







서버 가상화와 컨테이너의 이해

이병호

2023.01.17

CONTENTS

-  PART 1. PR
-  PART 2. 가상화란?
-  PART 3. 서버 가상화
-  PART 4. 컨테이너
-  PART 5. 퍼블릭 클라우드
-  PART 6. 클라우드 네이티브

PART 1.

PR

PR

- 웹에이전시 - JSP, PHP, ASP
- SI/SM - On-premise
- 프리랜서 - 기획 , 일정관리
- 연구원 - 빅데이터 기반 AI 연구
- DevOps - 클라우드 네이티브



PR



PART 2.

가상화란?

가상화란?

- 정의
 - 사전적 : 실체가 없는 것을 마치 존재하는 것처럼 보이게 하는 기술
- 종류
 - VM (CPU, RAM, storage, network, etc)
 - Network virtualization (SDN , NFV)
 - Storage virtualization
 - Desktop virtualization (VDI ,Daas)
 - Application virtualization
 - Data virtualization
- 일단 virtual 붙으면 다 가상화?
 - Virtualization
 - Containerization

단점

- 성능저하
 - 리소스 분배를 위한 중간단계 필요
 - 리소스 리미트 시 전체적 성능하락
 - 고가용성을 위한 전문 인력의 관리 포인트 필요
- 비용
 - 초기 도입 비용
 - 사용량에 따른 비용 증가
 - 비용증가에 따른 유지보수 필요
 - 전문인력 부족

장점

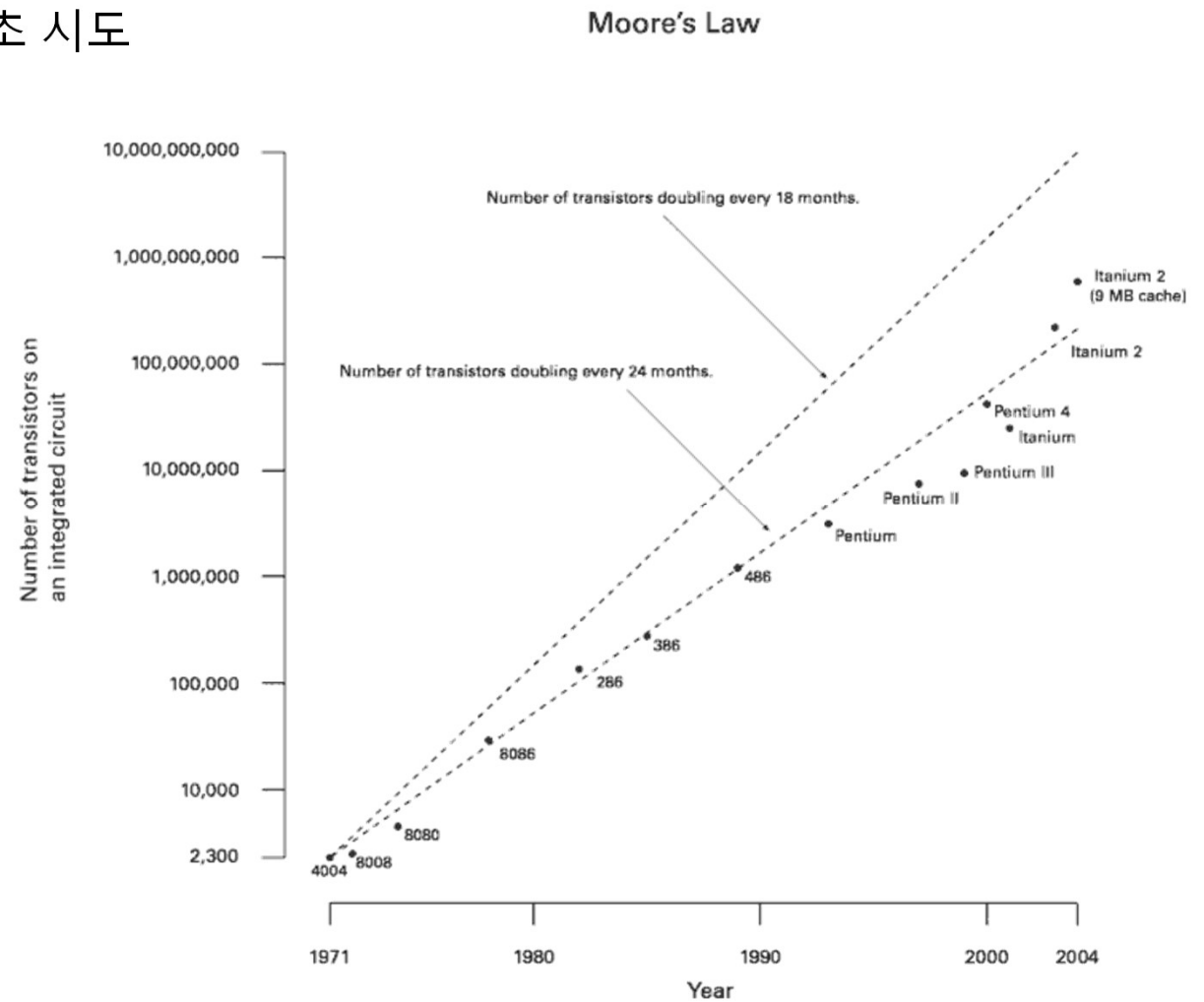
- 유연성 - Flexibility
 - 환경 변화에 대한 빠른 반응
 - ScaleUp, ScaleOut
- 가용성 - Availability
 - 무중단 서비스를 위한 인프라 설계
 - IaC 를 통한 빠른 프로비저닝
 - 클라우드 네이티브
- 민첩성 - Agility
 - 물리적 시간 감소
 - Downtime 감소
- 효율성 - Efficiency
 - 비용감소

PART 3.

서버 가상화

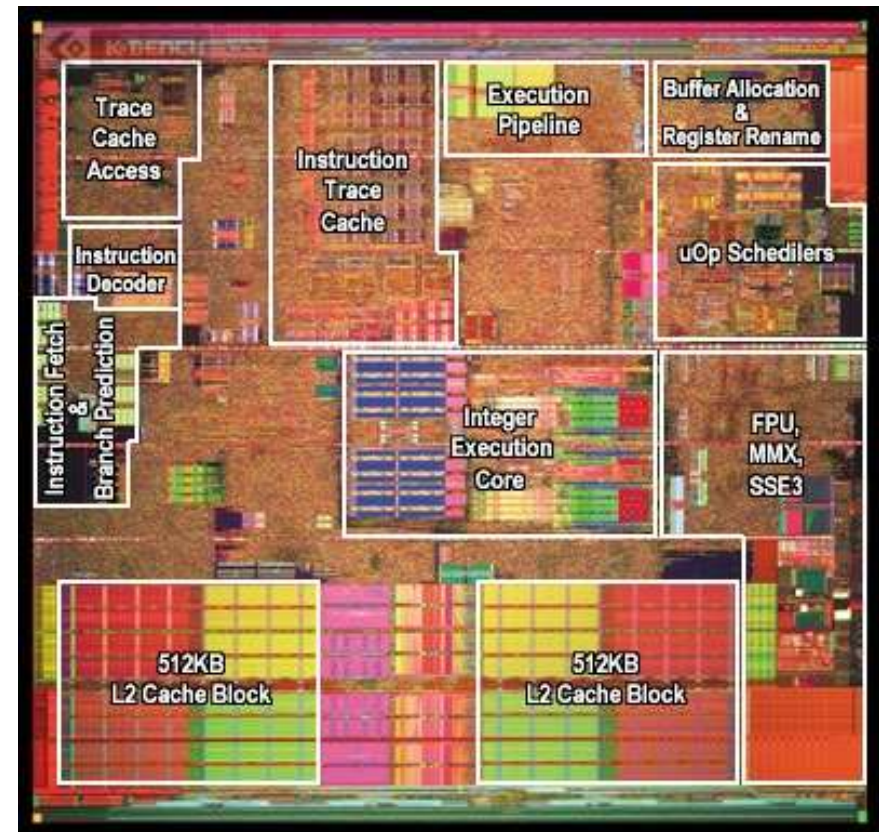
서막

- 가상화의 시작
 - 1964년 IBM 메인프레임에서 최초 시도
 - 무어의 법칙
 - 서버 : 어플리케이션 = 1:1
 - 서버 성능 최대 활용 방안



CPU

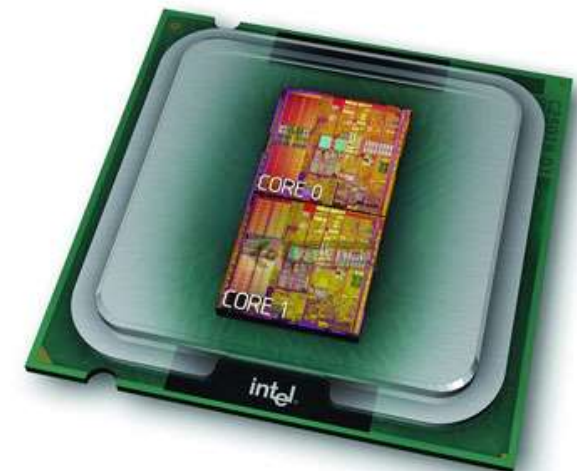
- 싱글코어
 - 클럭 : 1초에 몇 번 작동 하는가 (~ 3.8GHz)
 - 공정의 소형화 (~ 65nm)
 - 폴락의 법칙에 의해 한계에 달함



CPU

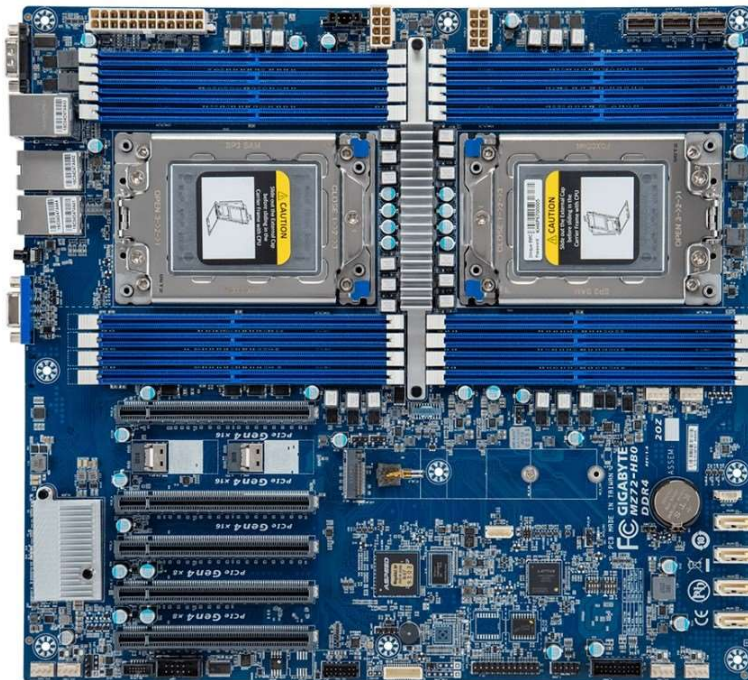
- 듀얼코어

- 2005년 등장 (intel 스미스필드)
- 무어의 법칙
- 서버 : 어플리케이션 = 1:1
- 서버 성능 최대 활용 방안



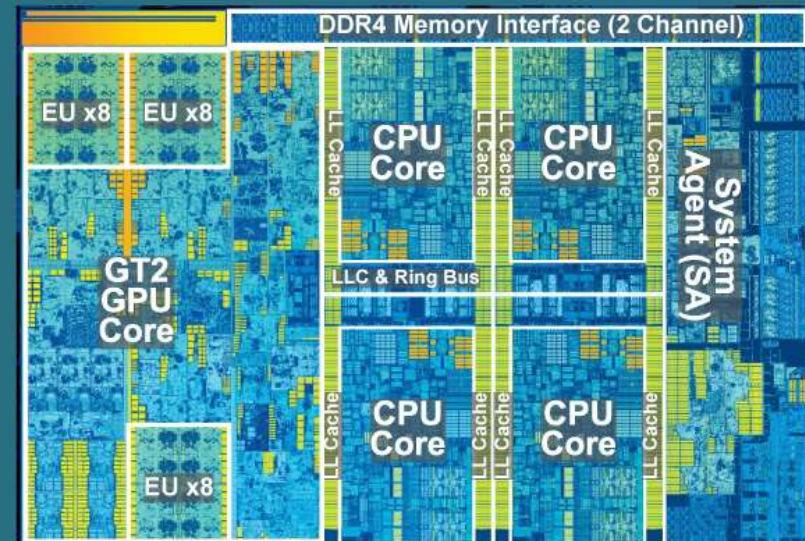
CPU

- 멀티코어
 - 64코어
 - 5nm
 - 5.8GHz
 - 서버는 2소켓
 - Simultaneous multithreading



Skylake Die Layout

Skylake 14nm




4 CPU cores
GPU core (GT2)
8MB LL Cache

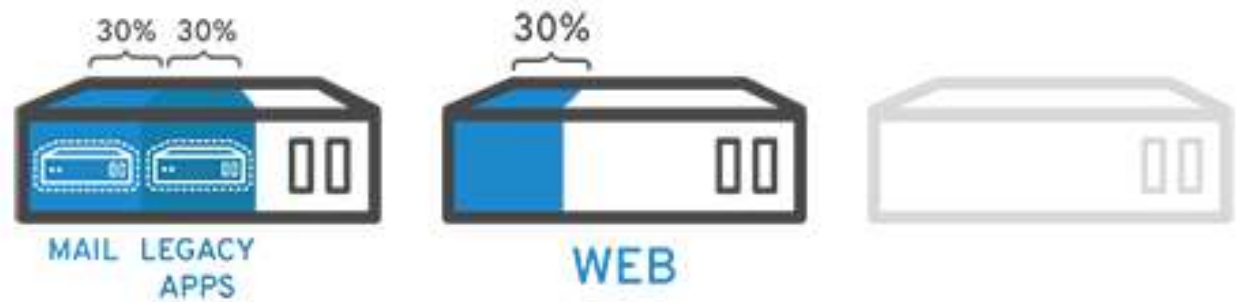
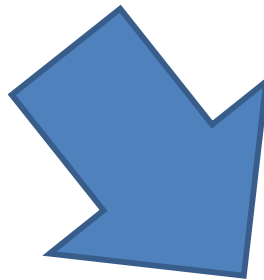
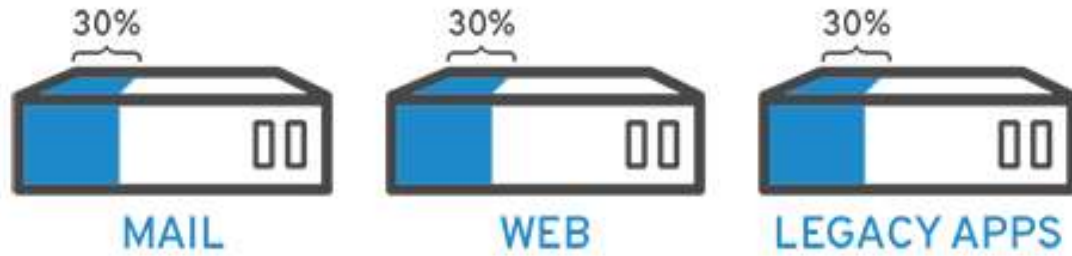
14 nm Process
-122 mm²

CPU

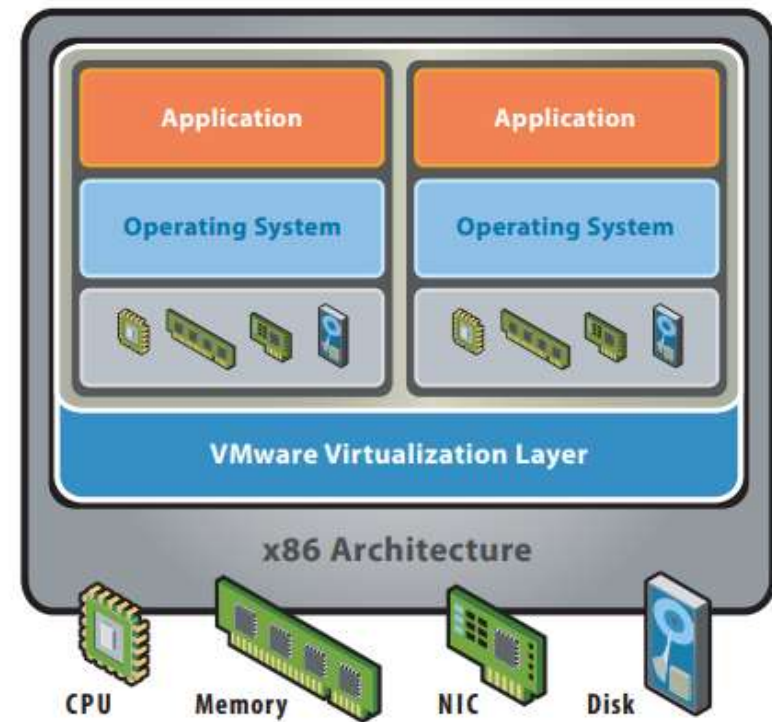
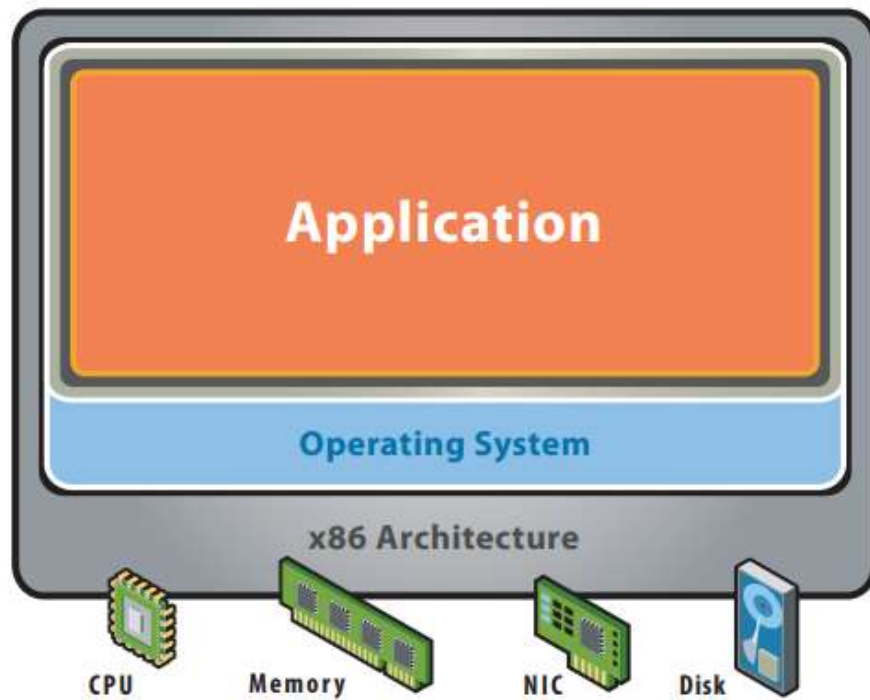


Pad. 

자원의 효율화



Hypervisor



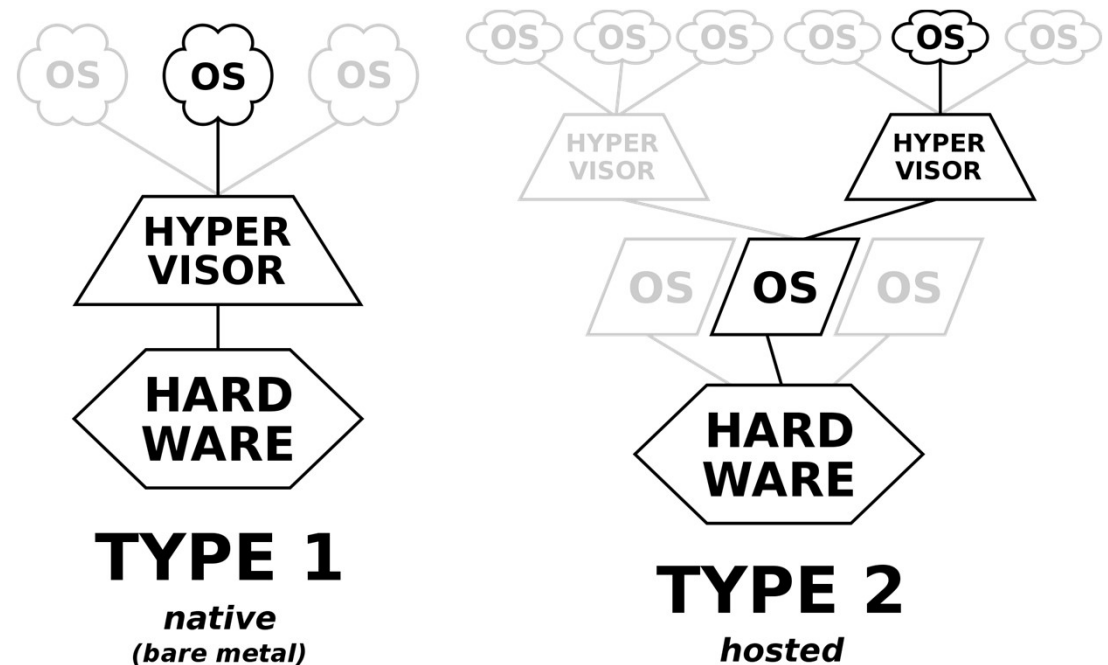
Hypervisor

- TYPE1 - Native (Bare metal)

- Xen
- vSphere
- Hyper-V
- KVM
- OVM

- TYPE2 - Hosted

- Hyper-V
- VMware Workstation Player
- VMware Horizon
- VMware Fusion
- Virtual Box
- Parallels Desktop
- QEMU
- Open Stack



PART 4.

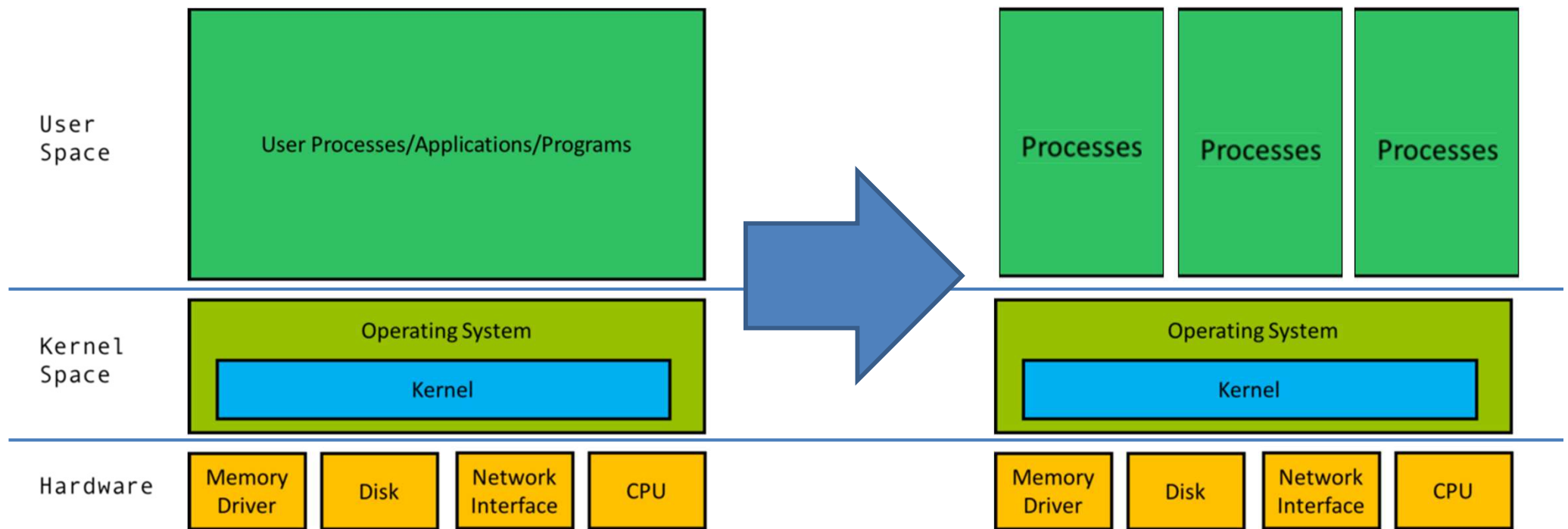
컨테이너

서막

- 컨테이너의 시작
 - 1979년 chroot
 - 2000년 FreeBSD Jail OS 컨테이너화
 - 2003년 google Borg 사용
 - 2008년 LCX
 - 2013년 docker
 - 2015년 kubernetes
- 기술적 정의
 - 시스템/애플리케이션의 가상화
 - 커널을 컨트롤 할 수 있는 어플리케이션
 - 프로세스의 격리

격리

- 컨테이너 기술의 근간
 - 프로세스간 리소스 및 데이터의 격리



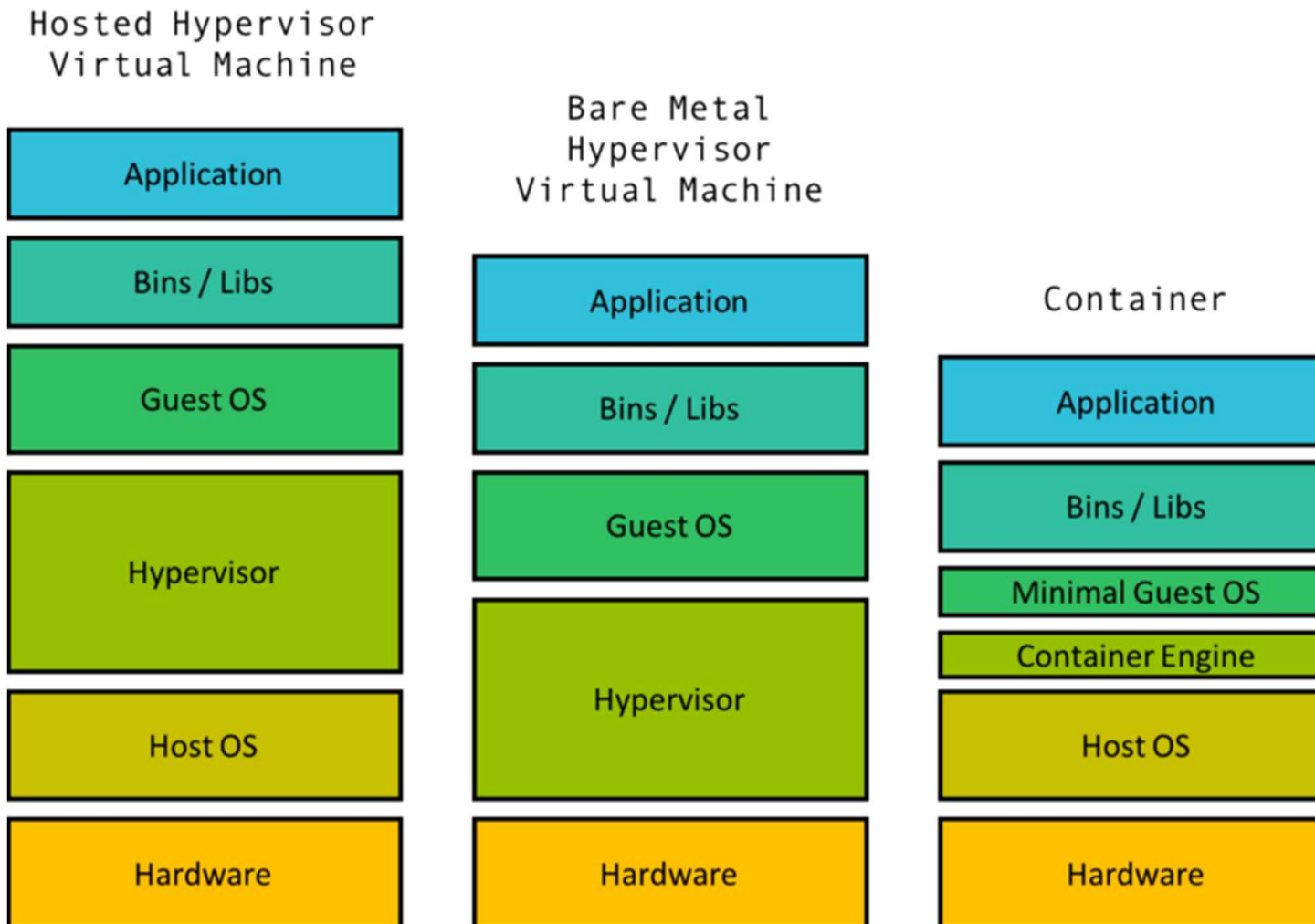
커널

- Cgroups(Control Groups)
 - Cpu
 - Memory
 - I/O
 - Network
 - device 노드
- Namespaces
 - mnt : file system
 - pid : process
 - net : network
 - ipc : 프로세스 간 통신통로
 - uts : 독립적인 hostname 할당
 - user : 독립적 사용자 할당

장점

- 적은 오버헤드
 - 기존 하이퍼바이저 방식 보다 적은 오버헤드

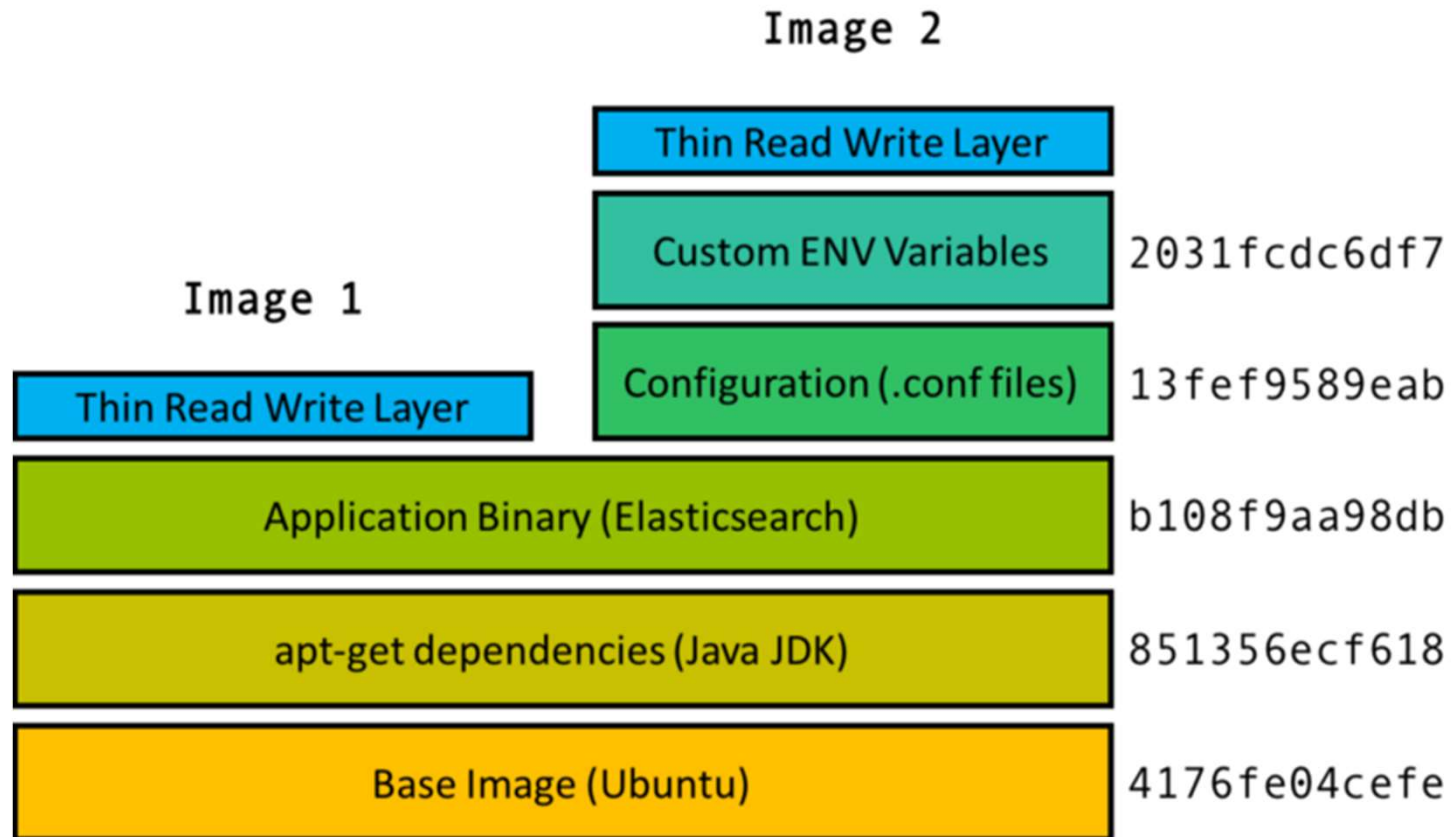
장점



장점

- 적은 오버헤드
 - 기존 하이퍼바이저 방식 보다 적은 오버헤드
- 높은 집적도
 - 자원 분배에 있어 고밀도화 가능
 - CPU와 MEMORY만 필요하기에 낮은 사양에서 실행 가능
- 빠른 실행
 - OS를 거치지 않기때문에 빠른 실행 가능
- 데이터 처리방식
 - Container에 직접 저장 하지 않아서 재사용 용이
- 이미지 경량화
 - 차분관리에 의한 이미지 경량화

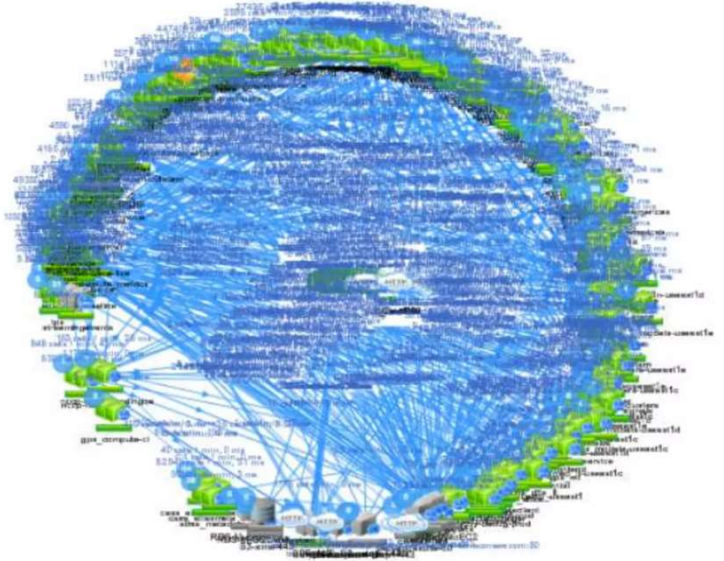
장점



단점

- 복잡도 향상
 - 서비스 관리 포인트 증가
- 새로운 보안이슈
 - 격리 기능은 정말 안전한가?

Netflix & Micro-Services



Meltdown

Meltdown breaks the most fundamental isolation between user applications and the operating system. This attack allows a program to access the memory, and thus also the secrets, of other programs and the operating system.

If your computer has a vulnerable processor and runs an unpatched operating system, it is not safe to work with sensitive information without the chance of leaking the information. This applies both to personal computers as well as cloud infrastructure. Luckily, there are [software patches](#) against Meltdown.

Docker

- Docker
 - 사실상 컨테이너의 기준
- Docker Engine
 - Docker Server
 - Docker REST API
 - Docker CLI
- Docker Server
 - Persistent Storage
 - Containerd
 - Networking

Kubernetes

- 대규모 환경을 위한 준비
 - 컨테이너 시스템의 클러스터 관리
 - 분산시스템을 위한 기능
- Google
 - Google 의 증명
 - 2003년 부터 내재화
- 효율적인 자동화
 - 컨테이너 라이프 사이클을 활용 한 자동화
 - 자동 배포,롤아웃,롤백,오토 스케일링
 - 셀프힐링
 - 서비스 디스커버리, 로드 밸런싱

PART 5.

퍼블릭 클라우드

차이점

- 가상화 ≠ 클라우드

	가상화	클라우드
정의	기술	방법
목적	1개의 물리적 하드웨어에 다수의 가상환경	온디맨드 사용을 위한 가상 리소스 풀링과 자동화
용도	특정 용도의 패키징된 리소스를 특정 사용자에게 제공	다양한 용도의 다양한 리소스를 사용자 그룹에게 제공
설정	이미지 기반	템플릿 기반
확장성	스케일업	스케일아웃
워크로드	Stateful	Stateless
테넌시	싱글 테넌트	멀티플 테넌트

서비스 비교

	AWS	Azure	GCP
Analytics	Athena	Azure Synapse Analytics Azure Data Lake Analytics	BigQuery
Application Integration	Simple Queue Service	Azure Queue Storage	Cloud Pub/Sub, Cloud Tasks
Compute	Amazon EC2	Azure Virtual Machines	Compute Engine
network	Virtual Private Cloud	Azure Virtual Network	Virtual Private Cloud
Kubernetes	Elastic Kubernetes Service	Azure Kubernetes Service	Google Kubernetes Engine
Database	Aurora	Azure SQL Database Hyperscale	AlloyDB for PostgreSQL
Machine Learning	SageMaker	Azure Machine Learning	Vertex AI
Monitoring	CloudWatch	Azure Monitor	Cloud Monitoring
Managemnet	AWS CloudFormation	Azure Resource Manager	Cloud Deployment Manager

<https://sharpLee7.tistory.com/152>

PART 6.

클라우드 네이티브

Cloud Native

- 의미

- 2015년 리눅스에서 CNCF 재단을 설립
- 클라우드의 이점을 최대한 활용한 애플리케이션 구축과 실행
- <https://github.com/cncf/toc/blob/main/DEFINITION.md>

클라우드 네이티브 기술은 조직이 퍼블릭, 프라이빗, 그리고 하이브리드 클라우드와 같은 현대적이고 동적인 환경에서 확장 가능한 애플리케이션을 개발하고 실행할 수 있게 해준다. 컨테이너, 서비스 메쉬, 마이크로서비스, 불변(Immutable) 인프라, 그리고 선언형(Declarative) API가 이러한 접근 방식의 예시들이다.

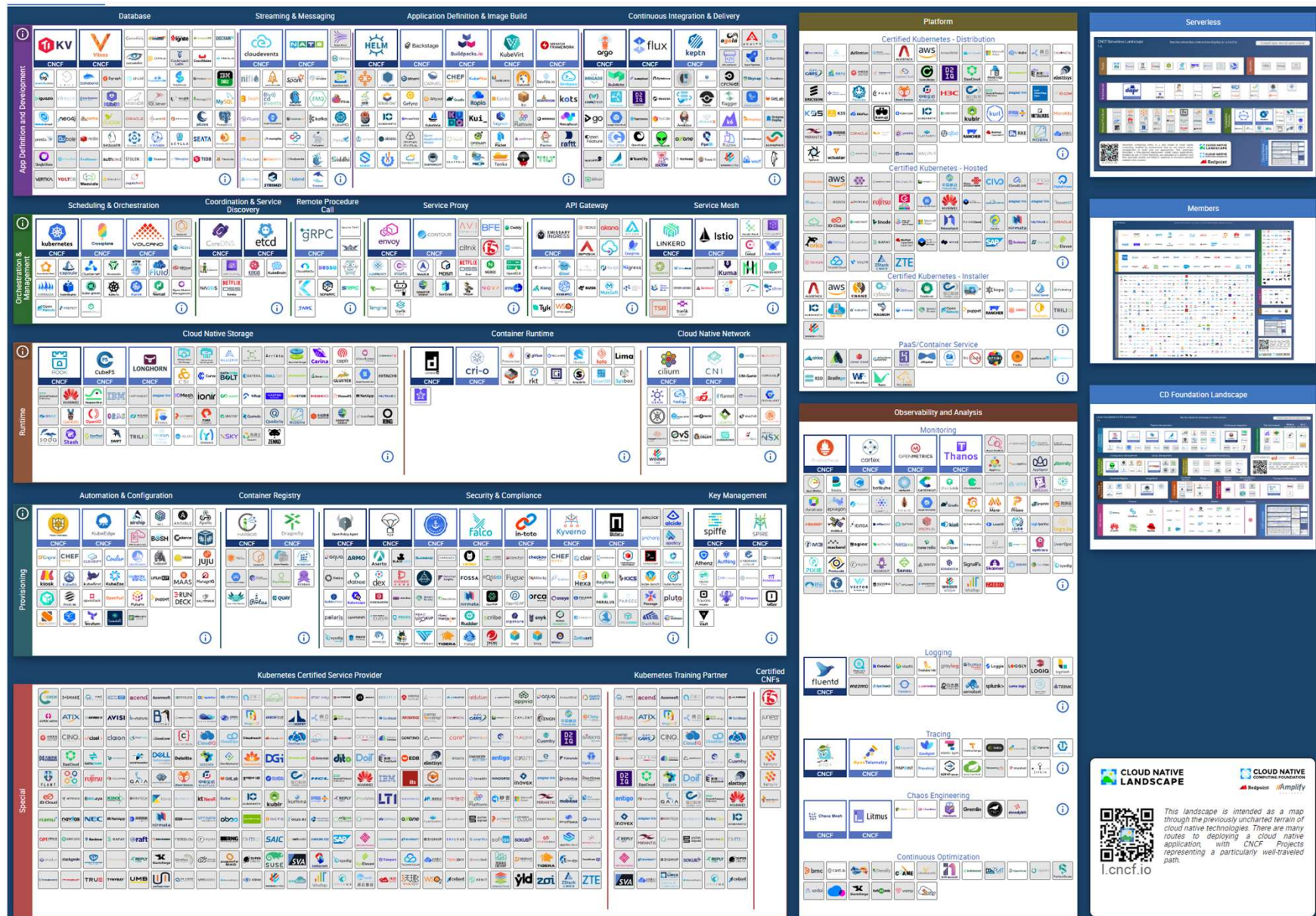
이 기술은 회복성, 관리 편의성, 가시성을 갖춘 느슨하게 결합된 시스템을 가능하게 한다. 견고한 자동화 기능을 함께 사용하면, 엔지니어는 영향이 큰 변경을 최소한의 노력으로 자주, 예측 가능하게 수행할 수 있다.

Cloud Native Computing Foundation은 벤더 중립적인 오픈 소스 프로젝트 생태계를 육성하고 유지함으로써 해당 패러다임 채택을 촉진한다. 우리 재단은 최신 기술 수준의 패턴을 대중화하여 이런 혁신을 누구나 접근 가능하도록 한다.

- 요소

- DevOps : 개발-운영 간의 협업 프로세스
- CI/CD : 지속적인 통합, 지속적인 배포
- Container : 빠른 가상화 기술
- MSA : 확장성, 생산성, 안전성을 고려한 아키텍처

<https://landscape.cncf.io/>



참고문헌

- <https://namu.wiki/w/멀티코어%20프로세서>
- <http://www.opennaru.com/cloud/virtualization-vs-container/>
- <https://en.wikipedia.org/wiki/Hypervisor>
- <https://www.freecodecamp.org/news/demystifying-containers-101-a-deep-dive-into-container-technology-for-beginners-d7b60d8511c1>
- <https://www.redhat.com/ko/topics/cloud-computing/cloud-vs-virtualization>

감사합니다.

이병호(neo)