sidion

MICROSERVICES-SCHNITT – JETZT MIT GRAPHQL!

05.07.2018

Matthias Koch

Zuhören. Analysieren. Beraten.

Agenda

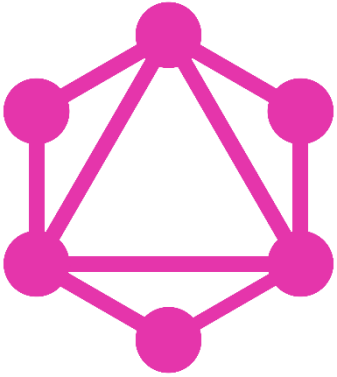
1. Überblick über das Framework
2. GraphQL vs. REST
3. GraphQL als API-Gateway?

Agenda

1. **Überblick über das Framework**
2. GraphQL vs. REST
3. GraphQL als API-Gateway?

Was ist GraphQL?

- Graph Query Language
- Ursprünglich von Facebook als proprietäre API-Technologie entwickelt
- 2015 in Open-Source Projekt umgewandelt
- Lizenziert seit 2017 über Open Web Foundation Agreement (OWFa)
  - Tritt Patentrechte ab
- Im Kern eine Spezifikation
  - Referenzimplementierung in JavaScript
  - Mittlerweile in vielen Sprachen verfügbar



Wer benutzt alles GraphQL?

Quelle: https://www.howtographql.com/basics/0-introduction/

Was sind die Bestandteile von GraphQL?

- GraphQL Schema Language
  - Typisierte Schema Definition Language
  - Definiert Daten und Methoden
  - Ermöglicht starke Entkopplung von Client und Endpoint

- GraphQL Query Language
  - Deklarative Abfragesprache

- GraphQL Runtime
  - Generische Implementierung des Basissets der GraphQL API
    - Transformation einer Query
    - Schema-Validierung
    - Transformation der Rückgabedaten
  - Beliebige Backends

Graph

- Daten einer Applikation können durch einen Graphen beschrieben werden.
- Abfrage der Daten durch Navigation in diesem Graphen.
- In diesen Graphen können auch Zyklen auftreten.

Graph - Beispiel

- Queries können beliebig komplex werden.
- Objekte können gegebenenfalls mehrfach in der selben Query evaluiert werden.

Root Types

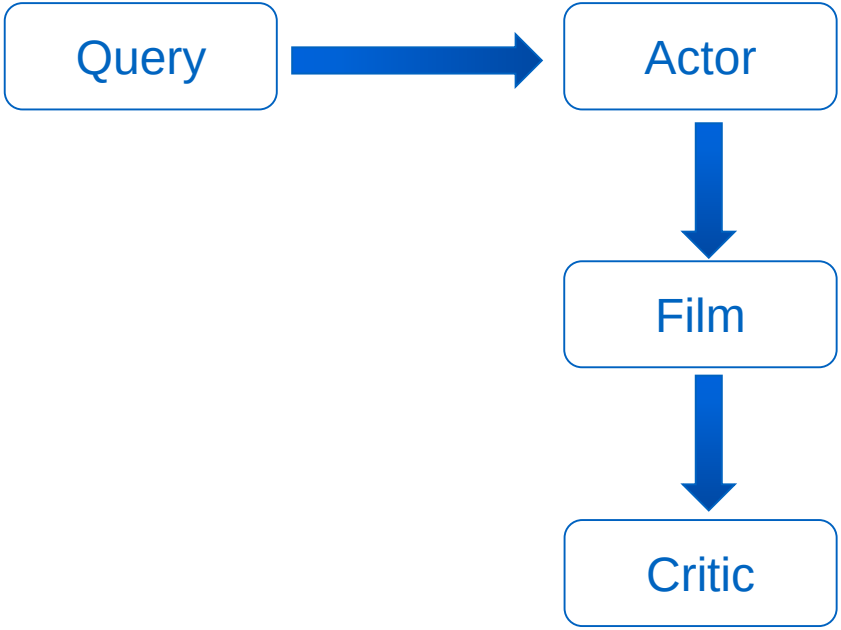
Container für die durchzuführende Funktionen:

- Queries
- Mutators
  - Create
  - Update
  - Delete
- Subscribers
  - Ohne Long Time Polling
  - Ohne Webhooks

GraphQL-Java

- Interessant wegen Migration von Java REST-Endpoints
- Aktuell noch viele Versionssprünge
- Niedrige Downloadzahlen
- Noch nicht durchgängige Dokumentation
- Subscription erst seit kurzem Bestandteil von GraphQL-Java

Beispiel: Movieking



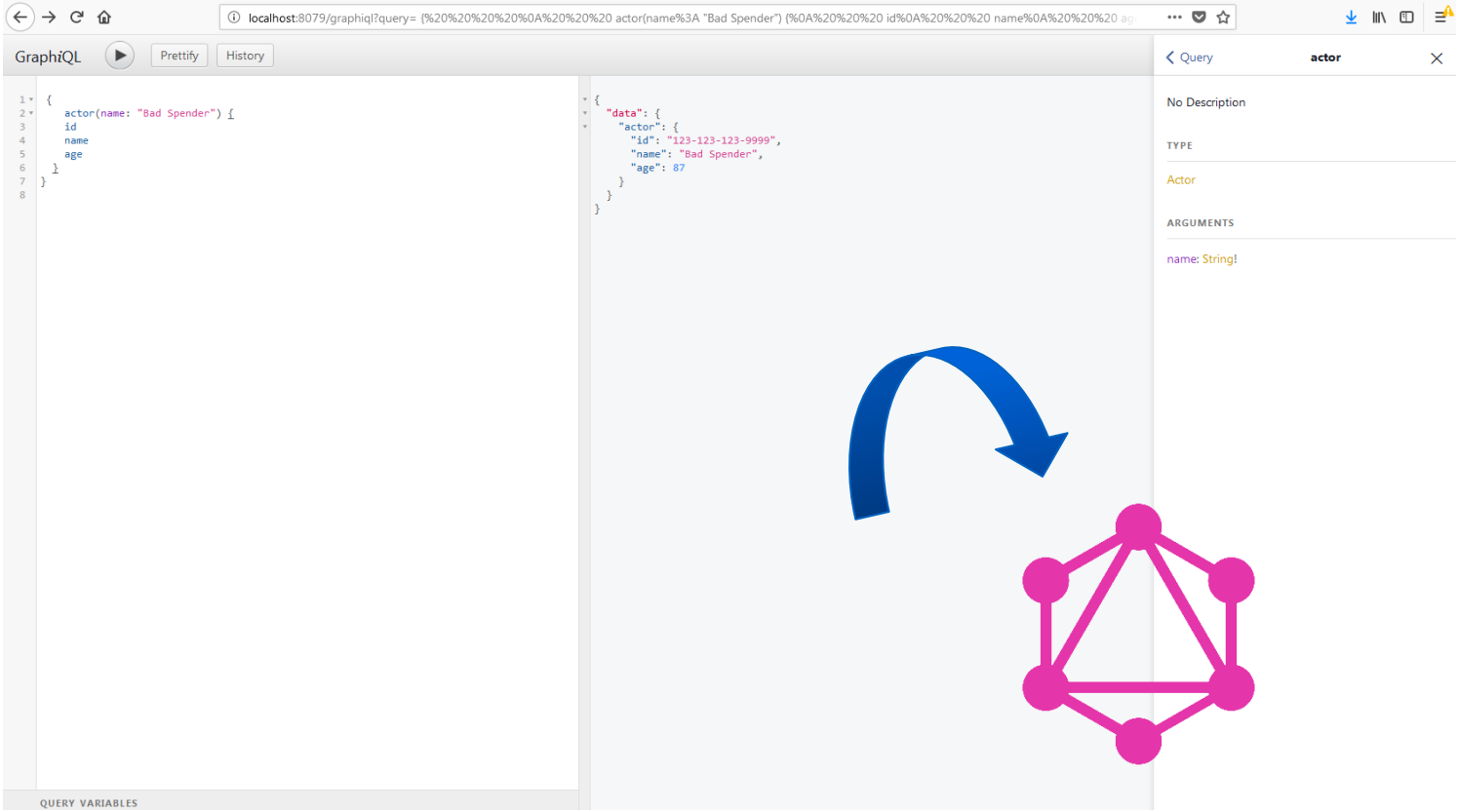
Tool: GraphiQL

- In-Browser IDE
- Queries und Mutations können damit abgesetzt werden.
- Schema kann damit untersucht werden.

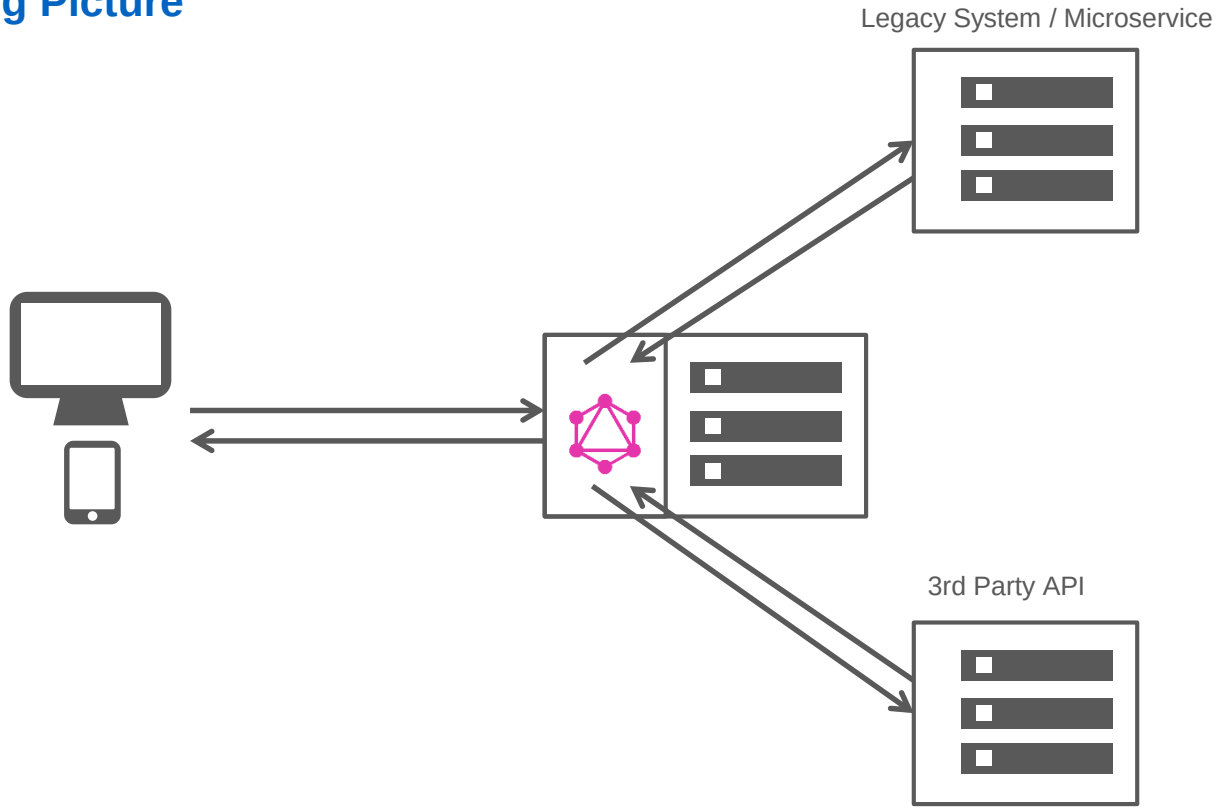
Schema

```
schema {
   query: Query
   mutation: Mutation
}

type Query {
    actor(name: String!): Actor
    film(title: String): Film
}

type Mutation {
    setAge(name:String!, age: Int!) : Actor
}

type Actor {
  id: ID!
  name: String!
  age: Int
  films(title: String): [Film]
  pets: [String]
}

type Film {
  id: ID!
  title: String!
  critics: [Critic]
}

type Critic {
  id: ID!
  title: String!
  lazyBonesSummary: String
  score: Float
}
```

Tool: GraphiQL



Big Picture



DataLoader

- 2010 von Facebook entwickelt
- Referenzimplementierung wieder JavaScript
- Implementierungen in vielen Sprachen auch in Java
- Funktionalität für Batching und Caching
- Simple, intuitive API
- Apache Commons v2.0 Lizenz

java-dataloader

- Alle Felder in GraphQL werden unabhängig evaluiert.
- Bei „naiver“ Implementierung wird die selbe Information ggf. mehrfach berechnet.
- DataLoader hat per default Caching aktiviert.

DataLoader - Simples Beispiel

```
@Test
  public void test_batch_loads() throws Exception {
    BatchLoader<String, String> batchLoader = CompletableFuture::completedFuture;
    DataLoader<String, String> loader = new DataLoader<>(BatchLoader);

    loader.load("A");
    loader.load("B");
    loader.loadMany(asList("C", "D"));

    loader.dispatch();

    Statistics stats = loader.getStatistics();
    assertEquals(stats.getLoadCount(), 4L);
    assertEquals(stats.getBatchInvokeCount(), 1L);
    assertEquals(stats.getBatchLoadCount(), 4L);
    assertEquals(stats.getCacheHitCount(), 0L);

    loader.load("A");
    loader.load("B");

    loader.dispatch();

    stats = loader.getStatistics();
    assertEquals(stats.getLoadCount(), 6L);
    assertEquals(stats.getBatchInvokeCount(), 1L);
    assertEquals(stats.getBatchLoadCount(), 4L);
    assertEquals(stats.getCacheHitCount(), 2L);
  }
```

Agenda

1. Überblick über das Framework
2. **GraphQL vs. REST**
3. GraphQL als API-Gateway?

Movieking REST

GET /movieking.com/actor/name/Buck%20Borris


```
{
  "id": "123-123-123-888",
  "name": "Buck Borris",
  "age": 79,
  "moreFilms": true,
  "films": [
    {
      "title": "Missing the action",
      "links": {
        "rel": "film",
        "href": "movieking.com/film/id/456-456-456-888„
      }
    },
    {
      "title": "Firetruck",
       "links": {
         "rel": "film",
         "href": "movieking.com/film/id/456-456-456-999"
       }
    }, …
  ] ,
  "pets": [
    "stinky“, "winky"
  ]
}
```


GET /movieking.com/film/id//456-456-456-888


```
{
  "id": "456-456-456-888",
  "title": "Missing the action",
  "iinks": [
    {
      "rel": "critic",
      "href": "movieking.com/critic/id/567-567-567-888"
    },
    {
      "rel": "critic",
      "href": "imovieking.com/critic/id/567-567-567-888"
    }
  ]
}
```


GET /movieking.com/critic/id//567-567-567-888


```
{
  ""id": "567-567-567-888",
  "title": "Missing the action",
  "lazyBonesSummary": "Masterpiece of pacifist film",
  "score": 7.46
}
```

Overfetching/Underfetching

- Overfetching
  - Daten werden übermittelt, weil die Ressource immer die gleiche Menge an Daten liefert.
  - Unnütze Daten (z.B. Actor‘s pets) werden immer mit übertragen.

- Underfetching / n+1 Problem
  - Bei großer Menge Daten wird die Zahl der Datensätze begrenzt.
  - Möglicherweise sind die gesuchten Daten nicht enthalten.
    → Mehrere Requests werden ggf. benötigt, um an die gewünschten Daten zu kommen.

Versionierung von GraphQL APIs

- Versionierung wird nicht empfohlen.
- In der Regel wird das Schema größer.
- Macht bei GraphQL nichts, da beim „Declarative Data Fetching“ immer nur die explizit angefragten Daten übermittelt werden.

Direkter Vergleich

Kriterium	GraphQL	REST
Spezifikation	ja	nein
Server-Roundtrips	wenige	viele
Fehlerbehandlung	einfacher	schwerer
Overfetching	nein	ja
Underfetching	seltener	häufiger
Zahl der Endpoints	weniger	mehr
Versionierung	einfacher	schwerer
Typisierung	stark	schwach
Schema	ja	nein
Service-Discovery	nein	ja
Query-Komplexität	hoch	einfach

Alternativen zu REST und GraphQL

- GRPC

- Falcor von Netflix

Agenda

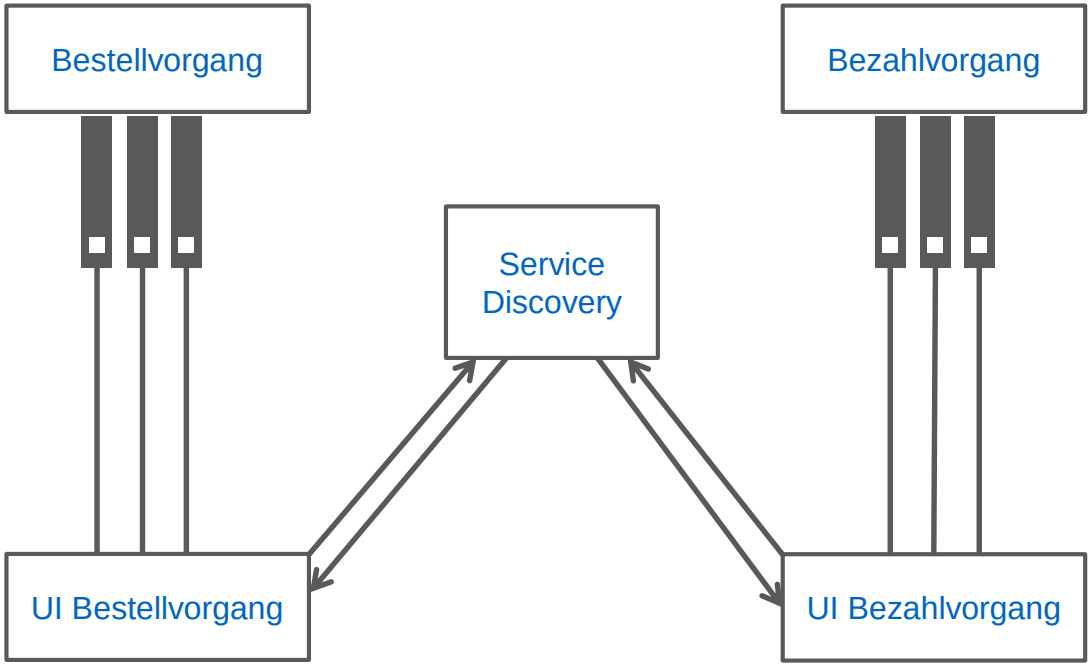
1. Überblick über das Framework
2. GraphQL vs. REST
3. **GraphQL als API-Gateway?**

GraphQL als API-Gateway

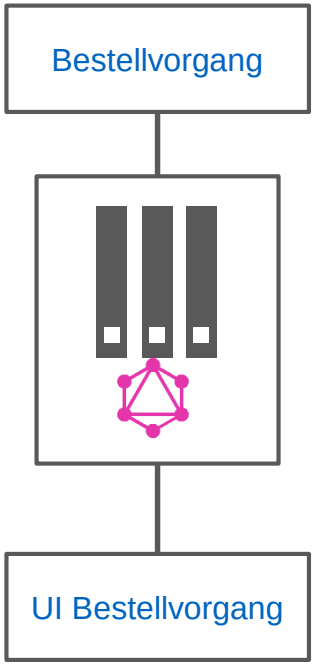
- Vorteile:
  - Alle Daten können mit einem Server-Roundtrip ermittelt werden.
    - einfaches Fehlerhandling
    - einfaches Transaktionshandling

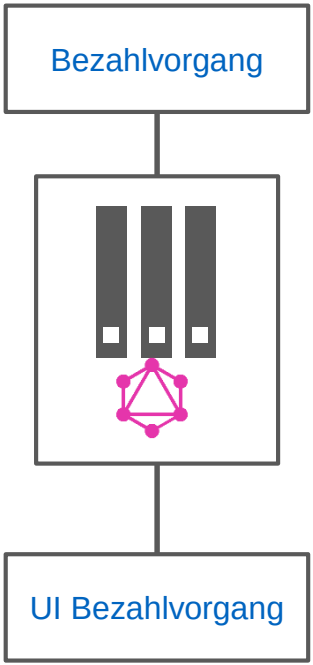
- Nachteil:
  - Absicherung gegen schadhafte Queries schwieriger/risikoreicher

Vertikaler Schnitt

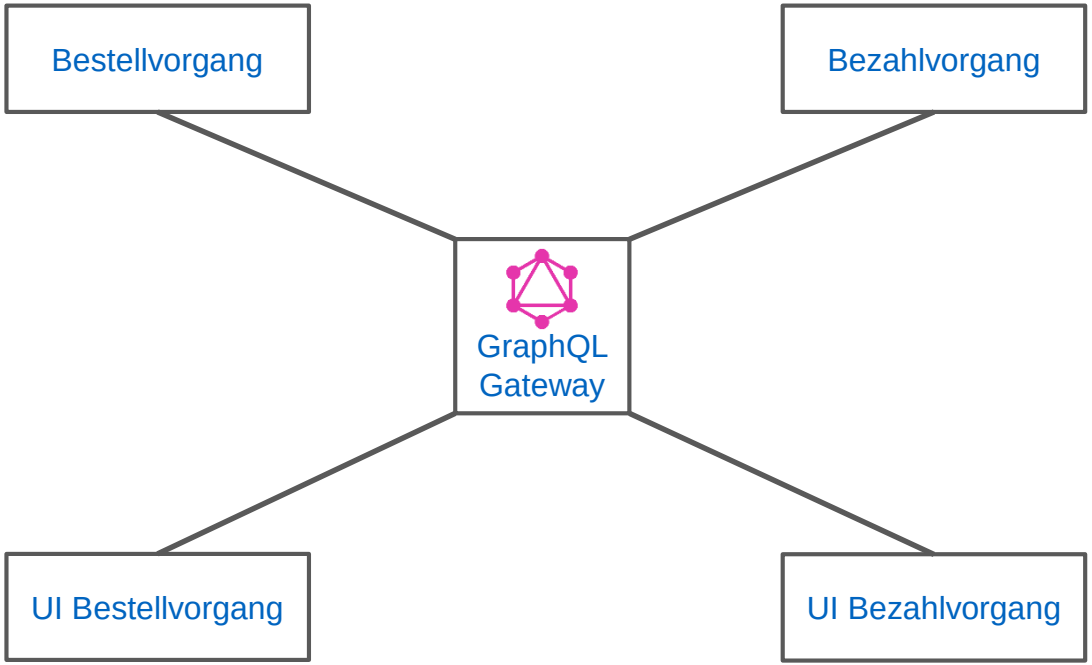


Vertikaler Schnitt mit GraphQL

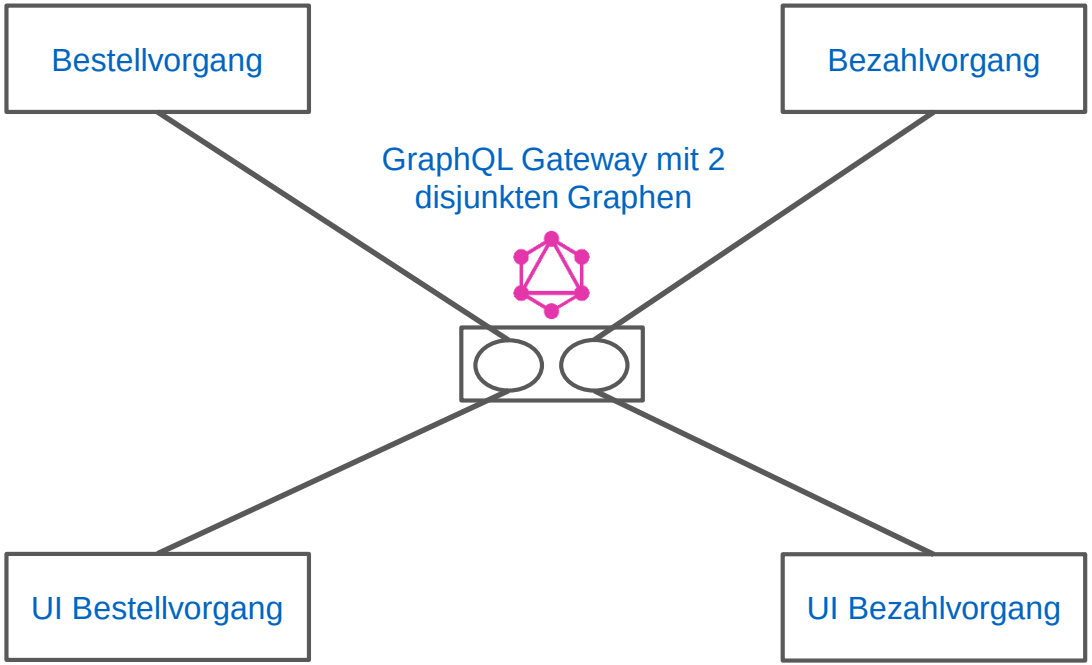




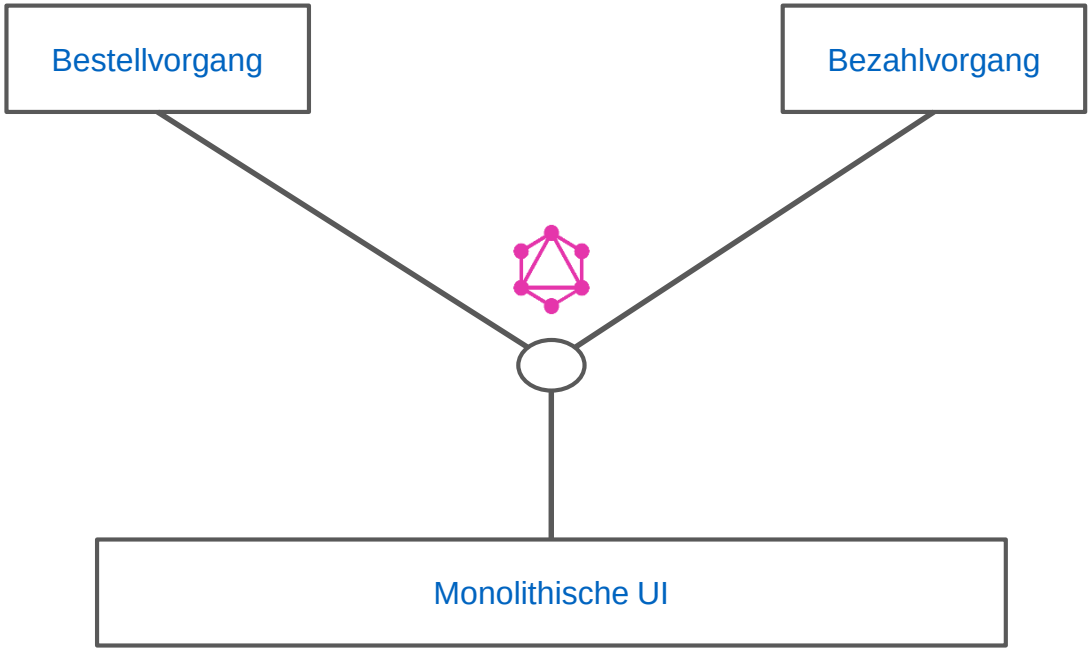
GraphQL als API-Gateway



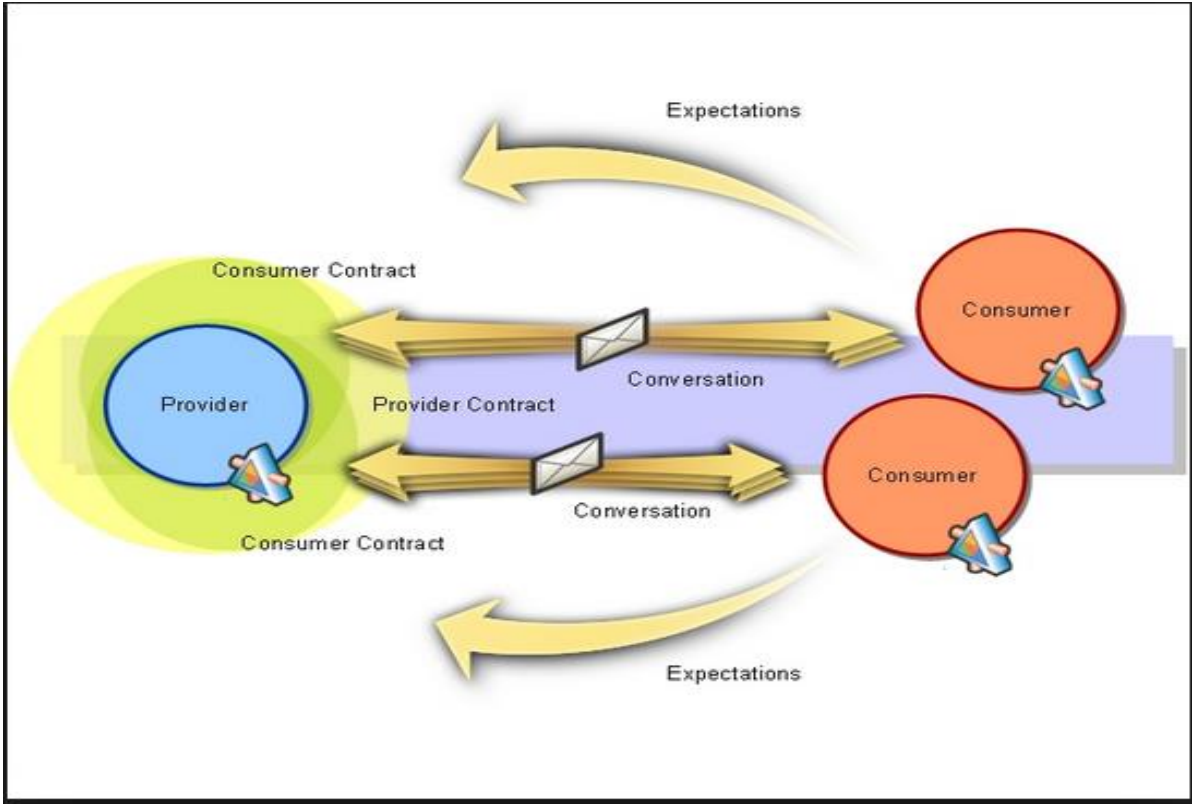
API-Gateway mit disjunkten Graphen



API-Gateway – monolithische UI



Consumer-Driven Contracts

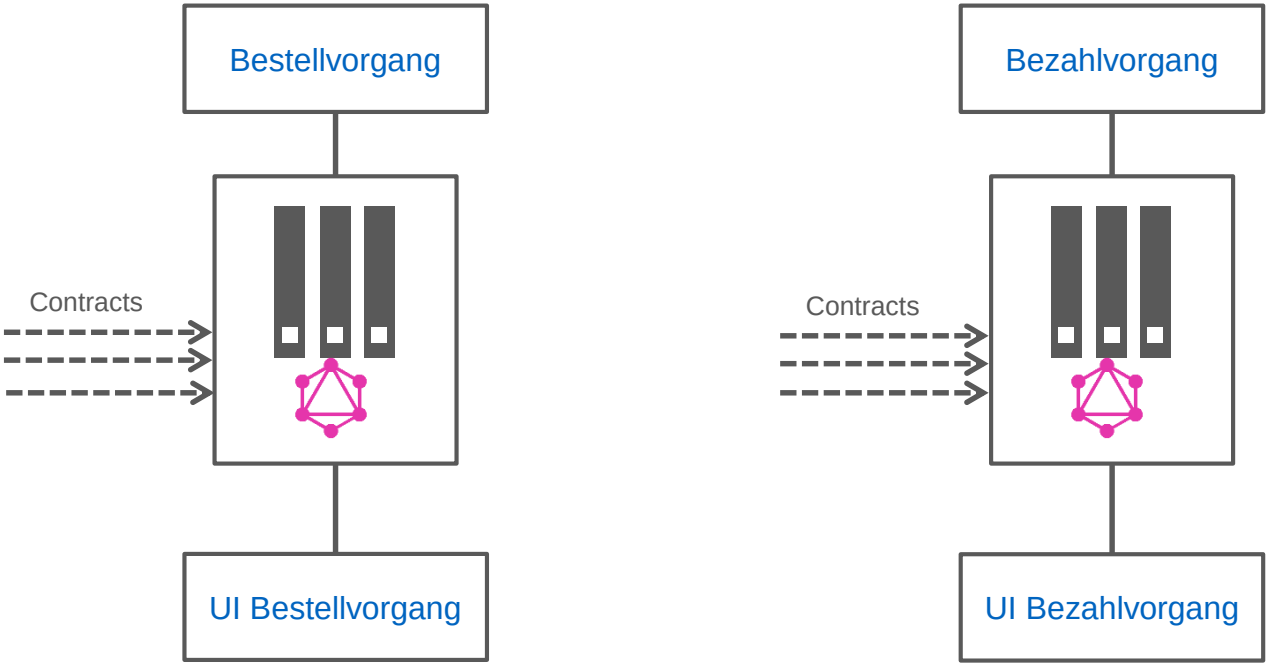


Quelle: https://martinfowler.com/articles/consumerDrivenContracts.html

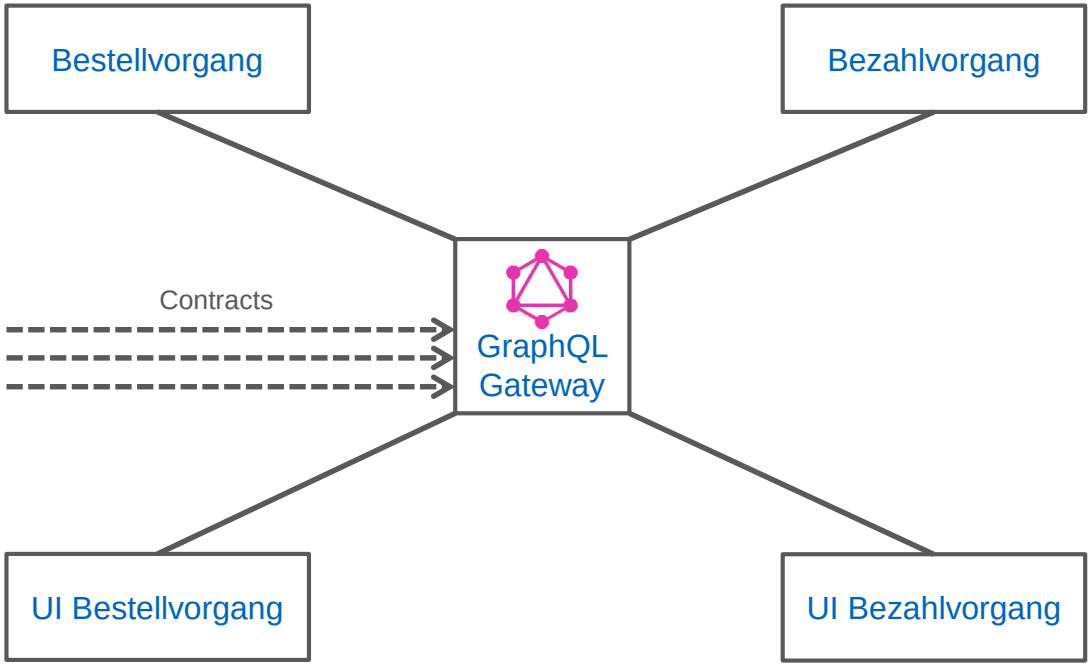
Consumer-Driven Contracts GraphQL

- Gibt es auch für GraphQL: GraphQL Contract Test

Consumer-Driven Contracts vertikaler Schnitt



Consumer-Driven Contracts API-Gateway



Fazit

- GraphQL ist noch eine vergleichsweise junge Technologie.
- Formalisiertes Schema sorgt für starke Entkopplung der Schichten.
- Versionierung der API wird wesentlich erleichtert.
- GraphQL hat gute Konzepte um die Limitierungen von REST aufzulösen.

DANKE!

FRAGEN?

Gerne stehe ich am sidion Stand noch für Gespräche zur Verfügung.

Referenzen

- https://code.facebook.com/posts/121714468491809/relicensing-the-graphql-specification/
- https://github.com/facebook/dataloader
- https://github.com/graphql-java/java-dataloader
- http://graphql.org/
- GraphQL API Design (API-University Series Book 5) (English Edition) Kindle Edition
- https://github.com/symm/graphql-contract-test
- https://grpc.io/
- https://github.com/Netflix/falcor
- https://dev-blog.apollodata.com/the-graphql-stack-how-everything-fits-together-35f8bf34f841

VIELEN DANK.

sidion

Heßbrühlstr. 15
70565 Stuttgart

Tel. + 49 711 490 109 – 0
Fax + 49 711 490 109 – 79

www.sidion.de