



Praca z danymi w aplikacji A.D. 2016. To naprawdę nie będzie tylko o Entity Framework!

Tomasz Kopacz
Architekt, Microsoft

I to nie jest track DBA/SEC tylko DEV!!!





Platinum Sponsor



Silver Sponsors



Brown Sponsors



Strategic Sponsor



Gold Sponsors



Zamiast agendy ... Dwa światy

Sposób zapisu informacji

Pojemniki relacyjne, nierelacyjne, bazy grafowe, bazy dokumentowe, płaskie pliki

Azure Table, DocumentDB, SQL Database, Azure Blob, „Files”,
RavenDB, Cassandra, Hbase, Spark,

Sposób przetwarzania informacji w aplikacji

Pytanie czy programista zna pojęcie tabeli, sam definiuje strukturę bazy.

Każda z baz ma swoje „niuanse”

Programista – zna kod OBIEKTOWY

Postać DANYCH ma być wygodna dla APLIKACJI

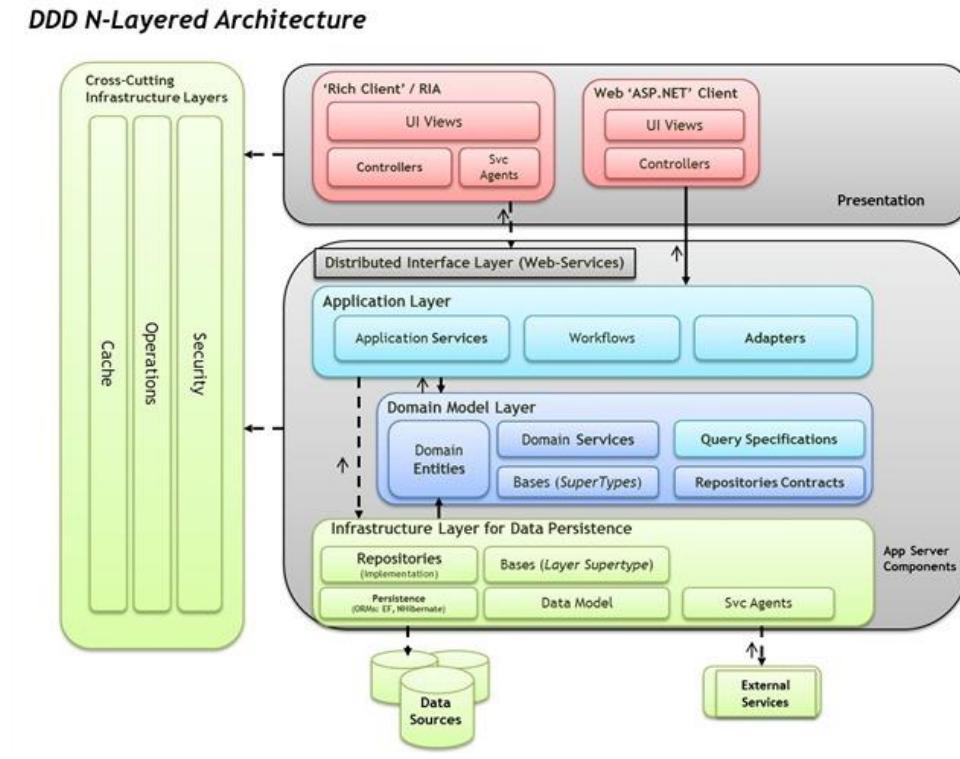
Technologie / style architektoniczne

Klient/serwer, „Disconnected” klient-serwer, WWW + baza,

Usługi SOAP, Usługi REST i mikroserwisy

Podział „kodu”, warstwy

Repository, DTO, Domain Model, Unit of Work, ...



ORM i baza relacyjna – bo
wszyscy znają SQL-a...
A w każdym razie – rozumieją tą koncepcję

Baza relacyjna a program obiektowy

Zadanie ORM: zrobić dla „tabelek” i „relacji” schemat obiektowy

Albo:

Odwzorować schemat obiektowy na „tabelki” i „relacje” by było dobrze

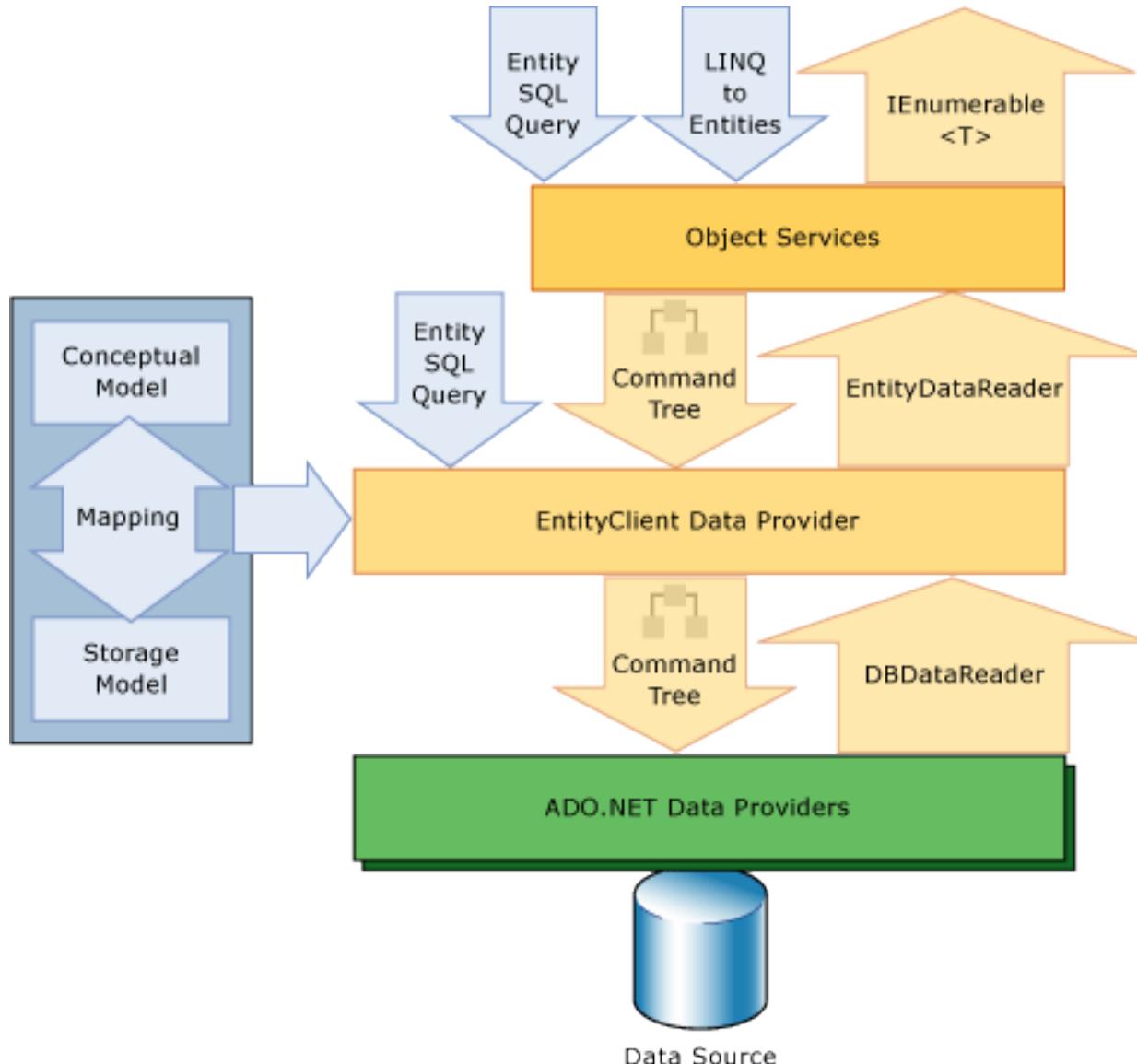
Do tego: LINQ

PROGRAMISTA NIE ZNA SQL-a

Ale za to zna zasady pracy ze zbiorami ☺

Ramowa architektura

*EDM,
używane w
ODATA, C#
do opisu encji,
DocumentDB,
SDK do Azure
Table, ...*



Co do czego

Entity Framework 6.x

API: DBContext, LINQ

Model: EDMX, kod

Baza relacyjna

Cel: FEATURES

Pełny .NET

PRODUKCJA

Entity Framework Core

API: DBContext, LINQ

Model: kod

Różne typy pojemników

Cel: LEKKOŚĆ i PROSTOTA

.NET Core, UWP, ...

2016: ... EKSPERYMENTY

Entity Framework 6.x – wybrane cechy

Model: Designer lub kod

Code First, Database First, synchronizacja, migracje

Cache planów kwerend

Rozgrzewanie „modeli”, generowanie widoków

Śledzenie zmian w grafie po stronie klienta: proxy lub snapshot.

„Improve performance”

Interceptors

Asynchroniczność, automatyczne odnawianie połączenia

Mapowanie sp do operacji insert / update

/Dygresja: Wydajność.../

View Generation

Mapping Views – wykonywalny widok dla każdej encji w formie Canonical Query Trees).

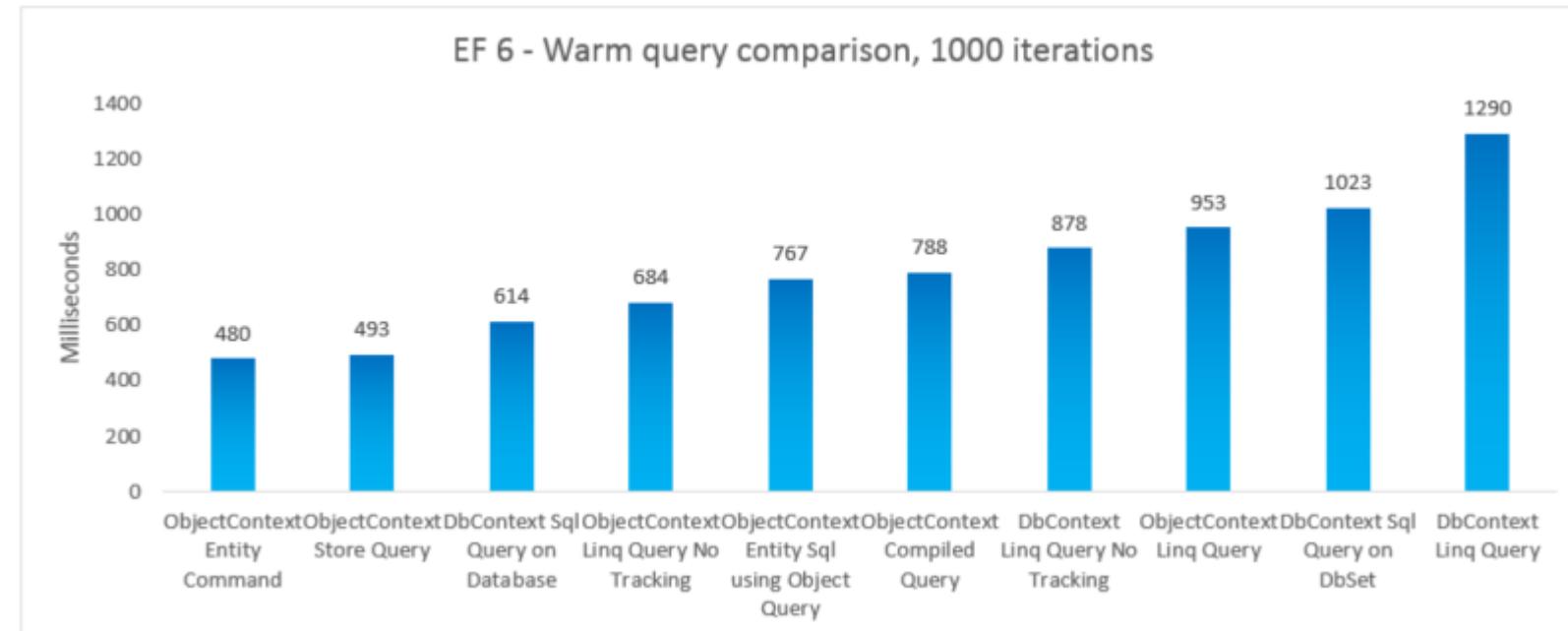
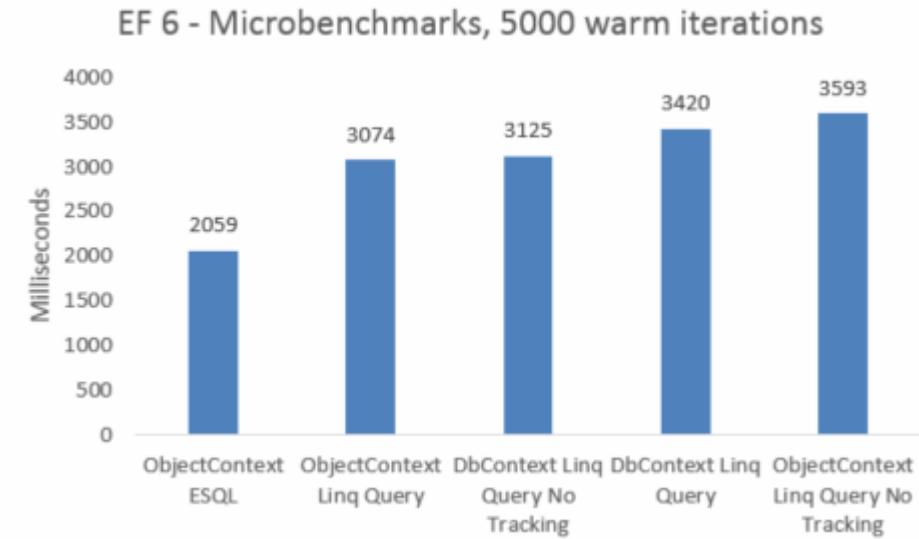
Query Views (db schema -> conceptual) / Update Views.

NoTracking

CompiledQuery

(i inne opcje wykonywania zapytań)

„Large models”



Ważne: Mikrobenchmarki są „mikro”

Demo

Entity Framework 6.x – „normalnie”

1. Wywołanie procedury składowanej
2. Lazy loading, śledzenie itp.
3. Podgląd jaki SQL się wygenerował

A Entity Framework ~~Everywhere~~ 7 Core?

Różne pojemniki – w tym InMemory

Wysokopoziomowe API „wspólne” dla różnych pojemników.

Core = metadata, generowanie SQL, śledzenie zmian.

Rozszerzenia dla specjalistycznych usług. Im więcej, tym większy „narzut” robiony przez Entity Framework

V1 – relacyjne

Modelowanie – POCO, relacje FK / PK, konwencje

Demo

Entity Framework Core – „co może teraz”
EFCore101
FromSql
ConsumingServices

Zaraz, ORM?

Potrzebny bo INACZEJ reprezentujemy dane

Baza relacyjna, postać (trzecia) normalna, ...

Program obiektowy, listy, MVVM, denormalizacja (bo trzeba pokazać w UI)

To może zupełnie inaczej...

Baza nierelacyjna

Not (only) SQL

Schemat danych może się zmieniać

Produkt to zarówno manuskrypt, książka, okręt jak i majtki

Inny zbiór cech! Ale to nadal jest PRODUKT (czyli wpis w jednej tabeli!)

Tradycyjne schematy SQL: „a dopiszmy kolumnę”

Potem – kupmy wersję bazy z np. kolumnami rzadkimi, kompresją

Programiści radzą sobie: dziedziczenie, elastyczne typy, informacje o typie w runtime,

....

(też: **dynamic** w C#, „cały” JavaScript, **any** w TypeScript)

Do tego – bazy relacyjne są WZGLĘDΝIE WOLNE



Co to Azure Table Storage

Baza NoSQL typu klucz-wartość

Dane w 3 replikach. **Koszt 1GB / miesiąc: 0.07\$ + \$0.0036 za 100 000 operacji REST**

PartitionKey (PK) – klucz partycjonujący [wydajność zapytań i skalowalność].

RowKey (RK) – **jednoznacznie** identyfikuje wiersz w **ramach partycji** (inaczej: unikalny w partycji)

Timestamp – Read Only – współbieżność + mechanizm ETAG

TB w pojedynczej partycji + wiele partycji ☺. Automatyczne skalowanie!

1 MB w encji („wierszu”) w tabeli

Typy danych (odpowiedniki w CLR): byte[] (do 64KB), bool, DateTime (8 bajtów!), double, Guid, int, long, String (do 64KB)

Indexy i sortowanie: PK, RK

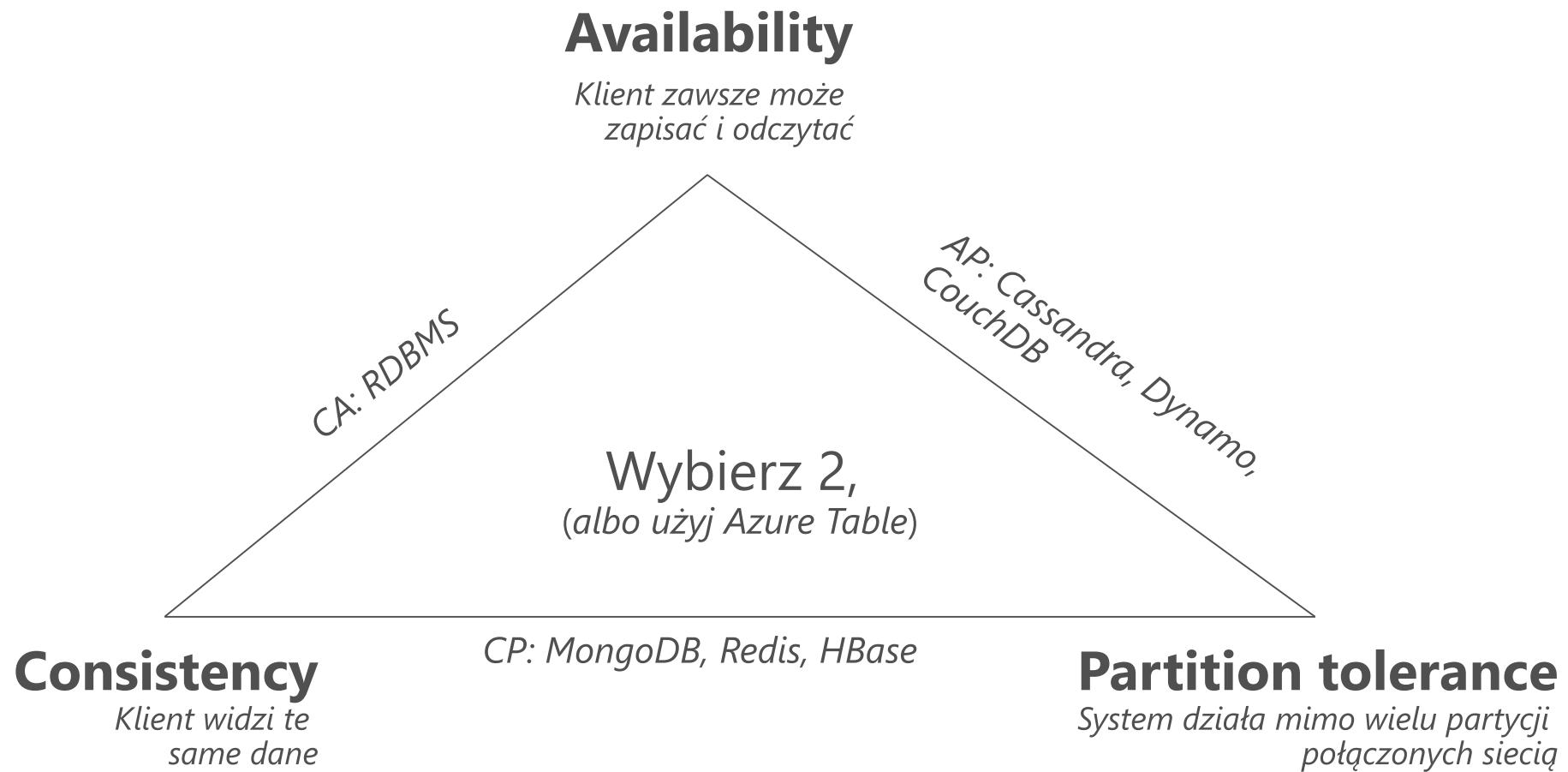
Dla:

Dużo rozproszonych zapytań do różnych przedziałów danych.

Nie dla: dużych, pojedynczych bytów, aktualizowanych przez 1 klienta

Przykłady: koszyk w sklepie, profile użytkownika, metadane urządzeń, katalogi, logi...

Azure Table a CAP Theorem i rozproszone DB



Azure Table – pełny „CAP”

<http://sigops.org/sosp/sosp11/current/2011-Cascais/printable/11-calder.pdf>

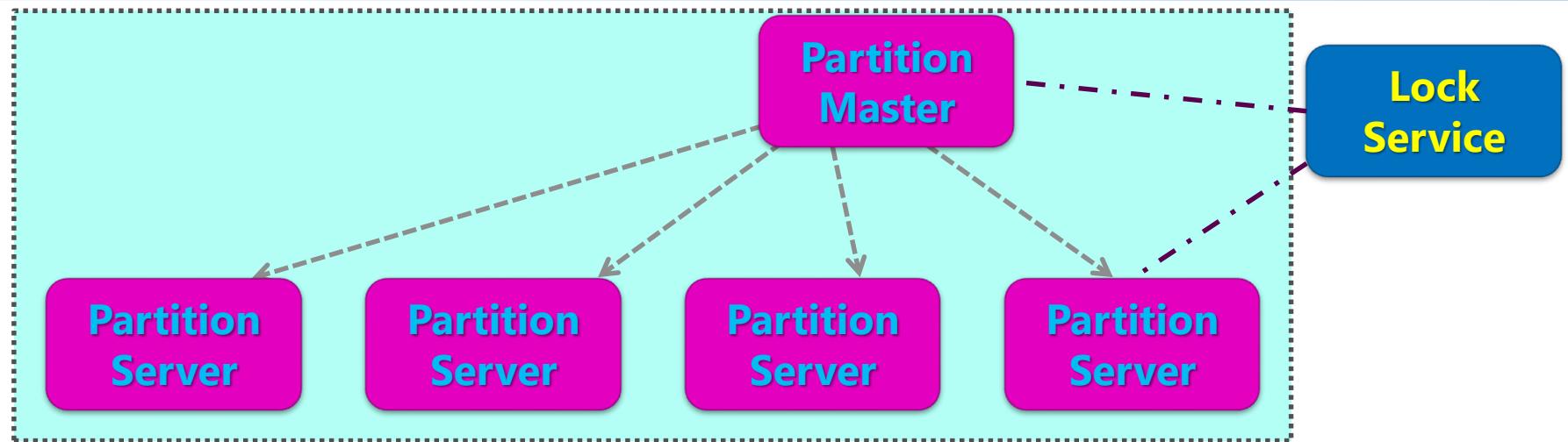
Architektura Storage

Bezstanowe serwery. Autoryzacja i identyfikacja. Przekierowywanie żądań

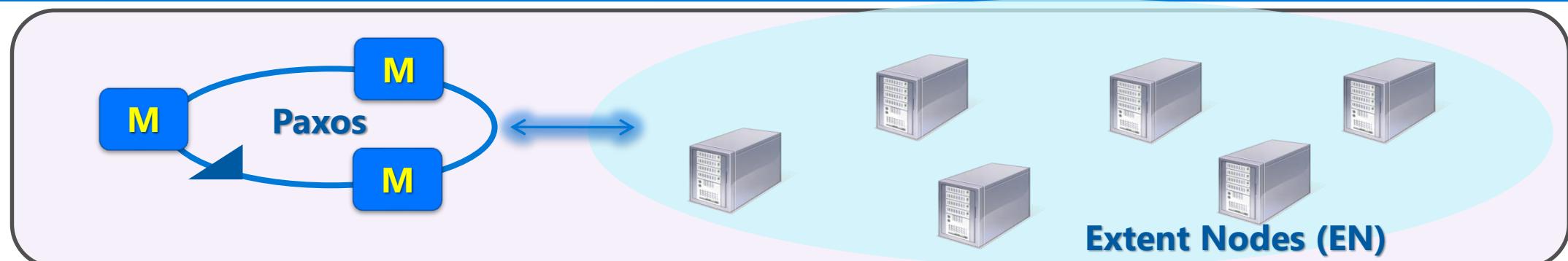
Front End



Warstwa serwerów partycji

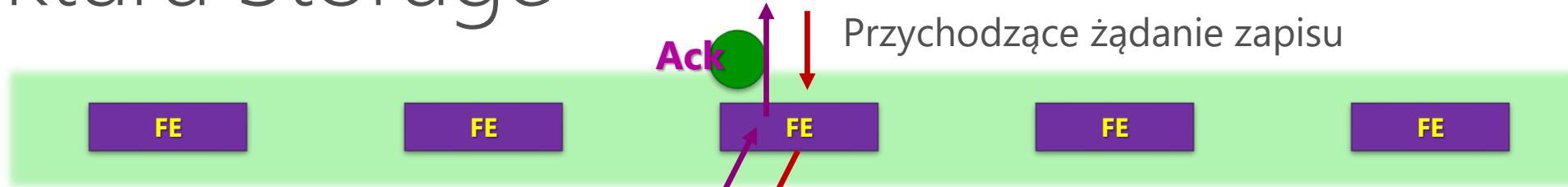


Warstwa strumienia danych (DFS)

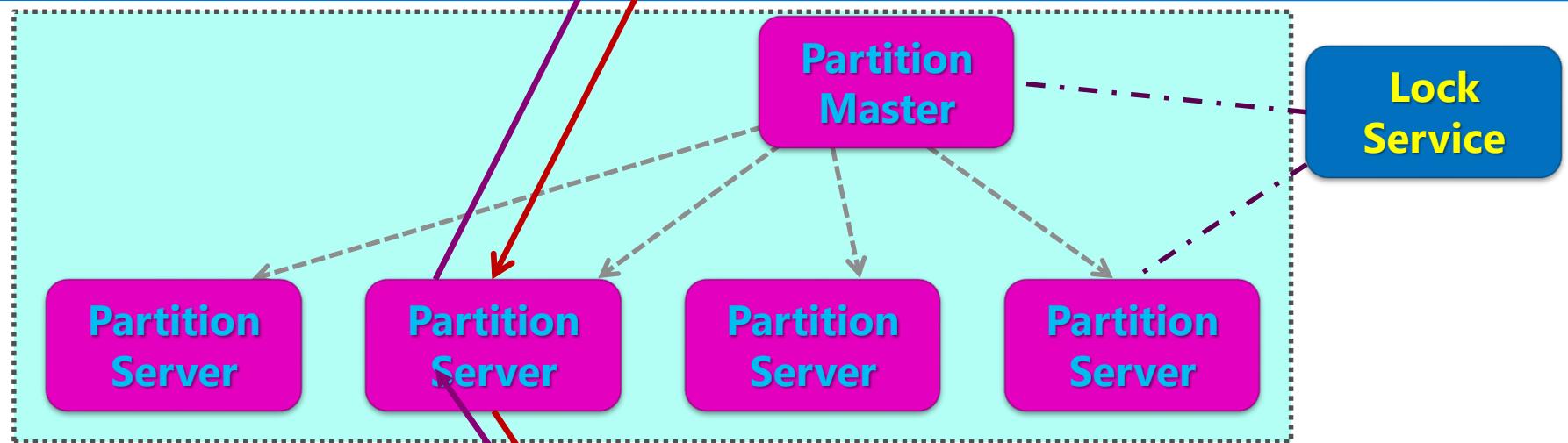


Architektura Storage

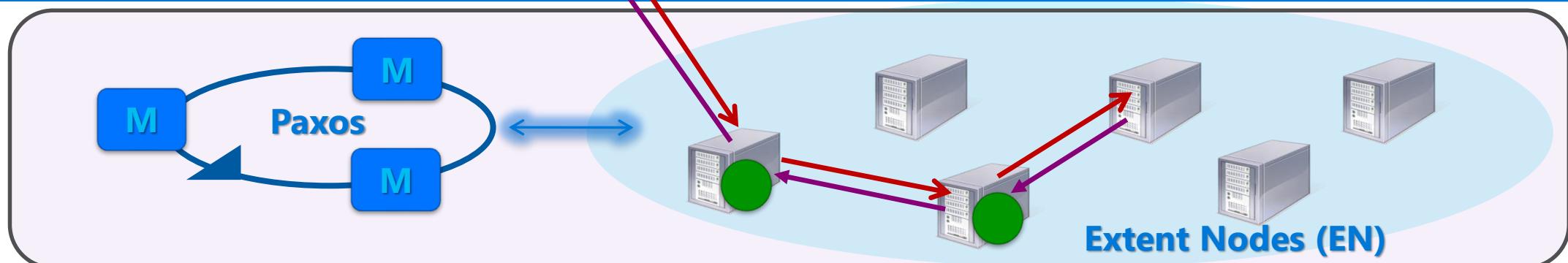
Front End



Warstwa serwerów partycji



Warstwa strumienia danych (DFS)



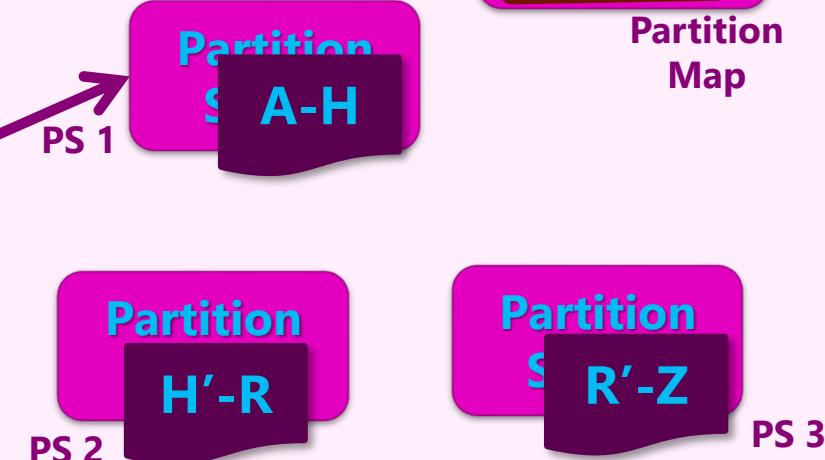
Partycjonowanie indeksów

Dzieli indeks na RangePartitions bazując na obciążeniu
Dzieli na granicy PartitionKey
PartitionMap śledzi przypisanie RangePartition do serwerów partycjonujących
Serwer Front-End „cacheuje” PartitionMap by przekierowywać żądania użytkownika
Każda część indeksu jest przypisana do jednego i tylko jednego serwera partycji w danym momencie

Azure Table

Konto	Partycja	Pola w encji
aaaa	aaaa	aaaaaa
.....
.....
harry	pictures	sunrise
.....	sunset
harry	photos	soccer
richard	videos	tennis
.....
zzzz	zzzz	zzzzzz

Storage Stamp



PS 3

Transakcje i współbieżność

Każda operacja REST jest transakcyjna

Entity Group Transactions (API: TableBatchOperation)

Atomowa aktualizacja do 100 encji w tej samej partycji. Różne operacje: dodaj, aktualizuj, usuń ...

Transakcja oparta o kolejkę

Współbieżność

Azure Table obsługuje: Pesymistyczną, Optymistyczną, Last-Win

Domyślnie: Optymistyczna (tzn – wysłany ETAG musi pasować)

Scenariusz – wypożyczalnia

Informacje o koncie i wypożyczeniach w jednej tabeli

Utrzymać ten sam PartitionKey by wymusić transakcje (Account ID jako PartitionKey)

Aktualizować sumę i dodawać nowe wypożyczenia używając Entity Group Transaction

Poprzedzić RowKey z kodem rodzaju (A = Account, R = Rental)

(Azure Table może mieć różne kolekcje encji w jednej tabeli!)

Row key dla account : [Kind Code]_[AccountId]

Row Key dla rental : [Kind Code]_[Title]

Właściwości Rental nie są ustawiane dla Account i odwrotnie

C#: DynamicTableEntity pomoże!

Kwerendy i wydajność

(Podzbiór OData: \$filter, \$top, \$select)

Najszybsze: PK, RK – singleton, jedna encja, SLA

\$filter=(PartitionKey eq 'Sales') and (RowKey eq '2')

Szybkie: PK, filtr na RK

\$filter=PartitionKey eq 'Sales' and RowKey ge 'S' and RowKey lt 'T'

PartitionSkan: PK, filtr na właściwościach encji

\$filter=PartitionKey eq 'Sales' and LastName eq 'Smith'

TableSkan: Filtr na właściwościach

\$filter=LastName eq 'Jones'

Też: \$filter=PartitionKey eq 'Sales' and (RowKey eq '121' **or** RowKey eq '322')

Sortowane po PK, potem RK

Demo

Dodawanie dużej ilości elementów do różnych partycji (Log). Sumowanie.

Customer i szukanie
Czyli: kilka zabaw z Azure Table

Przykład – baza z „opublikowanymi” hasłami

<https://www.troyhunt.com/working-with-154-million-records-on/>

154 mln rekordów

„Importing 22,500 rows per second into Azure Table Storage”

<https://haveibeenpwned.com/>

<https://haveibeenpwned.com/HowFastIsAzureTableStorage/?email=foo@foo.com>



Podsumowanie – Azure Table

Po co?

1. BARDZO TANIE
2. BARDZO SZYBKIE
3. TRUDNE (dla skomplikowanych schematów)
4. PODSUMOWANIE: OK, ale TRZEBIA MYŚLEĆ

Dygresja: Azure Table + Azure Search

Azure Search: usługa „szukania”, search as a service

Mozliwości:

Analiza lingwistyczna, składnia Lucene, nawigacja po „facets”, profile scoringowe, sugestie, tagi, ...
Ale też: znajdź coś co ma konkretną wartość w danym polu w encji

Indeksuje:

SQL Database, DocumentDB, Azure Table, Azure Blob
Można „ręcznie” bazę ręcznie wypełniać

Czyli: wyszukiwanie w Azure Table. Bez pisania własnego kodu ☺

DocumentDB

„Reklamówka” DocumentDB



> Pełny RDBMS

> Przetwarzanie transakcyjne

> Rozbudowane kwerendy

Zarządzane, skalowalne, łatwość budowy kwerend, bez schematu, oparta o JSON usługa bazodanowa

Zarządzane jako usługa <

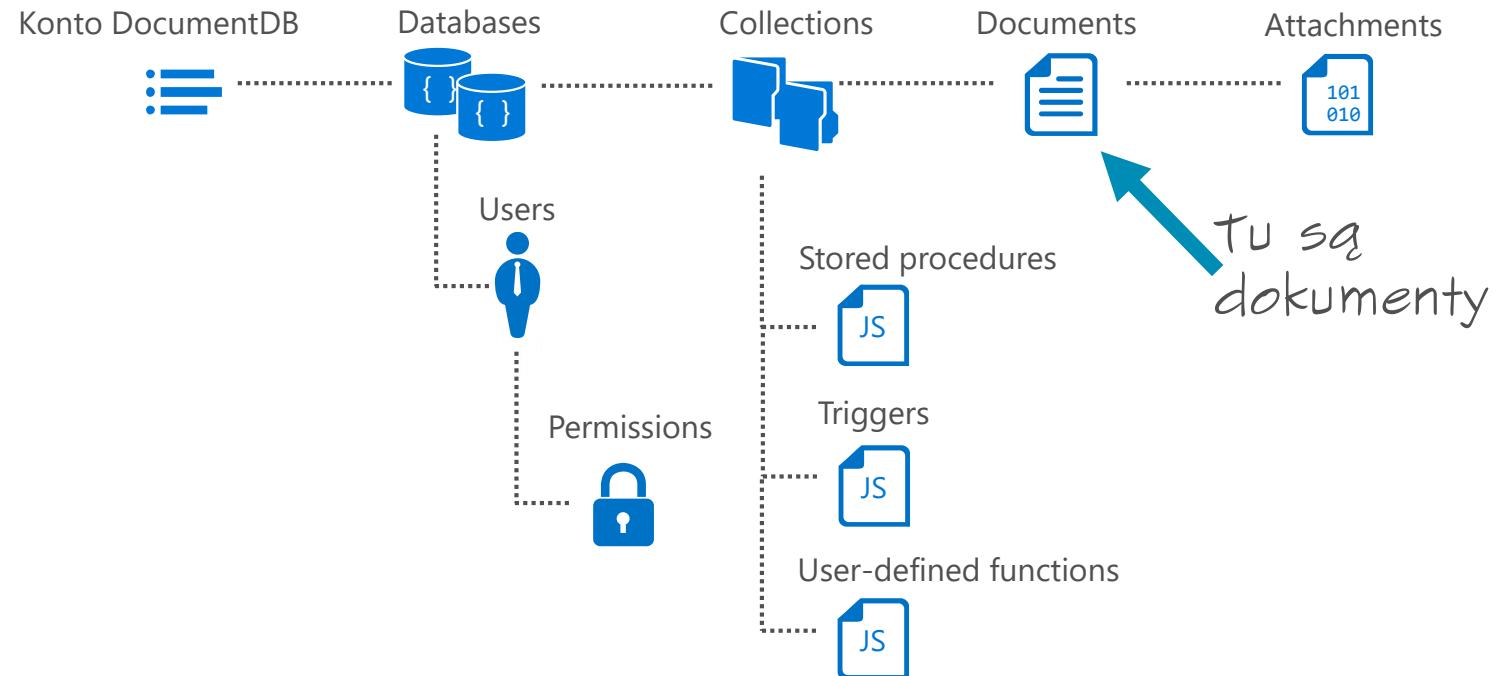
Elastyczne i skalowalne <

Model bez schematu <

Dostępność przez http/rest <

Arbitralne formaty <

Podstawy



Model zasobów

Encje adresowalne przez logiczny URI
Partycjonowane do skalowania wszerz
Replikowane dla zapewnienia wysokiej dostępności
Encje to dokumenty JSON
Konta / kolekcje skalowane przez dodawania „RU” – jednostek skalowania

Model interakcji

HTTP i RESTful
Połączenia albo HTTP albo TCP
Standardowe polecenia HTTP

Dewelopment

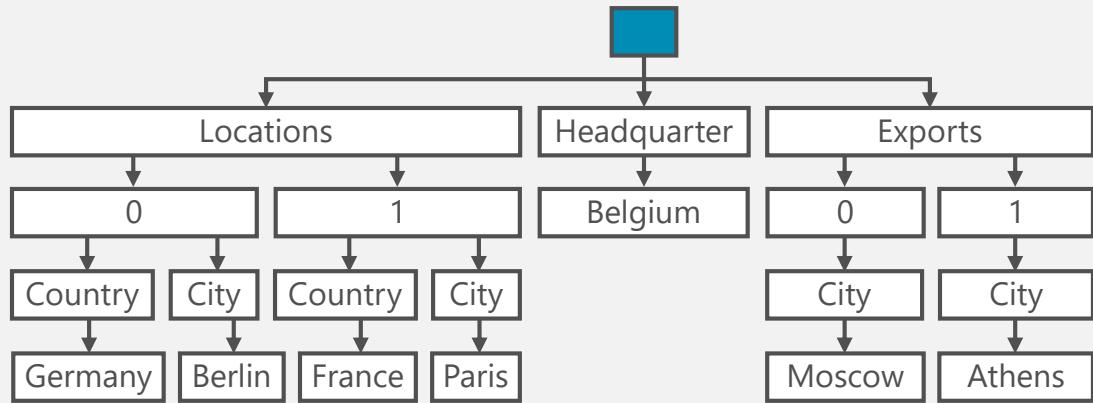
Klienci w .Net, Node, Python, Java i JavaScript
SQL do budowy zapytań, .Net LINQ
Logika po stronie serwera w JavaScript

WAŻNE Kolekcja != tabela

Jednostka partycjonowania
Granica transakcji, jednostka „odpytywania”
Elastyczna, bez schematu

Dokumenty JSON

```
{  
  "locations": [  
    {"country": "Germany", "city": "Berlin"},  
    {"country": "France", "city": "Paris"},  
  ],  
  "headquarter": "Belgium",  
  "exports": [{"city": "Moscow"}, {"city": "Athens"}]  
}
```



JSON

Rozumie to większość współczesnych systemów

Wartości JSON

Samoopisujące się!

Bez dziedziczenia, polimorfizmu, kontroli, ...

Trywialna serializacja na/z tekst

Typy JSON

Number: „double”, trochę ograniczeń w JS

String: znaki Unicode

Boolean: **true** lub **false**

Array: w [] inne obiekty json

null: pusta wartość

DocumentDB a JSON:

Przechowywanie, indeksowanie, kwerendy, motor uruchomieniowy JavaScript

Ciekawostki:

TTL (od daty modyfikacji)

Dane geograficzne ([GeoJSON](#), [WGS-84](#))

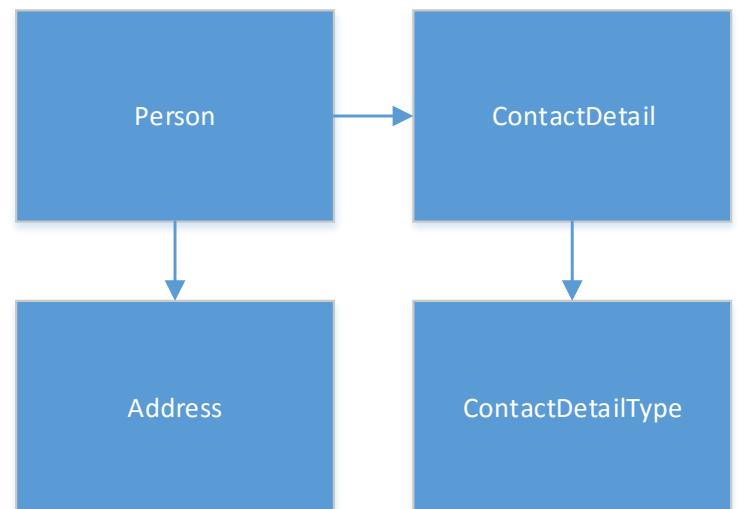
"location": { type": "Point", "coordinates": [-122.12, 47.66] }

Też: linestrings, polygon

Skrót – jak modelować dane ([szczegóły tu](#))

One-To-Few, „Embedded”, denormalizacja:

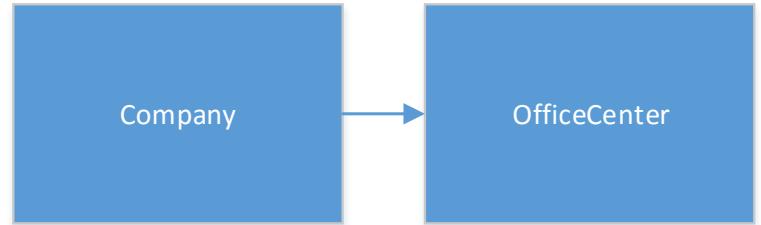
```
[ {  
    "id": "1", "type": "company", "name": "Tomasz",  
    "addresses": [ { "city": "Seattle" }, { "city": "Redmond" } ],  
    "contactDetails": [  
        { "email": "tkopacz@microsoft.com"},  
        { "phone": "+1 555 555-5555", "extension": 5555},  
        { "fax": "+1 222 333-4444" } ]  
}, {  
    "id": "2", "type": "company", "name": "Sędzimir",  
    "addresses": [ { "city": "Warszawa" }, { "city": "Kraków" } ],  
    "contactDetails": [  
        { "email": "tkopacz@microsoft.com"},  
        { "phone": "+1 555 555-5555", "extension": 5555},  
        { "fax": "+1 222 333-4444" } ],  
    "category": "VIP"  
} ]
```



Wbudowana relacja w encje, razem stanowią spójną całość. „one-to-few”. Dane osadzone zmieniają się rzadko (razem z „rodzicem”).

Skrót – jak modelować dane ([szczegóły tu](#))

One To Many (referencing), gdy dużo elementów albo gdy niezależnie aktualizujemy powiązane informacje



```
{
  "companyv2": [
    { "id": "3", "name":"Firma1", "officeCenterRef":"6" },
    { "id": "4", "name":"Firma2", "officeCenterRef":"6" },
    { "id": "5", "name":"Firma3", "officeCenterRef":"7" }
  ],
  "officecenter": [
    { "id": "6", "name":"Budynek1", "address": { "city":"Warszwa", "street": "Al. Jerozolimskie" } },
    { "id": "7", "name":"Budynek2", "address": { "city":"Kraków", "street": "Warszawska" } }
  ]
}
```

Dygresja - kwerenda

```
SELECT c.name as CompanyName, o.name as BuildingName,  
o.address.city as BuildigCity FROM doc JOIN c IN doc.companyv2  
JOIN o IN doc.officecenter WHERE c.officeCenterRef=o.id
```

```
[  
  {  
    "CompanyName": "Firma1",  
    "BuildingName": "Budynek1",  
    "BuildigCity": "Warszwa"  
  },  
  {  
    "CompanyName": "Firma2",  
    "BuildingName": "Budynek1",  
    "BuildigCity": "Warszwa"  
  },  
  {  
    "CompanyName": "Firma3",  
    "BuildingName": "Budynek2",  
    "BuildigCity": "Kraków"  
  }  
]
```

Information

REQUEST CHARGE 4.4 RU

ROUND TRIPS 1

SHOWING RESULTS 1-3

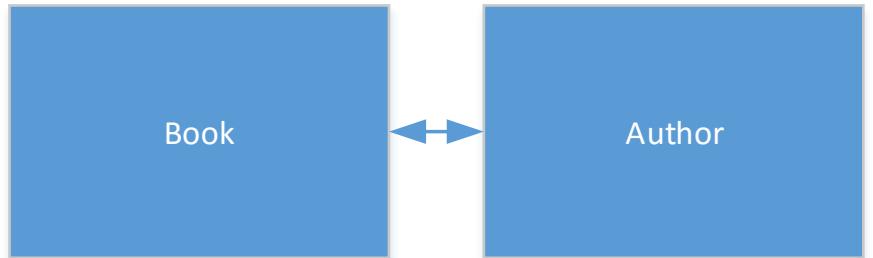
HAS MORE RESULTS false

ACTIVITY ID 6d32752b-3dfd-4859-81a6-0d7761da5400

Skrót – jak modelować dane ([szczegóły tu](#))

Many To Many

```
{  
    "books": [  
        {"id": "8", "title": "Learning Azure DocumentDB", "authors": ["9"] },  
        {"id": "10", "title": "C# 6.0 in a Nutshell, 6th Edition", "authors": ["11", "12"] }  
    ],  
    "author": [  
        {"id": "9", "name": "Riccardo Becker"},  
        {"id": "11", "name": "Joseph Albahari"},  
        {"id": "12", "name": "Ben Albahari"}  
    ]  
}  
  
{  
    "books": [  
        {"id": "15", "title": "C#. Leksykon", "authors": ["12", "14", "13"] }  
    ],  
    "author": [  
        {"id": "13", "name": "Peter Drayton"},  
        {"id": "14", "name": "Brad Merrill"}  
    ]  
}
```



„Mała”, ale WAŻNA uwaga! Po co nam „tabela łącząca” gdy typem pola może być tablica

Autor 12 – istnieje w innym dokumencie

Indeksy w DocumentDB

Na ścieżkach w dokumencie JSON. Typy:

Hash: dla /prop/? (lub /*), używane przez `SELECT * FROM collection c WHERE c.prop = "value"`
dla /props/[]/? (lub /* lub /props/*) używane przez `SELECT tag FROM collection c JOIN tag IN c.props WHERE tag = 5`

Range: `SELECT * FROM collection c WHERE c.prop > 5`

Spatial: `SELECT * FROM collection c WHERE ST_DISTANCE(c.prop, {"type": "Point", "coordinates": [0.0, 10.0]}) < 40`

Można wykluczać – dokumenty, ścieżki itp.

Różnej dokładności

Balans pomiędzy rozmiarem indeksu a wydajnością kwerendy. Dla liczbowych najlepiej -1 (max).

Przełączanie polityk

Pozwolenie na „table scan”

Nagłówek `x-ms-documentdb-enable-scans` albo `EnableScanInQuery` w .NET

Document DB Consistency Levels

(Skalowalne i niezawodne = repliki. 3+ kopie. Read quorum i Write Quorum)

Strong

Wartości widziane po zapisaniu do quorum replik. Odczyt – z read quorum. Najwolniejsze zapisy i odczyty

Session

Klient odczytuje swoje zapisy. Spójność odczytów. Inni mogą zobaczyć starsze wartości. Balans pomiędzy spójnością i prędkością

Bounded staleness

Podobnie jak session, ale określamy jak te dane mogą być stare. Podaje się liczbę operacji (100) i sekund(5)

Eventual

Starsze wartości, może być nie widać własnych zapisów, chaos. Najszybszy zapis

Request Units (jednostka „skalowania”)

1 RU = 1 odczyt lub zapis dokumentu 1KB. 5 RU – zapis 1KB

Tworzenie dokumentu ze 100-ma obiektami JSON i indeksami – np. 20 RU

Limity

Co to ten Request Unit

Kwerendy (na portalu) pokazują ilość RU

Wybiera się za ile maksymalnie RU się zapłaci (za wydajność).

Są jeszcze partycje... (jednostki ACID)

Jak problem z nadmiarem używanych RU:

HTTP Status 429

Status Line: RequestRateTooLarge

x-ms-retry-after-ms :100

“Realne” limity

Pojedyncza partycja: 10GB / 10 000* RU/s

Wiele partycji: 250GB* / 250 000* RU/s

* Please [contact us](#) for storage sizes higher than
250 GB or provisioned throughput over 250k
RU/s.

(co nie zmienia faktu że zwykle lepiej/taniej jest
pomyśleć)

Kolekcja to nie tabela ani schemat „relacyjny”

Demo

Documentdb

Oczywiście – nie tylko DocumentDB

CouchDB

MongoDB – DocumentDB obsługuje protokół MongoDB

RavenDB

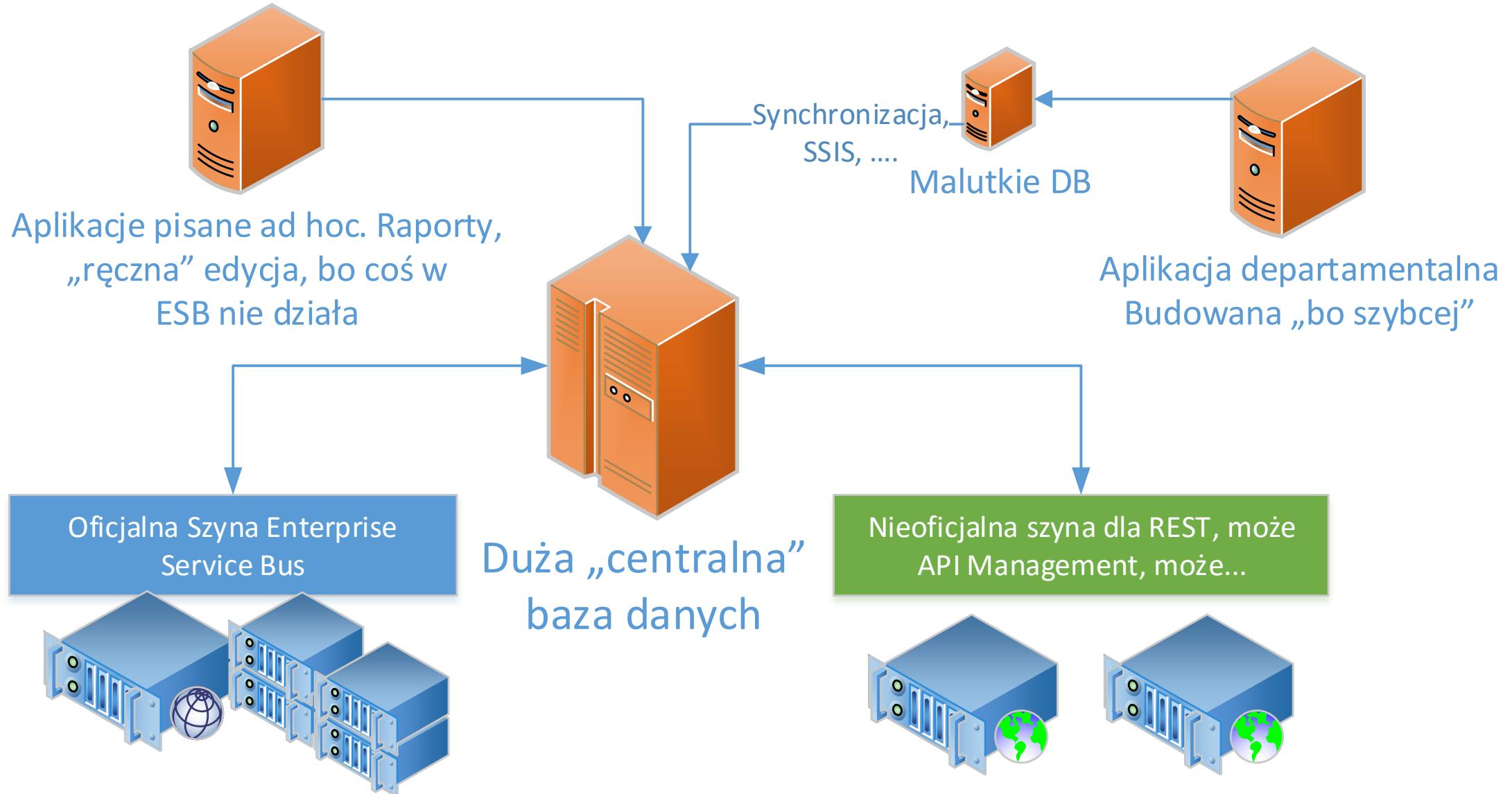
...

Też starsze – Tamino DB (chyba pierwsza baza używająca „natywnie” XML)

Rozszerzenia baz relacyjnych, na przykładzie SQL Server:
typ XML/JSON w kolumnie, głównie dane hierarchiczne
albo postać wyniku kwerendy

Jak to połączyć w aplikacji?

Taka „architektura”, „klasyczna”, Spaghetti OA, REST



Architektura oparta o mikroserwisy

(Jeden ze sposobów modelowania architektury, aktualna „**moda**”)

Budowanie dookoła zdolności biznesowych

Niezależne, samodzielne, w PEŁNI odpowiedzialne za konkretny element rozwiązania

Technologie:

W pełni automatyczny wdrożenie oraz uruchomienie

Aplikacja: zbiór małych usług. Każda może stosować dowolną technologię, **sposób przechowywania danych**, inne framework. Skalowanie wszerz

Każdy serwis działa we własnej przestrzeni (proces | maszyna | kontener)

Lekka komunikacja, zwykle HTTP (REST style). Brak transakcyjności pomiędzy usługami. Pasuje do orkiestracji w stylu [IFTTT](#) lub [Logic Apps](#).

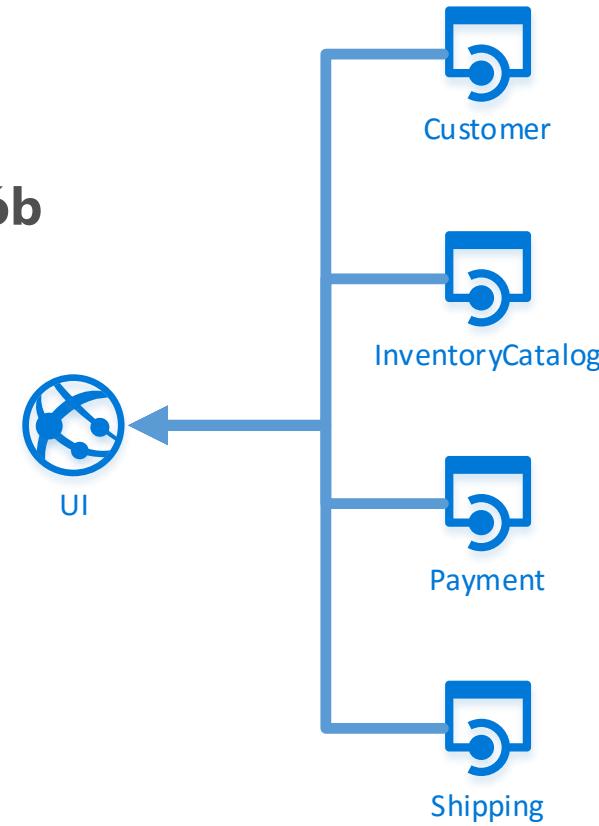
Zespół: komponenty w stylu „Agile”

Nowa osoba szybko staje się produktywna

Aplikacja musi być łatwa do zrozumienia i modyfikacji

Eksperymenty z nowymi technologiami

Nie ma „funkcyjnej” odpowiedzialności – ty jest DBA, ty od UI, ty od myślenia itp..



Modelowanie danych – co do czego

Bierzemy pod uwagę UMIEJĘTNOŚCI ZESPOŁU
NoSQL:

Przykłady: koszyk w sklepie, profile użytkownika, metadane urządzeń, katalogi, logi, dane przestrzenne...

Łatwość konsumpcji z JavaScript

Swoboda!

SQL:

Struktura! Excel! Raporty! Entity Framework! Uczone w szkołach....

DocumentDB – pośrodku

Dokument JSON, hierarchie, referencje, tablice wartości, ...

Ale – podział na kolekcje, partycje – też wymaga zastanowienia się

Przykłady: [Cortana Analytics Gallery](#), [Halo 5 \(część socialna gry\)](#), [News Republic \(czytanie i reagowanie na informacje\)](#), ...

Modelowanie serwisów

WAŻNE: Mikroserwis ODPowiada za dane

Jednostka „transakcyjna” w środku
Z zewnątrz – wywołanie REST

REST

Znamy obiekt (zasób), jego URI (DocumentDB/Azure Table – encja ma URI!)

Standardowe klasy operacji: **GET** (daj) | **POST** (dodaj) | **PUT** (aktualizuj) | **DELETE** (kasuj) | MERGE | HEAD

Zasób

Ale – jeżeli zasób chcemy filtrować, sortować, to jak zbudować język zapytań REST?

Może: używając OData?

Standard tu: <http://www.odata.org/>, REST + formalizacja URI + JSON / BSON / XML /...

Udostępnianie danych relacyjnych (EF + Web API)

Udostępnianie danych „dowolnych” (Web API + kod)

Przykłady zapytań (schemat Northwind):

/Products?\$filter=Price mul 2.0M eq '5.10M,

/Category(1)/Products?\$top=2&\$orderby=name

/Products?\$select=Rating,ReleaseDate

/Products?\$select=Namespace.BestSellingProduct/Spokesperson,Supplier/Namespace.PreferredSupplier/AccountRepresentative

/Customers?\$filter=indexof(CompanyName, 'lfred's) eq 1

/Products?\$filter=not endswith(Name, 'ilk')

Ponieważ mamy sformalizowany model...

```
services.odata.org      +  
services.odata.org/V4/Northwind/Northwind.svc/$metadata  
  
<?xml version="1.0" encoding="UTF-8"?>  
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">  
  <edmx:DataServices>  
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="NorthwindModel">  
      <EntityType Name="Category">  
        <Key>  
          <PropertyRef Name="CategoryID"/>  
        </Key>  
        <Property Name="CategoryID" xmlns:p5="http://schemas.microsoft.com/ado/2009/02/edm/annotation" p5:StoreGeneratedPath="true" Nullable="false" Type="Edm.String" MaxLength="15"/>  
        <Property Name="CategoryName" Nullable="false" Type="Edm.String" MaxLength="max"/>  
        <Property Name="Description" Type="Edm.String" MaxLength="max"/>  
        <Property Name="Picture" Type="Edm.Binary" MaxLength="max"/>  
        <NavigationProperty Name="Products" Type="Collection(NorthwindModel.Product)" Partner="Category"/>  
      </EntityType>  
      <EntityType Name="CustomerDemographic">  
        <Key>  
          <PropertyRef Name="CustomerTypeID"/>  
        </Key>  
        <Property Name="CustomerTypeID" Nullable="false" Type="Edm.String" MaxLength="10"/>  
        <Property Name="CustomerDesc" Type="Edm.String" MaxLength="max"/>  
        <NavigationProperty Name="Customers" Type="Collection(NorthwindModel.Customer)" Partner="CustomerDemographics"/>  
      </EntityType>  
      <EntityType Name="Customer">  
        <Key>  
          <PropertyRef Name="CustomerID"/>  
        </Key>  
        <Property Name="CustomerID" Nullable="false" Type="Edm.String" MaxLength="5"/>  
        <Property Name="CompanyName" Nullable="false" Type="Edm.String" MaxLength="40"/>  
        <Property Name="ContactName" Type="Edm.String" MaxLength="30"/>  
        <Property Name="ContactTitle" Type="Edm.String" MaxLength="30"/>  
        <Property Name="Address" Type="Edm.String" MaxLength="60"/>  
        <Property Name="City" Type="Edm.String" MaxLength="15"/>  
        <Property Name="Region" Type="Edm.String" MaxLength="15"/>  
        <Property Name="PostalCode" Type="Edm.String" MaxLength="10"/>  
        <Property Name="Country" Type="Edm.String" MaxLength="15"/>  
        <Property Name="Phone" Type="Edm.String" MaxLength="24"/>  
        <Property Name="Fax" Type="Edm.String" MaxLength="24"/>  
      </EntityType>
```



Szybkie Demo OData

Ale naprawdę szybkie

A „szybka formatka”?

Problem: aktualność. HTTP = Request/Response

Oczywiście, asynchroniczność, pooling itp.

SignalR – komunikacja dwustronna, realtime po HTTP

Transport: WebSocket, ServerSentEvents, LongPooling, PoolingRequest, AsyncStream

[Kod tu, opis protokołu tu](#)

Przepustowość

Bardzo wysoka wydajność, setki tysięcy komunikatów na sekundę

Bardzo niskie zapotrzebowanie na pamięć

Skalowanie

W pełni asynchroniczny, tysiące połączeń per serwer

Skalowanie przez jakiś message bus (Azure Service Bus, Redis, SQL, Azure Table, ...)

WebHook – (wzorzec), publish/subscribe po HTTP

Subskrybent wysyła URI pod który dostawca ma wysłać komunikat gdy coś zajdzie

Szybkie demo SignalR i WebHook

Ale naprawdę szybkie

Podsumowanie – świat się „skomplikował”

Pojemnik dostosowany do zadań – nieoczywisty wybór!

Mikroserwisy, REST + ODATA

Orkiestracja na poziomie „REST”

Azure Logic Apps, **IF This Then That**

Działa, bo każdy mikroserwis „odpowiada” za czynność – end to end!

Pytanie: transakcja pomiędzy serwisami

Zespół ...

Ps. a raporty?

Screenshot of Power BI Desktop showing a map of the western United States and a bar chart titled "Error Rate by Time (%)".

The "Get Data" dialog box is open, displaying various data sources:

- All
- File (highlighted with a blue arrow)
- Database
- Azure
- Online Services
- Other
- Teradata Database
- Microsoft Azure SQL Database
- Microsoft Azure SQL Data Warehouse
- Microsoft Azure Marketplace
- Microsoft Azure HDInsight
- Microsoft Azure Blob Storage
- Microsoft Azure Table Storage
- Web
- SharePoint Online List
- SharePoint List
- OData Feed
- Hadoop File (HDFS)
- Active Directory
- Microsoft Exchange
- Microsoft Exchange Online
- Dynamics CRM Online

Blue arrows point from the "File" and "OData Feed" items in the list to the respective sections in the main Power BI interface.

The main interface includes:

- Home ribbon tab
- Clipboard, External Data, Insert, View, Relationships, Calculations, Share buttons
- Visualizations pane
- Fields pane
- Map visualization showing California, Nevada, and Utah with city labels like Sacramento, San Francisco, Las Vegas, and St. George.
- Bar chart titled "Error Rate by Time (%)". The Y-axis ranges from 0.0 to 2.5. The X-axis shows time intervals from 11:00 PM to 2:00 AM. The bars show error rates of approximately 1.1%, 0.6%, 1.2%, 2.1%, and 2.2% respectively.
- Page navigation buttons (Page 1, Page 2, etc.)
- Page 1 OF 1

A przykłady tu:

<https://github.com/tkopacz/2016SQLDay>

Dziękuję za uwagę,

Uwagi, komentarze:
tkopacz@microsoft.com



Platinum Sponsor



Strategic Sponsor



Silver Sponsors



Brown Sponsors

