, pomysły, charakterystyczną terminologię biznesową i wszystko, co jest związane z tą przestrzenią problemów, *niezależnie od tego, czy leży to w gestii zainteresowań danej firmy*. Dziedzina istnieje niezależnie od istnienia firmy.

**Poddziedzina**

Komponent głównej dziedziny. Każda poddziedzina koncentruje się na określonym podzbiorze obowiązków i zazwyczaj odzwierciedla część struktury organizacyjnej firmy (np. magazyn, dział sprzedaży czy dział inżynierii). Poddziedzina może być sama w sobie postrzegana jako dziedzina. Poddziedziny, podobnie jak sama dziedzina, należą do przestrzeni problemów.

**Model dziedziny (i poddziedziny)**

Abstrakcja rzeczywistej dziedziny przydatna do celów biznesowych. Do wygenerowania tego modelu wykorzystuje się elementy i właściwości dziedziny, które są najważniejsze dla danej działalności gospodarczej. Główny model dziedziny firmy można rozpoznać poprzez produkty, które firma dostarcza swoim klientom, interfejsy, za pomocą których klienci wchodzą w interakcję z produktami, oraz różne inne procesy i funkcje, dzięki którym firma realizuje swoje określone cele. Modele często wymagają dopracowywania w miarę zmieniania się dziedziny i priorytetów biznesowych. Model dziedziny jest częścią przestrzeni rozwiązań, ponieważ jest to konstrukcja używana przez firmę do rozwiązywania problemów.

**Kontekst ograniczony**

Granice logiczne, w tym dane wejściowe, dane wyjściowe, zdarzenia, wymagania, procesy i modele danych istotne dla poddziedziny. Chociaż najlepiej byłoby, gdyby ograniczony kontekst i poddziedzina całkowicie się pokrywały, często mamy do czynienia z wyjątkami tworzonymi przez starsze systemy, zapóźnienie technologiczne oraz integracje z elementami zewnętrznymi. Konteksty ograniczone są również właściwością przestrzeni rozwiązań i mają znaczący wpływ na wzajemne interakcje mikrousług.

Konteksty ograniczone powinny charakteryzować się dużą spójnością. Wewnętrzne operacje kontekstu powinny być intensywne i ściśle ze sobą powiązane, przy czym zdecydowana większość

komunikacji odbywa się raczej wewnętrznie niż transgranicznie. Posiadanie bardzo spójnych obowiązków pozwala na ograniczenie zakresu projektowego i tworzenie prostszych implementacji.

Połączenia między kontekstami ograniczonymi powinny być luźno powiązane, ponieważ zmiany dokonywane w jednym kontekście ograniczonym powinny minimalizować lub eliminować wpływ na sąsiednie konteksty. Luźne powiązania mogą zapewnić, że zmiany wymagań w jednym kontekście nie spowodują propagowania fali zależnych zmian do sąsiednich kontekstów.

## Wykorzystywanie modeli dziedzin i kontekstów ograniczonych

Każda organizacja tworzy pojedynczą dziedzinę wydzielającą ją ze świata zewnętrznego. Wszyscy pracownicy danej organizacji działają na rzecz potrzeb tej dziedziny.

Dziedzina jest podzielona na poddziedziny — w przypadku przedsiębiorstwa technologicznego mogą to być na przykład dział inżynierii, dział sprzedaży i dział obsługi klienta. Każda poddziedzina ma swoje własne wymagania i obowiązki, i sama może być podzielona. Ten proces podziału powtarza się, dopóki modele poddziedzin nie będą wystarczająco szczegółowe, zapewniające podstawy do działania i możliwe do przełożenia przez zespoły implementujące na małe i niezależne usługi. Wokół tych poddziedzin są ustanawiane konteksty ograniczone, które stanowią podstawę do tworzenia mikrousług.

## Dopasowywanie kontekstów ograniczonych do wymagań biznesowych

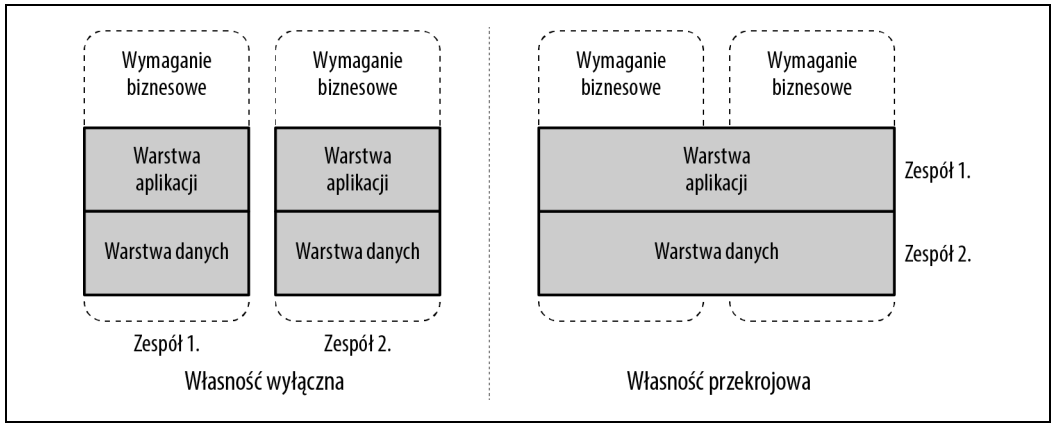
Wymagania biznesowe produktu często zmieniają się w trakcie jego życia, na przykład z powodu zmian organizacyjnych lub zapotrzebowania na nowe funkcjonalności. Natomiast rzadko zdarza się, aby firma musiała zmieniać bazową implementację jakiegokolwiek produktu bez towarzyszących temu procesowi zmian wymagań biznesowych. Dlatego konteksty ograniczone należy budować wokół wymagań biznesowych, a nie technologicznych.

Dopasowanie kontekstów ograniczonych do wymagań biznesowych umożliwia zespołom wprowadzanie zmian w implementacjach mikrousług w luźno powiązany i wysoce spójny sposób. Zapewnia zespołowi autonomię w zakresie projektowania i implementowania rozwiązań określonych potrzeb biznesowych, co znacznie ogranicza zależności między zespołami i umożliwia każdemu z nich skupienie się wyłącznie na własnych wymaganiach.

Z kolei dopasowanie mikrousług do wymagań technologicznych jest problematyczne. Ten wzorzec jest często spotykany w nieprawidłowo zaprojektowanych synchronicznych mikrousługach typu punkt-punkt oraz w tradycyjnych monolitycznych systemach obliczeniowych, w których do zespołów przypisane są określone warstwy techniczne aplikacji. Głównym problemem związanym z dopasowaniem technologicznym jest rozłożenie odpowiedzialności za realizowanie funkcji biznesowych na wiele kontekstów ograniczonych, które mogą obejmować wiele zespołów z różnymi harmonogramami i obowiązkami. Ponieważ żaden zespół nie jest wyłącznie odpowiedzialny za implementację rozwiązania, poszczególne usługi stają się ze sobą powiązane zarówno poprzez granice zespołów, jak i interfejsów API, co sprawia, że wprowadzanie zmian staje się trudne i kosztowne. Pozornie niewinna zmiana, jakiś błąd lub awaria pojedynczej usługi mogą powodować efekt domina, który będzie miał poważny wpływ na możliwości obsługi biznesowej wszystkich usług korzystających z danego systemu technicznego. W architekturach mikrousług opartych na zdarzeniach (ang. *Event-Driven*

*Microservice* — EDM) dopasowanie technologiczne jest rzadko stosowane i należy go unikać całkowicie, gdy tylko jest to możliwe. Wyeliminowanie przekrojowych zależności technologicznych i zespołowych zmniejsza wrażliwość systemu na zmiany.

Na rysunku 1.2 przedstawiono oba scenariusze: wyłączną własność po lewej stronie i przekrojową własność po prawej. W przypadku wyłącznej własności zespół jest w pełni zorganizowany wokół dwóch niezależnych wymagań biznesowych (kontekstów ograniczonych) i ma pełną kontrolę nad kodem aplikacji i warstwą bazy danych. Po prawej stronie zespoły zostały zorganizowane według wymagań technologicznych, w których warstwa aplikacji jest zarządzana oddzielnie od warstwy danych. Tworzy to bezpośrednie zależności między zespołami oraz domyślne zależności między wymaganiami biznesowymi.




*Rysunek 1.2. Porównanie dostosowania do kontekstów biznesowych i dostosowania do kontekstów technologicznych*

Jeśli chodzi o architektury mikrousług opartych na zdarzeniach, to preferowane jest ich modelowanie wokół wymagań biznesowych, chociaż takie podejście wiąże się z pewnymi kompromisami. Kod może być wielokrotnie replikowany, a duża część usług może używać podobnych wzorców dostępu do danych. Twórcy produktów mogą próbować ograniczyć replikowanie poprzez współdzielenie źródeł danych z innymi produktami lub przez połączenie granic. W takich przypadkach tworzące się ścisłe powiązania mogą być na dłuższą metę dużo bardziej kosztowne niż powtarzanie logiki i przechowywanie podobnych danych. Będę omawiał te kompromisy szczegółowo w dalszych częściach książki.

Utrzymuj luźne powiązania między kontekstami ograniczonymi i koncentruj się na minimalizowaniu zależności międzykontekstowych. W razie konieczności umożliwi to zmianę implementacji kontekstu ograniczonego bez naruszania w następstwie wielu (lub jakichkolwiek) innych systemów.

Ponadto od każdego zespołu może być wymagana pełna wiedza na temat stosu, co może powodować pewne komplikacje ze względu na konieczność posiadania specjalistycznych zestawów umiejętności i uprawnień dostępu. Organizacja powinna wprowadzić odpowiednie procedury dla tych najpowszechniejszych wymagań, aby zespoły na poszczególnych szczeblach mogły wspierać się nawzajem,

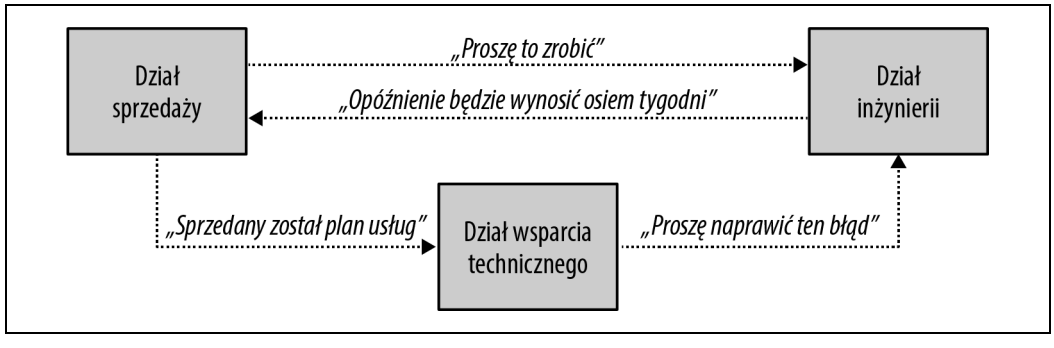
podczas gdy bardziej wyspecjalizowane zestawy umiejętności mają być zapewniane w ramach projektów międzyzespołowych, w zależności od potrzeb. Te najlepsze praktyki zostały opisane szczegółowo w rozdziale 14.

# Struktury komunikacji

Zespoły, systemy i ludzie w organizacji muszą komunikować się ze sobą, aby osiągać wyznaczone cele. Ta komunikacja tworzy połączoną topologię zależności zwaną **strukturą komunikacji**. Istnieją trzy główne struktury komunikacyjne, a każda z nich wpływa na sposób działania firm.

## Struktury komunikacji biznesowej

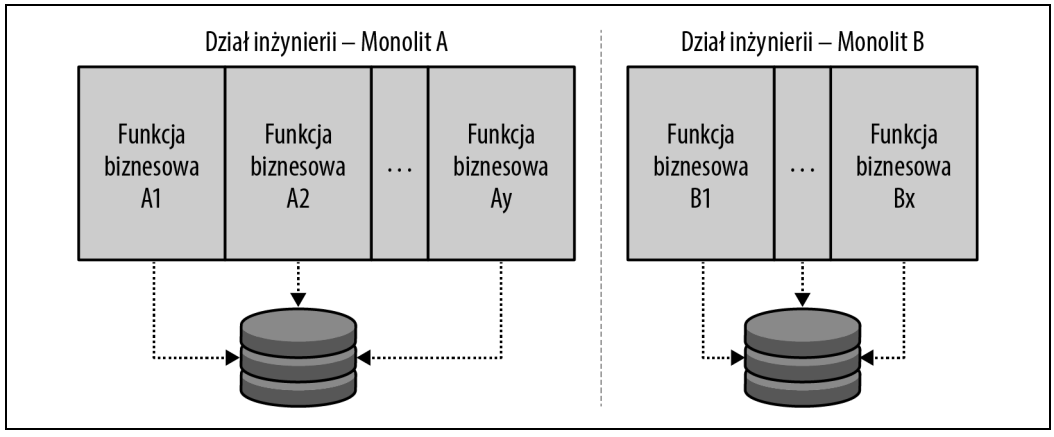
Struktura komunikacji biznesowej (zobacz rysunek 1.3) dyktuje komunikację między zespołami i działami firmy, z których każdy kieruje się głównymi wymaganiami i przypisanymi mu obowiązkami. Dział inżynierii produkuje na przykład oprogramowanie, dział sprzedaży zajmuje się sprzedawaniem produktu klientom, a dział wsparcia technicznego stara się zapewnić, aby klienci byli usatysfakcjonowani. Tej strukturze podlega organizacja zespołów i dostarczanie im celów, co dotyczy tak samo głównych jednostek biznesowych, jak i pracy indywidualnego pracownika. Wymagania biznesowe, ich przypisanie do zespołów i skład zespołów zmieniają się wraz z upływam czasu, co może mieć duży wpływ na relacje między strukturą komunikacji biznesowej a strukturą komunikacji implementacyjnej.




*Rysunek 1.3. Przykładowa struktura komunikacji biznesowej*

## Struktury komunikacji implementacyjnej

**Struktura komunikacji implementacyjnej** (zobacz rysunek 1.4) to dane i logika odnoszące się do modelu poddziedziny narzuconej przez organizację. Formalizuje procesy biznesowe, struktury danych i projektowanie systemów, aby operacje biznesowe mogły być przeprowadzane szybko i wydajnie. Wiąże się to z kompromisem dotyczącym elastyczności struktury komunikacji biznesowej, ponieważ przedefiniowanie wymagań biznesowych, które muszą zostać spełnione przez implementację, wymaga przepisania logiki. Te przepisywania logiki to najczęściej iteracyjne modyfikacje modelu poddziedziny i związanego z nim kodu, które z biegiem czasu odzwierciedlają ewolucję implementacji, mającą na celu spełnienie nowych wymagań biznesowych.




*Rysunek 1.4. Przykładowa struktura komunikacji implementacyjnej*

Typowym przykładem struktury komunikacji implementacyjnej dla inżynierii oprogramowania jest monolityczna aplikacja bazodanowa. Logika biznesowa aplikacji komunikuje się z nią wewnętrznie za pośrednictwem wywołań funkcji lub współdzielonego stanu. Z kolei ta monolityczna aplikacja jest wykorzystywana do spełniania wymagań biznesowych podyktowanych strukturą komunikacji biznesowej.

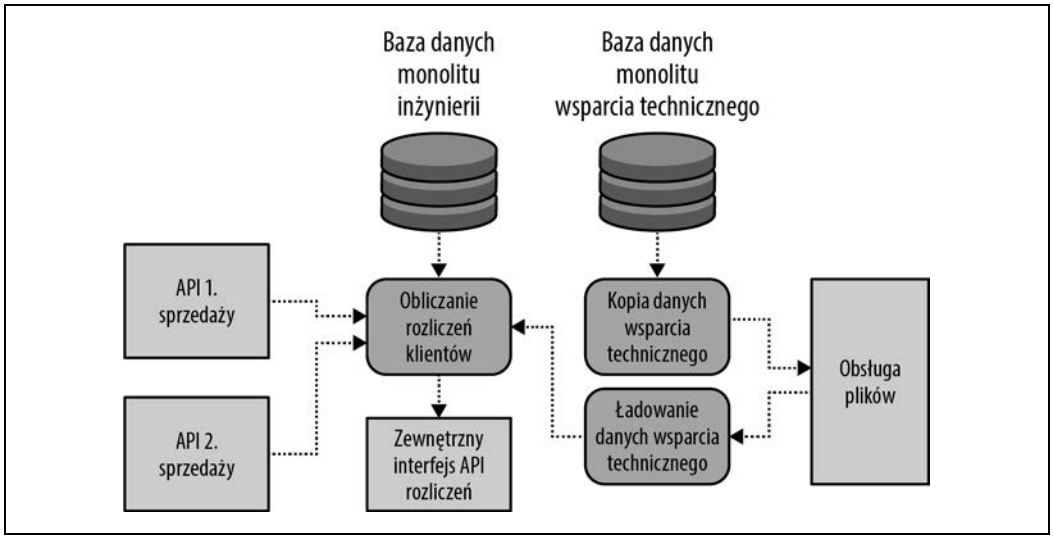
## Struktury komunikacji danych

Struktura komunikacji danych (zobacz rysunek 1.5) to proces, za pomocą którego dane są przekazywane w całej firmie, a zwłaszcza między implementacjami. Chociaż struktura komunikacji danych obejmująca pocztę elektroniczną, komunikatory i zebrania jest często używana do informowania o zmianach biznesowych, została w dużej mierze zaniedbana w zakresie implementacji oprogramowania. Jej zadania były zwykle realizowane ad hoc, od systemu do systemu, przy czym struktura komunikacji implementacyjnej często odgrywała podwójną rolę, realizując oprócz własnych wymagań również funkcje komunikacji danych. Powodowało to wiele problemów związanych ze sposobem, w jaki z biegiem czasu przedsiębiorstwa rozwijały się i zmieniały. Wpływ takiego podejścia omówimy w następnym punkcie rozdziału.

## Prawo Conwaya i struktury komunikacji

> *Organizacje projektujące systemy (…) są ograniczone do tworzenia projektów, które są kopiami struktur komunikacyjnych tych organizacji.*
>
> — Melvin Conway, *How Do Committees Invent?* (kwiecień 1968 r.)

Ten cytat, znany jako **prawo Conwaya**, implikuje, że zespół będzie budować produkty opierając się na strukturach komunikacji jego organizacji. Struktury komunikacji biznesowej organizują ludzi w zespoły, te zaś zazwyczaj wytwarzają produkty wyznaczone przez granice ich zespołów. Struktury komunikacji implementacyjnej zapewniają dostęp do modeli danych poddziedziny dla danego produktu, ale ograniczają także dostęp do innych produktów ze względu na słabe możliwości komunikacji danych.




*Rysunek 1.5. Przykładowa struktura komunikacji danych ad hoc*

Ponieważ koncepcje dziedzin obejmują całe przedsiębiorstwo, dane dziedzinowe są często wymagane przez inne konteksty ograniczone w organizacji. Struktury komunikacji implementacyjnej zasadniczo słabo radzą sobie z zapewnianiem tego mechanizmu komunikacji, chociaż doskonale zaspokajają potrzeby własnego kontekstu ograniczonego. Wpływają na projektowanie produktów na dwa sposoby. Po pierwsze, ze względu na nieefektywność przekazywania niezbędnych danych dziedzinowych w całej organizacji, zniechęcają do tworzenia nowych, logicznie odseparowanych produktów. Po drugie, zapewniają łatwy dostęp do istniejących danych dziedzinowych, co wiąże się z ryzykiem ciągłego rozszerzania dziedziny w celu uwzględniania nowych wymagań biznesowych. Ten konkretny wzorzec jest reprezentowany przez projekty monolityczne.

Struktury komunikacji danych odgrywają kluczową rolę w sposobie, w jaki organizacja projektuje i wytwarza produkty, ale w wielu organizacjach od dawna brakuje tej struktury. Jak stwierdziłem wcześniej, struktury komunikacji implementacyjne często pełnią tę rolę oprócz swojej własnej.

Niektóre organizacje próbują załagodzić problem związany z niemożliwością zapewnienia dostępu do danych dziedzinowych z innych implementacji, ale te działania mają swoje wady. Nierzadko używane są na przykład współdzielone bazy danych, chociaż z reguły takie rozwiązania promują antywzorce i często nie mogą być dostatecznie skalowane, aby spełnić wszystkie wymagania dotyczące wydajności. Bazy danych mogą udostępniać repliki tylko do odczytu. Może to jednak niepotrzebnie ujawniać ich wewnętrzne modele danych. Przetwarzanie wsadowe może zrzucać dane do magazynu plików w celu ich odczytywania przez inne procesy, ale takie podejście może powodować problemy dotyczące spójności danych i wielu źródeł prawdy. Wreszcie, wszystkie te rozwiązania skutkują powstaniem silnych powiązań między implementacjami i dalszym ugruntowaniem w architekturze bezpośrednich relacji punkt-punkt.

Jeżeli stwierdzisz, że dostęp do danych w Twojej organizacji jest zbyt trudny, albo że produkty podlegają zmianom zakresu, ponieważ wszystkie dane są zlokalizowane w pojedynczej implementacji, prawdopodobnie doświadczasz skutków słabych struktur komunikacji danych. Ten problem będzie się powiększał w miarę rozwoju organizacji, opracowywania nowych produktów i zwiększającej się potrzeby uzyskiwania dostępu do powszechnie używanych danych dziedzinowych.

# Struktury komunikacji w tradycyjnych systemach obliczeniowych

Struktury komunikacji organizacji mają duży wpływ na sposób tworzenia implementacji inżynierskich. Dzieje się tak również na poziomie zespołów: struktury komunikacji zespołu wpływają na rozwiązania, które buduje dla określonych, przypisanych mu wymagań biznesowych. Zobaczmy, jak działa to w praktyce.

Rozważ następujący scenariusz. Pojedynczy zespół ma jedną usługę wspieraną przez jeden magazyn danych. Z radością pełni on swoją funkcję biznesową i wszystko jest pięknie i wspaniale. Pewnego dnia szef zespołu przedstawia nowe wymaganie biznesowe. Jest ono w pewnym sensie związane z tym, co zespół robi dotychczas, i może zostać po prostu dodane do istniejącej usługi. Różni się jednak od pierwotnych zadań na tyle, że mogłoby zasługiwać na własną, nową usługę.

Zespół jest w rozterce: czy implementować nowe wymaganie biznesowe w nowej usłudze, czy po prostu dodać je do istniejącej usługi? Przyjrzyjmy się szczegółowo obu opcjom.

## Opcja 1. Utworzenie nowej usługi

Wymóg biznesowy różni się od dotychczasowego na tyle, że umieszczenie go w nowej usłudze może mieć sens. Ale co z danymi? Ta nowa funkcja biznesowa wymaga niektórych starych danych, ale dane te są obecnie zablokowane w istniejącej usłudze. Ponadto zespół tak naprawdę nie ma opracowanego procesu uruchamiania nowych, w pełni niezależnych usług. Z drugiej strony zespół robi się coraz większy, a firma szybko się rozwija. Jeżeli w przyszłości zespół będzie musiał zostać podzielony, posiadanie modułowych i niezależnych systemów może znacznie ułatwić podział własności.

Z takim podejściem wiąże się ryzyko. Zespół musi znaleźć sposób na pozyskanie danych z pierwotnego magazynu danych i skopiowanie ich do nowego magazynu. Członkowie zespołu muszą upewnić się, że nie ujawnią wewnętrznych mechanizmów działania i że zmiany, które wprowadzą w swoich strukturach danych, nie wpłyną na żadne inne zespoły kopiujące ich dane. Ponadto kopiowane dane zawsze będą nieco nieaktualne, ponieważ można sobie pozwolić na kopiowanie danych produkcyjnych w czasie rzeczywistym co 30 minut, aby nie obciążać magazynu danych zapytaniami. To połączenie będzie musiało być monitorowane i utrzymywane, aby mieć pewność, że działa poprawnie.

Istnieje również ryzyko związane z uruchomieniem nowej usługi. Pracownicy będą musieli zarządzać dwoma magazynami danych i dwoma usługami oraz opracować dla nich procesy rejestrowania, monitorowania, testowania, wdrażania i przywracania. Muszą również zadbać o synchronizację wszelkich zmian struktury danych, aby nie wpływać na system zależny.

## Opcja 2. Dodanie funkcjonalności do istniejącej usługi

Drugą opcją jest utworzenie nowych struktur danych i logiki biznesowej w ramach istniejącej usługi. Wymagane dane znajdują się już w magazynie danych, a procesy rejestrowania, monitorowania, testowania, wdrażania i przywracania są już zdefiniowane i używane. Zespół zna system i może od razu przystąpić do pracy nad implementacją logiki, a ich monolityczne wzorce obsługują takie podejście do projektowania usług.

Z takim podejściem również wiążą się zagrożenia, choć są one nieco bardziej subtelne. W miarę wprowadzania zmian granice implementacji mogą się zacierać, zwłaszcza że moduły są często zgrupowane w tej samej bazie kodu. Bardzo łatwo jest szybko dodawać funkcjonalności przekraczając te granice, bezpośrednio powiązując je w module. Możliwość tak szybkiego działania jest wielkim dobrodziejstwem, ale odbywa się to kosztem ścisłych powiązań, zmniejszonej spójności i braku modułowości. Chociaż zespoły mogą się przed tym ustrzec, wymaga to doskonałego planowania i ścisłego przestrzegania granic, które często zanikają w obliczu napiętych harmonogramów, braku doświadczenia i zmieniającej się własności usług.

## Wady i zalety obu opcji

Większość zespołów wybrałaby drugą opcję — dodanie funkcjonalności do istniejącego systemu. W tym wyborze nie ma nic złego. Architektury monolityczne to użyteczne i wszechstronne struktury, które mogą dostarczyć przedsiębiorstwu wyjątkową wartość. Pierwsza opcja napotyka od razu dwa problemy związane z tradycyjnymi systemami obliczeniowymi:

- Trudno w sposób niezawodny zapewnić dostęp do danych z innego systemu, zwłaszcza na dużą skalę i w czasie rzeczywistym.
- Tworzenie nowych usług i zarządzanie nimi wiąże się ze znacznymi nakładami pracy i ryzykiem, zwłaszcza jeśli w organizacji nie ma dla tego ustalonej procedury.

Uzyskiwanie dostępu do danych lokalnych jest zawsze łatwiejsze niż zapewnienie dostępu do danych z innego magazynu danych. Trudno jest pozyskać jakiekolwiek dane zhermetyzowane w magazynie danych innego zespołu, ponieważ wymaga to przekroczenia granic zarówno komunikacji implementacyjnej, jak i biznesowej. Wraz ze wzrostem ilości danych, liczby połączeń i wymagań dotyczących wydajności, staje się to coraz trudniejsze do utrzymywania i skalowania.

Chociaż kopiowanie niezbędnych danych jest wartym rozważenia podejściem, nie jest niezawodne. Ten model promuje tworzenie wielu bezpośrednich powiązań typu punkt-punkt, których utrzymywanie staje się problematyczne w miarę rozwoju organizacji, zmieniania się jednostek biznesowych i własności oraz dojrzewania i wycofywania produktów. Stwarza to ścisłą techniczną zależność struktur komunikacji implementacyjnej obu zespołów (zespołu przechowującego dane i zespołu kopiującego je), wymagając od nich synchronicznej pracy przy każdej zmianie danych.

Należy podjąć szczególne starania, by wewnętrzny model danych implementacji nie został udostępniony w zbyt szerokim zakresie, żeby inne systemy nie zostały z nim ściśle powiązane. Skalowalność, wydajność i dostępność systemu są często problematyczne dla obu systemów, ponieważ kwerenda replikacji danych może spowodować niezrównoważone obciążenie systemu źródłowego. Nieudane procesy synchronizacji mogą zostać zauważone dopiero po wystąpieniu sytuacji awaryjnej. Wiedza plemienna może spowodować, że zespół będzie kopiował kopię danych, myśląc, że jest to oryginalne źródło prawdy.

Zanim zostanie zakończona obsługa zapytania, a dane zostaną przesłane, skopiowane dane zawsze będą już w pewnym stopniu nieaktualne. Im większy zbiór danych i bardziej złożone jego źródła, tym większe prawdopodobieństwo, że kopia nie będzie zsynchronizowana z oryginałem. Jest to problematyczne, gdy systemy oczekują od siebie posiadania doskonałych, aktualnych kopii, szczególnie podczas komunikacji między sobą w kwestii tych danych. Z powodu nieaktualności usługa raportowania może na przykład zgłaszać inne wartości niż usługa rozliczeniowa. Może to mieć poważne konsekwencje dla jakości usług, raportowania, analiz i podejmowania decyzji finansowych.

Brak możliwości prawidłowego rozpowszechnienia danych w całym przedsiębiorstwie nie wynika z fundamentalnej wady koncepcji. Wręcz przeciwnie — jest to spowodowane słabą lub nieistniejącą strukturą komunikacji danych. W powyższym scenariuszu struktura komunikacji implementacyjnej zespołu pełni dodatkową rolę jako bardzo ograniczona struktura komunikacji danych.

Jednym z założeń mikrousług opartych na zdarzeniach jest to, że podstawowe dane biznesowe powinny być łatwe do uzyskania i wykorzystania przez każdą usługę, która tego wymaga. W tym scenariuszu zastępuje to strukturę komunikacji danych ad hoc sformalizowaną strukturą komunikacji danych. W przypadku hipotetycznego zespołu taka struktura komunikacji danych mogłaby wyeliminować większość trudności związanych z pozyskiwanie danych z innych systemów.

## Kontynuacja scenariusza z zespołami

Załóżmy, że od naszego ostatniego spotkania z zespołem minął rok. Zespół zdecydował się na opcję 2. i dołączenie nowych funkcjonalności do tej samej usługi. Było to szybkie, łatwe zadanie i od tamtej pory wprowadzono szereg nowych funkcjonalności. Wraz z rozwojem firmy zespół się powiększał, a teraz nadszedł czas, aby go przeorganizować i podzielić na dwie mniejsze, bardziej skoncentrowane grupy.

Każdemu nowemu zespołowi należy teraz przypisać określone funkcje biznesowe z poprzedniej usługi. Wymagania biznesowe każdego zespołu zostały starannie podzielone na podstawie obszarów firmy, które wymagają największej uwagi. Podział struktury komunikacji implementacyjnej nie okazuje się jednak łatwy. Tak jak poprzednio, wydaje się, że oba zespoły do realizacji swoich zadań potrzebują dużej ilości tych samych danych. Pojawiają się nowe pytania:

- Który zespół powinien posiadać określone dane?
- Gdzie powinny znajdować się dane?
- Co będzie się działo z danymi, w przypadku gdy oba zespoły będą musiały modyfikować wartości?

Kierownicy zespołów decydują, że najlepiej będzie po prostu współdzielić usługę, a oba zespoły będą mogły pracować nad różnymi częściami. Będzie to wymagało znacznie większej komunikacji między zespołami i synchronizacji działań, co może obniżyć produktywność. A jeżeli w przyszłości rozmiary zespołów znów się podwoją? Albo jeśli wymagania biznesowe zmienią się na tyle, że zespoły nie będą już w stanie ich w stanie spełniać w całości przy tej samej strukturze danych?

### Sprzeczne naciski

Pierwotny zespół został poddany dwóm sprzecznym naciskom. Zalecono utrzymywanie wszystkich jego danych lokalnie w jednej usłudze, aby przyspieszyć i ułatwić dodawanie nowych funkcji biznesowych kosztem rozbudowy struktury komunikacji implementacyjnej. Rozwój zespołu wymagał ostatecznie podzielenia struktury komunikacji biznesowej — wymusiło to ponowne przypisanie wymagań biznesowych do nowych zespołów. Struktura komunikacji implementacyjnej w swojej obecnej formie nie mogła jednak wspierać tego ponownego przypisania i musiała zostać podzielona na odpowiednie komponenty. Żadne z podejść nie jest skalowalne i oba wskazują na potrzebę działania w inny sposób. Wszystkie te problemy mają tę samą pierwotną przyczynę: słabe, źle zdefiniowane środki przekazywania danych pomiędzy strukturami komunikacji implementacyjnej.

## Struktury komunikacji oparte na zdarzeniach

Podejście oparte na zdarzeniach stanowi alternatywę dla tradycyjnego działania struktur komunikacji implementacyjnej i struktur komunikacji danych. Komunikacja oparta na zdarzeniach nie jest bezpośrednim zastępstwem dla komunikacji typu żądanie-odpowiedź, ale raczej całkowicie innym sposobem komunikacji między usługami. Struktura komunikacji danych poprzez strumieniowe przesyłanie zdarzeń oddziela dostęp do danych dla środowiska produkcyjnego i własności danych. Usługi nie są już powiązane bezpośrednio przez interfejs API żądanie-odpowiedź, ale zamiast tego wykorzystują dane dotyczące zdarzeń zdefiniowane w strumieniach zdarzeń (ten proces zostanie szerzej omówiony w rozdziale 3.). Obowiązki producentów są ograniczone do generowania dobrze zdefiniowanych danych w odpowiednich strumieniach zdarzeń.

### Zdarzenia są podstawą komunikacji

Wszystkie odpowiednie do udostępniania dane są publikowane w zestawie strumieni zdarzeń, tworząc ciągłą, kanoniczną narrację opisującą szczegółowo wszystko, co wydarzyło się w organizacji. Staje się to kanałem, za pomocą którego komunikują się ze sobą systemy. W postaci zdarzeń można przesyłać prawie wszystko, od prostych wystąpień po złożone rekordy stanowe. Zdarzenia **są** danymi. To nie tylko sygnały wskazujące, że gdzieś indziej gotowe są pewne dane, czy też zaledwie środek bezpośredniego przesyłania danych z jednej implementacji do drugiej. Działają raczej zarówno jako magazyn danych, jak i środek asynchronicznej komunikacji między usługami.

### Strumienie zdarzeń zapewniają jedno źródło prawdy

Każde zdarzenie w strumieniu jest stwierdzeniem faktu, a wszystkie te stwierdzenia tworzą razem jedno źródło prawdy — podstawę komunikacji dla wszystkich systemów w organizacji. Struktura komunikacyjna jest tak dobra, jak wiarygodność jej informacji, dlatego bardzo ważne jest, aby organiza-

cja przyjęła narrację strumienia zdarzeń jako pojedyncze źródło prawdy. Jeżeli niektóre zespoły zdecydują się zamiast tego umieszczać sprzeczne dane w innych lokalizacjach, funkcja strumienia zdarzeń jako szkieletu komunikacji danych w organizacji zostanie znacznie ograniczona.

## Konsumenty przeprowadzają własne modelowanie i kwerendowanie

Struktura komunikacji danych oparta na zdarzeniach różni się od nadmiernie rozbudowanej struktury komunikacji implementacyjnej pod tym względem, że nie jest w stanie zapewnić żadnej funkcjonalności kwerendowania lub wyszukiwania danych. Cała logika biznesowa i logika aplikacji muszą być zhermetyzowane w producentach i konsumentach zdarzeń.

Wymagania dotyczące dostępu do danych i modelowania są całkowicie przenoszone na konsumenta, a każdy konsument otrzymuje własną kopię zdarzeń ze źródłowych strumieni zdarzeń. Wszelka złożoność kwerendowania również zostaje przeniesiona ze struktury komunikacji implementacyjnej właściciela danych do struktury konsumenta. Konsument pozostaje w pełni odpowiedzialny za mieszanie danych z wielu strumieni zdarzeń, specjalną funkcjonalność kwerendowania lub inną logikę implementacji charakterystyczną dla danej działalności gospodarczej. W przeciwnym razie zarówno producenty, jak i konsumenty zostają zwolnione z obowiązku zapewniania mechanizmów kwerendowania, mechanizmów przesyłania danych, interfejsów API (ang. *Application Programming Interface*) i międzyzespołowych usług w zakresie środków przekazywania danych. Ich odpowiedzialność zostaje wtedy ograniczona jedynie do zaspokajania potrzeb związanych z ich bezpośrednim kontekstem ograniczonym.

## Komunikacja danych zostaje usprawniona w całej organizacji

Użycie struktury komunikacji danych jest inwersją, w której wszystkie możliwe do współdzielenia dane są udostępniana poza strukturą komunikacji implementacyjnej. Nie wszystkie dane muszą być współdzielone, a zatem nie wszystkie muszą być publikowane w zestawie strumieni zdarzeń. Jednak wszelkie dane, które leżą w gestii zainteresowań jakiegokolwiek innego zespołu lub jakiejś usługi, muszą być publikowane we wspólnym zestawie strumieni zdarzeń, aby środowisko produkcyjne i własność danych zostały całkowicie rozdzielone. Zapewnia to sformalizowaną strukturę komunikacji danych, nieobecną od dawna w architekturach systemów, która ponadto zapewnia większą zgodność z zasadami kontekstu ograniczonego dotyczącymi luźnych powiązań i dużej spójności.

Aplikacje mogą teraz uzyskiwać dostęp do danych, których uzyskanie w przeciwnym przypadku byłoby pracochłonnym procesem wykorzystującym połączenia typu punkt-punkt. Nowe usługi mogą po prostu pozyskiwać wszelkie potrzebne dane z kanonicznych strumieni zdarzeń, tworzyć własne modele i stany oraz wykonywać wszystkie niezbędne funkcje biznesowe bez konieczności polegania na bezpośrednich połączeniach punkt-punkt lub interfejsach API związanych z jakąkolwiek inną usługą. Dzięki temu organizacja może odblokować potencjał do skuteczniejszego wykorzystywania ogromnych ilości swoich danych w dowolnym produkcie, a nawet mieszania danych z wielu produktów w unikatowe i wielofunkcyjne sposoby.

## Dostępne dane wspierają zmiany w komunikacji biznesowej

Strumienie zdarzeń zawierają podstawowe zdarzenia dziedzinowe, które mają kluczowe znaczenie dla działania firmy. Chociaż zespoły mogą się restrukturyzować, a projekty mogą się rozpoczynać i kończyć, ważne podstawowe dane dziedzinowe pozostają łatwo dostępne dla każdego nowego produktu, który ich wymaga, *niezależnie od jakiejkolwiek konkretnej struktury komunikacji implementacyjnej*. Daje to firmie niespotykaną elastyczność, ponieważ dostęp do podstawowych zdarzeń dziedzinowych nie jest już uzależniony od żadnej konkretnej implementacji

# Asynchroniczne mikrousługi oparte na zdarzeniach

Mikrousługi oparte na zdarzeniach umożliwiają przeprowadzanie transformacji logiki biznesowej i operacji niezbędne do spełnienia wymagań kontekstu ograniczonego. Zadaniem tych aplikacji jest spełnienie tych wymagań i emitowanie własnych niezbędnych zdarzeń do innych downstreamowych konsumentów. Oto kilka głównych zalet korzystania z mikrousług opartych na zdarzeniach:

**Stopień szczegółowości**

Usługi doskonale mapują się na konteksty ograniczone i można je łatwo przepisywać, gdy zmieniają się wymagania biznesowe.

**Skalowalność**

W razie potrzeby poszczególne usługi można skalować pionowo (w górę lub w dół).

**Elastyczność technologiczna**

Usługi korzystają z najbardziej odpowiednich języków i technologii. Pozwala to również na łatwe prototypowanie przy użyciu pionierskich technologii.

**Elastyczność wymagań biznesowych**

Można łatwo reorganizować własność szczegółowych mikrousług. W porównaniu z dużymi usługami istnieje mniej zależności między zespołami, a organizacja może szybciej reagować na zmiany wymagań biznesowych, co w innym przypadku byłoby utrudnione z uwagi na bariery w dostępie do danych.

**Luźne powiązania**

Mikrousługi oparte na zdarzeniach są powiązane z danymi dziedzinowymi, a nie z interfejsem API konkretnej implementacji. Schematy danych mogą znacznie poprawić sposób zarządzania zmianami danych; zostanie to omówione w rozdziale 3.

**Wsparcie dla ciągłego dostarczania**

Małą, modułową mikrousługę łatwo jest zarówno dostarczyć, jak i wycofać, gdy zajdzie taka potrzeba.

**Wysoka testowalność**

Mikrousługi mają zwykle mniej zależności niż duże monolity, co ułatwia tworzenie atrap wymaganych punktów końcowych testowania i zapewnia odpowiednie pokrycie kodu testami.

## Przykładowy zespół korzystający z mikrousług opartych na zdarzeniach

Wróćmy do zespołu z wcześniejszego przykładu, ale tym razem ze strukturą komunikacji danych opartą na zdarzeniach.

Zespół otrzymuje nowy wymóg biznesowy. Jest on w pewnym stopniu związany z funkcjonalnościami obecnych produktów zespołu, ale z uwagi na pewne różnice w specyfikacji może równie dobrze zostać zrealizowany za pośrednictwem osobnej usługi. Czy dodanie go do istniejącej usługi naruszy zasadę pojedynczej odpowiedzialności i nadmiernie rozszerzy obecnie zdefiniowany kontekst ograniczony? A może jest to proste rozszerzenie, na przykład dodanie do istniejącej usługi pewnych nowych powiązanych danych albo jakiejś funkcjonalności?

Wcześniejsze problemy techniczne — takie jak ustalanie, gdzie ma być źródło danych i jak je pobierać, obsługa synchronizacji wsadowej oraz implementowanie synchronicznych interfejsów API — zostały w dużej mierze usunięte. Zespół może uruchomić nową mikrousługę i pobierać niezbędne dane ze strumieni zdarzeń, cofając się aż do początku strumieni, gdy zachodzi taka potrzeba. Istnieje możliwość, aby zespół wprowadzał domieszki do wspólnych danych używanych w innych usługach, o ile dane te będą wykorzystywane wyłącznie do zaspokojenia potrzeb nowego kontekstu ograniczonego. Przechowywanie i struktura tych danych są zależne całkowicie od zespołu, który może wybrać, które pola zachować, a które odrzucić.

Mniejsze jest również ryzyko biznesowe, ponieważ małe, szczegółowe usługi umożliwiają powierzenie własności pojedynczemu zespołowi, co pozwala zespołom na skalowanie i reorganizację, jeśli pojawia się taka potrzeba. Gdy zespół staje się zbyt duży, aby działać jako właściciel pojedynczej funkcji biznesowej, może zostać podzielony zgodnie z wymaganiami, a własności mikrousług mogą zostać ponownie przypisane. Własność danych o zdarzeniach przenosi się wraz z usługą produkcyjną i można podjąć decyzje organizacyjne w celu zmniejszenia ilości komunikacji między zespołami wymaganej do wykonywania przyszłych prac.

Charakter mikrousługi zapobiega powstawaniu tzw. kodu spaghetti i rozprzestrzenianiu się ekspansywnych monolitów, pod warunkiem, że nakłady związane z tworzeniem nowych usług i uzyskiwaniem niezbędnych danych będą minimalne. Kwestie skalowania koncentrują się teraz na poszczególnych usługach przetwarzania zdarzeń, co umożliwia odpowiednie skalowanie użycia przez nie procesora, pamięci i dysku oraz liczby instancji. Pozostałe wymagania dotyczące skalowania są przenoszone na strukturę komunikacji danych, która musi zapewniać, że jest w stanie obsłużyć różne obciążenia usług konsumujących zdarzenia ze strumieni zdarzeń i produkujących zdarzenia do tych strumieni.

Aby to wszystko zrobić, zespół musi jednak upewnić się, że w strukturze komunikacji danych dane rzeczywiście są obecne. Musi także mieć środki umożliwiające łatwe uruchamianie floty mikrousług i zarządzanie nią. Wymaga to przyjęcia architektury EDM w całej organizacji.

# Mikrousługi synchroniczne

Mikrousługi mogą być implementowane asynchronicznie przy użyciu zdarzeń (jest to podejście zalecane w tej książce) lub synchronicznie, co jest powszechne w architekturach usługowych. Synchroniczne mikrousługi są zwykle realizowane przy użyciu podejścia typu żądanie-odpowiedź, w którym usługi w celu spełnienia wymagań biznesowych komunikują się bezpośrednio przez interfejsy API.

## Wady mikrousług synchronicznych

Z mikrousługami synchronicznymi wiąże się szereg problemów, które utrudniają ich stosowanie na dużą skalę. Nie oznacza to, że firma nie może działać z powodzeniem, korzystając z mikrousług synchronicznych, o czym świadczą osiągnięcia takich firm jak Netflix, Lyft, Uber czy Facebook. Ale wiele przedsiębiorstw dorobiło się również fortun na archaicznych i potwornie splątanych monolitach z kodem spaghetti, więc nie należy mylić ewentualnego sukcesu firmy z jakością jej bazowej architektury. Istnieje wiele książek opisujących sposób implementacji mikrousług synchronicznych, zalecam więc ich lekturę[1], abyś mógł lepiej zrozumieć podejście synchroniczne.

Ponadto należy pamiętać, że żadne z tych typów usług nie są zasadniczo lepsze jedne od drugich. Zarówno mikrousługi typu żądanie-odpowiedź opierające się na połączeniach punkt-punkt, jak i asynchroniczne mikrousługi oparte na zdarzeniach mają swoje miejsce w organizacji, ponieważ niektóre zadania są znacznie lepiej dostosowane do jednego, a inne do drugiego typu mikrousług. Jednak moje własne doświadczenie, jak również doświadczenie wielu moich kolegów i koleżanek, wskazuje, że architektury EDM oferują niezrównaną elastyczność, której brakuje w synchronicznych mikrousługach typu żądanie-odpowiedź. Być może postanowisz zgodzić się z tym stwierdzeniem w miarę lektury tej książki, a jeśli nie, to przynajmniej zrozumiesz wady i zalety architektury mikrousług opartych na zdarzeniach.

W kolejnych podpunktach opisano niektóre z największych wad synchronicznych mikrousług typu żądanie-odpowiedź.

### Powiązania typu punkt-punkt

Mikrousługi synchroniczne opierają się na innych usługach, które pomagają im wykonywać zadania biznesowe. Te usługi z kolei mają własne usługi zależne, które również mają własne usługi zależne — i tak dalej. Może to prowadzić do nadmiernej obciążalności wyjściowej i utrudniać śledzenie, które usługi są odpowiedzialne za wypełnianie określonych części logiki biznesowej. Liczba połączeń między usługami może stać się zdumiewająco duża, co dodatkowo umacnia istniejące struktury komunikacyjne i utrudnia wprowadzanie zmian w przyszłości.

### Skalowanie zależne

Możliwość skalowania pionowego własnej usługi zależy również od zdolności skalowania pionowego wszystkich usług zależnych i jest bezpośrednio związana ze stopniem obciążenia wyjściowego komunikacji. Technologie implementacyjne mogą stanowić wąskie gardło w zakresie skalowalności. Komplikują to dodatkowo bardzo zmienne obciążenia i wzorce rosnącej liczby żądań, które muszą być obsługiwane synchronicznie w całej architekturze.

### Obsługa awarii usług

Jeśli nie działa usługa zależna, należy podjąć decyzje dotyczące obsługi wyjątku. Im więcej usług znajduje się w ekosystemie, tym trudniejsze staje się decydowanie, jak radzić sobie z awariami, kiedy

[1] Możesz zapoznać się na przykład z pozycjami *Budowanie mikrousług* Sama Newmana (wyd. Helion) oraz *Microservices for the Enterprise* Kasuna Indrasiriego i Prabatha Siriwardeny (wyd. Apress).

ponawiać próby, kiedy stwierdzić awarię usługi i jak odzyskać sprawność po awarii, aby zapewnić spójność danych.

### Zarządzanie wersjami i zależnościami interfejsu API

Często konieczne będzie jednoczesne istnienie wielu definicji interfejsu API i wielu wersji usług. Nie zawsze jest możliwe lub pożądane zmuszanie klientów do aktualizacji do najnowszego interfejsu API. Może to wprowadzić znaczną złożoność w zakresie instrumentalizacji żądań zmian interfejsu API dla wielu usług, zwłaszcza jeśli towarzyszą im zmiany w bazowych strukturach danych.

### Dostęp do danych powiązanych z implementacją

Jeśli chodzi o uzyskiwanie dostępu do danych zewnętrznych, mikrousługi synchroniczne mają te same problemy, co tradycyjne usługi. Chociaż istnieją strategie projektowania usług w celu minimalizacji potrzeb uzyskiwania dostępu do danych zewnętrznych, mikrousługi i tak często będą musiały uzyskać dostęp do powszechnie używanych danych z innych usług. W takim przypadku ciężar zapewnienia dostępu do danych i skalowalności przenoszony jest z powrotem na strukturę komunikacji implementacyjnej.

### Monolity rozproszone

Usługi mogą być skomponowane w taki sposób, że będą działać jak monolit rozproszony, z wieloma przeplatającymi się wywołaniami między nimi. Do takiej sytuacji często dochodzi wtedy, gdy zespół dekomponuje monolit i decyduje się na użycie synchronicznych wywołań typu punkt-punkt w celu naśladowania istniejących granic w obrębie danego monolitu. Usługi punkt-punkt ułatwiają zacieranie się linii między kontekstami ograniczonymi, ponieważ wywołania funkcji wysyłane do systemów zdalnych mogą przeplatać się z liniami istniejącego kodu monolitycznego.

### Testowanie

Testowanie integracji może być trudne, ponieważ każda usługa wymaga w pełni operacyjnych zależności, które z kolei wymagają własnych zależności — i tak dalej. Stosowanie namiastek usług może działać w czasie realizacji testów jednostkowych, ale rzadko okazuje się wystarczające w przypadku większych wymagań testowych.

## Korzyści ze stosowania mikrousług synchronicznych

Mikrousługi synchroniczne zapewniają wiele niezaprzeczalnych korzyści. Niektóre wzorce dostępu do danych preferują bezpośrednie powiązania typu żądanie-odpowiedź, na przykład uwierzytelnianie użytkownika i raportowanie z przeprowadzania testów A/B. Integracje z zewnętrznymi rozwiązaniami prawie zawsze wykorzystują mechanizm synchroniczny i generalnie zapewniają elastyczny, niezależny od języka mechanizm komunikacji za pośrednictwem protokołu HTTP.

Śledzenie operacji w wielu systemach może być łatwiejsze w środowisku synchronicznym niż w asynchronicznym. Szczegółowe dzienniki zdarzeń mogą pokazywać, które funkcje były wywoływane w poszczególnych systemach, co zapewnia duże możliwości debugowania i wgląd w operacje biznesowe.

Usługi hostujące internetowe i mobilne interfejsy użytkowników są ogólnie rzecz biorąc oparte na projektach typu żądanie-odpowiedź, niezależnie od ich synchronicznego czy asynchronicznego charakteru. Klienty otrzymują terminową odpowiedź w całości dostosowaną do ich potrzeb.

Dość istotny jest również czynnik doświadczenia, zwłaszcza że wielu programistów obecnych na dzisiejszym rynku ma większe doświadczenie w synchronicznym kodowaniu w stylu monolitycznym. To sprawia, że zasadniczo łatwiej jest pozyskiwać utalentowanych programistów do systemów synchronicznych niż do asynchronicznego programowania opartego na zdarzeniach.

Architektura firmy rzadko, jeśli w ogóle, może całkowicie bazować na mikrousługach opartych na zdarzeniach. Z pewnością normą będą architektury hybrydowe, w których rozwiązania synchroniczne i asynchroniczne są wdrażane obok siebie zgodnie z wymaganiami przestrzeni problemów.

# Podsumowanie

Struktury komunikacyjne kierują sposobem tworzenia oprogramowania i zarządzania nim przez cały okres istnienia organizacji. Struktury komunikacji danych są często niedopracowane i tworzone ad hoc, ale wprowadzenie trwałego, łatwo dostępnego zestawu zdarzeń dziedzinowych (tak jak w systemach opartych na zdarzeniach) umożliwia stosowanie mniejszych, specjalnie zaprojektowanych implementacji.

# Mikrousługi: odblokuj potencjał danych!

Mikrousługi oparte na zdarzeniach mogły powstać dzięki rozwojowi konteneryzacji i łatwiejszemu pozyskiwaniu zasobów obliczeniowych. Szczególnie ciekawą propozycją są rozproszone, odporne na błędy, wysokowydajne i szybkie brokery zdarzeń. Te rozwiązania technologiczne pozwalają na korzystanie z wielkich zbiorów danych i zapewniają przetwarzanie zdarzeń w czasie zbliżonym do rzeczywistego. Taka architektura ma znaczenie nie tylko dla inżynierów systemu, ale także dla struktury i sposobu działania firm i organizacji. Nagle okazuje się, że można bezproblemowo wykonywać działania, które do niedawna nie były jeszcze możliwe.

To praktyczny przewodnik, dzięki któremu samodzielnie ocenisz, jak dużą wartość biznesową mogą zyskać duże wolumeny danych, jeśli są wykorzystywane w organizacji przy użyciu architektury mikrousług opartych na zdarzeniach. Dowiesz się także, jak przygotować i przeprowadzić proces budowania organizacji, która wykorzystuje mikrousługi oparte na zdarzeniach. Poznasz wszechstronne, a przy tym proste wzorce odblokowujące wartość tych danych. W książce znalazło się również mnóstwo wskazówek i sugestii dotyczących projektowania systemu opartego na zdarzeniach, ponadto wyjaśniono tutaj kluczowe zasady architektoniczne. Wymieniono też przydatne narzędzia oraz opisano dokładnie techniki testowania mikrousług i wdrażania ich w środowisku produkcyjnym.

## W książce między innymi:

- rola architektury opartej na zdarzeniach w dostarczaniu wyjątkowej wartości biznesowej
- mikrousługi w projektowaniu opartym na zdarzeniach
- najlepsze wzorce architektoniczne
- wzorce aplikacji do tworzenia wielofunkcyjnych mikrousług opartych na zdarzeniach
- komponenty i narzędzia wymagane do uruchomienia ekosystemu mikrousługowego

**Adam Bellemare** jest inżynierem kadrowym platformy danych w firmie Shopify. Wcześniej był programistą w BlackBerry, gdzie zaczął zajmować się systemami opartymi na zdarzeniach. Zdobył duże doświadczenie w zakresie zarządzania technicznego, przetwarzania rozproszonego i asynchronicznego oraz inżynierii danych behawioralnych.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl



KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-7439-3

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 69,00 zł