10주차 발표_민서연

API Gateway

01 MSA

MSA

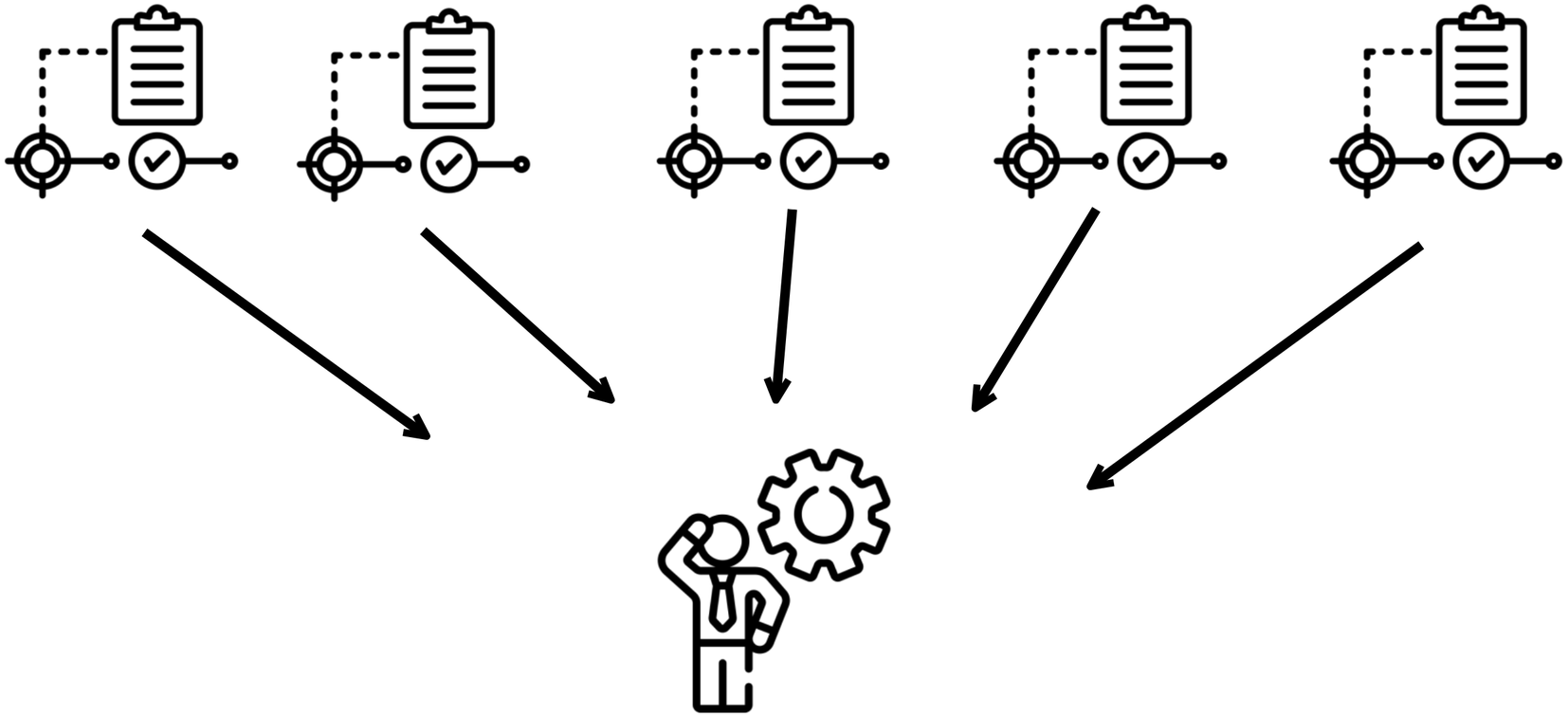
MicroService Architecture

독립적으로 배포 가능한 각각의 기능을 수행하는 서비스로 구성된 프레임워크 구조

API를 통해서만 상호작용이 가능
즉, end-point(접근점)을 API 형태로 외부에 노출하고 실질적인 세부 사항은 모두 추상화

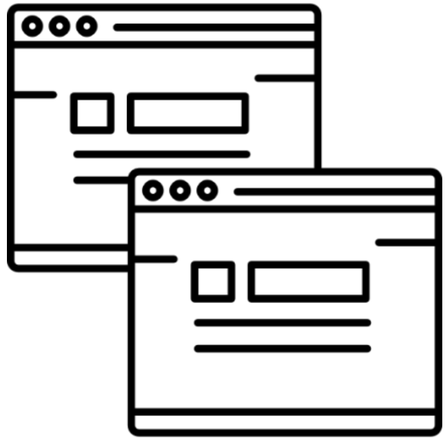
서비스의 복잡도 줄여줌
변경에 따른 영향을 최소화 하면서 개발과 배포를 할 수 있음

01 MSA



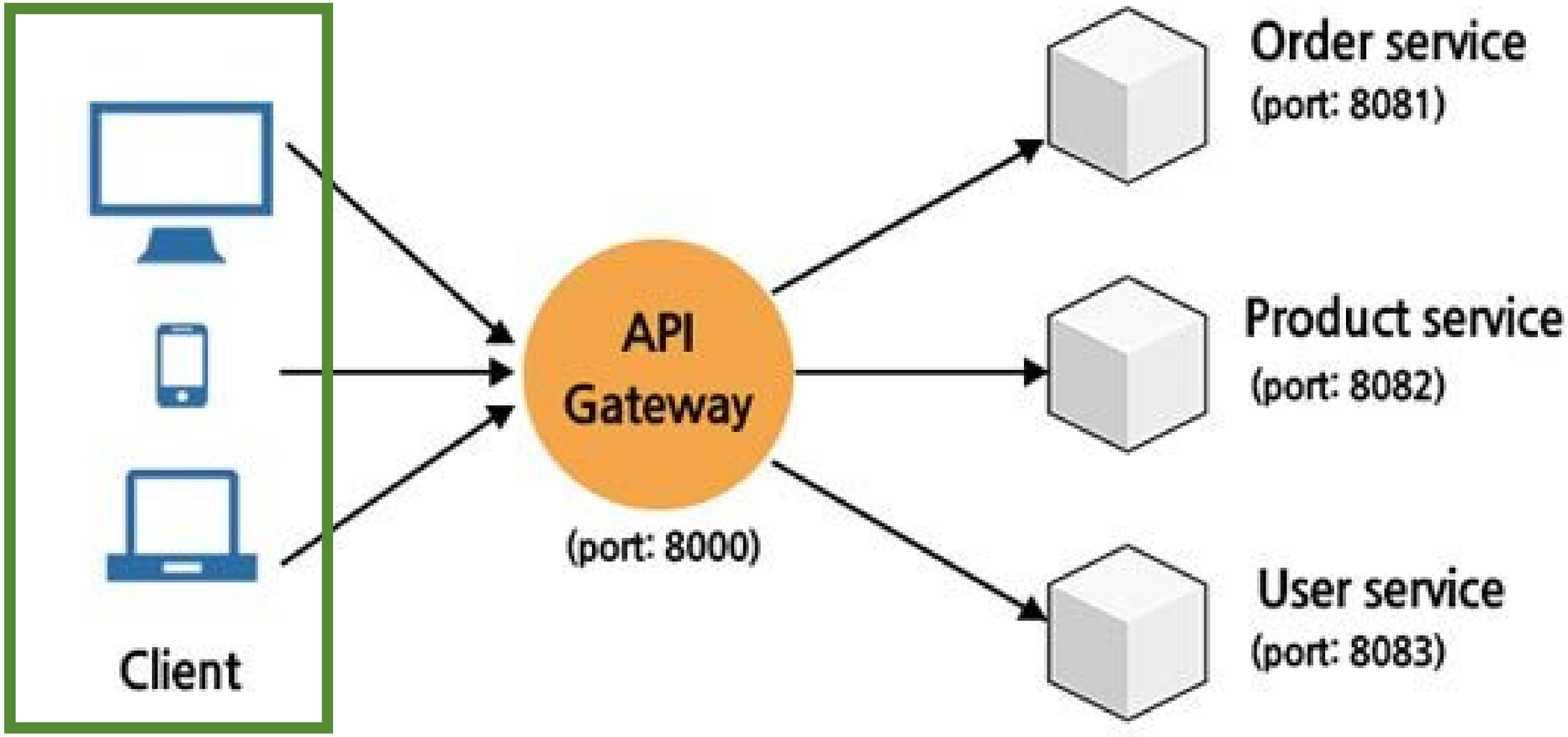
작은 단위의 서비스가 많아질수록 많은 서비스들의 엔드포인트를 관리하는데 있어서 어려움이 생김

01 MSA



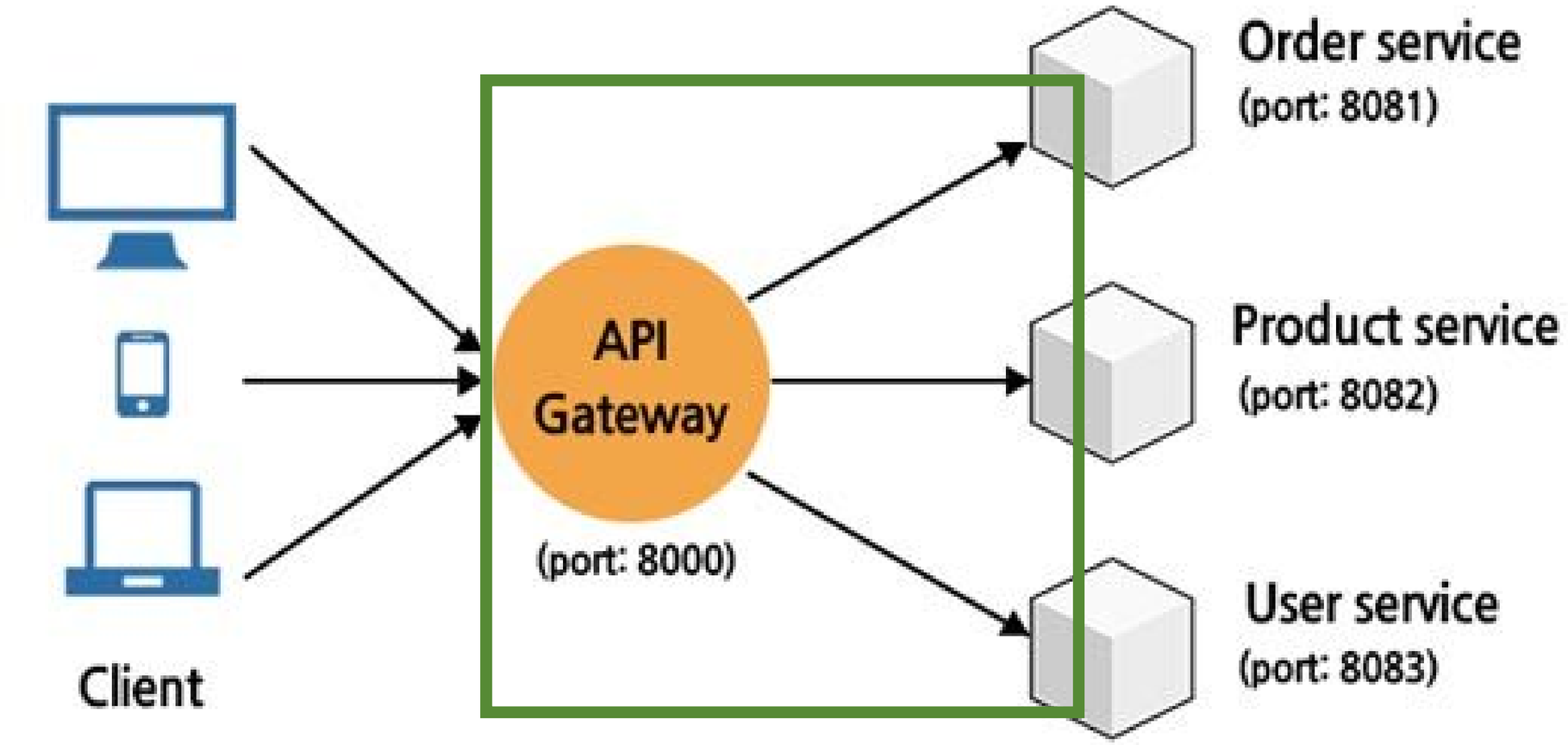
각각의 서비스마다 공통적으로 들어가는 기능(인증/인가, 로깅등)을 중복으로 개발해야 함

02 API Gateway



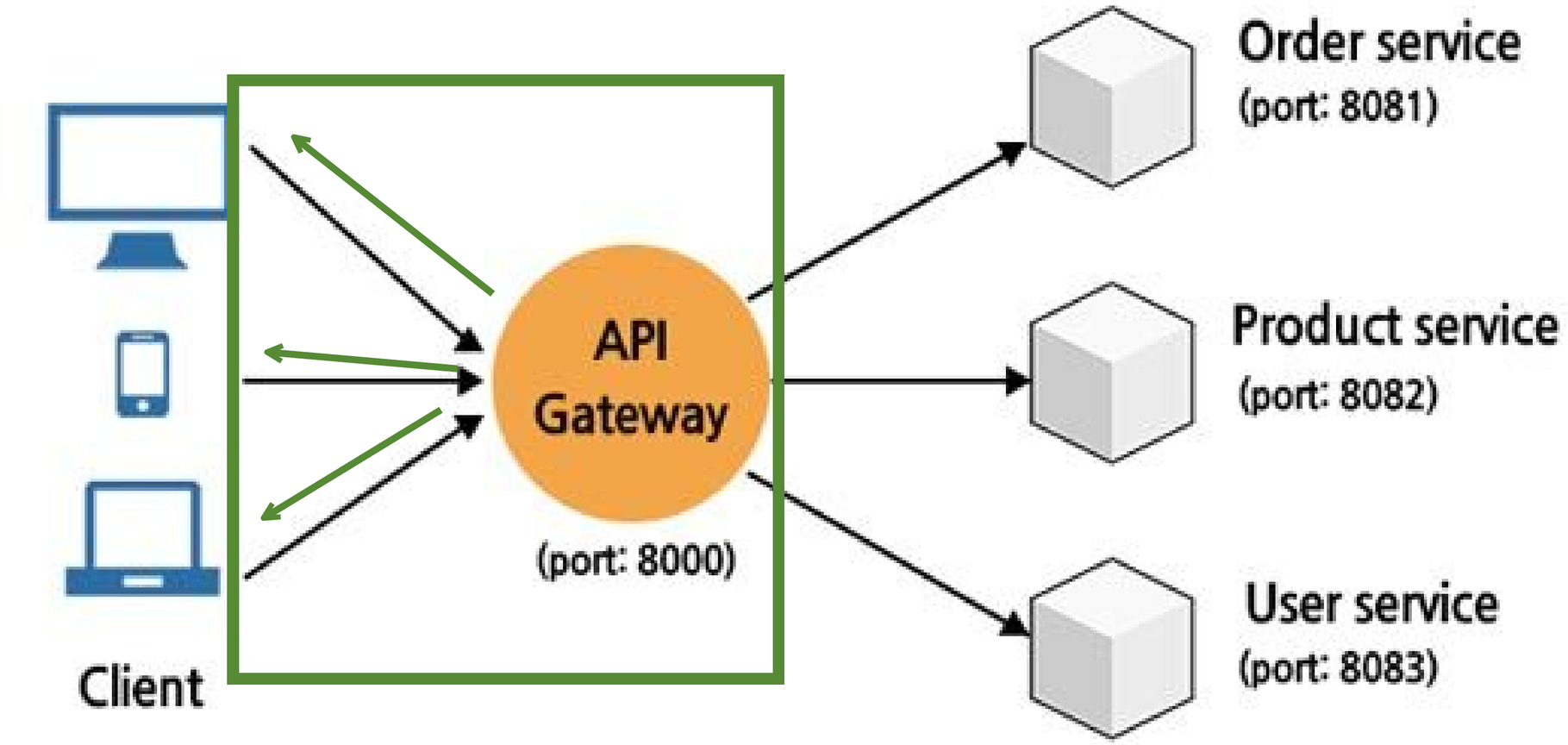
클라이언트는 엔트포인트 대신 API Gateway로 요청을 보냄

02 API Gateway



요청 받은 API Gateway는 설정에 따라 각 엔드포인트로 클라이언트를 대신 해서 요청

02 API Gateway

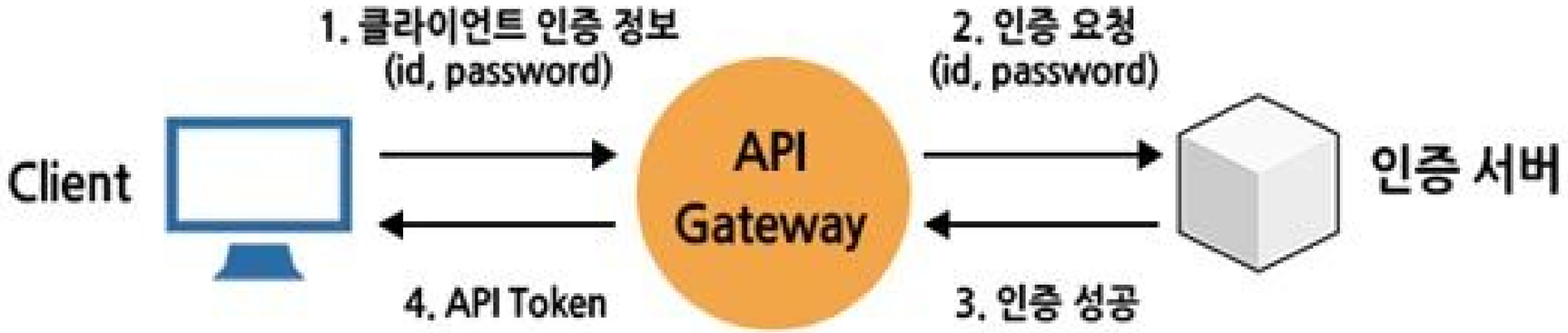


응답을 받으면 다시 클라이언트에게 전달함

02 API Gateway

주요기능

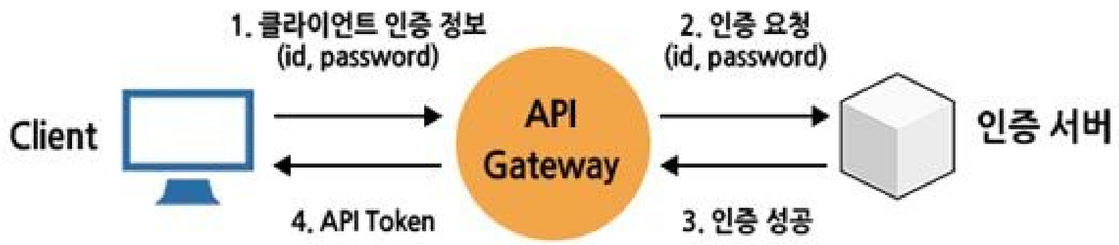
인증/인가 및 토큰 발급



02 API Gateway

주요기능

인증/인가 및 토큰 발급



+

공통 로직 처리
로드 밸런싱
메디에이션 기능
.
.
.

03 Spring Cloud Gateway

Spring에서 제공하는 오픈 소스 기반의 Gateway 서비스

Netty 기반의 비동기 요청 처리 제공

Spring에서 제공하는 다양한 component를 조합해서 효율적인 개발 가능

Netty: 비동기 이벤트 기반 네트워크 애플리케이션 프레임워크.

03 Spring Cloud Gateway

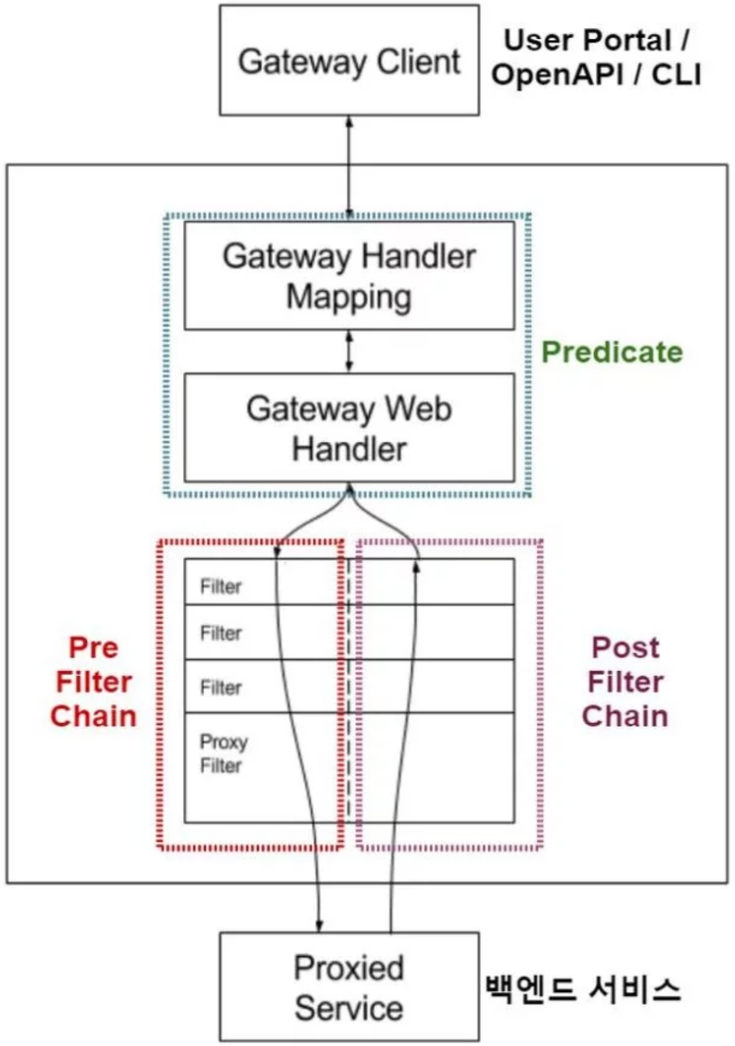


그림. 2 Spring Cloud Gateway의 개념도

내부 기능 특성에 따라 **Predicate**와 **Filter Chain**으로 요청 전달

Predicate는 요청 받은 API가 지정된 Route 경로에
포함되는지 식별 후 Filter Chain으로 요청 전달

Filter Chain은 각각의 하나의 서비스를 제공하는 Filter를 등록

PreFilter Chain: 백엔드 서비스에 요청하기 전
PostFilter Chain: 백엔드 서비스에 요청한 후

03 Spring Cloud Gateway

장점

Spring 오픈소스 기반 사용으로
초기 라이선스 투자 비용 없음

Spring에서 제공하는 다양한 서비스
(Spring Boot, Spring Security, OAuth2 Client, Redis등)
와 조합해서 서비스 가능

단점

다양한 기본 서비스가 제공되는 만큼
적절한 기능 선택이 어려움

상용 SW에서 제공하는 GUI 기반에
관리 기능 미 제공

03 Spring Cloud Gateway

API 게이트웨이 요구사항

웹 콘솔에서 OAuth2를 이용한 사용자 인증과 HMAC 해시 인증키를 사용한 사용자 인증을 지원

사용자 인증Session 관리를 위해 Redis를 사용

K8S 기반의 DevOps 운영환경을 구성하여 원활한 Scale Out 지원

인증 방식 및 지원 API에 따라 독립적인 서비스 제공 필요

03 Spring Cloud Gateway

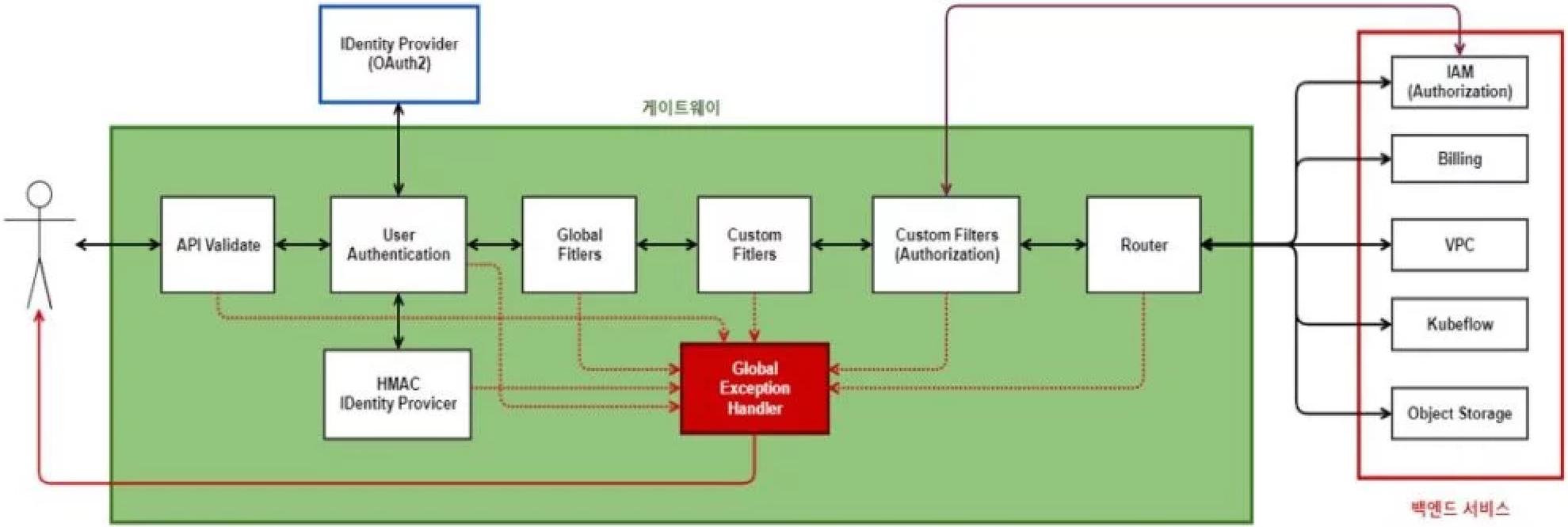


그림4. API 게이트웨이 내부 구성

03 Spring Cloud Gateway

Spring Security에서 제공하는 OAuth2 인증 처리를 사용하여 OAuth2 인증 구현
대량의 사용자 인증 요청에 대응하기 위해 session 관리를 Redis 기반의 인증 Session 사용 (Spring 라이브러리 지원)

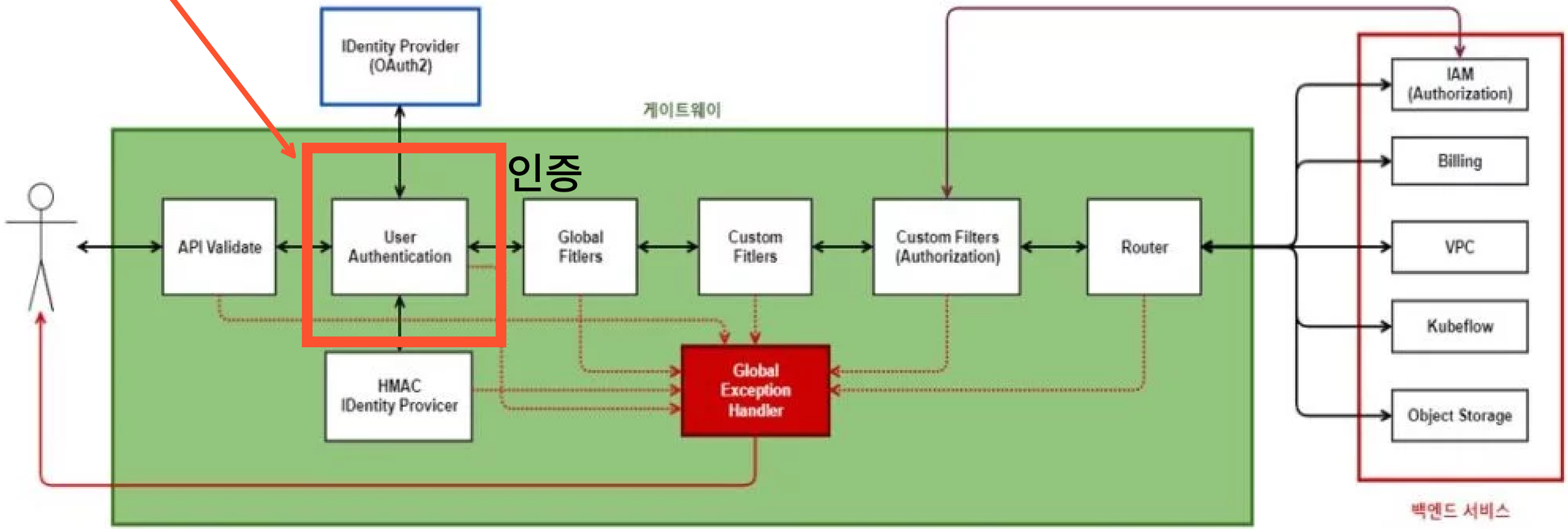


그림4. API 게이트웨이 내부 구성

03 Spring Cloud Gateway

MSA 백엔드 서비스의 권한 인가를 위해 별도의 IAM(Identity and Access Management) 서비스를 구현
사용자 인증을 통과한 API 요청에 대해 IAM 서비스에 해당 요청에 대한 권한 보유 여부를 식별

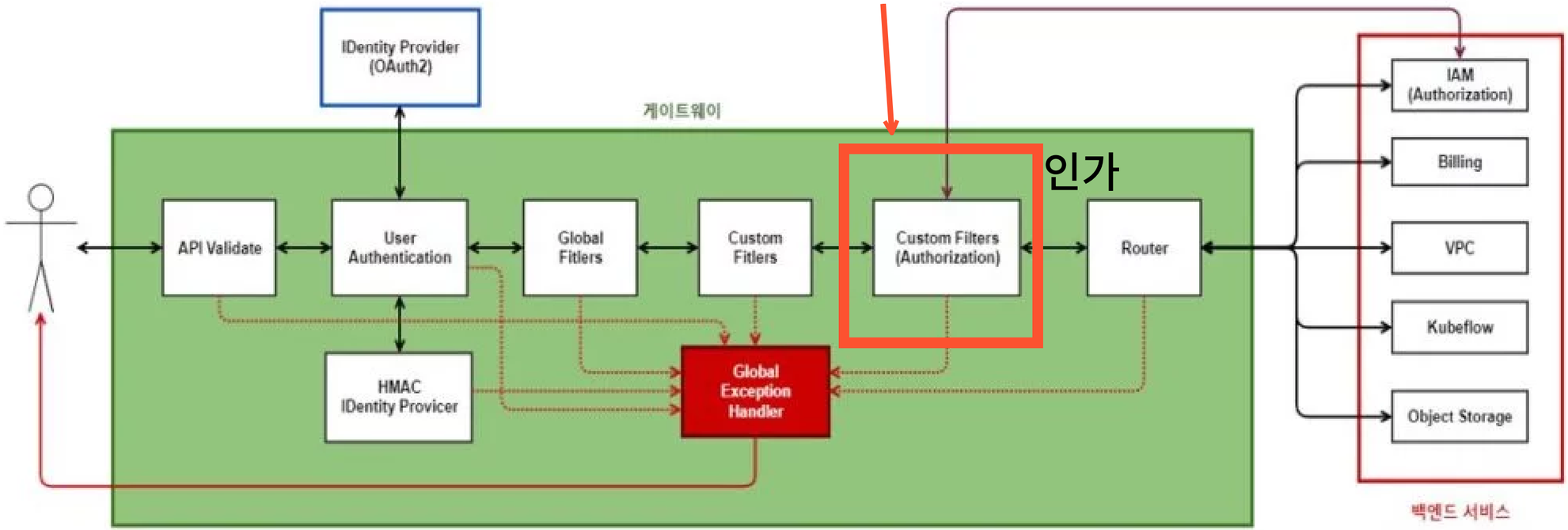


그림4. API 게이트웨이 내부 구성

03 Spring Cloud Gateway

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-web-services'
    implementation 'org.springframework.cloud:spring-cloud-starter-gateway'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.security:spring-security-test'
}
```

의존성 주입

03 Spring Cloud Gateway

```yaml
spring:
  cloud:
    gateway:
      routes:
        - id: auth-service
          uri: http://localhost:8081
          predicates:
            - Path=/auth/**
          filters:
            - StripPrefix=1
            - name: AuthorizationHeaderFilter
              args:
                tokenHeader: Authorization
```

gateway 설정 파일 생성

02 출처

https://wildeveloperetrain.tistory.com/205
https://s-core.co.kr/insight/view/spring-cloud-gateway-%EA%B8%B0%EB%B0%98%EC%9D%98-api-%EA%B2%8C%EC%9D%B4%ED%8A%B8%EC%9B%A8%EC%9D%B4-%EA%B5%AC%EC%B6%95/
https://zayson.tistory.com/entry/Spring-Cloud-Gateway%EB%A5%BC-%EC%9D%B4%EC%9A%A9%ED%95%9C-%EC%84%9C%EB%B9%84%EC%8A%A4-%EB%9D%BC%EC%9A%B0%ED%8C%85-%EB%B0%8F-JWT-%ED%86%A0%ED%81%B0-%EA%B2%80%EC%A6%9D