

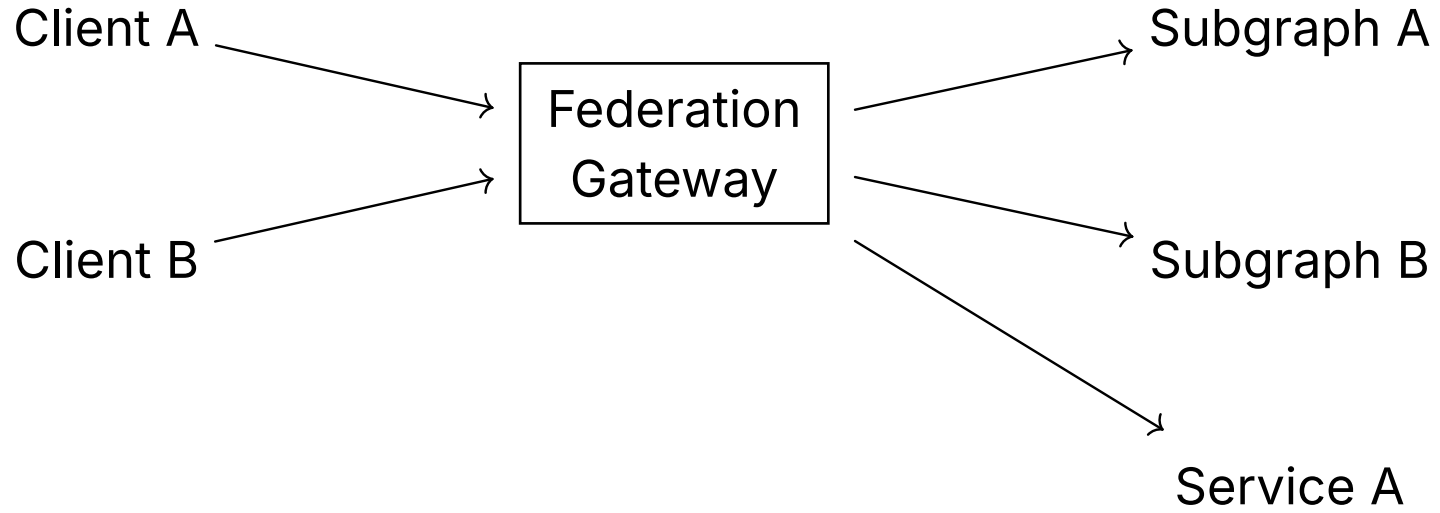


# The Federated GraphQL Subscriptions Zoo



Tom Houlé

# Federated GraphQL



# Subscriptions are special... in GraphQL

"subscription — a long-lived request that fetches data in response to a sequence of events over time"

— [GraphQL spec \(Sept 2025\)](#)

# Subscriptions are special... in GraphQL

"subscription — a long-lived request that fetches data in response to a sequence of events over time"

— [GraphQL spec \(Sept 2025\)](#)

"GraphQL supports type name introspection within any selection set in an operation, with the single exception of selections at the root of a subscription operation."

— [GraphQL spec \(Sept 2025\)](#)

# Subscriptions are special... in GraphQL

"Subscription operations must have exactly one root field.

To enable us to determine this without access to runtime variables, we must forbid the @skip and @include directives in the root selection set."

— [GraphQL spec \(Sept 2025\)](#)

"While each subscription must have exactly one root field, a document may contain any number of operations, each of which may contain different root fields. When executed, a document containing multiple subscription operations must provide the operation name as described in GetOperation()."

— [GraphQL spec \(Sept 2025\)](#)

# **Subscriptions are special... in GraphQL-over-HTTP**

# Subscriptions are special... in GraphQL-over-HTTP

"GraphQL Subscriptions are beyond the scope of this specification at this time."

— [GraphQL over HTTP spec \(draft\)](#)

# Subscriptions are special... in GraphQL-over-HTTP

"GraphQL Subscriptions are beyond the scope of this specification at this time."

— [GraphQL over HTTP spec \(draft\)](#)





**Subscriptions are actually not that special in Federated GraphQL**

# Subscriptions are actually not that special in Federated GraphQL

Schema of the sales subgraph:

```
1 type Product @key(fields: "id") {  
2   id: ID!  
3 }  
4  
5 type Subscription {  
6   productSales: Product  
7 }
```



Schema of the products subgraph:

```
1 type Product @key(fields: "id") {  
2   id: ID!  
3   name: String!  
4 }  
5  
6 type Query {  
7   productById(  
8     id: ID!  
9   ): Product @lookup  
10 }
```



# Subscriptions are actually not that special in Federated GraphQL

Client → Gateway

```
1 subscription ProductSalesWithName {  
2   productSales {  
3     name  
4   }  
5 }
```



Gateway → sales subgraph

```
1 subscription {  
2   productSales {  
3     id  
4   }  
5 }
```



Gateway → products subgraph

```
1 query {  
2   productById(id: $id) {  
3     name  
4   }  
5 }
```



# Subscriptions are actually not that special in Federated GraphQL

Data returned to the client:

```
1  {"name": "Labubu"}
2  {"name": "Labubu"}
3  {"name": "Crocs"}
4  {"name": "Zune"}
5  {"name": "Furbies (12 pack)"}
6  {"name": "Labubu"}
7  {"name": "Google Glass"}
```



# The problems with Federated Subscriptions

- Lack of transport standardisation has led to **fragmentation**:

# The problems with Federated Subscriptions

- Lack of transport standardisation has led to **fragmentation**:
  - WebSockets (HTTP/1.1)
    - Subprotocols with protocol negotiation

```
1 Sec-WebSocket-Version: 13
2 Sec-WebSocket-Protocol: graphql-ws, graphql-transport-ws
```

# The problems with Federated Subscriptions

- Lack of transport standardisation has led to **fragmentation**:
  - WebSockets (HTTP/1.1)
    - Subprotocols with protocol negotiation

```
1 Sec-WebSocket-Version: 13
2 Sec-WebSocket-Protocol: graphql-ws, graphql-transport-ws
```

# The problems with Federated Subscriptions

- Lack of transport standardisation has led to **fragmentation**:
  - WebSockets (HTTP/1.1)
    - Subprotocols with protocol negotiation

```
1 Sec-WebSocket-Version: 13
2 Sec-WebSocket-Protocol: graphql-ws, graphql-transport-ws
```

- SSE (HTTP/2 and 3)



# The problems with Federated Subscriptions

- Lack of transport standardisation has led to **fragmentation**:
  - WebSockets (HTTP/1.1)
    - Subprotocols with protocol negotiation

```
1 Sec-WebSocket-Version: 13
2 Sec-WebSocket-Protocol: graphql-ws, graphql-transport-ws
```

- SSE (HTTP/2 and 3)
- Multipart

# The problems with Federated Subscriptions

- Lack of transport standardisation has led to **fragmentation**:

- WebSockets (HTTP/1.1)
  - Subprotocols with protocol negotiation

```
1 Sec-WebSocket-Version: 13
```

```
2 Sec-WebSocket-Protocol: graphql-ws, graphql-transport-ws
```

- SSE (HTTP/2 and 3)
- Multipart
- **Resource consumption**: one connection between the Gateway and the relevant subgraph per subscribed client, even when they all subscribe to the same events

# The problems with Federated Subscriptions

- Lack of transport standardisation has led to **fragmentation**:

- WebSockets (HTTP/1.1)
  - Subprotocols with protocol negotiation

```
1 Sec-WebSocket-Version: 13
```





```
2 Sec-WebSocket-Protocol: graphql-ws, graphql-transport-ws
```

- SSE (HTTP/2 and 3)
- Multipart
- **Resource consumption**: one connection between the Gateway and the relevant subgraph per subscribed client, even when they all subscribe to the same events
- Multi-protocol subscriptions

# Multi-protocol subscriptions

-  Client —  → Gateway —  →  Subgraph





# Multi-protocol subscriptions

-  Client —  → Gateway —  →  Subgraph
- In the gateway, translations between:
  - SSE,
  - WebSockets
    - subscriptions-transport-ws
    - graphql-ws / graphql-transport-ws
- And different handshake shapes between each!
  - Headers vs websocket init payload shape mismatch

```
1 interface ConnectionInitMessage {  
2   type: 'connection_init';  
3   payload?: Record<string, unknown> | null;  
4 }
```

TS

# Multi-protocol subscriptions

-  Client —  → Gateway —  →  Subgraph
- In the gateway, translations between:
  - SSE,
  - WebSockets
    - subscriptions-t
    - graphql-ws / gra
- And different hands
  - Headers vs webs

```
1 interface Connect
2   type: 'connect'
3   payload?: Rec
4 }
```



TS

## Alternative: connect the gateway to a message queue

- The idea: the GraphQL federation gateway connects to a message queue (Kafka, NATS, ...), not the subgraphs directly
  - The subgraphs or other services post messages to that queue
- Two implementations
  - [Cosmo EDFS](#)
  - [Grafbase extensions](#)

# Grafbase Extensions

- Pluggable gateway extensions compiled to WebAssembly (WASI preview 2)
  - Define their own directives that will be used by the Gateway for query planning
  - Near-native performance, in-process secure sandbox.
  - Can perform arbitrary IO (but you can restrict that with permissions).
  - Open source extensions from the Grafbase Marketplace or build your own
  - They can act as **virtual subgraphs**



```
1  extend schema
2    @link(
3      url: "https://specs.grafbase.com/composite-schemas/v1"
4      import: ["@key", "@derive"]
5    )
6    @link(
7      url: "https://extensions.grafbase.com/extensions/nats/0.4.1"
8      import: ["@natsPublish", "@natsSubscription"]
9    )
10
11 input SellProductInput {
12   productId: ID!
13   price: Int!
14 }
15
16 type Mutation {
17   sellProduct(input: SellProductInput!): Boolean!
18   @natsPublish(
19     subject: "productSales",
20     body: { selection: "*" })
21 }
22
```



```
23 type Product @key(fields: "id") {
24   id: ID!
25 }
26
27 type ProductSale {
28   productId: ID!
29   product: Product! @derive
30   price: Int!
31 }
32
33 type Subscription {
34   sales(subject: String!): ProductSale
35     @natsSubscription(
36       subject: "{{ args.subject }}"
37       selection: "select(.price > 10)"
38     )
39 }
```

## Corresponding configuration

```
1 [extensions.nats]
2 version = "0.4.1"
3
4 [[extensions.nats.config.endpoint]]
5 servers = ["nats://localhost:4222"]
```

# Advantages of an extensions-based approach compared to EDFS

- Arbitrary data formats for the messages (not only JSON)
- Customizable and extensible without forking the Gateway. You can write extensions for other pub/sub systems (Kinesis, etc.).
- More powerful filters (jq expression language)

# Takeaways

# Takeaways

- Federated GraphQL subscriptions require some thinking and planning.

# Takeaways

- Federated GraphQL subscriptions require some thinking and planning.
- Pros of traditional federated subscriptions
  - **Reuse**: federate existing GraphQL subgraphs, no need to modify them
  - **Control**: subscription fields are managed directly in your own GraphQL subgraph

# Takeaways

- Federated GraphQL subscriptions require some thinking and planning.
- Pros of traditional federated subscriptions
  - **Reuse**: federate existing GraphQL subgraphs, no need to modify them
  - **Control**: subscription fields are managed directly in your own GraphQL subgraph
- Pros of subscriptions offloaded to a message queue
  - Stream deduplication
  - Non-GraphQL services can publish to subjects directly
  - Usually **higher performance**, lower memory footprint



## Takeaways

- Federated GraphQL subscriptions require some thinking and planning.
- Pros of traditional federated subscriptions
  - **Reuse**: federate existing GraphQL subgraphs, no need to modify them
  - **Control**: subscription fields are managed directly in your own GraphQL subgraph
- Pros of subscriptions offloaded to a message queue
  - Stream deduplication
  - Non-GraphQL services can publish to subjects directly
  - Usually **higher performance**, lower memory footprint

**You can mix and match both approaches**

**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!



# **Appendices**



# Links

- WebSockets
  - [subscriptions-transport-ws](#)
  - [Issues and security implications with subscriptions-transport-ws](#)
- SSE
  - [GraphQL-SSE spec](#)
- Multipart subscriptions
  - [Incremental delivery over HTTP](#)
  - [Apollo docs](#)
- [Grafbase extensions](#)
- [Cosmo EDFS](#)
- [Pen Pineapple Apple Pen](#)