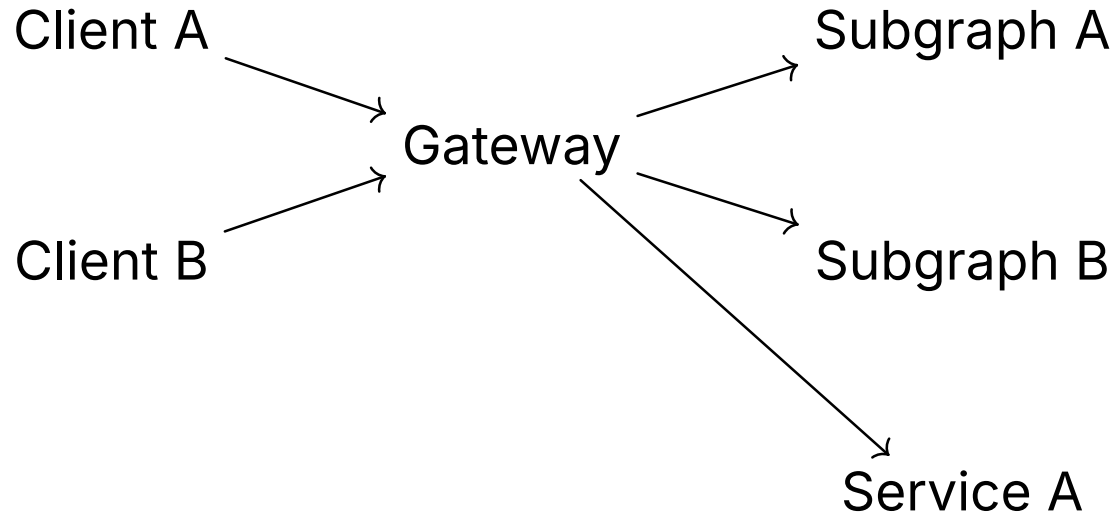




# Authorization in Federated GraphQL

Tom Houlé

# Federated GraphQL



# Why Authorize in the Gateway

- First point of contact to the outside world

# Why Authorize in the Gateway

- First point of contact to the outside world
- Whole schema view

# Why Authorize in the Gateway


- First point of contact to the outside world
- Whole schema view
- Whole request context

# Why Authorize in the Gateway

- First point of contact to the outside world
- Whole schema view
- Whole request context
- Entity resolvers make subgraphs lose context

# Entity resolvers make subgraphs lose context

```
1 query {  
2   currentUser {  
3     friends {  
4       profilePictureUrl  
5       name  
6       photos {  
7         url  
8       }  
9     }  
10  }  
11 }
```

 GraphQL

VS

```
1 query {  
2   _entities(representations:  
3     [{ __typename: "User", id: "1" }]) {  
4     ... on User {  
5       profilePictureUrl  
6       name  
7       photos {  
8         url  
9       }  
10    }  
11 }
```

 GraphQL

# Federation v2 Standard Directives



# Federation v2 Standard Directives

- They work with *claims*
- Claims are derived from:
  - JWT claims
  - Coprocessors

# Federation v2 Standard Directives


```
1 directive @authenticated on
2     FIELD_DEFINITION
3     | OBJECT
4     | INTERFACE
5     | SCALAR
6     | ENUM
```



Allows accessing the field or type when the request carries *any* accepted JWT.

# Federation v2 Standard Directives

```
1 directive @requiresScopes(scopes: [[federation__Scope!]!]) on
2     FIELD_DEFINITION
3     | OBJECT
4     | INTERFACE
5     | SCALAR
6     | ENUM
```

 GraphQL

Allows accessing the field or type when the request has the required scopes/claims.

The outer list wrapper is interpreted as OR. The inner list wrapper is interpreted as AND.

# Federation v2 Standard Directives

```
1 directive @policy(policies: [[federation__Policy!]!]) on
2     FIELD_DEFINITION
3     | OBJECT
4     | INTERFACE
5     | SCALAR
6     | ENUM
```



Calls coprocessors or scripts for the given policies. A policy is just a name. The coprocessor has access to claims and context like request headers.

# Federation v2 Standard Directives

```
1 type Query {  
2     adminDashboard: AdminDashboard  
3     @policy(policies: [  
4         ["ip_not_marked_as_potentially_fraudulent"],  
5         ["is_support_agent", "in_business_hours"],  
6     ])  
7 }
```



## Limitations

- The directives above are sufficient to enable RBAC and limited ABAC. Scopes can match roles, or more targeted permissions.

# Limitations

- The directives above are sufficient to enable RBAC and limited ABAC. Scopes can match roles, or more targeted permissions.
- The authorization decisions can however not be tied to the GraphQL query contents
  - The GraphQL document itself: what fields are requested, and at what paths (example: `User.friends`)
  - The data passed in: arguments from literals and variables

# Limitations

- The directives above are sufficient to enable RBAC and limited ABAC. Scopes can match roles, or more targeted permissions.
- The authorization decisions can however not be tied to the GraphQL query contents
  - The GraphQL document itself: what fields are requested, and at what paths (example: `User.friends`)
  - The data passed in: arguments from literals and variables
- → *Relationships* cannot be enforced
  - Users can see the photos on the profile of their friends
  - I can see the balance on my own bank account
  - I can see the medical records of my own patients



# Comprehensive authorization in the Gateway

# Comprehensive authorization in the Gateway

- We want to make authorization decisions based on:
  - Request data

```
1 query {  
2     user(id: "user_015f91b8-eb7a-418a-8193-f72ddea5760d") {  
3         socialSecurityNumber  
4     }  
5 }
```



# Comprehensive authorization in the Gateway

- We want to make authorization decisions based on:
  - Request data

```
1 query {  
2     user(id: "user_015f91b8-eb7a-418a-8193-f72ddea5760d") {  
3         socialSecurityNumber  
4     }  
5 }
```



- And in some cases in response data too

# Comprehensive authorization in the Gateway

```
1 type User @key(fields: "id") {  
2   id: ID!  
3   email: String!  
4   userType: UserType  
5   socialSecurityNumber: String @policy(policies:  
6     ["check_access_to_user_ssn"])  
6 }
```



Assume we need the `id` and `userType` of the user in addition to the current request context to control access to the social security number.

# Comprehensive authorization in the Gateway

Looks good, but...

```
1 query {  
2   userByEmail(email: "george@pizzahut.com") {  
3     socialSecurityNumber  
4   }  
5 }
```



The `id` and `userType` fields are not going to be available, so our plugin / coprocessor does not have the data it needs to make authorization decisions.

# Query planner involvement

It sounds like we need @requires:

---

Like require, but *without external*.

- Pass the relevant input data

Realization: we sometimes need more than the requested data to make authorization decisions

```
1 query {  
2   apartments {  
3     tenant {  
4       socialSecurityNumber  
5     }  
}
```



```
6 }
```

```
7 }
```

I'm only allowed to see the social security number on tenants.

# Why integration with the query planner matters

- Batching



# ABAC and ReBAC

**Also**

**Also**

Workshop!

**Also**

Workshop! Tomorrow!

**Also**

Workshop! Tomorrow!

Grote Zaal - 2nd Floor.

**Also**

Workshop! Tomorrow!

Grote Zaal - 2nd Floor. 10:45am.

**Also**

Workshop! Tomorrow!

Grote Zaal - 2nd Floor. 10:45am.

Thank you!

# Links