API GateWay란?

MSA(Microservice Architecture) API Gateway란?

MSA(Microservice Architecture) API Gateway는 서버 최앞단에 위치하여 모든 API 호출을 받습니다. 받은 API 호출을 인증한 후, 적절한 서비스들에 메세지를 전달될 수 있도록 합니다.

API Gateway는 MSA에서 언급되는 컴포넌트 중 하나이며, 모든 클라이언트 요청에 대한 엔드 포인트(End Point)를 통합하는 서버입니다. 마치 프록시 서버(Proxy Server)처럼 동작합니다. 그리고 인증 및 권한, 모니터링, 로깅(Logging) 등 추가적인 기능이 있습니다.

모든 비지니스 로직이 하나의 서버에 존재하는 모놀리식 아키텍처(Monolithic Architecture)와 달리 MSA는 도메인별 데이터를 저장하고 도메인별로 하나 이상의 서버가 따로 존재합니다. 한 서비스에 한개 이상의 서버가 존재하기 때문에 이 서비스를 사용하는 클라이언트 입장에서는 다수의 엔드 포인트가 생기게 되며, 엔드 포인트 변경이 일어났을때 관리가 힘듭니다. 따라서 MSA 환경에서 서비스를 하나로 통합할 수 있는 API Gateway가 필요한 것입니다.

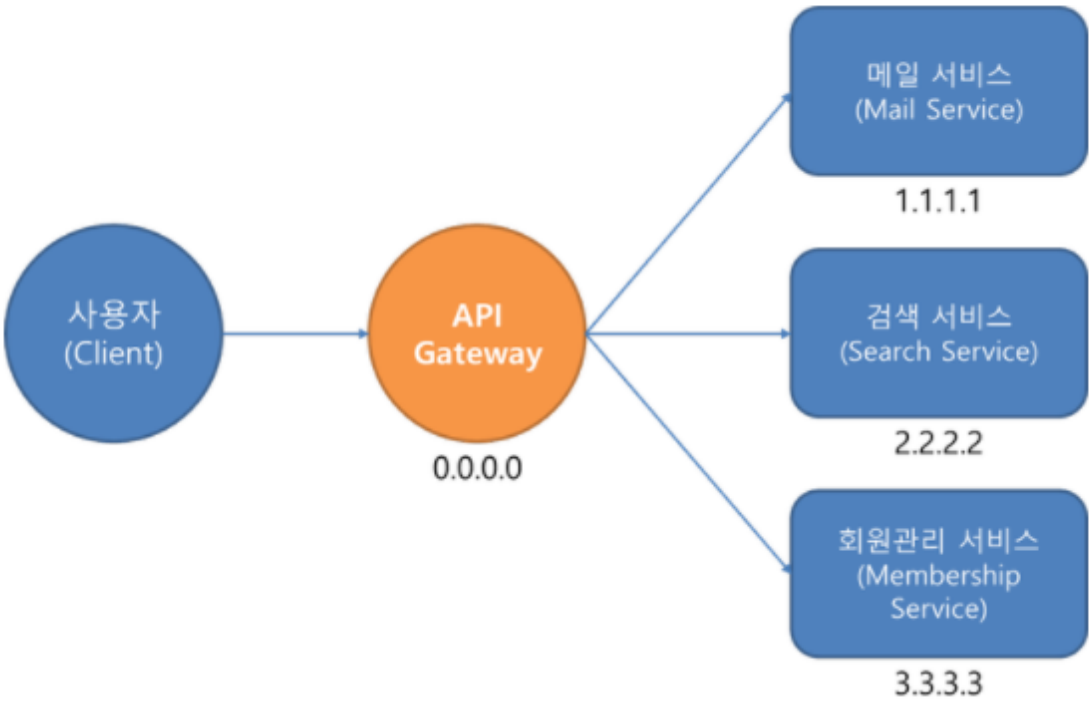
※ 마이크로서비스 아키텍처(Microservice Architecture) 관련 내용은 아래 글을 참고하시길 바랍니다.

여기서 이해가 필요한 부분은 그림 도식화에서도 보았듯이, 도메인별 API server가 다다른데, 그것을 모놀리식 아키텍처(Mononlithic Architecture) 가 아니라 MSA로 통합적으로 관리하는 것임.

MSA는 큰 서비스를 잘게 쪼개어 개발/운영 하는 아키텍처입니다. 하나의 큰 서비스는 수많은 작은 서비스로 나뉘어지며, 만약 이를 클라이언트에서 서비스를 직접 호출하는 형태라면 다음과 같은 문제점이 생길 수 있습니다.

▶ 각각의 서비스마다 인증/인가 등 공통된 로직을 구현해야하는 번거로움이 있습니다.
▶ 수많은 API 호출을 기록하고 관리하기가 어렵습니다.
▶ 클라이언트에서 여러 마이크로서비스에 대한 번거로운 호출을 해야합니다.
▶ 내부의 비즈니스 로직이 드러나게 되어 보안에 취약해집니다.

특히 이러한 문제점들은 마이크로서비스의 갯수가 많아질 수록 기하급수적으로 늘어나게 될 것입니다. 어느 규모 이상의 마이크로서비스 기반 어플리케이션에는 API Gateway를 도입하는 것이 효율적입니다.



위와 같은 문제점이 발생할 수 있는거지, 특히 각각 서브시마다 인증/인가를 공통된 로직으로 구현해야하는 번거러움이 존재하는거.

> API Gateway의 시작은 SOA의 핵심 인프라라고 할 수 있는 ESB(Enterprise Service Bus)에서 시작되었습니다. 따라서 API Gateway의 많은 부분들이 ESB로 부터 승계되었습니다. ESB가 SOAP/XML 기반의 무거운 기능을 가졌다면, API Gateway는 REST/JSON 기반으로 보다 가볍게 설계된 것이 특징입니다.

이게 진짜 핵심 부분이다. 잘 읽어봐야 하는 부분

MSA API Gateway의 주요 기능

1. 인증 및 인가 (Authentication and Authorization)

마이크로서비스 아키텍처에서 각각의 서비스에 API 호출에 대한 인증 및 인가를 하는 것은, 같은 소스코드를 서비스 인스턴스들마다 심어주어야한다는 것을 의미합니다. 이러한 경우, 소스의 중복이 심하여 유지 관리가 어려운 것은 물론, 로깅 및 모니터링을 관리하는 것도 매우 어려워집니다. 이러한 이유로 인증서 관리나, 인증, SSL, 프로토콜 변환과 같은 기능들은 API Gateway에서 오프로드 함으로, 각각의 서비스의 부담을 줄이고, 서비스의 관리 및 업그레이드를 보다 쉽게 할 수 있게 합니다.

2. 요청 절차의 단순화

여러 마이크로서비스를 대상으로하는 기능을 이용하려 할 때, 만약 API Gateway가 없다면 클라이언트에서 여러 서비스들에 대해 요청을 진행해야 했을 것입니다. 하지만, API Gateway는 여러 클라이언트의 요청을 단일 클라이언트의 요청으로 대체 가능하도록 합니다. 따라서 클라이언트와 백엔드 간의 API 통신량을 줄여주어 대기시간을 줄이고 효율성을 높여줄 수 있습니다.

3. 라우팅 및 로드밸런싱

API Gateway는 클라이언트로 부터 접수된 메세지에 따라, API 호출을 적절한 서비스에 라우팅 할 수 있는 기능이 있습니다. 또한, 서비스 인스턴스들에 대한 부하분산을 할 수 있습니다.

4. 서비스 오케스트레이션

오케스크레이션은 여러개의 마이크로서비스를 묶어 새로운 서비스를 만드는 개념입니다. 오케스트레이션 로직을 과도하게 넣는 것은, API Gateway의 부담을 늘리는 것으로, 성능 저하를 일으킬 수 있어, MSA와 API Gateway에 대한 높은 수준의 기술적 이해를 바탕으로 이루어져야 합니다.

5. 서비스 디스커버리

API Gateway는 각 서비스를 호출하기위해, 서비스마다의 IP주소와 포트번호를 알고 있어야 합니다. legacy 환경에서는 고정적인 IP주소를 가지고 있기 때문에 크게 문제될 것이 없지만, 클라우드 환경에서는 동적인 환경에서 배포되기 때문에 서비스의 위치를 찾는 것이 어렵습니다. 이러한 서비스의 위치(IP 주소와 포트번호)를 찾는 것을 "Service Discovery"라 하며, API Gateway에서는 서버사이드나, 클라이언트 사이드를 기준으로 하여 서비스 디스커버리를 구현할 수 있습니다.

> API Gateway 오픈소스

1. Kong (https://konghq.com)
git : https://github.com/kong
Nginx + Cassandra + Lua Script
MIT 라이센스
Docker 지원
Plugin 형태로 다양한 기능 지원

2. API Umbrella (https://apiumbrella.io/)
git : https://github.com/NREL/api-umbrella
NGINX + node.js + varnish + MongoDB + Redis + Elastic Search
MIT 라이센스
RoR로 구현한 포털 제공

3. tyk.io(https://tyk.io)
git : https://github.com/TykTechnologies/tyk
MongoDB + Redis + Nginx + golang
MPL 2.0 라이센스
API Documentation - Swagger 지원
Docker 지원

4. Zuul(https://github.com/Netflix/zuul)
git : https://github.com/Netflix/zuul
Spring Nexflix에 속함
아파치 2.0 라이센스
Java

정리하자면,

마이크로 서비스(mircro service)는 하나의 큰 애플리케이션을 여러 개의 다른 역할을 수행하는 애플리케이션을 분리하였을때 각 애플리케이션을 의미한다.

그럼 결국 Framework는 MSA로 잘 되어 있는 것이라고 할 수 있고,

API gateway가 없는 경우 클라이언트가 모든 마이크로 서비스의 호스트명은 물론 end-point를 알고 있어야 한다. 또한 요청의 횟수가 증가할수록 응답속도가 늦어지게 되는 것이다.

그리고 end-point를 client에서 구현하려면 코드가 너무 길어지고, 서버의 호스트 명시를 일일이 해야한다. 일부 마이크로 서비스가 웹 통신에 적합한 프로토콜로 통신을 안하기에 API gateway가 중요한 것이다.