API gateway 자료조사

목차

1. API gateway

APIg gateway의 필요성

마이크로서비스 아키텍처는 큰 서비스를 잘게 쪼개어 개발 및 운영하는 아키텍처이다. 하나의 큰 서비스는 수많은 작은 서비스로 나뉘어지며, 만약 이를 클라이언트에서 서비스를 직접 호출하는 형태에는 여러 문제들이 따르게 된다.

- 각각의 서비스마다 인증/인가 등 공통된 로직을 구현하는 번거로움이 있다.
- 수많은 API 호출을 기록, 관리가 어렵다.
- 클라이언트에서 여러 마이크로서비스에 대한 호출로 번거롭다.
- 내부의 비즈니스 로직이 드러나게 되어 보안에 취약점이 발생한다.

이러한 문제들로부터 어플리케이션을 효율적으로 관리하기 위해서 API-Gateway가 필요하다.

API gateway의 기능

API-Gateway는 API 서버 앞단에서 모든 API 서버들의 엔드포인트를 단일화해주는 또 다른 서버이며, 아래의 그림 1은 API-Gateway의 기능들을 보여주는 그림으로 가장 기본적인 기능인 API 라우팅 기능을 시작으로 클라이언트에서 호출하는 요청과 API 서버가 제공하는 API의 스펙의 차이가 발생하는 경우에 중재를 해주는 메디에이션 기능과 같은 다양한 기능들을 제공한다.

그림 1 API-Gateway의 기능

2. API gateway 관련 오픈소스

1. Kong Gateway

Kong Gateway (OSS)는 범용 배포를 위해 구축된 인기있는 오픈 소스의 고급 클라우드 네이티브 API 게이트웨이다. 모든 플랫폼에서 실행할 수 있으며, Lua 프로그래밍 언어로 작성되었으며 하이브리드 및 다중 클라우드 인프라를 지원하며 마이크로 서비스 및 분산 아키텍처에 최적화되었다.

그 핵심에서 Kong은 고성능, 확장 성 및 이식성을 위해 구축되었고, ""Kong은 또한 가볍고 빠르며 확장 가능하다. 데이터베이스 없이 메모리 내 저장소 만 사용하는 선언적 구성과 네이티브 Kubernative CRD를 지원한다.

2. Tyk

Tyk은 Go 프로그래밍 언어를 사용하여 처음부터 작성된 오픈 소스의 강력하고 가볍고 완전한 기능을 갖춘 API 게이트웨이로 개방형 표준을 기반으로 쉽게 확장 및 플러그 가능한 아키텍처를 갖춘 클라우드 네이티브이며 고성능이다.

독립적으로 실행할 수 있으며 데이터 저장소로 Redis가 필요하고 이를 통해 사용자는 레거시, REST 및 GraphQL을 비롯한 다양한 서비스를 안전하게 게시하고 관리할 수 있다.

다양한 인증 방법, 할당량 및 속도 제한, 버전 제어, 알림 및 이벤트, 모니터링 및 분석을 포함하는 많은 기능으로 가지고 있다. 또한 서비스 검색, 즉석 변환 및 가상 엔드 포인트를 지원하며 출시 전에 모의 API를 생성 가능하다.

위의 내용 외에도 API 문서를 지원하고 API 개발자 포털, CMS (콘텐츠 관리 시스템)와 같은 시스템을 제공하며 이 시스템에서 관리 API를 게시하고 타사 개발자가 등록하고 API에 등록하고 관리 가능하다.

중요한 것은 Tyk API Gateway 버전이 하나뿐이며 100 % 오픈 소스라는 것입니다. Community Edition 사용자이든 엔터프라이즈 사용자이든 동일한 API 게이트웨이를 사용할 수 있다. 기능 잠금 및 블랙 박스없이 완전한 사용성을 위해 필요한 모든 부분이 제공된다. Tyk을 사용하면 데이터가 어떻게 처리되고 있는지 정확히 알 수 있습니다."

3. Goku API gateway

Goku API Gateway는 Go를 사용하여 구축된 클라우드 네이티브 아키텍처가 있는 오픈 소스 마이크로 서비스 게이트웨이다. 마이크로 서비스 아키텍처의 API 게이트웨이로 작동합니다. 통합 인증, 흐름 제어, 보안 보호를위한 플랫폼 내부 OPEN API 개발 플랫폼으로 타사 API를위한 통합 플랫폼이다.

고성능 HTTP 전달 및 동적 라우팅, 서비스 오케스트레이션, 다중 테넌시(소프트웨어 인스턴스) 관리, API 액세스 제어 등을 제공한다. 클러스터 배포 및 동적 서비스 등록, 백엔드 부하 분산, API 상태 확인, API 연결 해제 및 재 연결 기능, 핫 업데이트 (노드를 다시 시작하지 않고 구성을 지속적으로 업데이트)를 지원한다.

Goku는 또한 구성을 쉽게 해주는 내장 대시 보드, 기능을 확장하는 강력한 플러그인 시스템, 명령 줄을 통해 Goku를 시작/중지/리로드 하는 CLI를 제공한다.

3. Kong Gateway

개요

Kong Gateway는 OSS(Open Source Software)와 Enterprise 버전으로 나누어진다.

- OSS는 마이크로서비스에 최적화된 경량 API gateway로 기본 사항만이 적용된 버전이다.
- Enterpreise는 기업, 회사용으로 더 많은 기능으로 보안, 협업, 대규모 성능 및 고급 프로토콜 사용을 위한 플러그인을 사용하며 고급 기능을 제공하는 버전이다.

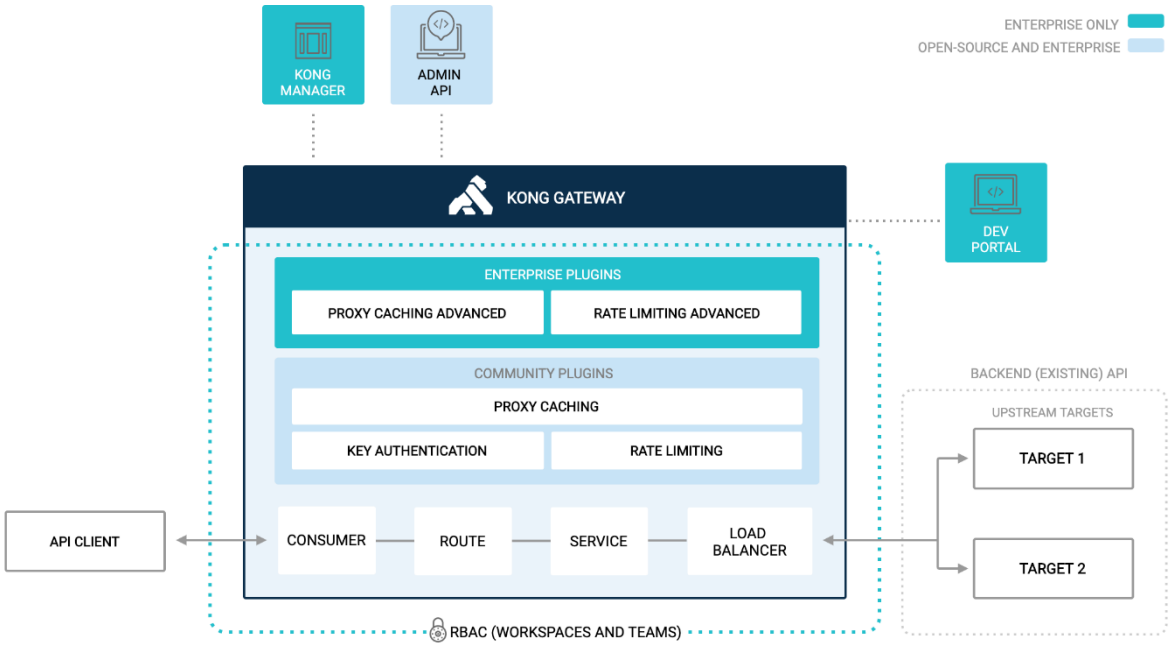

그림 2 Kong Gateway 제공 서비스 영역

그림 2는 버전에 따른 제공되는 서비스를 색으로 구분하며, OSS의 경우에는 KONG MANAGER와 DEV PORTAL을 사용할 수 없다. 아래 표 1은 앞에 나온 그림에서 OSS버전에서 제공되어지는 기능들과 용어들에 대한 설명이 적혀 있는 표이다.

표 1 용어 개념 정리

용어	설명
Service	서비스 개체는 마이크로서비스를 참조하는 ID이다.
Routes	API 게이트웨이에 도달한 요청이 서비스로 전송되는 방법(경로)
Consumers	API의 최종 사용자를 의미하며, 소비자 개체는 API에 엑세스권한 설정을 할 수 있으며, 트래픽을 보고할 수 있다. (Kong Vitals 사용)
Admin API	Kong Gateway는 관리를 위해 내부 RESTful API를 제공한다.
Plugins	모듈식 시스템을 제공하는 기능으로 액세스 제어, 캐싱, 속도 제한, 로깅 등을 포함한 다양한 기능을 사용할 수 있다.
Rate Limiting plugin	클라이언트가 주어진 시간 내에 만들 수 있는 HTTP 요청 수를 제한할 수 있다.
Proxy Caching plugin	역방향 프록시 캐시 구현을 제공하며, 주어진 기간 동안 응답 코드, 콘텐츠 유형, 요청 방법을 기반으로 응답 엔터티를 캐시한다.
Key Authentication plugin	서비스 경로 또는 경로에 키 인증을 추가할 수 있다(API key)
Load Balancing	Kong Gateway는 DNS 기반과 링 밸런서를 사용하는 두 가지 방법을 제공한다.
User Authorization (RBAC)	RBAC 기반으로 사용자 권한 부여를 처리한다.

Traffic flow 이해

Kong Gateway는 기본적으로 구성된 프록시 포트 8000 및 8443에서 트래픽을 수신한다.

클라이언트 API요청을 평가하고 적절한 백엔드 API로 라우팅한다. 그림 3은 위에서 설명한 트래픽에 대해서 보여주며, 추가로 Kong Gateway는 인증을 처리하기 때문에 서비스 자체에 인증 로직이 필요 없으며, 모든 요청이 기록된다.

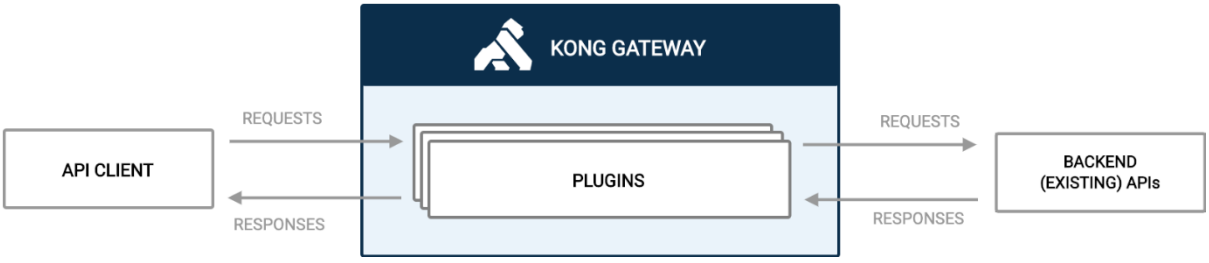

그림 3 Traffic flow

4. 마이크로서비스 모니터링

서비스 메쉬

애플리케이션의 다양한 부분들이 서로 데이터를 공유하는 방식을 제어하는 방법이다.

서비스 간 커뮤니케이션을 관리하는 다른 시스템들과 달리, 서비스 메쉬는 애플리케이션에 구축된 전용 인프라 계층이며, 가시적인 인프라 계층은 서로 다른 애플리케이션 부분이 얼마나 원활하게 상호작용하는지를 기록할 수 있으므로, 커뮤니케이션을 최적화하고 애플리케이션 확장에 따른 다운 타임을 방지할 수 있다.

동작

서비스 메쉬는 애플리케이션 런타임 환경에 새로운 기능을 도입하지 않고 아키텍처 내의 애플리케이션에는 요청이 A지점에서 B지점으로 전달되는 방식을 지정하는 룰이 항상 필요하다. 서비스 메쉬의 차이점은 개별 서비스로부터 서비스 간 커뮤니케이션을 통제하는 로직을 통해 인프라 계층에 추상화한다는 점이다.

이를 위해 서비스 메쉬는 네트워크 프록시의 배열로서 애플리케이션에 구축된다.

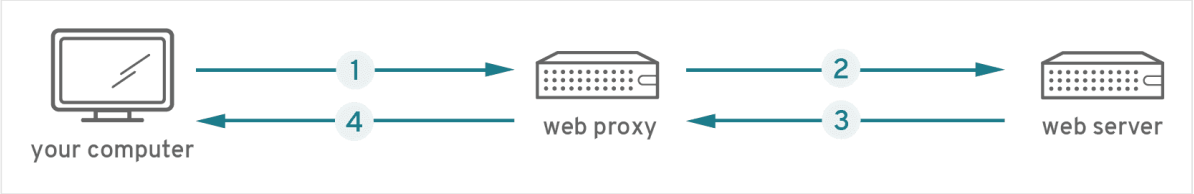

그림 4 프록시 동작

서비스 메쉬에서는 요청이 자체 인프라 계층의 프록시를 통해 마이크로서비스 간에 라우팅되고 이러한 이유로 서비스 메쉬를 구성하는 개별 프록시는 서비스 내부가 아니라 각 서비스와 함께 실행되므로 'sidecar'라고 명칭된다. 각 서비스에서 분리된 이러한 sidecar 프록시들이 모여 그림 5와 같은 메쉬 네트워크가 형성된다.

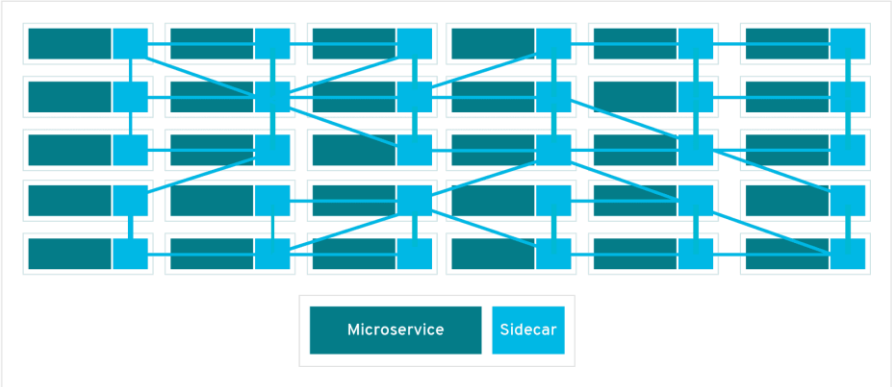

그림 5 메쉬 네트워크

이스티오(Istio)

서비스 메쉬의 기본 아키텍처는 서비스 간 직접 호출 대신 앞에서 언급한 경량화된 사이드카 패턴의 프록시를 배치하여 통신한다. 이런 서비스 메쉬 기술의 대표적인 구현체가 이스티오이며, Google, IBM, Lyft 등이 참여하는 오픈소스 프로젝트로 2018년 일반에 공개된 이스티오는 Spring Cloud Netflix와 많은 유사점이 있지만 Java에 국한된 Spring Cloud Netflix와 달리 플랫폼 영역에서 동작하기 때문에 개발 언어와 무관하게 사용할 수 있다.

이스티오는 쿠버네티스와 클라우드 기반에 적합하여 이미 Red Hat OpenShift, VMware Tanzu 등 PaaS 플랫폼의 서비스 메쉬로 선택되기도 하였습니다.

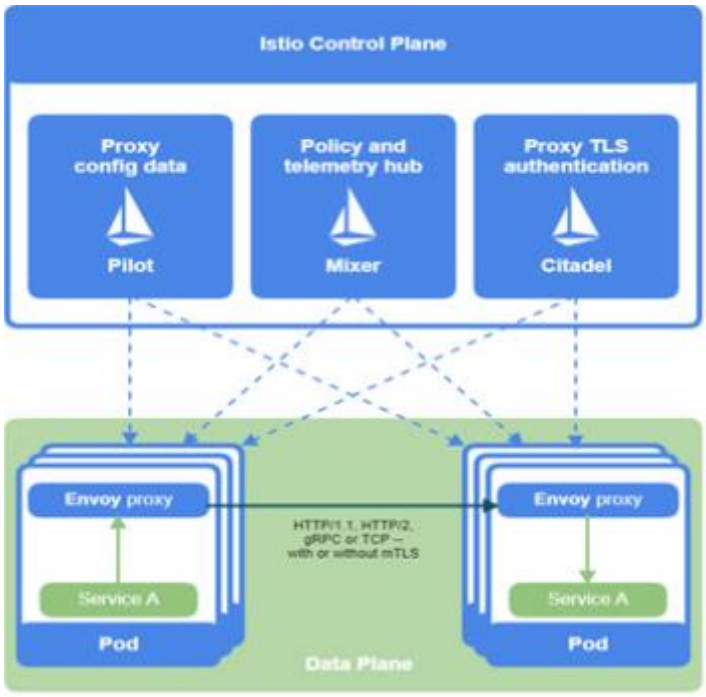

그림 6 이스티오 구조

부하분산과 트래픽 관리

이스티오는 파일럿과 사이드카 프록시인 Envoy를 통해 트래픽을 제어하고 관리한다

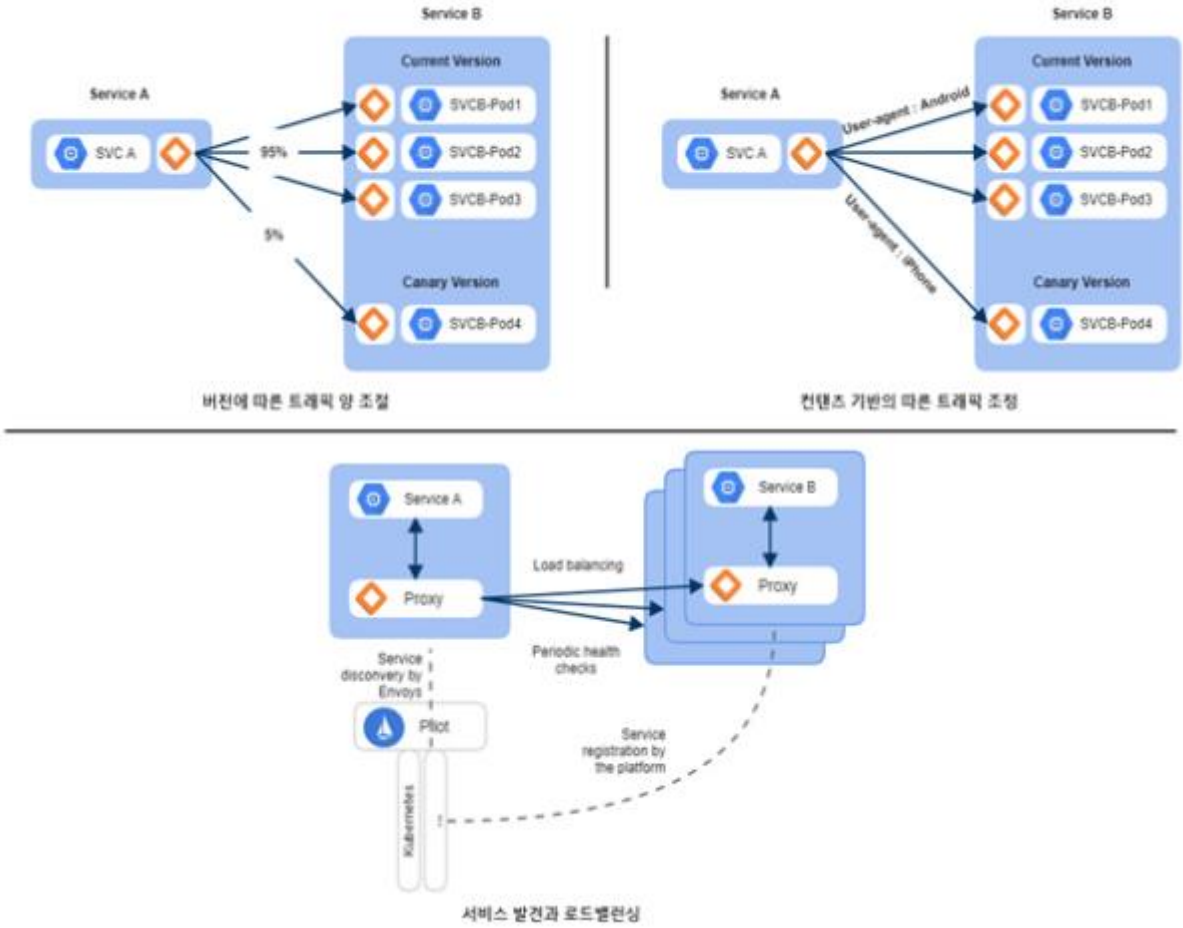

그림 7 이스티오의 트래픽 관리

그림 7은 이스티오의 트래픽 관리에 대한 그림으로 믹서는 파일럿을 이용하여 서비스 버전별 트래픽 양을 분산하거나 요청 콘텐츠에 따라 특정 버전별로 트래픽을 분할하여 전송할 수 있으며, 파일럿을 통해 서비스를 디스커버리하고 로드밸런싱한다. 서비스 상태를 주기적으로 체크하여 비정상적인 인스턴스는 자동으로 제거가 가능하다. 서비스 간 호출 안전성을 위해 호출 재시도 횟수를 통제하거나 응답 시간에 따른 에러 처리 등 통신 상태도 관리할 수 있다.

모니터링

이스티오는 믹서를 통해 서비스 간 호출 관계, 서비스 응답시간, 처리량 등의 네트워크 트래픽 지표를 수집하고 모니터링한다. 모든 서비스는 호출될 때 앞단의 Envoy를 거치기 때문에 각 서비스의 호출 시 또는 주기적으로 믹서에 정보를 전달하고 이를 로깅할 수 있다. 믹서는 플러그인(Plugin)이 가능한 어댑터(Adapter) 구조로 여러 모니터링 시스템에 연결된다. 예를 들면 쿠버네티스에서 많이 사용하는 Heapster, Prometheus, StackDriver, Datadog 등의 모니터링 도구들과 연계하여 시각화할 수 있으며 Grafana를 통해 각 서비스의 응답시간, 처리량 등을 확인할 수 있다. 또한 Jaeger를 이용하면 분산 트랜잭션 구간별로 응답 시간을 모니터링할 수도 있다.

문제점

- **복잡성**: 서비스 메쉬를 사용하면 런타임 인스턴스 수가 증가한다.
- **사이드카 컨테이너 수 증가**: 각 서비스는 서비스 메쉬의 사이드카 프록시를 통해 호출되므로 개별 프록시 수가 증가하게 된다.

서비스 메쉬와 API gateway

API Gateway와 서비스 메쉬가 하는 일은 라우팅, 인증, 모니터링, 서비스 검색, 서비스 등록 등으로 동일하지만 외부에 노출되는 것과 작동 위치에서 차이점이 있다. API Gateway 적용 위치는 Client-to-Server이고 외부 노출이 되지만 Service Mesh 적용 위치는 Server-to-Server로 외부노출은 없다.

5. 결론

API gateway와 서비스 메쉬를 같이 사용이 가능한 통합을 고려해볼 필요가 있으며, 개발에 앞서 서비스 메쉬는 서비스가 컨테이너 환경을 기반에 최적화된 구조로 보이며, 서버리스 구조와 함께 사용되기 위해서는 AWS Fargate(Fat Lambda)와 같은 하이브리드 도구에 대한 추가적인 공부가 필요하며 API gateway와 서비스 메쉬 그리고 서버리스와 컨테이너가 서로 보완하는 방식으로 사용할 수 있는지에 대한 추가적인 조사가 필요할 것으로 보인다.