CLOUD NATIVE COMPUTING FOUNDATION

글로벌 클라우드 플랫폼의 최대 점유율 3사 : amazon aws, ms azure, google cloud

용어 설명과 정리

소프트웨어 : 하드웨어의 기능을 원할하게 수행하기 위한 명령어들의 집합.

플랫폼/프레임워크 : 소프트웨어를 작동시키는 환경이나 도구. 혹은 기반 기술이 있는 소프트웨어.

네트워크 : 전송 매체를 통해 데이터를 교환하는 시스템의 모음. 다수의 시스템을 전송 매체로 연결해 구성한 시스템들의 집합체.

프론트엔드 : 주문(데이터) 전달. 음식점 카운터.

백엔드 : 만든 음식(처리된 데이터)을 전달. 음식점 주방.

프로토콜 : 컴퓨터 네트워크에서 데이터를 주고받을 때 수행되는 절차.

cloud

클라우드 컴퓨팅 : 컴퓨터 리소스를 데이터센터에 대량으로 집적시킨 후, 개별 이용자가 요구하는 만큼 가상으로 분리하여 정보통신망을 통해 제공하는 서비스. 사용량에 비례하여 비용 청구.

클라우드 특징 :   
주문형 셀프 서비스(사업자와 직접 상호 작용X, 사용자 개별 관리화면을 통해 이용)  
광범위한 네트워크 접속(다양한 디바이스를 통해 서비스에 접속)  
리소스 공유  
신속한 확장성(필요에 따라 스케일 업/다운)  
측정 가능한 서비스(종량제)

클라우드 유용성 :   
경제성, 유연성, 가용성(장애 대응), 빠른 구축 속도

클라우드 발전

엣지 컴퓨팅 : 데이터가 발생한 현장 혹은 근거리(중간 플랫폼)에서 실시간 처리하는 방식으로 데이터 흐름 가속화를 지원하는 컴퓨팅 기술. 기존의 중앙 데이터 처리 방식의 한계를 보완.

엣지 컴퓨팅 활용 : 자율주행 자동차, 증강 가상 현실, 산업 현장, 로봇, 드론

클라우드 컴퓨팅 트렌드

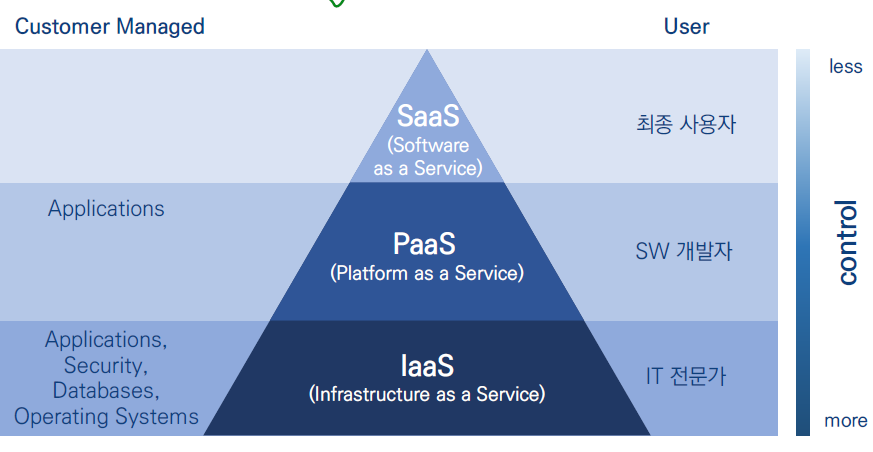
멀티 클라우드 수요 증가 : 한 플랫폼에만 종속되면 휘둘리니까 이전하고 나눠 쓰고 골라 쓰는 것이 가능해져 멀티 클라우드 수요 매우 증가.

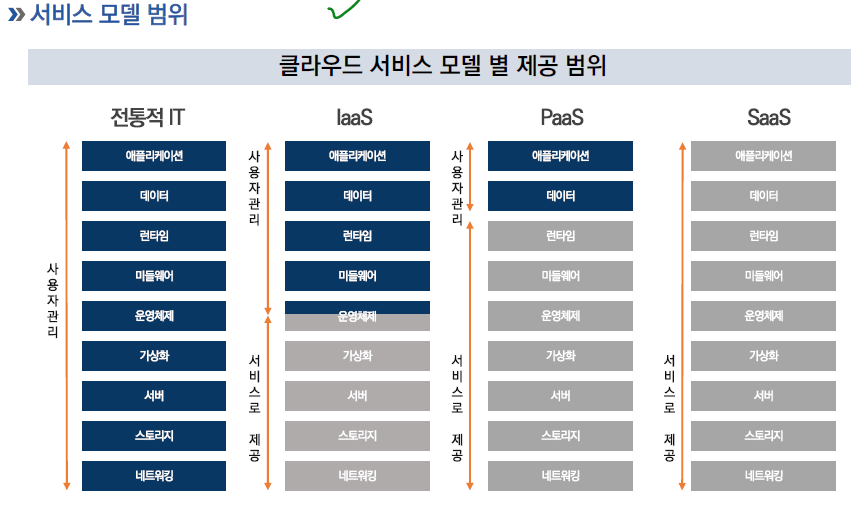
AI/ML 기반 클라우드 : ai 인프라를 자체적으로 구축할 수 있는 기업이 거의 없기에.

ML ops : 기계학습에 대한 지식과 경험을 토대로 비즈니스 프로세스 자동화와 관리를 담당. DevOps + ML 인력 역량

AI ops : 사람의 개입 없이 스스로 IT이슈를 해결. MLOps + AI기술.

클라우드 모델







클라우드 배치에 따른 모델

퍼블릭 클라우드 : 서비스 제공업체가 구축한 IT 인프라를 기업들이 사용료를 내고 이용하는 방식. 불특정 다수 대상.

프라이빗 클라우드 : 기업 자체적으로 데이터센터 안에 클라우드 환경을 구축해 사용하는 방식. 인트라넷을 통한 단일 조직 대상.

하이브리드 클라우드 : 프라이빗 클라우드(온 프레미스) + 퍼블릭 클라우드

클라우드 모델 별 특징

퍼블릭 클라우드 특 : 비용 절감, 유지관리 필요X, 높은 안정성, 무제한에 가까운 확장성.

프라이빗 클라우드 특 : 특정 비즈니스를 위한 유연성 향상, 높은 확장성, 보안 강화.

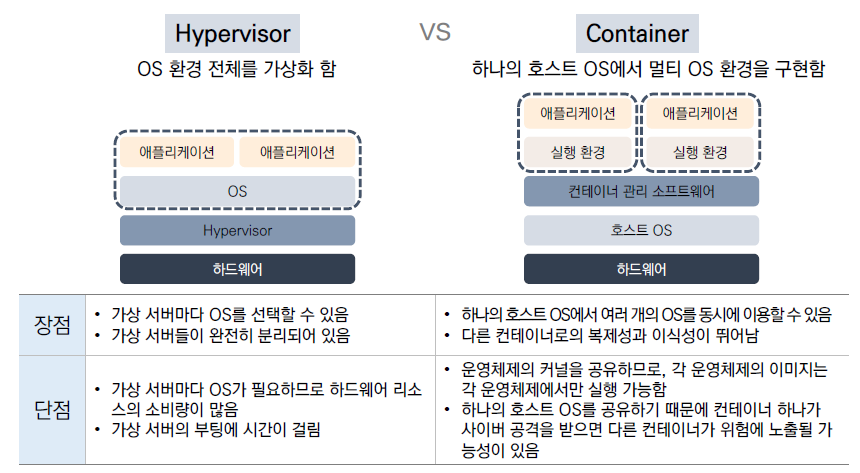
하이브리드 클라우드 특 : 유연성(필요시 퍼블리 클라우드의 추가적 리소스 활용, 정책 조정 가능), 비용 효율성(퍼블릭 클라우드의 규모 조정과 온 디맨드), 용이성(부담 없이 클라우드 전환 가능), 제어(프라이빗 클라우드를 통해 중요 데이터 직접 유지 관리)

멀티 클라우드 : 공급업체 종속으로 인한 리스크 대응 가능, 최신기술 도입 편함, 서비스 개선 편함, 가격경쟁력 확보, 전략에 따라 유연함.

IaaS

Infrastructure 자원을 가상화된 서비스로 사용자에게 제공하는 클라우드 서비스.

표준화된 HW의 설치 및 구성을 자동화함. SW관련해서는 TA전문가가 설치, 구성해야함.



public IaaS : aws, ms azure, google cloud, Alibaba cloud, oracle cloud, IBM cloud

private IaaS : openstack, apachecloudstack, nutanix

hybrid IaaS : AWS outposts, azure stack, google cloud Anthos

PaaS

CSP가 언어와 개발 환경을 포함한 플랫폼 기능을 제공하여 사용자가 앱을 배포, 관리, 실행할 수 있는 클라우드 서비스.

응용 SW 개발에 필요한 프레임워크, 미들웨어, OS등 HW뿐만 아니라 표준화된 SW의 설치 및 구성을 자동화함.

미들웨어, OS, 개발 프레임워크 성격을 띄며, 컨테이너를 기반으로 SW 플랫폼 서비스 제공. APP Engine, K-PaaS, Pivotal.

IaaS기반에서 SaaS개발 시의 문제점을 해결 가능.

표준화된 HW, SW의 설치, 구성을 자동화하여 신속한 개발, 테스트 가능.

개발을 위한 플랫폼 구축 필요 없이 웹에서 쉽게 빌려 쓸 수 있음.

미리 환경이 구축되어 있어 개발에서 배포까지 라이프 사이클이 짧아 DevOps문화를 적용하기 용이함.

SaaS

CSP가 사용자가 필요로 하는 앱을 가상화된 서비스로 제공하는 클라우드 서비스.

IaaS, PaaS 환경 위에서 SW 및 관련 데이터를 중앙에서 관리하고 인터넷을 통해 사용자에게 SW를 이용할 수 있도록 하는 모델. 기능과 지불 온 디맨드.

환경설정을 통한 맞춤화(configuration) – 수정 없이 사용자 요구사항 수용 지원.

다중 사용자 지원(multi-tenancy) – 다수 사용자가 공유하여 사용하는 아키텍처 지원.

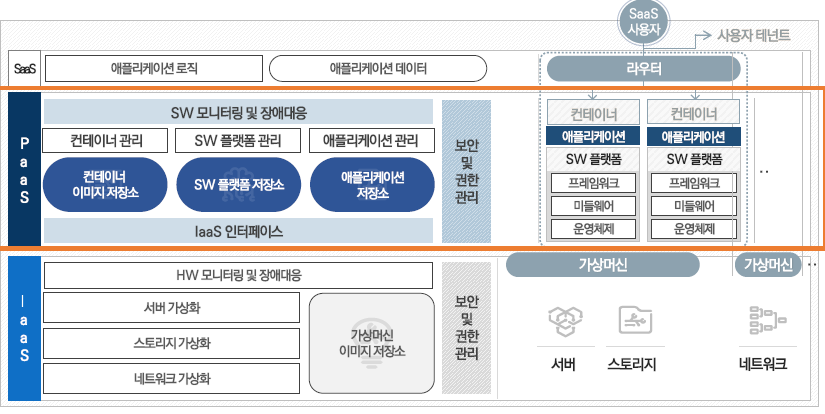
확장성(scalability) – 가용성 및 성능 지원을 위해 다수 인스턴스 생성, 데이터 분산 관리, 가상화, 분상병렬 처리 등을 해서 확장성 있는 서비스 제공.

SaaS 레벨3 성숙도(multi tenant) – 모든 고객을 하나의 인스턴스로 지원. 메타데이터를 통해 커스터마이징.

SaaS 레벨4 성숙도(scalable multi tenant) – 부하분산 시스템에서 모든 고객을 지원. 데이터 분산 관리.

별도 설치 없이 바로 사용가능, 구독형, 이용 규모와 기간이 고정X, 데이터가 클라우드에 저장돼 보안성이 높고 접근이 자유로움, 업데이트와 버전업 시간X, 유지 관리에 소요되는 리소스와 비용X.

클라우드 서비스 구성



공인 IP : ISP가 부여 받아 사용. 전세계적으로 유일한 IP.

사설 IP : 공인 IP 개수 모자름 보완. 내부적으로 사용하는 내부에서 유일한 IP.

Gateway : 현재 사용자가 위치한 네트워크에서 다른 네트워크로 이동하기 위해 반드시 거쳐야 하는 거점.

NAT : Network Address Translation – 네트워크 주소 번역.

DHCP : Dynamic Host Configuration Protocol – 네트워크 주소의 자동 할당.

SSH : Secure Shell – 원격지 호스트에 접속하기 위해 사용되는 패킷 암호화 기법이 사용된 통신 프로토콜.

API : Application Programming Interface – 어떤 서버의 특정한 부분에 접속해서 그 안에 있는 데이터와 서비스를 이용할 수 있게 해주는 소프트웨어 도구. 앱과 OS간, 앱과 프레임워크간 기능을 제어할 수 있게 만든 인터페이스. 기계, 운영체제 등 과 독립됨.

SOAP : simple object access protocol – 네트워크를 통해서 컴퓨팅 자원 간 통신을 가능하게 해주는 프로토콜. 보안 수준이 엄격하며 일관적인 데이터 전송 및 성공/반복 실행 로직이 규정돼 있기 때문에 통신의 신뢰성을 보장. 완벽한 규격화 지향.

REST : representational state transfer – 웹 서비스와 모바일 앱 경량의 필요에 맞춘 아키텍처 원칙 세트이며 설계적 지침. 인간과 기계 친화적. http 프로토콜을 기반으로 웹 최적화. 고유 URL을 통해 직접 바로 전송하기 때문에 특정 개발환경 셋팅 X. URL로 리퀘스트, JSON, XML 등 데이터 포맷 파일로 응답 받음.

데이터 포맷 : json(JS object notation. 키-값 쌍) bson(Binary JSON) xml yaml(데이터 직렬화 방식, 문법 쉬움, 고급 컴퓨터 언어에 적합)

SOAP vs REST



스토리지 가상화 : 물리적인 저장장치를 논리적으로 나누거나 결합시켜 사용률을 극대화 시키는 추상화기술. RAID된 장치를 가상화.

스토리지 가상화 특징 :   
저장 디바이스의 복잡한 원리 등 실제 구조를 숨김.  
여러 개의 디스크 드라이브를 그룹화하여 하나의 가상 드라이브로 통합 운용 가능.  
일부 디스크에 장애가 발생해도 서비스 운용 계속 가능.  
사용 중에도 디스크 용량의 재할당 가능.

IaaS 스토리지 가상화 종류 : File storage, Block storage, Object storage

File Storage :

폴더/디렉토리 단위 구조를 통해 계층 구조에 데이터를 파일 형태로 저장하는 방식(예 NAS).

파일들은 제한적인 메타데이터를 갖고 있어 파일이 늘어나면 데이터가 늘어나고 파일을 찾는데 문제 발생 가능. 검색이 느림.

운영체제와 파일시스템은 종속 되어있음.

Block Storage :

데이터를 고정된 크기의 블록으로 처리하여 저장하는 방식(예 SAN).

각각의 블록은 고유한 주소를 가지고 이를 통해 블록을 재구성하여 데이터를 불러옴.

계층구조X, 매핑하는 구조이므로 데이터에 빠르게 접근 가능.

파티션 분할 등이 가능하여 운영체제와 독립적이므로 대규모 DB운영에 적합.

구축 비용이 많이 들고 메타데이터 제한적이어 관리자의 부담.

Object Storage :

각각의 데이터 단위로 저장되는 데이터 저장소 유형. 파일 및 블록 스토리지의 단점을 극복.

파일 유형별로 객체화하고 색인하는 테이블이 존재, 키가 객체의 이름.

계층구조X, 평면 구조로 데이터를 저장하고 키를 통해 색인 테이블을 참조하여 해당 데이터에 접근.

객체를 수정하기 힘들기에 덮어쓰는 방식.

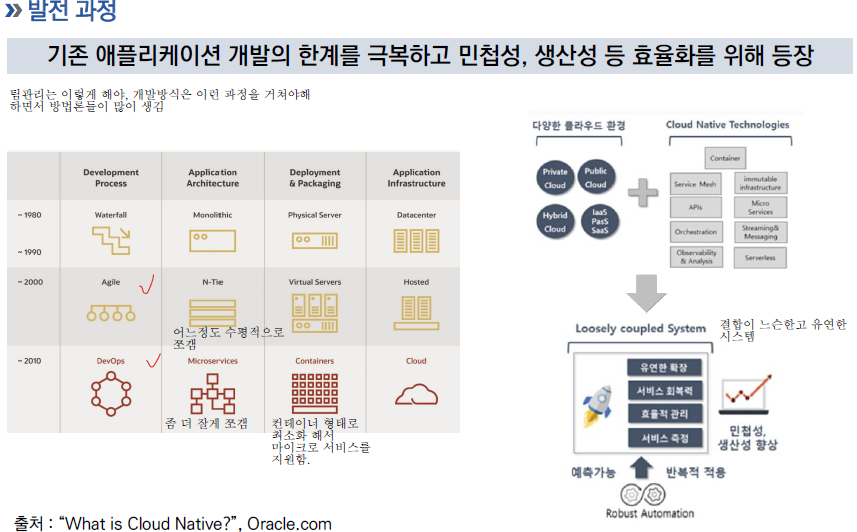
자주 변경되는 데이터에는 맞지 않고 수정이 잘 일어나지 않는 데이터에 적합.

관리자 부담 때문에 파일 및 객체 스토리지를 많이 사용.

Cloud Native : 방식. 클라우드의 이점을 최대로 활용할 수 있도록 앱을 구축하고 실행하는 방식.

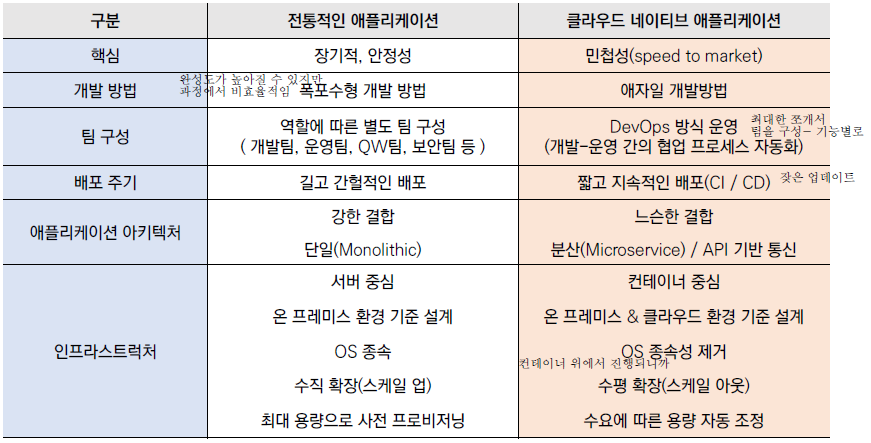
Cloud Native 조건 :   
퍼블릭, 프라이빗, 하이브리드 클라우드 환경에서 확정성 있는 앱을 만들고 운영할 수 있다.  
컨테이너, 서비스 메시(연계된 서비스), 마이크로 서비스, 불변의 인프라, 미리 정의된 API가 전형적인 접근 방식에 해당.  
회복성이 있고, 관리 편의성을 제공, 가시성을 갖는 느슨하게 결합된 시스템(독립적이고 결합성이 낮아 부분 탈부착이 가능한 시스템)을 사용할 수 있다.  
견고한 자동화와 함께 사용한다면, 엔지니어는 최소한의 수고로 영향력이 크고 예측 가능한 변경 가능.

Cloud Native의 핵심요소 :   
MSA(서비스 안정성과 스케일링 용이성), Container(작은 서비스 단위 통해 IT이식성과 유연성), DevOps(MSA맞게 작게 구현 운영 테스트 자동화, 개선속도 증가), CI/CD(자동으로 계속 통합, 모니터링 개발-운영 업무 속도 증가)

Cloud Native까지의 발전 과정  
  
CNCF : 쿠버네티스를 기반으로 컨테이너 앱을 운용할 대 필요한 다양한 개발자 도구들을 공동 개발하는 리눅스 재단의 오픈소스 단체.  
CNCF 하는 일 :   
CNCF Open Source Projects(성숙도에 따라 나누어 관리)  
CNCF Landscape(아키텍처 영역별로 전체 구성요소를 정리해 제품 및 기술 지원)  
자격/인증(쿠버네티스 관련 교육 운영 및 자격증 발급)  
오픈소스 커뮤니티(CNA 생태계 조성)

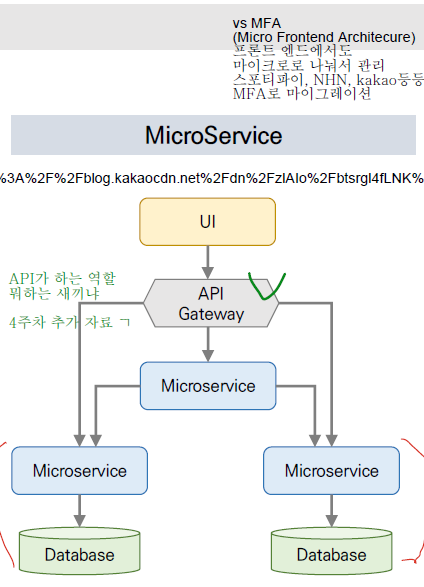
Cloud Native Application : 클라우드 환경에 최적화 되어 서비스 되도록 개발된 앱

Cloud Native Application 핵심요소 :   
최소의 상태(가장 단순한 일만 하는 상태)Stateful 컴포넌트들이 격리된 상태의 마이크로 서비스로 구성되며 각각의 서비스는 분산되고, 탄력적이며 수평적 확장성 있는 시스템으로 구성.  
앱을 서로 독립적인 기능을 하는 여러 개의 서비스로 구분.  
‘서비스’들을 묶어서 ‘하나의 통합된 비즈니스 서비스’를 할 수 있도록 하기 위한 다양한 기능과 기술 필요.  
서비스들을 어떻게 구성, 연결, 관리하는지가 핵심.

Cloud Native Application 특징 :   
  
+ 특정업무 트래픽 증가 시 처리 방법 : 전체서버 증설 트래픽이 증가한 서비스의 컨테이너만 유연하게 확장/축소

Cloud Native Application 효과 :   
On Demand Delivery – 필요한 컴퓨팅 자원을 즉시 제공.  
Consistency & Continuous – 이미지 기반으로 구성, 배포 효율화(복붙 쉬워). 개발과 운영환경의 일관성  
Rolling Update – 업그레이드 또는 패치 시 시간 최소화.  
Self Recovery – 장애 발생시, 정상 서버 노드로 자동 방향 재배치.  
Application Scaling - VM단위가 아닌 앱 단위의 오토 스케일링.  
Portable – 다양한 클라우드 기반 앱/서비스 운영.

Cloud Native Application 개발원칙 12factors  
대전제 : stateless(컨테이너 단위), 느슨한 결합, 독립적(언어, 플랫폼, 분산), CI/CD(DevOps, 자동화)  
코드베이스(버전관리 및 형상관리 가능)  
종속성(특정 시스템에 종속성X, 모든 종속성들은 명시적으로 선언 분리)  
설정(코드에 상수로 저장X, 설정정보는 환경에 저장되고 분리되어 관리)  
백엔드 서비스(로컬 서비스와 서드파티 서비스 구별X, 연결된 리소스로 취급)  
빌드, 릴리즈, 실행(단계가 분리되어야 함)  
무상태 프로세스(컨테이너 간 종속성 없이 단독 컨테이너만으로도 실행 가능한 상태)  
포트 바인딩(특정 포트를 바인딩하여 서비스를 공개하고 요청처리, 필요할 때 연결)  
동시성(동시에 다양한 작업 부하를 처리할 수 있도록 설계, H/V 확장 부하분산(load balancing))  
폐기 가능(문제없이 종료 가능, 바로 시작 가능하게 설계)  
개발/운영환경 최대한 비슷하게 유지(지속적인 배포 유지)  
로그(실시간으로 이벤트 기록 저장)  
admin 프로세스(관리자/일반 사용자 모드 나눠야함)

Cloud Native 핵심요소 MSA  
Micro Service Architecture : 대규모 SW 개발에 적용하기 위한 것으로 단독으로 실행 가능하고 독립적으로 배치될 수 있는 작은 단위로 기능을 분해하여 서비스하는 아키텍처.  
마이크로 서비스간 연결은 API 이용. 마이크로 서비스는 자원 표현이나 데이터 관리 등에 있어서 기능적으로 완전해야 함.  
REST API 두 시스템이 인터넷을 통해 정보 교환 위한 인터페이스  
MSA 특 : 한가지 기능, 가장 작은 단위 독립 서비스, API를 통한 연계, 개발/운영시 각 서비스 독립적  
MSA 장점 : 분산형 개발로 동시 개발 참여, 빠르고 유연한 조직문화 기여, 높은 신기술 수용력, 뛰어난 복구 능력, 유연한 확장(부하 분산 H/V 확장)

MSA 단점 : 배포/운영 자동화 필요, 중복성, 구조/개발/운영 복잡성 증가, 테스트/디버깅/파악 어려움

Cloud Native 핵심요소 Container  
Container : 가상화된 OS 위에서 앱의 독립적 실행에 필요한 파일(소스코드, lib, 부분 커널, 프레임워크 플랫폼, 툴 등)을 모은 패키지.   
Cloud Native SW의 가장 작은 단위. 배포/관리/공유 단순. 빠른 실행속도. 높은 안정성.  
크기 작고 가볍기에 VM과 비교하면 동일한 서버 환경에서 더 많은 앱 구동 가능. 자원 낭비 적음.

Cloud Native 핵심요소 DevOps  
DevOps : 기존 개발/운영 업무가 나눠진 역할 사이의 소통, 협업, 통합을 의미  
프로세스 자동화를 목표로 개발자/운영자가 협업하여 짧은 주기 내 신뢰성 있는 SW 생성, 테스트, 배포 할 수 있는 “문화와 환경”을 의미.  
앱 개발-운영 간 협업 프로세스 자동화하여 개발과 개선 속도 업.  
등장 배경 : 시스템과의 연결이 복잡해지고 기술과 요구사항이 지속적으로 변하는 시점에서 개발과 운영의 정보가 공유되지 않아 처리속도 느리고 책임소재 불문명한 상황 발생. ->협업, 소통, 통합 강조  
DevOps 특 : 속도/신속한 제공, 안전성, 확장/보안, 협업강화  
DevOps 필수요소 : 애자일(실질적 코딩 기반으로 프로토타입 형성 피드백 신속한 반복 작업을 통해 SW를 개발하는 적응형 개발방법), CI/CD(지속적 통합, 서비스 제공, 배포의 자동화)

Cloud Native 핵심요소 CI/CD  
CI/CD : 방법. 지속적인 통합, 서비스 제공, 배포의 자동화를 통해 짧은 주기로 app제공하는 방법.  
효과적으로 SW 제공하는 데 초점을 맞춤. 통합 및 테스트 단계와 제공 및 배포 단계에서 모니터링 및 자동화를 도입하여 앱 개발 프로세스 개선.

Continuous Integration : 앱 코드의 새로운 변경 사항이 정기적으로 빌드/테스트를 거쳐 공유 repository에 통합되는 것을 의미

Continuous Delivery(Deployment) : 변경사항이 테스트를 거쳐 자동으로 repository에 업로드 되는 것을 의미, repository에서 고객이 사용 가능한 프로덕션 환경가지 자동으로 배포되는 것을 의미

CI/CD와 DevOps 관계 :   
DevOps는 개발 효율 향상 목표하는 문화/프로세스. CI/CD는 그것을 실현하기 위한 필수적인 방법.

DevOps는 SW 개발의 라이프 사이클 전반을 포괄적으로 포함(팀구조, 관찰, 버전 관리 등)  
CI/CD는 일련의 특정 단계로, DevOps 사이클을 실현하는 툴, 자동화(배포, 통합 과정)에 포함

K-PaaS == 개방형 클라우드 플랫폼 : 정부에서 쿠버네티스 기반으로 만든 오픈소스를 통한 클라우드 플랫폼. 전자정부 표준 프레임워크 기반 open PaaS. 클라우드 인프라 위에서 SW/서비스를 개발 실행 운영 관리하는 기반 SW 환경.

PaaS를 구현하는 쿠버네티스 == Container Platform

Container Platform 구축환경 : 다양한 인프라를 지원하여 간편하게 Container Platform 구축 및 운영 가능. 대부분의 IaaS에서 쿠버네티스가 사용가능하게끔 지원. Aws, Azure, GCP, openstack, vmware vsphere

컨테이너 플랫폼의 쿠버네티스 기반의 배포 방법 :  
단독 배포 : 컨테이너 플랫폼을 단독으로 배포하여 독립된 쿠버네티스 환경 제공.  
엣지 배포 : 엣지 환경에 단독으로 배포하여 독립된 쿠버네티스 환경 제공