EF Core 5.0

Cosmos DB

Thorsten Kansy (tkansy@dotnetconsulting.eu)

Meine Person- Thorsten Kansy

Freier Consultant, Software Architekt,
Entwickler, Trainer & Fachautor

Microsoft SQL Server

.NET Core

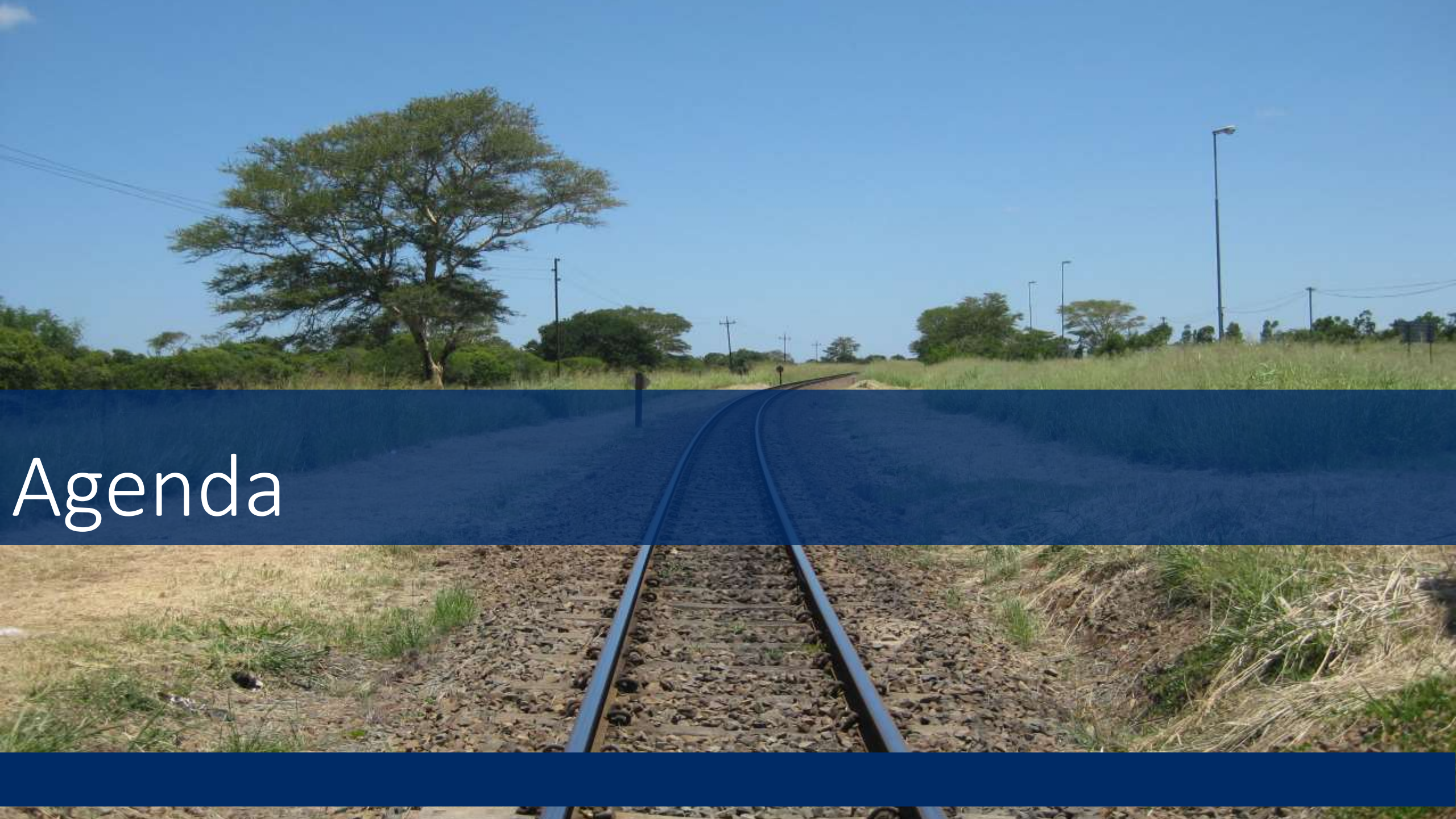
ASP.NET Core

Azure Cosmos DB

DOTNETCONSULTING

Mein Service- Ihr Benefit

- Individuelle Inhouse Trainings
- (Online on-demand) Projektbegleitung
- Beratung
  - Problemanalyse und Lösungen
  - Technologieentscheidungen



Agenda

Agenda

- Grundlagen
  - Vergleich zu RDBMS
  - Aufbau
- DbContext
- Entity Types
- Owned Types
- Shadow States

Grundlagen

RDBMS vs. NoSQL

SQL Server

Festes Schema

Wenig Redundanzen

Vertikal skaliert

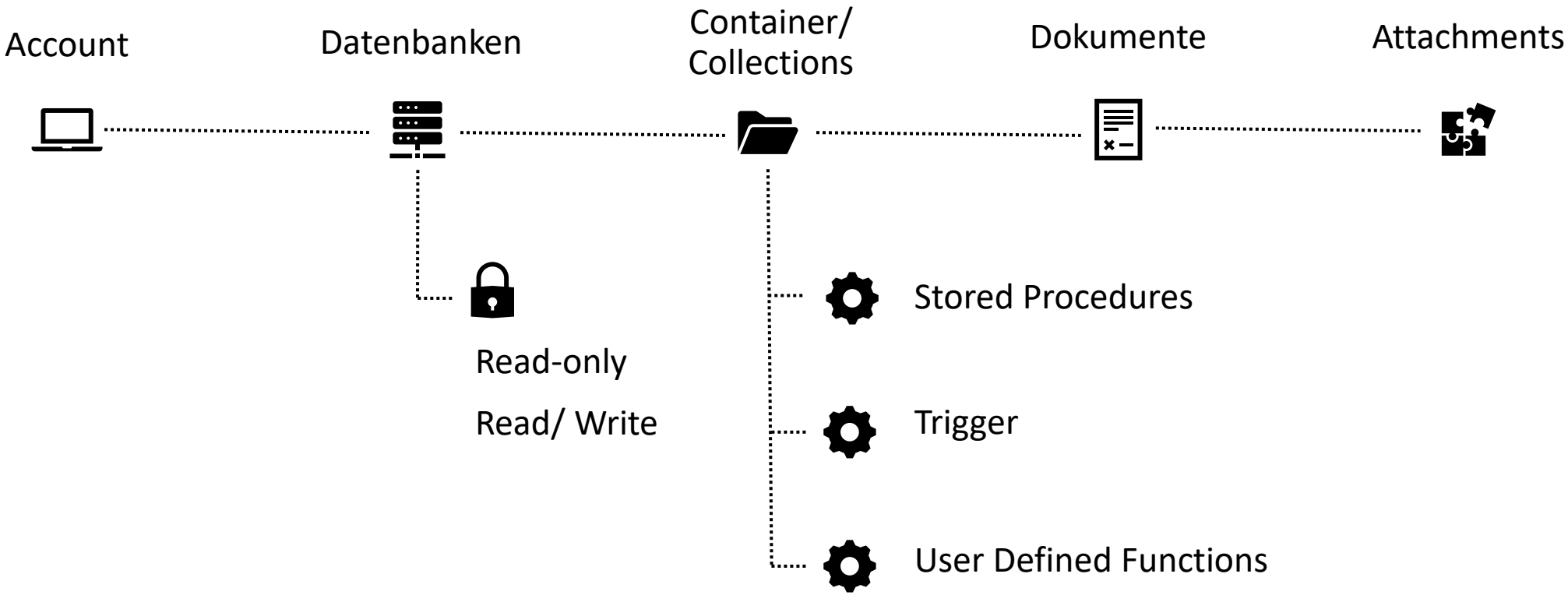
Azure Cosmos DB

Schema frei (NoSQL)

Viele Redundanzen

Horizontal skaliert

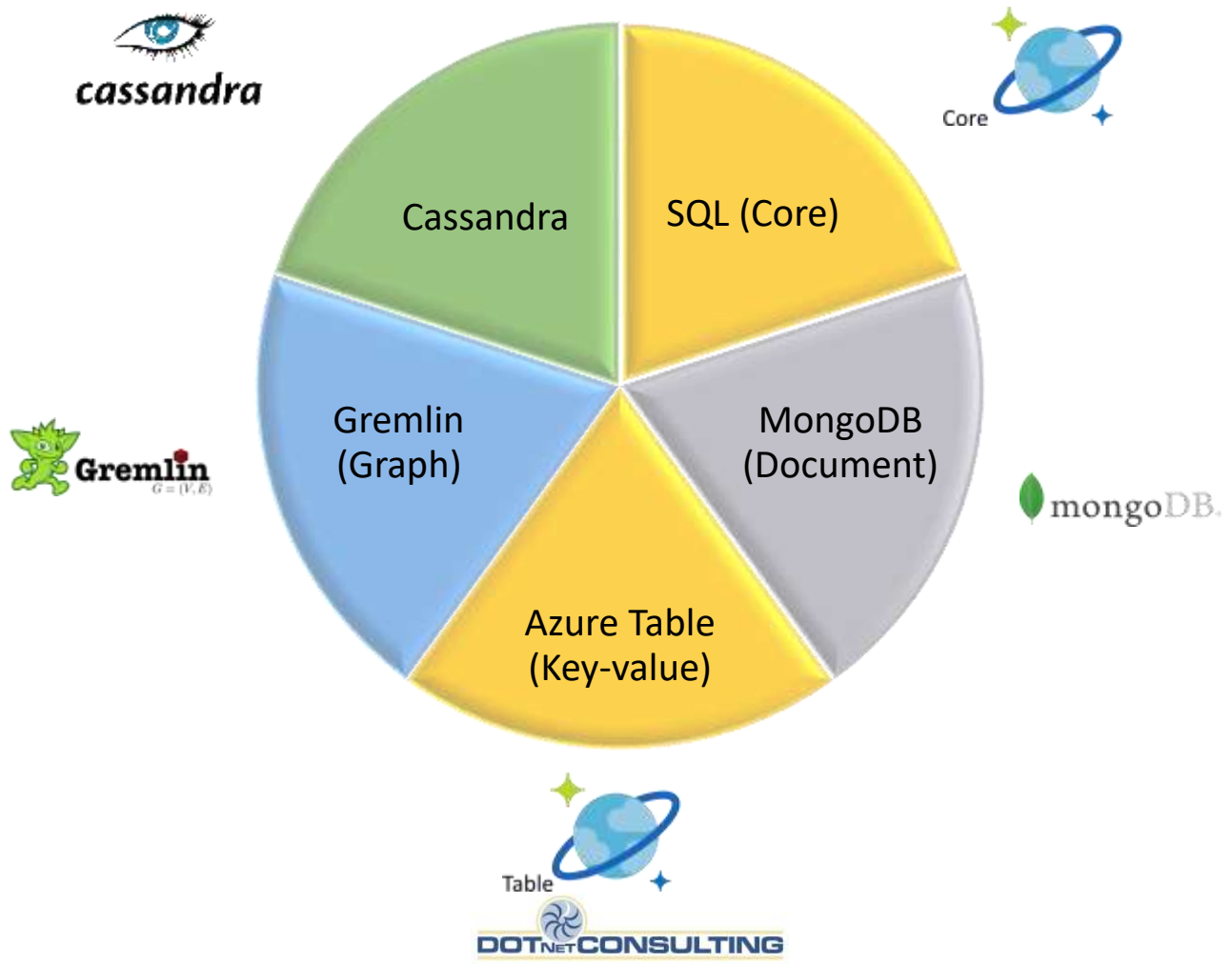
Grundaufbau



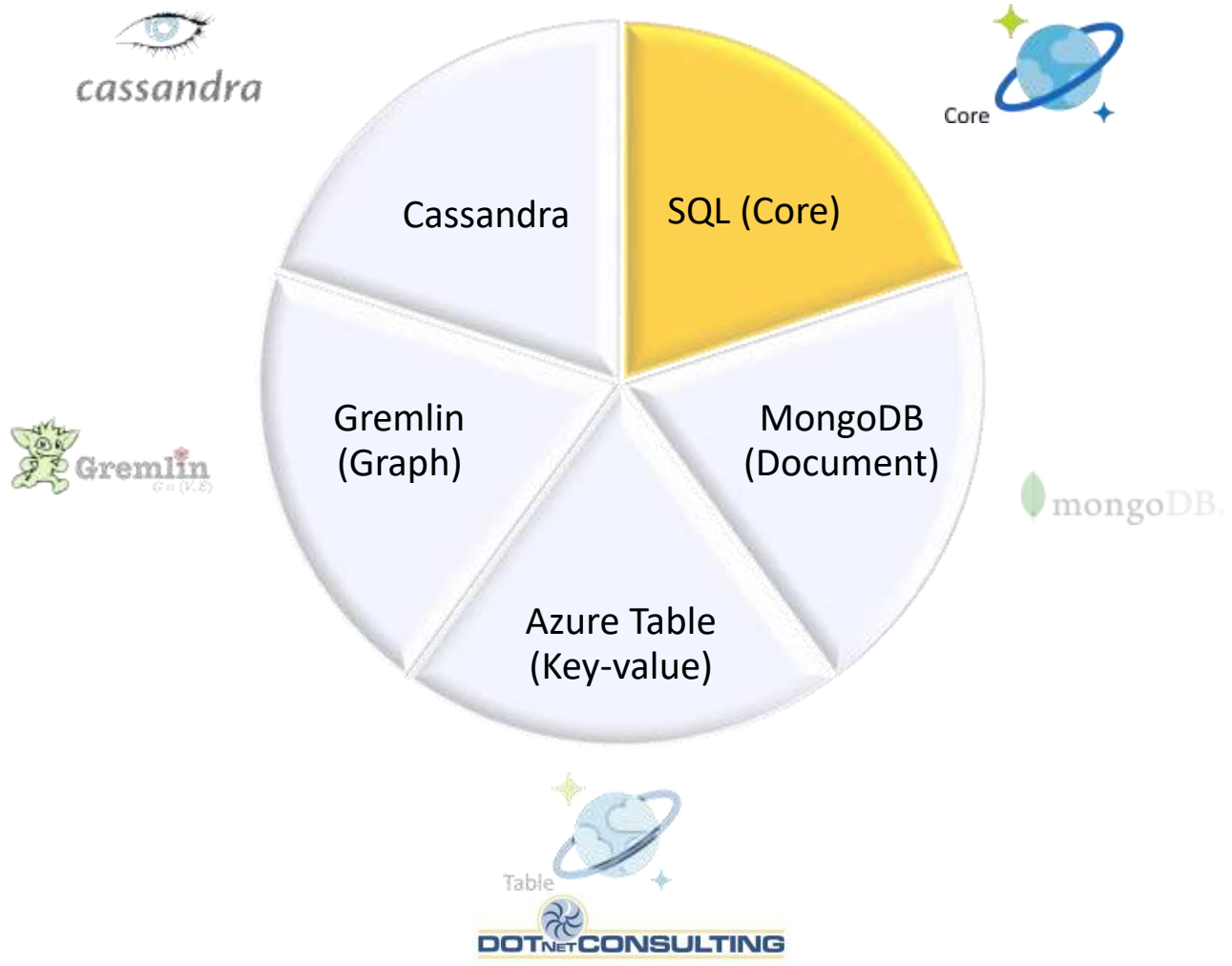
Grundaufbau

- Datenbank
  - Umfaßt 1..n Container/ Collections
- Container/ Collection
  - Umfaßt n Dokumente
  - Einstellungen
    - Partition Key
    - Index Policies
    - Unique Key
  - Stored Procedures
  - User Definied Functions
  - Triggers

Unterstützte APIs

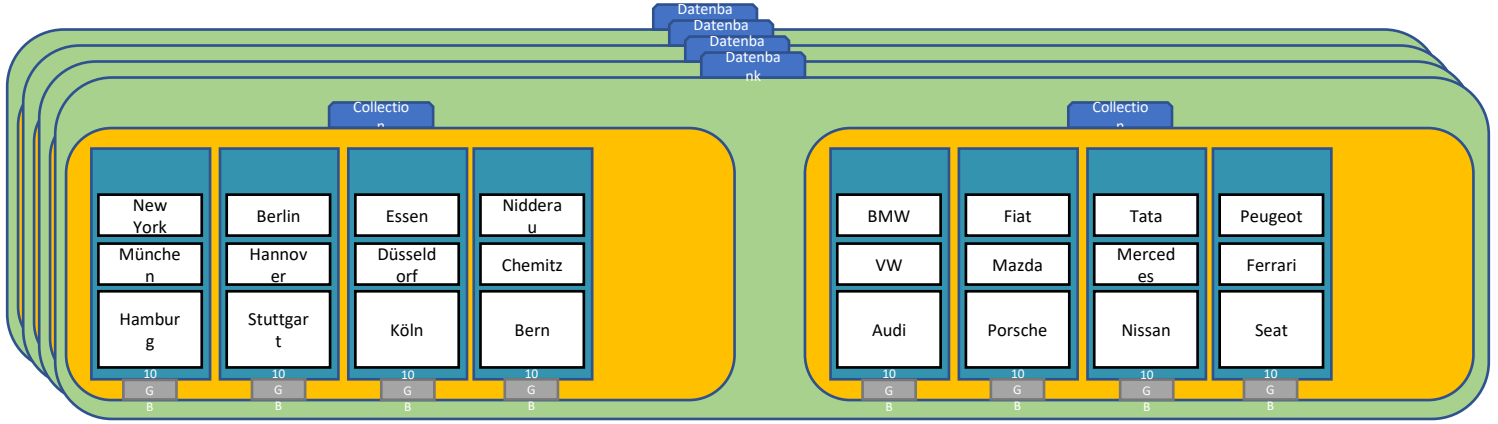


Von EF Core unterstützte API



Physikalischer Ausbau

- Mindestens eine Region pro Datenbank
- Min 4 Instanzen pro Datenbank & Region
  - Lastverteilung
- Automatische Replikation
  - Intra- und Interregional



DOTNETCONSULTING

Anforderungen an das JSON-Dokument

~~Eindeutige id-Eigenschaft (Key)~~

ODER

Eindeutige id-Eigenschaft + PartitionKey

Einige Bezeichner sind verboten

id

- Kleingeschrieben(!)
- GUID (optional)

PartitionKey

- Beliebige Eigenschaft

Performance relevant

Request Units (RUs)

- Request Unit (RU) ≠ Request
- RU
  - Maß für Berechnungsaufwand („Comos DB-Währung“)
  - CPU • Speicher • Disk I/O • Netzwerk I/O
  - Deterministisch
- Reservierte RU/s
  - 400..10.000..∞ (je mehr, desto teurer)
  - Kostenberechnung nach RU/s je Container/ Collection oder Datenbank
  - Bei Überschreitung folgt HTTP 429

```
https://www.documentdb.com/capacityplanner
https://azure.microsoft.com/en-us/pricing/details/cosmos-db/
https://httpstatusdogs.com/
```

SQL für Azure Comos DB

SQL für Azure Comos DB

- Nur lesende Abfragen (SELECT)
- Pflicht zur Qualifizierung
- Case-sensitive (Eigenschaften und Werte)

- Operatoren & eingebaute Funktionen
- Benutzerdefinierte Funktionen

```
https://docs.microsoft.com/en-us/azure/cosmos-db/sql-api-query-reference
```

Sprache Fragmente

- SELECT
  - Rückgabe von Eigenschaften und Berechnungen (inkl. AS)
- WHERE
  - Filter
- ORDER BY
  - (Einfache) Sortierung
- JOIN IN
  - Inner Join mit Array als Sub-Property

Eingebaute Funktionen

Arithmetic	ABS, CEILING, EXP, FLOOR, LOG, LOG10, POWER, ROUND, SIGN, SQRT, SQUARE, TRUNC, ACOS, ASIN, ATAN, ATN2, COS, COT, DEGREES, PI, RADIANS, SIN, TAN
Type	IS_ARRAY, IS_BOOL, IS_NULL, IS_NUMBER, IS_OBJECT, IS_STRING, IS_DEFINED, IS_PRIMITIVE
String	CONCAT, CONTAINS, ENDWITH, INDEX_OF, LEFT, LENGTH, LOWER, LTRIM, REPLACE, REPLICATE, REVERSE, RIGHT, RTRIM,TOSTRING STARTSWITH, SUBSTRING, UPPER
Array	ARRAY_CONCAT, ARRAY_CONTAINS, ARRAY_LENGTH, ARRAY_SLICE
Aggregate	COUNT, SUM, MIN, MAX, AVG
Spartial	ST_DISTANCE, ST_WITHIN, ST_INTERSECTS, ST_ISVALID, ST_ISVALIDDETAILED

Demo

Transaktionen

Transaktionen

- ACID (Atomicity, Consistency, Isolation, Durability)
  - Snapshot Isolation
  - Exception => Rollback & Abbruch

- Transaktionsschutz pro Partition
  - Prozeduren
  - Trigger
  - Funktionen werden in einer Transaktion ausgeführt

```
https://docs.microsoft.com/en-us/azure/cosmos-db/database-transactions-optimistic-concurrency
```

DbProvider in Stellung bringen

DbProvider in Stellung bringen

Nuget, what else?

```
Install-package Microsoft.EntityFrameworkCore.Cosmos
```

```
<ItemGroup>
  <PackageReference Include="Microsoft.EntityFrameworkCore.Cosmos" Version="5.0.0-rc.1.20451.13" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Relational" Version="5.0.0-rc.1.20451.13" />
  <PackageReference Include="Microsoft.Extensions.Logging.Console" Version="5.0.0-rc.1.20451.14" />
  <PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="5.0.0-rc.1.20451.14" />
  <PackageReference Include="Microsoft.Extensions.DependencyInjection" Version="5.0.0-rc.1.20451.14" />
  <PackageReference Include="Microsoft.Extensions.Configuration.Json" Version="5.0.0-rc.1.20451.14" />
</ItemGroup>
```

DbContext

DbContext

- Zentrales Element
- Ausführen von Abfragen
- Hinzufügen/ Entfernen von Entitäten
- Registrierung von Änderungen an Entitäten

Konfiguration

- Verhalten des Kontextes
- Datenbank Provider festgelegt
- Logging

Konfiguration für Azure Comos DB

- Angabe von EndPoint, Key & Database

```
optionsBuilder.UseCosmos("https://localhost:8081",
                         "C2y6yDjf5/R…IZnqyMsEcaGQy67XIw/Jw==",
                         "EFCosmosDb");
```

Konfiguration für Azure Comos DB

5.0

- Plus Konfiguration

```
o.UseCosmos(accountEndpoint, accountKey, databaseName, options =>
{
    // options.WebProxy(myWebProxy);
    options.LimitToEndpoint();
    options.RequestTimeout(TimeSpan.FromSeconds(30));
    options.OpenTcpConnectionTimeout(TimeSpan.FromSeconds(60));
    options.IdleTcpConnectionTimeout(TimeSpan.FromSeconds(300));
    options.GatewayModeMaxConnectionLimit(10);
    options.MaxTcpConnectionsPerEndpoint(10);
    options.MaxRequestsPerTcpConnection(10);
})
```

Demo

Entity Types (Model für Cosmos DB)

HasDefaultContainer (Fluent API)

- Legt den Standardcontainer für alle Objekte fest
- Standard: -

```
modelBuilder.HasDefaultContainer("Books");
```

ToContainer (Fluent API)

- Legt den Container für eine Entität fest
- Standard: Name des Kontextes

```
modelBuilder.Entity<Session>()
    .ToContainer("Sessions");
```

ToJsonProperty (Fluent API)

- Legt den Namen einer Entität im Json-Dokument fest
- Standard: Name der Eigenschaft

```
modelBuilder.Entity<Session>()
    .JsonProperty("TechSession");
```

HasPartitionKey (Fluent API)

- Legt den Namen des PartionKeys fest
- Standard: - (`__partitionKey`)

```
modelBuilder.Entity<Session>()
    .HasPartitionKey(p => p.Difficulty); // Scalar-Eigenschaft
```

IsEtagConcurrency (Fluent API)

5.0

- Legt den Namen den Namen für die Optimistic concurrency-Eigenschaft fest
- Standard: - (`_etag`)

```
modelBuilder.Entity<Session>()
    .Property(p => p.Etag)
    .IsEtagConcurrency(); // Scalar-Eigenschaft
```

Demo

Owned Types

Owned Types (1:0-Elemente)

2.0

Aka Complex Types (Nested Documents bei NoSQL)

Owned-Attribute 2.1

C# Copy

```
[Owned]
public class StreetAddress
{
    public string Street { get; set; }
    public string City { get; set; }
}

public class Order
{
    public int Id { get; set; }
    public StreetAddress ShippingAddress { get; set; }
}
```

Owned Types (n-Elemente)

2.2

```
modelBuilder.Entity<Book>()
    .OwnsMany<Chapter>("Chapters")
    .WithOwner();

modelBuilder.Entity<Book>()
    .OwnsOne<Author>("Author")
    .WithOwner();
```

Demo

Shadow Properties

Shadow Properties

- Eigenschaften, die nicht in der Entität angelegt sind
  - Domain-Driven Design Pattern

```
modelBuilder.Entity<TechEvent>()
    .Property<String>("Code")
    .ToJsonProperty("SecretCode");

Entry(techEvent1).Property("Code").CurrentValue

EF.Property<string>(w, "Code")
```

Demo

Fragen?

Links

http://dotnetconsulting.eu/blog/

@Tkansy

tkansy@dotnetconsulting.eu

www.dotnetconsulting.eu

DOTNETCONSULTING