API Gateway & BFF

Основные стадии развития приложения

Традиционное web-приложение - приложение содержащее в себе все функции, такие как: обработка бизнес-логики, отрисовка и отдача контента клиенту, описание модели данных. На этом этапе приложение может быть развернуто с помощью какого-либо web-сервера или application-сервера.

Возрастающая нагрузка

По мере роста нагрузки на приложение, необходимо обеспечить бесперебойность работы приложения и добавить возможность масштабирования в зависимости от роста нагрузки. Запуск web-приложения в несколько экземпляров покрывает эти требования.

На данном этапе появляется необходимость балансировки нагрузки между экземплярами приложения. Этого можно достичь с помощью Reverse proxy.

Reverse proxy - тип proxy-сервера, маршрутизирующий запросы из внешней сети к серверам внутри сети. При этом для клиента это происходит таким образом, как будто запрашиваемые ресурсы находятся на proxy-сервере.

При помощи использования reverse-proxy обычно достигались следующие цели:

- Балансировка запросов между экземплярами приложения
- Обработка ошибок. Например, можно сделать несколько повторных запросов с proxy-сервера, если один из экземпляров вернул ошибку и только потом вернуть ответ клиенту.

Возрастающая смысловая нагрузка

В приложение добавляется новая функциональность, его кодовая база растет. Отдельные части постепенно выделяются в отдельные приложения. Постепенно наступает необходимость добавления технического функционала: авторизация, метрики и прочее. Так же, с ростом сложности приложения, его интерфейс может быть выделен в отдельное приложение и получать данные от других приложений. Таким образом, мы приходим к микро-сервисной архитектуре.

При обращении к сервисам напрямую из клиента будут проблемы:

- Необходимо менять код клиента каждый раз при изменении ландшафта.
- В каждом сервисе нужно будет реализовывать логику связанную с обработкой ошибок, авторизацией, метриками и прочее.
- Некоторые клиенты чувствительны к объему загружаемых данных и к стабильности подключения. Например, для мобильного клиента может быть важно загружать только необходимый объем данных в один запрос.
- Балансировка нагрузки на разные экземпляры сервисов.

Решить эти задачи может помочь API Gateway - узел, через который потребители взаимодействуют с сервисами.

На API Gateway так же можно возложить дополнительную функциональность:

- Мониторинг - можно собирать метрики по использованию сервисов
- Логирование - логирование ошибок и прочего
- Терминирование SSL - SSL стал де-факто стандартом. Терминируется он обычно на входной точке, которой является Gateway.
- Ограничение запросов (rate limits) - Можно ограничивать количество запросов в единицу времени по какому-то признаку клиента
- Обработка ошибок: ретраи, circuit breaker - С помощью обработки ошибок можно закрывать какие-то инфраструктурные проблемы
- Трансформация протоколов - На стороне сервисап может использоваться отличный от HTTP протокол, но API Gateway может осуществлять трансляцию HTTP в протокол сервиса

- Кэширование - Можно кэшировать ответы от сервисов, например на получение даных.
- Распределение трафика - Есть возможность пустить часть трафика на новую версию сервиса
- Авторизация - Авторизацию и аутентификацию можно осуществлять на уровне API Gateway, а сервис будет получать данные о пользователе

Шаблоны применения API Gateway

API Gateway можно применять для решения определенных вопросов. Рассмотрим основные шаблоны его применения.

Backends for frontends

По мере роста разновидностей клиентов для сервисов, например, появления мобильного клиента, появления публичного API, появляется необходимость по-разному обрабатывать запросы от различных клиентов.
Эту проблему решает шаблон Backends for frontends - использование различных узлов API Gateway под разные клиенты. При использовании этого шаблона мы можем получить следующие возможности:

- Возможность изменять схему данных для различных клиентов. Каждому клиенту нужен разный набор данных
- Возможность композиции ответов от сервисов и обработки нескольких запросов одновременно
- Реализовать специальную логику обработки запросов для различных клиентов

API Composer

Иногда необходимо конкатенировать ответы от различных сервисов или исполнять несколько запросов одновременно. С этим поможет шаблон API Composer. API Gateway вызывает необходимые методы, обрабатывает ошибки по мере необходимости, записывает ответы в память и отдает конкатенированный ответ.

Каскадный API Gateway

API Gateway могут обслуживать, как внешние запросы, так и внутренние. Например, можно использовать API Gateway внутри сети для следующих нужд:

- Разделение ответственности между несколькими Gateway. Например, один обслуживает роутинг, второй авторизацию
- Распиливание монолитного приложения или сокрытие внутренней реализации