MSA 1주차

스터디원	소현
Status	Done
비고	MSA Section0
제출 마감일	@September 13, 2023 5:00 PM

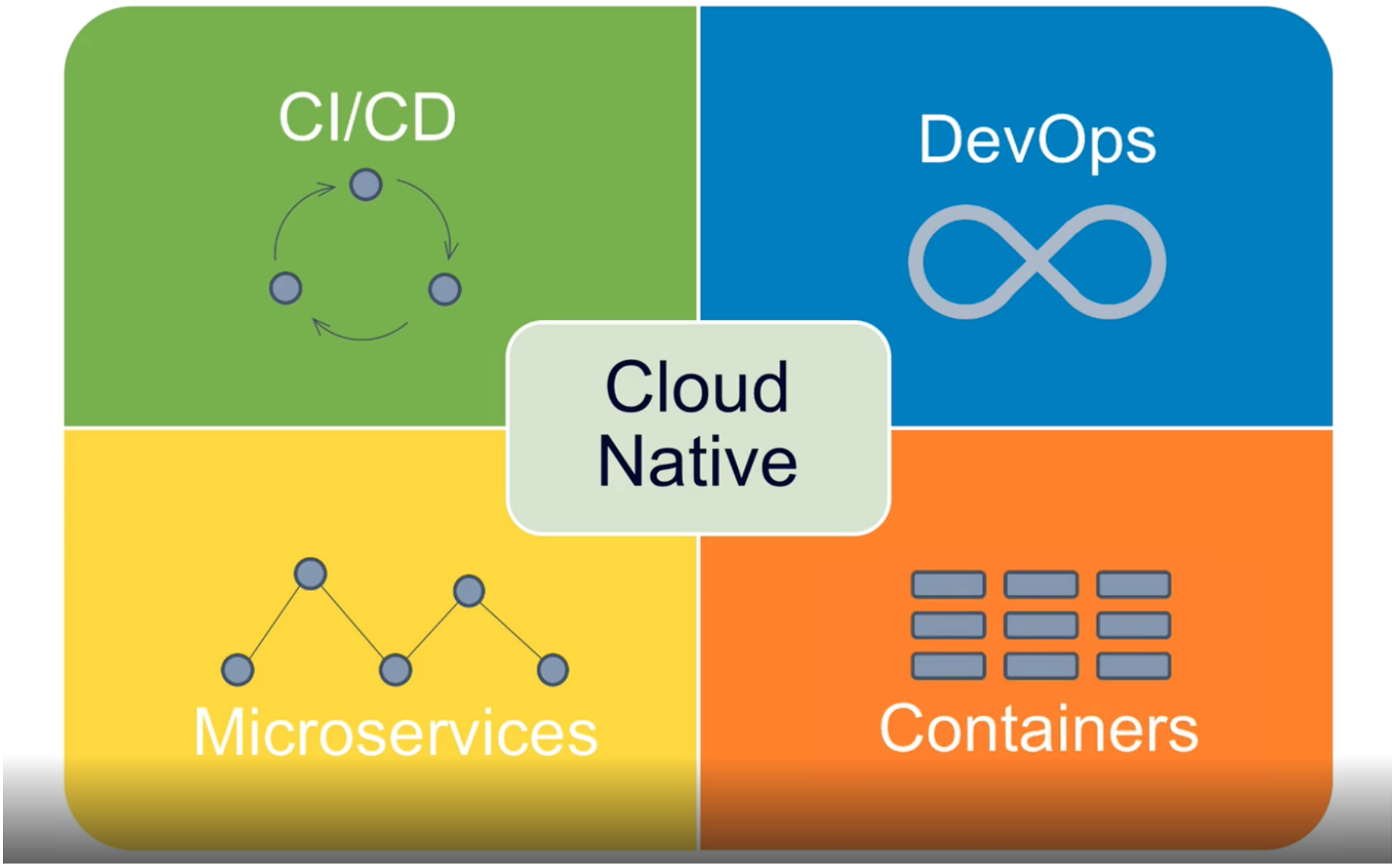
Microservice와 Spring Cloud 소개

Cloud Native Architecture

> 💡 **Cloud Native**
> 클라우드 제공 모델에서 제공하는 분산 컴퓨팅을 활용하기 위해 애플리케이션을 구축 및 실행하는 개념

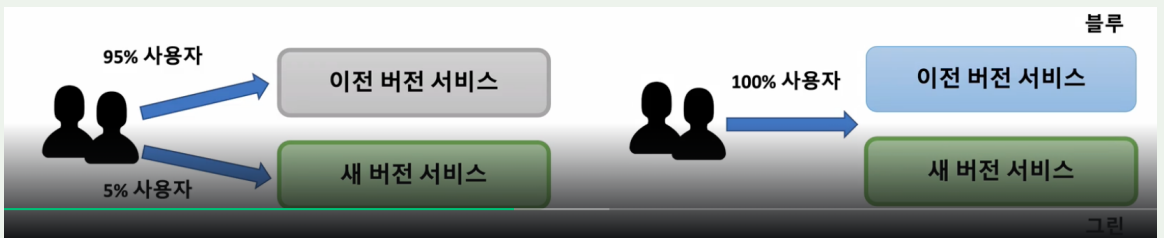
- **확장 가능한 아키텍처**
  - 시스템의 수평적 확장에 유연
  - 확장된 서버로 시스템의 부하 분산, 가용성 보장
  - 시스템 또는 서비스 애플리케이션 단위의 패키지 (컨테이너 기반 패키지)
  - 모니터링
- **탄력적 아키텍처**
  - 서비스 생성 - 통합 - 배포, 비즈니스 환경 변화에 대응 시간 단축
  - 분할된 서비스 구조
  - 무상태(Stateless) 통신 프로토콜
  - 서비스의 추가와 삭제 자동으로 감지
  - 변경된 서비스 요청에 따라 사용자 요청 처리 (동적 처리)
- **장애 경리 (Fault Isolation)**
  - 특정 서비스에 오류가 발생해도 다른 서비스에 영향을 주지 않음

Cloud Native Application



▼ **CI/CD**

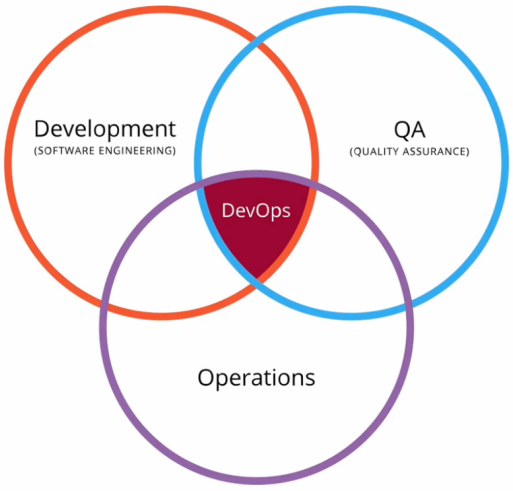
- **지속적인 통합, CI(Continuous Integration)**
  - 통합 서버, 소스 관리(SCM), 빌드 도구, 테스트 도구
  - Ex) Jenkins, Team CI, Travis CI
- **지속적 배포, CD**
  - Continuous Delivery (실행 파일 → 실행 환경에 `수동 반영`)
  - Continuous Deployment (실행 파일 → 실행 환경에 `자동 반영`)
  - Pipe line
  - 카나리 배포와 블루그린 배포

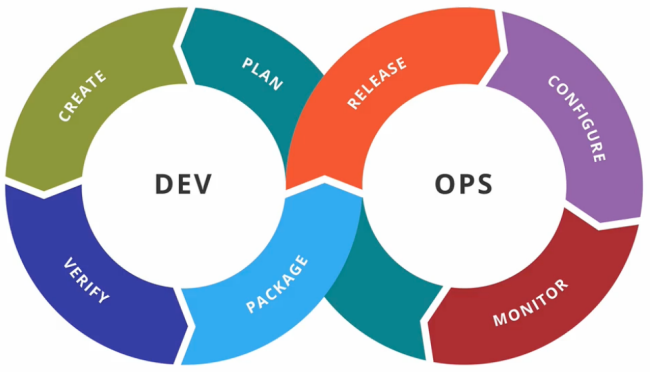


블루

그린

▼ **DevOps**

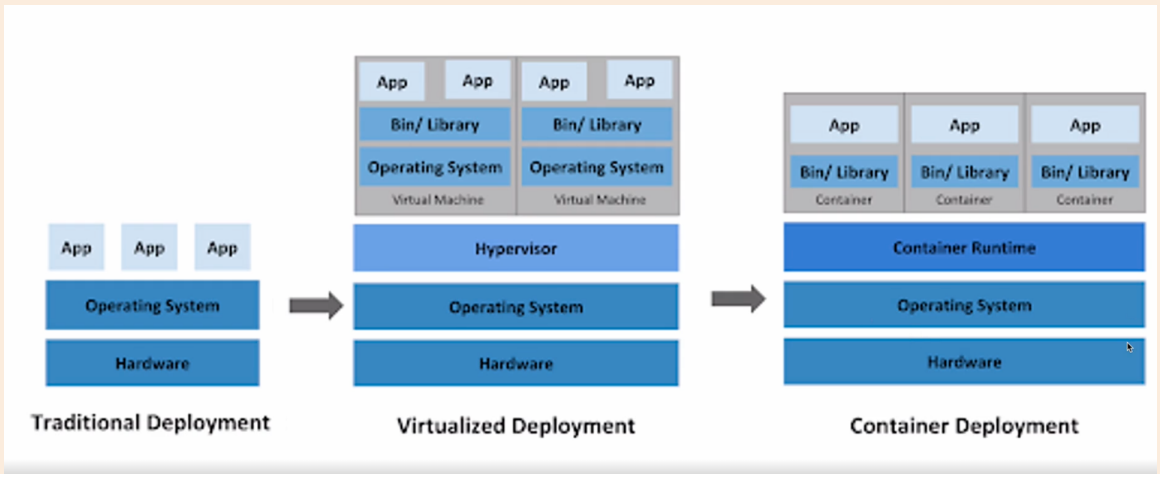




- 개발 조직과 운영 조직의 통합
- 고객의 요구사항을 빠르게 반영하고 만족도 높은 결과를 제시하는 것이 목표
- 서비스의 구조를 작은 단위로 분할하여 자주 통합, 테스트, 배포할 수 있는 구조가 될 수 있도록 함

▼ **Container 가상화**

- Local → Cloud 환경으로 이동하여 적은 비용으로 탄력성 있는 시스템을 구축할 수 있는 기술
- Traditional Deployment → Virtualized Deployment → Container Deployment



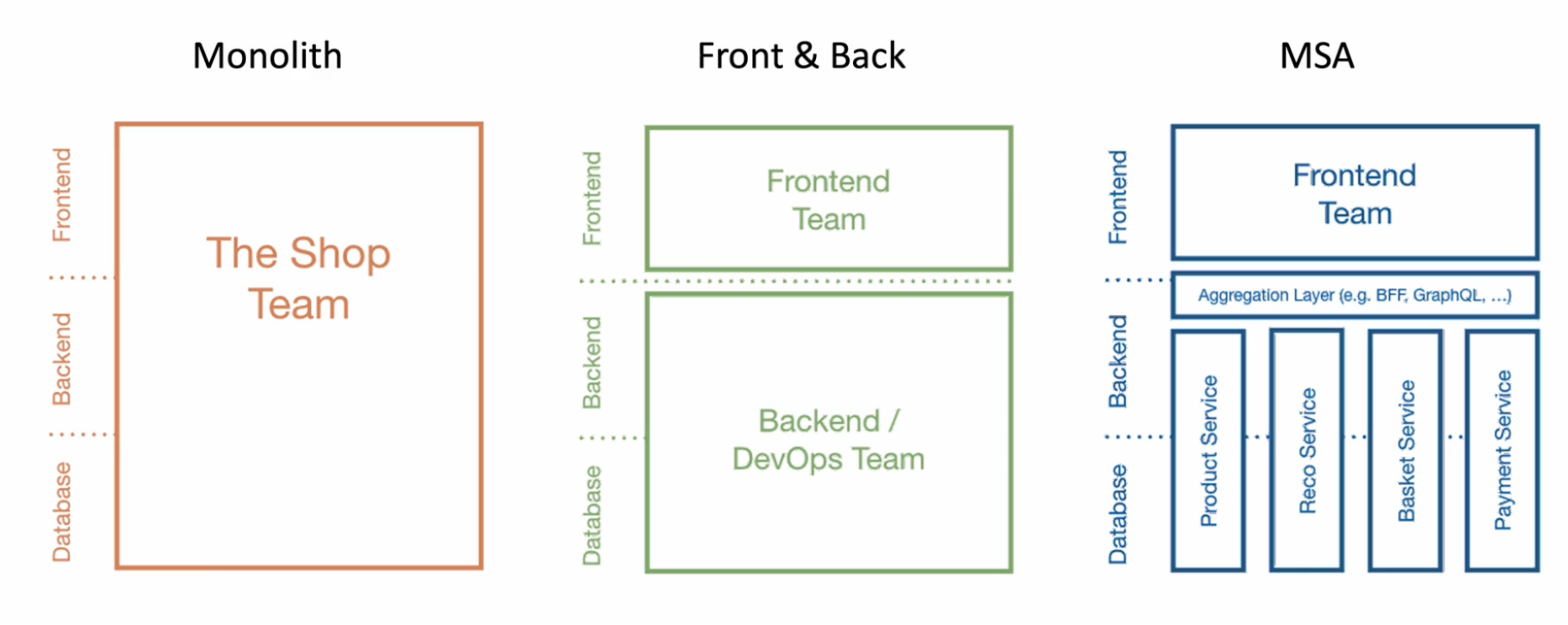
▼ **Microservice**

12 Factors (https://12factor.net)

- **Base code**
  - 버전, 형상 관리 목적
- **Dependency isolation (종속성)**
  - 종속성을 가지고 있어서 전체 시스템에 영향을 주지 않고 변경될 수 있어야 함
- **Configurations (구성 정보)**
  - 시스템 코드 외부에서 구성 관리 도구를 통해서 마이크로서비스에 필요한 것들을 제어
- **Linkable backing services (서비스 지원)**
  - 마이크로서비스가 필요한 추가적인 지원
- **Stages of creation (Build, Release, Run 분리)**
  - 자동화 된 시스템을 구축하는 것이 중요
- **Stateless Process**
  - 각 마이크로서비스는 독립된 프로세스에서 실행되어야 함 (종속성과 같은 의미)
- **Port binding**
  - 각 마이크로서비스는 자체 포트에서 노출되는 인터페이스 및 기능과 함께 자체 포함되는 기능이 존재
- **Concurrency (동시성)**

    - 하나의 서비스가 여러 인스턴스의 동일한 형태로 복사되어 운영됨으로써 부하 분산
- **Disposability**
    - 서비스 인스턴스 자체가 삭제 가능, 확장성 기회 향상, 정상적인 종료가 가능해야 함
- **Development & Production parity**
    - 개발 단계와 프로덕션 단계를 구분할 수 있어야 함
- **Logs**
    - 로그만은 정상적으로 작동되어야 함
- **Admin processes for eventual processes**
    - 실행 중인 마이크로서비스를 파악하고 관리하기 위한 도구가 필요
- 최근 3가지 항목 추가
    - **API first** : API 형태로 정보 제공, 사용자 측에서 어떤 형태로 사용할 것인지 먼저 고민
    - **Telemetry** : 모든 지표는 수치화, 시각화 되어 관리할 수 있는 항목이어야 함
    - **Authentication and authorization** : API 사용함에 있어서 인증 인가는 필수

Monolithic vs Microservice



- **Monolithic Architecture**
    - 모든 업무 로직이 하나의 애플리케이션 형태로 패키지 되어 서비스
    - 애플리케이션에서 사용하는 데이터가 한 곳에 모여 참조되어 서비스되는 형태
- **MSA(Microservice Architecture)**
    - 🔽

MSA

- **Microservice 특징**
    - Challenges
    - Small Well Chosen Deployable Units
    - Bounded Context
    - RESTful
    - Configuration Management
    - Cloud Enabled

    - Dynamic Scale Up And Scale Down
    - CI/CD
    - Visibility
- **Everything should be a microservice ?** `No`
    - Multiple Rates of Change
    - Independent Life Cycles
    - Independent Scalability
    - Isolated Failure
    - Simplify Interactions with External Dependencies
    - Polyglot Technology

SOA vs MSA

	SOA	MSA
서비스의 공유 지향점	재사용을 통한 비용 절감	서비스 간의 결합도를 낮추어 변화에 능동적으로 대응
기술 방식	공통의 서비스를 ESB에 모아 사업 측면에서 공통 서비스 형식으로 서비스 제공	각 독립된 서비스가 노출된 REST API를 사용

- **RESTful API**
    - Consumer first
    - Make best use of HTTP
    - Request methods
    - Response Status
    - No secure info in URI
    - Use plurals
    - User nouns for resources
    - For exceptions : define a consistent approach

Microservice Architectures Structures

- **Service Mesh Capabilities**
    - MSA 인프라 → 미들웨어(추상적인 개념)
        - 프록시 역할, 인증, 권한 부여, 암호화, 서비스 검색, 요청 라우팅, 로드 밸런싱
        - 자가 치유 복구 서비스
    - 서비스 간의 통신과 관련된 기능을 자동화
- **MSA 기반 기술**
    - Gateway
    - Resilient Service Mesh/Meta Services
    - Runtime
    - Backing Services
    - Frameworks
    - Automation
    - Telemetry

Spring Cloud

- **Spring Boot + Spring Cloud** (버전 유의)
- **Main Projects (사용할 프로젝트)**
  - Spring Cloud Config
  - Spring Cloud Netflix
  - Spring Cloud Security
  - Spring Cloud Sleuth
  - Spring Cloud Starters
  - Spring Cloud Gateway
  - Spring Cloud OpenFeign
- **필요한 서비스 구성**
  - Centralized configuration management (환경 설정 관리)
    - Spring Cloud Config Server
  - Location Transparency
    - Naming Server (Eureka)
  - Load Distribution (Load Balancing)
    - Ribbon (Client Side)
    - Spring Cloud Gateway
  - Easier REST Clients
    - FeignClient
  - Visibility and monitoring
    - Zipkin Distributed Tracing
    - Netflix API gateway
  - Fault Tolerance
    - Hystrix

필수 SW 설치

- [x] ~~IntelliJ IDEA Ultimate~~
- [x] ~~Git~~
- [x] ~~Visual Studio Code~~
- [x] ~~Postman~~