Microservice와 Spring Cloud의 소개-1

작성 일시	@2022년 10월 19일 오후 8:25
최종 편집 일시	@2022년 10월 24일 오후 8:28
유형	Spring Cloud로 개발하는 마이크로서비스 애플리케이션(MSA)
작성자	
참석자	

소프트웨어 아키텍처
Cloud Native Architecture
Cloud Native Application
12 Factors (https://12factor.net)

소프트웨어 아키텍처

Anti-fragile

1. Auto scaling (자동 확장성) - 인스턴스 개수를 동적으로 관리 할 수 있음.
2. MicroServices (모듈화) - 서비스를 각각의 모듈로 나눠서 서비스 함.
3. Chaos engineering (정상 상태) - 변동, 불확실성에 대해 서비스는 안정적으로 동작해야함.
4. Continuous deployment (지속적인 CI/CD) - CI 빌드 테스트 자동화 과정, CD 지속적인 서비스 제공, 두 의미 모두 파이프 라인의 추가 단계, MicroService 각각 모듈을 모두 테스트, 빌드 등 지속적으로 관리하기 어려움으로 필요.

Cloud Native Architecture

확장 가능한 아키텍처

- 시스템의 수평적인 확장에 유연

- 확장된 서버로 시스템의 부하 분산, 가용성 보장
  - 스케일 업 (하드 웨어 스팩 업)
  - 스케일 아웃 (서버 클러스터링)
- 시스템 또는 서비스 애플리케이션 단위의 패키지 (컨테이너 기반 패키지)
- 모니터링

탄력적 아키텍처

- 서비스 생성 - 통합 - 배포, 비즈니스 환경 변화 대응 시간 단축
  - CI - CD
- 분할된 서비스 구조
- 무상태 통신 프로토콜 (REST API)
- 서비스 추가와 삭제 자동 감지
- 변경된 서비스 요청에 따라 사용자 요청 처리 (동적)

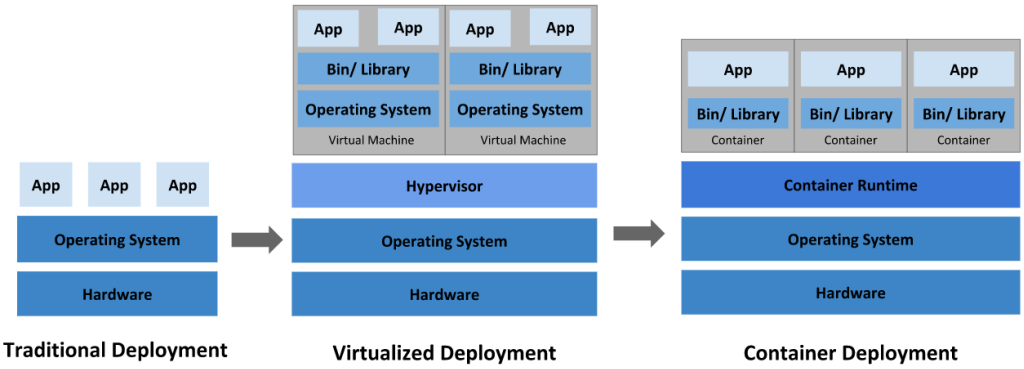
장애 격리

- 특정 서비스에 오류가 발생되어도 다른 서비스에 영향이 가지 않음

Cloud Native Application

- **Microservices**
- **CI/CD**
  - 지속적인 통합 CI
    - 통합 서버, 소스 관리(SCM), 빌드 도구, 테스트 도구
    - ex) Jenkins, Team CI
  - 지속적 배포
    - Continuous Delivery - 수동 반영
    - Continuous Deployment - 자동 반영
  - 카나리 배포와 블루그린 배포
    - 카나리 배포 (대부분 구 서비스, 나머지 신 서비스)

        - 블루그린 배포 (이전 사용 중인 서비스를 점진적으로 새 서비스로 할당)
- **DevOps (개발 + 운영 조직)**
- **Containers 가상화**
    - Traditional Deployment
        - 머신 하나에 여러 어플리케이션을 실행하면, 리소스를 대부분 차지 하는 인스턴스가 있을 수 있고 결과적으론 나머지 애플리 케이션 성능이 떨어질 수 있다.
        - 이를 해결 하는 방법으로, 여러 머신을 두고 머신당 하나의 애플리케이션을 실행하는 방법이 있으나, 머신 리소스가 충분히 활용되지 않을 뿐더러, 여러 머신을 관리하기 위해 비용이 많이든다.
    - Virtualized Deployment
        - 단일 머신에서 CPU에 여러 가상 시스템을 실행 할 수 있게 한다. 가상화를 사용하면 VM 간에 애플리케이션을 격리하고 다른 애플리케이션에 엑세스 할 수 없음으로, 일정 보안을 제공한다.
        - 각 VM은 가상화된 하드웨어 상에서 자체 운영체제를 포함한 모든 구성 요소를 실행하는 하나의 머신이다.
    - Container Deployment
        - VM 과 유사하지만, 격리 속성을 완화하여 애플리케이션 간에 운영체제를 공유한다. 컨테이너는 OS를 포함하지 않음으로 보다 가볍다고 여겨진다. VM 과 마찬가지로 컨테이너에는 자체 파일 시스템, CPU, 메모리, 프로세스 공간 등이 있다.
        - VM 이미지는 너무 무겁지만, 컨테이너 이미지는 보다 효율적임.
        - 개발과 운영의 관심사가 분리된다.
        - 개발 테스팅 및 운영 환경에 걸친 일관성
        - 분산 및 플렉서블 함으로 마이크로서비스에 맞다.
        - 리소스 격리
        - 이식성, 고효율, 고집적임.
        - 이미지



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

- Cloud Native Application 고려할 항목
  - BASE CODE (코드 통합)
    - 형상 관리
  - DEPENDENCY ISOLATION (종속성 배제)
    - 각 모듈 서비스는 자체 패키징의 종속성이 있음으로, 다른 서비스에 종속성이 없어야 됨.
  - CONFIGURATIONS (환경 설정의 외부 관리)
    - 시스템 코드 외부에서 구성 관리 도구를 통해 내부 작업을 제어한다.
  - LINKABLE BACKING SERVICES (백업 서비스 분리)
    - 보조 서비스 (DB, Cache, Message Broker) 각 서비스기능에 추가 지원 할 수 있어야함.
  - STAGES OF CREATION (개발 환경, 테스트 운영 환경 분리)
    - Build, Release, run 환경 분리.
  - STATELESS PROCESSES (상태 관리)
    - 서비스는 독립적으로 있어야하며, 공유되어야 하는 자원은 캐쉬, 메모리 같은 외부 공유 자원을 둬서 관리한다. (데이터 동기화)
  - PORT BINDING (포트 바인딩)
    - 마이크로 서비스는 자체 포트에서 노출되는 인터페이스 기능과 함께 자체 포함되는 기능이 있어야함. 다른 마이크로 서비스와 분리가 가능함으로
  - CONCURRENCY (동시성)

        - 서비스는 사용가능한 가장 강력한 인스턴스 확장하는 것과 반대로 아주 많은 수에 서비스를 동일한 프로세스를 복사해 생성해야함.
    - DISPOSABILITY (서비스 상태 유지)
        - 서비스 인스턴스 자체가 삭제가 가능해야하고, 확장성이 있어야 하며, 정상 삭제가 되는 상태가 되어야함.
    - DEVELOPMENT & PRODUCTION PARITY (개발과 운영 구분)
        - 개발과 운영을 중복, 종속적이지 않도록 한다.
    - LOGS (로그 분리)
        - 이벤트 stream 으로 log를 관리해야한다. 정상 적인 마이크로 서비스가 실행되지 않더라도 log는 관리 되어야한다. (모니터링 도구)
    - ADMIN PROCESSED FOR EVENTUAL PROCESSES (관리 프로세스)
        - 모든 마이크로 서비스가 어떻게 사용되고 있는지 적절한 관리 도구가 있어야 함. (report, data analyze 등)
- + 3 Element
    - API FIRST
        - 모든 서비스는 API 로 구성, 제공되어야함.
    - TELEMETRY
        - 모든 지표는 수치화 되어, 시각화 관리 할 수 있어야함.
    - Authentication and Authorization
        - API 사용에 인증 및 인가 는 필수