gRPC

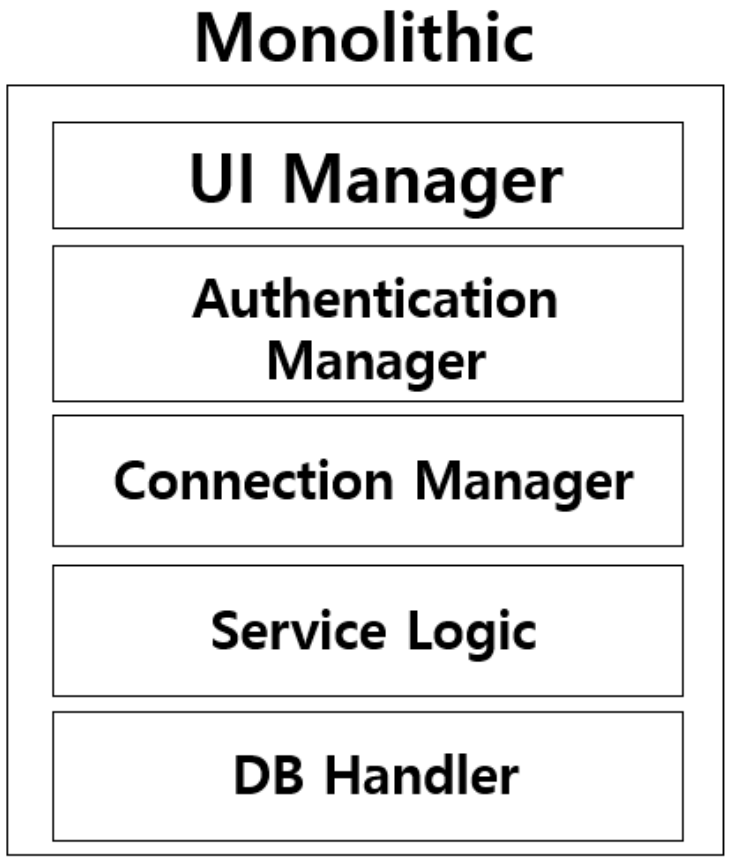
https://youtu.be/30GlT3mMJ4Y

Monolithic

- 서비스 혹은 애플리케이션의 전통적인 구조

- 장점
  - 각 컴포넌트들이 통일성을 가짐
  - 모든 개발이 하나의 서비스에만 집중 가능

- 단점
  - 규모가 커질수록 복잡도가 심각하게 증가
  - 종속성이 강함
  - 유연하지 못함

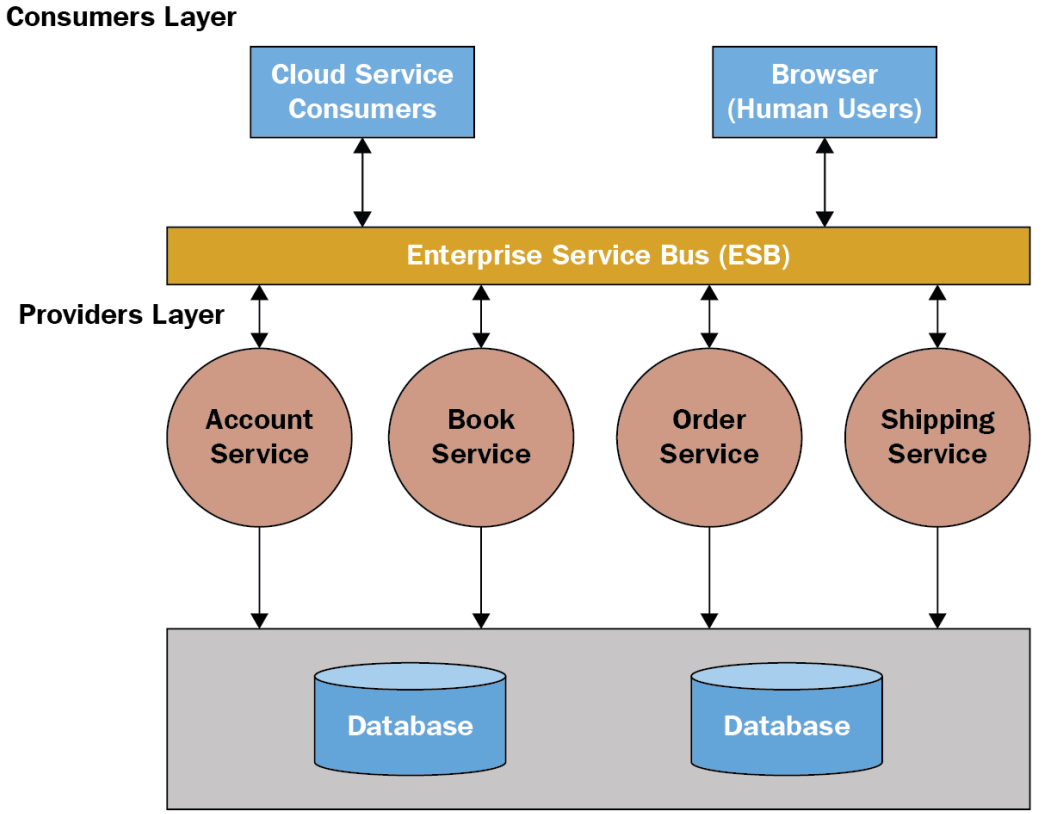


SOA

- 서비스 지향 아키텍처 (Service Oriented Architecture)
- 대규모 컴퓨터 시스템 구축 시 각 서비스(소프트웨어 기능)를 캡슐화
- 서비스 : 고차원의 비즈니스 개념을 캡슐화 한 것
- 서비스의 특징
  - 서비스 인터페이스는 플랫폼에 독립적
  - 서비스는 동적으로 검색 및 호출될 수 있음
  - 서비스는 자신의 상태를 스스로 유지

SOA

- 서비스 사용자 (Service Consumer)
  - ‘서비스 제공자’에 의해 제공되는 하나 이상의 서비스를 사용
- 서비스 제공자 (Service Provider)
  - ‘서비스 이용자’가 호출 시 결과값을 제공
  - ‘서비스 제공자’는 다른 서비스의 사용자가 될 수 있음
- 서비스 레지스트리 (Service Registry)
  - ‘제공자’는 자신이 제공하는 서비스를 등록하고, ‘사용자’는 원하는 서비스를 이용

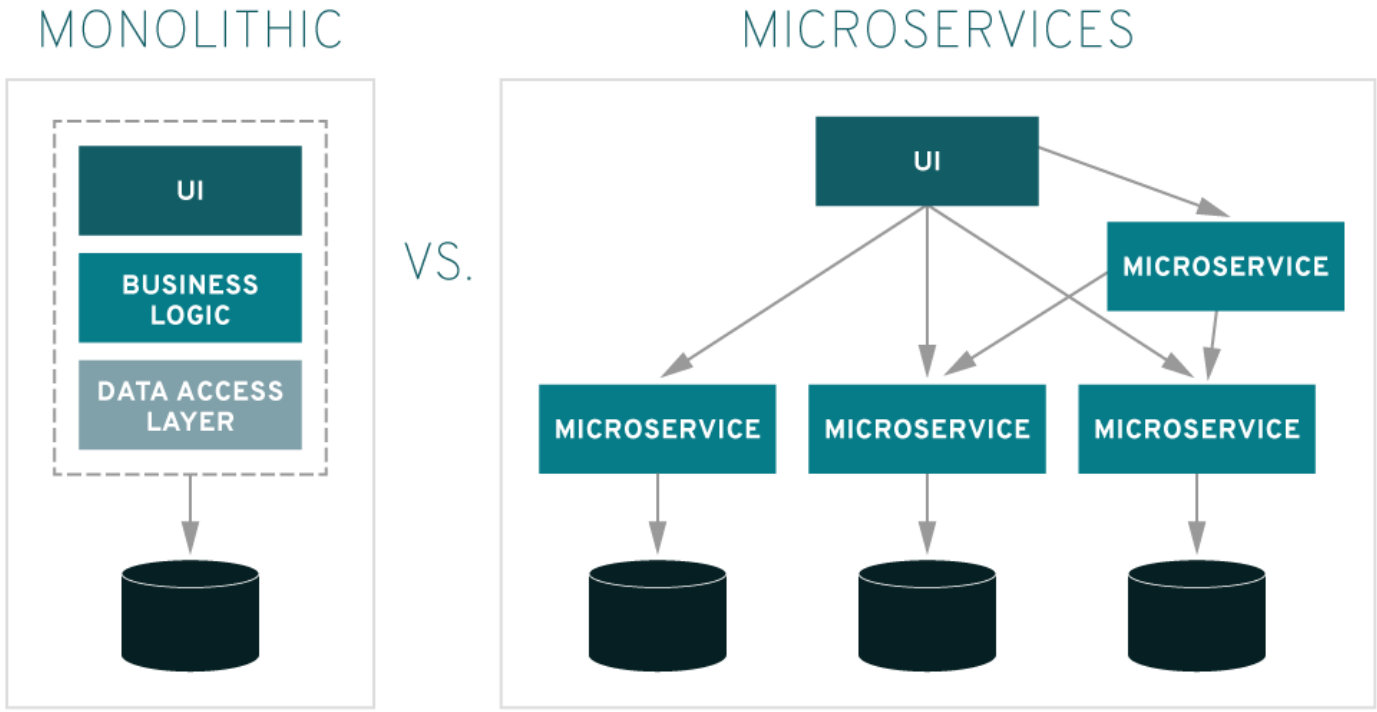


SOA

- 유연성 향상
- 서비스 재사용
- 손쉬운 유지관리
- 확장성
- 안정성

Microservice

- 애플리케이션을 핵심 기능에 따라 세분화 (모듈화)
- loosely-coupled
- 유연한 확장성
- 배포의 용이성
- 코드 재사용 가능



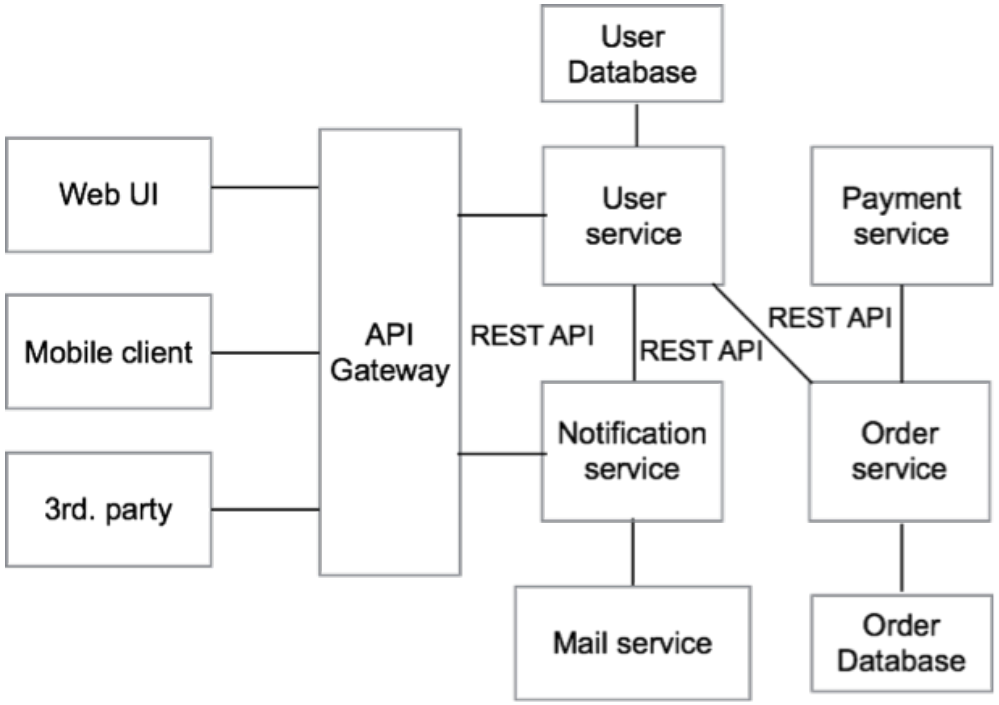
SOA vs Microservice

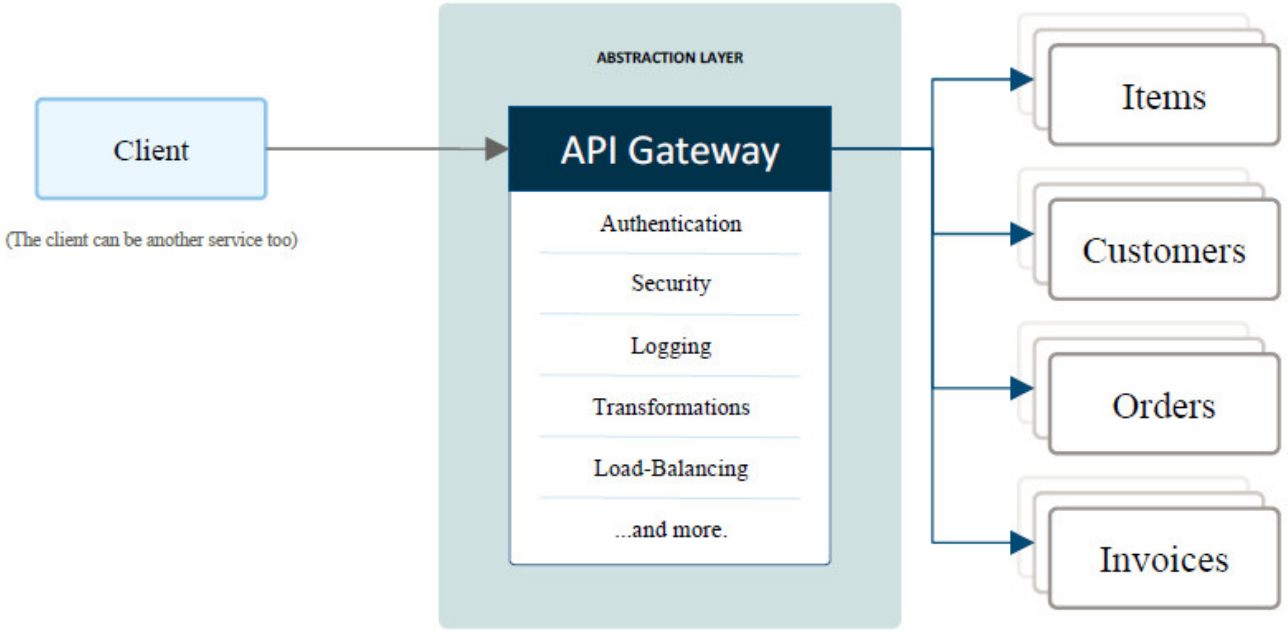
- SOA
  - 아키텍처 접근 방식
  - 통신에 ESB(비즈니스 구성 요소를 통합하는 미들웨어) 이용

- 마이크로서비스
  - 애플리케이션 구현 전략
  - API를 통한 통신으로 언어의 제약이 없음

Microservice Architecture (MSA)

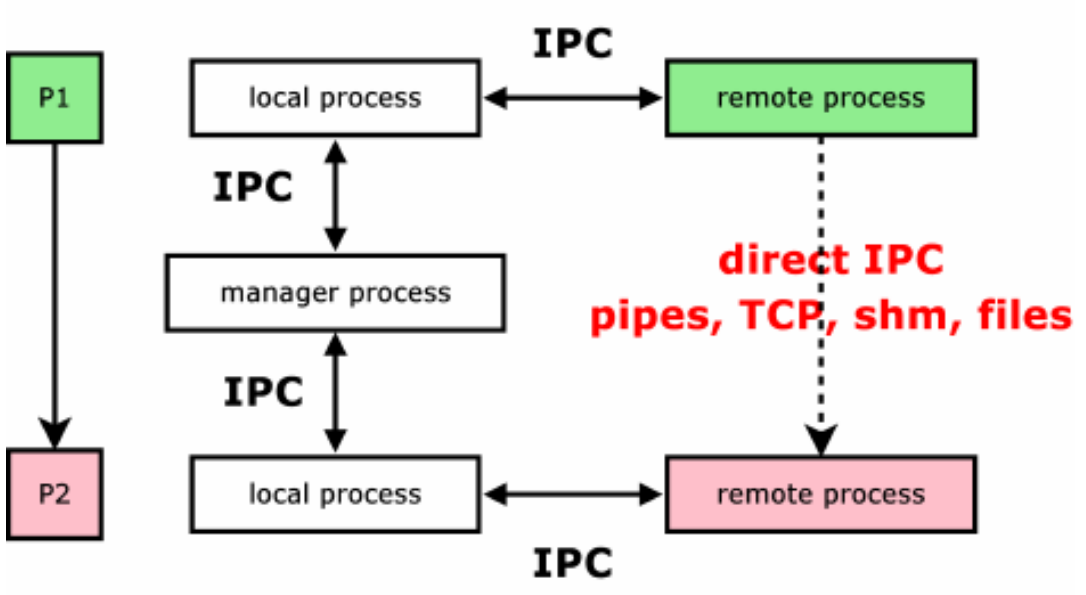
- 마이크로서비스를 설계하는 패턴





IPC

- 프로세스 간 통신 (Inter-Process Communication)
- 커널에서 프로세스끼리 서로의 주소공간에 침범하지 못하도록 제어
- 프로세스끼리는 IPC를 통해 통신
- 종류
  - 시그널
  - 파이프
  - 메세지 큐
  - etc.



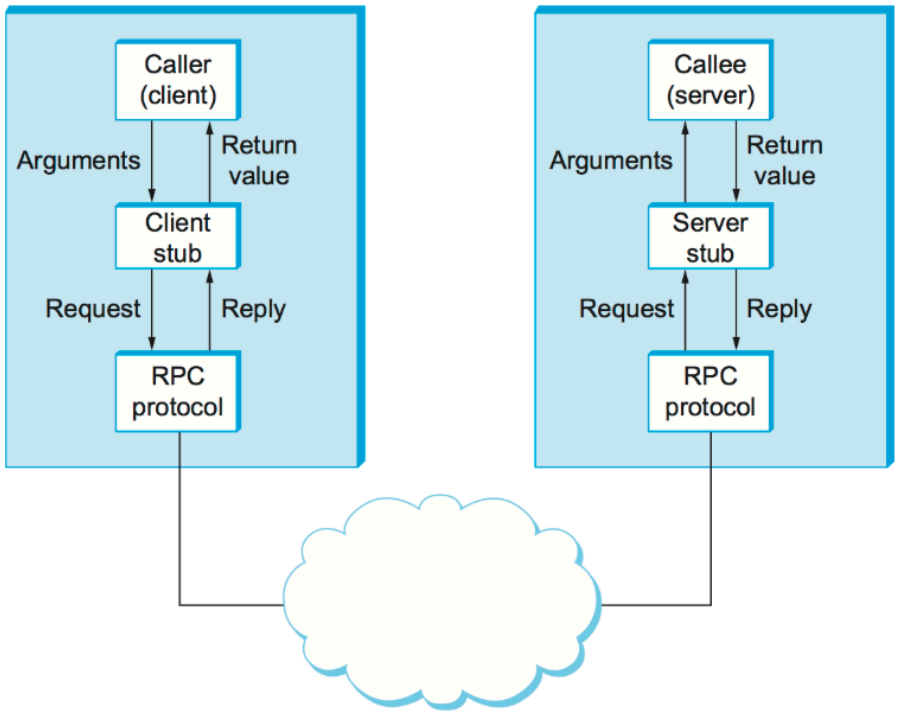
RPC

- Remote Procedure Call
- IPC의 한 종류
- 원격지의 프로세스에 접근하여 프로시저 또는 함수를 호출하여 사용하는 방법
- 서비스 간의 프로시저 호출을 가능하게 해주어, 언어에 구애 받지 않고 확장이 가능

RPC

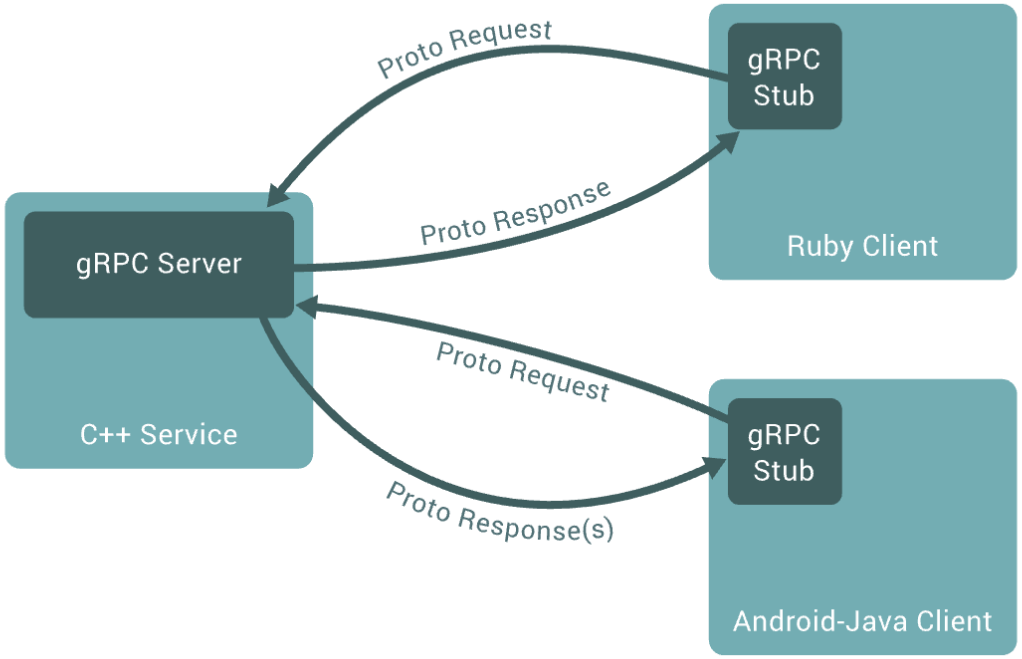
- IDL(Interface Definition Language)을 통해 호출 인터페이스 정의

- IDL을 기반으로 rcpgen 컴파일러를 이용하여 클라이언트의 stub, 서버의 skeleton 생성



gRPC

- gRPC Remote Procedure Calls
- 구글의 단일 범용 RPC 인프라인 Stubby에서 시작
- 높은 생산성과 효율적인 유지보수, 다양한 언어와 플랫폼 지원, 높은 메시지 압축률
- HTTP/2



gRPC

- Official Support

Language	OS	Compilers / SDK
C/C++	Linux, Mac	GCC 4.9+, Clang 3.4+
C/C++	Windows 7+	Visual Studio 2015+
C#	Linux, Mac	.NET Core, Mono 4+
C#	Windows 7+	.NET Core, NET 4.5+
Dart	Windows, Linux, Mac	Dart 2.2+
Go	Windows, Linux, Mac	Go 1.13+
Java	Windows, Linux, Mac	JDK 8 recommended (Jelly Bean+ for Android)
Kotlin	Windows, Linux, Mac	Kotlin 1.3+
Node.js	Windows, Linux, Mac	Node v8+
Objective-C	macOS 10.10+, iOS 9.0+	Xcode 7.2+
PHP	Linux, Mac	PHP 7.0+
Python	Windows, Linux, Mac	Python 3.5+
Ruby	Windows, Linux, Mac	Ruby 2.3+

gRPC

- 서비스와 메시지 정의를 위해 Protocol Buffers(ProtoBuf) IDL 사용

- 아래와 같은 코드 작성을 통해 서버와 클라이언트(stub) 코드 자동 생성

```
// 헤더 부분 생략..
// Greeter 서비스 정의
service Greeter {
  // 서비스의 RPC 정의
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}
// 요청 메시지 정의
message HelloRequest {
  string name = 1;
}
// 응답 메시지 정의
message HelloReply {
  string message = 1;
}
```

MSA with gRPC

- 비즈니스 로직에 집중하여 빠른 서비스 개발 가능
- 간단한 설치 및 빠른 배포가 가능
- 다양한 언어 및 플랫폼에 적용 가능
- API 호출로 인한 성능 저하 개선

gRPC vs REST API

- REST API를 제공하는 서비스 개발에는 gRPC가 적합하지 않음
- gRPC의 학습 곡선이 REST API에 비해 가파름
- 브라우저에서 직접 호출할 경우 gRPC 사용 불가

gRPC in Hyperledger Fabric

- 하이퍼레저 패브릭의 체인코드는 gRPC를 통하여 동작

- Simple service definition
  - 수많은 노드가 요청/응답을 주고 받아야 하는 상황에 적합

- Bi-directional streaming
  - REST 방식에서는 양방향 통신을 지원하지 않음
  - 양방향의 복잡한 통신을 빠르게 처리해야 함

Q & A