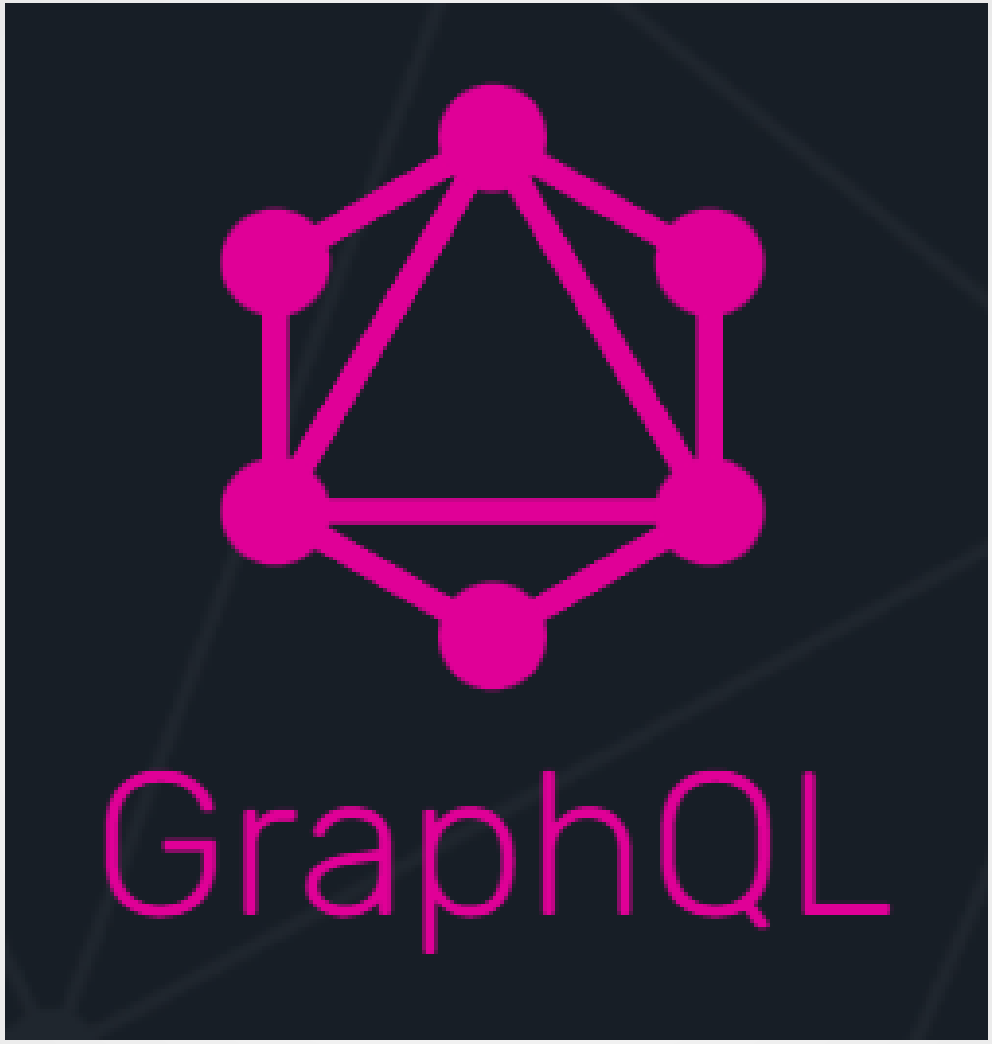

GraphQL

# GraphQL

Überblick

# GraphQL

- Abfragespache für APIs
  am Backend
- Abfrage-Library um die Queries auf
  Livedaten auszuführen

# GraphQL

- Abfragen werden mittels JSON Schema abgesetzt
- Man bekommt nur das, wonach man fragt...
- -> NICHT RESTful

```
{
  hero {
    name
  }
}
```

```
{
  "hero": {
      "name": "Luke Skywalker"
  }
}
```

# GraphQL

Describe your data

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

Ask for what you want

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

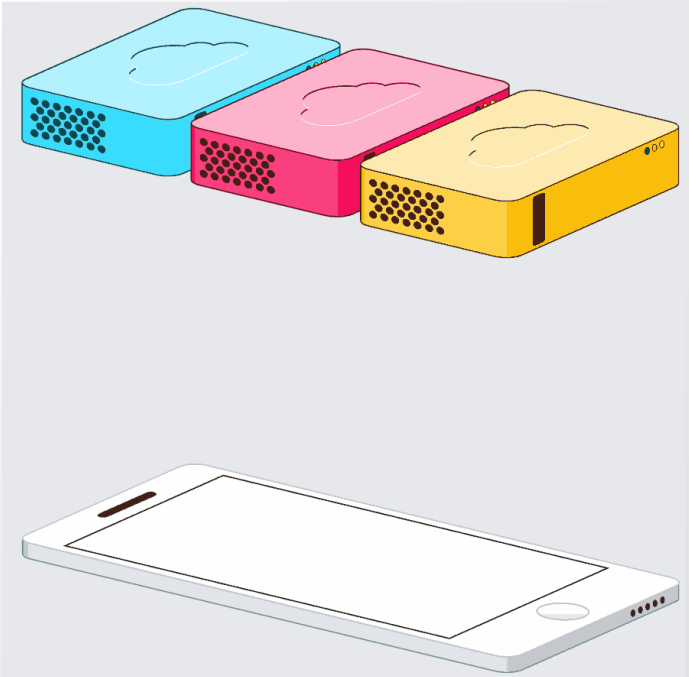
Get predictable results

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

- ...

# GraphQL

- Kann mehrere Ressourcen gleichzeitig abfragen
- Alles mit einem Request
- -> NICHT RESTful



# GraphQL

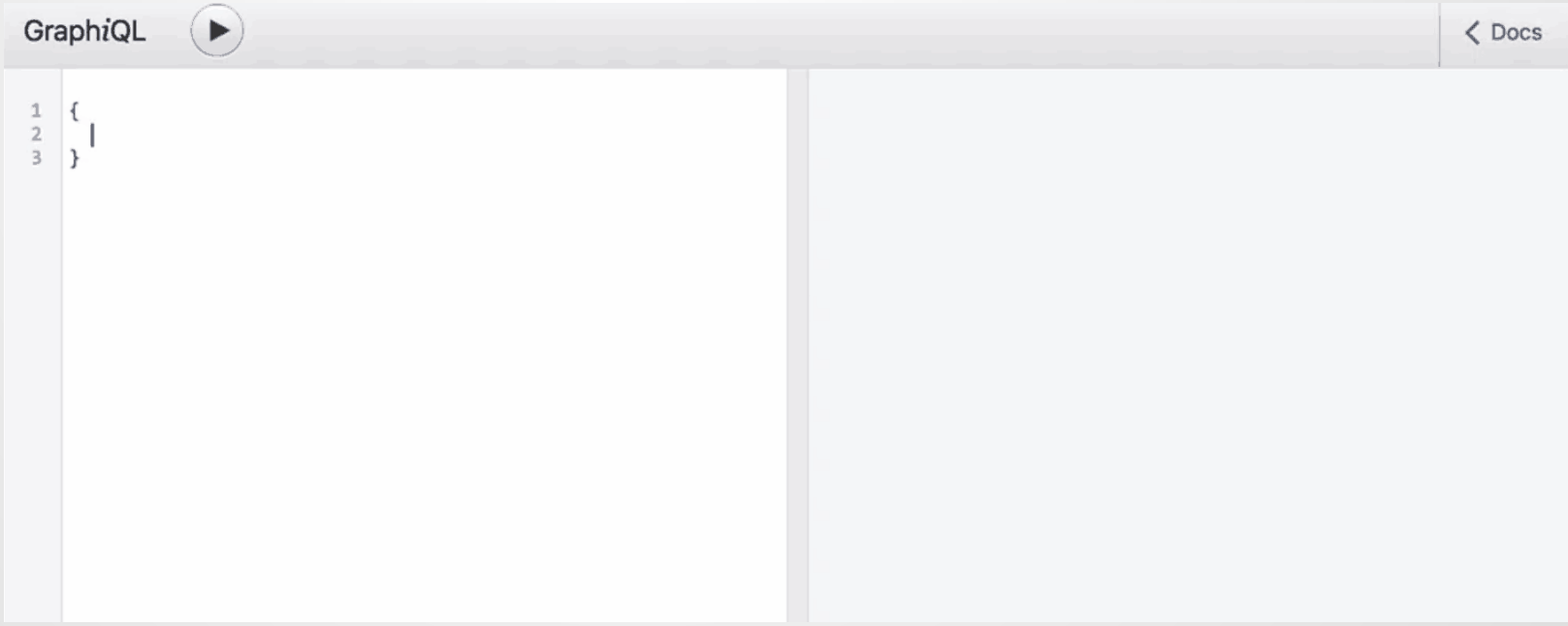
- Basiert auf Typen und Feldern
  Nicht auf Endpoints
- Fragt alles über einen Endpoint ab
- Types -> Felder => Schema

```
{
  hero {
    name
    friends {
      name
      homeWorld {
        name
        climate
      }
      species {
        name
        lifespan
        origin {
          name
        }
      }
    }
  }
}
```

```
type Query {
  hero: Character
}

type Character {
  name: String
  friends: [Character]
  homeWorld: Planet
  species: Species
}

type Planet {
  name: String
  climate: String
}

type Species {
  name: String
  lifespan: Int
  origin: Planet
}
```

# GraphQL

- Es gibt verschiedene Tools, die beim Entwickeln helfen
- Zum Beispiel GraphiQL...



# GraphQL

- Versionierung der API ist nicht notwendig
- Man kann Felder deprecaten oder einfach hinzufügen (bricht nix)

```
type Film {
  title: String
  episode: Int
  releaseDate: String



}
```

# GraphQL

Vorteile / Nachteile

# Vorteile

- Man braucht sich keine Gedanken über RESTful Endpoints machen
- Nur ein Endpoint zu warten
- Language-Agnostisch (Ruby, Python, Scala, Java, Closure, Go, .NET)
- Transport-Agnostisch (nicht zwingend HTTP)
- Datenbank-Agnostisch (gerne NoSQL, Mongo, aber auch MySQL oder eigene DB - DGraph)
- Der Client holt sich immer genau das, was er gerade braucht

# Nachteile

- Nicht RESTful
- Cachen schwierig bis unmöglich, da bei jedem Request quasi alle von allen Ressourcen gleichzeitig angefordert werden können
- Wenn eigene DB (DGraph) benutzt wird ... ev. schwierigere Wartung
- Datenbank-Optimierung wird schwierig bis unmöglich (bei RESTful services sind die Abfragemöglichkeiten eingeschränkter und deswegen ist die Optimierung einfacher)

# GraphQL

Conclusio

# Conclusio

- Gut für schnelle Prototypen
- Toll für Projekte, bei denen sehr freie Abfragen extrem wichtig sind
- Sehr cool als zusätzliche, eingeschränkte Query-API für schon bestehende Projekte

- Nicht RESTful
- Caching ist ein ungelöstes Problem
- Schwierig zu beherrschen, wenn langsam

# References

- https://graphql.org/
-