

표준프레임워크를 활용한 MSA 구현



Contents

I MSA 소개

II 마이크로서비스 개발

III MSA 배포 및 운영



Contents



MSA 소개

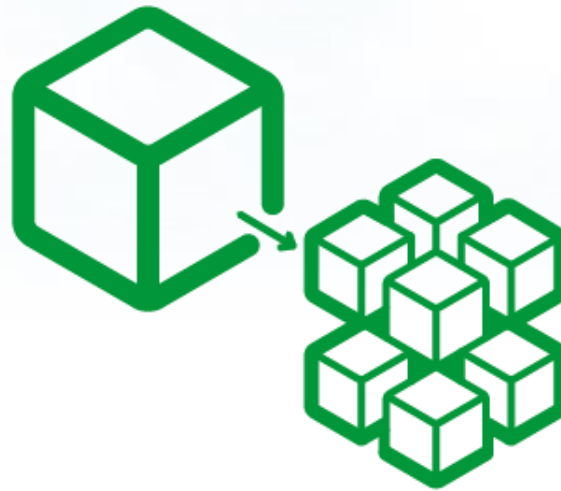
1. MSA 란 무엇인가?
2. MSA 장점
3. MSA 단점 및 도입 시 고려사항



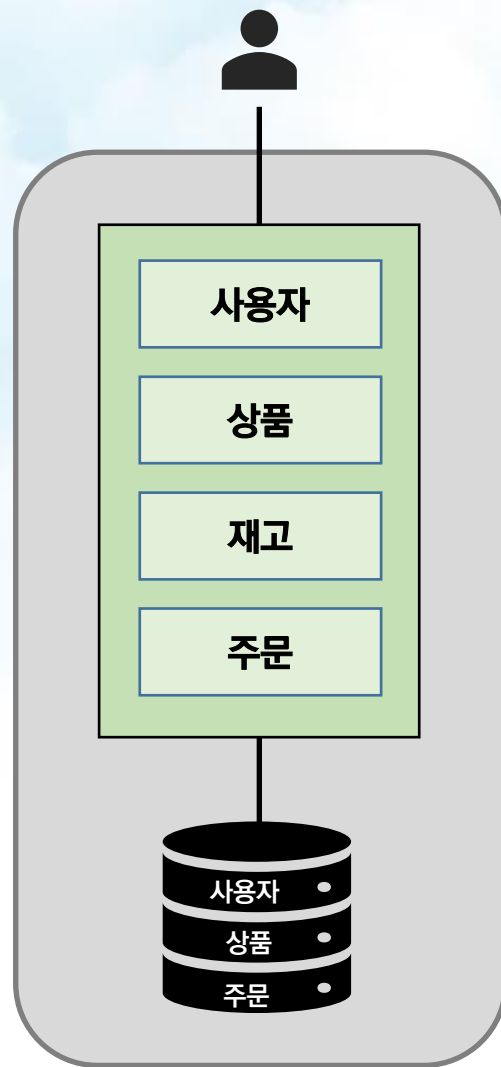
MSA 란 무엇인가?

- MSA (Microservice Architecture) 정의

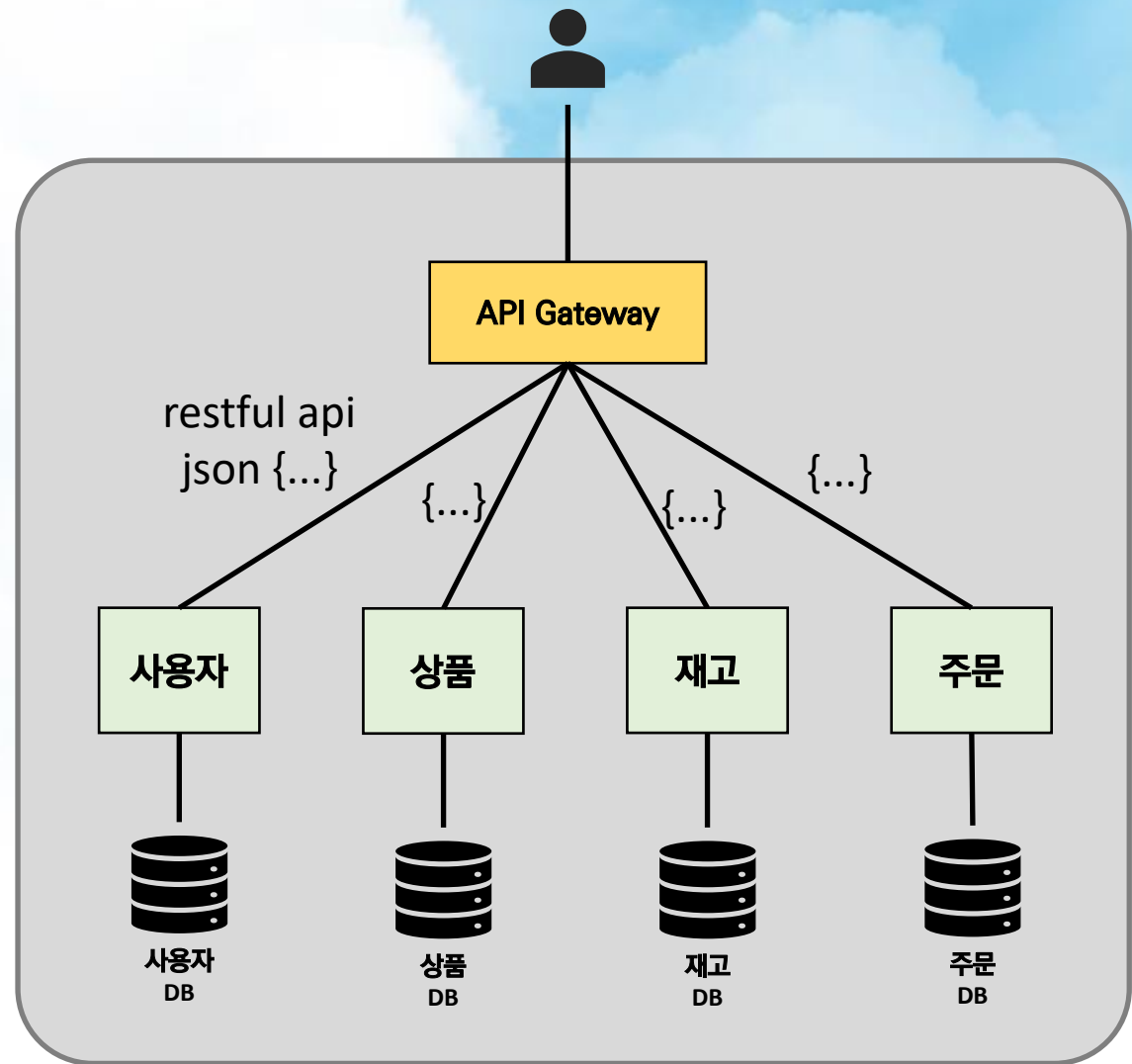
하나의 어플리케이션을 여러 개의 서비스로 나누어 이 서비스들을 조합하여 본래의 목적인 서비스를 제공하는 것



Monolith vs MSA 비교



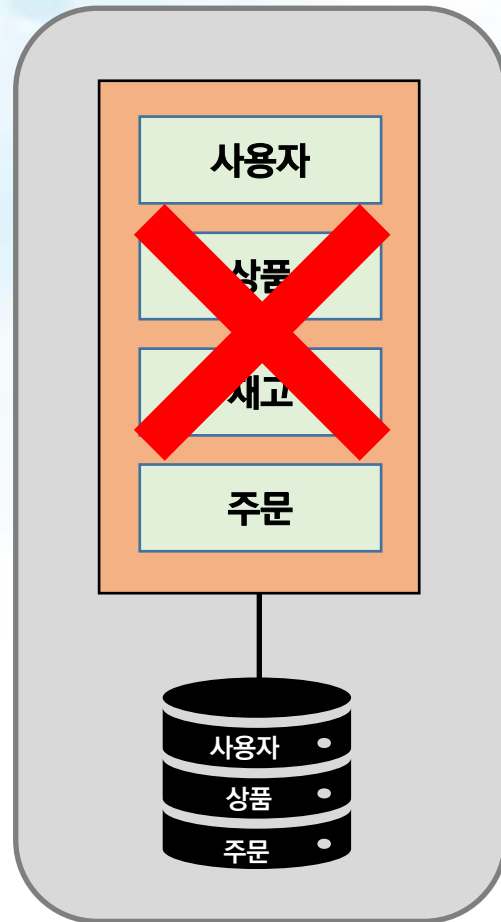
모놀리스
아키텍처



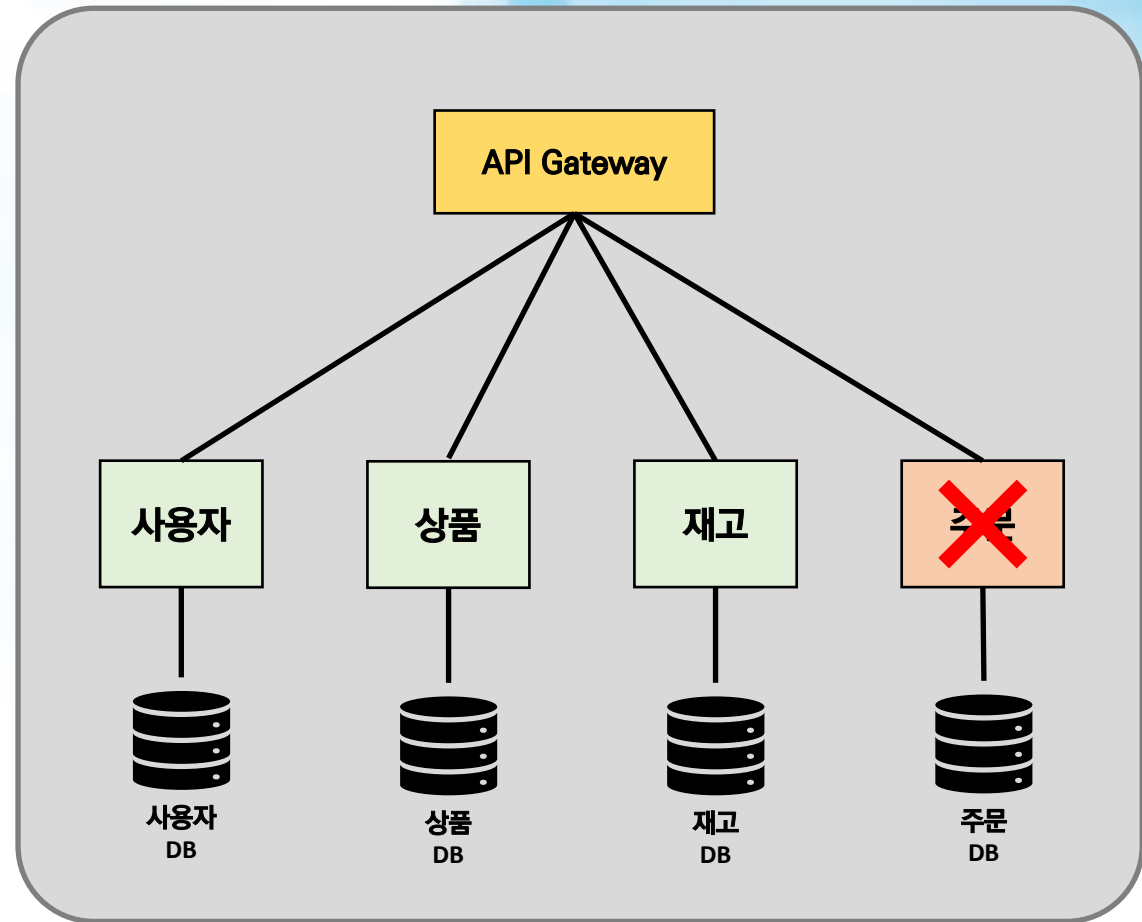
마이크로서비스
아키텍처

MSA 장점 1

- 장애 격리와 복구가 쉽다.



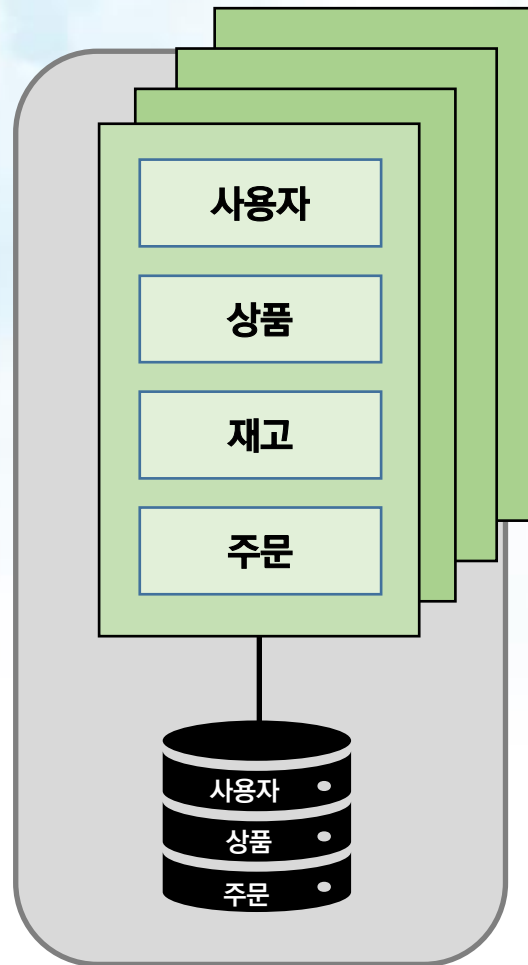
모놀리스
아키텍처



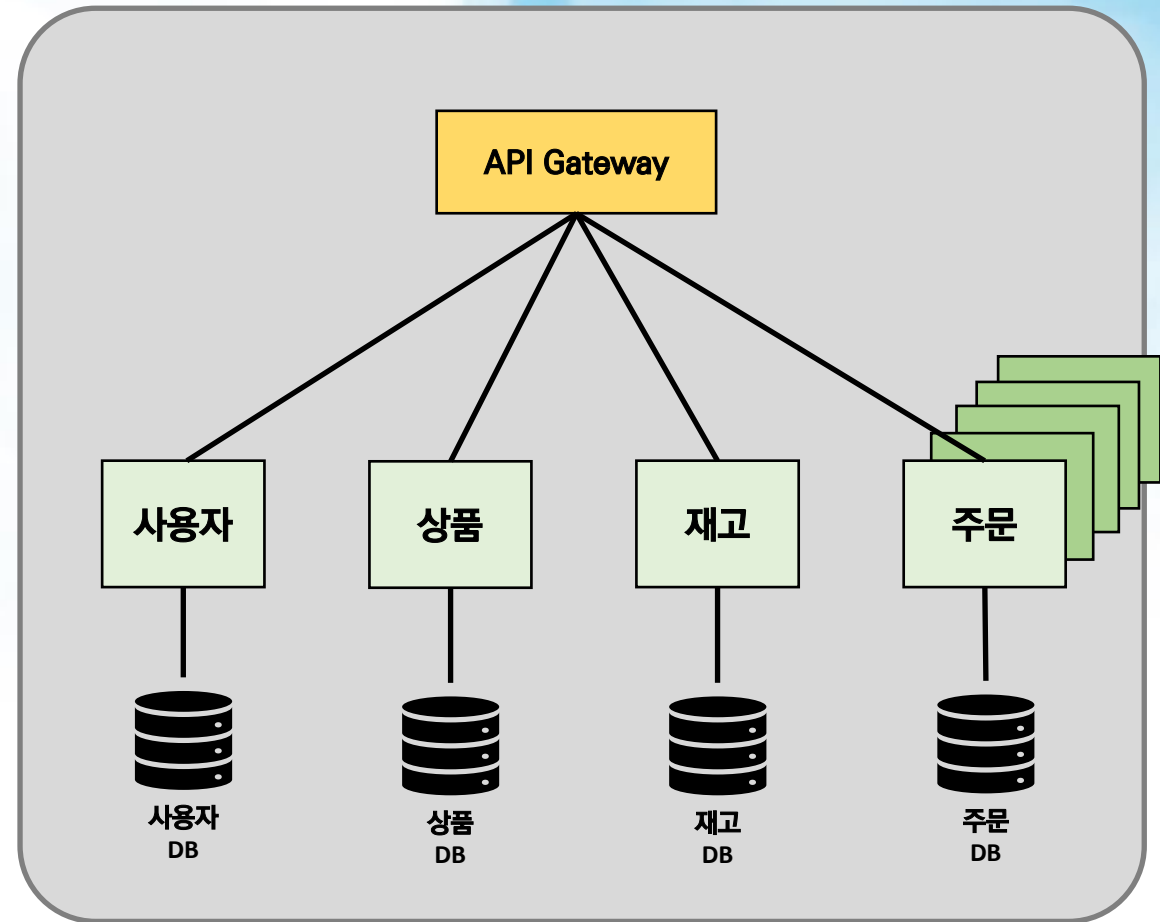
마이크로서비스
아키텍처

MSA 장점 2

- 비용 효율적으로 확장 가능



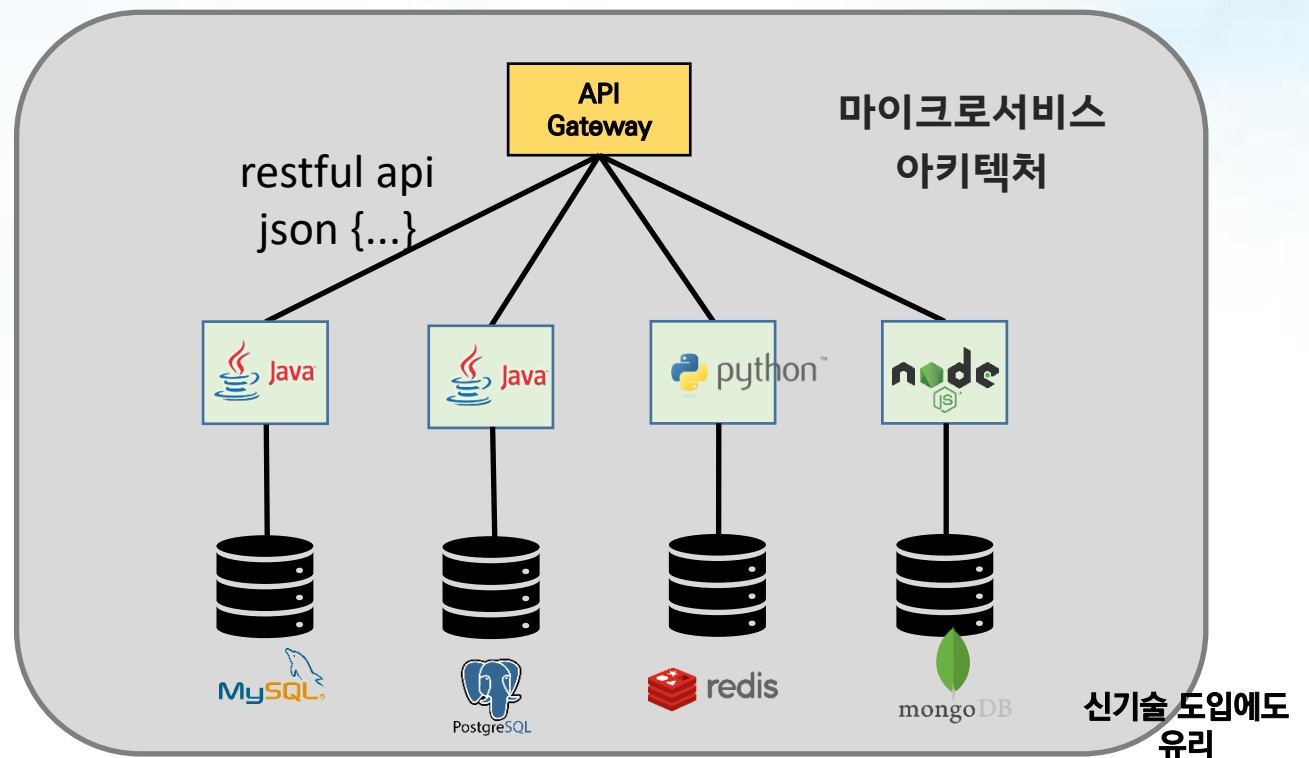
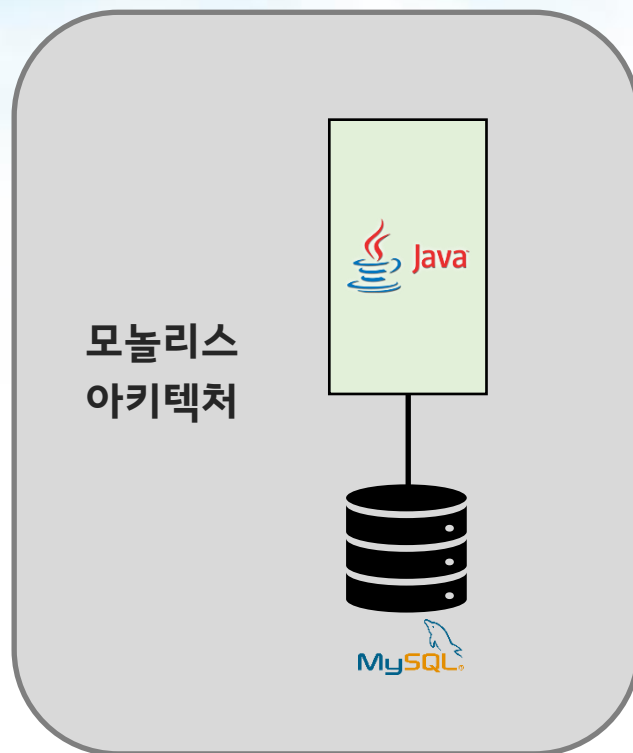
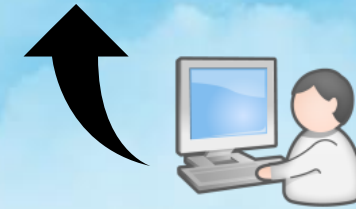
모놀리스
아키텍처



마이크로서비스
아키텍처

MSA 그외 장점

- 코드 규모가 작아 유지보수 용이 (생산성 향상)
- 빠른 배포와 실행가능 (서비스 개선 속도가 증가)
- 서비스 별 최적화된 개발 언어와 DB 선택 사용 가능 (Polyglot)



MSA 단점 및 도입 시 고려사항

■ MSA에 적합한 서비스인지 고려해볼만한 사항들

1. 빠르고 잦은 배포가 필요한가?

MSA의 근본적인 목적은 빠른 비즈니스 대응, 민첩성에 있음.
얼마나 사용자의 요구사항을 빠르게 반영하고 개선해 나갈 수 있을지에 대해 초점이 맞추어져 있음.
자주 개선해야 되거나 배포해야 되는 시스템(서비스)이 아닌 경우는 MSA가 적합하지 않을 수 있음.

2. 성능이 중요한 서비스인가?

분리된 서비스들은 물리적인 네트워크 통신을 하게 되므로 네트워크 대기시간(latency)이 발생할 수 있음.
모놀리스에 비해 성능이 저하될 수 있으므로 성능에 민감한 시스템(서비스)인 경우 도입을 고려해야 함.

3. 분산트랜잭션이 필요한 서비스인가?

모놀리스에서 여러개의 서비스들을 묶어서 단일트랜잭션으로 처리할 수 있었다면, MSA에서는 단일 트랜잭션 처리가 어려움.
이벤트 기반의 트랜잭션을 구현하여 데이터의 정합성을 유지할 수 있음.

4. 모놀리스에 비해 비용을 절감할 수 있는가? (비용 대비 이득을 따져볼것)

서비스들이 늘어난다는것은 관리할 서버가 늘어난다는것을 의미함. 그러므로 운영관리비용이 증가할 수 있음.
하지만 MSA가 주는 장점의 이득이 더 크다고 판단될 경우 MSA를 도입하는것이 좋은 선택이 될 수 있음.

5. 조직의 개발문화는 준비되어 있는가?

단순히 아키텍처만 도입한다고 해서 MSA가 구현되는것은 아니며,
MSA에 적합한 조직의 개발문화(devOps)를 정착시킬 수 있도록 노력해야함.

Contents



마이크로서비스 개발

1. 12 Factors Rule
2. Spring Boot
3. Service Mesh
4. Spring Cloud



12 Factors Rule

- 소프트웨어를 서비스 형태로 제공하는 SaaS 앱을 만들기 위한 방법론
- 클라우드 네이티브 한 어플리케이션을 개발하기 위한 규칙 (12가지)

코드베이스

종속성

설정

백엔드
서비스

빌드, 릴리즈,
실행

프로세스

포트 바인딩

동시성

폐기 가능

개발/운영
환경 일치

로그

Admin
프로세스

- 참조 문서 URL : <https://12factor.net/ko/>

Spring Boot



+

표준프레임워크
eGovFrame

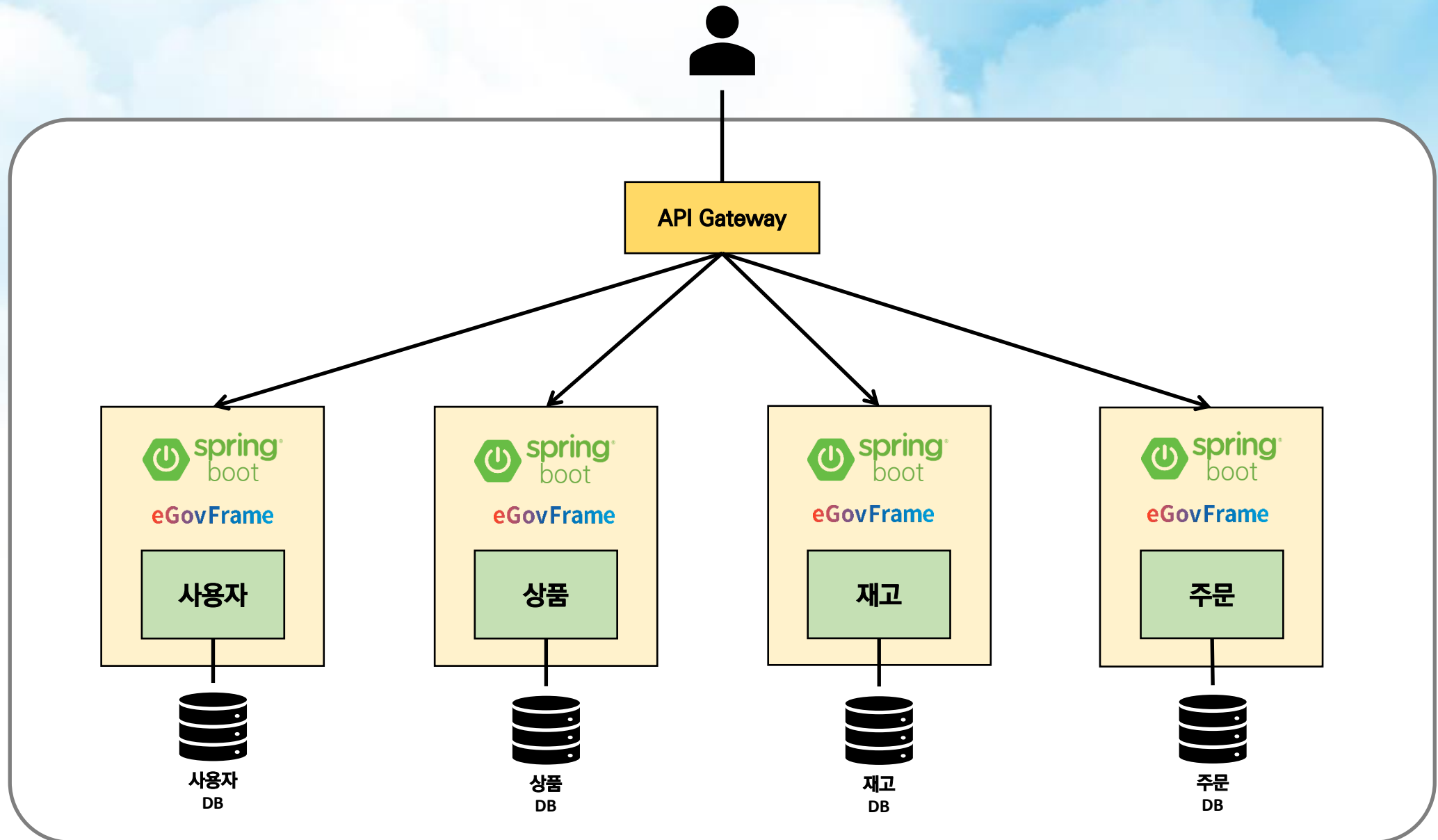


- spring core
- spring mvc
- spring security
- spring data
- spring batch

·
·

- 표준프레임워크는 Spring Framework 기반 (친숙)
- 어플리케이션을 쉽고 빠르게 실행해주는 도구
- 서블릿 컨테이너가 내장되어, 별도의 WAS 없이 단독 실행 가능
- 관례적으로 자주 쓰이는 설정들을 자동으로 설정
 - 설정간소화
- 다양한 스타터가 지원되어 의존성 관리가 간편
 - spring-boot-starter
 - spring-boot-starter-web
 - spring-boot-starter-logging
 -
 -
- 컨테이너(가상화)화해서 실행하기에 적합

Spring Boot + 표준프레임워크를 활용하여 MSA 구축



Spring Boot 적용방법

1. Spring Boot Starter 와 표준프레임워크 실행환경 dependency 추가

```

<!-- Spring Boot Starter -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<!-- 표준프레임워크 실행환경 -->
<dependency>
  <groupId>org.egovframe.rte</groupId>
  <artifactId>org.egovframe.rte.fdl.cmmn</artifactId>
  <version>${org.egovframe.rte.version}</version>
</dependency>
<dependency>
  <groupId>org.egovframe.rte</groupId>
  <artifactId>org.egovframe.rte.ptl.mvc</artifactId>
  <version>${org.egovframe.rte.version}</version>
</dependency>

```



eGovFrame

2. @SpringBootApplication 어노테이션 추가

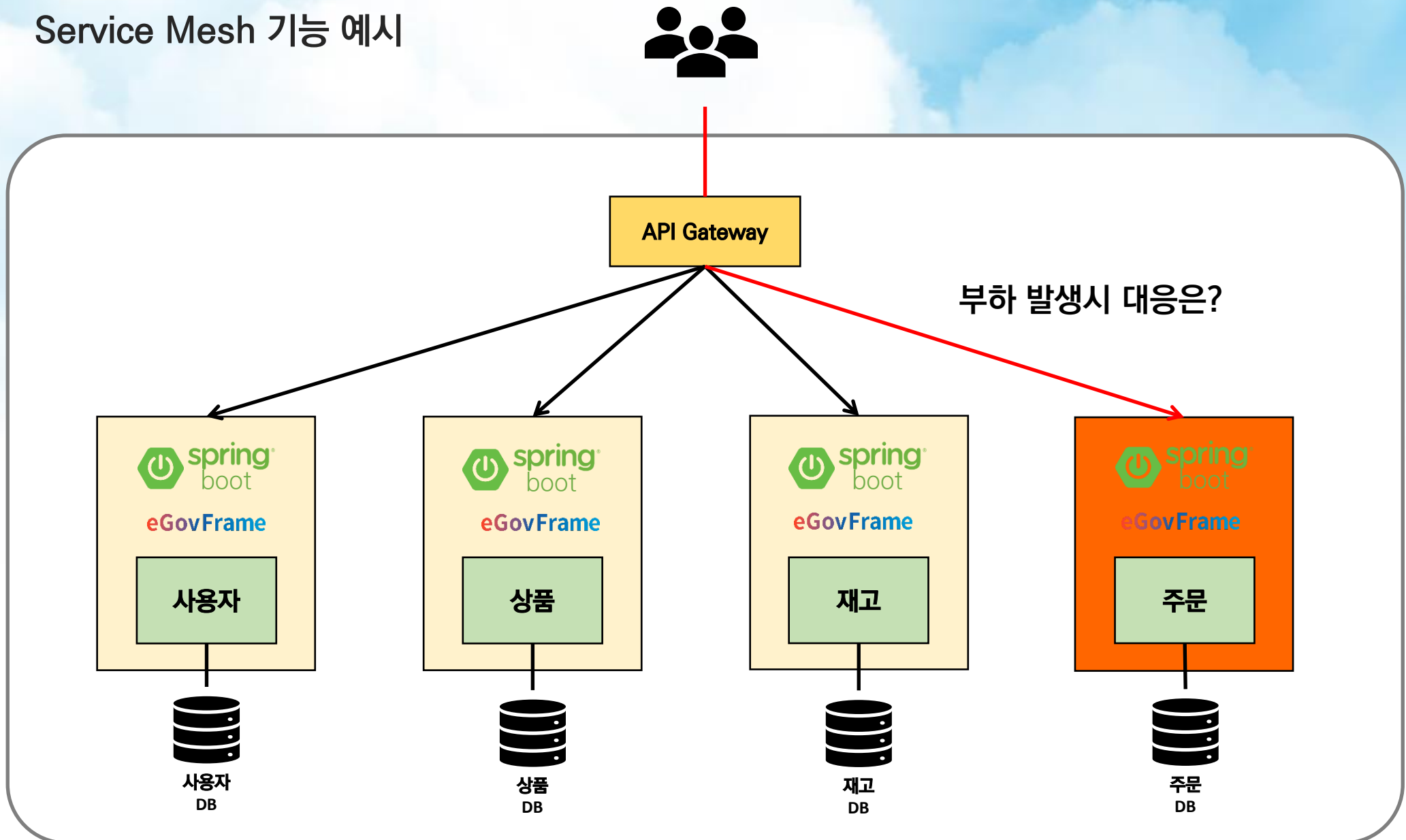
```

@SpringBootApplication
public class App {
    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
}

```

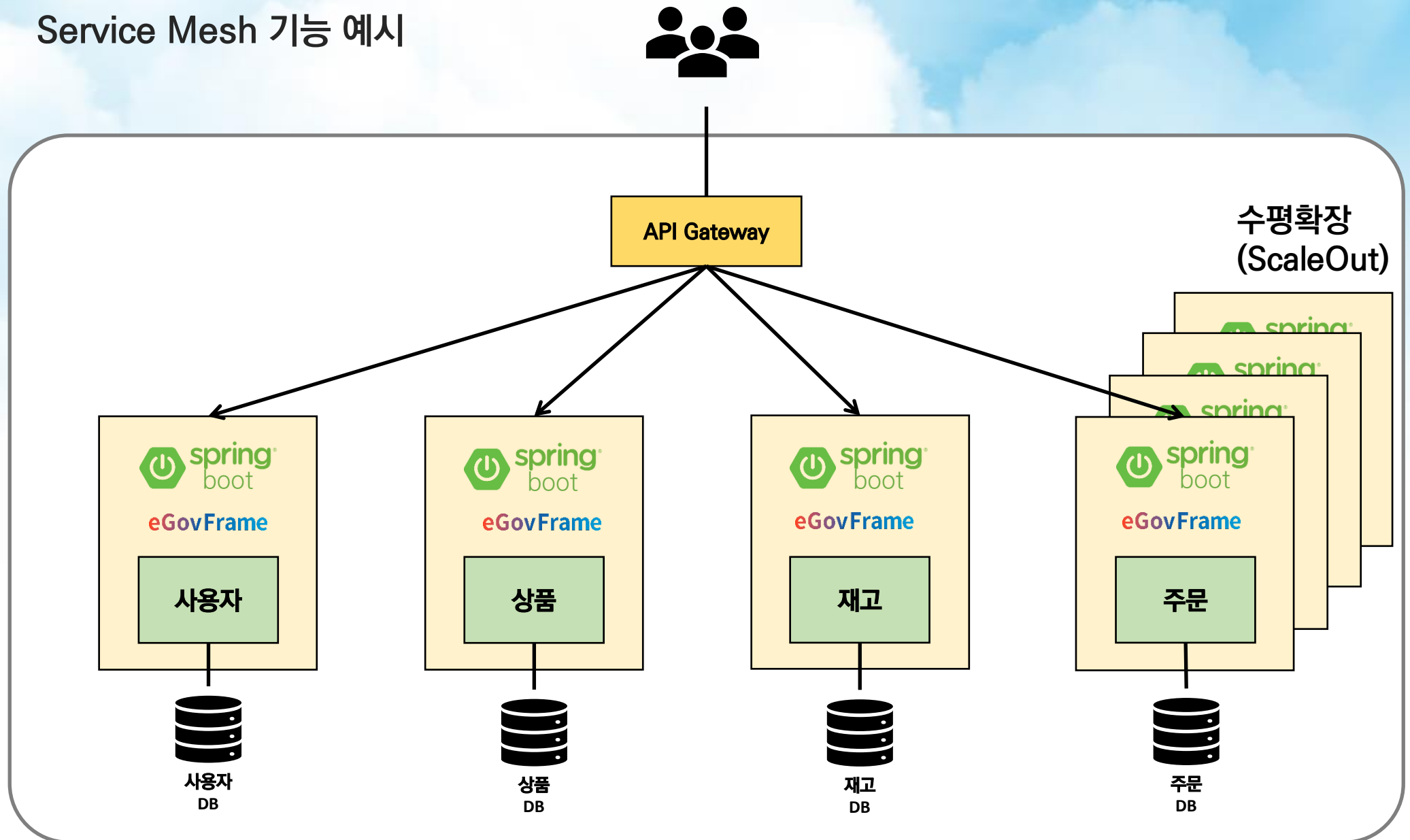
Service Mesh

- Service Mesh 기능 예시



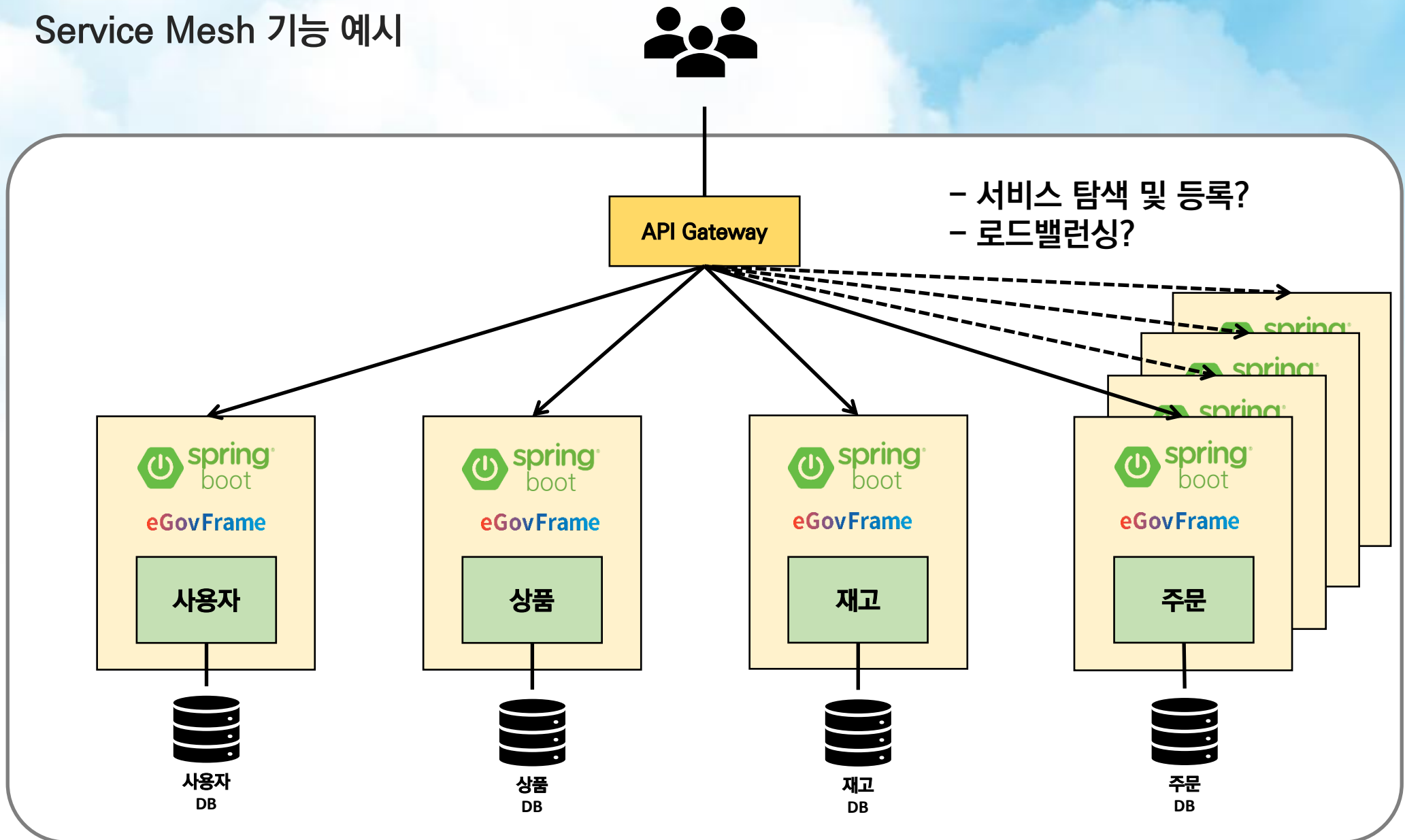
Service Mesh

- Service Mesh 기능 예시



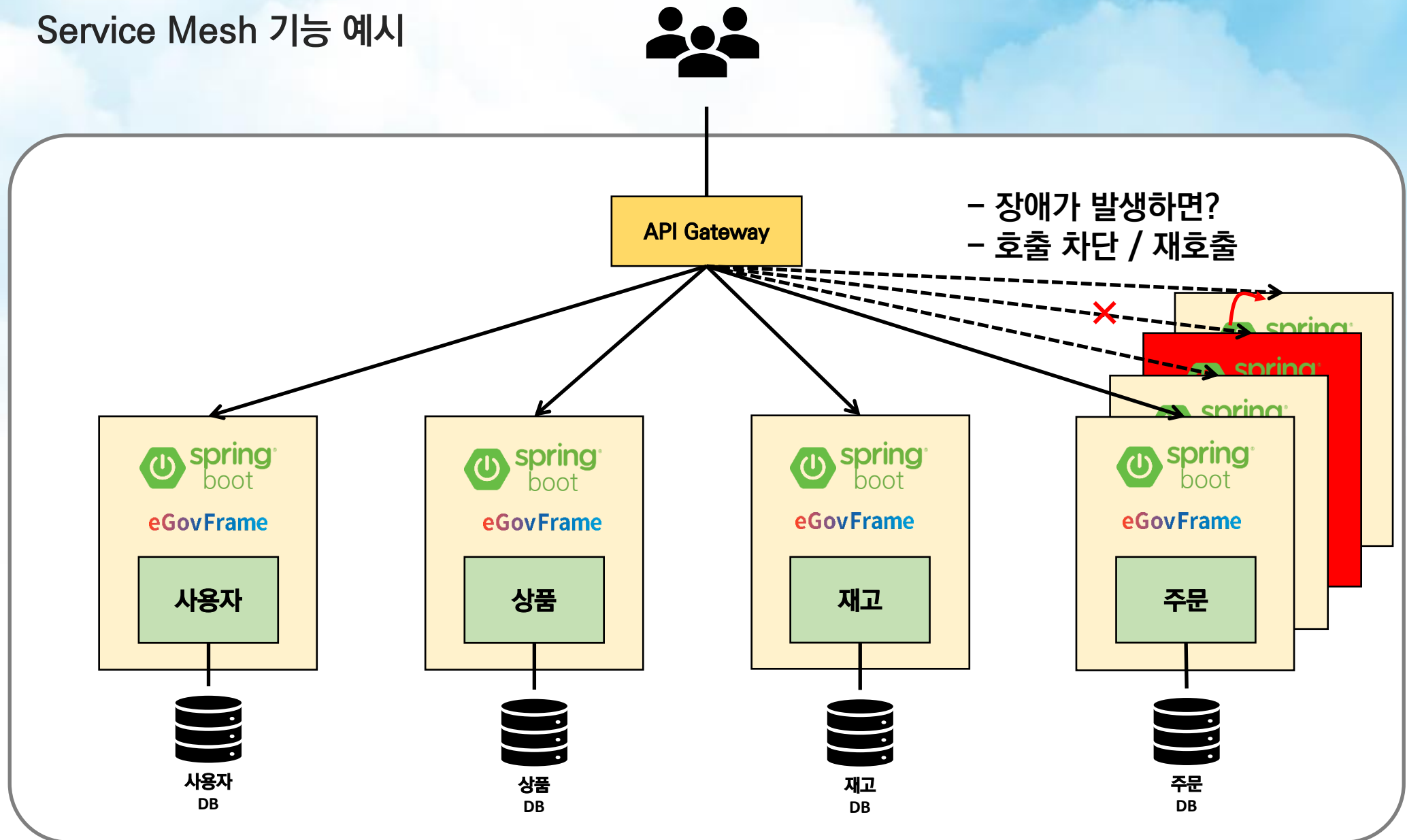
Service Mesh

Service Mesh 기능 예시



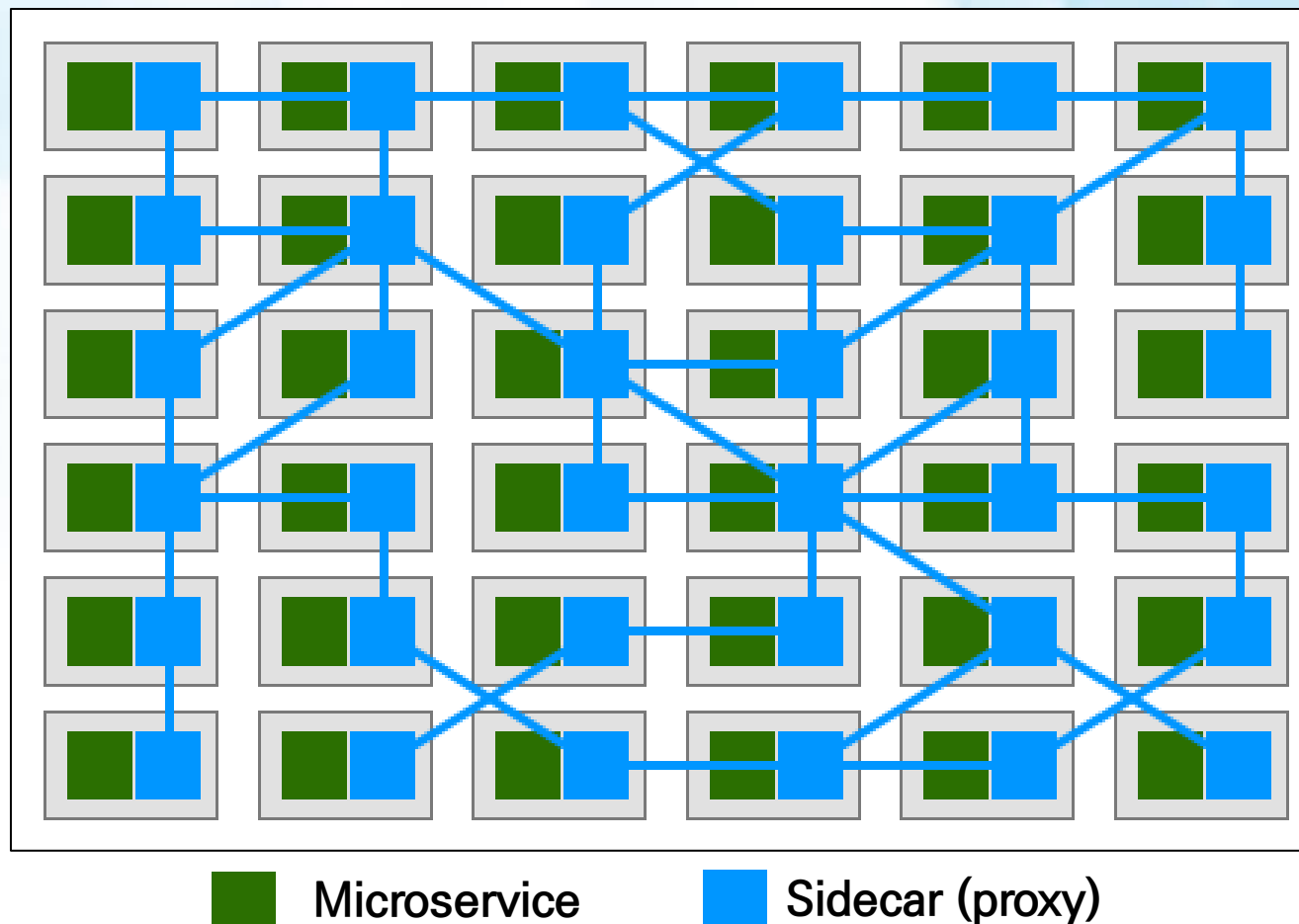
Service Mesh

Service Mesh 기능 예시



Service Mesh

- 마이크로서비스간의 인터페이스 이슈 발생
- MSA가 유발하는 여러 문제점들을 보완하는 개념 (네트워크 기술)



https://philcalcado.com/2017/08/03/pattern_service_mesh.html

Service Mesh

■ 주요 기능들



설정
(외부 중앙화)



서비스
탐색



서비스
라우팅



로드
밸런싱



상태
체크



장애 차단
/ 회복




인증



보안

Service Mesh

Service Mesh를 구현하는 방법

	Application 레벨에서 직접 구현 (library)	클라우드 플랫폼에서 지원하는 기능 활용	컨테이너 관리 도구 또는 Sidecar Proxy OSS 활용
종류	 Spring Cloud  Netflix OSS  finagle  Hydra  steeltoe by Pivotal	 PaaS-TA  CLOUDFOUNDRY  kubernetes  AWS App Mesh  Azure Fabric  Google Cloud Anthos	 kubernetes  docker SWARM  MESOS  envoy  Istio  LINKERD  CONDUIT

Spring Cloud



Spring
Cloud

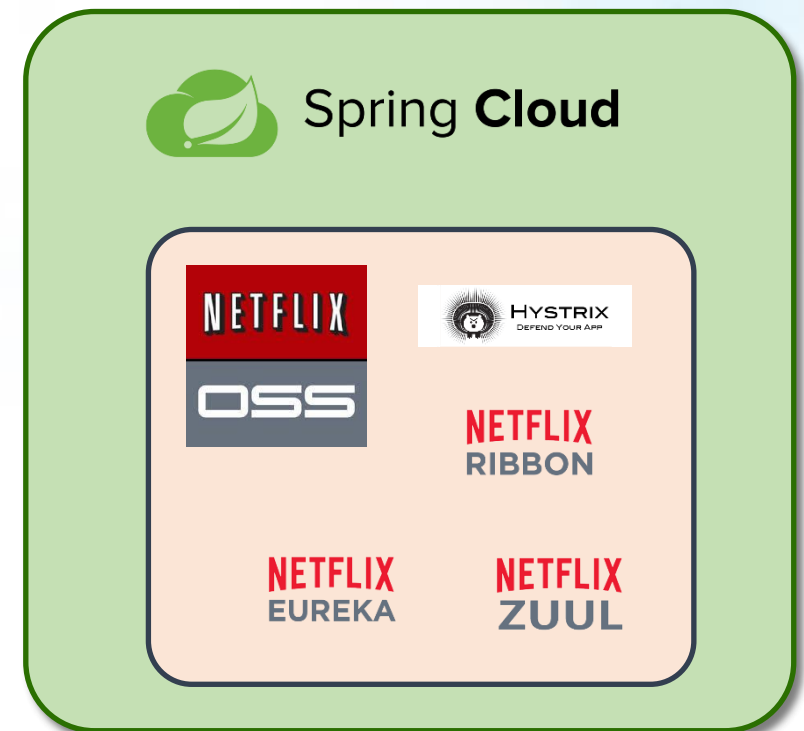
- MSA 구축에 특화된 JAVA 라이브러리들의 집합체
- Application 레벨에서 MSA 구축 가능 (JVM 만 있다면 OK)
- 다양한 미들웨어 인터페이스를 제공, 여러 오픈소스들을 통합하여 제공
- 대표적으로 **NETFLIX** | **OSS** 가 있음



Spring
Cloud



- 서비스 탐색/등록 (Spring Cloud Netflix – Eureka)
- 서비스 라우팅 (Spring Cloud Netflix – Zuul)
- 로드밸런싱 (Spring Cloud Netflix – Ribbon)
- 장애 차단/회복 (Spring Cloud Netflix – Hystrix)
- 설정중앙화 (Spring Cloud Config)
- 분산 추적/로그 중앙화 (Spring Cloud Sleuth / zipkin)
- Http Client Binder (Spring Cloud OpenFeign)
- 이벤트 기반 트랜잭션 처리 (Spring Cloud Stream)
- 보안/인증 (Spring Cloud Security, OAuth2)
- Polyglot (Spring Cloud Sidecar)



Contents



MSA 배포 및 운영

1. 마이크로서비스 실행환경
2. 마이크로서비스 배포
3. 컨테이너
4. 도커
5. 쿠버네티스 (Kubernetes)
6. 향후 표준프레임워크 MSA 지원 계획



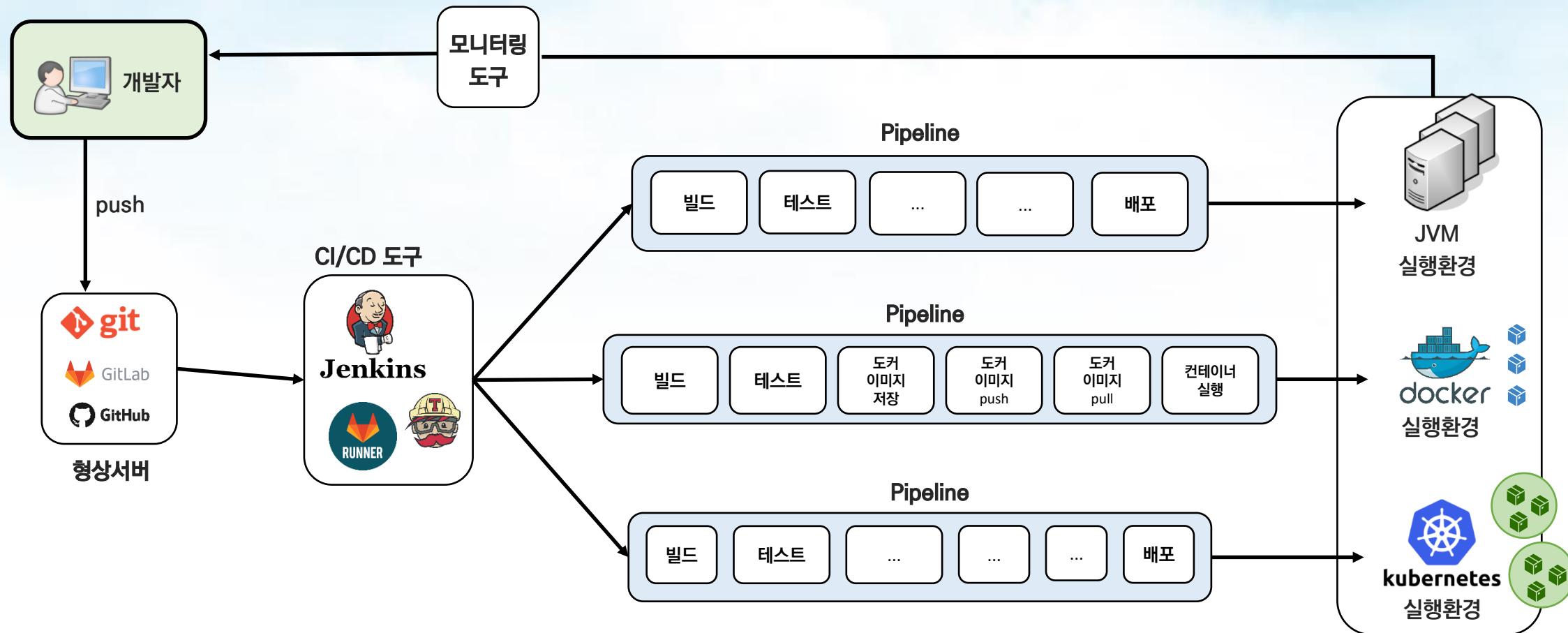
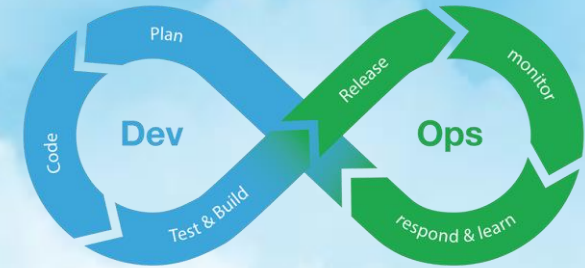
마이크로서비스 실행환경

- 마이크로서비스 실행환경은 크게 3가지정도
- 비용, 기간, 역량에 따라 판단하여 선택

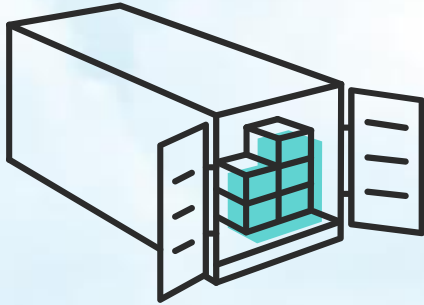
실행 환경	어플리케이션 직접 실행 (JVM)	컨테이너 실행 (OCI, Docker...)	컨테이너 관리 도구 (Kubernetes...)
특징	JVM 설치 후 어플리케이션들을 직접 실행 (중/소규모 시스템에 적합)	어플리케이션을 컨테이너화하여 실행 (중/소규모 시스템에 적합)	컨테이너들을 관리해주는 오케이스트레이션 툴 활용 (대규모 시스템에 적합)

마이크로서비스 배포

- CI/CD 도구를 활용한 배포자동화 필요
- 실행환경에 따라 배포 방법이 다름
- 각 실행환경에 맞는 CI/CD 파이프라인을 구성하여 배포

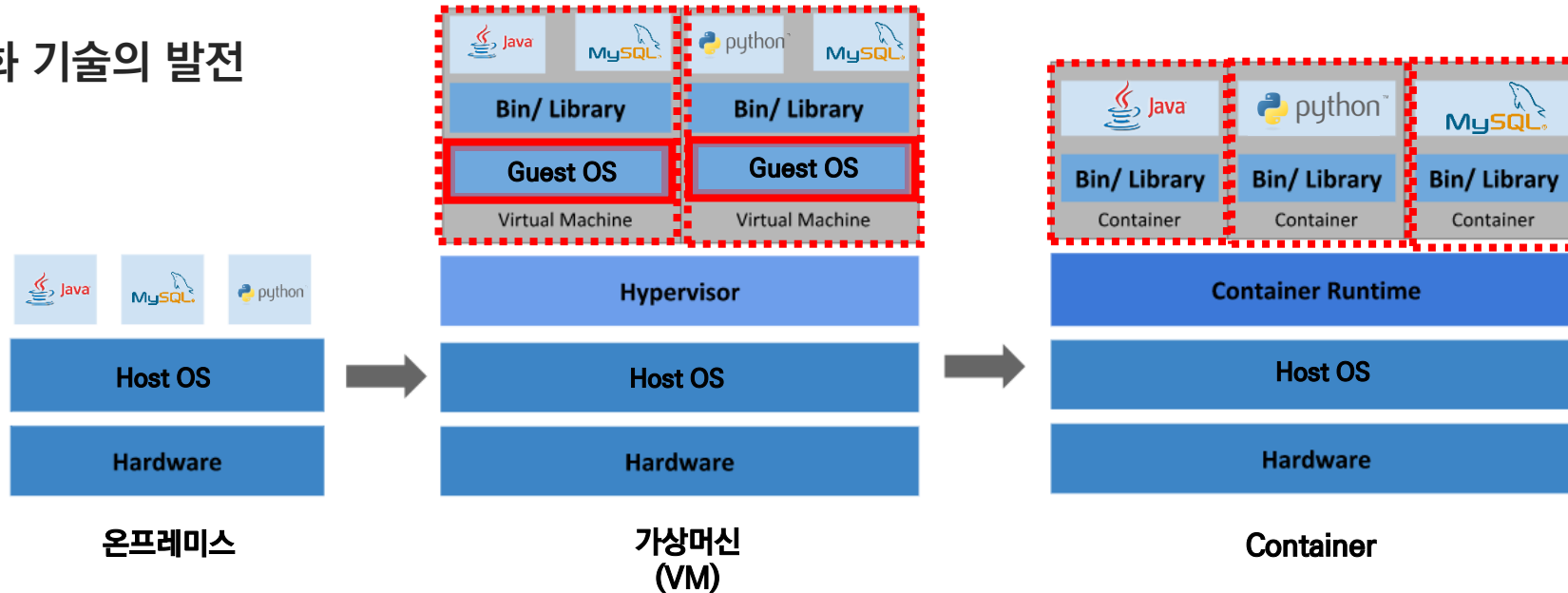


컨테이너

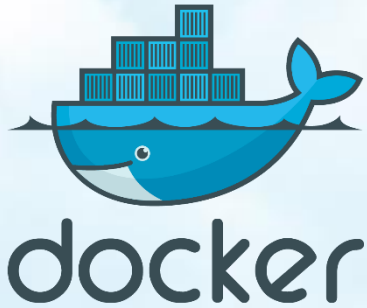


- 가상화 기술의 일종 (**경량화 된 VM**)
어플리케이션을 가상화 환경에서 구동하는 환경을 격리한 공간을 의미
- 게스트OS가 없기 때문에 VM보다 가볍고 빠르게 동작
- 특정 환경에 관계없이 일관성 있는 격리된 실행 환경 제공

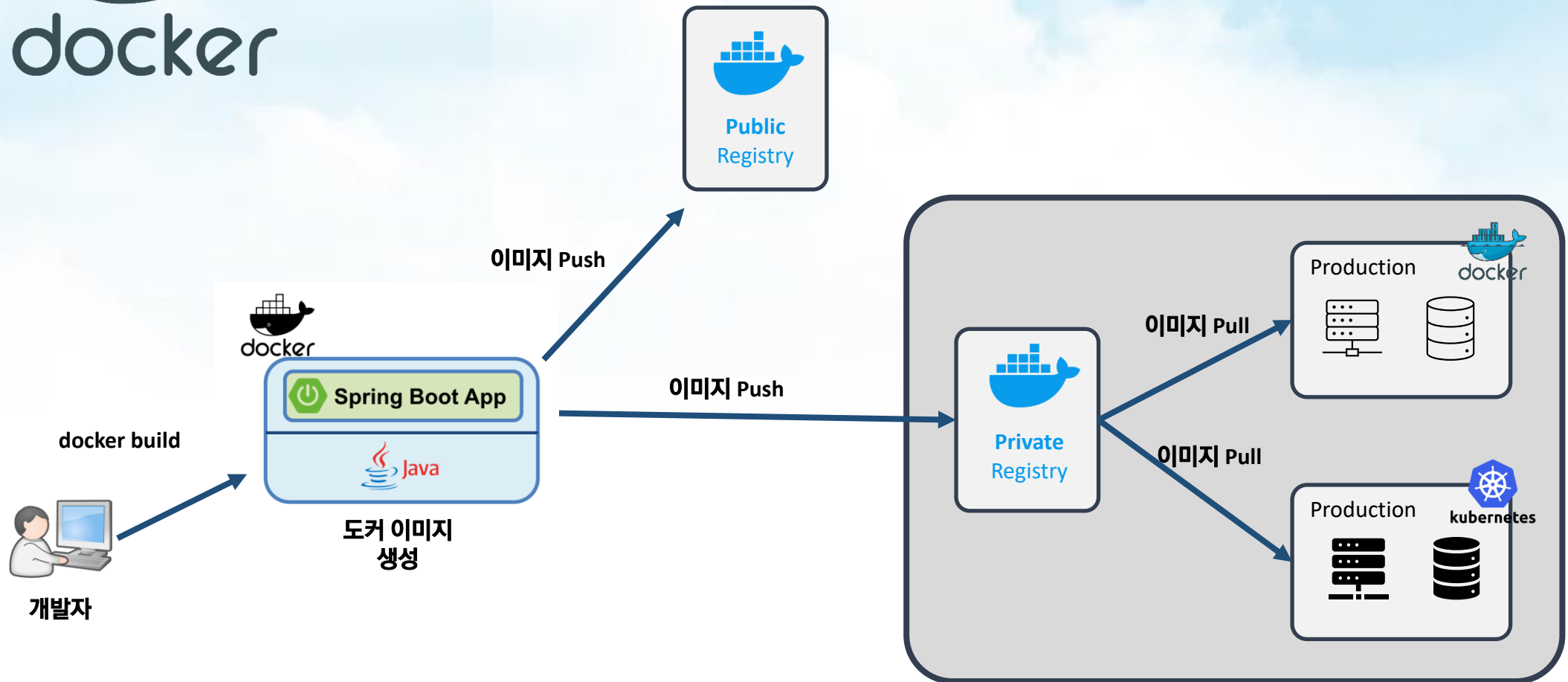
가상화 기술의 발전



도커



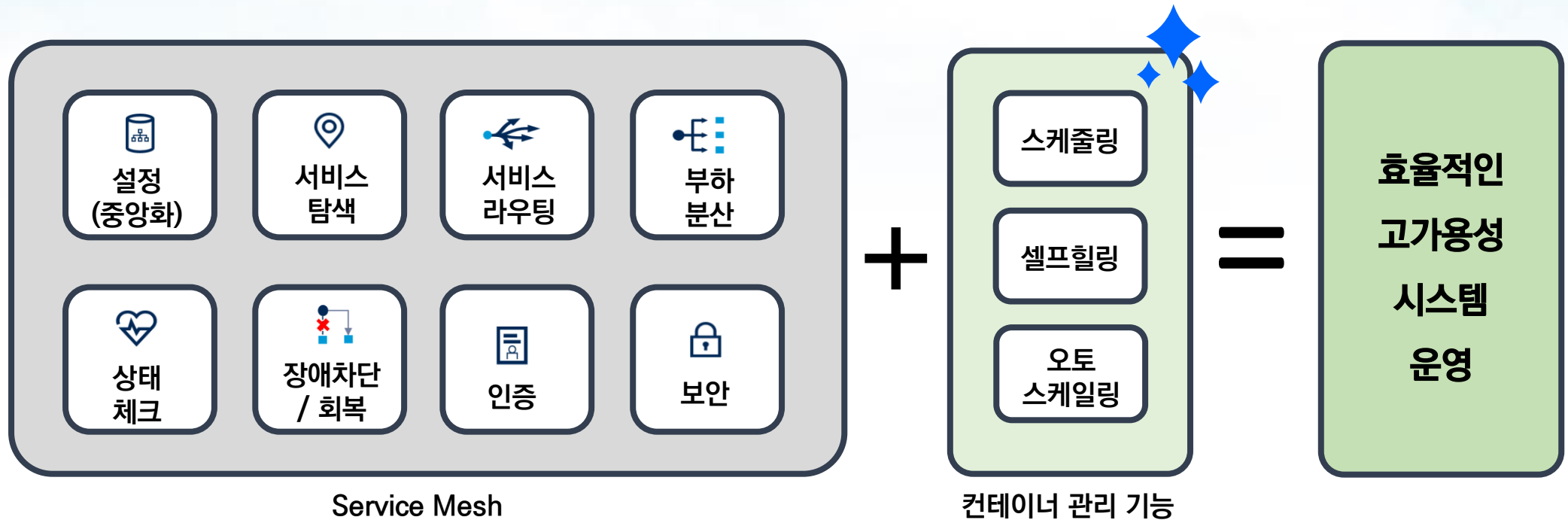
- 컨테이너 기술중의 하나로 가장 널리 알려진 오픈소스 가상화 플랫폼
- 배포절차 : 어플리케이션 -> 도커 이미지 생성 -> 저장소 -> 컨테이너 배포



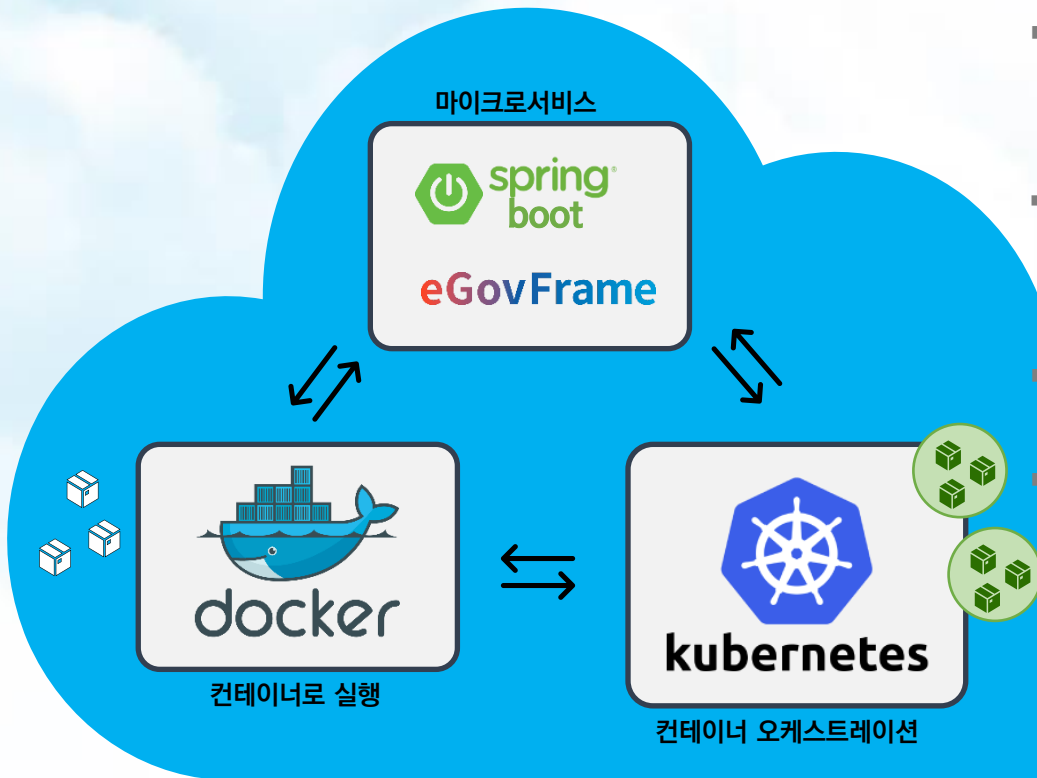
쿠버네티스 (Kubernetes)



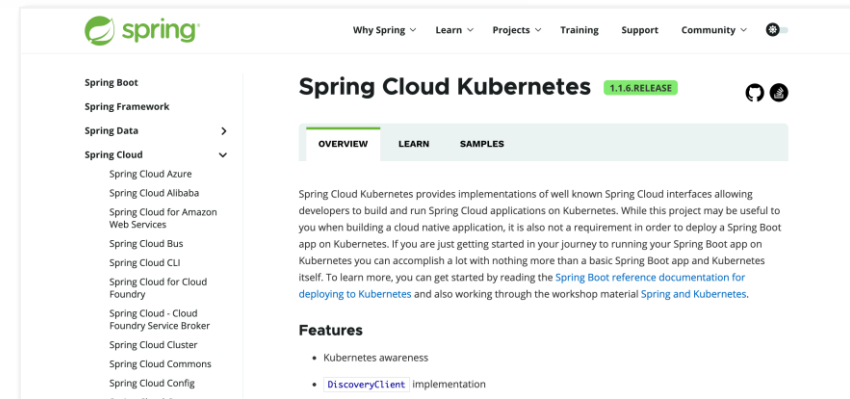
- 그리스어로 조타수/항해사 라는 의미
- K8S (K . . . S) , 구글에서 공개한 오픈소스
- 도커 = 선박 / 쿠버네티스 = 선박 총괄 지휘 (컨테이너 오케스트레이션)
- 목적 : 서버들을 잘 관리하기 위함
(컨테이너들을 효율적으로 자동으로 관리)
- 강력한 컨테이너 자동 관리 기능 제공



표준프레임워크 on Kubernetes



- Spring Boot + Kubernetes
 - 어플리케이션 상태 관련 연동 (actuator ↔ probe)
 - 부트 플러그인으로 도커 이미지 생성, 효율적인 도커 이미지 생성 지원 (이미지 계층화)
- Spring Cloud + Kubernetes 조합도 가능
- Spring Cloud Kubernetes
 - (현재 버전 stable 1.1.6 (베타2.0))
 - (대체 활용 예시)
 - Config Server → Config Map
 - API Gateway → Service



향후 표준프레임워크의 MSA 지원 계획

- 표준프레임워크 4.0 업데이트 (Spring Boot 환경 지원)
- 공통컴포넌트의 마이크로서비스화 (템플릿)
- Spring Cloud 컴포넌트 활용 가이드 추가
- 컨테이너 관련 기술 가이드 (개념이해)



감사합니다.

for Your Attention

