마이크로 서비스 아키텍처로 개발하기

안재우
Platform Architecture팀
SK Planet

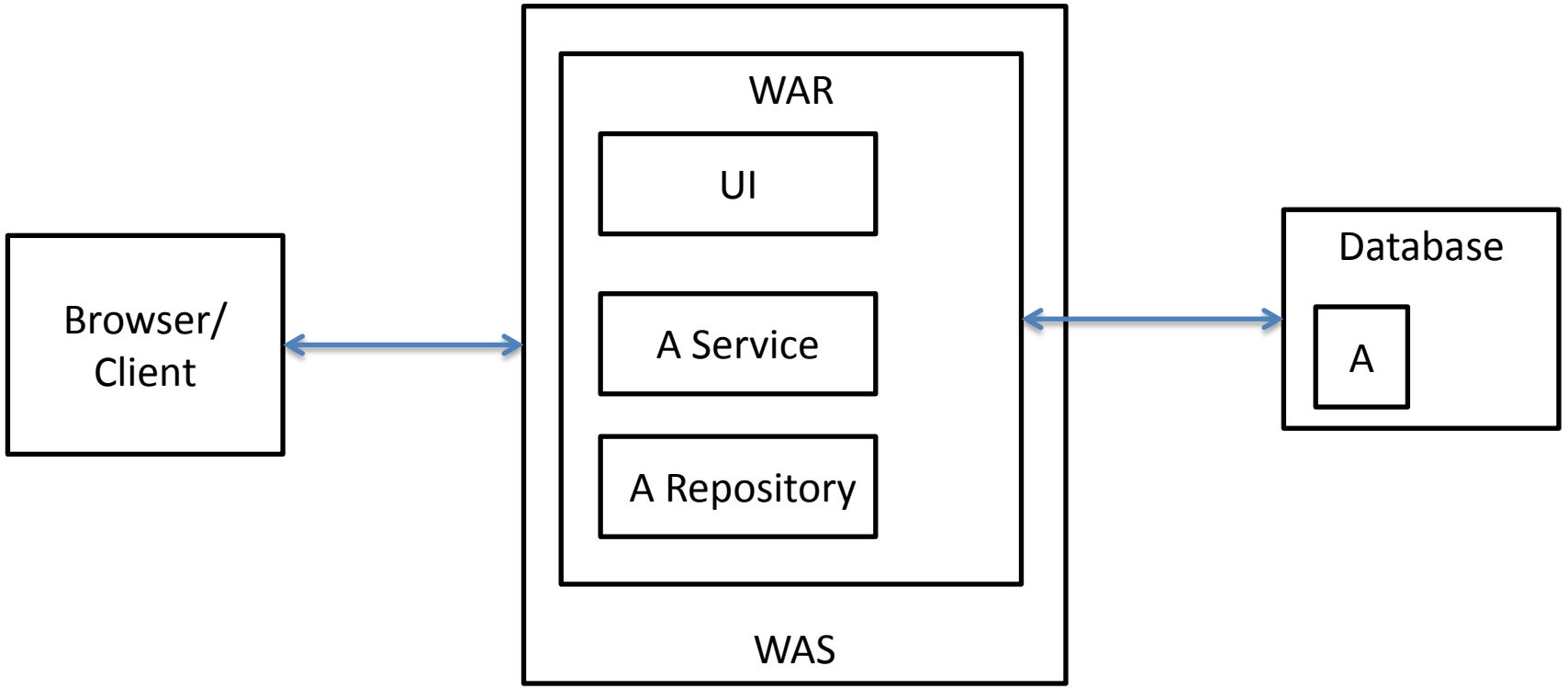
About me

- SK 플래닛 Platform Architecture 팀
- 전 NCSOFT 인프라플랫폼팀
- 전 닷넷엑스퍼트 수석컨설턴트

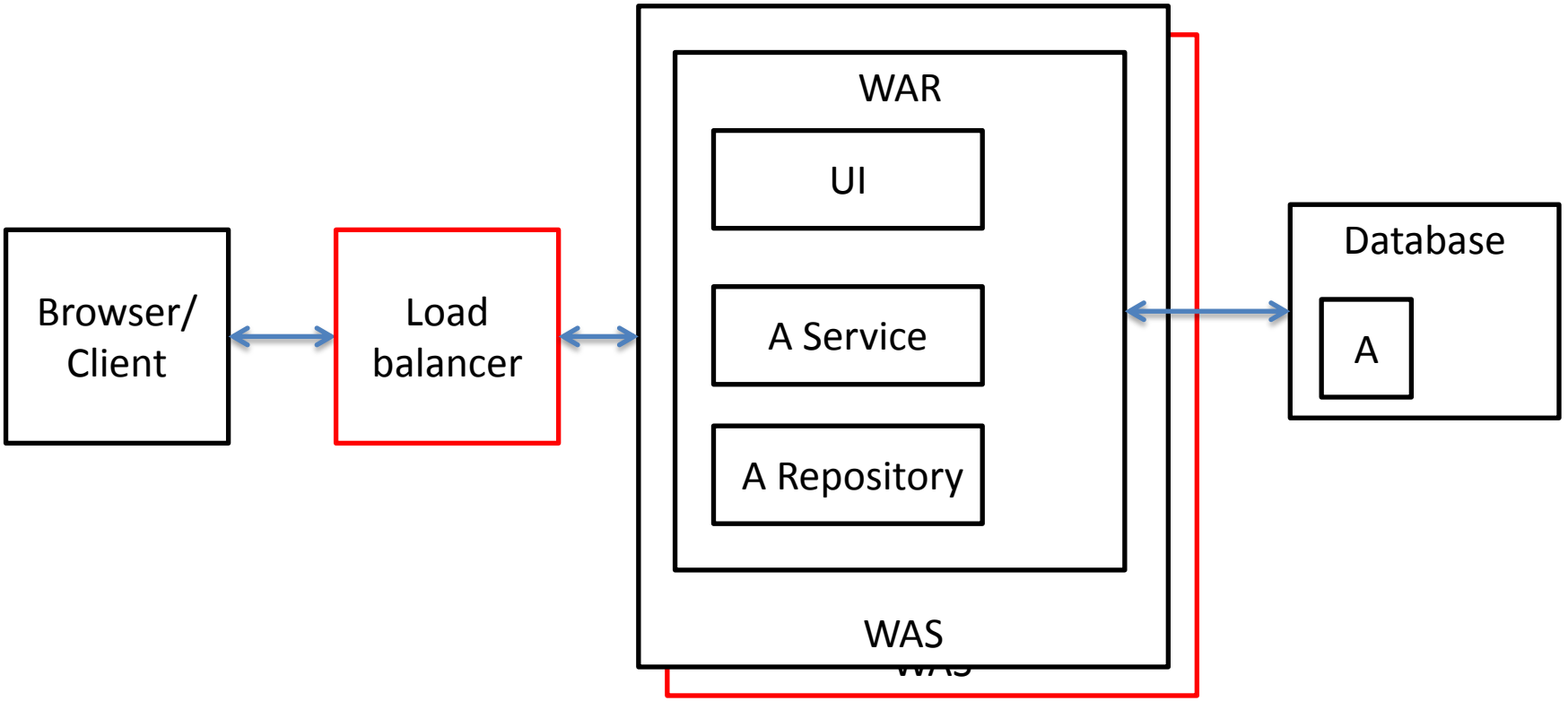
오늘 얘기할 내용은

- 마이크로 서비스 아키텍처란?
- 장단점은?
- 무엇이 필요하고, 어떤 점을 고려해야 하는지?
- 우리는 어떻게 하고 있는지?

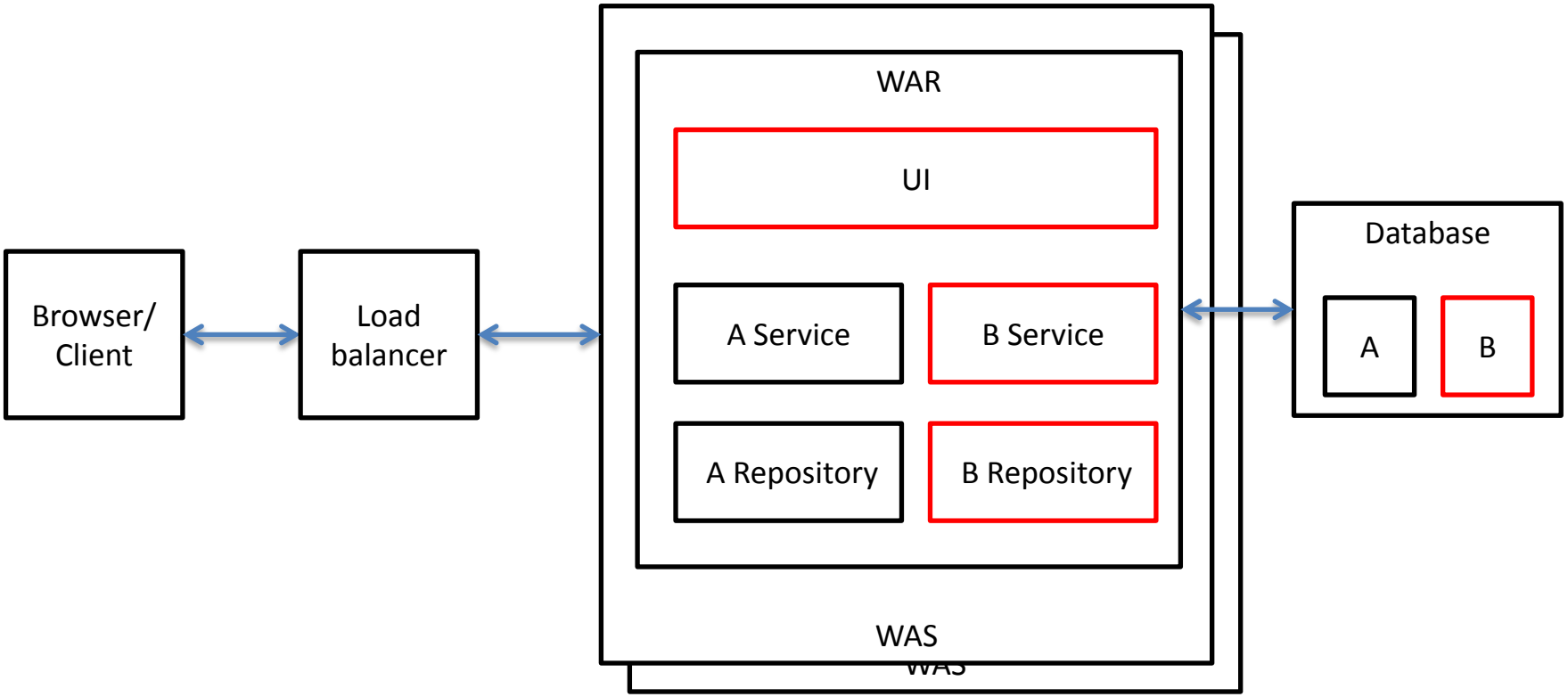
전통적인 Web App 아키텍처



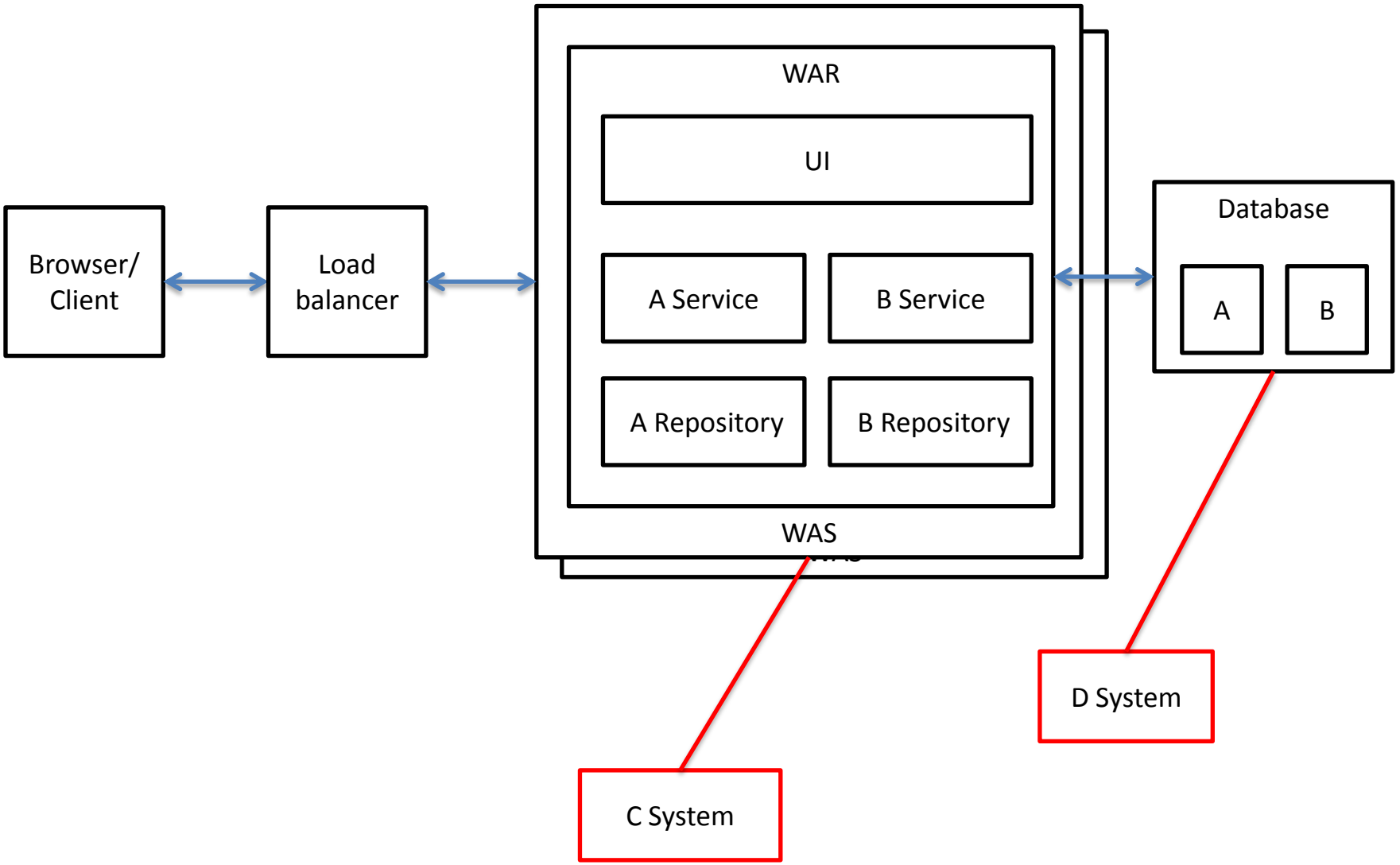
이중화/로드밸런싱


Browser/
Client
Load
balancer
WAR
UI
A Service
A Repository
WAS
Database
A

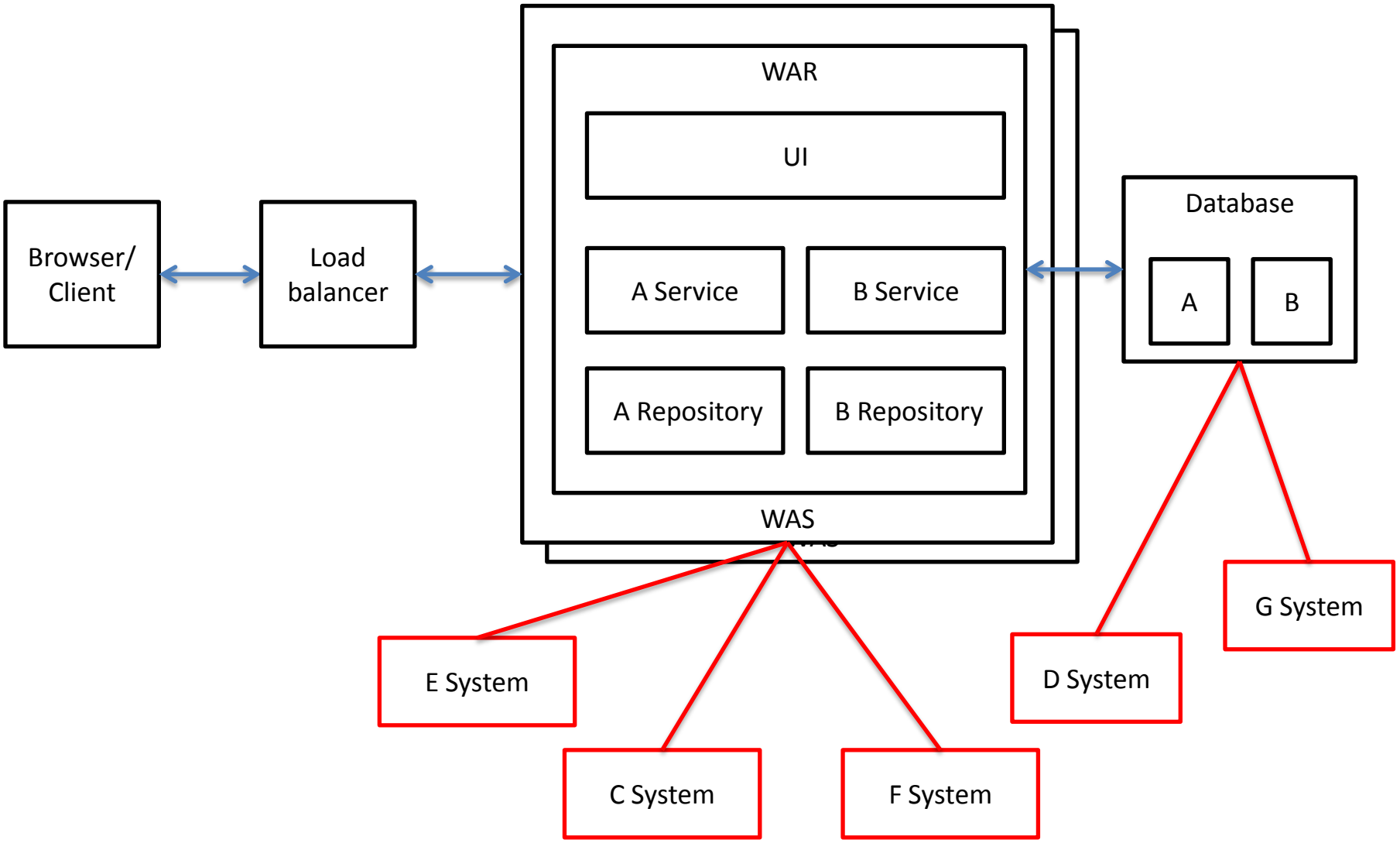
기능 추가


Browser/
Client
Load
balancer
WAR
UI
A Service
B Service
A Repository
B Repository
WAS
Database
A
B

시스템 연계/통합


Browser/
Client
Load
balancer
WAR
UI
A Service
B Service
A Repository
B Repository
WAS
Database
A
B
C System
D System

시스템 연계/통합


Browser/
Client
Load
balancer
WAR
UI
A Service
B Service
A Repository
B Repository
WAS
Database
A
B
E System
C System
F System
D System
G System

문제점

- 코드가 너무 커져서 유지보수하기 힘들어요.
- 시스템을 분리하고 싶어요.
- DB를 분리하고 싶어요.
- 연계 시스템이 변경된대요.
- 연계 시스템이 장애나서 우리 서비스도 장애예요.
- 사소한 수정인데도 전체 배포를 하고, QA를 거쳐야 해요.
- 새로운 걸 추가하는 건 상관없는데, 기존 로직/데이터를 변경하면 무슨 문제가 생길지 몰라요.
- 저희도 새로운 버전/기술을 써보고 싶은데...

가면 갈수록

- 뭔가 바꾸는게 두려워진다.
- 개발자들이 구닥다리 기술의 족쇄에서 벗어나지 못하고, 기술 격차는 계속 벌어진다.
- 모든 것은 '차세대'가 해결해야 줄 것이다. (정말?)

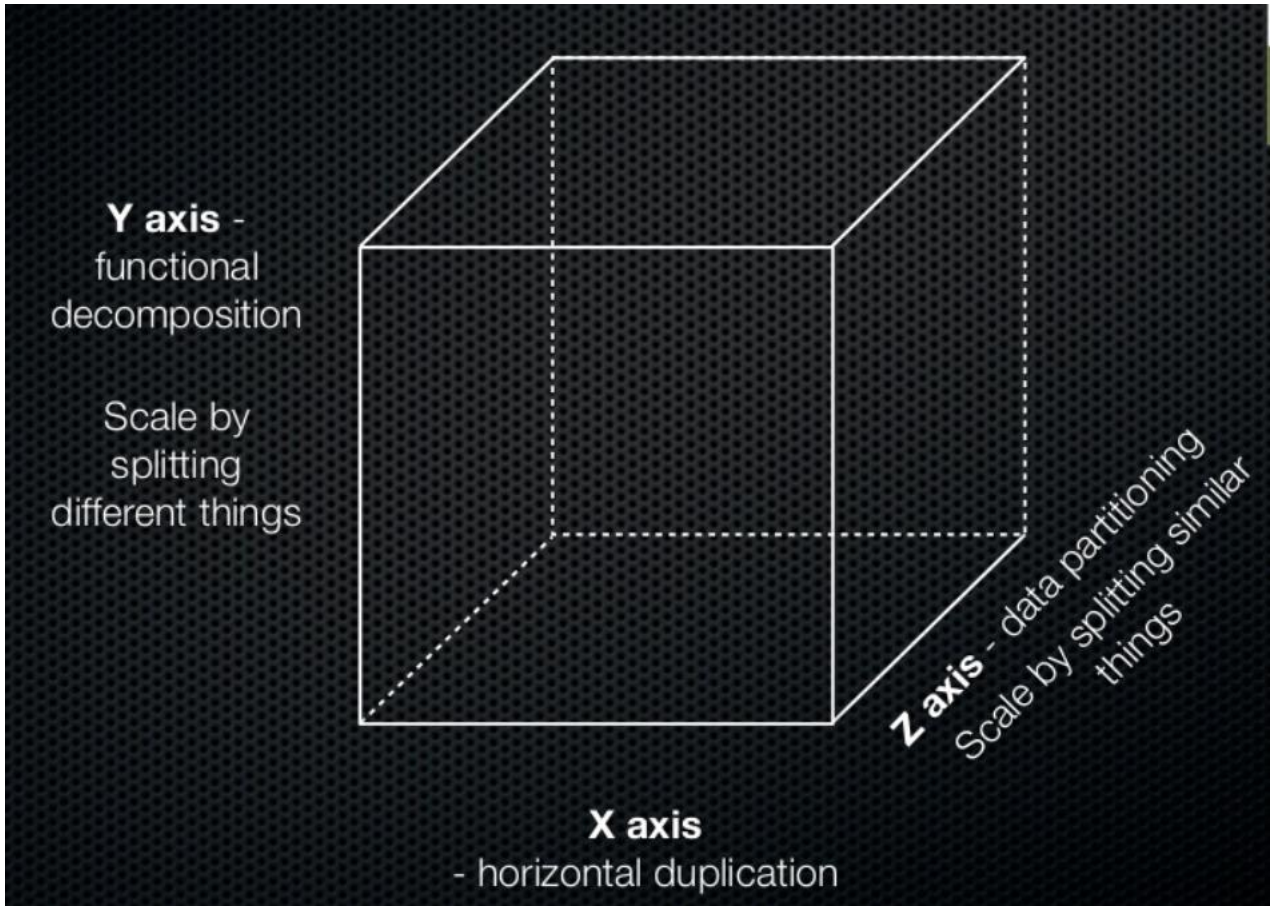
나랑 상관없는 상상 속 단어들

Domain Driven Design	Continuous Delivery	On-demand Virtualization
Elastic, Scalable, Resilience	Polyglot Programming	Infrastructure Automation
Agile Development	Reusability	Self-government Team

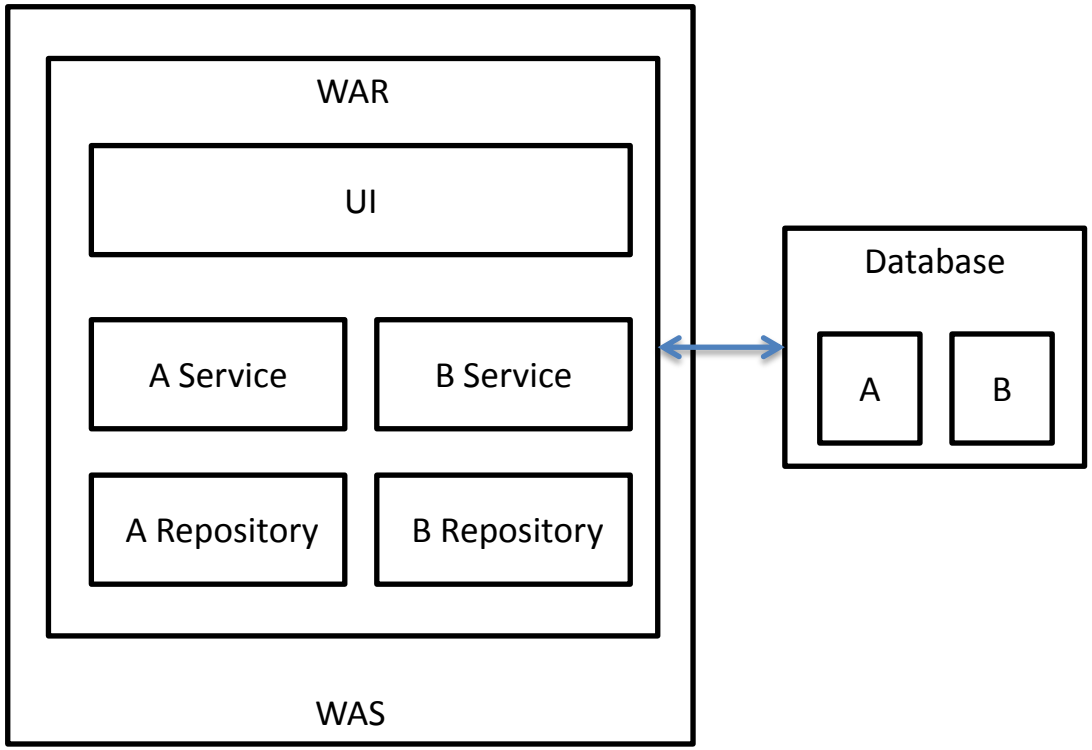
마이크로 서비스 아키텍처의 배경

- Domain Driven Design
- Continuous Delivery
- On-demand Virtualization
- Elastic, Scalable, Resilience
- Polyglot Programming
- Infrastructure Automation
- Agile Development
- Reusability
- Self-government Team

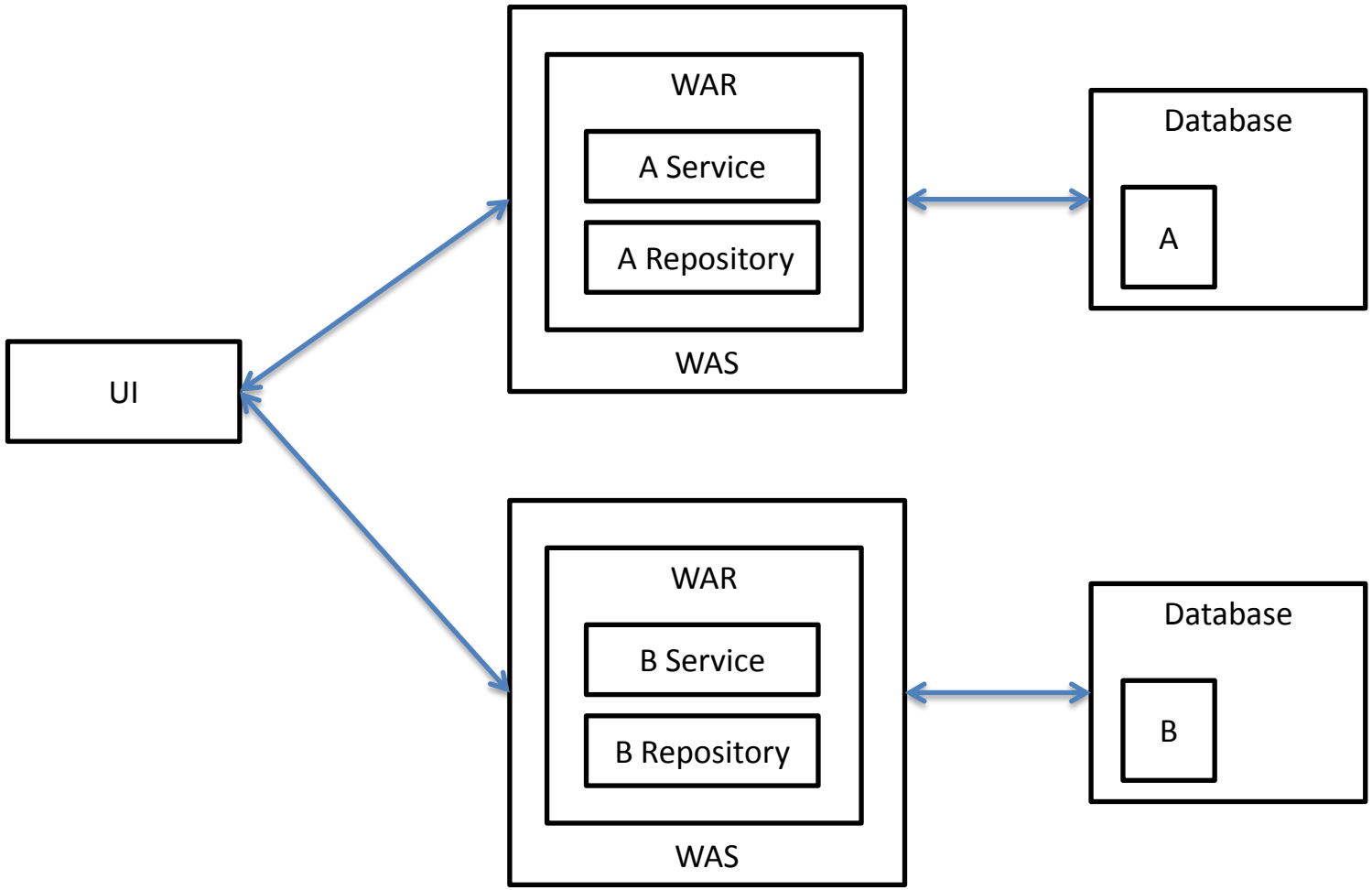
Scale Cube


Y axis -
functional
decomposition
Scale by
splitting
different things
Z axis - data partitioning
Scale by splitting similar
things
X axis
- horizontal duplication

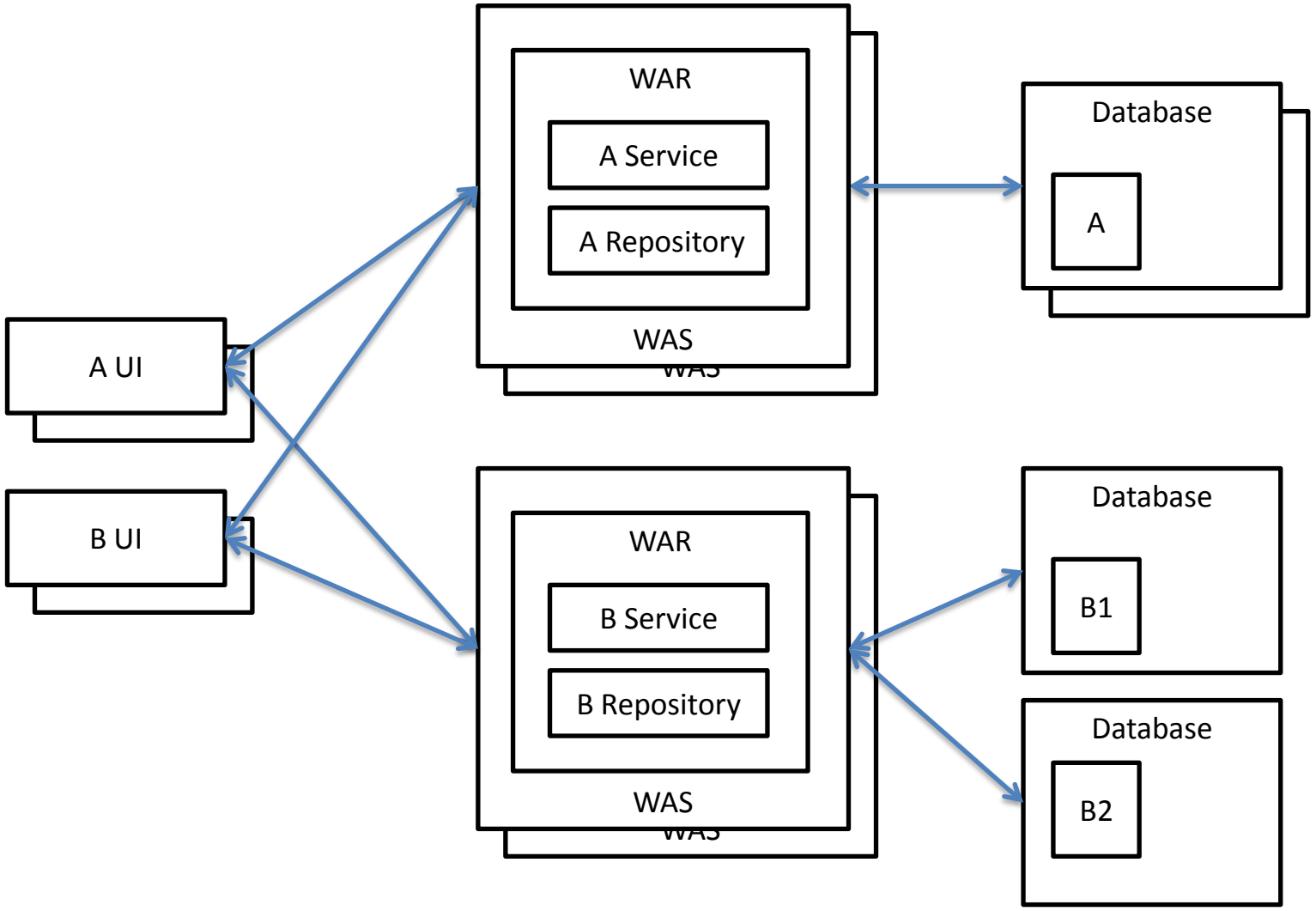
Y축 확장


WAR
UI
A Service
B Service
A Repository
B Repository
WAS
Database
A
B

Y축 확장


WAR
A Service
A Repository
WAS
Database
A
UI
WAR
B Service
B Repository
WAS
Database
B

Y축 확장 + X축/Z축 확장


WAR
A Service
A Repository
WAS
Database
A
A UI
B UI
WAR
B Service
B Repository
WAS
Database
B1
Database
B2

마이크로 서비스란?

- 작고(small)
- API로 다른 서비스와 연계하며(communicate with APIs)
- 자율적이며(autonomous)
- 한 가지 일을 잘하는데 초점을 맞춘 서비스(focused on doing one thing well)

장점

- Technology Heterogeneity
- Resilience
- Scaling
- Ease of Deployment
- Organizational Alignment
- Composability
- Replaceability

단점

- Complexity
- Multiple Database & Transaction Management
- Complicated Test
- Require Automation for Deploy/Operation
- Hard to develop features span multiple services

이거 SOA 얘기 아니에요?

- 비슷하지만, 달라요.
- SOA는 개념 상으로는 잘못되지 않았어요.
- 다만 방식이 잘못되었을 뿐이죠.
  - SOAP Protocol
  - WS-*
  - Vendor-Driven
  - ESB가 모든 걸 다 해결해줄 거라는 잘못된 믿음 (그렇게 선전했던 나쁜 XX)

MSA는

- ~~Vendor Driven~~ -> Service Company Driven
- 오픈테크놀로지 기반
- SOAP/XML vs. REST/JSON
- ‘스펙 먼저’가 아닌 ‘현실에서 검증된 Practice들’의 모음
- Agile 개념과의 결합
- Cloud 환경의 활용

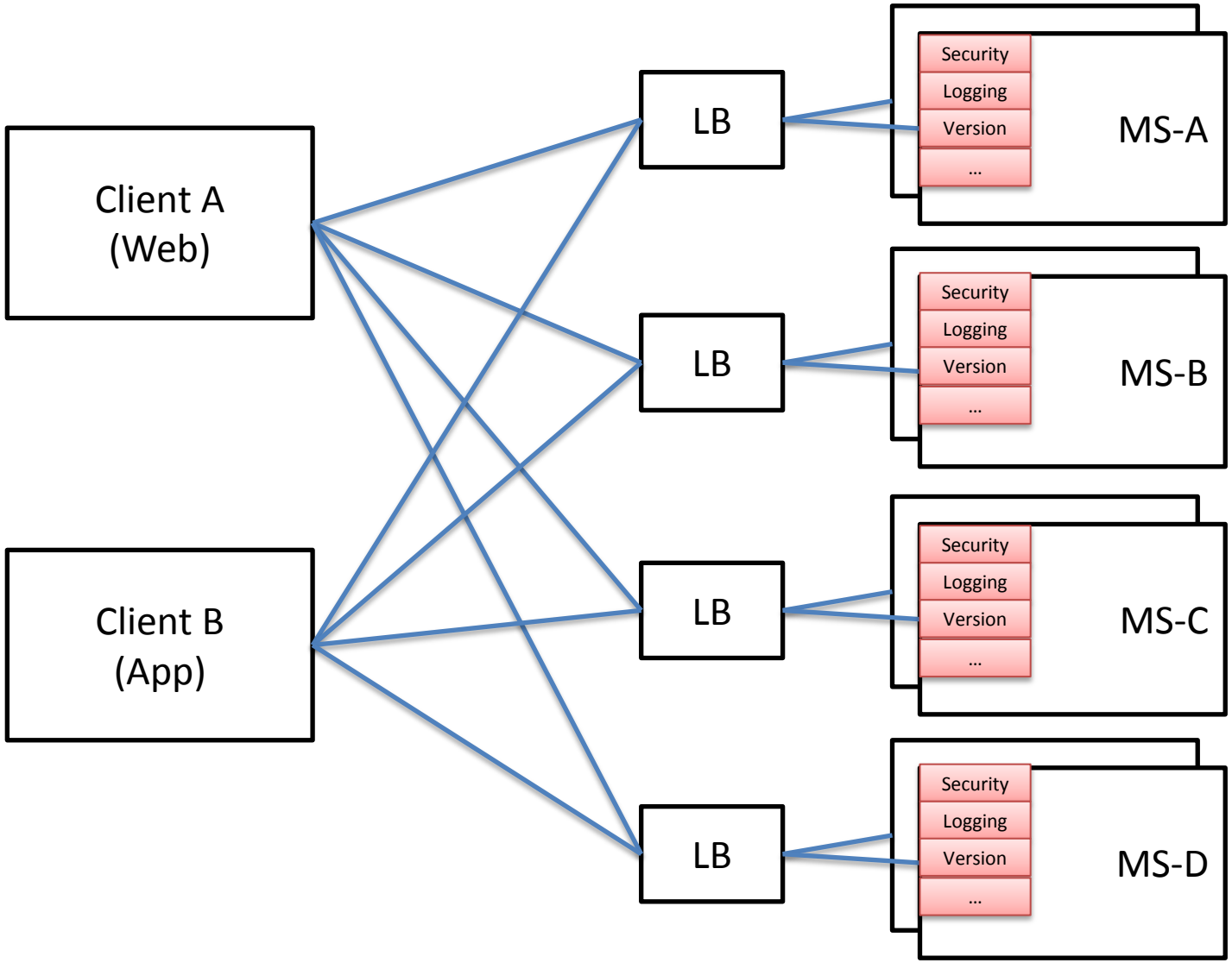
마이크로 서비스 모델링

- Domain Driven Design
- Bounded Context
- Contract-First(API-First) Design
- Decomposed database
- Event-Driven Architecture

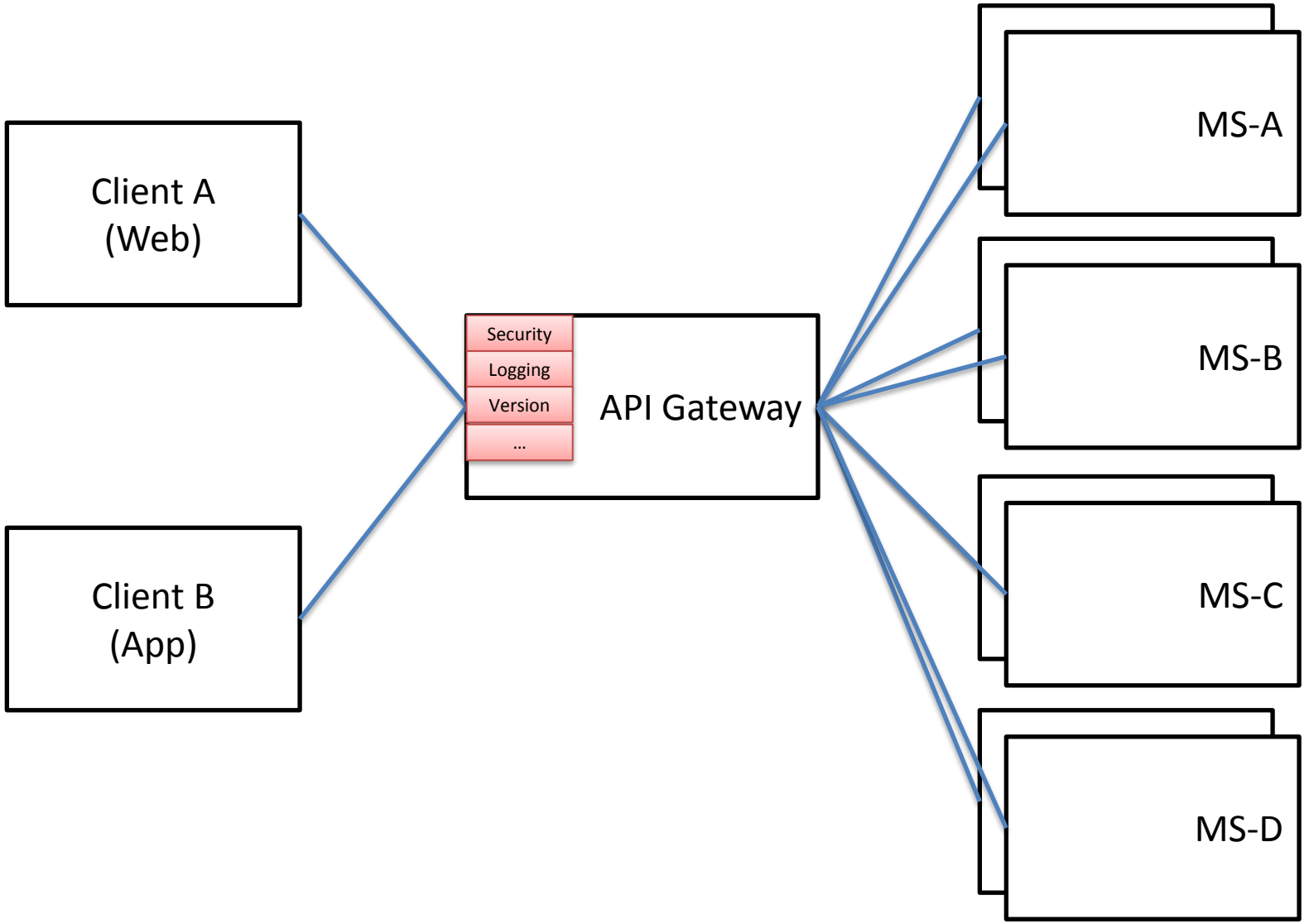
모델링/구현 Tip

- API를 먼저 정의하라.
- API를 REST API Maturity Level 2 이상이 되도록 강제화하라.
- API 문서를 유지하라(예: Swagger)
- ORM을 활용하라
- DB에 너무 의존하지 마라
- 도메인 내부에서만 의미있는 값을 외부에 노출하는 것을 지양하라
- 마이크로 서비스가 별다른 설정 없이 바로 기동가능하게 하라
  (예: Java의 경우, Spring Boot + Embedded WAS 활용)

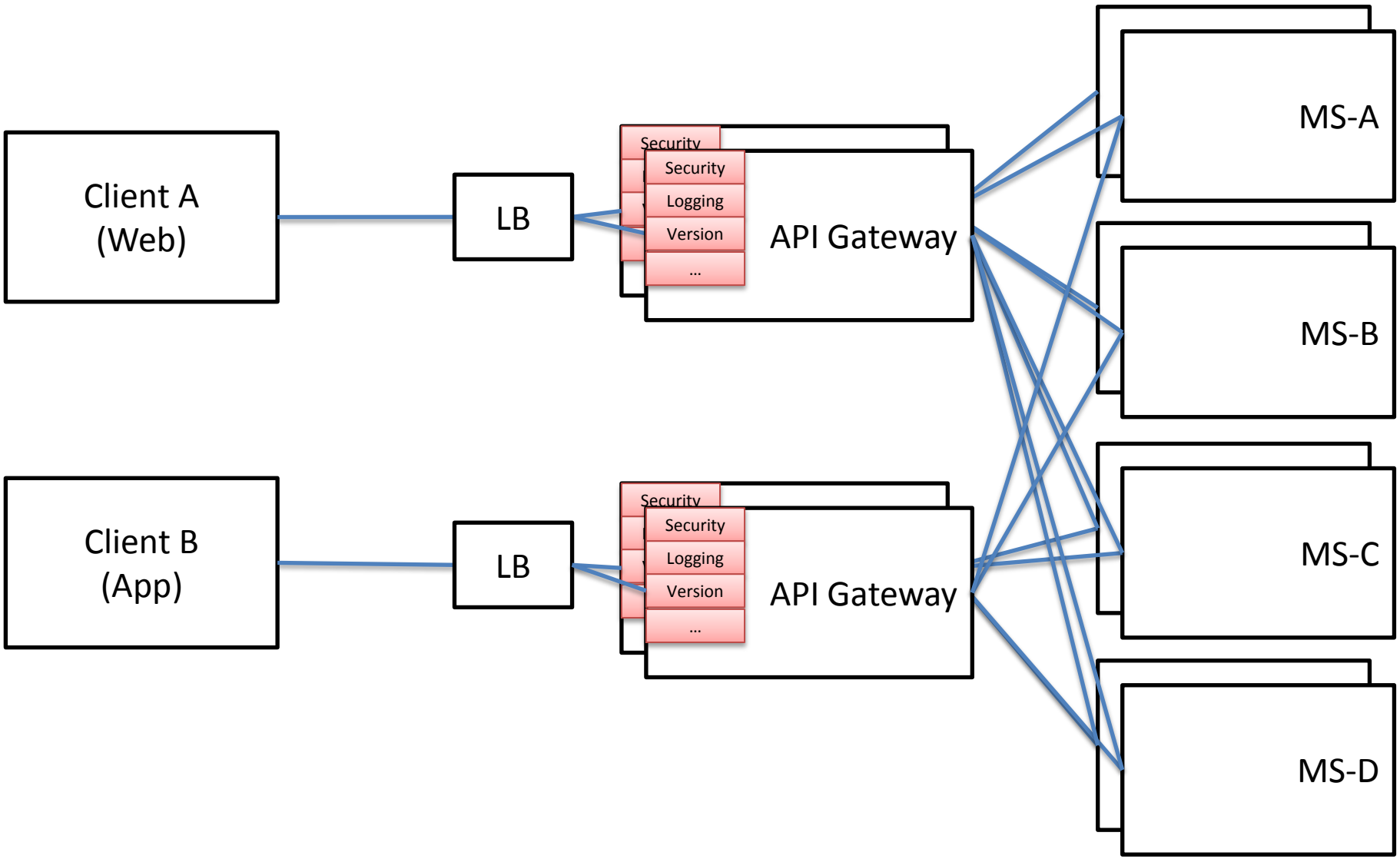
클라이언트-서비스 간 통합



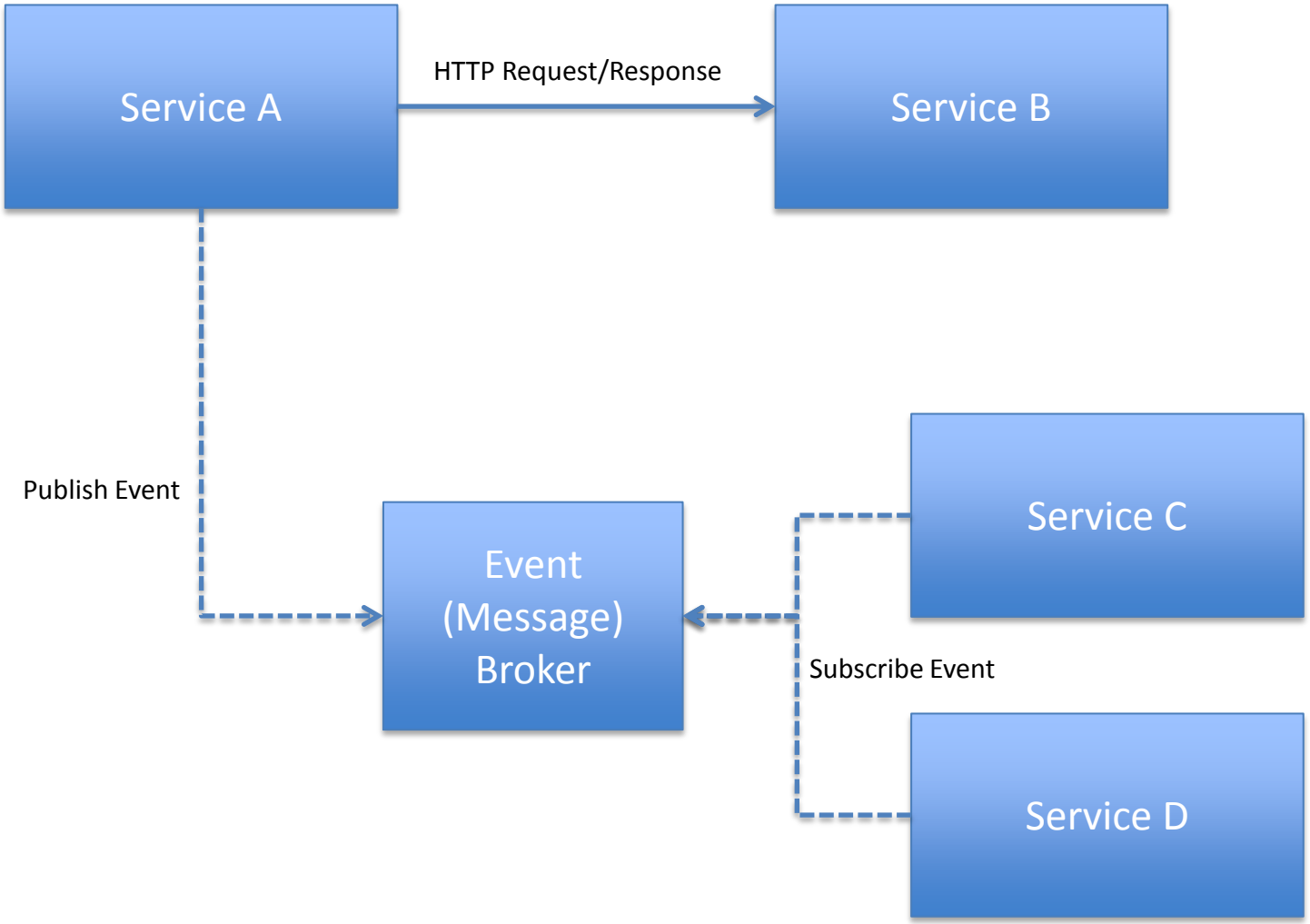
클라이언트-서비스 간 통합



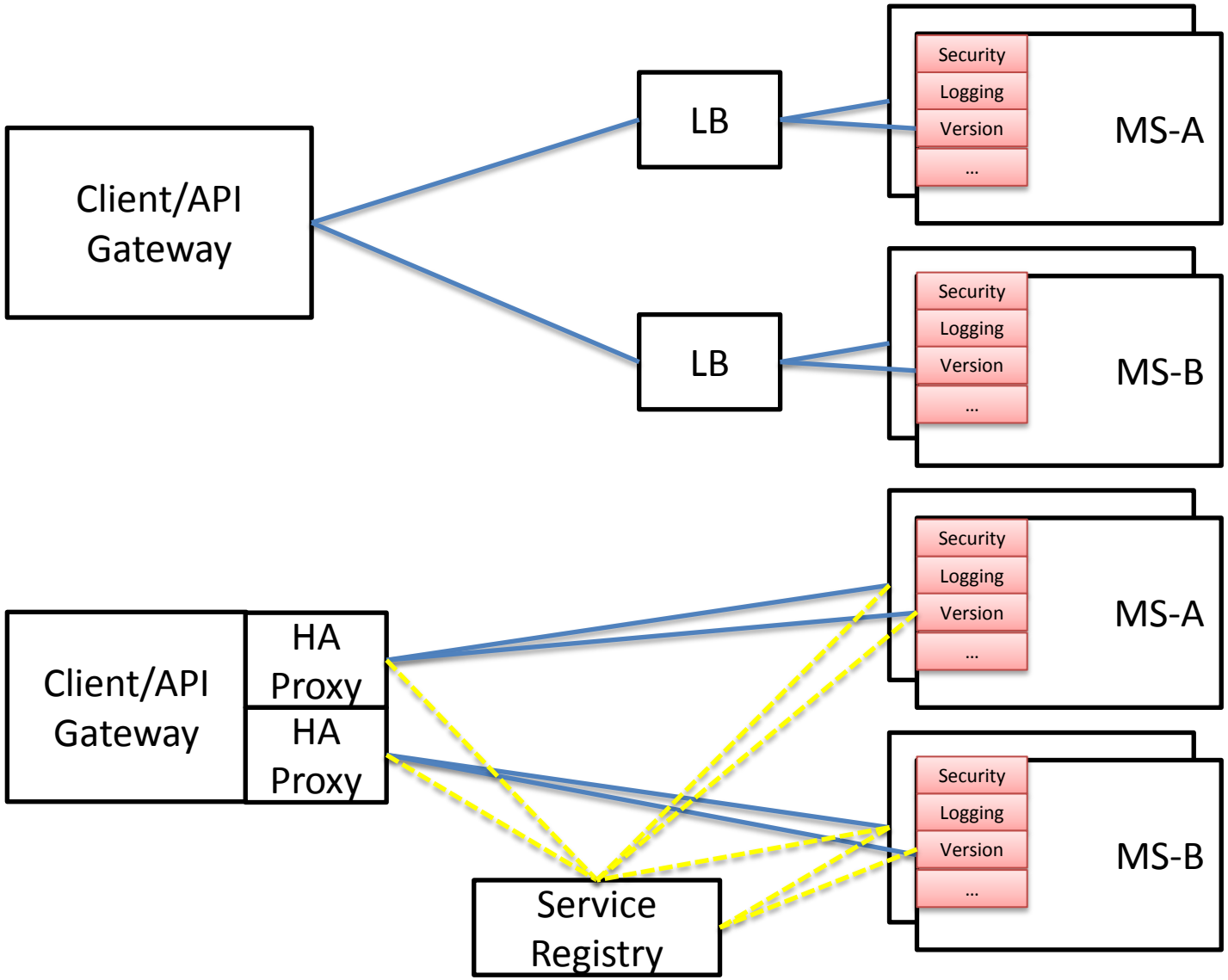
클라이언트-서비스 간 통합



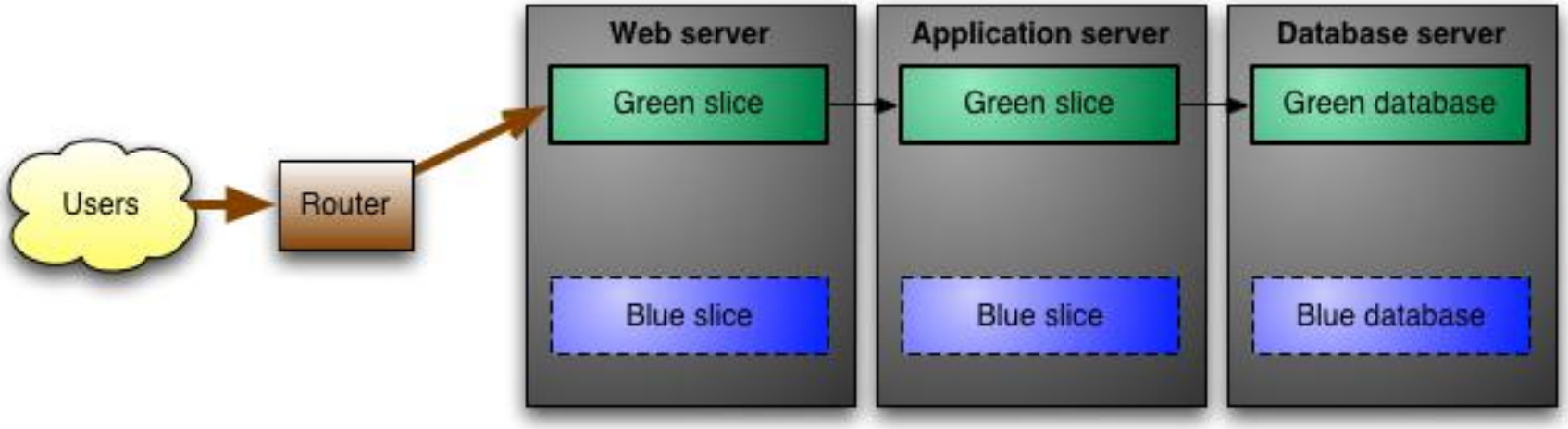
서비스 간 통신


Service A
HTTP Request/Response
Service B
Publish Event
Event
(Message)
Broker
Service C
Subscribe Event
Service D

Service Discovery

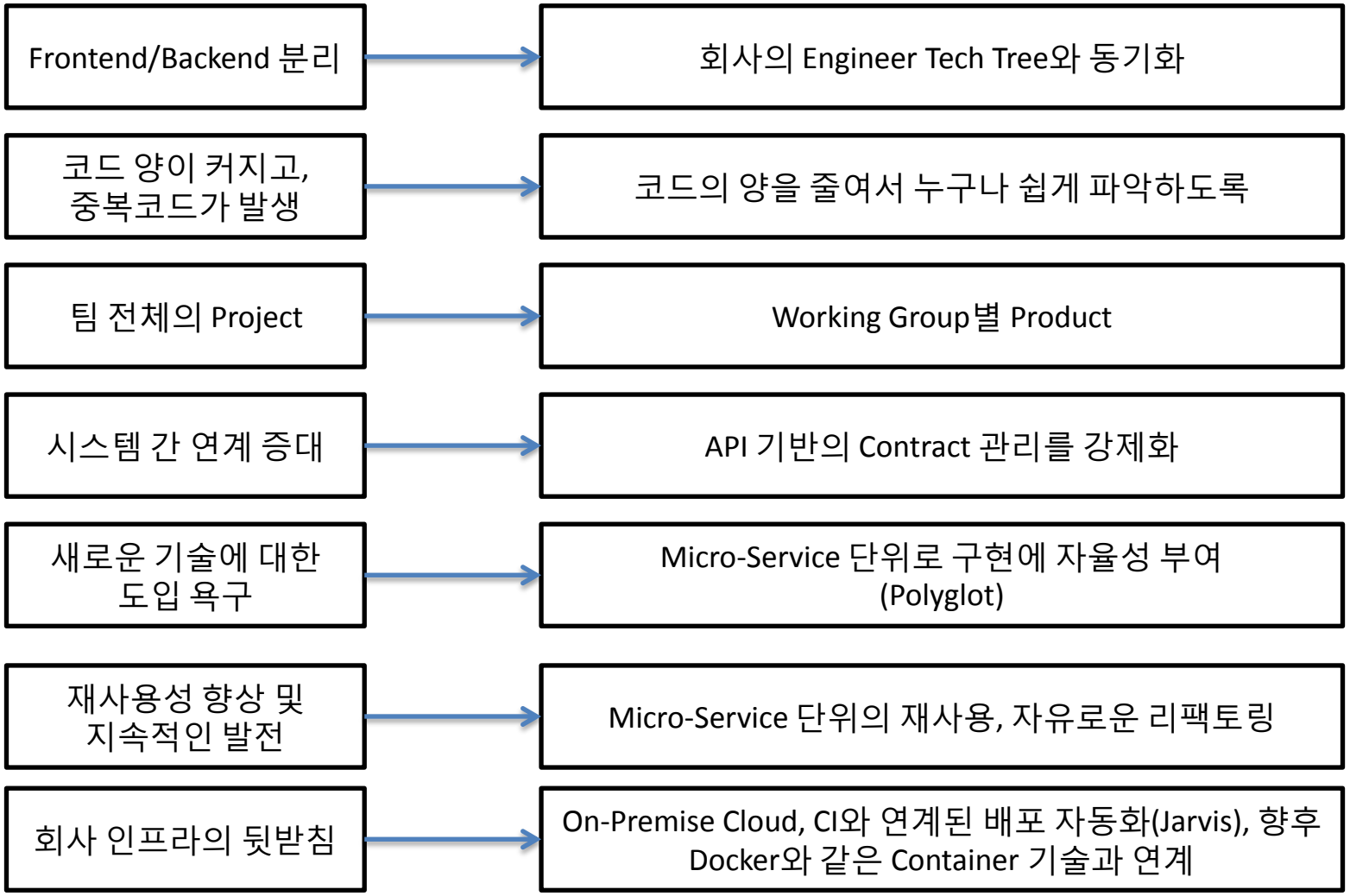


Blue/Green Deployment

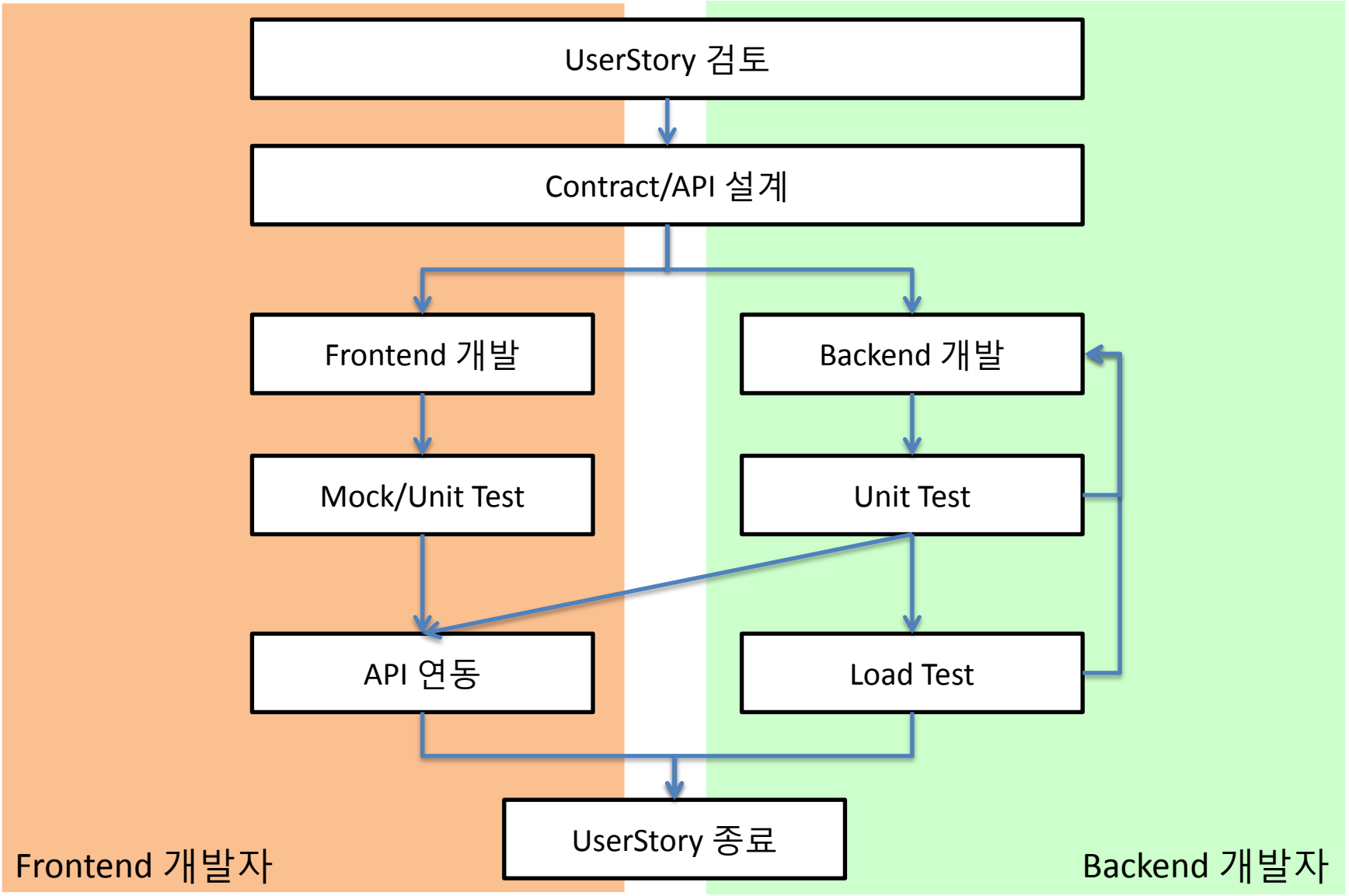


http://martinfowler.com/bliki/BlueGreenDeployment.html

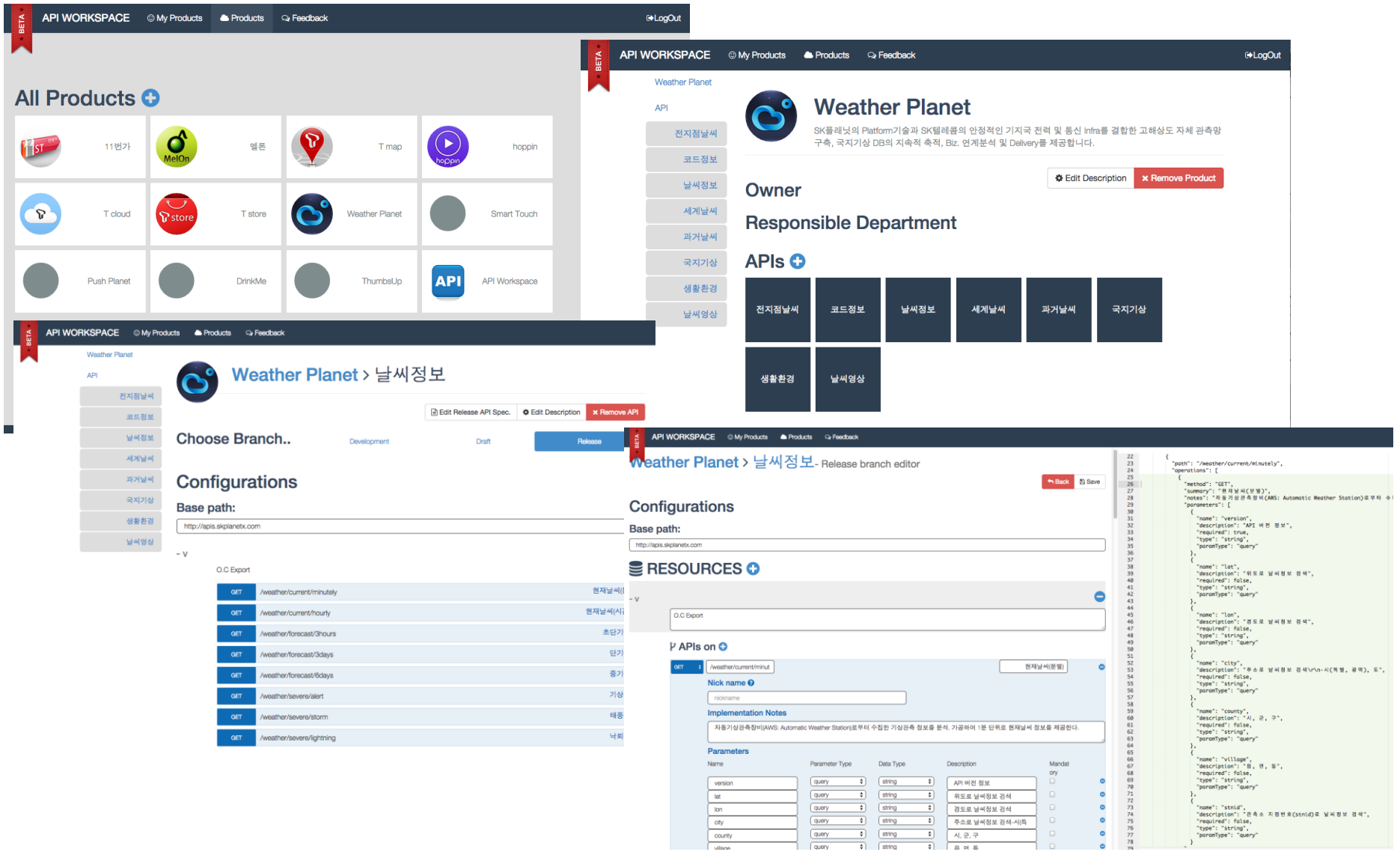
MSA를 선택한 이유


Frontend/Backend 분리
회사의 Engineer Tech Tree와 동기화
코드 양이 커지고,
중복코드가 발생
코드의 양을 줄여서 누구나 쉽게 파악하도록
팀 전체의 Project
Working Group별 Product
시스템 간 연계 증대
API 기반의 Contract 관리를 강제화
새로운 기술에 대한
도입 욕구
Micro-Service 단위로 구현에 자율성 부여
(Polyglot)
재사용성 향상 및
지속적인 발전
Micro-Service 단위의 재사용, 자유로운 리팩토링
회사 인프라의 뒷받침
On-Premise Cloud, CI와 연계된 배포 자동화(Jarvis), 향후
Docker와 같은 Container 기술과 연계

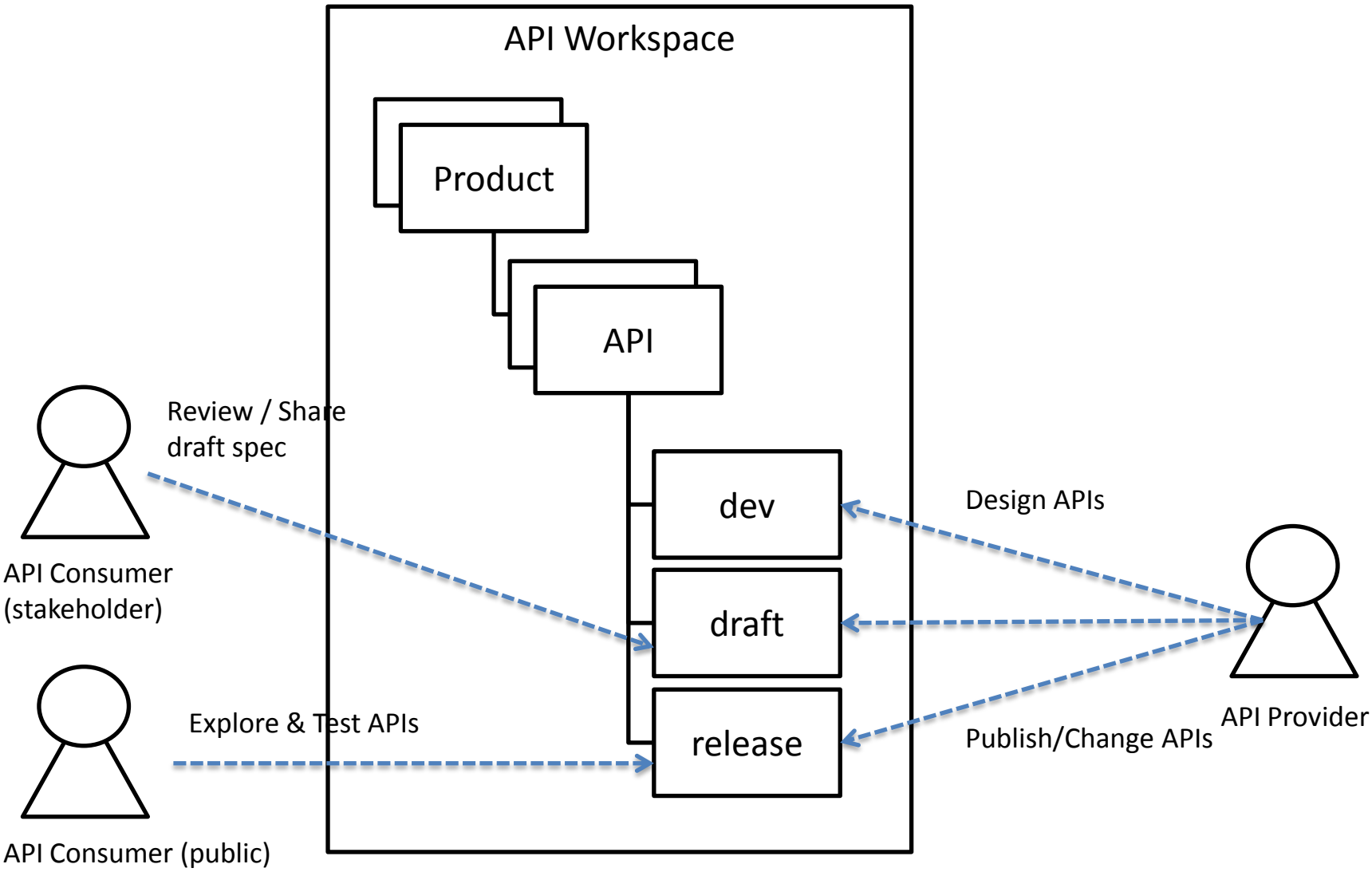
어떻게 개발하나요?


UserStory 검토
Contract/API 설계
Frontend 개발
Backend 개발
Mock/Unit Test
Unit Test
API 연동
Load Test
UserStory 종료
Frontend 개발자
Backend 개발자

Contract/API의 설계/공유

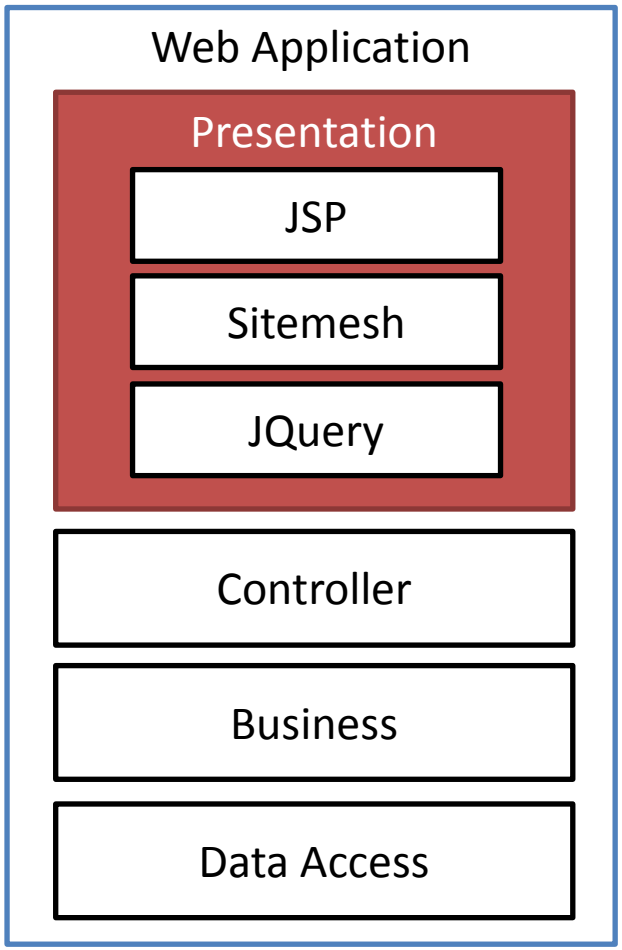


Contract/API의 설계/공유

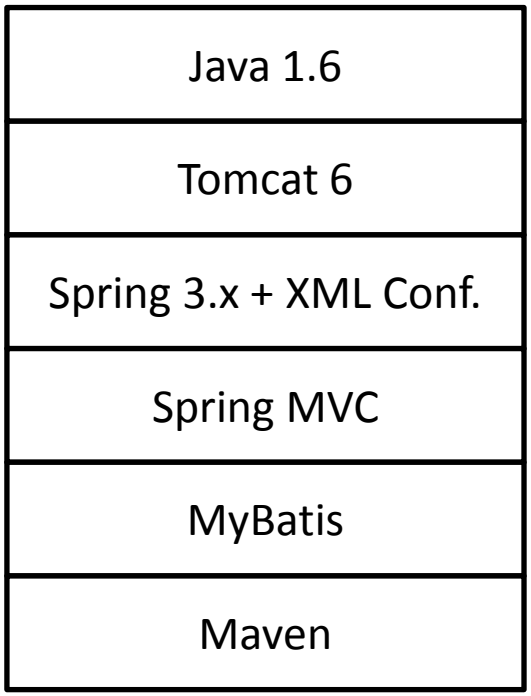


진짜 Polyglot을 하나요?

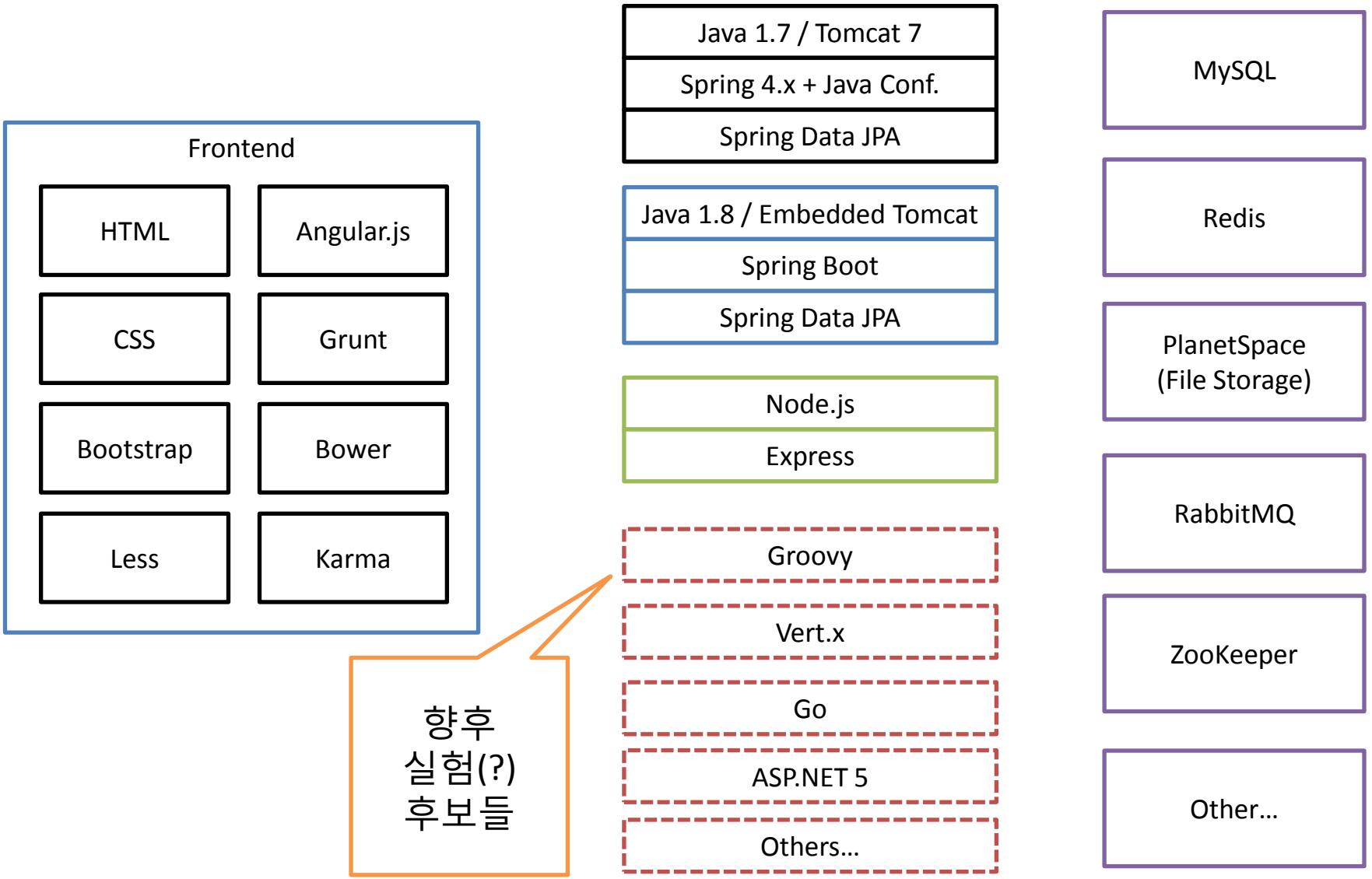
기존 시스템들의 Frontend



기존 시스템들의 Backend

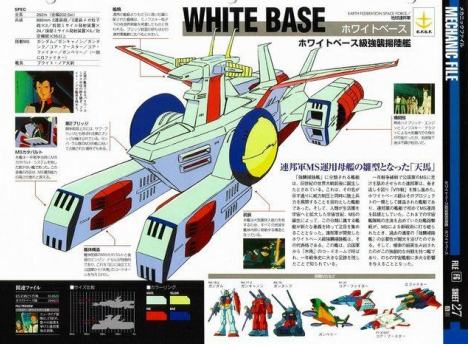


네, 합니다.

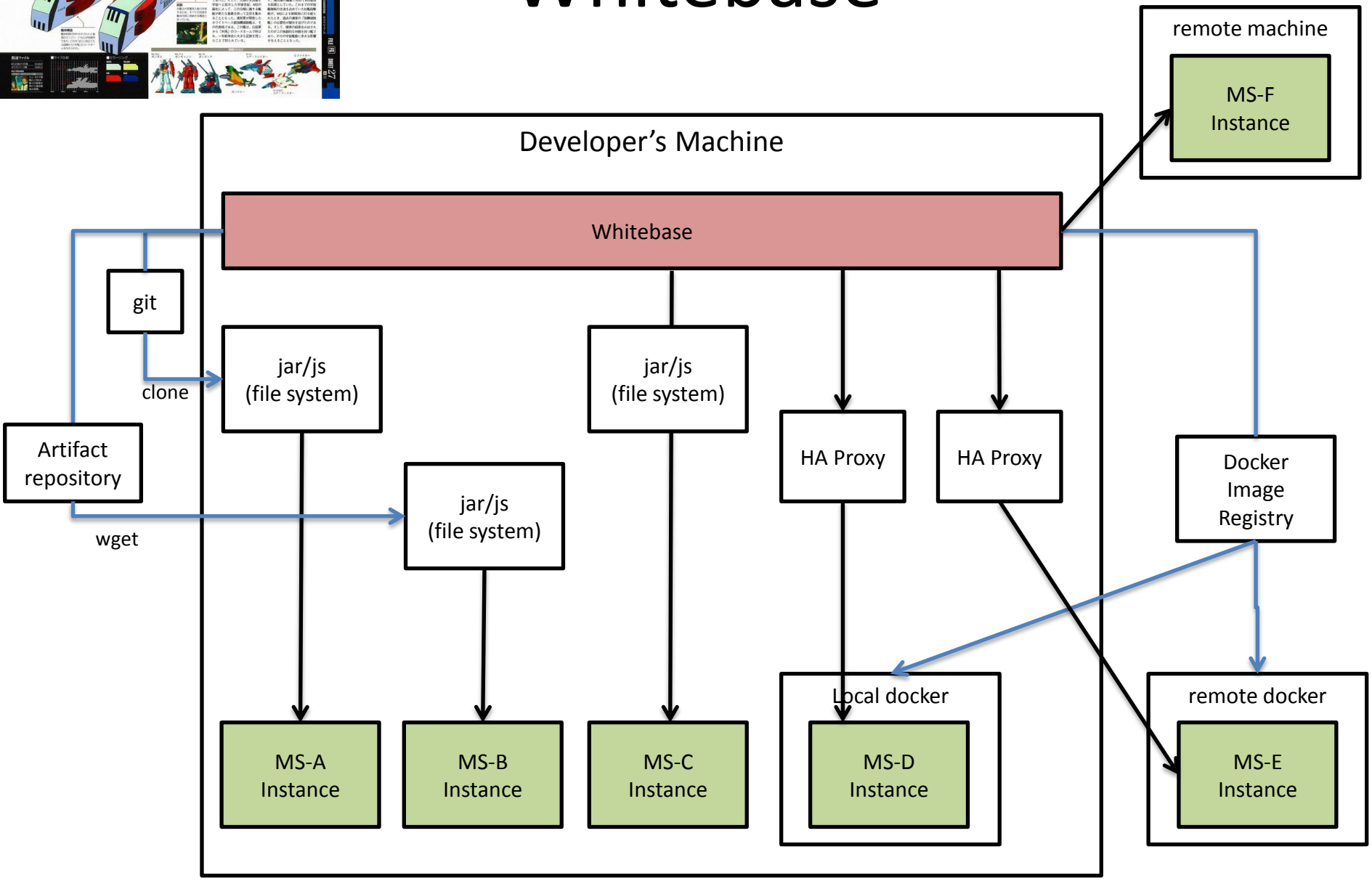

Frontend
HTML
Angular.js
CSS
Grunt
Bootstrap
Bower
Less
Karma
향후
실험(?)
후보들
Java 1.7 / Tomcat 7
Spring 4.x + Java Conf.
Spring Data JPA
Java 1.8 / Embedded Tomcat
Spring Boot
Spring Data JPA
Node.js
Express
Groovy
Vert.x
Go
ASP.NET 5
Others...
MySQL
Redis
PlanetSpace
(File Storage)
RabbitMQ
ZooKeeper
Other...

개발 환경의 문제

- 개발하다 보면, 여러 개의 마이크로서비스들을 구동시켜야 하는 경우가 많다.
- 마이크로서비스마다 설정/기동방식이 상이한 경우, 다소 괴롭다.
- 구동된 마이크로 서비스들을 위한 API Gateway를 빠르게 설정하기
- git clone/pull하는 것도 일이다.

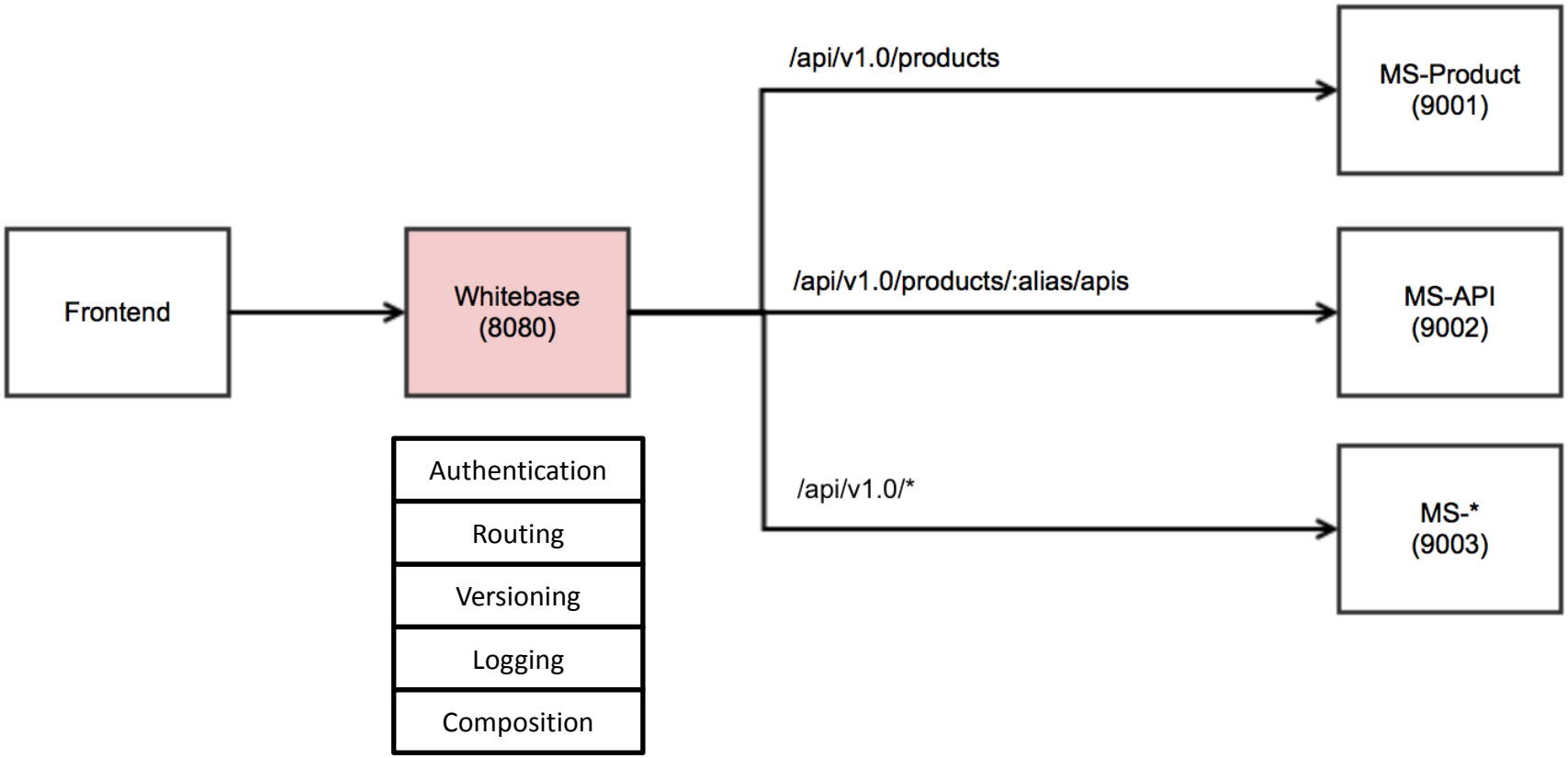

WHITE BASE
ホワイトベース
MECHANIC FILE

Whitebase


remote machine
MS-F Instance
Developer's Machine
Whitebase
git
clone
jar/js (file system)
jar/js (file system)
HA Proxy
HA Proxy
Artifact repository
wget
jar/js (file system)
Docker Image Registry
Local docker
remote docker
MS-A Instance
MS-B Instance
MS-C Instance
MS-D Instance
MS-E Instance

Whitebase

- URL-based Router + Proxy



MSA로 진행하며 얻은 것

- 팀원들이 다루는 기술 범위가 크게 확장
- 최신 기술의 도입/실험에 부담이 없음
- 팀원 간의 경험 공유 활성화
- API 디자인 능력 향상
- 향후 변화/확장에 대해서도 아키텍처적 변화 없이 대응이 가능
- 높은 재활용성/조합 지원
- 향후 사내/대외 오픈소스화가 가능할 듯한 솔루션들

MSA로 진행 시의 단점

- 초기 개발 시에는 개발 시간이 많이 소요됨
- MSA로 원활하게 개발하기 위한 기반 구성요소와 인프라 준비가 필요함
- 기존 개발 방식이 편하다고 주장하는 사람들로부터의 저항
- 서비스 간 연계 시 협의가 필요
- 여러 서비스에 걸치는 기능의 경우, 주체가 애매한 경우가 생김

제언

- 갑자기 한번에 MSA로 넘어가는 것은 쉽지 않은 작업입니다.
- 기존 시스템에서 분할이 가능한 항목을 하나씩 분할해가거나, 신규 개발되는 기능을 하나씩 마이크로 서비스로 분리하여 추가해가는 것이 좋습니다.
- MSA는 만능이 아니며, 조직 문화(점진적 발전, 구성원들의 Skill-up에 대한 관심과 투자)가 뒷받침해줘야 합니다. 조직 문화가 선행되지 않으면 차라리 시도하지 않는 것이 좋습니다.

질문/피드백

```
var you = {};

if (you.like||you.dislike||you.suggest||you.request)
{
  var id = "jaewoo.ahn";
  var domain = "sk.com";

  mail.send(String.format("{0}@{1}", id, domain ));
}
```