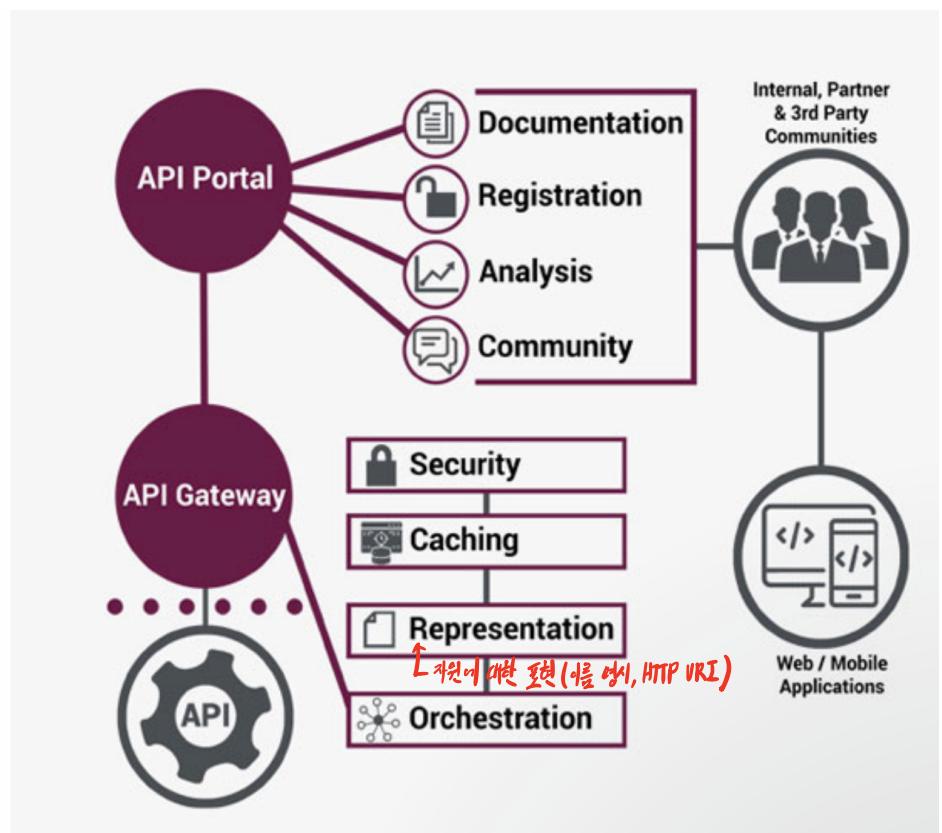


API architecture

Let's take a quick step back to understand how an API gateway fits into an API architecture. First, what's an API architecture?

(Unlike API design, which focuses on why the API is being created, the outcome, and how it will be executed, API architecture is defining the entire methodology and process for running and exposing APIs. It encompasses the API gateway (and how API security, caching, orchestration will work), developing an API portal for API analysis, API documentation, marketing APIs, making sure they work with web/mobile applications, and defining how they are exposed to internal, partner, and third-party developers.)

Having a complete API architecture will help your business with the entire API lifecycle management process.



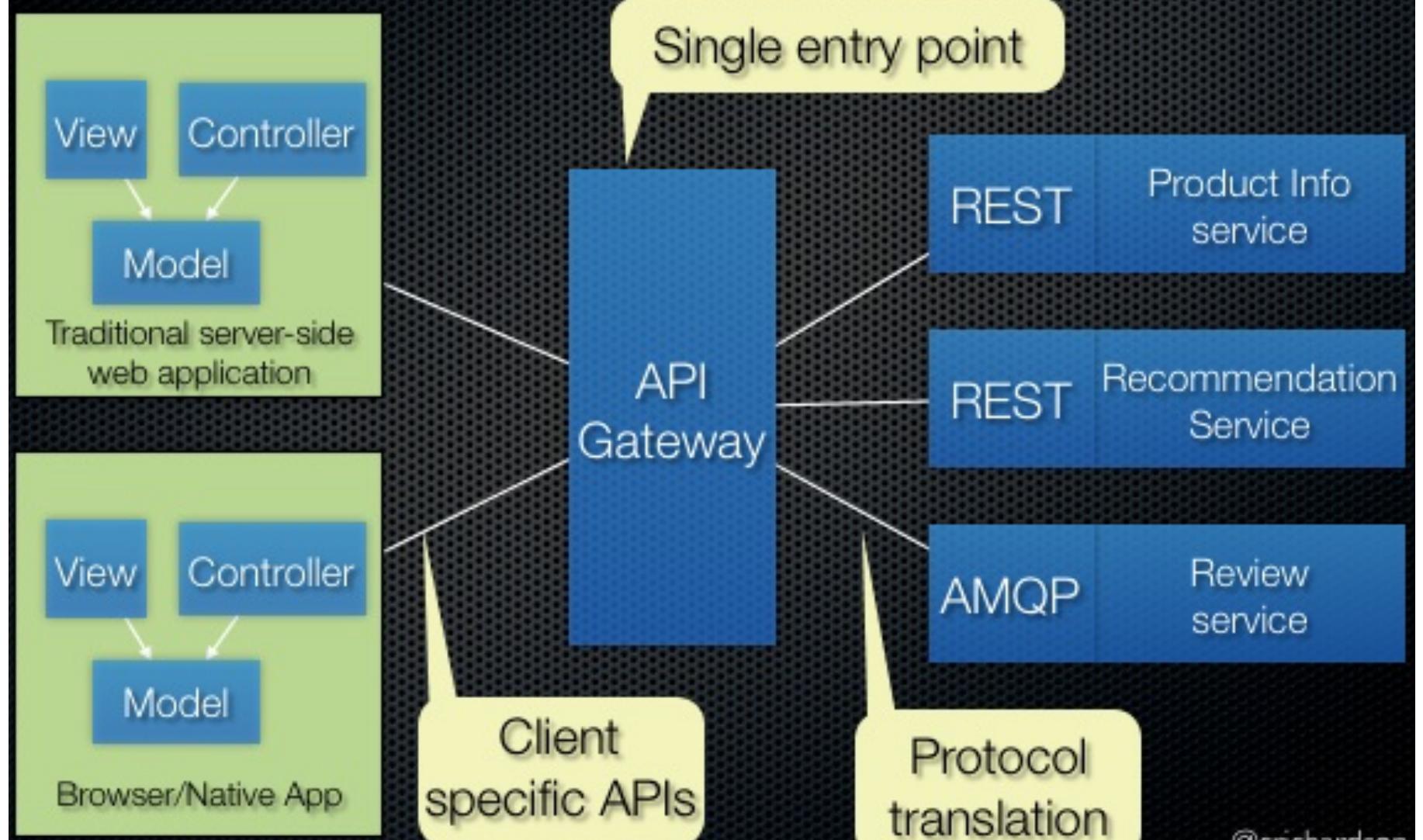
What is an API gateway?

An API gateway is programming that sits in front of an API (Application Programming Interface) and is the single entry point for defined back-end APIs and microservices (which can be both internal and external). Sitting in front of APIs, the gateway acts as protector, enforcing security and ensuring scalability and high availability. To put it simply, the API Gateway takes all API requests from a client, determines which services are needed, and combines them into a unified, seamless experience for the user.

Why are they important?

An API is useless unless it is delivered with consistent quality. A gateway is critical to help ensure great performance, high availability and elastic scalability of APIs by enabling enterprises to initiate delivery with uniform supporting services, including traffic management, transformation and system integration.

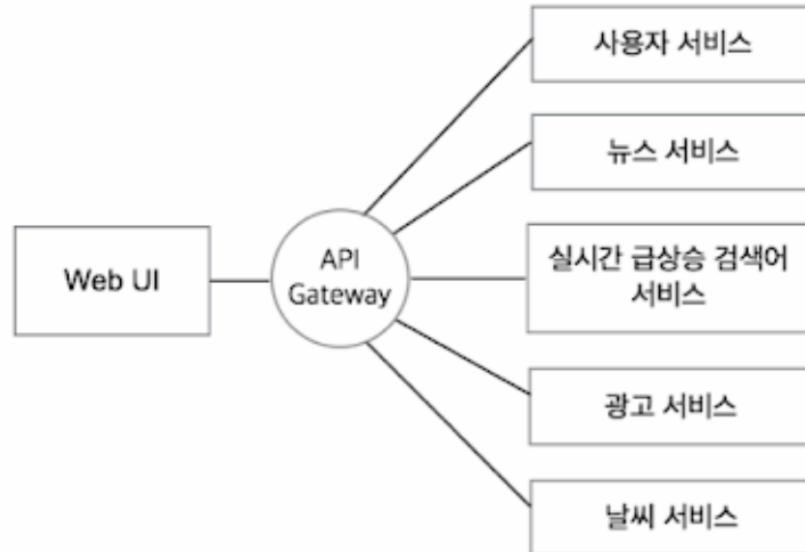
Use an API gateway



API GATEWAY

API GATEWAY는 이름에서도 유추할 수 있듯이 서비스로 전달되는 모든 API 요청의 관문(Gateway) 역할을 하는 서버 시스템의 아키텍처를 내부로 숨기고 외부의 요청에 대한 응답만을 적절한 형태로 응답

즉, 클라이언트는 (내부 구조가 모놀로티 아키텍처인지, 마이크로 소프트 아키텍처인지 알 필요가 없이) 서로 약속한 형태의 API 요청만을 서버로 보내면 됨



API Gateway의 장점

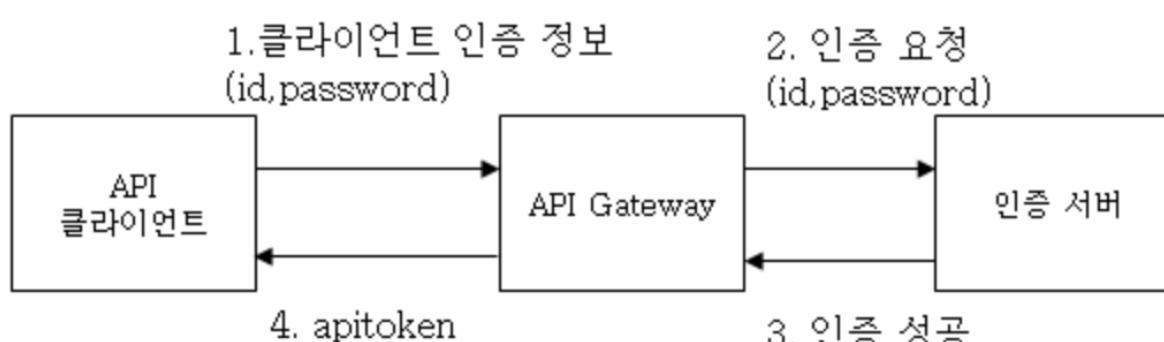
- 클라이언트의 요청을 일괄적으로 처리
- 전체 시스템의 부하를 분산 시키는 로드 밸런서의 역할
- 동일한 요청에 대한 불필요한 반복작업을 줄일 수 있는 캐싱
- 시스템상에 오고가는 요청과 응답에 대한 모니터링
- 시스템 내부에 아키텍처를 숨길 수 있음

이렇게 API Gateway를 이용하면 서비스 요청에 대한 처리를 하게되면 특정 서비스의 변경사항이 생기거나 서비스가 통합/분리 되더라도 클라이언트는 그 사실을 인지할 필요가 없이 API Gateway 내부의 변경사항만으로 처리가 가능

API 토큰 발급

인증 인가를 거칠 때 마다 매번 사용자의 인가/인증 절차를 거치기는 불편하다. 사용자로부터 매번 사용자 ID와 비밀번호를 받기는 번거롭고, 그렇다고 사용자 ID와 비밀번호를 저장해놓는 것은 해킹의 빌미를 제공한다. 그래서 보통 사용하는 방식이 토큰이라는 방식을 사용하는데, 사용자 인가가 끝나면, 사용자가 API를 호출할 수 있는 토큰을 발급해준다. API 서버는 이 토큰으로 사용자의 identity와 권한을 확인한 후, API 호출을 허가해준다.

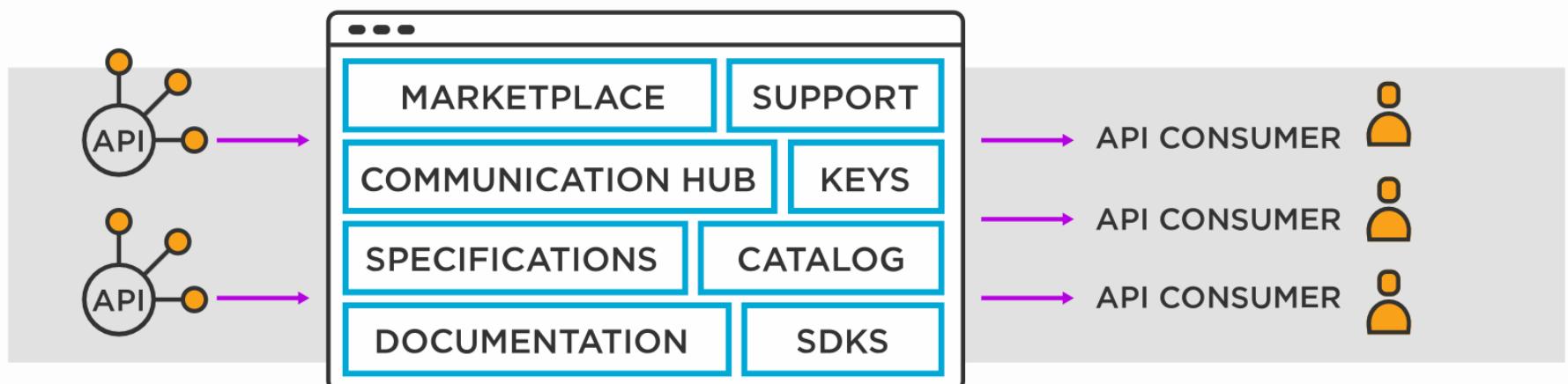
API 게이트웨이는 클라이언트를 인증한 후, 이러한 API 토큰을 생성 및 발급해주는 역할을 한다.



<그림. 일반적은 토큰 발급 절차>

API 포털이란 무엇입니까?

API 포털은 기본적으로 API 제작자(API 생성자)와 API 소비자(일반적으로 개발자 커뮤니티) 사이의 다리 역할을 합니다. API 포털을 통해 API 소비자는 가입하여 API를 사용할 수 있으며 API 통합 방법, API 관련 교육, 사용자 액세스 부여 또는 제공, 클라이언트 키 생성 등에 대한 개발자 안내를 포함하는 전체 라이프 사이클 전제에서 API에 대해 필요한 정보를 취득합니다. 성공적인 API 포털은 개발자에게 프로덕션 데이터가 로드된 샌드 박스 환경에 대한 액세스 권한을 부여하므로 개발자가 API를 쉽게 테스트할 수 있습니다. 대부분의 개발자는 API 테스트를 선호하기 때문에 우수한 API 포털은 해당 서비스를 제공하고 쉽게 액세스할 수 있도록 합니다.



API 포털을 통해 API를 개발자가 쉽게 검색하고 액세스하며 애플리케이션에 통합할 수 있는 제품으로 변환합니다. API 포털을 사용하여 API를 제품으로 패키징하고 홍보할 수 있으며, 개발자 온보딩, 참여 및 권한 부여 프로세스를 훨씬 더 간소화할 수 있습니다. API 제품을 광고하거나 호스팅할 수 있는 진열장이나 브로셔와 같은 API 포털을 생각해보십시오.

API 포털에는 문서, 사양, 보안, 가격 책정, 법적 고지, API 설계에 대한 완전한 투명성을 포함하여 개발자가 API에 대해 원하는 모든 정보가 들어있습니다. API 구현의 비즈니스 이점에 대한 추가 정보와 API의 성공적인 사용 사례도 포함될 수 있습니다. 실제로 이런 것들은 개발자가 해당 API를 사용할 가능성을 높이기 위해 권장됩니다. API 포털에는 알려진 문제, 해결 시간 및 도움 요청 방법도 포함되어어야 합니다.

API 포털은 단순한 단방향 통신이 아닙니다. API에 대한 문서 호스팅 외에도 FAQ, 기사, 토론 포럼 및 블로그와 같은 기타 기능이 포함되어 있습니다. 이를 통해 소비자와 생산자는 제안을 하고, 질문을 할 수 있으며 자신의 경험에 대해 이야기할 수 있습니다. API 제공업체는 포럼 및 블로그를 사용하여 API에 대한 새로운 이니셔티브 또는 변경 사항과 같은 추가 정보를 개발자 커뮤니티에 제공할 수 있습니다. 포럼을 통해 API 소비자는 피드백을 기재하고 버그를 보고할 수도 있습니다. 철저하게 잘 구축된 API 포털은 가장 중요한 개발자 참여를 장려하여 API 사용을 증가시킵니다.

API 게이트웨이 대 API 포털

API 게이트웨이는 조직 내 API 트래픽을 제어하고 API 사용량 및 부하 분산을 관리하는 것입니다. 이 기능은 자동으로 수행되며 일반적으로 인간의 개입이 많이 필요하지 않습니다. API 게이트웨이는 백엔드에 가깝지만 API 전략에서 매우 중요한 부분입니다.

API 포털은 API 전략의 프런트 엔드입니다. 소비자가 API에 등록하고 문서, 블로그 및 커뮤니티 포럼과 같은 필요한 모든 정보를 가져와 성공적인 통합을 보장하고 피드백 및 버그를 보고하거나 추가 지원을 받을 수 있는 장소입니다.