O'REILLY®

Siatka danych

Nowoczesna koncepcja samoobsługowej infrastruktury danych

Helion

Zhamak Dehghani

Tytuł oryginału: Data Mesh: Delivering Data-Driven Value at Scale

Tłumaczenie: Lech Lachowski

ISBN: 978-83-8322-037-6

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

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

ROZDZIAŁ 9.
Architektura logiczna

Forma wynika z funkcji.

— Louis Sullivan

W tym rozdziale przedstawię logiczną architekturę siatki danych, w tym jej wysokopoziomowe komponenty strukturalne i ich relacje.

Aby dotrzeć do architektury, przeprowadzę Cię przez zasady siatki danych i pokażę, jak każda z tych zasad wpływa na ogólną architekturę za pomocą nowych komponentów i integracji — *aby dotrzeć do formy, podążamy za funkcją i zamiarem*.

Oto oparte na poszczególnych zasadach krótkie podsumowanie koncepcji architektonicznych, które omówię w tym rozdziale:

Własność dziedzinowa rozszerza dziedziny o interfejsy udostępniania danych analitycznych

Z dziedzinowej własności danych wynika oparta na dziedzinach organizacja danych analitycznych. Oznacza to, że interfejsy dziedziny muszą uwzględniać udostępnianie jej danych analitycznych. Notacyjnie zobrazowałam to rozszerzenie na rysunku 9.1.

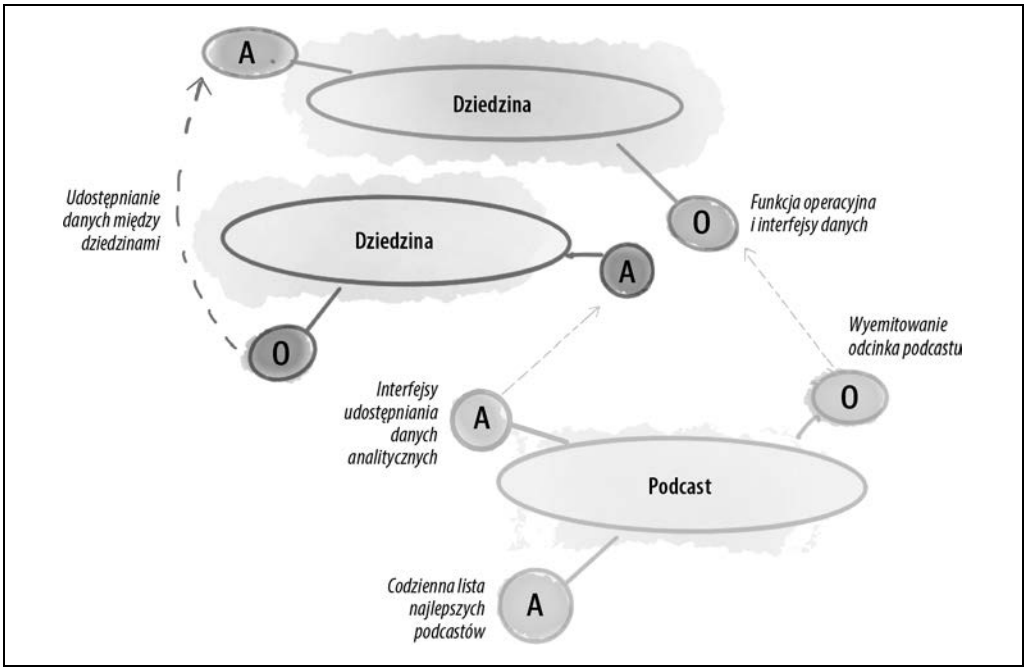
Dane jako produkt wprowadzają nowy kwant architektury quantum, czyli kwant danych

Siatka danych reprezentuje każdy produkt danych jako kwant architektury. Będąc kwantem architektonicznym, produkt danych hermetyzuje wszystkie komponenty niezbędne do zaimplementowania jego cech użyteczności i zachowania bezpiecznego udostępniania danych analitycznych.

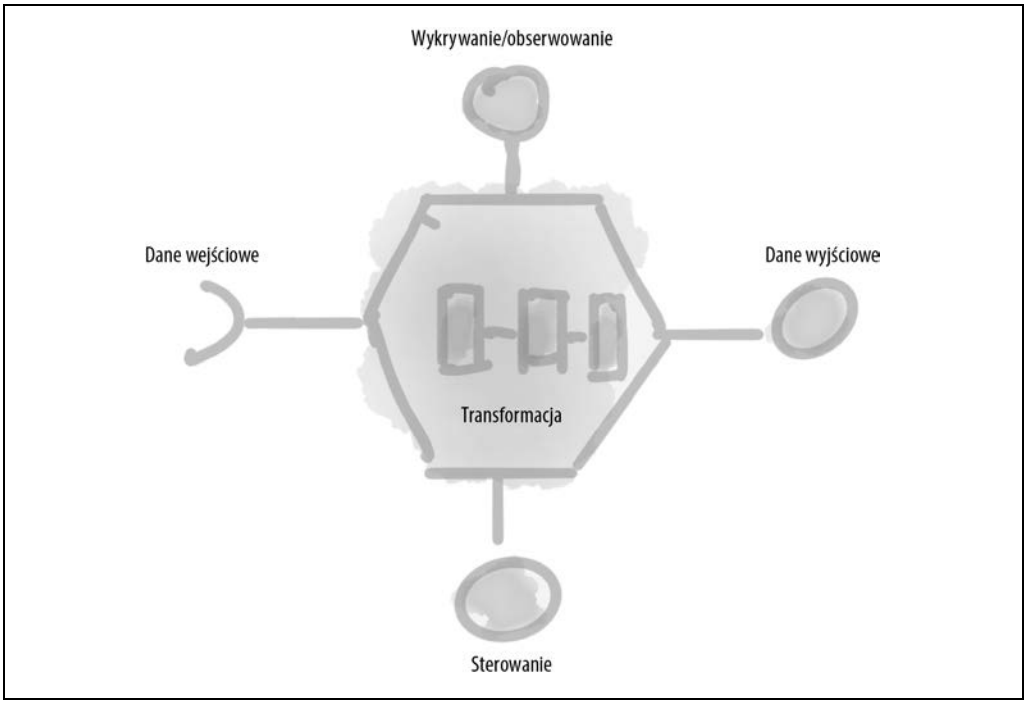
Kwant danych jest podstawową jednostką architektury siatki danych. Przechowuje dane dziedzinowe i kod, który przeprowadza niezbędne transformacje danych i udostępnia dane oraz reguły rządzące danymi. Główne składniki kwantu danych przedstawiłam na rysunku 9.2.

Samoobsługowa platforma danych wprowadza wielopłaszczyznową architekturę platformy

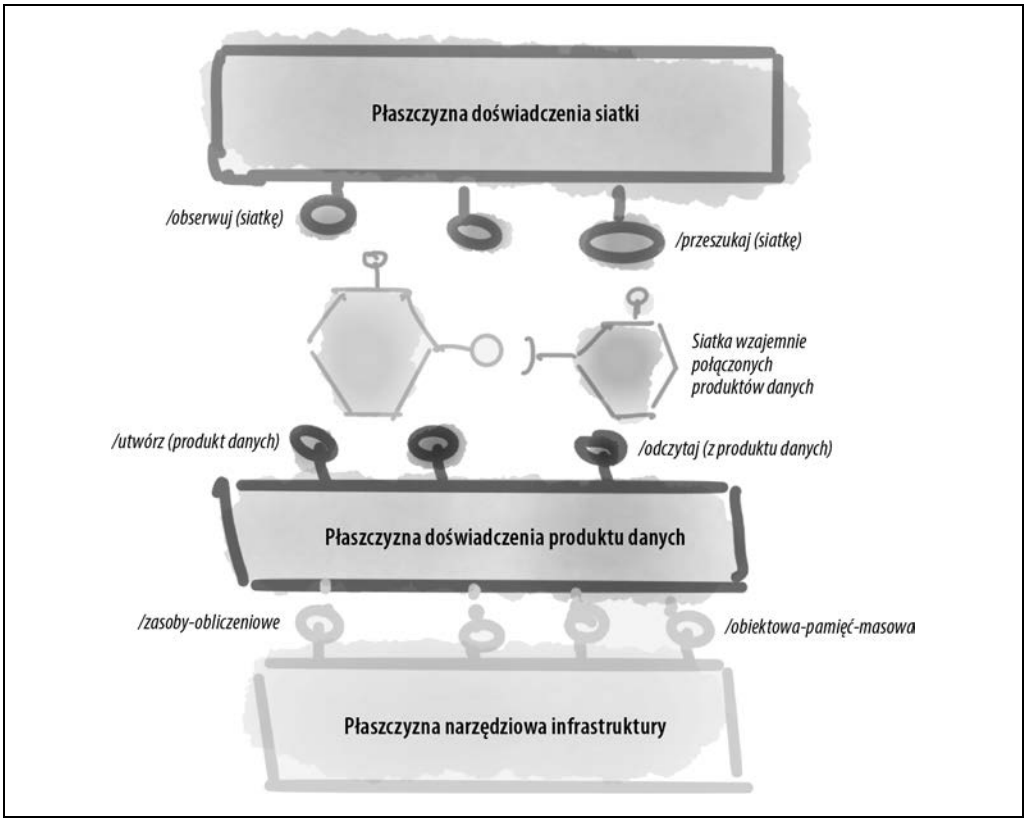
Platforma samoobsługowa oferuje proliferację usług, które umożliwiają wykonywanie pracy szerokiemu spektrum użytkowników siatki danych: producentom danych (twórcom produktów danych i właścicielom produktów danych), konsumentom danych (analitykom danych i badaczom danych) oraz członkom zespołu zarządzania danymi. Siatka danych projektuje tę platformę jako trzy płaszczyzny — grupy współpracujących usług zorganizowane na podstawie doświadczenia użytkowników siatki danych. Płaszczyzny platformy pokazałam na rysunku 9.3.



Rysunek 9.1. Interfejsy dziedzin rozszerzone o interfejsy udostępniania danych analitycznych



Rysunek 9.2. Nowa jednostka architektury — kwant (produktu) danych



Rysunek 9.3. Wielopłaszczyznowa platforma danych z interfejsami deklaratywnymi

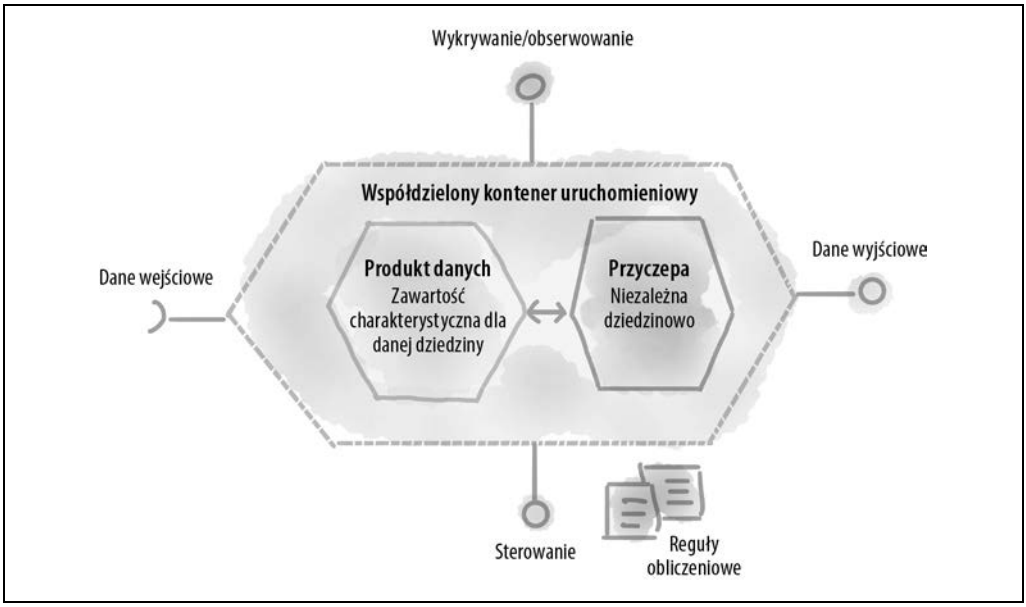
Sfederowane zarządzanie obliczeniowe osadza reguły obliczeniowe w każdym produkcie danych

Zasada sfederowanego zarządzania obliczeniowego prowadzi do rozszerzenia każdego produktu danych o kontener obliczeniowy. Może on hostować proces **przyczepy** (ang. *sidecar*), który osadza reguły obliczeniowe w tym produkcie danych i egzekwuje je w odpowiednim momencie i w przepływie danych, np. budowania, wdrażania, uzyskiwania dostępu, odczytywania lub zapisywania.

Przyczepa produktu danych to proces, który współdzieli kontekst wykonywania produktu danych i jest odpowiedzialny jedynie za kwestie niezależne dziedzinowo i przekrojowe, takie jak wykonywanie reguł. Oprócz egzekwowania reguł proces przyczepy może zostać rozszerzony w celu implementowania innych znormalizowanych funkcjonalności produktu danych, takich jak wykrywanie.

Implementacja przyczepy jest wspólna dla wszystkich produktach danych.

Koncepcję przyczepy produktu danych współdzielącego kontener obliczeniowy (kontekst) ze swoim produktem przedstawiłam na rysunku 9.4.



Rysunek 9.4. Produkt danych rozszerzony o przyczepę do osadzania konfiguracji i egzekwowania reguł

W momencie, gdy piszę ten rozdział, istnieje wiele prywatnych implementacji tych komponentów architektonicznych, nad którymi pracowałam razem z klientami. Każda z tych implementacji jest inna. W związku z tym nie ma jeszcze de facto publicznie dostępnych standardów ani referencyjnych implementacji. Mam nadzieję, że zmieni się to w nadchodzących latach.

Przyjrzyjmy się bliżej każdemu z tych elementów architektonicznych.

Dziedzinowe interfejsy udostępniania danych analitycznych

Aby promować dziedzinową dekompozycję własności danych, musimy wymodelować architekturę, która uporządkuje dane analityczne[1] według dziedzin. W tej architekturze interfejs dziedziny, który jest przeznaczony do użytku przez pozostałą część organizacji, obejmuje nie tylko funkcjonalności operacyjne, ale udostępnia także dane analityczne generowane i posiadane przez tę dziedzinę.

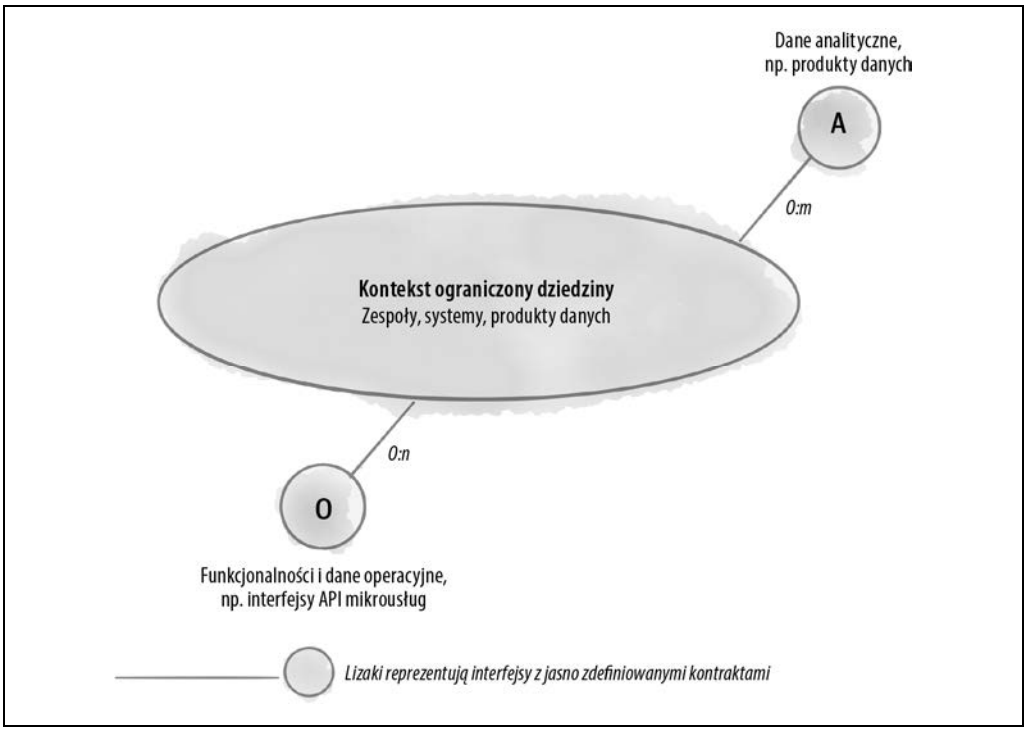
Dziedzina ***podcastu*** zapewnia operacyjne interfejsy API do semantycznego tworzenia nowego odcinka podcastu, ale także interfejs danych analitycznych do pobierania wszystkich danych epizodów podcastu i ich statystyk użytkowania dla długiego przedziału czasu.

Każda dziedzina kontroluje swoje dane — operacyjne i analityczne. Przyjmuję założenie, że dziedziny mają już interfejsy operacyjne, jakimi są interfejsy API aplikacji. Nie jest to coś, co wprowadza siatka danych, ale integruje się ona z operacyjnymi interfejsami API. Oprócz operacyjnych API dziedziny kontrolują swoje interfejsy API udostępniania danych analitycznych i nimi zarządzają.

[1] Definicje danych analitycznych i danych operacyjnych podałam w rozdziale 1.

Aby zapewnić skalowanie, architektura musi wspierać autonomię zespołów dziedzinowych w zakresie wydawania i wdrażania ich aplikacji operacyjnych i produktów danych analitycznych.

Notację graficzną, której używam do wyrażenia, że każda dziedzina może udostępniać co najmniej po jednym interfejsie operacyjnym i analitycznym, przedstawiłam na rysunku 9.5.



Rysunek 9.5. Notacja: dziedzina oraz jej interfejsy analityczne i operacyjne

Przyjrzyjmy się bliżej każdemu z tych interfejsów i ich zależnościom.

Projekt interfejsu operacyjnego

Obecnie dziedziny implementują swoje funkcjonalności operacyjne za pomocą szeregu rozwiązań. Dziedzina może mieć np. aplikacje GUI skierowane do klienta, aplikacje bezinterfejsowe, starsze systemy, produkty SaaS, mikrousługi lub oparte na zdarzeniach funkcje jako usługi.

Niezależnie od rodzaju rozwiązań każda dziedzina pod względem logicznym służy swoim użytkownikom — systemom lub ludziom, wewnętrznym lub zewnętrznym w stosunku do organizacji — za pośrednictwem zestawu interfejsów.

W przypadku mikrousługi implementującej funkcję dziedzinową interfejs dziedziny obejmuje interfejsy API, np. GraphQL, RESTful, gRPC itd. — które są udostępniane przez tę mikrousługę. Na rysunku 9.5 te interfejsy dziedzin zostały przedstawione za pomocą lizaka *O*.

Pod względem semantycznym interfejsy operacyjne implementują funkcje, np. wynagrodzenia dla artysty lub subskrybowanie słuchacza. Syntaktycznie funkcje te mogą być zaimplementowane jako operacje na zasobach deklaratywnych, które są modyfikowane za pomocą operacji CRUD, np. `http POST/wynagrodzenia-artystów`.

Interfejsy operacyjne są zwykle projektowane w celu uzyskiwania dostępu do danych o mniejszej objętości i zapewniają bieżący, zbliżony do rzeczywistego obraz stanu znanego systemowi. Operacyjne API usługi ***subskrypcji słuchaczy*** może mieć np. afordancję pobierania listy (teraz już z paginacją) subskrybujących słuchaczy w Ameryce Północnej, np. `http GET/subskrypcje-słuchaczy?region=NA`.

Siatka danych nie zmienia tego wcześniej istniejącego aspektu architektury przedsiębiorstwa; po prostu uznaje jego istnienie. W niektórych przypadkach siatka danych wykorzystuje interfejsy operacyjne do tworzenia produktów danych.

Projekt interfejsu danych analitycznych

Wraz z rozszerzeniem odpowiedzialności dziedzin za udostępnianie danych analitycznych wprowadzony został nowy zestaw interfejsów. Na rysunku 9.5 interfejsy te są oznaczone jako *A*.

Interfejsy analityczne to API, które produkty danych udostępniają w celu umożliwienia *odkrywania*, *zrozumienia*, *obserwowania* oraz *współdzielenia* swoich danych. W chwili, gdy piszę ten rozdział, nie ma żadnych powszechnie przyjętych konwencji dotyczących wysokopoziomowych interfejsów API hermetyzujących wszystkie te funkcje.

W przypadku udostępniania danych analitycznych interfejsy API po wyegzekwowaniu reguł dostępu mogą decydować o przekierowywaniu klientów do bazowej pamięci masowej danych. W zależności od implementacji produktu danych interfejs udostępniania danych może przekierować dostęp do obiektowej pamięci masowej (np. plików Parquet w AWS S3), strumienia zdarzeń (np. tematów Kafki), tabeli (np. tabeli BigQuery) lub czegoś zupełnie innego.

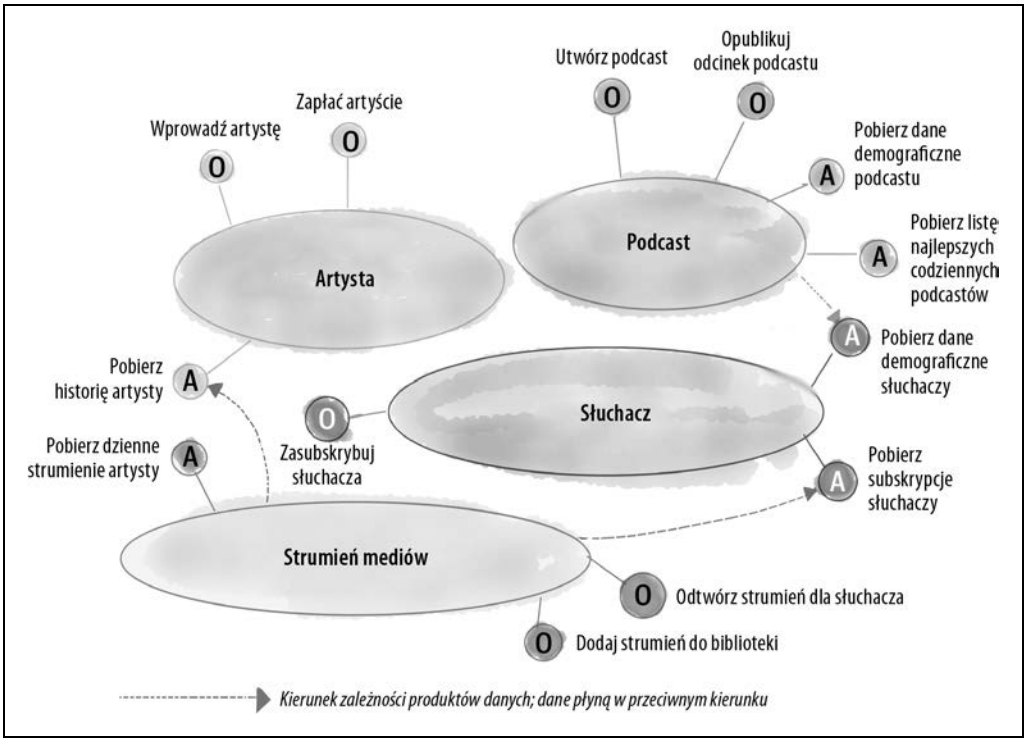
Międzydziedzinowe zależności danych analitycznych

Oczywiście każda dziedzina może mieć zależności od interfejsów danych operacyjnych i analitycznych innych dziedzin.

W poniższym przykładzie (zobacz rysunek 9.6) każda dziedzina zapewnia kilka interfejsów operacyjnych i analitycznych. Dziedzina ***podcastu*** udostępnia np. kilka operacyjnych interfejsów API do tworzenia lub publikowania odcinków podcastu. Dostarcza również danych analitycznych, takich jak ***dane demograficzne (słuchacza) podcastu*** i ***najlepsze codzienne podcasty***.

Każda dziedzina ma zależności. Dziedzina ***podcastu*** konsumuje np. informacje o ***danych demograficznych słuchaczy*** z dziedziny ***słuchacza***, aby przedstawić obraz demograficzny słuchaczy podcastu.

Mechanizm ustalania zależności — asynchroniczne używanie zdarzeń lub pobieranie synchroniczne — jest kwestią implementacji. Ważne jest jednak, że produkt danych wyraźnie definiuje zależności od swoich źródeł.



Rysunek 9.6. Przykład: dziedzinowe zależności danych

W architekturze siatki danych zależność danych jest definiowana i kontrolowana przez produkt danych w kierunku do upstreamowego źródła. Produkt danych ma pełną kontrolę nad tym, jakie są jego źródła, jakie dane z nich pobiera i w jaki sposób.

W tym przykładzie etykiety, których użyłam dla interfejsów, definiują semantykę interfejsu, a nie jego składnię. Pod względem syntaktycznym semantyka ***zapłać artyście*** może zostać zaimplementowana chociażby jako zasoby deklaratywne — zasoby RESTful lub zapytanie GraphQL — np. jako ***wynagrodzenia-artystów***.

Produkt danych jako kwant architektury

Kwant architektoniczny (*https://oreil.ly/Lrd6t*) zdefiniowany w książce *Architektura ewolucyjna. Projektowanie oprogramowania i wsparcie zmian* (Helion) jest najmniejszą jednostką architektury, która może być wdrażana niezależnie, ma wysoką spójność funkcyjną i obejmuje wszystkie „elementy strukturalne wymagane przez jego funkcję".

W przypadku siatki danych kwantem architektonicznym jest produkt danych. To najmniejsza jednostka architektury, którą można niezależnie wdrożyć i zarządzać. Ma wysoką spójność funkcyjną, tj. przeprowadzania określonej transformacji analitycznej i bezpiecznego udostępniania wyniku jako dziedzinowych danych analitycznych. Zawiera wszystkie komponenty strukturalne,

których potrzebuje do pełnienia swojej funkcji: kod transformacji, dane, metadane, reguły rządzące danymi i zależności od infrastruktury.

W stricte architektonicznych dywagacjach zawartych w tym rozdziale mogę skrócić określenie **„produkt danych (jako kwant architektury)”** do postaci **„kwant produktu danych”** lub — z lenistwa — nawet do formy **„kwant danych”**. Wszystkie te pojęcia można stosować wymiennie.

W przypadku siatki danych kwant danych to uznany sposób projektowania architektury produktu danych.

Wiedza o tym, dlaczego wybrałam pozornie trudne i dla wielu nieznane wyrażenie **„kwant danych”**, może okazać się pomocna. Dlaczego nie „usługa danych”, „aktor danych”, „agent danych”, „operator danych” czy chociażby „transformator danych”? Wszystkie te sformułowania mogą wydawać się właściwe. Większość określeń, takich jak operator, transformator itp., mocno podkreśla pojedynczy aspekt kwantu danych, czyli jego zadanie i wykonywanie transformacji. Podobnie wyrażenie „usługa danych” kładzie nacisk na inny aspekt, jakim jest serwowanie danych. Pojęcie kwantu, wprowadzone w pozycji *Architektura ewolucyjna. Projektowanie oprogramowania i wsparcie zmian*, a następnie rozwinięte w książce *Złożone zagadnienia architektury oprogramowania. Jak analizować kompromisy i podejmować trudne decyzje* (Helion), doskonale ucieleśnia intencję kwantu danych, czyli wszystko, co jest wymagane do autonomicznego wykonywania jego zadania — niezależnie od tego, jakie ono będzie. Mam nadzieję, że akceptujesz moje uzasadnienie dla wprowadzenia tego neologizmu do mocno już rozbudowanego słownictwa technicznego.

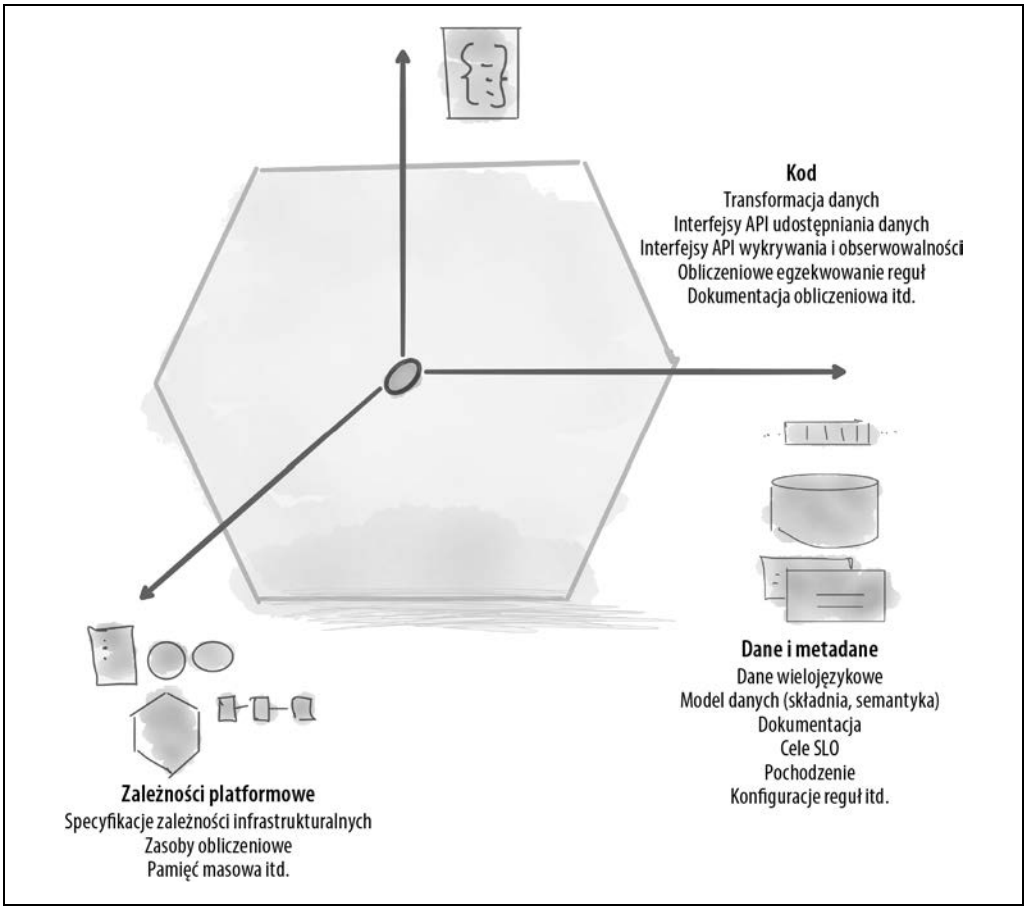
Kwant architektury to oś, wzdłuż której system może się poziomo skalować. Siatka danych jest skalowana poziomo przez dodawanie i łączenie kolejnych produktów danych. Architektura kwantowa integruje się z innymi komponentami za pośrednictwem powiązań statycznych (np. bibliotek kompilacji) lub powiązań dynamicznych (takich jak asynchroniczne interfejsy API środowiska wykonawczego). Produkt danych integruje się z innymi produktami danych przez dynamiczne powiązania interfejsów udostępniania danych.

Różne aspekty produktu danych jako kwantu architektury omówię szczegółowo w części IV.

Komponenty strukturalne produktu danych

Produkt danych hermetyzuje nie tylko dane. Musi zawierać wszystkie komponenty strukturalne potrzebne do autonomicznej manifestacji swoich podstawowych cech użyteczności — wykrywalności, zrozumiałości, adresowalności itd. — przy jednoczesnej kontynuacji udostępniania danych w sposób zgodny i bezpieczny.

Zasadniczo produkty danych mają trzy rodzaje komponentów strukturalnych: **kod**, **dane (oraz metadane i konfigurację)** oraz specyfikacje swoich **zależności infrastrukturalnych** (zobacz rysunek 9.7).



Rysunek 9.7. Rodzaje komponentów strukturalnych produktu danych

Kod

Aby produkt danych mógł samodzielnie kontrolować cykl życia danych analitycznych — utrzymywać generującą je logikę biznesową, kontrolować ich wersje, zarządzać dostępem do nich i udostępniać ich zawartość — musi zawierać kod i uruchamiać go w swoim kontekście obliczeniowym. Jest to zasadnicza różnica między tym, co siatka danych nazywa produktem danych, a innymi formami artefaktów danych. Produkt danych jest aktywny, podczas gdy inne artefakty danych, takie jak pliki lub tabele, są pasywne.

Przyjrzyjmy się kilku różnym typom kodów kontrolowanych przez produkt danych.

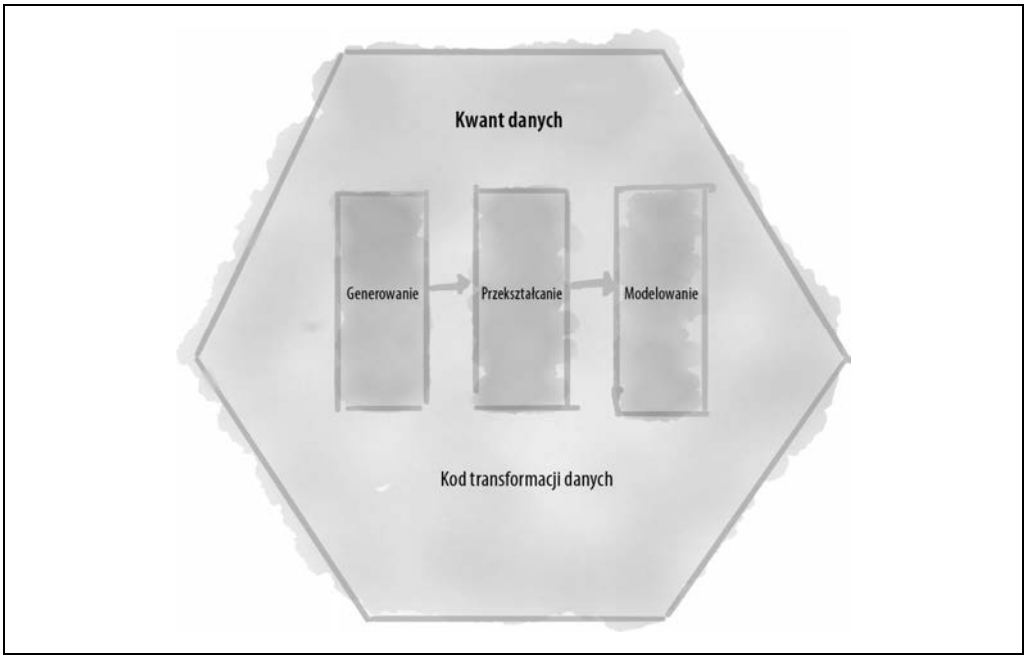
Transformacja danych jako kod Produkty danych przekształcają dane otrzymywane z upstreamowego źródła, np. z ich sąsiedniego systemu operacyjnego, lub same generują dane. Tak czy inaczej, w celu generowania i udostępniania danych trzeba przeprowadzać obliczenia analityczne.

Dziedzina ***demografii podcastu*** konsumuje np. ***demografię słuchaczy*** i ***odtwarzane (przez słuchaczy) podcasty***. Jej analityczny kod transformacji agreguje dane z obu tych dziedzin, wywodzi i dodaje inne informacje o demografii słuchaczy na podstawie ich zachowań dotyczących słuchania podcastów, takich jak kategoria słuchanych podcastów, a następnie udostępnia wynik jako ***demografię podcastu***. Kod ten działa w sposób ciągły wraz z odtwarzaniem kolejnych podcastów.

W tradycyjnych architekturach kod ten rezyduje poza produktem danych jako **potok danych**. Potok jest zarządzany jako niezależny artefakt, np. jako konfiguracja skierowanego grafu cyklicznego. Jego dane wyjściowe są zarządzane niezależnie, np. jako tabela w hurtowni.

Siatka danych usuwa koncepcję *zewnętrznego* potoku i wprowadza *wewnętrzny* kod transformacji. Kod ten może być zaimplementowany jako potok, ale nie musi. Ważnym odróżnieniem jest to, że kod jest zhermetyzowany jako wewnętrzna implementacja produktu danych. Jego cykl życia jest kontrolowany przez produkt danych i jest on uruchamiany w kontekście wykonywania przypisanym do tego produktu danych.

Kod transformacji jest charakterystyczny dla dziedziny i hermetyzuje takie zadania jak *dziedzinowa logika biznesowa oraz agregowanie i modelowanie danych*. Temu kodowi programiści danych poświęcają większość czasu i uwagi. Notację, której używam w tej książce do demonstrowania kodu transformacji produktu danych, pokazałam na rysunku 9.8.



Rysunek 9.8. Kod transformacji produktu danych

Ten kod obejmuje zarówno implementację, jak i zautomatyzowane testy, które ją weryfikują.

Zwróć uwagę, że w kodzie transformacji pominęłam tradycyjne etapy potoku danych, takie jak *przyswajanie danych* lub *oczyszczanie danych*. Ma to na celu podkreślenie różnicy między siatką danych a tradycyjnymi potokami.

Oczyszczanie danych jest prawie zawsze obowiązkiem źródła upstreamowego, np. produktów danych. Zapewnia integralność danych. W związku z tym typowe czynności oczyszczania są rzadko wymagane w kodzie produktu danych. Tradycyjnie czynności oczyszczania danych w potokach obejmują obsługę niekompletnych, nieprawidłowych, niedokładnych lub nieistotnych danych. Oczekiwanie, że dane będą „zabrudzone", jest normą. W siatce danych upstreamowe produkty danych dostarczają oczyszczone dane, zgodnie z ich gwarancjami. W przypadku ***danych demograficznych słuchaczy*** produkt danych gwarantuje jakość pod względem kompletności, integralności i terminowości danych. Interfejsy udostępniania danych określają te gwarancje i o nich informują. Otrzymywanie niespodzianek w postaci zabrudzonych danych to błąd — wyjątek, a nie norma. Te gwarancje są najprawdopodobniej wystarczająco dobre dla ***demografii podcastów***, które konsumują te dane do dalszej agregacji. Dlatego nie wymagają dalszego oczyszczania.

W niektórych przypadkach poziom gwarancji może być niższy niż wymagany przez downstreamowy produkt danych, a wtedy jego kod transformacji może obejmować oczyszczanie. Produkt danych ***subskrypcji słuchaczy*** generuje np. zdarzenia subskrypcji w czasie zbliżonym do rzeczywistego bez informacji o płatnościach, ponieważ Daff obsługuje subskrypcję asynchronicznie w stosunku do rzeczywistych płatności. W takiej sytuacji można wyobrazić sobie produkt danych ***płatności za subskrypcje***, który musi skonsolidować zdarzenia z informacjami o płatnościach, aby utworzyć pełny obraz subskrypcji. Nawet w takim przypadku nie nazywam ***subskrypcji słuchaczy*** brudnymi danymi. Jest to całkowicie rozsądny produkt danych dla wielu innych przypadków użycia, nawet przy brakujących informacjach. Brakujące informacje o płatności są częścią kontraktu. Jeśli okaże się, że będziesz musiał zbudować zadanie oczyszczania, sugeruję sprawdzić, czy poprawy nie wymaga coś upstreamowego.

Jeśli chodzi o konsumowanie, traktuję tę część jako funkcję *wejściową* produktu danych, a nie jako jego kod transformacji. Omówię to dalej w rozdziale.

Interfejsy jako kod Produkt danych zapewnia dostęp do swoich danych, informacji o wykrywalności, dokumentacji użyteczności, wskaźników obserwowalności, celów SLO itp., za pośrednictwem swoich **interfejsów** (API). Te interfejsy API przestrzegają zdefiniowanych **kontraktów** — umów dotyczących tego, jakie informacje przekazuje produkt danych i w jaki sposób.

Do zaimplementowania takich interfejsów wymagany jest kod związany z dostarczaniem niezbędnych informacji i ich serwowaniem. Notacja lizakowa hermetyzuje interfejs i obsługujący go kod. Listę tych interfejsów pokazałam na rysunku 9.9:

Interfejsy API danych wyjściowych

Istnieje zestaw współpracujących ze sobą interfejsów API, które udostępniają dane wyjściowe produktu danych w zrozumiały i godny zaufania sposób.

Interfejsy API danych wynikowych mogą odbierać i uruchamiać zdalne kwerendy swoich danych, np. interfejs do uruchamiania zapytań SQL na bazowych tabelach zawierających dane.

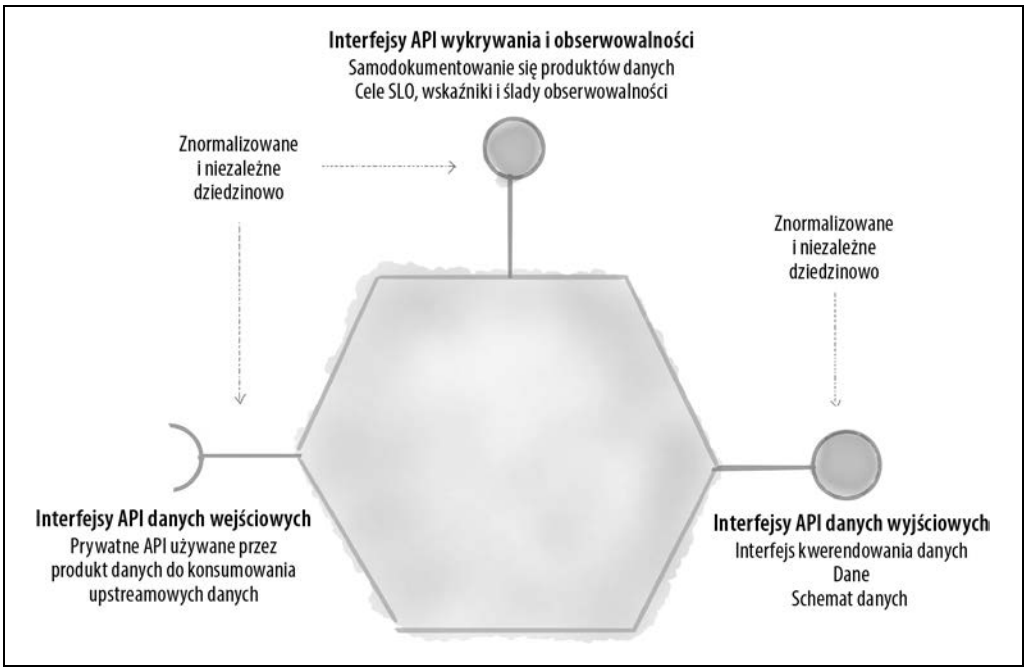
Wyjściowe API udostępniają dane w wielu formatach. Interfejsy API odczytują np. częściowo ustrukturyzowane pliki z obiektowej pamięci masowej lub subskrybują strumień zdarzeń.

Interfejsy API danych wejściowych

Te API są używane wewnętrznie przez produkt danych do konfigurowania i odczytywania danych z upstreamowych źródeł. Mogą być implementowane jako subskrypcja upstreamowych interfejsów API danych wyjściowych i asynchronicznie odbierać dane, gdy stają się dostępne. Mogą być także implementowane jako zapytania wykonywane na upstreamowym źródle i odczytywane jako ramki danych. Przybycie danych wejściowych uruchamia kod transformacji.

Interfejsy API wykrywania i obserwowalności

Te interfejsy API dostarczają dodatkowych informacji o produkcie danych, aby zapewnić pomoc w zakresie jego wykrywalności i debugowania. API wykrywania może dostarczać np. informacji o zespole zajmującym się utrzymywaniem produktów danych, znacznikach pomagających wyszukiwać produkty, opisach ich semantyki itp.



Rysunek 9.9. Interfejsy API produktu danych jako kod

Zarządzanie standaryzuje definiowanie tych interfejsów dla wszystkich produktów danych, a platforma zapewnia mechanizmy ich implementacji.

W siatce danych wymiennie z pojęciem interfejsu API stosowane jest określenie „port". Implementacja tych interfejsów API może być synchroniczna, np. RESTful, lub (optymalnie) asynchroniczna, np. z wykorzystaniem systemu wymiany komunikatów pub-sub.

Tradycyjnie te interfejsy implementuje scentralizowana i współdzielona usługa. Cykle życia API są niezależne od dziedzinowych danych lub potoku. Aby uzyskać np. dostęp do danych, konsument danych przechodzi do centralnego katalogu w celu znalezienia zbioru danych, a następnie bezpośrednio z poziomu tego centralnego katalogu dociera do serca systemu pamięci masowej platformy. Siatka danych wprowadza natomiast celowo zaprojektowane interfejsy udostępniane przez każdy produkt danych jako kod w celu wykrywania danych i uzyskiwania do nich dostępu. Interfejsy te mogą ostatecznie skierować żądającego dostępu konsumenta do bazowej pamięci masowej, ale w sposób, który produkty danych mogą w trakcie działania kontrolować, określając, gdzie i jak to zrobić. Interfejsy mają nadawane numery wersji i pozostają zsynchronizowane z resztą kodu produktu danych i jego danymi.

Reguła jako kod Ostatnią kategorią kodu hermetyzowanego przez implementację produktu danych jest kod odpowiedzialny za konfigurację i egzekwowanie różnych reguł behawioralnych i strukturalnych, takich jak szyfrowanie, kontrola dostępu, jakość i zgodność.

Jeśli produkt danych ustawi np. chronioną klasyfikację prywatności, jego dane będą szyfrowane przy zapisie, a odczytywane jako niezaszyfrowane dopiero po sprawdzeniu, czy podmiot odczytujący ma niezbędne uprawnienia. Kod związany z interpretacją i walidacją klasy prywatności, weryfikacją uprawnień użytkowników odczytujących, szyfrowaniem i odszyfrowywaniem danych itp. jest wykonywany w przepływie danych i w kontekście obliczeniowym produktu danych.

Kod ten jest dostarczany przez platformę, ale wywoływany w przepływie skierowanym do funkcji odczytu i zapisu produktu danych.

Wyjaśnię to dokładniej dalej w podrozdziale „Osadzone reguły obliczeniowe”.

Dane i metadane

Dostęp do danych analitycznych jest powodem istnienia produktu danych. Produkt danych zarządza cyklem życia swoich danych dziedzinowych. Zapewnia dostęp do nich w wielu trybach, np. za pomocą plików, tabel, kolumn itp., w zależności od charakteru danych i potrzeb ich użytkowników. Kod transformacji produktów danych zapewnia świeżość danych. Dane te są udostępniane przez interfejsy API udostępniania danych.

Dane analityczne odzwierciedlają fakty dotyczące działalności, które miały miejsce, zaobserwowane trendy lub prognozy i zalecenia na przyszłość.

Aby dane były użyteczne, produkty danych utrzymują i serwują powiązany zestaw informacji o danych — często nazywany **metadanymi**. Obejmują one np. dokumentację, deklaracje semantyczne i składniowe oraz cele SLO danych. Cykl życia metadanych i częstotliwość ich zmian zależą od ich typu. Modele danych, semantyczne lub składniowe, zmieniają się np. rzadziej i w czasie kompilacji. Jednak większość metadanych zmienia się, gdy produkt danych generuje nowe dane. Dotyczy to m.in. właściwości statystycznych danych i ich wartości wskaźników aktualnych celów SLO.

W słownictwie siatki danych rzadko używam terminu „metadane”. Uważam to ogólne i bardzo pojemne określenie za mylące, a nawet szkodliwe. Łączy ono wiele semantycznie odmiennych koncepcji, które w tym worze wszystkich rzeczy zwanych *metadanymi* zasługują na swój własny celowy projekt i wsparcie systemowe. To połączenie koncepcji prowadzi do powstania kruchej

i scentralizowanej architektury, która z czasem staje się splątaną złożonością. Architektura siatki danych oczekuje, że metadane oraz ich kategorie i typy będą częścią kontraktu interfejsów produktów danych.

Duże odejście od tradycyjnej architektury danych polega na tym, że sam produkt danych jest odpowiedzialny za generowanie metadanych, inaczej niż w tradycyjnych systemach, w których metadane są wyodrębniane, ekstrapolowane i rzutowane przez system zewnętrzny — często *po* wygenerowaniu danych. W tradycyjnym świecie system zewnętrzny, taki jak centralny katalog danych, próbuje wyodrębniać, gromadzić i serwować metadane ze wszystkich zbiorów danych.

Zależności platformy

Platforma umożliwia tworzenie, wdrażanie i uruchamianie produktów danych. Chociaż platforma zarządza zasobami infrastrukturalnymi centralnie, zapewnia i przydziela zasoby z izolacją dla każdego produktu danych, aby wspierać jego autonomiczne działanie. Ważne jest, aby wdrożenie lub aktualizacja jednego produktu danych nie miały wpływu na inne produkty danych. Wdrożenie jednego produktu danych w jego infrastrukturze hostingowej nie powinno np. zakłócać działania innych produktów ze współdzielonej infrastruktury.

Produkt danych definiuje i kontroluje swój oczekiwany stan docelowy i jego zależności od platformy. Oczekiwania dotyczące czasu przechowywania danych lub rodzajów dostępu do danych wyjściowych mogą być np. wyrażone jako zależności od platformy i wykorzystane przez platformę do zapewnienia właściwego rodzaju pamięci masowej i zarządzania nią.

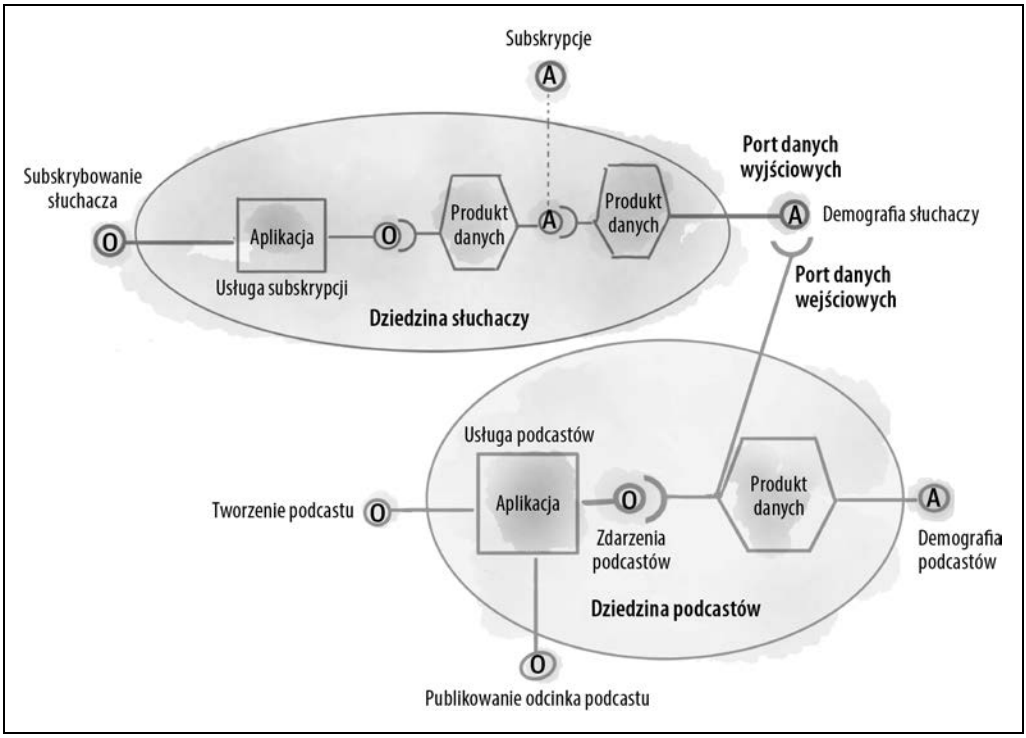
Interakcje współdzielenia danych przez produkty danych

Większość interakcji produktów danych dotyczy współdzielenia danych — serwowania i konsumowania. Na rysunku 9.10 przybliżyłam obraz dziedziny i interfejsu analitycznego, które przedstawiłam w poprzednim punkcie. Pokazuje to, w jaki sposób produkty danych integrują się ze współpracującymi źródłowymi systemami operacyjnymi lub innymi produktami danych.

Każda dziedzina może mieć wiele produktów danych. W tym przykładzie dziedzina ***słuchaczy*** ma dwa produkty danych: ***subskrypcje*** i ***demografię słuchaczy***. Te produkty danych i ich interfejsy udostępniania danych, zwane inaczej **portami danych wyjściowych**, stają się interfejsami analitycznymi, które ta dziedzina udostępnia.

Produkt danych może konsumować dane ze współpracującego z nim systemu operacyjnego lub z upstreamowego produktu danych. W tym przykładzie produkt danych ***demografia podcastów*** konsumuje ***zdarzenia podcastów*** ze współpracującej z nim aplikacji operacyjnej zwanej ***usługą podcastów*** i z upstreamowego produktu danych zwanego ***demografią słuchaczy***. Jego interfejs wejściowy — **port danych wejściowych** — implementuje integrację ze źródłami i odbieranie od nich danych.

Bazowa technologia implementacji portu danych wejściowych produktu danych i portu danych wyjściowych może się różnić w zależności od implementacji siatki danych, ale ma kilka wspólnych właściwości. Opiszę je w kolejnych podpunktach.



Rysunek 9.10. Przykład interakcji związanych z udostępnianiem danych

Porty danych wejściowych

Są to wewnętrzne mechanizmy integracji, które produkt danych implementuje w celu konsumowania danych z co najmniej jednego źródła. Produkt danych może konsumować np. zdarzenia dziedzinowe w czasie zbliżonym do rzeczywistego ze współpracującej z nim mikrousługi, a następnie konwertować je na historyczne dane analityczne z długim okresem retencji.

Port danych wejściowych może być też ewentualnie wyzwalanym czasowo zapytaniem, które produkt danych wywołuje na upstreamowym produkcie danych w celu otrzymania wyniku zapytania i przekształcenia go w dane zagregowane.

Na rysunku 9.10 ***demografia podcastów*** jest zagregowanym produktem danych, który dostarcza statystycznego podsumowania charakterystyki populacji słuchaczy. Wykorzystuje model uczenia maszynowego klasyfikacji do identyfikacji kohort podcastów na podstawie danych demograficznych ich słuchaczy. Jest to pomocna informacja analityczna zarówno dla producentów podcastów, jak i dla firmy Daffa, aby lepiej dopasować podcasty do słuchaczy.

Produkt danych ***demografia podcastów*** wykorzystuje swoje porty danych wejściowych do ciągłego odbierania danych ze współpracującej z nim mikrousługi zwanej ***usługą podcastów***, a także z produktu danych ***demografia słuchaczy*** dziedziny ***słuchaczy***.

Porty danych wyjściowych

Są to zewnętrznie adresowalne interfejsy API, które udostępniają bazowe dane produktu danych. Każdy produkt danych może mieć co najmniej jeden port danych wyjściowych. Siatka danych sugeruje posiadanie dla portu danych wyjściowych znormalizowanych interfejsów API, które wszyscy konsumenci będą wywoływać przed uzyskaniem dostępu do danych. Przed przekierowaniem konsumenta do danych bazowych te interfejsy API egzekwują reguły takie jak kontrola dostępu.

Na rysunku 9.10 ***demografia podcastów*** klasyfikuje i rozszerza podcasty na podstawie ich słuchaczy. Nowo sklasyfikowane dane demograficzne podcastów serwuje w sposób ciągły za pośrednictwem portu danych wyjściowych.

Interfejsy API wykrywania i obserwowalności danych

Produkty danych dostarczają informacji, które są niezbędne do ich wykrywania, zrozumienia, debugowania i audytowania. Robią to za pośrednictwem zestawu interfejsów synchronicznych lub asynchronicznych. Interfejsy API wykrywania udostępniają np. identyfikator produktu danych, zespół, który jest jego właścicielem, semantykę danych, dokumentację, znaczniki pomagające go wyszukiwać itp. Interfejsy API obserwowalności zapewniają dzienniki dostępu, informacje o pochodzeniu i wskaźniki dotyczące udostępniania danych.

Zawartość i funkcje tych interfejsów API omówię szerzej w części IV.

Wielopłaszczyznowa platforma danych

Można sobie wyobrazić długą listę funkcjonalności i usług, które platforma danych musi zapewnić, aby spełnić obietnice siatki danych omówione w rozdziale 4. Mówiąc w skrócie, chodzi o umożliwienie interdyscyplinarnym zespołom dziedzinowym autonomicznego zarządzania produktami danych oraz użytkownikom danych odkrywania, uczenia się i konsumowania siatki produktów danych, a wszystko to w sposób bezpieczny i zgodnie z regułami siatki.

Istnieją dwa ogólne podejścia do projektowania takiej platformy: przyjęcie pojedynczej i ściśle zintegrowanej platformy (często kupowanej od głównego dostawcy) lub skorzystanie z zestawu luźno zintegrowanych usług, które udostępniają standardowe i otwarte interfejsy, często zaimplementowane jako połączenie elementów budowanych oraz kupowanych od różnych dostawców.

Projekt platformy z siatką danych mieści się w tej drugiej kategorii. Preferuje dzielenie złożonej platformy na jej funkcjonalności (obsługiwane przez interfejsy API) z otwartymi interfejsami do integracji z innymi współpracującymi funkcjonalnościami.

Chociaż idea pojedynczej platformy jest atrakcyjna, tj. łatwiej jest radzić sobie z jednym dostawcą, ma ona ograniczenia w zakresie skalowania. Pojedyncza platforma ogranicza np. miejsca hostowania usług i danych i tworzy powiązanie z dostawcą. Niemniej z czasem złożoność organizacji prowadzi do przyjęcia wielu platform, ale nie współdziałają one prawidłowo, ponieważ każda ma wbudowane założenie, że jest jedynym administratorem danych lub aplikacji. Prowadzi to do przeprowadzania doraźnych i drogich integracji oraz kosztownego kopiowania danych między różnymi platformami.

Siatka danych domyślnie zakłada, że złożoność organizacji prowadzi do tworzenia środowisk wieloplatformowych i wielohostingowych. Aby wspierać ten model, siatka danych oferuje takie podejście do architektury platformy, które polega na komponowaniu usług ze standardowymi interfejsami. Dzięki temu dane mogą być udostępniane między różnymi usługami i środowiskami hostingowymi.

Zarządzanie cyklem życia produktu danych wymaga np. zbioru funkcjonalności, takich jak *inicjowanie produktu danych*, *budowanie produktu danych*, *testowanie produktu danych*, *wdrażanie produktu danych*, *monitorowanie produktu danych* itd. Platforma siatki danych kładzie nacisk na projektowanie (otwartych) interfejsów, które tworzą warstwę abstrakcji dla tych funkcjonalności. Podejście oparte na interfejsach API zapewnia elastyczność w obliczu rosnącej złożoności. Umożliwia rozszerzanie z czasem doświadczenia zarządzania produktem danych o nowe funkcjonalności w czasie lub przenoszenie zarządzania produktem danych do innej infrastruktury bez zakłócania pracy użytkownikom siatki.

Jeśli chodzi o **płaszczyzny** platformy, celowo unikam używania określenia „warstwa". Warstwy często kojarzą się ze ściśle hierarchicznym dostępem z przepływem informacji i sterowania przechodzącym z jednej warstwy do następnej. Koncepcja architektury aplikacji warstwowej wymaga np. ukrycia dostępu do bazy danych w najniższej warstwie i umożliwienia dostępu do danych tylko za pośrednictwem zmapowanych obiektów wyższych warstw.

W przypadku płaszczyzn platformowych użytkownik siatki może zdecydować się na skorzystanie z usług wyższej lub niższej płaszczyzny abstrakcji, w zależności od swoich potrzeb.

Płaszczyzna platformy

W logicznym projektowaniu funkcjonalności platformy siatki danych posługuję się pojęciem **płaszczyzny**. Stanowi ona logiczny zbiór funkcjonalności z uzupełniającymi się celami i wysoką spójnością funkcjonalną w osiąganiu kompleksowych wyników.

Każda płaszczyzna, podobnie jak **powierzchnia**, tworzy abstrakcję złożoności infrastruktury i oferuje zestaw interfejsów (API) dla implementowanych funkcjonalności. Dostęp do funkcjonalności płaszczyzny uzyskuje się za pośrednictwem jej interfejsów.

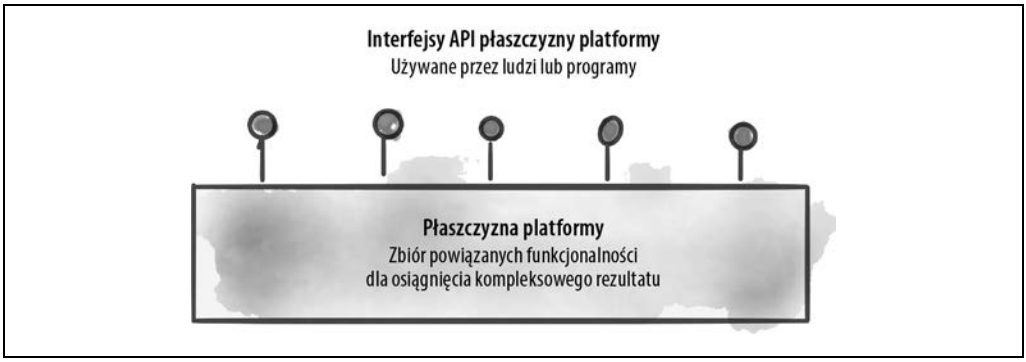
Użytkownik końcowy, taki jak programista produktu danych, może wchodzić w bezpośrednie interakcje z wieloma płaszczyznami platformy. Skoro istnieje luźna zależność między płaszczyznami, nie ma silnego pojęcia warstw lub hierarchicznego modelu dostępu. Płaszczyzny uwzględniają koncepcję **separacji zagadnień** bez silnych ograniczeń budowy warstwowej.

Logiczna architektura platformy siatki danych skupia się wyłącznie na interfejsach płaszczyzn, do których dostęp jest programowy lub ręczny. Koncentruje się na projektowaniu protokołów do uzyskiwania dostępu do funkcjonalności platformy w celu osiągnięcia określonego rezultatu w tworzeniu, wdrażaniu lub używaniu produktów danych w siatce oraz zarządzaniu nimi.

W modelowaniu architektury przywołuję tylko interfejsy logiczne. Fizyczna implementacja interfejsu wykracza poza zakres projektu architektonicznego — bez względu na to, czy będzie to implementacja

za pośrednictwem **interfejsu wiersza poleceń** (ang. *Command-Line Interface* — CLI), interfejsów API lub **graficznego interfejsu użytkownika** (ang. *Graphical User Interface* — GUI) i rzeczywistej liczby fizycznych wywołań każdego interfejsu logicznego.

Na rysunku 9.11 przedstawiłam notację, której będę używała do opisywania poszczególnych płaszczyzn platformy infrastrukturalnej, z naciskiem na interfejsy logiczne, które zapewnia jedna płaszczyzna. Użytkownicy płaszczyzny — systemy lub ludzie, produkty danych, inne płaszczyzny lub użytkownicy danych — powinni być w stanie korzystać z tych interfejsów w sposób samoobsługowy oraz odkrywać je, rozumieć, wywoływać i uzyskiwać do nich dostęp.



Rysunek 9.11. Notacja: udostępniane interfejsy API płaszczyzny platformy

Z uwagi na złożoność platformy siatki danych i jej szeroki zakres funkcjonalności, jej logiczna architektura składa się z wielu płaszczyzn. Do tej pory wskazałam trzy różne płaszczyzny infrastruktury siatki danych.

(Narzędziowa) płaszczyzna infrastruktury danych

Ta płaszczyzna jest odpowiedzialna za zarządzanie zasobami niskopoziomowej infrastruktury służącymi do budowania i uruchamiania siatki, takimi jak pamięć masowa, zasoby obliczeniowe, system określania tożsamości itp. Integruje się ona i pokrywa z infrastrukturą aplikacji cyfrowych, która buduje i uruchamia systemy operacyjne. Infrastruktura danych może np. współdzielić ten sam silnik CI/CD dla produktów danych, którego reszta zespołu dziedzinowego używa do tworzenia aplikacji. Kolejnym przykładem jest zarządzanie standardowymi typami pamięci masowej, takimi jak magazyny obiektowe, zarówno dla systemów analitycznych, jak i operacyjnych.

Płaszczyzna doświadczenia produktu danych

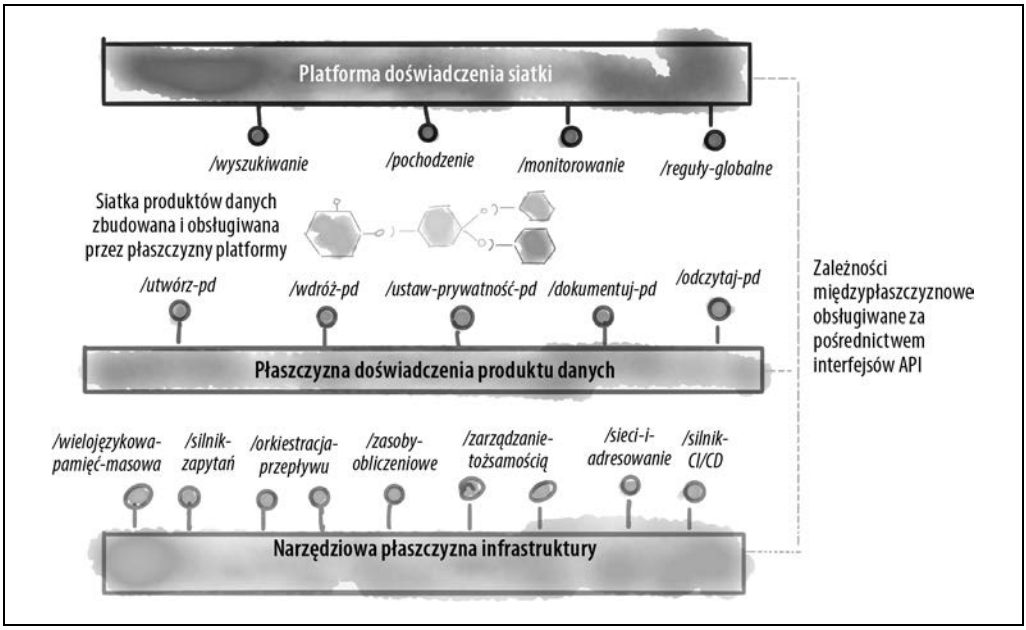
Jest to wysokopoziomowa abstrakcja do tworzenia, utrzymywania i konsumowania produktów danych, zbudowana przy użyciu płaszczyzny infrastruktury. Jej interfejsy współpracują bezpośrednio z produktem danych. Wspiera np. programistów produktów danych w zarządzaniu cyklem życia produktu: budowaniu produktu danych, wdrażaniu go itp. Wspiera również konsumentów produktu danych w korzystaniu z niego, np. w subskrybowaniu danych wyjściowych i odczytywaniu danych.

Płaszczyzna doświadczenia siatki

Płaszczyzna ta tworzy abstrakcję dla funkcjonalności poziomu siatki działających na wielu produktach danych. Obsługuje takie operacje jak wyszukiwanie produktów danych w siatce lub trawersacja pochodzenia (powiązanie między wejściowymi i wyjściowymi produktami danych) dla wielu produktów danych w siatce.

Przykład

Na rysunku 9.12 pokazałam z perspektywy lotu ptaka płaszczyzny platformy i przykład interfejsów logicznych, które każda z nich może zapewniać. Chociaż płaszczyzny nie narzucają ścisłego warstwowania, co oznacza, że każdy autoryzowany użytkownik (program lub osoba) może uzyskać dostęp do interfejsów dowolnej płaszczyzny, istnieje między nimi określona zależność.



Rysunek 9.12. Wiele płaszczyzn samoobsługowej platformy danych

Płaszczyzna doświadczenia siatki zależy od interfejsów *płaszczyzny produktów danych*, ponieważ je agreguje, a *płaszczyzna doświadczenia produktu danych* zależy od interfejsów niskopoziomowej *narzędziowej płaszczyzny infrastruktury*, gdyż zapewnia dla niej abstrakcję.

Interfejsy wymienione na rysunku 9.12 to jedynie przykłady i nie są wyczerpujące. Muszę wyjaśnić, że użyłam języka imperatywnego, podczas gdy w rzeczywistości prawdopodobnie będzie stosowany projekt deklaratywny zorientowany na zasobach. Aby jasne były intencje interfejsu, użyłam np. */wdróż-pd* w celu wskazania możliwości „wdrożenia produktu danych jako jednostki", podczas gdy w praktyce można zaprojektować deklaratywny interfejs API, np. `HTTP POST/wdrożenia-produktów-danych`.

W istniejących platformach, takich jak sieć usług, koncepcja dwóch oddzielnych współpracujących płaszczyzn, *sterowania* i *danych*, została zapożyczona z routingu sieciowego. Chociaż tę separację zagadnień można zastosować również do platformy siatki danych (można sobie wyobrazić, że produkty danych stanowią płaszczyznę danych, a platforma stanowi płaszczyznę sterowania), celowo trzymałam się z dala od tego projektu.

Wykorzystałam koncepcję **doświadczenia**, aby skupić się na *afordancjach* i *doświadczeniach*, które zapewnia platforma na różnych poziomach — na poziomie poszczególnych produktów danych i na poziomie ich kolekcji jako siatki. Platforma oferuje np. doświadczenie **wykrywania usług** i **adresowalności**: zdolności jednych produktów danych do programowego wyszukiwania drugich produktów danych po nazwie i adresowania ich w celu konsumowania ich danych. Można to sobie oczywiście wyobrazić jako implementację płaszczyzny podobnej do sterowania, która implementuje dynamiczny rejestr, adresowanie, routing itp. Traktuje to jako szczegóły implementacji, które będą następstwem doświadczenia.

Zasadniczo architektura siatki danych projektuje platformę na podstawie *kontraktu* doświadczenia, jakie zapewnia swoim użytkownikom.

Szczegółowo funkcjonalności platformy omówię w rozdziale 10.

Osadzone reguły obliczeniowe

W rozdziale 5. wprowadziłam kilka rodzajów obliczeń, które wspierają zarządzanie siatką danych: **reguły obliczeniowe**, **znormalizowane protokoły produktów danych**, **zautomatyzowane testy** i **zautomatyzowane monitorowanie**.

Sposób modelowania architektury pod kątem konfigurowania i egzekwowania reguł narzuca standardy i utrzymuje zgodność produktów danych z oczekiwaniami jakościowymi, co ma bezpośredni wpływ na to, jak skuteczna będzie funkcja zarządzania. Architektura jest kanałem dla efektywnego zarządzania, za pomocą którego reguły i standardy są stosowane do wszystkich produktów danych w siatce.

Kwant danych jest potężną i rozszerzalną konstrukcją do osadzania polityk obliczeniowych w sposób rozproszony. Pozwala definiować i wykonywać *reguły jako kod*.

Reguły wyrażone jako kod — w sposób jednolity w całej siatce — są udostępniane każdemu produktowi danych, który następnie może je ewaluować i stosować w odpowiednim momencie.

Architektura siatki danych wprowadza kilka kolejnych logicznych komponentów do zarządzania regułami produktów danych jako kodem:

Przyczepa produktu danych

> **Przyczepa** (ang. *sidecar*) to proces, który implementuje wykonywanie reguł i innych aspektów produktów danych, które muszą być znormalizowane w całej siatce. Zapewnia to platforma. Wdraża i uruchamia z produktem danych jako pojedynczą jednostkę.

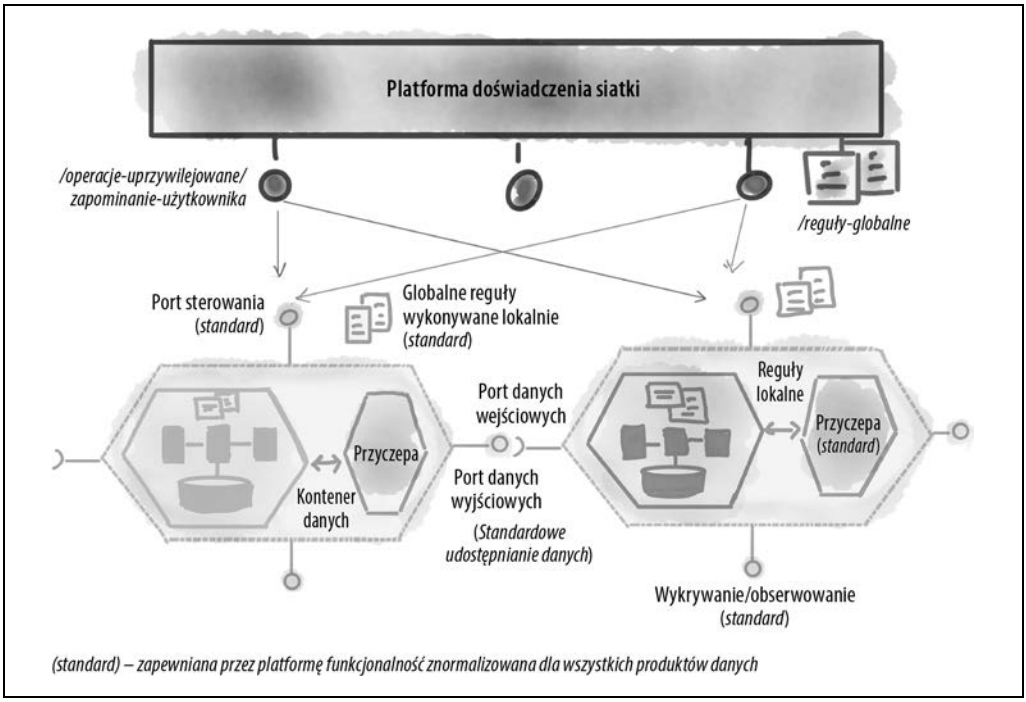
Kontener obliczeniowy produktu danych

Jest to sposób, w jaki platforma hermetyzuje wykonywanie reguł jako pojedynczą jednostkę wdrażaną z produktem danych. Dla zwięzłości nazywam to czasem **kontenerem danych**.

Port sterowania

Port sterowania zapewnia zestaw standardowych interfejsów do zarządzania i sterowania regułami produktów danych.

Te logiczne komponenty przedstawiłam na rysunku 9.13. Najlepiej jest, jeśli platforma dostarcza i normalizuje niezależne dziedzinowo komponenty, takie jak przyczepa, implementacja portów sterowania oraz porty wejściowe i wyjściowe.



Rysunek 9.13. Komponenty logicznej architektury wbudowanych polityk obliczeniowych

Zwróć uwagę, że każdy z tych komponentów logicznych może być zaimplementowany jako jeden komponent fizyczny lub wiele komponentów. Przyczepa jest np. rozszerzalnym i logicznym symbolem zastępczym dla różnych rodzajów agentów egzekwowania reguł.

Przyczepa produktu danych

Przyczepa jest logicznym komponentem, który dekoruje kontekst środowiska uruchomieniowego każdego produktu danych i wykonuje przekrojowe zadania, które mają być znormalizowane w całej siatce.

Przyczepa jest często implementowana jako oddzielny proces w celu tworzenia dynamicznych i luźnych powiązań z produktem danych. Często jest wstrzykiwana w czasie wdrażania w kontener produktu danych. Z czasem możemy mieć więcej niż jedną przyczepę lub rozszerzać istniejącą przyczepę o dodatkowe funkcjonalności interdyscyplinarne. Niemniej jej konstrukcja musi być rozszerzalna, aby z czasem objąć większą liczbę reguł.

Przyjrzyjmy się kilku sposobom, na jakie można rozszerzyć przyczepę, aby zapewnić spójność wszystkich produktów danych.

Wykonywanie reguł

Jednym z obowiązków przyczepy produktu danych jest wykonywanie reguł. Chociaż zarządzanie siatką danych obejmuje różnorodność produktów danych w wielu wymiarach, takich jak modelowanie danych, wymaga ono jednak spójnego stosowania określonych reguł w odniesieniu do wszystkich produktów danych.

W architekturze rozproszonej, takiej jak siatka danych, celem jest usunięcie wszelkich przypadkowych punktów rywalizacji o zasoby i wąskich gardeł. Z tego powodu najlepiej jest konfigurować i egzekwować reguły niezależnie w lokalnym kontekście wykonywania poszczególnych produktów danych, a nie w scentralizowanej bramie.

Typową strategią osiągnięcia tego celu jest wstrzykiwanie wykonywania reguł w kontekst lokalny. W takim przypadku przyczepa towarzysząca każdemu produktowi danych może ewaluować i stosować reguły kontroli dostępu, gdy uzyskiwany jest dostęp do portu wyjściowego, lub stosować szyfrowanie, gdy produkt danych zapisuje nowe dane.

Strategia ta została zaimplementowana w architekturze systemów operacyjnych za pomocą wzorca Przyczepa (*https://oreil.ly/IILMb*). Przyczepa to proces, który rozszerza zachowanie aplikacji o kwestie przekrojowe. Wzorzec ten jest wykorzystywany np. przez siatkę usług do implementowania routingu, bezpieczeństwa, odporności i innych reguł dotyczących wywołań przychodzących i wychodzących z usługi.

Znormalizowane protokoły i interfejsy

Ważnym celem zarządzania jest tworzenie interoperacyjnego ekosystemu produktów danych. Interfejsy produktów danych, które udostępniają ich dane i zachowanie — **porty danych wejściowych**, **porty wykrywania**, **porty danych wyjściowych** i **port sterowania** — mają zasadnicze znaczenie dla interoperacyjności między produktami danych.

Przyczepa umieszczona w przepływie wszystkich przychodzących i wychodzących komunikatów produktów danych może zaoferować znormalizowane interfejsy API. Może zapewnić np. znormalizowane API do udostępniania wskaźników SLO z poszczególnych produktów danych. Implementacja produktu danych wykorzystuje standardową implementację tych interfejsów API do udostępniania swojej unikatowej zawartości.

Kontener obliczeniowy produktu danych

Jak widać na rysunku 9.13, wiele komponentów strukturalnych jest połączonych razem, aby utworzyć autonomicznie użyteczny produkt danych: przyczepa, konfiguracje reguł, kod transformacji produktu danych, dane produktu danych, jego interfejsy udostępniania danych itp.

Kolejnym elementem zapewniającym skuteczność sfederowanego zarządzania jest kontener, który opakowuje wszystkie elementy strukturalne produktu danych jako kwantu architektury.

Płaszczyzna doświadczenia produktu danych w spójny sposób zapewnia implementację kontenera dla wszystkich produktów danych.

Port sterowania

Zarządzanie obliczeniowe wprowadza nowy interfejs dla wszystkich produktów danych — port sterowania. Port sterowania udostępnia zbiór interfejsów API do konfigurowania reguł dla każdego produktu danych oraz wykonywania niewielkiego zestawu operacji z wysokimi uprawnieniami w celu spełnienia wymogów zarządzania.

Konfigurowanie reguł

Port sterowania jest interfejsem do konfigurowania reguł, które muszą być wykonywane w kontekście produktu danych. Reguły te mogą być tworzone i kontrolowane lokalnie przez sam produkt danych lub centralnie przez płaszczyznę doświadczenia siatki.

Reguły dotyczące anonimizacji danych najlepiej jest np. *tworzyć lokalnie* przez deklarowanie atrybutów danych osobowych produktu danych. Właściciele dziedzinowego produktu danych mają najbardziej istotne informacje pozwalające wskazać i skonfigurować ewentualne dane osobowe produktu danych. Natomiast definicje ról i ich kontrole dostępu są często tworzone centralnie i egzekwowane lokalnie w każdym produkcie danych.

Dla pojedynczego produktu danych mogą współistnieć obie formy tworzenia reguł: **konfigurowanie lokalne** lub **centralne**.

Projekt portu sterowania musi być rozszerzalny, aby zapewniać obsługę nowych rodzajów reguł w miarę ich wprowadzania do siatki.

Operacje z wysokim poziomem uprawnień

Funkcja zarządzania potrzebuje sposobu na wykonywanie na wszystkich produktach danych niewielkiego zestawu operacji z wysokim poziomem uprawnień, np. operacji egzekwowania prawa RODO do *bycia zapomnianym* — prawa osoby fizycznej do usunięcia wszystkich jej danych osobowych zebranych przez organizację.

Wyobraź sobie, że w przykładach firmy Daff jeden ze słuchaczy zdecyduje się usunąć wszystkie informacje na swój temat zebrane przez firmę. Globalna funkcja zarządzania musi mieć sposób, aby oznaczyć to we wszystkich produktach danych w siatce. **Porty sterowania** mogą udostępniać określony interfejs z wysokimi uprawnieniami, który implementuje *prawo jednostki do bycia zapomnianym*.

Platforma standaryzuje port sterowania dla wszystkich produktów danych.

Podsumowanie

W tym rozdziale przedstawiłam logiczną architekturę, która sprawiła, że zasady siatki danych stały się bardziej konkretne i bliższe implementacji.

W momencie, gdy piszę ten rozdział, różne aspekty architektury logicznej są na różnych etapach dojrzałości — niektóre są implementowane i testowane, a inne są bardziej eksperymentalne. Ich stan dojrzałości wskazałam w tabeli 9.1.

Tabela 9.1. Logiczne komponenty architektoniczne siatki danych i stopień ich dojrzałości

Komponent architektoniczny	Opis
Dziedzina	Systemy, produkty danych i interdyscyplinarne zespoły dostosowane do obsługi dziedzinowej funkcji biznesowej i udostępniania jej analitycznych i operacyjnych funkcjonalności całej organizacji i szerszemu gronu i klientów. *Jest to dobrze ugruntowana koncepcja.*
Dziedzinowe interfejsy danych analitycznych (zobacz punkt „Projekt interfejsu danych analitycznych”)	Znormalizowane interfejsy, które wykrywają i udostępniają dziedzinowe produkty danych oraz zapewniają do nich dostęp. *W chwili, gdy piszę ten rozdział, implementacja tych interfejsów API jest niestandardowa lub charakterystyczna dla danej platformy.* Platformy płatne muszą oferować otwarte interfejsy, aby ułatwić udostępnianie danych i zapewnić interoperacyjność z innymi platformami hostingowymi.
Dziedzinowe interfejsy operacyjne (zobacz punkt „Projekt interfejsu operacyjnego”)	Interfejsy API i aplikacje, za pośrednictwem których dziedziny biznesowe udostępniają szerszej organizacji swoje funkcjonalności transakcyjne i stan. *Ta koncepcja ma dojrzałe implementacje.* Jest obsługiwana przez zatwierdzone standardy, takie jak RESTful, GraphQL, gRPC itd.
Kwant (produktu) danych (zobacz podrozdział „Produkt danych jako kwant architektury”)	Produkt danych zaimplementowany jako kwant architektury, który hermetyzuje wszystkie komponenty strukturalne wymagane do wykonania swojej pracy — kod, dane, specyfikacje infrastruktury i reguły. Jest wspominany w dywagacjach dotyczących architektury. Używany wymiennie z pojęciem produktu danych. *W chwili, gdy piszę ten rozdział, jest to eksperymentalna koncepcja z niestandardowymi implementacjami.*
Kontener (produktu) danych (zobacz punkt „Kontener obliczeniowy produktu danych”)	Mechanizm grupowania wszystkich komponentów strukturalnych produktu danych, wdrażanych i uruchamianych jako pojedyncza jednostka ze swoją przyczepą. *W chwili, gdy piszę ten rozdział, jest to eksperymentalna koncepcja z niestandardowymi implementacjami.*
Przyczepa produktu danych (zobacz punkt „Przyczepa produktu danych”)	Proces towarzyszący produktowi danych. Jest uruchamiany w kontekście kontenera produktu danych i implementuje interdyscyplinarne i znormalizowane zachowania, takie jak wykonywanie reguł globalnych. *W chwili, gdy piszę ten rozdział, jest to eksperymentalna koncepcja z niestandardowymi implementacjami.*
Port danych wejściowych (zobacz podpunkt „Porty danych wejściowych”)	Mechanizmy produktu danych służące do ciągłego odbierania danych z co najmniej jednego źródła upstreamowego. *W chwili, gdy piszę ten rozdział, ma to niestandardowe implementacje z wykorzystaniem istniejących technologii strumieniowego przysyłania zdarzeń i zarządzania potokami.*

Tabela 9.1. Logiczne komponenty architektoniczne siatki danych i stopień ich dojrzałości (ciąg dalszy)

Komponent architektoniczny	Opis
Port danych wyjściowych (zobacz podpunkt „Porty danych wyjściowych")	Znormalizowane interfejsy API produktu danych do ciągłego udostępniania danych. *W chwili, gdy piszę ten rozdział, ma to niestandardowe implementacje charakterystyczne dla określonych dostawców.* *Dojrzała implementacja tej koncepcji wymaga otwartych standardów udostępniania danych z obsługą wielu trybów dostępu do tymczasowych danych.*
Interfejsy API wykrywania i obserwowalności (zobacz punkt „Interfejsy API wykrywania i obserwowalności danych")	Standardowe interfejsy API produktu danych do zapewniania informacji związanych z wykrywalnością — do wyszukiwania, adresowania, uczenia się i eksplorowania produktu danych — oraz informacji związanych z obserwowalnością, takich jak pochodzenie, wskaźniki, dzienniki itd. *W chwili, gdy piszę ten rozdział, zbudowane zostały już niestandardowe implementacje tych API.* *Dojrzała implementacja wymaga otwartych standardów dla modelowania i udostępniania informacji dotyczących wykrywalności i obserwowalności. Niektóre standardy[2] są aktualnie na etapie rozwoju.*
Port sterowania (zobacz punkt „Port sterowania")	Standardowe interfejsy API produktu danych służące do konfigurowania reguł lub wykonywania operacji zarządzania z wysokimi uprawnieniami. *W chwili, gdy piszę ten rozdział, ta koncepcja jest eksperymentalna.*
Płaszczyzna platformy (zobacz punkt „Płaszczyzna platformy")	Grupa samoobsługowych funkcjonalności platformy z wysoką funkcyjną spójnością udostępnianą za pośrednictwem interfejsów API. *Jest to ogólna i dobrze ugruntowana koncepcja.*
Narzędziowa płaszczyzna infrastruktury danych (zobacz punkt „(Narzędziowa) płaszczyzna infrastruktury danych")	Płaszczyzna platformy zapewniająca zarządzanie zasobami niskopoziomowej infrastruktury — zasobami obliczeniowymi, pamięcią masową, systemem identyfikacji tożsamości itd. *W chwili, gdy piszę ten rozdział, usługi składające się na płaszczyznę infrastruktury są dojrzałe i oferowane przez wielu dostawców ze wsparciem dla zautomatyzowanego przydzielania zasobów.*
Płaszczyzna doświadczenia produktu danych (zobacz punkt „Płaszczyzna doświadczenia produktu danych")	Płaszczyzna platformy zapewniająca operacje na produkcie danych. *W chwili, gdy piszę ten rozdział, istnieją niestandardowe implementacje usług składających się na płaszczyznę doświadczenia produktu danych, ale nie zostały udostępnione jeszcze żadne implementacje referencyjne.*
Płaszczyzna doświadczenia siatki (zobacz punkt „Płaszczyzna doświadczenia siatki")	Płaszczyzna platformy zapewniająca operacje na siatce połączonych produktów danych. *W chwili, gdy piszę ten rozdział, istnieje kilka niestandardowych implementacji usług składających się na płaszczyznę doświadczenia siatki, takich jak usługi wykrywania i wyszukiwania, ale nie zostały udostępnione jeszcze żadne implementacje referencyjne.*

Te komponenty uznaje się za logiczne, a ich fizyczna implementacja może przybierać różne formy. Pomysł wykonywania reguł za pomocą przyczepy można fizycznie zaimplementować np. jako proces (usługę) towarzyszący kwantowi danych (powiązany dynamicznie) lub jako współdzieloną bibliotekę (powiązaną statycznie). Mam nadzieję, że ta technologia rozwinie się do tego stopnia, że uda nam się jak najbardziej zbliżyć do siebie logiczną architekturę i jej fizyczną implementację.

W tabeli 9.1 przedstawiłam podsumowanie logicznych komponentów architektonicznych, które wprowadziłam w tym rozdziale.

[2] Przykładem otwartego standardu obserwowalności, który może zostać zaadaptowany przez produkty danych, jest OpenLineage (*https://oreil.ly/366Nj*).

Siatka danych: kolejny etap rozwoju technologii big data!

Dostęp do danych jest warunkiem rozwoju niejednej organizacji. Aby w pełni skorzystać z ich potencjału i uzyskać dzięki nim konkretną wartość, konieczne jest odpowiednie zarządzanie danymi. Rozwiązania stosowane obecnie w tym zakresie nie nadążają już za złożonością dzisiejszych organizacji, rozprzestrzenianiem się źródeł danych i rosnącymi aspiracjami inżynierów, którzy rozwijają techniki sztucznej inteligencji i analizy danych. Odpowiedzią na te potrzeby może być siatka danych, jednak praktyczna implementacja tej koncepcji wymaga istotnej zmiany myślenia.

Ta książka szczegółowo wyjaśnia paradygmat siatki danych, a przy tym koncentruje się na jego praktycznym zastosowaniu. Zgodnie z tym nowatorskim podejściem dane należy traktować jako produkt, a dziedziny — jako główne zagadnienie. Poza wyjaśnieniem paradygmatu opisano tu zasady projektowania wysokopoziomowej architektury komponentów siatki danych, a także przedstawiono wskazówki i porady dotyczące ewolucyjnej realizacji siatki danych w organizacji. Tematyka ta została potraktowana wszechstronnie: omówiono kwestie technologiczne, organizacyjne, jak również socjologiczne i kulturowe. Dzięki temu jest to cenna lektura zarówno dla architektów i inżynierów, jak i dla badaczy, analityków danych, wreszcie dla liderów i kierowników zespołów.

W książce:

- wyczerpujące wprowadzenie do paradygmatu siatki danych
- siatka danych i jej komponenty
- projektowanie architektury siatki danych
- opracowywanie i realizacja strategii siatki danych
- zdecentralizowany model własności danych
- przejście z hurtowni i jezior danych do rozproszonej siatki danych

Zhamak Dehghani jest autorką paradygmatu siatki danych. Pełni funkcję dyrektora do spraw technologii w firmie ThoughtWorks, gdzie zajmuje się systemami rozproszonymi i architekturą danych. Jest członkinią wielu organów doradczych do spraw technologii, a także zwolenniczką decentralizacji w technologii i w społeczeństwie.

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 978-83-8322-037-6
9 788383 220376
Cena: 89,00 zł