



API Gateway Pattern

 Property	
 Tags	

API Gateway

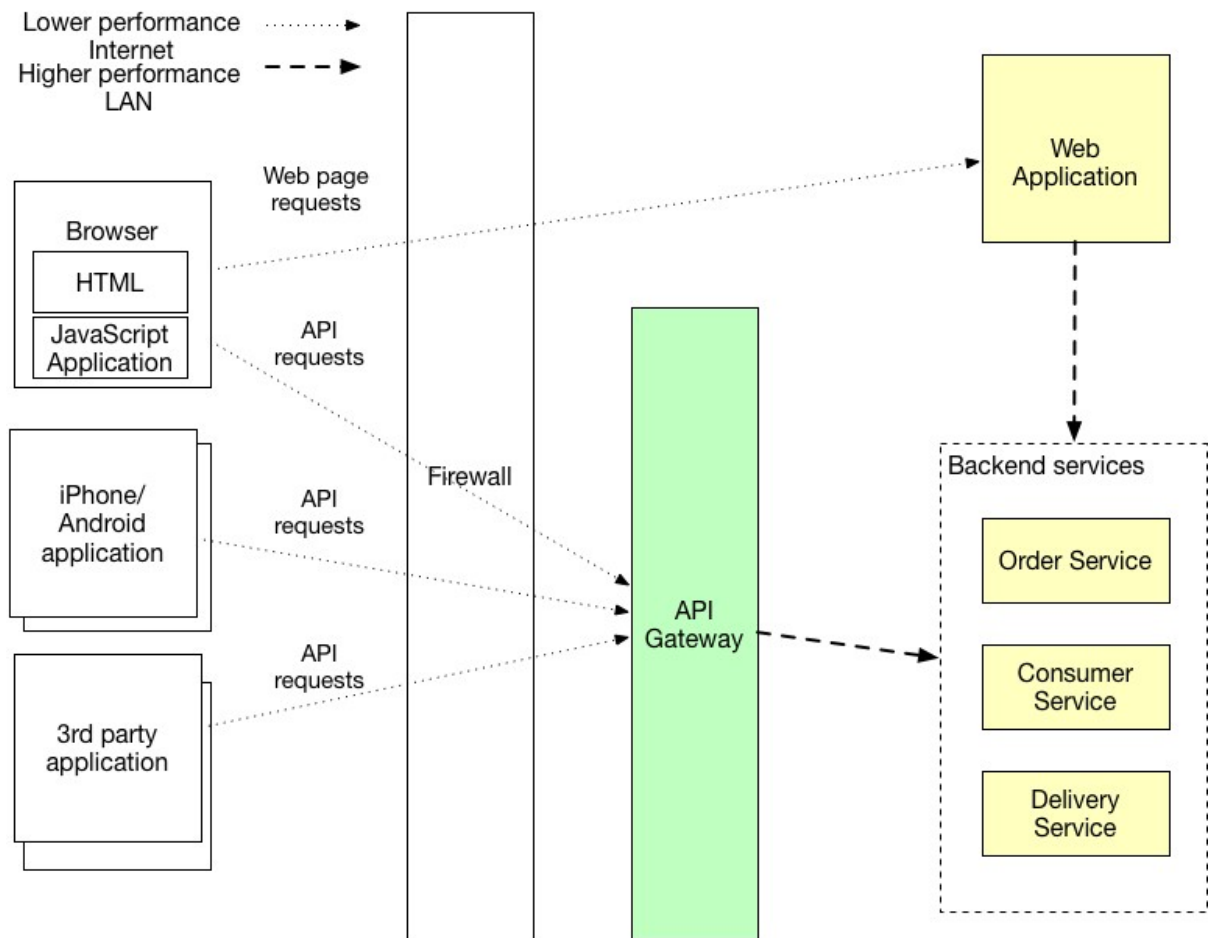
바깥 세상과 이어주는 어플리케이션의 Entry point이다.

하는 일은 다음과 같다.

- 요청에 대한 라우팅
- API 합성
- 인증
- 로드 밸런싱
- 캐싱
- circuit breaker
- protocol translation

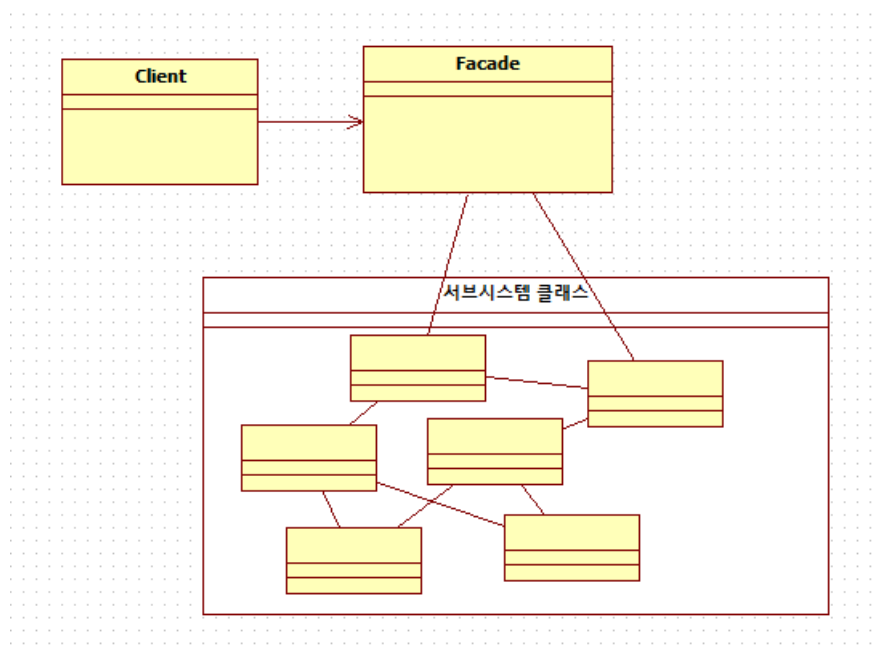
등등 다양한 기능을 수행한다.

API Gateway Pattern



유저에게 정보를 보여주기 위해 클라이언트에서 여러 번의 요청을 보내는 것은 좋지 않다. API Gateway를 사용하면 클라이언트가 한번의 요청을 보내도록 할 수 있다. API Gateway는 API 요청의 single entry point가 된다.

(cf. Object Oriented 디자인의 Facade 디자인(퍼사드 패턴)과 유사)



API Gateway는 어플리케이션의 내부 구조를 캡슐화하고 클라이언트들에게 API를 제공한다. 또한 인증, 모니터링, 속도 제한 등의 기능을 갖기도 한다.

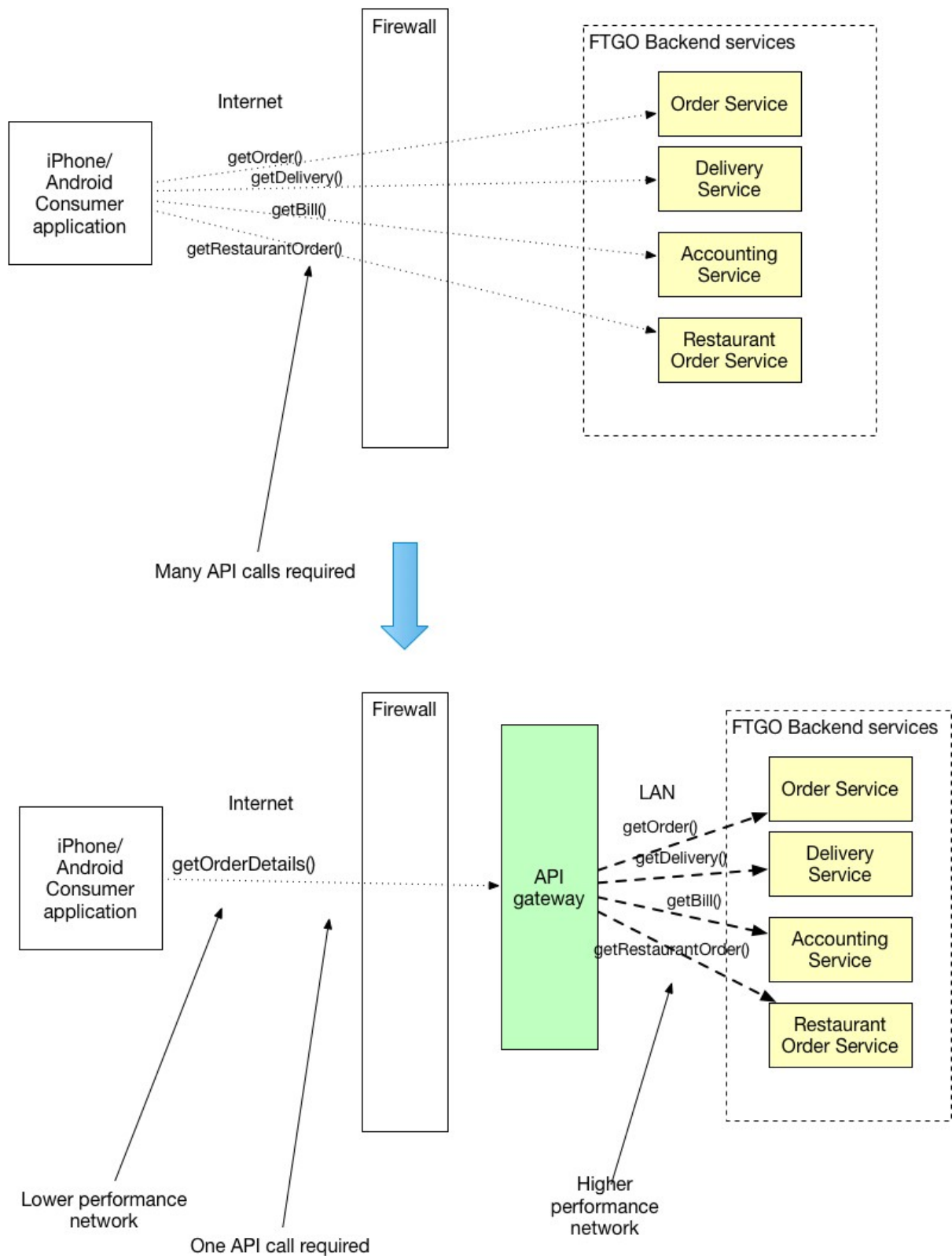
모든 클라이언트의 API요청은 항상 API Gateway를 먼저 거친다. Gateway는 API Composition Pattern을 사용하고 여러 서비스를 호출하고 결과를 모아서 요청을 처리한다. HTTP와 같은 클라이언트 친화적인 프로토콜과 클라이언트 친화적이지 않은 웹소켓같은 프로토콜 간에 통역도 처리할 수 있다.

Request Routing

API gateway는 요청을 라우팅하여 그에 맞는 서비스에 보내서 API 기능을 수행한다. 요청을 받으면 API Gateway는 라우팅 맵을 만든다. 웹서버의 reverse proxy 특징과 같다.

API Composition

API gateway는 reverse proxying 이외에도 많은 기능을 한다. 그림 1은 API Gateway를 사용하지 않은 구조이고 그림 2는 사용한 구조이다. 그림2에서는 모바일 어플리케이션이 하나의 요청을 보내서 API Gateway에서 서비스들에게 주문의 상세정보를 받아온다.



Protocol translation

API Gateway는 어플리케이션이 REST, gRPC같이 다른 프로토콜을 혼합적으로 사용하더라도 RESTful API를 외부 클라이언트에 제공할 수 있다.

The API Gateway 특정 클라이언트를 위한 API를 각 클라이언트에 제공

API Gateway는 하나의 one-size-fits-all(OSFA) API를 제공할 수 있다. 하나의 API가 가지는 문제점은 다른 클라이언트가 각각 다른 요구사항을 가질 수 있다는 것이다. API Gateway가 클라이언트마다 다른 요구사항에 맞춰 각기 다른 API를 제공할 수 있다.

Edge function

Edge function이란 어플리케이션의 끝에서 실행되는 요청 프로세싱 function을 말한다.

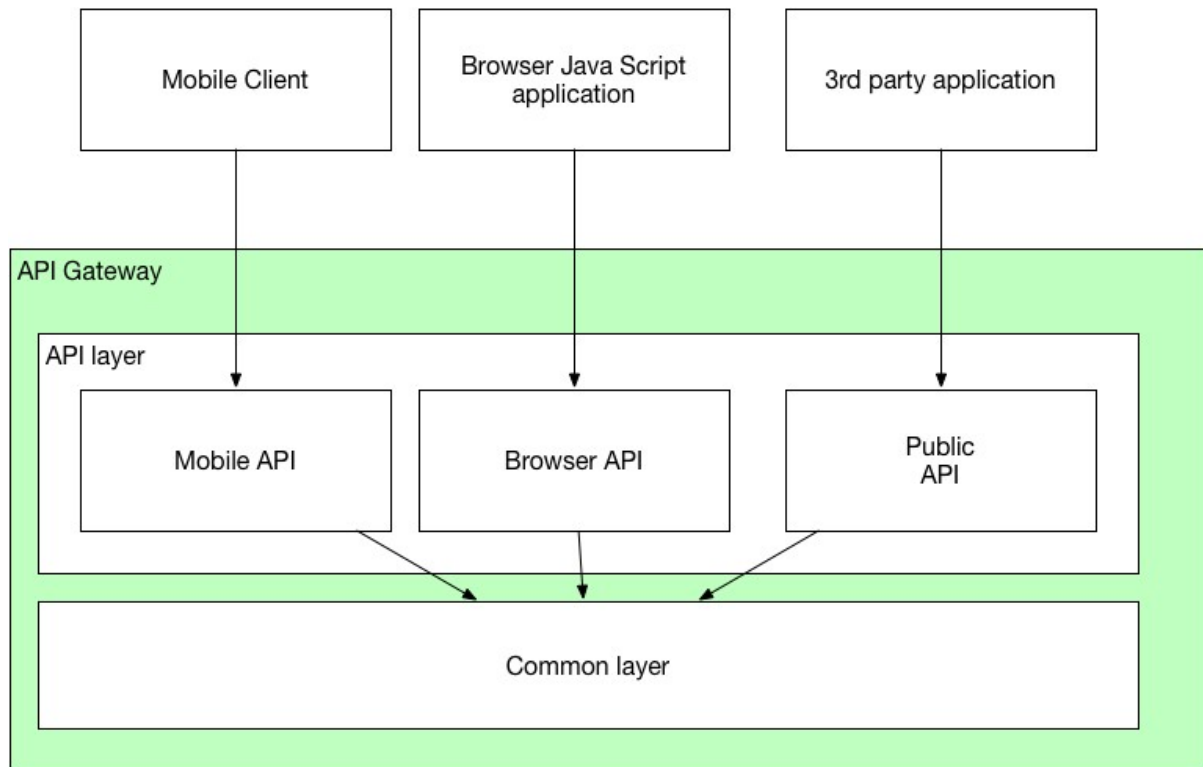
- 인증 - 클라이언트가 요청을 보낼 때 사용자인지 확인하는 것
- 권한 부여 (인가) - 특정 기능을 수행할 때, 클라이언트가 권한이 있는지 확인하는 것
- 속도 제한 - 특정 클라이언트 또는 모든 클라이언트로부터 받는 초당 요청량의 제한을 해두는 것.
- 캐싱 - 서비스에 들어오는 요청의 수를 줄이기 위해 응답을 저장하는 것.
- 측정 - 비용을 측정하기 위해 API 사용 측정
- 요청 로깅 - 요청을 기록하는 것.

여기 3가지 옵션이 존재함

1. 이 edge function을 백엔드 서비스에서 수행하게 함 : 하지만 요청이 서비스에 접근하기 전에 인증과 같은 기능을 수행하는 것이 더 안전함
2. 이 edge function을 edge 서비스에서 수행하도록함 (API Gateway의 upstream) : 외부 클라이언트들은 이 엣지 서비스에 먼저 접촉하게 되고 요청에 대한 인증과 다른 기능을 수행한 후에 API Gateway로 넘겨주는 방식. 엣지 서비스를 사용하면 기능을 나누어 API Gateway는 핵심 기능인 API routing과 composition에 집중할 수 있게 된다. 인증과 같은 중요한 책임을 하나의 엣지 서비스에 집중시킬 수 있다. 이 방법의 단점은 network latency를 증가시키고 어플리케이션의 복잡성을 키운다는 단점이 있다.
3. API Gateway에서 edge function 기능하도록 함 : network hop이 하나 줄고 latency 향상시킴

API Gateway Architecture

API Gateway는 계층적, 모듈식 구조이다. 그림 3에서 보면 API Layer와 Common Layer, 2개의 층으로 되어있다. API layer는 하나이상의 API 모듈로 되어있다. 각 API 모듈은 특정 클라이언트를 위한 API로써 동작한다. Common Layer는 edge function들을 포함한 공유된 기능을 한다.



위의 그림을 보면 API Gateway는 3개의 API 모듈로 되어있는 것을 알 수 있다. API 모듈은 각 API 동작을 하나 또는 두가지 방식으로 동작한다. API 동작 중 일부는 직접 하나의 서비스 API 동작으로 매핑된다. API 모듈은 이 동작을 상응하는 서비스 API 동작으로 바로 요청을 라우팅하여 동작한다. 라우팅 룰이 기술된 configuration 파일을 읽는 라우팅 모듈을 사용하여 구현한다.

API Gateway ownership model

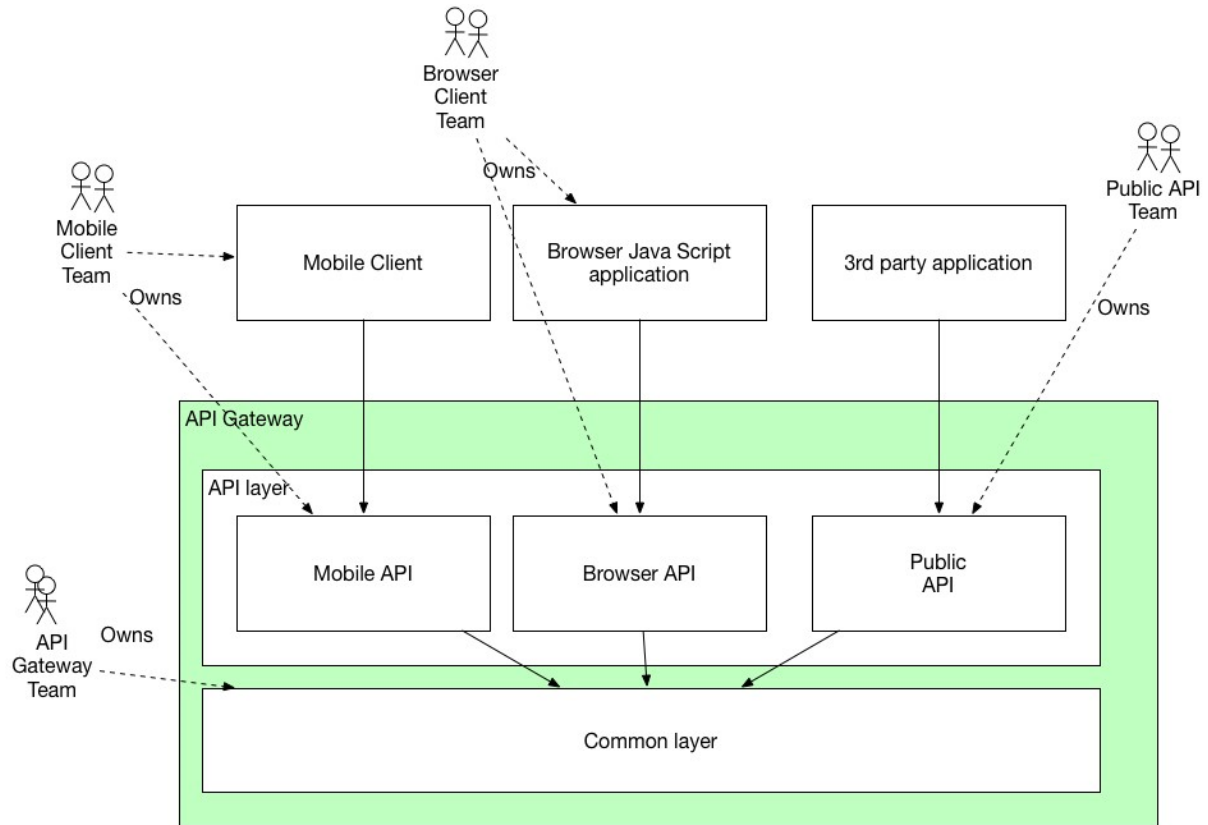
누가 API를 관리하는가?

하나의 옵션은 API Gateway를 관리하는 별도의 팀을 두는 것이다. 하지만 이 방식은 SOA와 비슷한 단점을 가진다. (ESB팀이 모든 ESB 개발에 대한 책임이 있다.)

모바일 어플리케이션을 담당하는 개발자가 다른 서비스에 접근할 필요가 있으면 API gateway 팀에 요청을 해야하고 API를 줄 때까지 기다려야한다. 이것은 조직에게 있어서 병목이 된다. 마이크로서비스의 철학과는 맞지 않다.

더 나은 접근을 위해 Netflix에서와 같이 client team (모바일팀, 웹팀, public API 팀) 으로 나눈다. API gateway 팀은 공통 모듈에 대한 책임을 진다. 각 팀에서는 각 팀에서 필요한 API를 관리한다.

팀이 그들의 API를 변경해야하면 API Gateway의 소스 레포지토리를 변경하면 된다. 대신 API gateway의 배포 파이프라인은 모두 자동화가 되어있어야 한다. 그렇지 않으면 클라이언트 팀은 API gateway 팀이 또 새로운 버전을 배포할때 기다려야하기 때문이다.

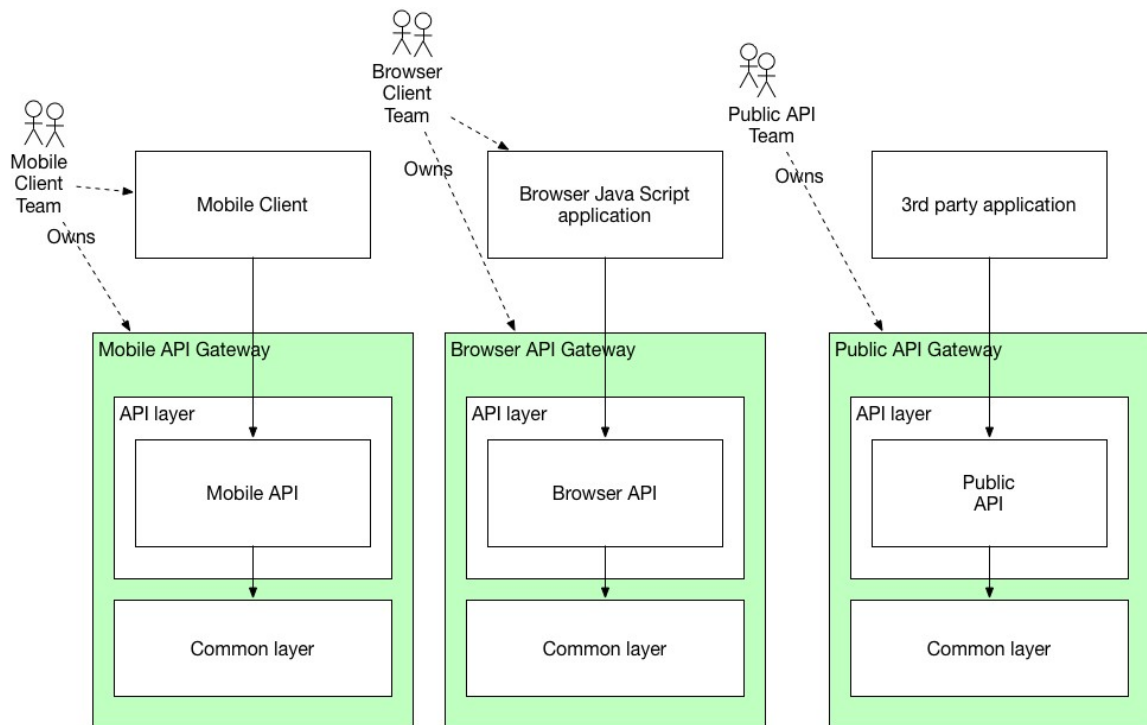


Backends for frontends 패턴 사용하기

API Gateway의 역할이 흐려질 수 있다. 여러 팀이 같은 코드베이스에서 일하게 된다. API gateway 팀은 API gateway의 동작에 책임을 가진다. SOA의 ESB보다 나쁘지는 않지만 마이크로서비스의 철학에 위배된다. 이에 대한 솔루션으로 각각의 클라이언트에 API gateway를 가지게 하는 방법이 있다. 바로 BFF 패턴이다. 그림을 보면 하나의 클라이언트 팀에 각각의 Gateway가 존재함을 확인할 수 있다.

이론적으로 각각의 Gateway에서는 다른 기술스택을 사용할 수 있지만 같은 기술 스택으로 사용할 것이 권장된다. API Gateway팀이 만든 공유된 라이브러리를 사용하여 같은 동작을 수행할 수 있도록 한다.

BFF패턴은 책임을 명확하게 구분하는 것과 더불어 신뢰성을 증가시킨다. 하나의 잘못된 API 동작이 다른 API들에 영향을 주지 않는다. 또한 각각의 API 모듈들마다 다른 프로세스를 가지므로 Observability(관측가능성)을 증가시킨다. 그리고 각 API가 독립적으로 확장가능성이 있다는 장점이 있다. 초기 시작 시간을 줄일 수도 있는데, 각 API Gateway는 더 작고 간단한 어플리케이션이기 때문이다.



API Gateway의 장단점

장점

- 어플리케이션의 내부 구조를 캡슐화
- 특정 클라이언트를 위한 API를 제공하여 클라이언트와 어플리케이션 간의 통신 수를 줄일 수 있으며 클라이언트 코드 양도 줄일 수 있다.

단점

- 개발진행에 있어서 개발하고 배포하고 관리해야하므로 병목이 될 수 있다.
- 그래서 API Gateway를 가능한 가볍게 만드는 것이 중요하다. 그렇지 않으면 다른 개발자들이 Gateway의 업데이트를 기다려야하기 때문이다.

이러한 단점에도 불구하고 실제 어플리케이션에서 API Gateway를 많이 사용한다.