

O'REILLY®

Mikroustugi 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

© 2021 Helion SA

Authorized Polish translation of the English edition of Building Event-Driven Microservices ISBN 9781492057895 © 2020 Adam Bellemare

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Helion SA dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Przedmowa	13
1. Dlaczego mikrouслуги oparte na zdarzeniach?	17
Czym są mikrouслуги oparte na zdarzeniach?	17
Wprowadzenie do projektowania dziedzinowego i kontekstów ograniczonych	19
Wykorzystywanie modeli dziedzin i kontekstów ograniczonych	20
Dopasowywanie kontekstów ograniczonych do wymagań biznesowych	20
Struktury komunikacji	22
Struktury komunikacji biznesowej	22
Struktury komunikacji implementacyjnej	22
Struktury komunikacji danych	23
Prawo Conwaya i struktury komunikacji	23
Struktury komunikacji w tradycyjnych systemach obliczeniowych	25
Opcja 1. Utworzenie nowej usługi	25
Opcja 2. Dodanie funkcjonalności do istniejącej usługi	26
Wady i zalety obu opcji	26
Kontynuacja scenariusza z zespołami	27
Sprzeczne naciski	28
Struktury komunikacji oparte na zdarzeniach	28
Zdarzenia są podstawą komunikacji	28
Strumienie zdarzeń zapewniają jedno źródło prawdy	28
Konsumenty przeprowadzają własne modelowanie i kwerendowanie	29
Komunikacja danych zostaje usprawniona w całej organizacji	29
Dostępne dane wspierają zmiany w komunikacji biznesowej	30
Asynchroniczne mikrouслуги oparte na zdarzeniach	30
Przykładowy zespół korzystający z mikrouslug opartych na zdarzeniach	31
Mikrouслуги synchroniczne	31
Wady mikrouslug synchronicznych	32
Korzyści ze stosowania mikrouslug synchronicznych	33
Podsumowanie	34

2. Podstawy mikrousług opartych na zdarzeniach	35
Budowanie topologii	35
Topologia mikrousługi	35
Topologia biznesowa	36
Zawartość zdarzenia	37
Struktura zdarzenia	37
Zdarzenie bez klucza	38
Zdarzenie encyjne	38
Zdarzenie z kluczem	38
Materializowanie stanu ze zdarzeń encyjnych	39
Definicje i schematy danych zdarzenia	41
Zasada pojedynczego źródła zapisu mikrousługi	41
Mikrousługi wyposażone w broker zdarzeń	41
Przechowywanie i serwowanie zdarzeń	42
Uwzględnianie czynników dodatkowych	43
Porównanie brokerów zdarzeń i brokerów komunikatów	44
Konsumowanie zdarzeń z niemutowalnego dziennika	45
Zapewnianie jednego źródła prawdy	47
Zarządzanie mikrousługami na dużą skalę	47
Umieszczanie mikrousług w kontenerach	48
Umieszczanie mikrousług w maszynach wirtualnych	48
Zarządzanie kontenerami i maszynami wirtualnymi	48
Podatek od mikrousług	49
Podsumowanie	50
3. Komunikacja i kontrakty danych	51
Kontrakty danych opartych na zdarzeniach	51
Używanie bezpośrednich schematów jako kontraktów	52
Komentarze definicji schematu	52
W pełni funkcjonalna ewolucja schematu	53
Obsługa generatora kodu	54
Przełomowe zmiany schematu	55
Wybór formatu zdarzenia	57
Projektowanie zdarzeń	57
Mów prawdę, całą prawdę i tylko prawdę	58
Używaj pojedynczej definicji zdarzenia na strumień	58
Używaj najwięzszych typów danych	58
Zachowuj specjalizację zdarzeń	59
Minimalizuj rozmiar zdarzeń	62
Angażuj potencjalne konsumenty w projektowanie zdarzeń	62
Unikaj zdarzeń jako semaforów lub sygnałów	62
Podsumowanie	63

4. Integracja architektur opartych na zdarzeniach z istniejącymi systemami	65
Czym jest uwalnianie danych?	66
Kompromisy związane z uwalnianiem danych	66
Konwersja uwolnionych danych na zdarzenia	68
Wzorce uwalniania danych	69
Frameworki uwalniania danych	70
Uwalnianie danych oparte na zapytaniach	70
Ładowanie zbiorcze	70
Ładowanie przyrostowe na podstawie znacznika czasu	71
Ładowanie na podstawie autoinkrementowanego identyfikatora	71
Kwerendowanie niestandardowe	71
Aktualizacja przyrostowa	71
Zalety aktualizacji opartej na zapytaniach	72
Wady aktualizacji opartej na zapytaniach	72
Uwalnianie danych oparte na dziennikach CDC	73
Zalety korzystania z dzienników magazynu danych	75
Wady korzystania z dzienników magazynu danych	75
Uwalnianie danych oparte na tablicach skrzynki nadawczej	75
Kwestie związane z wydajnością	77
Izolowanie wewnętrznych modeli danych	77
Zapewnianie kompatybilności schematu	79
Przechwytywanie danych o zmianach za pomocą wyzwalaczy	82
Dokonywanie zmian definicji danych w przechwytywanych zbiorach danych	85
Obsługa zmian definicji danych „po fakcie”	
dla wzorców opartych na zapytaniach i dziennikach CDC	86
Obsługa zmian definicji danych dla wzorców opartych	
na przechwytywaniu tablicy danych o zmianach	86
Zlewanie danych o zdarzeniach do magazynów danych	87
Wpływ zlewania i pozyskiwania danych na prowadzenie działalności biznesowej	87
Podsumowanie	89
5. Podstawy przetwarzania opartego na zdarzeniach	91
Tworzenie topologii bezstanowych	92
Transformacje	92
Rozgałęzianie i scalanie strumieni	93
Repartycjonowanie strumieni zdarzeń	93
Przykład: repartycjonowanie strumienia zdarzeń	94
Współpartycjonowanie strumieni zdarzeń	95
Przykład: współpartycjonowanie strumienia zdarzeń	95

Przypisywanie partycji do instancji konsumenta	96
Mechanizm przypisywania partycji	96
Przypisywanie partycji współpartycjonowanych	96
Strategie przypisywania partycji	97
Odzyskiwanie sprawności po awariach bezstanowej instancji przetwarzania	99
Podsumowanie	99
6. Deterministyczne przetwarzanie strumieni	101
Determinizm w przepływach pracy opartych na zdarzeniach	102
Znaczniki czasu	102
Synchronizacja rozproszonych znaczników czasu	103
Przetwarzanie zdarzeń ze znacznikami czasu	104
Planowanie zdarzeń i przetwarzanie deterministyczne	105
Niestandardowe dyspozytory zdarzeń	106
Przetwarzanie na podstawie czasów: zdarzenia, przetwarzania i pozyskania	106
Wyodrębnianie znacznika czasu przez konsumenta	106
Wywołania typu żądanie-odpowiedź wysyłane do systemów zewnętrznych	107
Znaki wodne	107
Znaki wodne w przetwarzaniu równoległym	108
Czas strumienia	109
Czas strumienia w przetwarzaniu równoległym	110
Zdarzenia w niewłaściwej kolejności i zdarzenia opóźnione	111
Zdarzenia opóźnione ze znakami wodnymi i czasem strumienia	112
Przyczyny i skutki występowania zdarzeń w niewłaściwej kolejności	113
Funkcje zależne od czasu i okna czasowe	114
Obsługa zdarzeń opóźnionych	117
Porównanie ponownego przetwarzania i przetwarzania w czasie zbliżonym do rzeczywistego	118
Okresowe awarie i opóźnione zdarzenia	119
Problemy z łącznością producenta (brokera zdarzeń)	119
Podsumowanie i dalsza lektura	121
7. Stanowe przesyłanie strumieniowe	123
Magazyny stanu i materializowanie stanu ze strumienia zdarzeń	123
Rejestrowanie stanu w strumieniu zdarzeń dziennika zmian	124
Materializacja stanu w wewnętrznym magazynie stanu	125
Materializacja stanu globalnego	126
Zalety korzystania z wewnętrznego stanu	126
Wady korzystania z wewnętrznego stanu	128
Skalowanie i przywracanie wewnętrznego stanu	128

Materializacja stanu w zewnętrznym magazynie stanu	131
Zalety zewnętrznego stanu	132
Wady zewnętrznego stanu	132
Skalowanie i odzyskiwanie w przypadku zewnętrznych magazynów stanu	133
Porównanie odbudowywania i migrowania magazynów stanu	135
Odbudowa	135
Migracja	136
Transakcje i przetwarzanie efektywnie raz	136
Przykład: usługa rachunkowości zapasów	137
Przetwarzanie efektywnie raz za pomocą transakcji klient-broker	137
Przetwarzanie efektywnie raz bez transakcji klient-broker	139
Podsumowanie	144
8. Tworzenie przepływów pracy za pomocą mikrouslug	145
Wzorzec Choreografia	146
Prosty przykład choreografii opartej na zdarzeniach	147
Tworzenie i modyfikowanie choreografowanego przepływu pracy	148
Monitorowanie choreografowanego przepływu pracy	148
Wzorzec Orkiestracja	149
Prosty przykład orkiestracji opartej na zdarzeniach	149
Prosty przykład orkiestracji wywołań bezpośrednich	151
Porównanie orkiestracji bezpośrednich wywołań i orkiestracji opartej na zdarzeniach	151
Tworzenie i modyfikowanie przepływu pracy orkiestracji	152
Monitorowanie przepływu pracy orkiestracji	153
Transakcje rozproszone	153
Transakcje choreografowane: wzorzec Saga	153
Transakcje orkiestrowane	155
Kompensacyjne przepływy pracy	157
Podsumowanie	158
9. Mikrouslugi wykorzystujące funkcję jako usługę	159
Projektowanie rozwiązań opartych na funkcjach jako mikrouslug	159
Zapewnianie ścisłej przynależności do kontekstu ograniczonego	159
Zatwierdzanie przesunięć dopiero po zakończeniu przetwarzania	160
Mniej znaczy więcej	161
Wybór dostawcy FaaS	161
Budowanie mikrouslug na podstawie funkcji	161
Zimny start i ciepłe starty	163

Uruchamianie funkcji za pomocą wyzwalaczy	163
Wyzwalanie oparte na nowych zdarzeniach: nasłuchiwanie strumienia zdarzeń	163
Wyzwalanie oparte na opóźnieniu grupy konsumentów	165
Wyzwalanie według harmonogramu	166
Wyzwalanie przy użyciu webhooków	166
Wyzwalanie oparte na zdarzeniach dotyczących zasobów	167
Wykonywanie pracy biznesowej za pomocą funkcji	167
Utrzymywanie stanu	167
Funkcje wywołujące inne funkcje	168
Wzorzec komunikacji opartej na zdarzeniach	168
Wzorzec bezpośrednich wywołań	169
Zakończenie działania i zamknięcie	172
Dostrajanie funkcji	172
Alokacja wystarczających zasobów	172
Parametry wsadowego przetwarzania zdarzeń	173
Skalowanie rozwiązań FaaS	173
Podsumowanie	174
10. Mikrouслуги BPC	175
Gdzie sprawdzają się mikrouслуги BPC?	175
Integracja z istniejącymi i starszymi systemami	176
Stanowa logika biznesowa, która nie jest zależna od kolejności zdarzeń	177
Gdy warstwa danych wykonuje większość pracy	178
Niezależne skalowanie przetwarzania i warstwy danych	178
Hybrydowe aplikacje BPC z zewnętrznym przetwarzaniem strumieni	179
Przykład: użycie zewnętrznego frameworku przetwarzania strumieni do łączenia strumieni zdarzeń	180
Podsumowanie	181
11. Mikrouслуги frameworków ciężkich	183
Krótka historia ciężkich frameworków	184
Wewnętrzne działanie ciężkich frameworków	185
Korzyści i ograniczenia	186
Opcje konfiguracji klastra i tryby wykonywania	188
Użycie usługi hostowanej	188
Budowanie własnego pełnego klastra	188
Tworzenie klastrów z integracją z systemem CMS	189
Tryby zatwierdzania aplikacji	190
Tryb sterownika	190
Tryb klastra	191
Obsługa stanu i używanie punktów kontrolnych	191

Skalowanie aplikacji i obsługa partycji strumienia zdarzeń	192
Skalowanie aplikacji podczas jej działania	193
Skalowanie aplikacji przez jej ponowne uruchomienie	196
Automatycznie skalujące się aplikacje	196
Odzyskiwanie sprawności po awarii	196
Kwestie wielodzierżawności	197
Języki i składnia	197
Wybór frameworku	198
Przykład: tworzenie okna sesji kliknięć i wyświetleń	198
Podsumowanie	201
12. Mikrouslugi frameworków lekkich	203
Zalety i ograniczenia	203
Lekkie przetwarzanie	204
Obsługa stanu i używanie dzienników zmian	204
Skalowanie aplikacji i odzyskiwanie sprawności po awarii	205
Tasowanie zdarzeń	205
Przypisywanie stanu	206
Replikacja stanu i aktywne repliki	206
Wybór lekkiego frameworku	207
Apache Kafka Streams	207
Apache Samza: tryb osadzony	207
Języki i składnia	208
Łączenie strumień-tablica-tablica: wzorzec Wzbogacanie	208
Podsumowanie	212
13. Integracja mikrouslug opartych na zdarzeniach z mikrouslugami typu żądanie-odpowiedź	213
Obsługa zdarzeń zewnętrznych	213
Zdarzenia generowane autonomicznie	214
Zdarzenia generowane reaktywnie	214
Obsługa automatycznie generowanych zdarzeń analitycznych	215
Integracja z zewnętrznymi interfejsami API żądanie-odpowiedź	216
Przetwarzanie i udostępnianie danych stanowych	218
Obsługa żądań w czasie rzeczywistym za pomocą wewnętrznych magazynów stanu	218
Obsługa żądań w czasie rzeczywistym za pomocą zewnętrznych magazynów stanu	221
Obsługa żądań w przepływie pracy opartym na zdarzeniach	224
Przetwarzanie zdarzeń dla interfejsów użytkownika	225
Mikrofrontendy w aplikacjach typu żądanie-odpowiedź	231
Zalety mikrofrontendów	233
Mikrouslugi oparte na kompozycji	233
Łatwe dostosowywanie do wymagań biznesowych	233

Wady mikrofrontendów	233
Potencjalnie niespójne elementy i style interfejsu użytkownika	234
Zmienna wydajność mikrofrontendu	234
Przykład: aplikacja do wyszukiwania i recenzowania wydarzeń	234
Podsumowanie	237
14. Narzędzia pomocnicze	239
System przypisywania mikrousług zespołom	239
Tworzenie i modyfikowanie strumienia zdarzeń	240
Znakowanie strumieni zdarzeń za pomocą metadanych	240
Kwoty	241
Rejestr schematów	241
Powiadomienia o tworzeniu i modyfikowaniu schematów	243
Zarządzanie przesunięciami	243
Uprawnienia i listy kontroli dostępu dla strumieni zdarzeń	244
Zarządzanie stanem i resetowanie aplikacji	245
Monitorowanie opóźnienia przesunięcia konsumenta	246
Zoptymalizowany proces tworzenia mikrousług	246
Kontrola zarządzania kontenerami	247
Tworzenie klastra i zarządzanie nim	247
Programowe uruchamianie brokerów zdarzeń	248
Programowe uruchamianie zasobów obliczeniowych	248
Replikacja danych zdarzeń między klastrami	249
Programowe uruchamianie narzędzi	249
Śledzenie zależności i wizualizacja topologii	249
Przykład topologii	251
Podsumowanie	253
15. Testowanie mikrousług opartych na zdarzeniach	255
Ogólne zasady testowania	255
Funkcje topologii testów jednostkowych	256
Funkcje bezstanowe	256
Funkcje stanowe	256
Testowanie topologii	257
Testowanie ewolucji i zgodności schematów	258
Testowanie integracyjne mikrousług opartych na zdarzeniach	258
Lokalne testy integracyjne	259
Tworzenie tymczasowego środowiska w ramach wykonywania kodu testowego	261
Tworzenie tymczasowego środowiska zewnętrznym względem kodu testowego	262
Integracja usług hostowanych przy użyciu opcji atrapy i symulatora	263
Integracja usług zdalnych, które nie mają opcji lokalnych	263

Pełne zdalne testy integracyjne	264
Programowe tworzenie tymczasowego środowiska testów integracyjnych	265
Testowanie przy użyciu środowiska współdzielonego	267
Testowanie przy użyciu środowiska produkcyjnego	268
Wybór strategii w pełni zdalnych testów integracyjnych	269
Podsumowanie	270
16. Wdrażanie mikrousług opartych na zdarzeniach	271
Zasady wdrażania mikrousług	271
Architektoniczne komponenty wdrażania mikrousług	272
Systemy ciągłej integracji, ciągłego dostarczania i ciągłego wdrażania	272
Systemy zarządzania kontenerami i powszechnie dostępny sprzęt	273
Podstawowy wzorzec Wdrożenie z Pełnym Zatrzymaniem	274
Wzorzec Aktualizacja Krocząca	275
Wzorzec Przełomowa Zmiana Schematu	276
Ostateczna migracja za pośrednictwem dwóch strumieni zdarzeń	277
Zsynchronizowana migracja do nowego strumienia zdarzeń	278
Wzorzec Wdrożenie Niebiesko-Zielone	279
Podsumowanie	280
17. Zakończenie	281
Warstwy komunikacyjne	281
Dziedziny biznesowe i konteksty ograniczone	282
Współużytkowanie narzędzi i infrastruktury	282
Uschematyzowane zdarzenia	282
Wyzwolenie danych i jedno źródło prawdy	283
Mikrousługi	284
Opcje implementacji mikrousług	284
Testowanie	285
Wdrażanie	285
Kilka słów na zakończenie	286

Dlaczego mikrouслуги 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 organizacje, 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 mikrouslug i zarządzanie nimi. Mikrouслуги 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 mikrouслуги oparte na zdarzeniach?

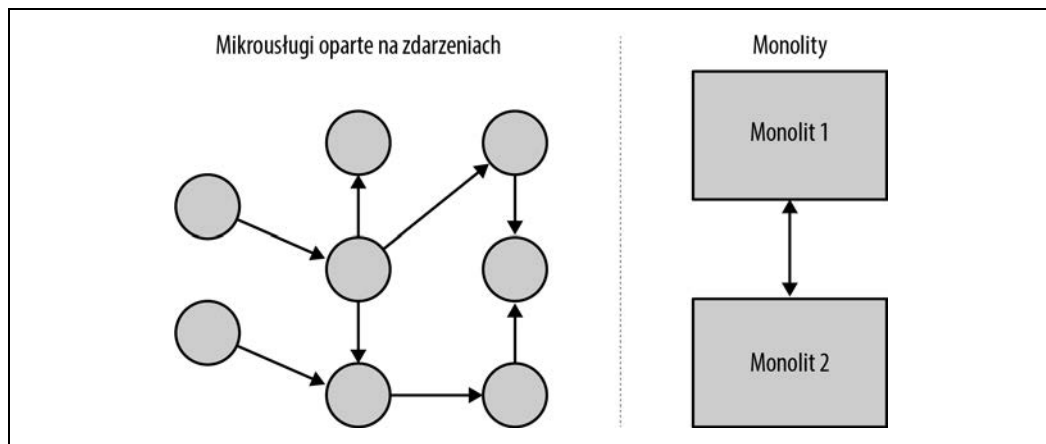
Mikrouслуги i architektury mikrouslugowe istnieją od lat w mnogich formach, pod wieloma odmiennymi nazwami. Architektury usługowe (ang. *service-oriented architecture* — SOA) często składają się z wielu mikrouslug, które komunikują się synchronicznie bezpośrednio ze sobą. Architektury opierające się 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 pomóc 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 (konceptyjnie) „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 działanie, 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 wchodzi 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

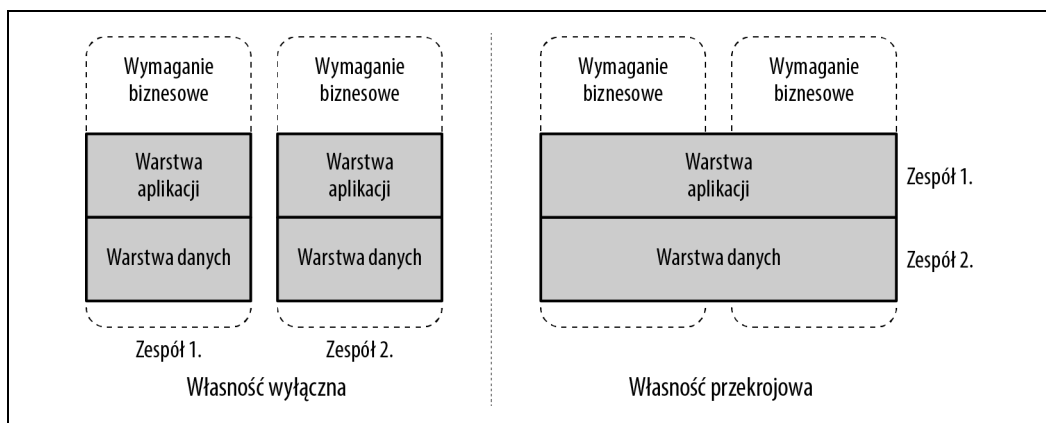
Wymagania biznesowe produktu często zmieniają się w trakcie jego życia, na przykład z powodu zmian organizacyjnych lub zapotrzebowania na nowe funkcjonalności. Natomiast rzadko zdarza się, aby firma musiała zmieniać bazową implementację jakiegokolwiek produktu bez towarzyszących temu procesowi zmian 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 wzorec 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 ściśle 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,

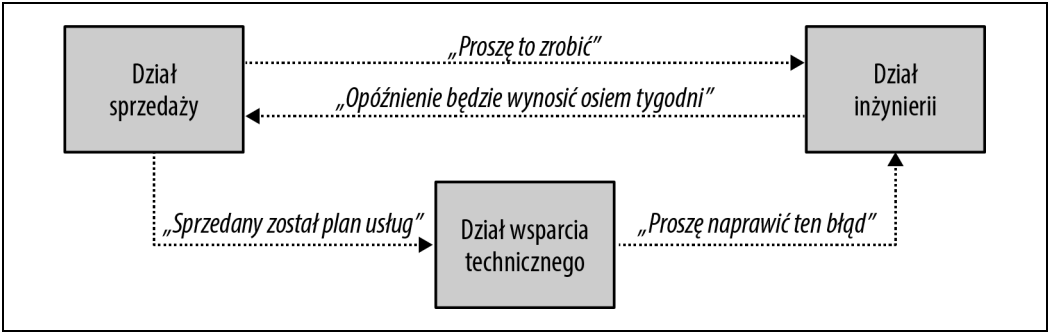
podczas gdy bardziej wyspecjalizowane zestawy umiejętności mają być zapewniane w ramach projektów międzyzespołowych, w zależności od potrzeb. Te najlepsze praktyki zostały opisane szczegółowo w rozdziale 14.

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

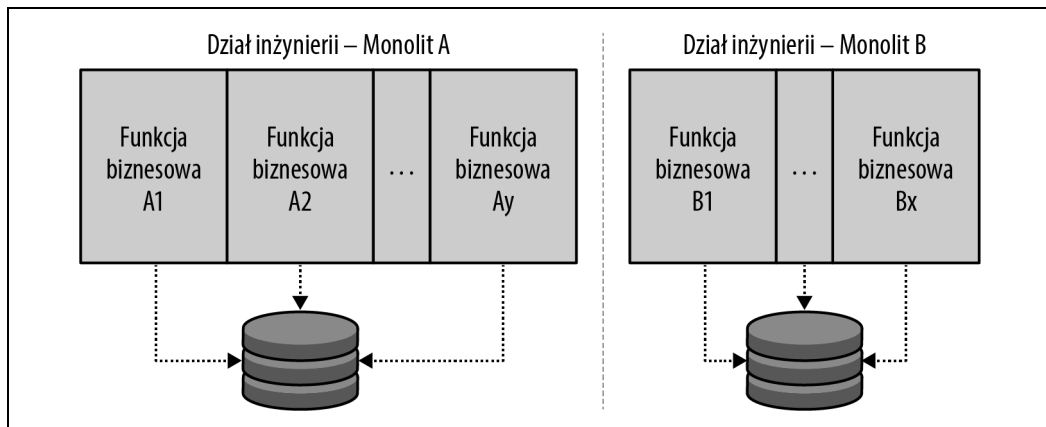
Struktura komunikacji biznesowej (zobacz rysunek 1.3) dyktuje komunikację między zespołami i działami firmy, z których każdy kieruje się głównymi wymaganiami i przypisanymi mu obowiązkami. Dział inżynierii produkuje na przykład oprogramowanie, dział sprzedaży zajmuje się sprzedawaniem produktu klientom, 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 upływem 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 przededefiniowanie 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

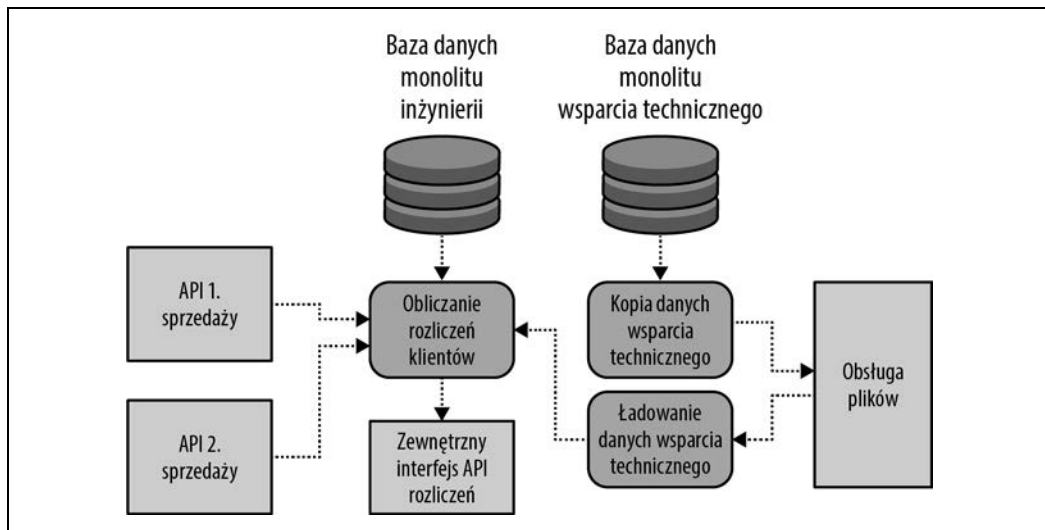
Struktura komunikacji danych (zobacz rysunek 1.5) to proces, za pomocą którego dane są przekazywane w całej firmie, a zwłaszcza między implementacjami. Chociaż struktura komunikacji danych obejmująca pocztę elektroniczną, komunikatory i zebrania jest często używana do informowania o zmianach biznesowych, 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 jakiegokolwiek 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 pozyskiwaniem danych z innych systemów.

Kontynuacja scenariusza z zespołami

Założmy, 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 zmieniają się na tyle, że zespoły nie będą już w stanie ich w pełni 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 przysyłanie 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 przysyłać 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 przysyłania danych z jednej implementacji do drugiej. Działają raczej zarówno jako magazyn danych, jak i środek asynchronicznej komunikacji między usługami.

Strumień 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 producenci, jak i konsumenci zostają zwolnione z obowiązku zapewniania mechanizmów kwerendowania, mechanizmów przysyłania danych, interfejsów API (ang. *Application Programming Interface*) i międzypespółowych 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 udostępniane 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 dziedziczne, które mają kluczowe znaczenie dla działania firmy. Chociaż zespoły mogą się restrukturyzować, a projekty mogą się rozpoczynać i kończyć, ważne podstawowe dane dziedziczne pozostają łatwo dostępne dla każdego nowego produktu, który ich wymaga, *niezależnie od jakiegokolwiek konkretnej struktury komunikacji implementacyjnej*. Daje to firmie niespotykaną elastyczność, ponieważ dostęp do podstawowych zdarzeń dziedzicznych 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.

Łuźne powiązania

Mikrousługi oparte na zdarzeniach są powiązane z danymi dziedzicznymi, 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 strumienia, 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 przenieszone 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 lekturę¹, abyś mógł lepiej zrozumieć podejście synchroniczne.

Ponadto należy pamiętać, że żadne z tych typów usług nie są zasadniczo lepsze jedno 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 istniejącą strukturę komunikacyjną 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. Komplikuja 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 najnowszej wersji 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. Klienci 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 mikrouslugach 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 dziedzicznych (tak jak w systemach opartych na zdarzeniach) umożliwia stosowanie mniejszych, specjalnie zaprojektowanych implementacji.

PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

Mikrouслуги: odblokuj potencjał danych!

Mikrouслуги 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 mikrouslug opartych na zdarzeniach. Dowiesz się także, jak przygotować i przeprowadzić proces budowania organizacji, która wykorzystuje mikrouslugi 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 mikrouslug i wdrażania ich w środowisku produkcyjnym.

W książce między innymi:

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

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

Sprawdź nasze szkolenia!

SZKOLENIA

AKADEMIA IT & BUSINESS
HELIONSZKOLENIA.PL

KOD KORZYŚCI
Sięgnij po więcej! ▶



ISBN 978-83-283-7439-3



INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 69,00 zł