GraphQL for the Enterprise: Exploiting your REST API Inventory to create better user experiences

Jim Laredo & Erik Wittern, IBM Research
GraphQL is a query language and implementation paradigm for data-centric web APIs

- **Single endpoint:** HTTP POST
- **Clients send queries:** define what data they need (and nothing more)
- **Type-checked:** Complete and understandable description of the data
- **Documentation:** always in sync with implementation
- **Nested and Linked data:** Request many resources in one request
Who Cares?

Selected users...

...and emerging providers

Source: http://graphql.org/users

...all “backend-as-a-service”
REST or GraphQL?

Both

**REST**
- Managing resource lifecycle
- Backend Integration
- API Management support
- Small, cacheable requests

**GraphQL**
- Leveraging Resource relationships
- Connected, graph-like data
- Multiple resources per request
- Fetching data for UIs (mobile)
GraphQL over REST.

Build a universal GraphQL API on top of your existing REST APIs, so you can ship new application features fast without waiting on backend changes.

GET STARTED  SUPPORT + TRAINING
Can we automatically convert existing APIs to GraphQL and build enriched aggregations?

Automatic transition

Queries are automatically resolved as API calls

Data-centric transition

“Links” are translated to graph structure
What is required to automate this conversion?

GraphQL Server

Schema

Resolvers

~id <-> _id
max-weight <-> max_weight
user: name <-> user_name
OASGraph builds a *GraphQL wrapper* in memory for a given OAS

Open-Source (MIT) at https://github.com/strongloop/oasgraph
The GraphQL interface is created around the data definitions in the given OAS, not around the endpoints, leading to a natural use of GraphQL.
Links defined in the OAS are used to compose data definitions.

Nested data
Automatically generated *resolvers* translate (nested) GraphQL queries to API requests. Request results are translated back to GraphQL responses.
OASGraph provides *viewers* to support passing API keys / credentials. OAuth 2.0 integration can be handled by the surrounding application.
Non-safe, non-idempotent API operations (e.g., POST, PUT, DELETE) are translated to GraphQL mutations. Input payload is type-checked.
GraphQL Management

Jim Laredo & Erik Wittern, IBM Research
API Management provides capabilities like threat prevention, rate limiting, pricing, or access control
The challenge for managing GraphQL backends is to understand what requests do

- In **REST APIs**, rates, prices, or access rules are defined for **endpoints**:
  
  ```
  GET .../profiles/me
  ``

  VS.

  ```
  POST .../resources/k8cluster
  ```

  - In **GraphQL**, rates, prices, or access rules depend on the **query**:

  ```
  POST ../graphql
  { me { name, age } }
  ```

  VS.

  ```
  POST ../graphql
  mutation {
    createK8Cluster (name: "c1") {
      clusterId
    }
  }
  ```
Demo
We decouple GraphQL management from GraphQL server implementations
Static query analysis results provide basis for defining higher-level policies

Think threat prevention…

Think access control or pricing…

Think rates…

```
{
    "maxNesting": 2,
    "operationType": "query",
    "resolveCounts": {
        "query:users": 1,
        "user:employerCompany": 5
    },
    "typeCounts": {
        "user": 5,
        "company": 5
    },
    "typeComplexity": 10,
    "resolveComplexity": 6
}
```
Thinking of GraphQL?

- Is GraphQL Management something you started to think about?
- Would you be interested in trying early version of our solution?
- Talk to us!