GraphQL & gRPC

GraphQL

GraphQL - язык запросов, созданный Facebook при разработке мобильного приложения. Технология была призвана решить несколько проблем, таких как:

- Отсутствие контракта при работе с сервисом
- Необходимость обработки данных на клиентской стороне
- Кроссплатформенные инструменты разработки API

GraphQL в качестве транспорта обычно использует HTTP протокол и один эндпоинт. GET-запросы используются для получения данных, а POST - для мутаций.

GraphQL предлагает для использования несколько видов операций:

- Query - запрос на получение данных
- Mutation - запрос на изменение данных
- Subscription - подписка на изменение данных

GraphQL предлагает следующие преимущества:

- Наличие языка для описания схемы данных (SDL). С помощью языка можно описать типы, операции, их параметры и исходящие данные. В итоге все совершаемые операции строго типизированы.
- Запрос возвращает только те данные, которые указал пользователь. В теле запроса можно указать те поля, которые необходимо получить на данный момент и не запрашивать остальные. Сервер вернёт только ту информацию, которая была запрошена.
- В одном запросе можно запрашивать информацию по нескольким сущностям. Например, можно запросить авторов, их книги и пользователей, которые их читают. Запрос строится в виде графа.

Сложности, с которыми можно столкнуться при разработке сервиса, использующего GraphQL:

- Сложность при разработке бэкенда. Например, проблема “query n + 1”. При запросе сущностей будут совершены дополнительные запросы на дополнительные сущности. Можно решить с помощью паттерна ленивой загрузки.
- Проблема мониторинга. Сервис отдает всегда код ответа 200 и находится за одним эндпоинтам. В итоге для реализации мониторинга нужно инструментировать код.
- Проблема безопасности. Схема может быть октрыта внешнему пользователю, соответственно есть вероятность пропустить чувствительную информацию. Так же, можно положить сервер неосторожными запросами, например зациклив запрос много раз.

Как используют GraphQL:

- Backend for frontend. Фронтэнд разработчикам нравится GraphQL, он ускоряет разработку сервисов из-за наличия контракта.
- API Composer. Объединение нескольких микросервисов в один сервис можно делать с помощью GraphQL. Таким же образом можно отдавать свой API внешним пользователям.

Можно ли использовать GraphQL для межсервисного взаимодействия? Можно, но это не принесет удобства, так как:

- При межсервисном взаимодействие выполняются конкретные запросы и важна оптимизация исполнения метода.
- При разработке межсервисного взаимодействия отсутствует преимущество от наличие в GraphQL схемы данных. А так же, возможности запрашивать произвольные данные.

Что выбрать, GraphQL или REST? При выборе обратите внимание на несколько вопросов:

- Необходимо получать множество данных с несколькими уровнями зависимостей? Или можно обойтись простыми запросами?
- Есть ограничение в количестве запросов (пропускной способности сети)?
- Критична ли ошибка в консистентности отдаваемых сервисом данных или ошибка в отдаваемых типах

данных?

Если получается несколько положительных ответов, то стоит обратить внимание в сторону GraphQL.

gRPC

gRPC - фреймворк, предназначенный для реализации удаленного взаимодействия. Его основные черты:

- Remote procedure call - вызываем методы выглядят как обычные вызовы функций и по сути напоминают их.
- Реализация транспорта отделена. Предлагается заботиться только о реализации бизнес функционала.
- В качестве транспортного протокола используется HTTP/2, а для передачи данных Protobuf.

gRPC предлагает следующие преимущества:

- Возможность описания схемы данных. Можно описать сообщения и методы, который реализует сервис.
- Использование потоков. Можно использовать поток на стороне клиента/сервера или двунаправленные потоки. Таким образом можно передавать большое количество данных между точками.
- Кодогенерация на стороне сервера и клиента. Можно сгенерировать код по описанной схеме. Инструменты предлагаются для большого количества языков.

Сложности при реализации сервиса с использованием gRPC:

- Сложный Runtime. При возникновении проблем на транспортном уровне очень сложно отследить проблему. Реализация рантайма скрыта в библиотеке и переопределить её не получится.
- Сложность балансировки. gRPC старается поддерживать одно соединение. Изза этого могут быть проблемы с использованием его в Kubernetes. Эти проблемы можно решить с помощью прокси-серверов или балансировке на стороне клиента.
- Сложность отладки. Инструменты, предлагаемые для отладки очень скудны, а обычными инструментами типа cURL пользоваться не получится.

Как используют:

- Для межсервисного взаимодействия. На основе gRPC можно организовать, как обычный API, так и более сложные вещи. Например, Envoy использует gRPC для реализации протокола, реализующего настройку прокси.
- Для использования совместно с frontend. Можно использовать gRPC для использования с frontend, но делается это через BFF или API Gateway с преобразованием протокола, например в виде grpc-web

При выборе gRPC для межсервисного обратите внимание на следующие вопросы:

- Важно ли наличие схемы сервиса и идентичности реализации серверной и клиентской части?
- Есть ограничение в пропускной способности сети между сервисом и потребителем?