마이크로서비스 아키텍처

Monolithic

모놀리식 아키텍처는 하나의 코드베이스 안에 모든 기능을 통합하는 방식

1. 도메인과 도메인의 경계가 모호할 때
2. 제공된 기능이 긴밀하게 결합됐으며, 모듈간 상호작용에서 유연함보다 성능이 절대적으로 더 중요할 때
3. 관련된 모든 기능의 애플리케이션 확장 요구사항이 알려져 있고 일관적일 때
4. 기능이 변동성이 없을 때, 즉 변화가 느리거나 변화 범위가 제한적일 때

마이크로 서비스 아키텍처

각 기능을 독립적인 서비스로 분할하여 디커플링하는 방식

서비스

데이터부터 비즈니스 로직까지 독립적으로 컴포넌트간의 의존성 없이 개발된 컴포넌트
REST API 와 같은 표준 인터페이스로 기능을 외부로 제공한다.

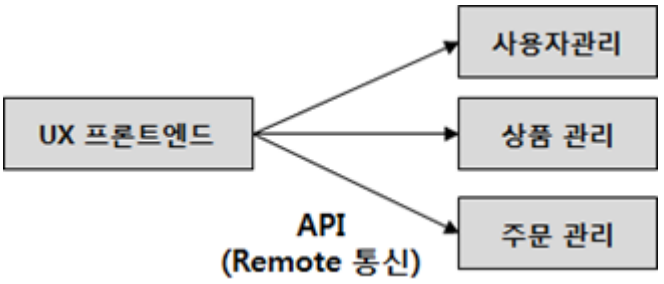
서비스 경계는 구문 또는 도메인의 경계를 따른다.

예를 들어 사용자 관리/상품 관리/주문 관리와 같은 각 업무 별로 서비스를 나눠서 정의한다.

사용자+상품 관리 처럼 여러개의 업무를 동시에 하나의 서비스로 섞어서 정의하지 않는다.

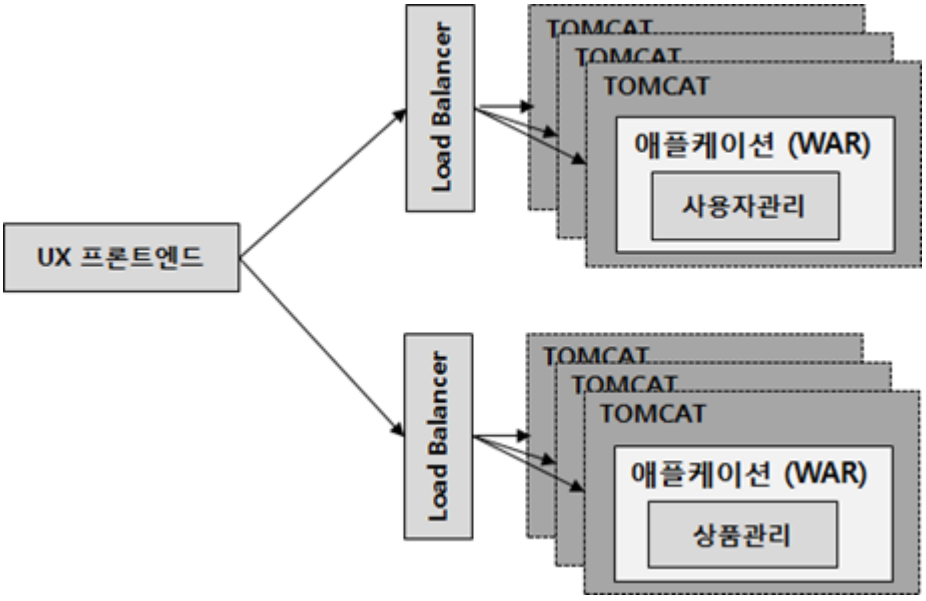
REST API에서 /users, /products와 같이 주요 uri도 하나의 서비스 정의의 범위로 좋은 예가 된다.

1. MSA의 구조



각 컴포넌트는 서비스라는 형태로 구현되고 API 를 이용해 타 서비스와 통신을 한다.

배포 구조관점에서도 각 서비스는 독립된 서버로 타 컴포넌트와 의존성 없이 독립적으로 배포가 된다.



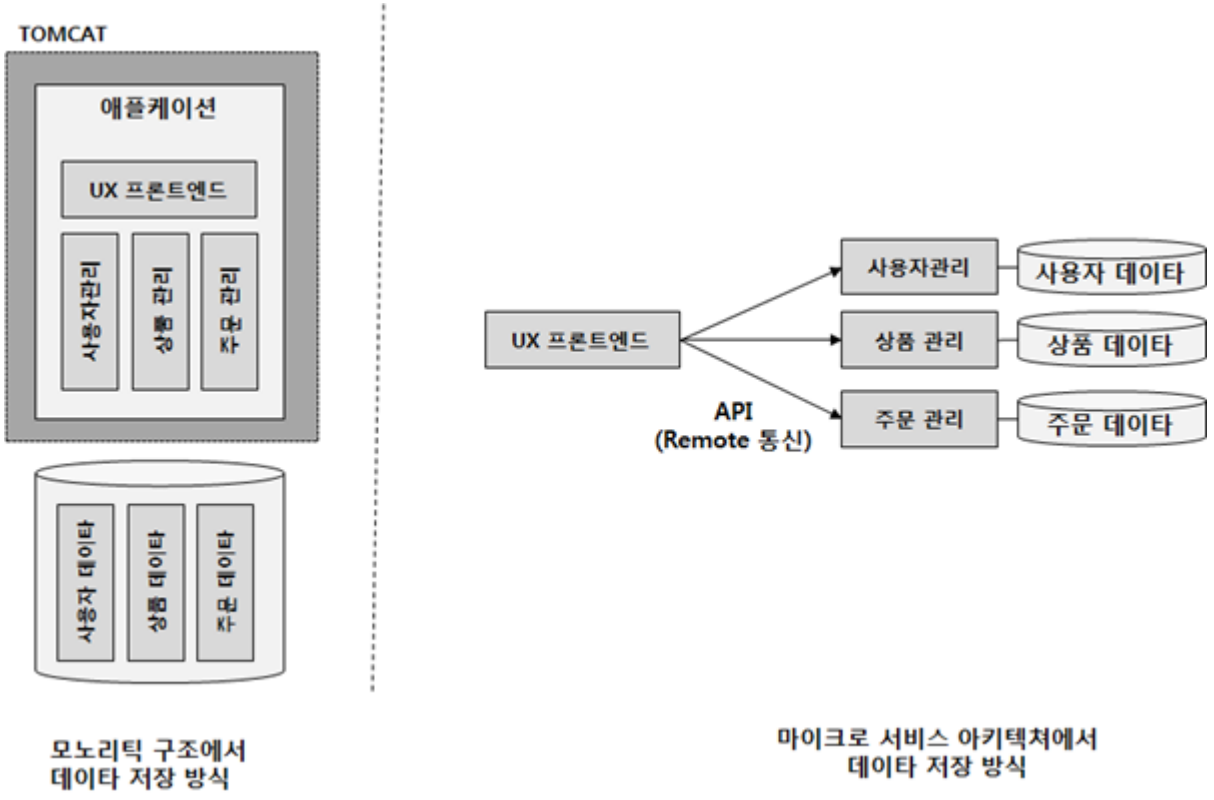
서버 확장을 위해서 서비스가 배치된 톰캣 인스턴스는 횡적으로 스케일이 가능하고, 로드 밸런서를 배치해 서비스간의 로드를 분산시킬 수 있다.

각 서비스가 다른 서비스와 물리적으로 분리되기 때문에 부분 배포가 가능하다.

2. 데이터 분리

데이터 저장관점에서 서비스별로 별도의 데이터베이스를 사용한다.

보통 MONO 서비스의 경우 하나의 통 데이터베이스를 사용하는 경우가 일반적이지만, 마이크로서비스 아키텍처의 경우, 서비스가 API에서 DB까지 분리되는 수직 분할 원칙에 따라서 독립된 데이터베이스를 가지게 된다.



모노리틱 구조에서
데이터 저장 방식

마이크로 서비스 아키텍쳐에서
데이타 저장 방식

데이터베이스의 종류 자체를 다르게 사용하거나, 같은 DB를 사용하더라도 DB를 나누는 방법을 사용한다.

이 경우, 다른 서비스 컴포넌트에 대한 의존성 없이 서비스를 독립적으로 개발 및 배포/운영 할수 있다는 장점을 가지고 있으나,

다른 컴포넌트의 데이터를 API 통신을 통해서만 가지고 와야하기 때문에 성능상 문제를 야기할 수 있고,

또한 이 기종 데이터베이스간 트렌젝션을 묶을 수 없는 문제점을 가지고 있다.

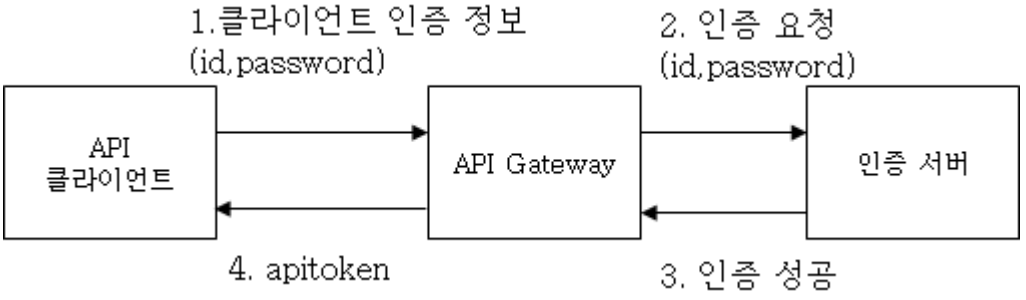
3. API gateway

마치 프록시 서버처럼 API들 앞에서 모든 API에 대한 END-POINT를 통합하고, 몇가지 추가적인 기능을 제공하는 미들웨어.

API 서버 앞단에서 모든 API 서버들의 엔드포인트를 단일화하며 묶어주고, API 인증/인가, 여러 서버로 라우팅하는 기능까지 담당할 수 있다.

3-1. 인증/인가 서비스

- 인증: API를 호출하는 클라이언트에 대한 신분을 확인
- 인가: 클라이언트가 특정 API를 호출할 수 있는 권한이 있는지 확인



일반적인 토큰 발급 절차

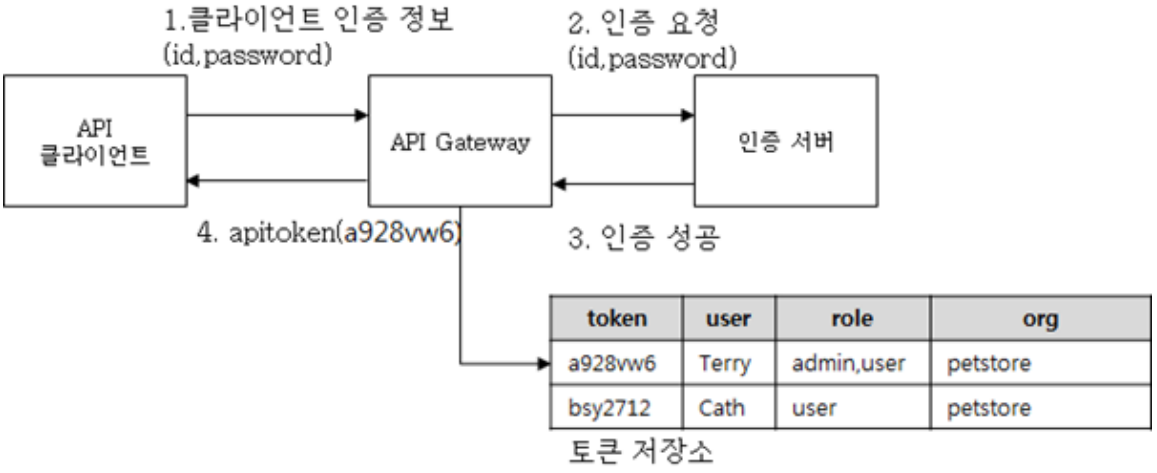
클라이언트에 대한 인증은 API 게이트웨이에서 직접 하지 않고 뒷단에 있는 인증 서버가 이를 수행한다. → Active Directoryh, LDAP 또는 RDBMS등이 될수 있으며, 외부 인증 서버로는 페이스북 구글 등에 요청을 보내 API TOKEN을 발급해준다.

이렇게 발급한 토큰을 API를 호출할 수 있는 권한 정보와 연관이 되는데, 이 권한 정보를 토큰 자체에 저장하느냐 또는 서버에 저장해놓느냐에 따라서 두 가지 종류로 나눌수 있다.

```
{
“name”:”Terry”,
“role”:[“admmin”,”enduser”]
“org”:”petstore”
}
```

Claim based token(jwt)

토큰 자체가 권한 정보를 갖는 형태를 Claim based token이라고 하는데, JWT 토큰이나 SAML 토큰 등이 이에 해당한다.



서버에 토큰을 저장하는 경우

클레임 기반 토큰이 아닌 경우, 서버에 클레임 정보를 저장하게 되는데, 클라이언트로는 unique한 string만을 리턴해 주는 경우이다.

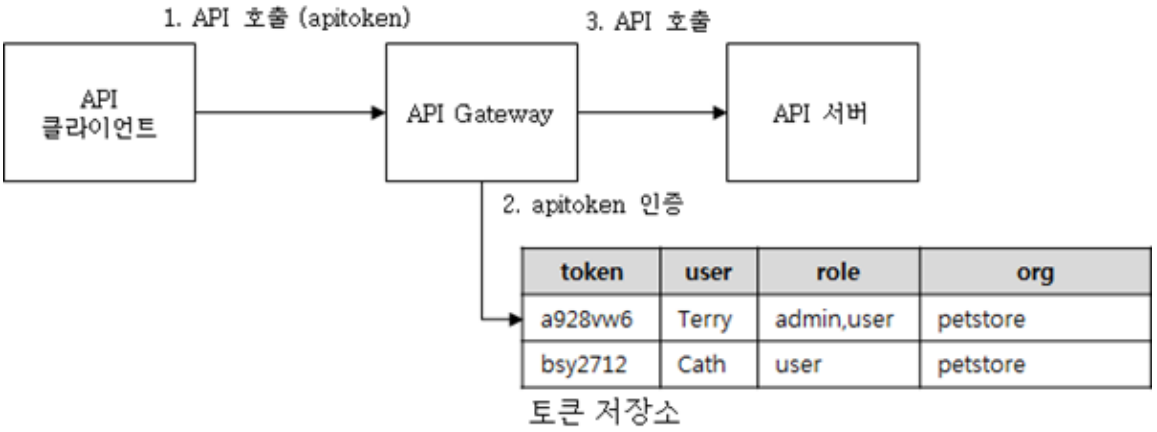
token에 연관되는 정보가 서버에 저장되기 때문에 안전하고, 많은 정보를 저장할 수 있으며, token에 대한 정보를 수정하기가 용이하다.

하지만 서버단에서 별도의 토큰 저장소를 유지해야 하기 때문에 구현 노력이 더 높게 든다. 토큰은 매 API 호출마다 정보를 가지고 와야 하기 때문에, DBMS와 FILE IO 기반의 저장소 보다는 redis, memcached같이 메모리 기반의 고속 스토리지를 사용하는 것이 좋다.

클레임 기반의 토큰의 경우 토큰 저장소가 없기에 구현은 쉽지만, 토큰 자체에 클레임 정보가 들어가 있기 때문에, 토큰의 길이가 커지기 때문에 일정 양 이상의 정보를 담기가

어려우며, 한번 발급된 토큰은 권한 변경이 어렵다.

→ 유효기간을 둬서 반드시 강제적으로 토큰을 재발급 받도록 해야한다.



apitoken을 이용한 api 호출 인증

서버에 토큰 정보가 저장되는 형태의 경우 매 api 호출마다 해당 apitoken을 가지고 연관 정보를 토큰 저장소로부터 읽어와 비교 후 , 그 정보를 기반을 api 호출 가능 여부를 결정한다.

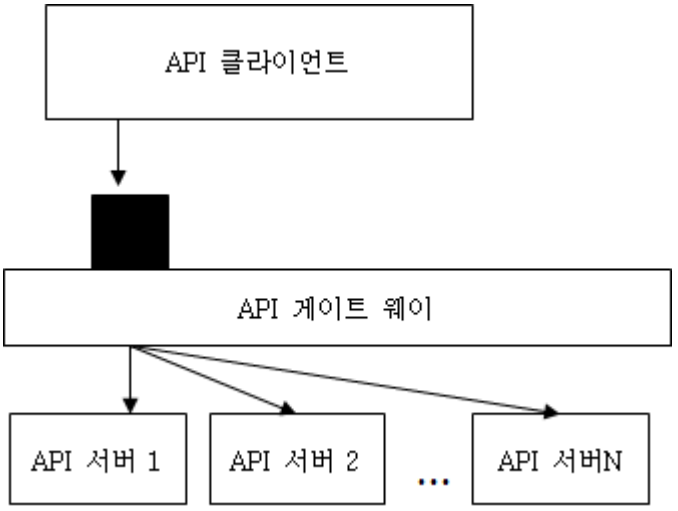
클레임 기반의 토큰의 경우에는 이러한 작업없이 api gateway에서 확인 후, 그 안에 있는 내용을 가지고 api 호출 가능 여부를 결정한다.

→ api token으로 인증을 하는 방법이 일반적이지만, 클라이언트나 서비스 별로 제공되는 엔드포인트에 대해 api 인증 방식이 다르기 때문에, api 게이트웨이에서는 각 엔드포인트별로 다양한 형태의 인증 방식을 제공해야 한다

3-2. API 라우팅

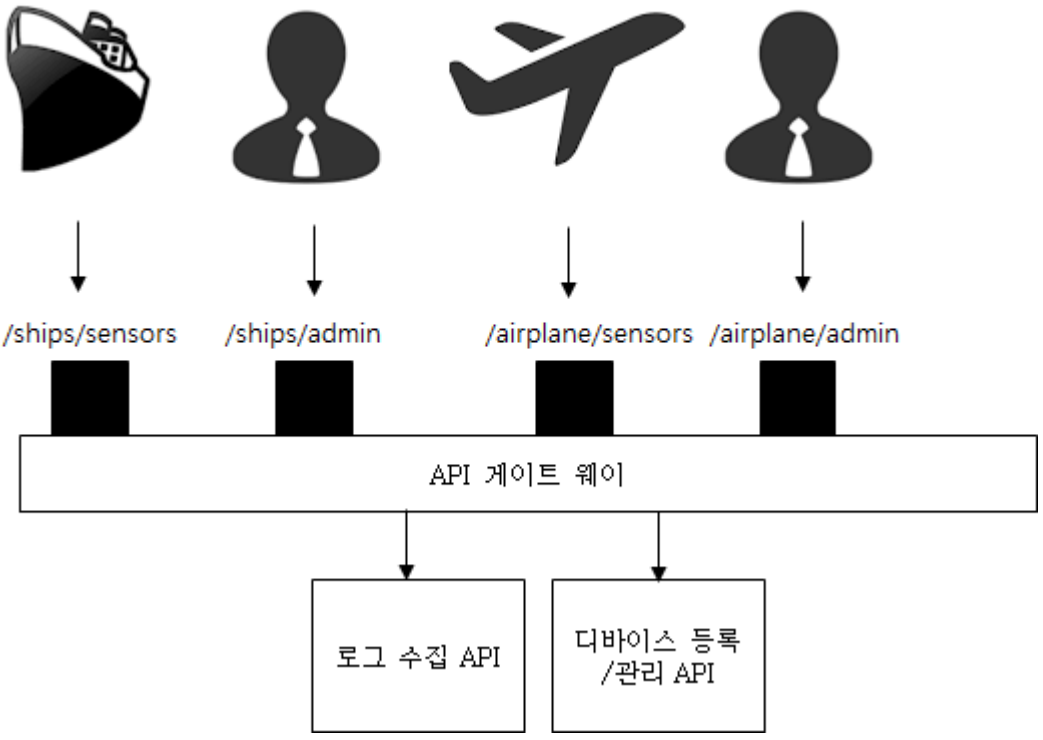
같은 api라도 사용하는 서비스나 클라이언트에 따라 다른 엔드포인트를 이용해 서비스를 제공하거나, 데이터 센터가 여러개일때, 센터간 라우팅을 지원하는 기능.

1. 백엔드 api 서버로의 로드 밸런싱



api gateway를 통한 api 서버로의 로드 밸런싱

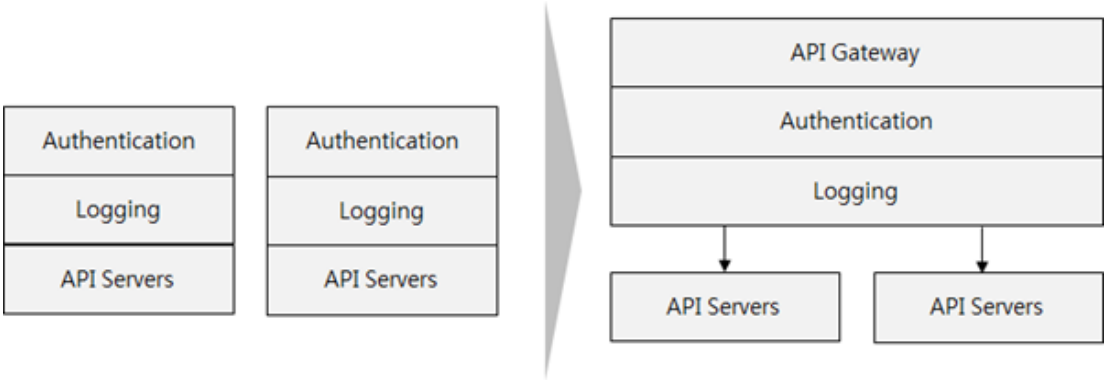
2. 서비스 및 클라이언트 별 엔드포인트 제공



같은 API를 여러개의 엔드포인트를 통해 서비스를 제공할수 있다.

3. 공통 로직 처리

API Gateway는 특성상 모든 api 서버 앞쪽에 위치하기 때문에, 모든 api 호출이 거쳐간다. 그렇기 때문에, 모든 api가 공통적으로 처리해야 하는 공통 기능이 필요한 경우 이러한 공통 기능을 api 게이트웨이로 옮기게 되면 비즈니스 로직 자체 구현에만 집중할수 있게 된다.

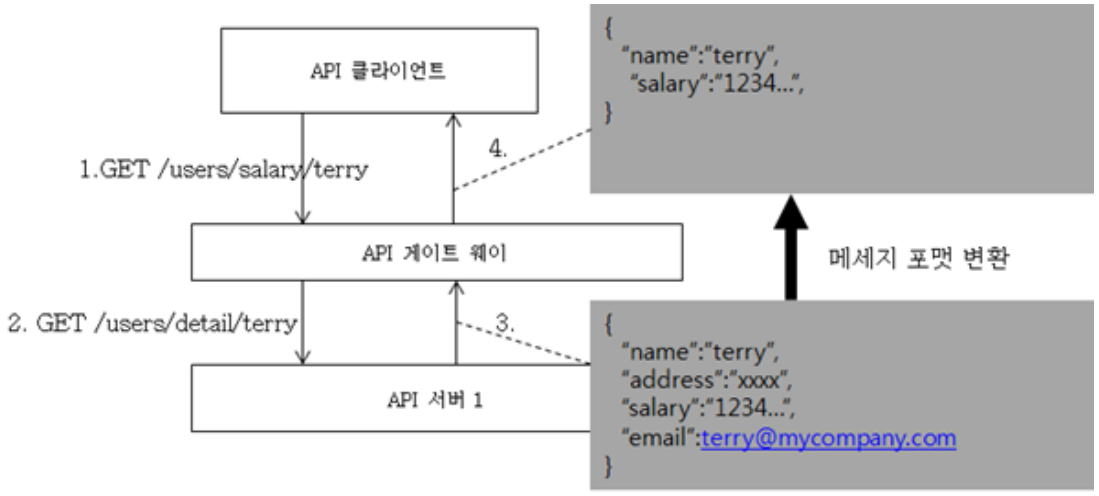


3-3. 중재/조정 기능(Mediation)

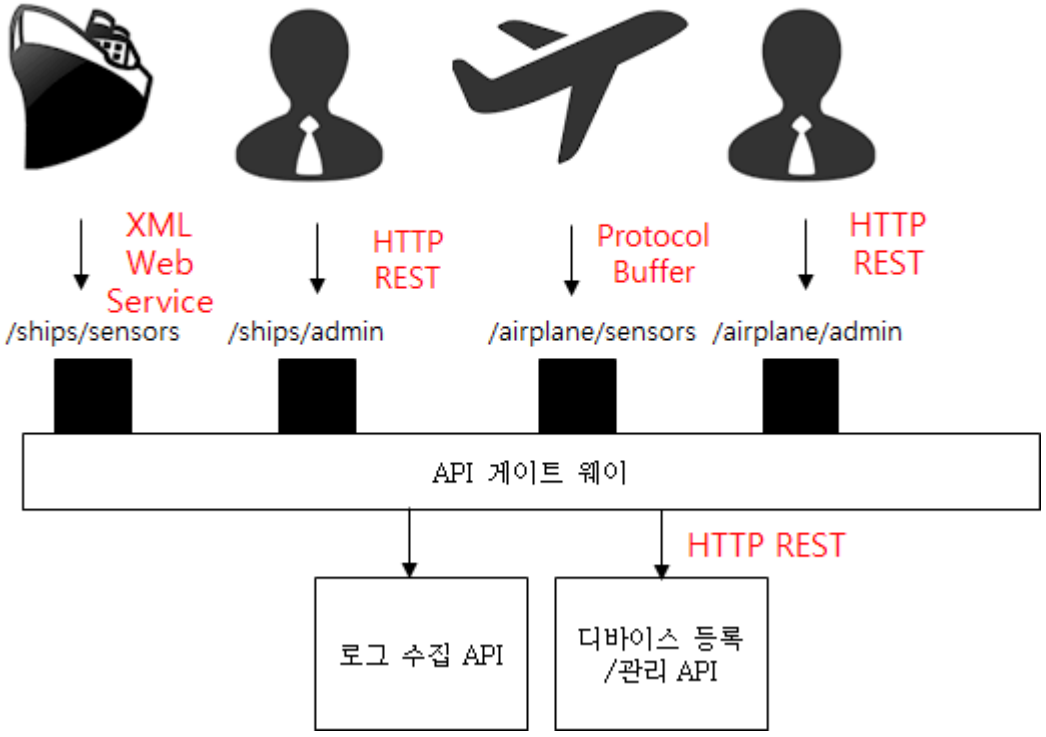
API 서버에서 제공되는 API 가 클라이언트가 원하는 API 형태와 다르때 , API GATEWAYT 가 이를 변경해주는 기능

1. 메시지 포맷 (Message format transformation)

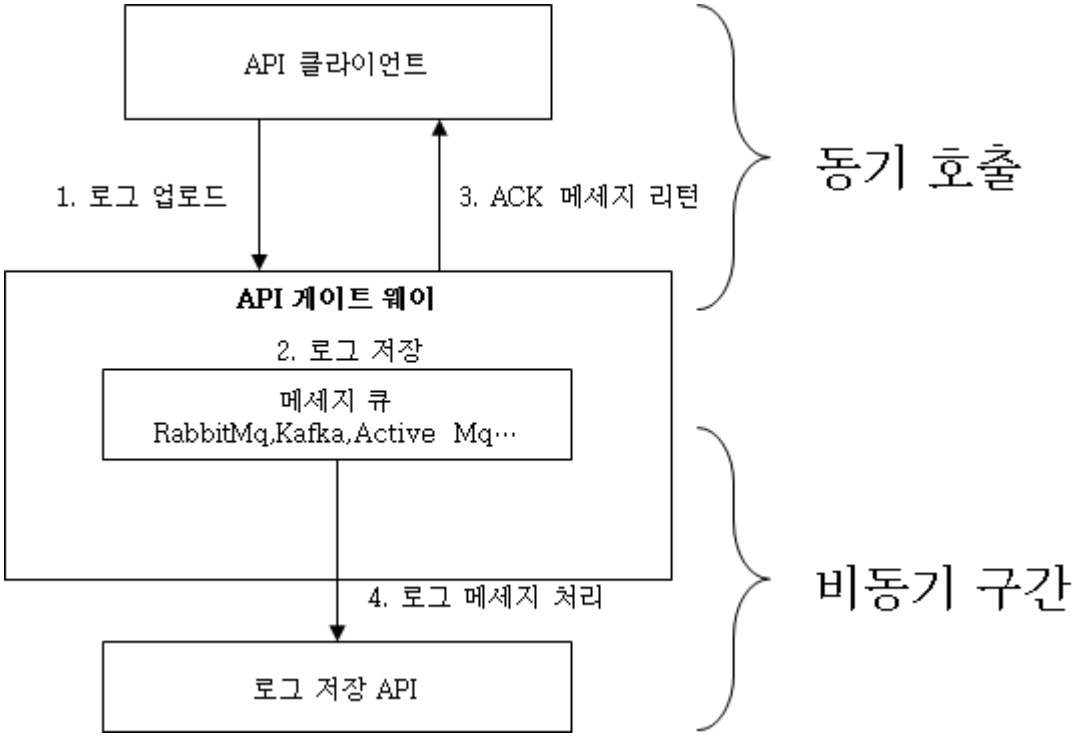
   JSON으로 된 요청 메시지가 왔을때, API 서버로 보낼때 변환해서 보내거나, API 서버에서 생성된 응답을 클라이언트에 리턴할때 변경해서 보내는 기능을 의미한다.



2. 프로토콜 변환



3. 메시지 호출 변환(Message Exchange Pattern)



로그저장 API 서버가 대용량 트래픽에 대한 대응 능력이 없다고 가정할 때,

API GATEWAY에서 큐를 이용해 API 요청을 받고,

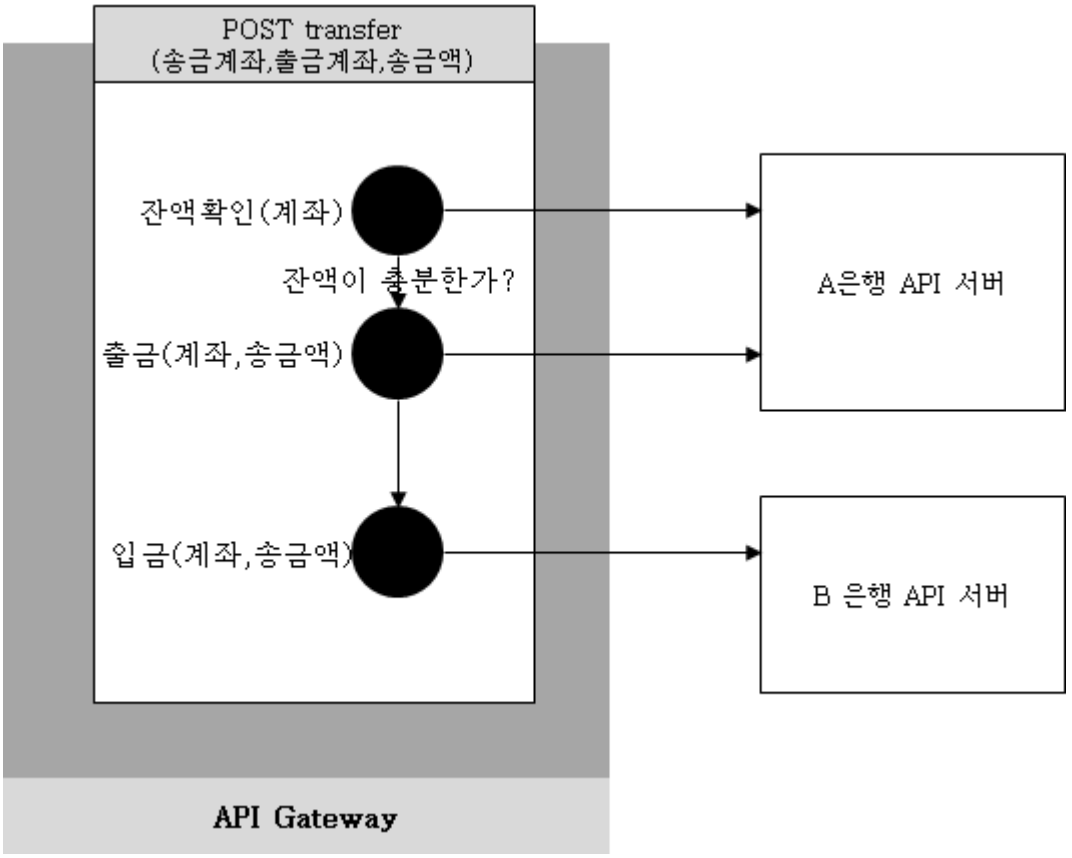
바로 클라이언트에게 ACK를 준후에 메시지 큐 연동을 이용해 메시지를 저장한 후 ,

로그 저장 API 서버의 성능이 맞춰 전달해준다.

클라이언트는 동기 호출이지만, 실제 메시지 흐름은 큐를 이용한 비동기 구조로 변경되었다.

4. 어그리게이션 (aggregation)

   여러개의 API를 묶어서 하나의 API로 만드는 작업을 이야기한다.



3-4. 로깅 및 미터링

1. api 호출 로깅

   모든 api 요청 로그를 api gateway에서 수집할 수 있기 때문에 로그 분석 기능이 있어야 한다.

2. api 미터링 차징

   API 미터링: 과금을 위한 API 호출 횟수, 클라이언트 IP, API 종류, IN/OUT 용량들을 측정, 기록하는 서비스

API 차징: 미터링이 된 자료를 기반으로 API 서비스 사용 금액을 금액 정책에 따라 계산해내는 서비스이다.