Extending Graphql Federation Support to WSO2 API-M

Design Document
Version 1.0.0

Prepared On: 16.03.2024

Table of Contents

1. Overview

This document outlines extending GraphQL Federation support to the API Manager. GraphQL Federation is a methodology for composing multiple GraphQL services into a single, cohesive API. Integrating this federation into the API Manager will enable efficient management and orchestration of federated GraphQL services.

2. Introduction

2.1. What is GraphQL Federation

GraphQL Federation is an architecture model that allows multiple GraphQL services, known as subgraphs or federated services, to be combined into a single schema or API.This unified data graph enables clients to query and receive responses from multiple services using a single request. Apollo, a prominent company in the GraphQL ecosystem, pioneered this concept to facilitate the seamless integration of various microservices and data sources.

2.2. Benefits of GraphQL Federation

- Increased Developer Productivity: The Federation promotes collaboration and accelerates development by separating concerns and allowing teams to work independently on their respective domains.
- Increased flexibility: By splitting the schema into smaller subgraphs, developers can make changes to the API more easily and without affecting other parts of the schema.
- Scalability: As the API grows in complexity, it can become challenging to manage a single, monolithic schema. With GraphQL Federation, developers can split the schema into smaller, more manageable parts that can be scaled independently.
- Better separation of concerns: By delegating parts of the schema to other services, developers can focus on their areas of expertise and not worry about the implementation details of other API parts.

2.3. How does Apollo support graphql federation?

- **Subgraph Creation**
  1. First Subgraph schema is defined. Each subgraph has a distinct schema that indicates which types and fields of your composed supergraph it can resolve. The use of federation directives (@key, @external) is important here.
  2. Further associated resolvers should also be defined for resolution process
  3. buildSubgraphSchema function from the @apollo/subgraph package is used to augment schema definition with federation support.

* Note : The `buildSubgraphSchema` function takes an object containing `typeDefs` and `resolvers` and returns a federation-ready subgraph schema. Implementation is shown here

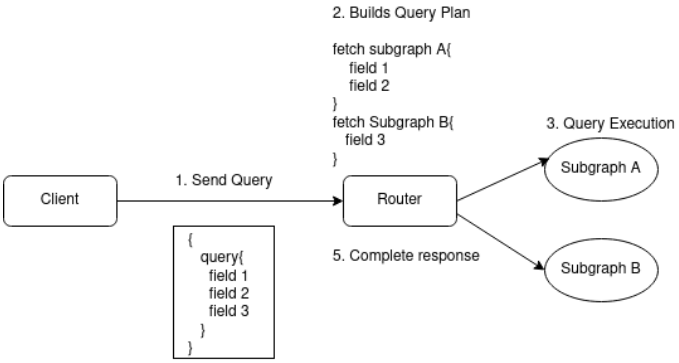
- **Supergraph Creation**

Through composition: the process of combining a set of subgraph schemas into a supergraph schema, Apollo builds the supergraph

* Note : Implementation is shown here

- **Querying**

The querying involves the composition of a query plan by resolving which subgraphs to call , executing the query plan and fetching data and finally completing the responses by combining the responses received from the subgraphs.



3. The Significance of GraphQL Federation in the API Management Gateway Layer

- **Enhanced Developer Experience** : : Integrating federation capabilities directly into the API management layer abstracts away the complexities associated with setting up and managing federated GraphQL services, they can rely on the API management platform to handle these tasks transparently, allowing them to focus on writing GraphQL APIs and implementing business logic.
- **Centralized Governance and Control**: By integrating GraphQL federation into the API manager, organizations gain centralized control over their entire API ecosystem, including GraphQL APIs alongside other types. This facilitates consistent governance practices, such as access control, rate limiting, and monitoring, across all APIs, ensuring adherence to organizational policies and regulatory requirements.
- **Efficient Resource Utilization**: Supporting federation directly within the API management layer enables more efficient resource allocation and utilization. Instead of each individual GraphQL service managing its federation logic, the API management gateway can handle federation at a higher level, reducing redundancy and optimizing performance.
- **Easier Migration and Integration**: Bringing federation capabilities to established API management platforms simplifies the process of adopting federated GraphQL APIs. Organizations can seamlessly integrate existing APIs into a federated architecture without significant changes to their infrastructure or development workflows.
- **Consistent Quality of Service (QoS)**: Providing support for federation in the API management layer ensures consistent QoS for federated APIs. Organizations can enforce policies, such as authentication, authorization, and rate limiting, uniformly across all federated services, guaranteeing a reliable and secure API experience for consumers.
- **Flexibility**: Centralizing federation capabilities within the API management layer also enhances flexibility. Organizations can easily adapt to changing requirements and evolving business needs without being constrained by the federation logic implemented in individual services.

Further bringing Federation to WSO2 APIM ensures,

- Distinguishing WSO2 API-M from competitors that lack such capabilities. This strengthens the platform's position in the market and attracts organizations seeking comprehensive API management solutions that support GraphQL federation. (Competitor analysis Document)

4. Federation Support

User Stories

- **User Story 1: API Developer**

Federate Independent GraphQL Services
As an API developer responsible for managing multiple independent GraphQL services, I want to federate these services, So that I can unify them under a single endpoint for efficiency.

- **User Story 2: API Developer**

Ensure Quality of Service for Federated GraphQL APIs
As an API developer responsible for maintaining GraphQL services, I want to ensure quality of service (QoS) for federated GraphQL APIs, So that subscribers experience reliable and efficient access to data.

- **User Story 3: Application Developer**

Subscribe to Unified GraphQL Service
As an application developer in need of data from multiple GraphQL services, I want to subscribe to a unified GraphQL service that federates independent services, So that I can access the data I need without having to manage multiple endpoints.

Stakeholders

- API publisher
- API subscriber

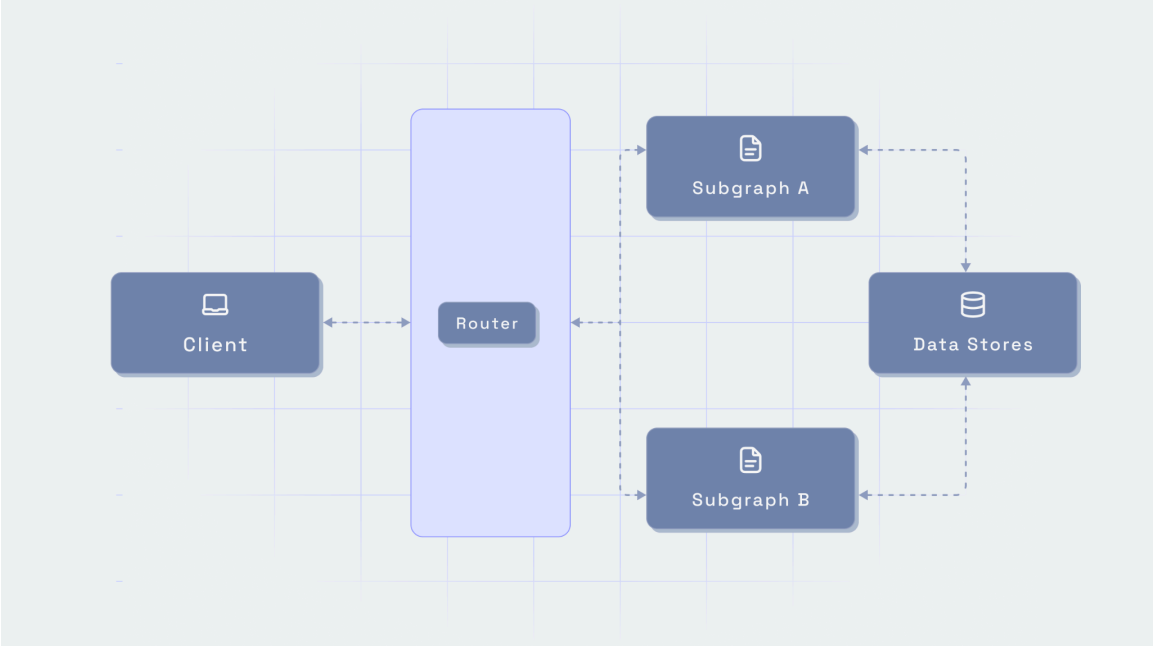
5. Method

5.1. Background

This section will describe how federation works and the details of a POC carried out to test the support of federation through a java library.

5.1.1. Federation Flow

Federation consists of the following high level architecture,



- **Subgraphs**
  - These are individual GraphQL services, each focused on a specific domain area or functionality within an organization.
  - Each subgraph defines its own GraphQL schema, tailored to the specific requirements and data model of its domain.
  - Subgraphs encapsulate domain-specific logic and data, providing clear separation of concerns and promoting modularity.

- **Gateway/ Router**
  - The Gateway serves as the single entry point for consumers of the federated GraphQL API.
  - It orchestrates requests to the various subgraphs based on the information that consumers expect, acting as a smart router for incoming queries.
  - The Gateway performs query planning to optimize network calls to the subgraphs, ensuring efficient data retrieval and minimal latency.
  - By analyzing the query structure and dependencies, the Gateway determines the most optimal execution plan, avoiding unnecessary network hops and data duplication.

From the perspective of frontend clients, the federated GraphQL API appears as a single, unified GraphQL service. Therefore multiple frontend applications or clients can interact with the federated API declaratively, fetching data across all subdomains seamlessly.And, each of these gateway or domain graph components can also be independently scaled based on demand.

- **Request Resolution**

In GraphQL federation, query planning analyzes query structure to determine required services. Sub-queries are dispatched to relevant services for independent execution. Results are aggregated for a unified response. This process optimizes performance by distributing workload and executing queries concurrently, ensuring efficient resolution of GraphQL requests.

5.1.2. Proof of Concept

A proof of concept was carried out to see the feasibility of implementing federation through Java , for which the outputs can be viewed through the link here

5.2. Implementation

This section will describe how API-M can be extended to support graphql federation.

5.2.1. How WSO2 Gateway will support GraphQL Federation

5.2.2. Handler (logic for implementation) -> mediation

5.2.3. UI changes - Publisher

5.3. Plan with Milestones

6. Appendix

```graphql
import { ApolloServer } from '@apollo/server';
import { startStandaloneServer } from '@apollo/server/standalone';
import gql from 'graphql-tag';
import { buildSubgraphSchema } from '@apollo/subgraph';

const typeDefs = gql`
  extend schema @link(url: "https://specs.apollo.dev/federation/v2.0",

  type Query {
    me: User
  }

  type User @key(fields: "id") {
    id: ID!
    username: String
  }
`;

const resolvers = {
  Query: {
    me() {
      return { id: '1', username: '@ava' };
    },
  },
  User: {
    __resolveReference(user, { fetchUserById }) {
      return fetchUserById(user.id);
    },
  },
};

const server = new ApolloServer({
  schema: buildSubgraphSchema({ typeDefs, resolvers }),
});

const { url } = await startStandaloneServer(server);
console.log(`🚀  Server ready at ${url}`);
```

```
const {serializeQueryPlan} = require('@apollo/query-planner');
const { ApolloServer, gql } = require('apollo-server');
const {ApolloGateway, IntrospectAndCompose} = require('@apollo/gateway')

const gateway = new ApolloGateway({
    experimental_didResolveQueryPlan: function(options) {
        if (options.requestContext.operationName !== 'IntrospectionQuery') {
          console.log(serializeQueryPlan(options.queryPlan));
        }
      },
    supergraphSdl: new IntrospectAndCompose({
        subgraphs: [
            { name: 'inventory', url: 'http://localhost:8084/graphql' },
            { name: 'product', url: 'http://localhost:8081/graphql' },
            { name: 'review', url: 'http://localhost:8083/graphql' },
            { name: 'user', url: 'http://localhost:8082/graphql' },
        ]
    })
});

const server = new ApolloServer({ gateway, subscriptions:false, tracing:true });
server.listen();
```