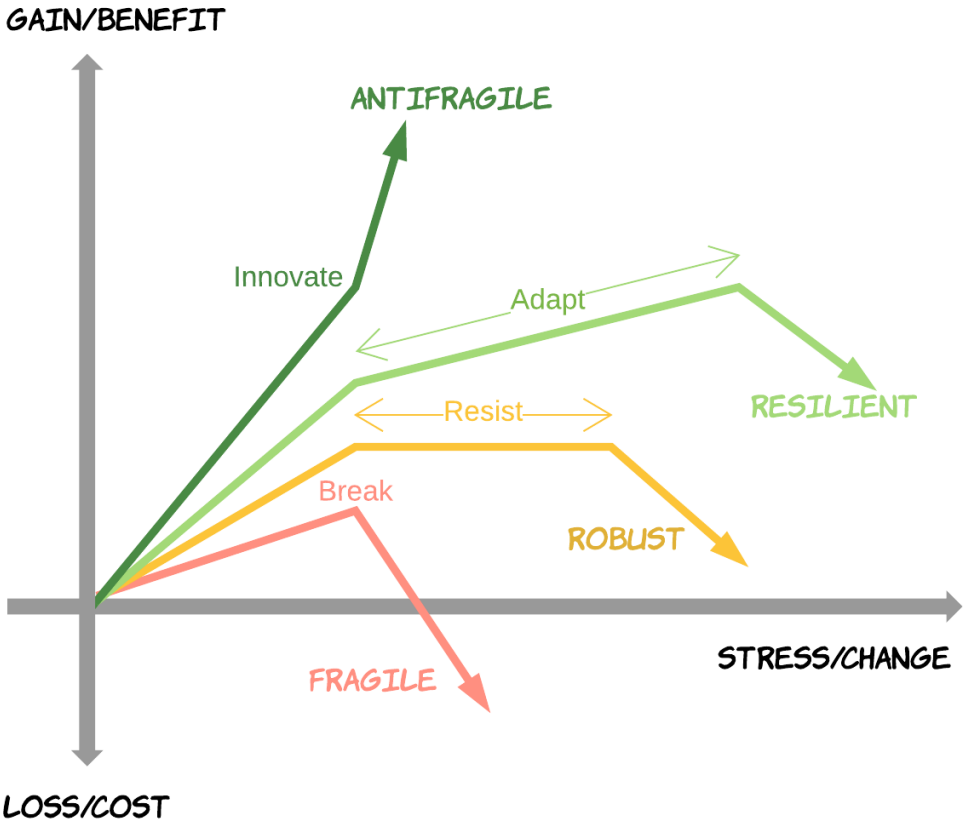
Microservice & Spring Cloud 소개

Created By	㉿ 영교
Class	Spring Cloud로 개발하는 마이크로서비스 앱(MSA)
Tags	MSA Microservice Spring Cloud SpringBoot
# index	1

▼ 목차

▼ The History of IT System

🤖 개요



🕐1960 ~ 1980s

: 하드웨어가 중심이던 Mainframe 시기 소프트웨어보다는 하드웨어의 사양이나 성격에 맞춰 서비스를 구축했고, 하드웨어의 가격자체가 무척 고가였기에 시스템을 변경하기 힘들던 시기.

이 시기는 `Fragile`이라는 깨지기 쉬운 시스템이라는 특징을 가지고 있었다.

🕐1990 ~ 2000s

: 분산이라는 키워드로 칭할 수 있는 이 시기는 `Robust`, `Distributes`로 시스템이 안정화되었고 그 덕의 어느정도의 서비스의 불확실한 변경이 일어나더라도 안정적으로 서비스를 제공할 수 있었다.

🕐2021s ~ current

: 1960~ 1980의 특징인 Fragile과는 반대인 `Anti-Fragile`이라는 특징과 `Cloud Native`로 시스템이 구축되는 시기다. 시스템은 Local → Cloud로 이전되었고, 지속적인 변경사항이 있더라도 탄력적으로 운용될 수 있도록 구축되었다.

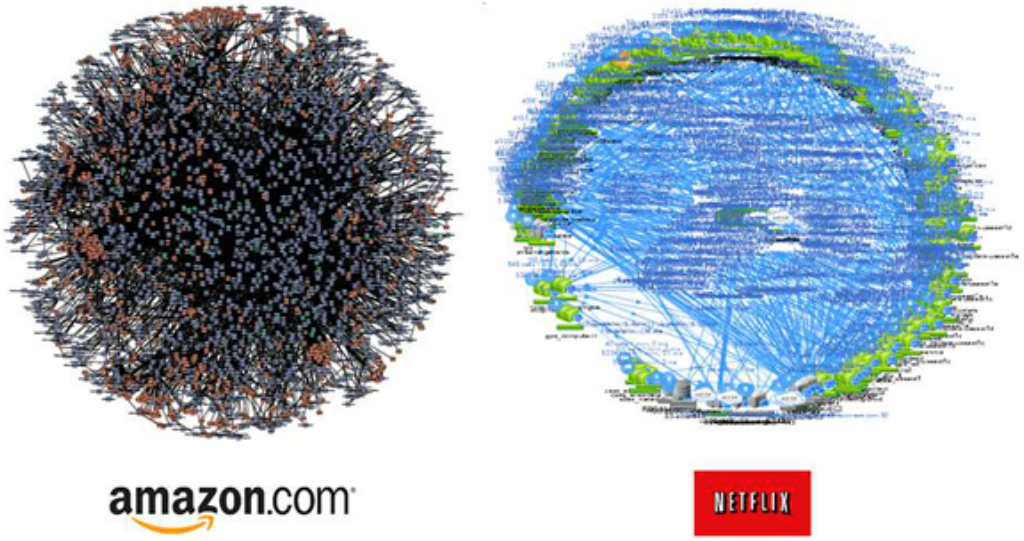
🤖 Anti-fragile 의 특징

특징 1. Auto Scaling

: 자동 확장성을 제공한다고 보면 되는데, 시스템을 구성하고 있는 인스턴스를 하나의 Auto Scaling 그룹으로 묶은 뒤 해당 그룹에서 유지되야하는 최소, 최대 크기를 지정하여 CPU나 메모리, 네트워크나 데이터베이스의 사용량이나 기타 조건에 따라 자동으로 인스턴스가 관리되는것을 Auto Scaling이라 한다.

특징 2. Microservices

: 애플리케이션 구축을 위한 아키텍처 기반의 접근 방식으로 이런 마이크로 서비스의 결합도는 느슨하기에 하나의 애플리케이션이 변경된다고 해서 전체 애플리케이션이 분할되지 않는다. 그 때문에 변화에 유연할 수 있고, 개발 팀에서 새로운 구성 요소 혹은 변경 요소를 빌드하여 변화하는 상황에 대처할 수 있다.

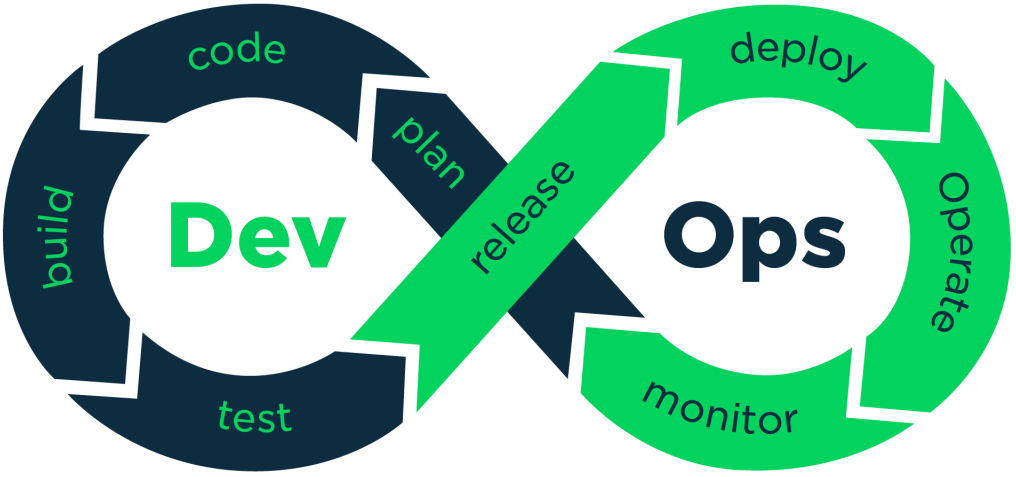


아마존과 넷플릭스의 MSA Architecture

특징 3. Chaos engineering

: 시스템이 예측할 수 있거나(혹은 예측할 수 없는) 상황에서도 시스템이 견딜 수 있고, 안정적인 서비스를 제공할 수 있도록 해야한다는 것을 카오스 엔지니어링(Chaos engineering)이라 한다.

특징 4. Continuous deployments



: CI/CD와 같은 배포 파이프라인을 특징으로 들 수 있는데, 이는 지속적인 통합 혹은 지속적인 배포라 할 수 있는데, Cloud Native Application은 수십~수백개이상의 MSA로 도메인이 개발되어 제공될 수 있는데, 이처럼 하나의 애플리케이션을 구성하는 수많은 서비스(MSA)를 수동으로 하나하나 빌드하고 배포하는것은 쉽지 않고 많은 비용이 요구된다.

그렇기에 이런 빌드와 배포에 대해 자동화 시스템을 구축하여 통합으로 다른 작업으로 파이프라인을 연결해두면 작은변화 혹은 큰 변화에도 빠른 적용을 할 수 있을 것이다.

▼ Cloud Native Architecture

Cloud Native Architecture

😒 2010년대 이후부터는 IT System은 Enterprise 혹은 Cloud Native Architecture 형태로 변경되어 왔다. 즉 기존에 로컬 혹은 IDC에 구축되어 있던 시스템들을 클라우드 환경으로 전환을 하고 있다는 것인데, 이처럼 Cloud Native Architecture형태로 전환하기 위해 어떤 아키텍처를 가져야 할까? 🤔

특징

- **확장 가능한 아키텍처**
  - 시스템의 수평적 확장에 유연하게 되어 더 많은 요청을 처리할 수 있게 되었다.
  - 시스템의 부하가 분산되고 고가용성을 보장하게 되었다.
  - `ScaleUp`: 하드웨어의 스펙을 업그레이드하는 것

    - `ScaleOut`: 같은 사양의 서버(or instance)를 늘리는 것
    - 클라우드 기반으로 가상의 인스턴스를 제공하는 업체로부터 서버를 대여하기에 비용의 최소화를 노릴 수 있고, 더 이상 서비스를 사용하지 않게 될 경우 서버를 반납해서 비용을 줄일 수 있게 되었다.
    - 가상 서버(컨테이너)방식이 필수적이다.
    - 모니터링 도구를 이용해 현재 사용중인 리소스 사용량을 체크할 수 있다.
- **탄력적 아키텍처**
    - 하나의 어플리케이션을 구성하는 기능들을 각각 별도의 서비스로 개발을 하고, 이러한 서비스들을 통합 - 배포를 CI/CD를 이용해 관리하며 시간을 단축한다.
    - MSA는 잘게 쪼개진 수많은 서비스들이기 때문에 각각의 책임에 맞게 경계를 잘 구분지어야 한다.
    - 각각의 서비스는 최대한 무상태(stateless) 통신 프로토콜을 지향해야 한다.
    - 서비스간에 결합도를 낮추기 위해 노력해야 한다.
    - 각각의 MSA들은 자신들이 배포될 때 자신의 위치를 등록을 해야한다.
        - *그래야 타 시스템에서 해당 시스템을 검색하고 사용할 수 있다.*
        - `Discovery Service`라는 곳에 등록(혹은 삭제)되어 관리된다.
- **장애 격리(Fault isolation)**
    - 특정 서비스에 오류가 발생하더라도 다른 서비스에 영향을 주지 않는다.
    - 오류 수정, 기능 변경 등 시스템에 변화가 생기더라도 애플리케이션 전체를 새로 배포하는것이 아닌 해당 서비스만 새로 배포하는 것으로 다른 서비스에 영향을 최소화 한다.

▼ Cloud Native Application

개요

Cloud Native Architecture 내에서 설계되고 구현되는 애플리케이션을 Cloud Native Application이라고 하는데, 이러한 애플리케이션은 다음과 같은 형태로 만들어진다.

1. Microservice

: 당연하게도 Cloud Native Application은 Microservice로 개발된다.

2. CI / CD

:CI/CD에 의해 지속적으로 통합되고, 빌드,테스트,배포라는 과정을 가지게 된다.

3. DevOps

: MSA는 문제가 발생했을 경우 바로 수정및 배포할 수 있는 형태가되는데 이를 DevOps라 한다. DevOps에서는 시스템의 기획-구현-테스트-배포라는 과정을 시스템이 종료될 때까지 반복한다.

4. Containers

: 하나의 애플리케이션을 클라우드 환경에 배포및 사용하기 위해서 컨테이너 기술을 사용한다.

CI/CD 살펴보기

CI(Continuous Intergration): 지속적인 통합

: 하나의 애플리케이션을 하나의 팀 혹은 개발자간의 협력상태일 경우 결과물을 통합하기위한 형상관리 도구일 수도 있고, 통합된 코드를 빌드하고 테스트하는 과정 자체를 의미할수도 있다. 보통 `Jenkins`, `Team CI`, `Travis CI`, `Git Action`과 같은 도구들을 `Git` 이나 `SVN`같은 형상관리 툴에 연동해서 사용할 수 있다.

해당 기능을 잘 사용하면, 개발자가 코드를 구현 및 수정하여 커밋 후 올리면 자동으로 코드 통합 및 빌드와 테스트까지 수행하여 문제점을 확인할 수도 있고, 배포까지 자동으로 가능하다.

CD: 지속적인 배포

:해당 키워드는 두 가지의 의미로 사용되는데, 소스 저장소(ex: git)에 업로드된 코드가 빌드된 배포파일을 실행환경에 배포하느냐에 따라 달라진다.

1. **Continuous Delivery**

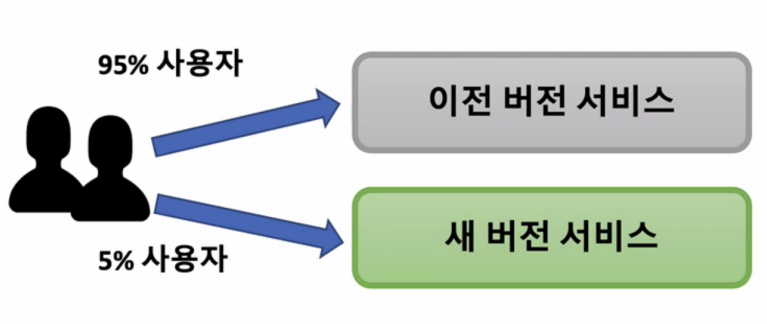
   : 실행환경에 결과물을 수작업으로 배포하는 과정이 필요하다면 Continuous Delivery라 한다.

2. **Continuous Deployment**

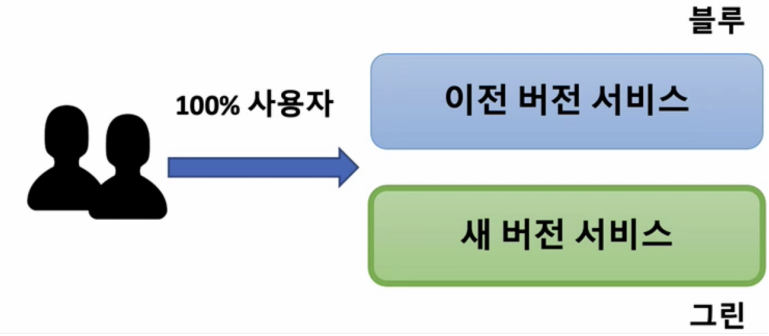
   : 완벽하게 모두 자동으로 반영되는 환경이라면 Continuous Deployment라 한다.

카나리 배포, 블루그린 배포

변경된 시스템을 무조건 다 새로 반영하지는 않는다. 기존 시스템과 같이 서비스를 제공하면서 사용자에게 최소한의 이질감혹은 문제점을 느끼게 해야하는데, 그러기 위해서 **카나리 배포** 혹은 **블루그린 배포**라는 전략을 선택할 수 있다.

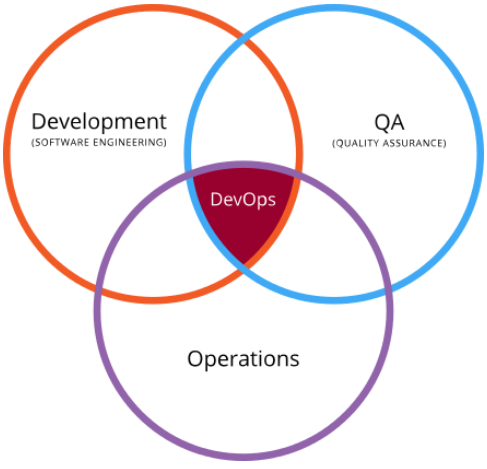


카나리 배포



블루그린 배포

DevOps



DevOps

💡 소프트웨어의 개발(Development)와 운영(Operations)의 합성어로 개발조직과 운영조직의 통합을 의미한다.

: 고객의 요구사항을 최대한 빠르게 반영하고 해결책을 제시하는 목적을 가진다.

시스템의 개발은 결국 고객에게 문제없는 시스템을 제공해주기 위해서이다. 그러기 위해서는 고객의 요구사항 혹은 기획 변경에 대해서도 유연하고 빠르게 반영할 수 있어야 하는데 그러기 위해선 계속해서 는 다음과 같이 와료 일정까지 반복되야 하는데

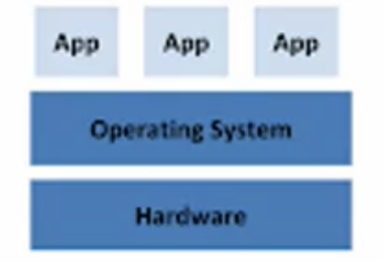
👉 `요구사항 분석(=도메인 분석) → 기능 구현 → 빌드 → 테스트 → 반영`

이를 **DevOps**라 한다. MSA 에서는 애플리케이션을 기능별로 잘게 쪼개어 개발하기에 이러한 사이클로 잦은 기능 변경과 빌드및 테스트에도 강력한 모습을 보여주기에 적절하다 할 수 있다.

Container 가상화

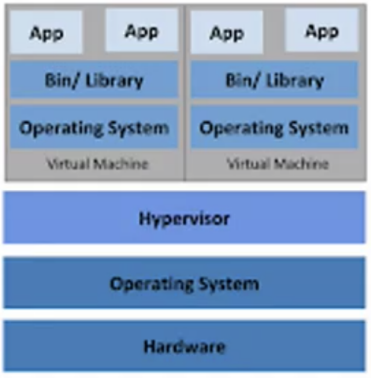
가상화는 Cloud Native Architecture의 핵심 개념중 하나로, 서비스를 클라우드 환경으로 이전해서 저비용 고가용성을 가지는 시스템을 구축하도록 해주는 기술이 바로 이 **Container 가상화**다. 그럼 과거 레거시한 배포 방식부터 현재 컨테이너 가상화 배포 방식까지 살펴보면서 Container 가상화를 알아보자.

1. Traditional Deployment



: 기존에 전통적인 방식에서는 하드웨어위에 운영체제, App을 올려 운영을 했었다.

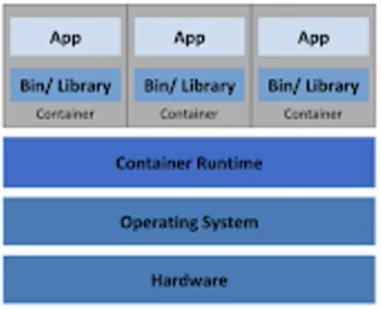
2. Vitualized Deployment



: 기존의 구조에서 운영체제 위에 Hypervisor라는 기술을 통해 가상머신을 기동하고 이런 가상머신은 호스트 시스템이 가진 물리적인 리소스를 쪼개어 사용하게되어 각각의 가상머신이 독립적으로 운영될 수 있다.

하지만, 이 방식은 호스트 운영체제에 큰 부하를 줄 수 있고 시스템 확장에도 한계를 가질 수 밖에 없다.

3. Container Deployment



: Container 가상화 기술을 이용해 시스템을 구축할 수 있는데, 운영체제 위에 Container 가상화를 구동하기 위한 SW를 구동하고, 해당 SW에서는 공통적인 리소스나 라이브러리는 공유하고 각자 필요한 부분만 독립적인 영역에 실행할 수 있는 구조로, 기존의 하드웨어 가상화 기술보다 적은 비용이 요구되고 각각의 서비스들을 가볍고 빠르게 운영될 수 있다.