하이브리드 클라우드 전략

소개

강사

과정

강사

이름: 최국현

메일: tang@linux.com, 회신은 다른 메일 주소로 드리고 있습니다. :)

사이트: tang.dustbox.kr

언제든지 질문 및 요청 환영 입니다.

목차

쿠버네티스 101

목차

과정개요

하이브리드 클라우드를 위한 전략

과정개요

총 이틀에 거쳐서 이론/방법 그리고 랩을 통해서 간단하게 멀티클라우드/하이브리드 클라우드의 개념 및 정의를 학습한다. 기간이 짧기 때문에, 모든 부분에 대해서 랩으로 구성할 수 없지만, 최대한 단순한 구조를 가지고 간단하게 구성 하도록 한다.

길이	설명
1일	멀티클라우드/하이브리드 클라우드 개념 정의
2일	클라우드 구성을 위한 주요 도구 확인 및 활용

과정개요

이 과정은 클라우드 전체적인 부분에 대해서 **전략/개념/오픈소스**에 대해서 전체적으로 다루는 부트캠프 자료. 여기에서 다루는 지식들은 다음과 같은 대상에 적합하다고 판단이 됩니다.

1. 기술적 기반 영업을 다루는 프리 세일즈 엔지니어
2. 기술 기반으로 서비스를 구성하려는 서비스 아키텍처
3. 전반적인 윤곽이 필요하는 인프라/클라우드 엔지니어
4. 개발자 입장에서 인프라를 이해하고 싶은 소프트웨어 엔지니어
5. 기관 혹은 산업에서 활동 중인 중역 관리자

진행자의 수준에 따라서 이 내용은 하루 혹은 이틀로 조정이 가능 합니다.

하이브리드 클라우드 기본 이론

이론

하이브리드 클라우드의 정의와 개념

이론

개요

현재 클라우드는 다음과 같은 주요 개념과 기술 요소로 대분류가 가능하다.

- **클라우드 컴퓨팅(Cloud Computing)**: 여러 컴퓨터 자원을 가상화 기반으로 네트워크를 통해 제공하는 컴퓨팅 방식
- **소프트웨어 배포 방식(Delivery Models)**: IaaS, PaaS, SaaS 등의 계층별 서비스 모델

개요

- **클라우드 유형(Cloud Types)**: 퍼블릭, 프라이빗, 하이브리드, 멀티 클라우드
- **가상화 기술**: 가상머신(VM)을 통해 물리적 서버를 논리적으로 분리
- **컨테이너 기술**: 애플리케이션 단위의 경량 가상화 (예: Docker, Podman)

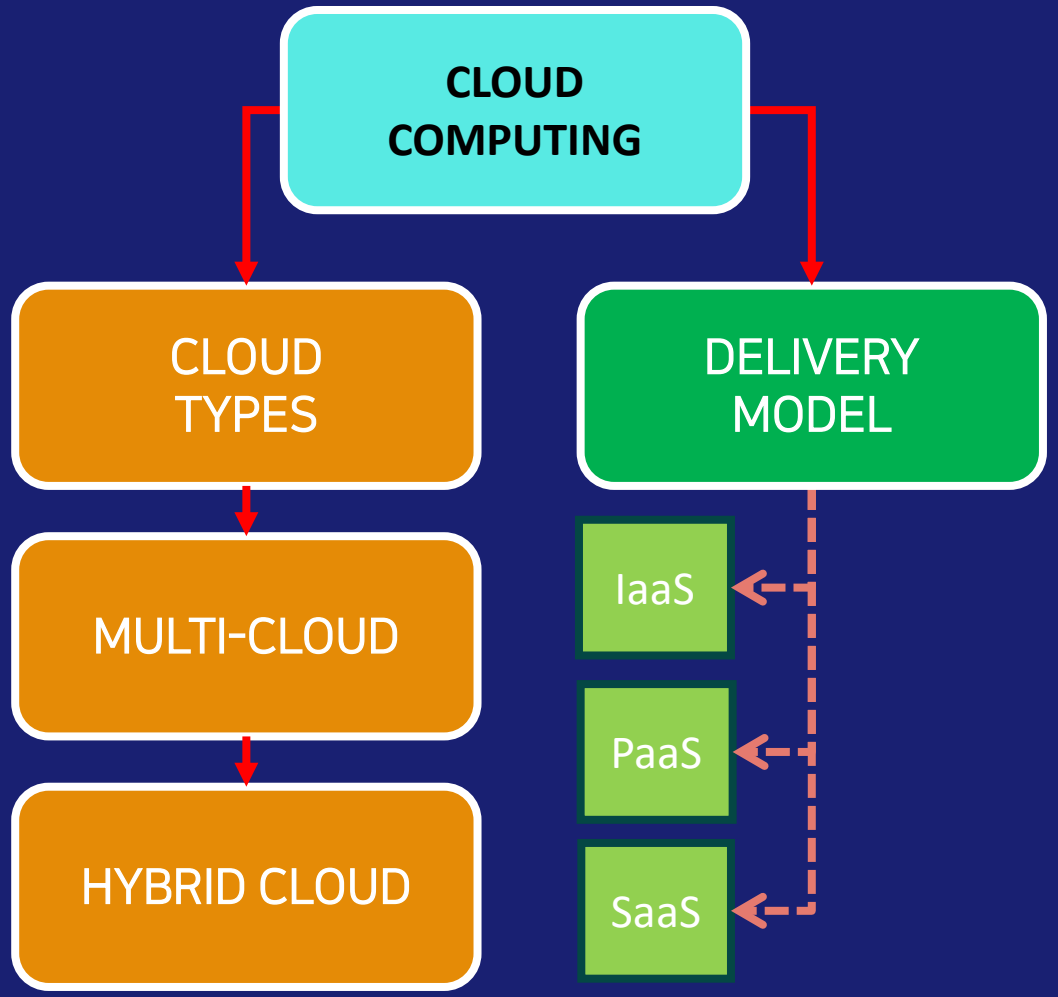
이러한 기술들을 활용하여, 단일 컴퓨트 노드가 아닌 **다수의 노드**를 묶어 **분산 컴퓨팅 환경**을 구성. 이를 통해 자원을 **효율적으로 할당 및 확장**할 수 있으며, **장애 발생 시 자동 복구 및 고가용성(HA)**을 지원하는 구조도 구현할 수 있습니다.

개요

위의 내용을 소분류 하면 다음과 같다.

대분류	소분류 / 설명
클라우드 컴퓨팅	분산 자원을 네트워크 기반으로 제공
소프트웨어 배포 방식	IaaS / PaaS / SaaS 등 계층적 서비스 제공
클라우드 유형	퍼블릭 / 프라이빗 / 하이브리드 / 멀티 클라우드
가상화 기술	하드웨어 위에서 VM 환경 제공 (예: KVM, VMware)
컨테이너 기술	애플리케이션 단위 경량 가상화 (예: Docker, Podman)

개요


CLOUD
COMPUTING
CLOUD
TYPES
DELIVERY
MODEL
MULTI-CLOUD
HYBRID CLOUD
IaaS
PaaS
SaaS

개요(NIST 기준)

클라우드 컴퓨팅은 다음과 같은 정의를 가지고 있다.

"**클라우드 컴퓨팅(Cloud Computing)**은 인터넷을 통해 컴퓨팅 자원(서버, 스토리지, 네트워크, 소프트웨어 등)을 필요에 따라 제공받는 IT 서비스 모델. 사용자는 물리적 하드웨어를 직접 소유하거나 관리하지 않고, 필요한 만큼의 자원을 유연하게 이용하며, 사용한 만큼 비용을 지불하는 방식."

개요(NIST 기준)

하지만, 위의 부분은 **상용환경**에서 정의가 된 부분이며, **학술적**으로 다음처럼 정의가 된다.

"클라우드 컴퓨팅은 인터넷을 통해 구성 가능한 **컴퓨팅 자원**(예: 네트워크, 서버, 스토리지, 애플리케이션 및 서비스)에 대한 편리하고 온디맨드 방식의 네트워크 액세스를 제공하는 모델로, 최소한의 관리 노력이나 서비스 제공자와의 상호작용으로 빠르게 제공되고 해제될 수 있다."

개요(NIST 기준)

NIST는 클라우드 컴퓨팅의 핵심 특성을 다음과 같이 다섯 가지로 정의한다.

클라우드 특성	설명
온디맨드 셀프서비스 (On-demand self-service)	사용자가 필요에 따라 자동으로 컴퓨팅 자원을 프로비저닝 가능
광범위한 네트워크 액세스 (Broad network access)	모바일, 태블릿, 노트북 등 다양한 플랫폼을 통해 네트워크를 통해 자원에 접근 가능
자원 풀링 (Resource pooling)	서비스 제공자가 자원을 풀링하여 여러 사용자에게 동적으로 할당
빠른 탄력성 (Rapid elasticity)	자원이 빠르게 확장 또는 축소되어 수요에 따라 유연하게 대응
측정된 서비스 (Measured service)	자원 사용량이 측정 및 모니터링되어 투명한 비용 청구가 가능

개요(NIST 기준)

링크

https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf

https://en.wikipedia.org/wiki/Cloud_computing

개요

보통 다음과 같이 클라우드 혹은 컴퓨팅 모델을 분류한다.

계층	포함 기술 예시
SaaS	Gmail, Salesforce
PaaS	Kubernetes, OpenShift, Cloud Foundry
IaaS	OpenStack, AWS EC2, VMware vSphere
컨테이너 런타임	Docker, Podman, containerd
가상화 계층	KVM, Xen, libvirt, QEMU

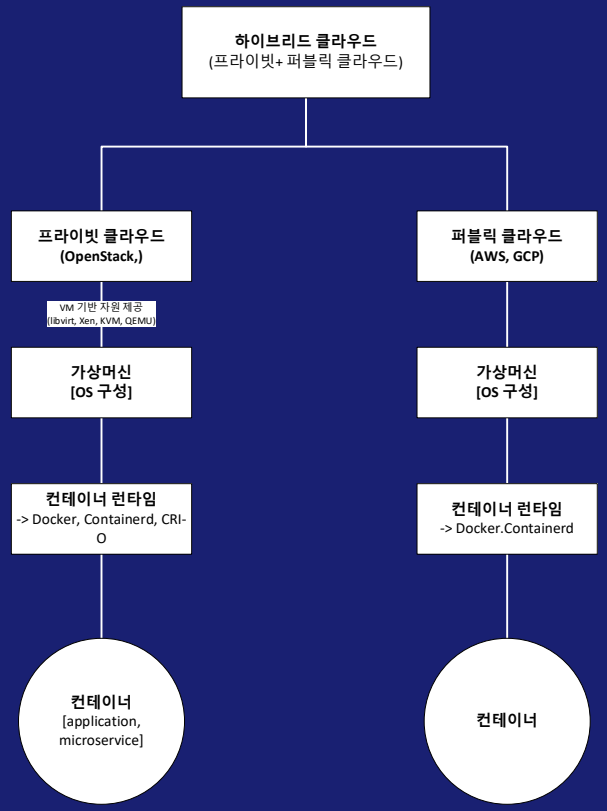
아키텍처

흔히, 많이 이야기하는 PaaS/IaaS는 소프트웨어 기반으로 분류하지 않는다.

기술	직접 분류	PaaS와의 관계
libvirt	PaaS 아님	I/PaaS 구성 기술로 활용됨 (예: kubevirt, oVirt , OpenStack Nova)
Xen	PaaS 아님	VM 기반의 PaaS 시스템 구성에 사용 가능
KubeVirt	PaaS 구성 요소 (K8s 위에서 VM 운영)	내부적으로 libvirt 사용
oVirt	웹 기반 가상화 관리 플랫폼 (IaaS~PaaS 중간)	libvirt, KVM, VDSM 등 사용하여 **PaaS 유사 플랫폼** 제공

아키텍처

위의 기술을 도식으로 그렸을 때 다음과 같다. **프라이빗/퍼블릭 클라우드**는 아키텍처 구조상 거의 동일하다.



하이브리드 클라우드

하이브리드 클라우드(Hybrid Cloud)

하이브리드 클라우드는 **프라이빗 클라우드와 퍼블릭 클라우드가 유기적으로 연결되어**, 하나의 통합된 인프라처럼 운영되는 환경을 말한다.

일반적으로 기업은 민감한 데이터나 핵심 시스템은 프라이빗 클라우드에, 높은 유연성과 확장성이 필요한 워크로드는 퍼블릭 클라우드에 배치 구성한다. 하이브리드 클라우드는 **워크로드를 환경 간에 이동하거나 분산 처리할 수 있으며**, 보안성과 확장성을 동시에 추구하는 전략적 선택으로 활용한다.

예를 들어, 내부 고객정보 시스템은 자체 데이터센터(OpenStack 등)에, 외부 쇼핑몰 웹사이트는 AWS에 위치시키고, 두 환경을 VPN이나 Direct Connect 등으로 연결해 통합 관리하는 경우가 대표적 예시.

하이브리드 클라우드 단점

하이브리드 클라우드(Hybrid Cloud)의 단점

하이브리드 클라우드는 **퍼블릭과 프라이빗**의 장점을 결합하지만, **두 환경 간의 네트워크 연결과 보안정책 통합이 매우 복잡**하다.

일관된 관리를 위해서는 **고도화된 모니터링, 인증 시스템, 정책 동기화**가 필요하며, 환경 간의 워크로드 이동은 **기술적 장애물**이 많아 설계부터 운영까지 매우 **고난이도 아키텍처** 및 기술에 포함이 된다.

하이브리드 클라우드와 비슷한 개념은 하이브리드 워크로드가 있다. 이 부분에 뒤에서 더 다루도록 한다.

하이브리드 클라우드 정리

하이브리드 클라우드 운영 시, 다음과 같은 부분들을 매우 고려해야 한다.

항목	설명
복잡한 네트워크 구성	퍼블릭과 프라이빗 환경 간의 **보안 연결(VPN, Direct Connect 등)** 설정이 복잡하며, 트래픽 흐름을 통합적으로 관리하기 어려움.
통합 운영 어려움	두 환경의 모니터링, 인증, 권한, 자원 스케줄링 등을 통합 관리하기 위한 툴이나 체계가 **복잡하고 비용이 큼**.
보안 정책 일관성 유지 어려움	퍼블릭과 프라이빗 간 **보안 기준 차이**로 인해 취약점이 발생할 수 있으며, 정책 적용 시 충돌 가능성 존재.
비용 예측 어려움	자원 분산과 연동 비용, 네트워크 전송 비용 등으로 인해 **TCO(Total Cost of Ownership) 계산이 어려움**.
의존성 증가	특정 클라우드에 연동된 워크로드가 많아질수록 **해당 서비스에 종속될 위험** 존재.

2025-05-18

CSP 업체의 특징

기업이 AWS, Azure, GCP를 각각 웹 서버, 머신러닝, 백업/아카이빙 용도로 사용하는 멀티 클라우드 전략은 각 플랫폼의 강점을 활용하여 비용 효율성과 성능을 극대화할 수 있는 접근 방식이다.

서비스가 업데이트가 되면서 조금씩 달라지겠지만, 일반적으로 다음 슬라이드처럼, 각 **클라우드 서비스 제공업체 (Cloud Service Provider)**의 비용 구조는 다음과 같은 특성을 가지고 있다.

CSP 업체의 특징

AWS: 웹 서버 운영에 적합

- **서비스**: Amazon EC2 (웹 서버 호스팅)
- **비용 구조**:
  - 온디맨드 인스턴스: 시간당 과금
  - 예약 인스턴스: 1년 또는 3년 약정 시 최대 75% 할인
  - 스팟 인스턴스: 최대 90% 할인 가능
- **특징**:
  - 글로벌 인프라와 다양한 인스턴스 유형 제공
  - 웹 서버와 같은 지속적인 워크로드에 적합
  - OpenAI가 사용하는 서비스

CSP 업체의 특징

Azure: 머신러닝 서비스에 강점

- **서비스**: Azure Machine Learning
- **비용 구조**:
  - 온디맨드 인스턴스: 시간당 과금
  - 예약 인스턴스 및 세이빙 플랜: 최대 65% 할인
  - 하이브리드 혜택: 기존 온프레미스 라이선스 활용 시 추가 할인
- **특징**:
  - Microsoft 생태계와의 통합 용이
  - 기업 환경에 적합한 보안 및 규정 준수 기능 제공

CSP 업체의 특징

GCP: 백업 및 아카이빙에 경쟁력

- **서비스**: Cloud Storage (Nearline, Coldline)
- **비용 구조**:
  - Nearline: 월 $0.01/GB
  - Coldline: 월 $0.004/GB
  - 아카이브 스토리지: 월 $0.0012/GB
- **특징**:
  - 장기 보관 데이터에 대한 저렴한 스토리지 옵션 제공
  - 자동화된 데이터 수명 주기 관리 기능
  - 최근 제미니 및 양자 컴퓨팅를 서비스에 포함

멀티 클라우드

멀티 클라우드는 **두 개 이상의 서로 다른 클라우드 서비스 제공자(AWS, Azure, GCP 등)**를 동시에 사용하는 전략.

이 구조는 각 클라우드의 특성과 강점을 조합하여 활용할 수 있게 해주며, 특정 벤더에 종속되지 않는 유연한 아키텍처를 구성이 가능하다. 일반적으로 멀티 클라우드는 각 클라우드가 **독립적으로 운용**되며, 서비스 간에 직접적인 연결이 없어도 운영 및 관리가 가능하다.

예를 들어, 기업이 AWS를 통해 웹 서버를 운영하고, Azure를 통해 머신러닝 서비스를 운영하며, GCP는 백업 및 아카이빙 용도로 사용하는 경우가 멀티 클라우드.

이 방식은 **비용 최적화, 장애 대응, 지역 분산** 등의 장점을 제공.

멀티 클라우드 단점

멀티 클라우드는 여러 클라우드 제공자(AWS, Azure, GCP 등)의 서비스를 병행하여 사용할 수 있는 유연성을 제공하지만, 그만큼 **운영 복잡성이 매우 높아지는** 단점.

각 클라우드는 고유한 API, 인증 체계, 보안 모델, 리소스 구조를 가지므로, 이를 통합 관리하려면 **복잡한 오케스트레이션 도구**나 **추가적인 관리 플랫폼**이 필요.

또한 서비스 간 네트워크 연결, 상태 모니터링, 비용 분석을 일관되게 유지하는 것이 어려워 **관제 인프라의 부담**도 증가. 뿐만 아니라 각 클라우드의 기능을 깊이 이해하고 관리할 수 있는 **전문 인력이 필요**하며, 그렇지 않으면 운영 리스크가 상승하는 단점이 있음.

멀티 클라우드 정리

항목	설명
운영 관리 복잡도	서로 다른 클라우드 플랫폼(AWS, Azure 등)의 **인터페이스, API, 관리도구**를 모두 익혀야 하며, 팀 역량이 분산됨.
통합 보안 구성 어려움	서로 다른 인증 체계와 IAM 정책으로 인해 **보안 취약점 발생 가능성** 증가. 중앙 집중형 보안 관리 어려움.
데이터 이동 제약	클라우드 간 데이터 이전 시 **전송 속도, 비용, 형식** 문제가 발생할 수 있으며, 경우에 따라 법적 제약도 존재.
벤더별 정책 차이	VM 크기, 로깅 시스템, 가격 정책 등 서비스 제공 방식이 달라 **표준화된 운영이 어려움**.
통합 모니터링/로깅 어려움	클라우드별 모니터링 도구가 달라서 **전체 가시성 확보가 어려워**, 장애 대응이나 성능 분석이 지연될 수 있음.

퍼블릭 클라우드 단점

퍼블릭 클라우드는 높은 확장성과 유연성을 제공하지만, **데이터 보안과 제어권 문제**가 가장 큰 단점.

민감한 정보를 외부 클라우드에 저장하거나 처리해야 하므로 **규제 산업(금융, 의료 등)**에서는 사용이 제한될 가능성이 높음.

또한, 트래픽이 증가할수록 **예상치 못한 요금 폭탄**이 발생할 수 있고, **벤더 종속(Vendor Lock-in)** 가능성이 매우 높음.

프라이빗 클라우드 단점

프라이빗 클라우드는 기업 내부에서 인프라를 완전히 제어할 수 있어 보안에 강점이 있지만, **초기 구축 비용과 유지 보수 비용이 매우 높다.**

인프라 운영을 위한 **전문 인력과 지속적인 투자**가 필요하며, 수요 급증 시 **신속한 리소스 확장**이 어렵다.

하지만, 안정적인 사용량 및 장기적인 관점에서는 퍼블릭 클라우드보다 비용이 저렴하다.

기업의 IT인프라 변화에 대한 배경

이론

설명

최근 몇 년 사이, 기업의 IT 인프라는 빠르게 변화하고 있다. 과거에는 물리적인 서버와 네트워크 장비를 기업 내 데이터센터에 직접 설치하여 운영하는 온프레미스 방식이 주를 이루었으나, 현재는 퍼블릭 클라우드, 프라이빗 클라우드, 그리고 이를 조합한 하이브리드 및 멀티 클라우드 환경으로 이동하고 있다.

이러한 변화의 **첫 번째 배경**은 **비즈니스의 민첩성 확보 요구**이다. 디지털 전환이 가속화되면서 기업은 시장의 변화에 빠르게 대응하고, 새로운 서비스를 신속하게 출시할 수 있는 IT 환경을 필요로 하게 되었다. 온프레미스 인프라는 구축과 확장에 시간이 오래 걸리고 유연성이 떨어져, 급변하는 환경에 적합하지 않다는 한계가 있었다.

설명

두 번째, **비용 절감과 자원 최적화**의 필요성이다. 전통적인 인프라는 서버나 스토리지를 사전에 대규모로 구매하여야 했고, 실제 사용량과 관계없이 유지보수 비용이 지속적으로 발생했다.

반면 클라우드 기반 인프라는 필요한 만큼만 자원을 사용하고, 사용한 만큼만 비용을 지불하는 모델을 제공하여 기업의 IT 운영비용을 유연하게 조절할 수 있게 했다.

설명

세 번째는 **보안과 규제 대응 강화**이다. 특히 금융, 의료, 공공 분야에서는 데이터의 보관 위치와 접근 통제가 중요하다.

이에 따라 퍼블릭 클라우드 단독 사용보다는 프라이빗 클라우드와의 조합, 즉 하이브리드 클라우드 전략이 필요. 기업은 민감 데이터를 내부에 보관하면서도, 퍼블릭 클라우드의 확장성과 유연성을 함께 활용할 수 있는 방식을 선택하고 있다.

마지막으로, **코로나19 팬데믹 이후의 원격 근무 확산**은 클라우드 전환을 더욱 가속화. 다양한 위치에서 접근이 가능하고, 빠르게 확장 가능한 클라우드 기반 인프라는 분산 근무 환경을 뒷받침하는 핵심 요소로 떠올랐다.

이와 같은 배경 속에서, 기업은 단일 환경에 의존하기보다는 여러 클라우드를 혼합해 사용하는 멀티 클라우드 전략을 병행하고 있으며, 이는 단순한 선택이 아니라 **생존을 위한 전략적 전환**으로 자리잡고 있다.

앞으로도 이러한 흐름은 지속될 것으로 보이며, 인프라의 유연성과 관리 효율성을 확보하기 위한 기술과 도구의 중요성은 더욱 커질 것이다.

2025-05-18

CapEx (자본적 지출, Capital Expenditure)

정의: 서버, 스토리지, 네트워크 장비 등 물리적 자산을 구매할 때 발생하는 비용

특징:

- 선불(초기 일괄 투자)
- 자산으로 회계 처리
- 장기간 감가상각 적용

예시: 데이터센터에 서버 100대를 구입하여 구축하는 경우

OpEx (운영비용, Operating Expenditure)

정의: 서비스 사용에 따라 정기적으로 발생하는 운영비용

특징:

- 종량제 또는 구독제 (월별/시간별 비용)
- 유연한 회계처리
- 즉시 비용 처리 가능

예시: AWS EC2 인스턴스를 시간 단위로 사용하며 매달 요금 지불

CapEx → OpEx의 전환 의미

1. 클라우드 도입의 핵심 이점 중 하나
2. 대규모 **선투자(CapEx)**를 줄이고, 필요한 만큼 쓰고 **비용을 내는 구조(OpEx)**로 전환
3. 예측 가능성은 낮지만, 유연성과 확장성은 높음
4. 초기 비용 부담 없이 신속한 비즈니스 실행 가능

왜 하필이면 클라우드?

기업의 IT인프라 변화에 대한 배경

2025-05-18

왜 기업은 클라우드를 도입할까?

1. 비용 효율성 (CapEx → OpEx 전환)

- IDC는 **서버·스토리지·전력·공간** 직접 투자 필요 (CapEx, 자본 지출)
- 클라우드는 **필요할 때만 지불하는 구조** (OpEx, 운영비)
- 초기 투자 부담 감소 + 예산 유연성 확보

왜 기업은 클라우드를 도입할까?

2. 확장성과 민첩성

- IDC는 서버 한 대 증설하려면 길면, 1~3주 걸림
- 항상 여분의 장비 혹은 부품을 가지고 있어야 됨
- 클라우드는 **몇 분 안에 인프라 생성 가능(온프레미스 환경과 프라이빗 클라우드는 다름)**
- 이벤트 대응, 캠페인 확장, 데이터 증가에 유연하게 대처 가능

왜 기업은 클라우드를 도입할까?

3. 운영 자동화와 DevOps 연계

- 클라우드에서는 IaC(Terraform, Ansible), CI/CD, 오토스케일 등으로 운영 효율화
- IDC는 수작업 위주 운영, DevOps 환경 적용이 어렵고 느림

4. 지리적 유연성

- 글로벌 서비스 → AWS/GCP 리전 배포로 **즉시 글로벌 진출 가능**
- IDC는 **해외 진출에 제약**이 큼

왜 기업은 클라우드를 도입할까?

5. 비즈니스 연속성 / 재해 복구

- 클라우드는 **백업/DR(재해복구)** 구성이 쉬움
- IDC는 이중화 구성이 고비용 + 구축 어려움

운영 모델 분류

운영 모델(Operation Model)은 클라우드가 어떻게 소유되고 운영되는지를 기준으로 분류.

(1) 퍼블릭 클라우드 (Public Cloud)

- 여러 사용자가 공유하는 형태로, 외부 제공자가 운영.
- 오픈소스 솔루션으로도 퍼블릭 클라우드 서비스를 구축할 수 있음.
- 예: CityCloud (OpenStack 기반), OVHcloud, NHN Cloud, KT Cloud, NCP(Naver)

운영 모델 분류

(2) 프라이빗 클라우드 (Private Cloud)

- 특정 조직이 전용으로 사용하는 클라우드 환경.
- 보안, 제어, 규제 측면에서 우수함.
- OpenStack, CloudStack, Haverster등이 자주 사용됨.

운영 모델 분류

(3) 하이브리드 클라우드 (Hybrid Cloud)

- 프라이빗과 퍼블릭 클라우드를 조합하여 운영.
- 민감한 데이터는 내부에, 일반 서비스는 외부 클라우드에 배치하는 전략에 적합.
- 오픈소스로 OpenShift, Rancher, KubeVirt 등이 하이브리드 구축에 활용됨.

(4) 멀티 클라우드 (Multi-Cloud)

- 여러 퍼블릭 클라우드 서비스를 동시에 사용하는 구조.
- 특정 벤더 종속을 피하고 가용성 및 탄력성 확보에 유리.
- 관리 도구로는 KubeFed, Crossplane, Rancher, ArgoCD 등이 사용됨.

운영 모델 분류

(5) 온프레미스 (On-Premises)

물리 서버 및 네트워크 인프라를 기업이 직접 구매하고 내부에서 운영. 자원 관리, 보안, 네트워크까지 전적으로 내부에서 책임지며, 클라우드 기능은 포함되지 않음.

자동화 없이 수동 관리가 많으며, 하이퍼바이저 기반으로 VMware, KVM 등이 사용됨. 인프라 구축 및 유지비용은 높지만, 완전한 통제와 특정 산업 규제에 적합.

시대별 IT 운영 모델 분류표

시대별로 IT 인프라 운영 모델은 기술 발전과 비즈니스 요구에 따라 점진적으로 변화해왔다. 다음은 그 흐름을 시기별로 정리한 표이다.

시대	주요 운영 모델	특징	예시
1980년대 ~ 1990년대	On-premise (전통적 데이터센터)	모든 인프라를 기업 내부에 구축, CapEx 중심, 장기 계획 수립 필요	자체 서버, 전산실, 메인프레임
2000년대 초반	가상화(Virtualization)	물리 서버 위에 여러 VM 운영, 자원 효율성 증가, 관리 복잡성 상승	VMware, Xen, Hyper-V
2010년대 중반까지	프라이빗 클라우드 / 퍼블릭 클라우드	자원 탄력성, 자동화, OpEx 전환, 웹 기반 관리 대중화	AWS, Azure, OpenStack, vSphere

2025-05-18

시대별 IT 운영 모델 분류표

앞에 내용 계속 이어서…

시대	주요 운영 모델	특징	예시
2015년 ~ 현재	하이브리드 클라우드	온프레미스 + 클라우드 혼합, 민감 데이터는 내부, 나머지는 외부 처리	AWS Outposts, Azure Stack, OpenShift, Rancher
2020년대 이후	멀티 클라우드 / 클라우드 네이티브 / 엣지 컴퓨팅	다양한 CSP 조합, 분산 서비스 구성, API 중심 운영, DevOps 중심	Kubernetes, Anthos, KubeVirt, Cloudflare Workers

오픈소스 클라우드 분류 요약 표

많은 사람들이 혼용 혹은 혼동하는 부분이 운영 모델 및 서비스 모델이다. 이 두개의 차이는 다음과 같다.

대분류	소분류	설명	대표 오픈소스
서비스 모델	IaaS	가상화된 인프라 자원을 제공	OpenStack, CloudStack, Eucalyptus
	PaaS	앱 개발을 위한 플랫폼 제공	PaaS-TA, Cloud Foundry, OpenShift
	SaaS	사용 가능한 소프트웨어 제공	Nextcloud, ONLYOFFICE, Zimbra
운영 모델	**퍼블릭 클라우드**	다수 사용자 대상 외부 제공	OVHcloud (OpenStack 기반)
	프라이빗 클라우드	조직 내부에서 운영	OpenStack, CloudStack, Haverster
	하이브리드 클라우드	내부+외부 혼합 환경	OpenShift, Rancher, KubeVirt
	멀티 클라우드	여러 클라우드 동시 활용	KubeFed, Crossplane, Rancher
	온프레미스	일반적으로 클라우드 환경 미사용	oVirt, libvirtd, VMware ESX

왜 기업은 클라우드를 도입할까?

여기서는 일단, 하이브리드/프라이빗 클라우드는 제외하고 CSP만 포함 하였다.

항목	IDC(온프레미스)	클라우드
초기 투자	고비용 (서버, 네트워크, 인프라)	저비용 (필요 시 사용, 서버리스 가능)
확장성	물리 장비 의존 → 느림	자동 확장 → 빠르고 유연
유지보수	직접 설치 및 교체, 수작업	클라우드 벤더 관리, 자동화 기능 제공
관리 도구	제한적, 벤더 종속	Terraform, Ansible, Helm, Salt, GitOps 등 다양
고가용성 구성	고비용, 복잡한 이중화 필요	기본적으로 고가용성 제공 (AZ, Region 기반)
보안 모델	물리적 접근에 강점	논리적 분리, IAM, Audit 기반
애플리케이션 배포	수동 배포 중심	CI/CD, Auto Deployment 연계 용이
글로벌 진출	매우 어렵고 고비용	리전 기반 글로벌 배포 가능

IDC의 종말?

기업의 IT인프라 변화에 대한 배경

IDC 기반 운영의 더 이상 필요 없는가?

기업이 자체 IDC(데이터센터) 기반으로 시스템을 운영하는 가장 큰 장점은 인프라에 대한 **완전한 제어권**을 확보할 수 있다는 점이다. 모든 서버와 네트워크 장비, 스토리지 구성까지 기업 내부의 정책에 따라 설계하고, 보안 설정 및 접근 통제도 세밀하게 조정할 수 있기 때문에 보안과 안정성 측면에서 높은 수준의 신뢰를 제공한다.

특히 **금융, 국방, 공공기관**과 같이 외부 퍼블릭 클라우드 사용이 제한되거나 금지된 분야에서는 자체 IDC 운영이 **법적·정책적 요구사항을 충족하기 위한 유일한 선택지**가 된다. 이 경우 물리적 보안과 데이터 주권을 유지하는 것이 기업의 핵심 운영 전략 중 하나로 작용한다.

또한, 장기적인 비용 관점에서도 IDC 운영은 경쟁력이 있을 수 있다. 퍼블릭 클라우드는 초기 비용 부담은 적지만, **사용량에 따라 비용이 지속적으로 증가**하는 구조이기 때문에, 고정적인 대용량 IT 자원을 장기간 사용하는 기업의 경우에는 자체 IDC를 통해 **비용 예측성과 총소유비용(TCO)**을 절감할 수 있다.

2025-05-18

IDC 기반 운영의 더 이상 필요 없는가?

성능 최적화 측면에서도 IDC는 강점을 가진다. 퍼블릭 클라우드는 인스턴스 타입이나 리소스 배분에 일정한 제약이 있지만, IDC는 **기업 맞춤형으로 하드웨어를 직접 구성**할 수 있어 고성능 **연산(HPC), 특수 GPU 서버, 로우 레이턴시 네트워크** 등 고도로 최적화된 인프라 설계가 가능하다.

뿐만 아니라, 자체 IDC는 **내부망 중심의 네트워크 설계**가 가능하다는 점에서 네트워크 지연이 적고, 내부 서비스 간 통신 속도 및 안정성이 높다는 실질적인 이점도 있다.

마지막으로, 퍼블릭 클라우드는 특정 벤더에 종속될 위험(Vendor Lock-in)이 존재하지만, IDC 기반 운영은 기업이 **전략적으로 오픈소스 기반 솔루션이나 원하는 상용 솔루션을 유연하게 선택**할 수 있어 기술적 자율성을 보장받을 수 있다.

IDC 기반 운영의 장점

IDC기반으로 운영 시, 다음과 같은 장점이 있다. 간단하게, 민감한 데이터는 자체적인 구축을 더 권장한다.

항목	설명
완전한 제어권	모든 하드웨어, 네트워크, 보안 정책을 기업이 직접 설정하고 통제 가능
보안 및 규제 대응	금융, 국방, 공공기관 등은 *외부 클라우드 사용 금지 또는 제한*이 존재함
예측 가능한 장기 비용	장기적으로 보면 서버/스토리지 직접 소유가 **비용 측면에서 유리**할 수 있음 (특히 대용량 처리 시)
네트워크 지연 최소화	내부 전산망 중심 구조 → 낮은 latency, 빠른 데이터 접근
맞춤형 인프라 설계	하이퍼컨버지드, 특수 고성능 컴퓨팅(HPC), GPU팜 등 **고도화된 커스터마이징** 가능
기존 자산 활용	이미 보유한 물리 자산 활용 가능 → *이전 자산의 ROI 확보*

퍼블릭 클라우드와 비교한 IDC의 "현실적인 강점"

대신 IDC의 최대 단점이라고 하면, 초기 투자 비용 및 IDC시설 비용이 지속적으로 투입이 된다. 하지만, 월 유지비용은 상대적으로 퍼블릭 클라우드보다 저렴한 편이다.

항목	퍼블릭 클라우드	자체 IDC 기반
초기 구축 비용	낮음 (시작 용이)	높음 (서버, 공간, 네트워크 등 직접 준비)
장기 운영 비용	사용량 따라 증가 (예측 어려움)	장기적으로 예측 가능
성능 최적화	제한된 인스턴스 타입, 공유 HW	HW 선택 가능, 최적의 성능 설계 가능
데이터 주권/보안	외부 의존, 국가별 리스크 존재	내부 보안 규제 준수 쉬움 (물리 통제 가능)
네트워크 구조	외부망 통신 중심	내부망 기반 통신 → 빠름 + 보안성 높음
벤더 종속성(Vendor lock-in)	존재	없음 (모든 것 직접 관리)

프라이빗 클라우드는 온프레미스의 '진화형'

온프레미스는 죽지 않았다. 여전히 발전이 되고 있으며, 그 역할이 바로 프라이빗 클라우드이다.

항목	온프레미스 (On-premise)	프라이빗 클라우드 (Private Cloud)
자원 소유권	기업이 전적으로 보유	기업이 보유하되, 클라우드 기술 적용
운영 방식	수작업 중심, 정적 구성	자동화, 오케스트레이션 도입 (예: OpenStack, VMware Cloud)
확장성	제한적, 수동 확장	유연한 확장성, 셀프서비스 지원
비용 구조	CapEx 중심	CapEx + OpEx 혼합 (기술 적용 방식에 따라)
사용자 경험	인프라 담당자 중심	개발자도 셀프서비스 접근 가능
기술 구성	물리 서버, 가상화	클라우드 API, 자동화, 멀티테넌시 가능
예시	전통적인 데이터센터	OpenStack, vSphere + vRA, Harvester 등

퍼블릭 클라우드 vs. 자체 IDC TCO 비교

1. 퍼블릭 클라우드의 TCO 절감 효과

- Accenture에 따르면, **퍼블릭 클라우드로 워크로드를 이전하면 TCO를 30~40%까지 절감**할 수 있습니다. 이는 클라우드의 유연성과 확장성, 그리고 운영 효율성에 기인.
- IDC의 연구에서도 **AWS 고객이 온프레미스 인프라 대비 운영 비용을 51% 절감 보고.**

퍼블릭 클라우드 vs. 자체 IDC TCO 비교

2. 자체 IDC의 장기적인 비용 구조

- Michael S. Kenny & Company의 백서에서는 **온프레미스 시스템의 TCO에서 인건비가 전체 비용의 50~85%를 차지**한다고 분석. 이는 하드웨어 유지보수, 시스템 업그레이드, 보안 관리 등에 필요한 인력 비용이 상당하다는 것을 의미합니다.
- IDC의 또 다른 보고서에서는 **프라이빗 클라우드 플랫폼을 운영할 경우 인프라 비용이 34% 낮아진다**고 발표. 이는 전통적인 온프레미스 환경보다 프라이빗 클라우드가 비용 효율적일 수 있음을 시사.

TCO 비교 그래프 예시

항목	퍼블릭 클라우드	자체 IDC	프라이빗 클라우드
초기 투자 비용	낮음 (즉시 시작 가능)	매우 높음 (서버, 공간, 전력 등)	중간 (기존 IDC 인프라에 SW만 구성 시 저렴)
운영비	사용량 기반 (예측 어려움)	고정적이나 전기/장비/인건비 고정 부담	자체 관리 + 자동화 도입 시 효율성 증가
유지보수/업그레이드	클라우드 사업자 책임	기업 책임 (장비 교체, 업데이트 등)	일부 사업자 위탁 가능 (VMware, OpenStack 기반 등)
인건비	낮음 (자동화 비중 높음)	높음 (직접 관리 필요)	중간 (관리 자동화 수준에 따라 달라짐)
유연성 / 확장성	매우 높음 (글로벌 리전, 오토스케일링)	낮음 (물리 장비 의존)	중간~높음 (프라이빗 클라우드 플랫폼에 따라 다름)
보안 / 데이터 주권	낮음~중간 (사업자에 따라)	매우 높음 (자체 통제)	높음 (물리 자산 통제 + 소프트웨어 기반 분리)
TCO (5년)	중간 (비용 증가 추세)	높음	중간 (비용 대비 유연성 확보)

프라이빗 vs 퍼블릭 클라우드 TCO 비교

IDC의 보고서에 따르면, **예측 가능한 워크로드**의 경우, **온프레미스 프라이빗 클라우드**가 퍼블릭 클라우드 대비 TCO가 절반 수준.

- **시나리오 1: 예측 가능한 워크로드**
  - **프라이빗 클라우드**: Nutanix 기반 인프라를 활용하여 5년간 TCO가 퍼블릭 클라우드 대비 약 50% 절감.
  - **퍼블릭 클라우드**: 사용량 기반 과금으로 인해 장기적으로 비용이 증가하는 경향.
- **시나리오 2: 혼합형 워크로드**
  - **프라이빗 클라우드**: TCO가 퍼블릭 클라우드 대비 약 66% 수준으로, 비용 효율성이 높음.
  - **퍼블릭 클라우드**: 유연한 확장성이 장점이나, 예측 가능한 워크로드에서는 비용이 높음.

프라이빗 vs 퍼블릭 클라우드 TCO 비교

- **시나리오 3: 고탄력성 워크로드**
  - **퍼블릭 클라우드**: 수요에 따라 리소스를 탄력적으로 조정할 수 있어 비용 효율적.
  - **프라이빗 클라우드**: 리소스 활용률이 낮아질 경우 비용 효율성이 떨어질 수 있다.

보고서 자료

https://media.bitpipe.com/io_14x/io_140838/item_1639508/IDC%20TCO%20Analysis%20Comparing%20Private%20and%20Public%20Cloud%20SOlutions%20for%20Running%20Enterprise%20Workloads.pdf

https://www.nutanix.com/uk/go/nutanix-pricing-vs-traditional-infrastructure-tco-roi-report

https://www.datacenterknowledge.com/cloud/report-openstack-private-cloud-tco-suffers-from-talent-shortage

https://www.rackspace.com/newsroom/rackspace-technology-launches-openstack-flex

오픈소스

기업의 IT인프라 변화에 대한 배경

오픈소스 클라우드

오픈소스 클라우드는 클라우드 컴퓨팅의 핵심 기술을 **오픈소스 소프트웨어로 구현**한 것으로, 누구나 자유롭게 사용, 수정, 배포가 가능.

이는 클라우드 인프라를 유연하고 비용 효율적으로 구축할 수 있게 해주며, 기술 독립성과 커뮤니티 기반 혁신을 촉진. 또한, 회사의 업무 및 워크로드에 맞추어서 언제든지 아키텍처 변형이 가능.

하지만, **벤더가 제공하는 오픈소스 기반**으로 제공하는 경우, 아키텍처에 제한 사항이 많이 발생.

이러한 이유로, **벤더 기반의 오픈소스**를 도입 시, **라이선스/아키텍처/기술 지원 부분**에 대해서 항상 조심하면서 확인이 필요하다.

2025-05-18

주요 오픈소스 클라우드 플랫폼

1. OpenStack

OpenStack은 NASA와 Rackspace가 공동 개발한 오픈소스 IaaS(서비스형 인프라) 플랫폼으로, 가상 서버, 네트워크, 스토리지 등 클라우드 인프라 자원을 통합 관리. 대규모 확장성과 유연한 구성으로 기업과 공공기관에서 널리 사용되고 있다.

2. CloudStack

CloudStack은 Apache 재단이 관리하는 오픈소스 IaaS 플랫폼으로, 사용자 친화적인 인터페이스와 자동화된 리소스 관리 기능을 제공. 중소규모 클라우드 환경에 적합하며, 다양한 하이퍼바이저와의 호환성을 갖추고 있다.

다만, 확장성이 오픈스택과 비교 하였을 때, 많이 부족하다는 평가가 있다.

주요 오픈소스 클라우드 플랫폼

3. Kubernetes

Kubernetes는 Google이 개발한 오픈소스 컨테이너 오케스트레이션 플랫폼으로, 컨테이너화된 애플리케이션의 자동 배포, 확장, 관리를 지원. 클라우드 네이티브 애플리케이션 개발에 표준 컨테이너 오케스트레이션 도구로 지원.

오픈소스 클라우드의 장점

- **비용 효율성:** 라이선스 비용 없이 클라우드 인프라를 구축할 수 있다.
- **유연성:** 조직의 요구에 맞게 시스템을 자유롭게 커스터마이징 할 수 있다
- **기술 독립성:** 특정 벤더에 종속되지 않고 자체 기술 역량을 강화할 수 있다.
- **커뮤니티 지원:** 활발한 오픈소스 커뮤니티를 통해 지속적인 기술 지원과 업데이트를 받을 수 있다.
- **자체 기술 개발:** 충분한 인력과 시간이 있는 경우, 상용 제품과 동일한 기능 및 아키텍처를 자체적으로 구현이 가능하다. 국내에서는 대표적으로 KT Cloud/NHN/NAVER가 있다.

비용 효율성

"라이선스 비용 없이 클라우드 인프라를 구축할 수 있다."

✓ 사실이다.

- OpenStack, Harvester, OKD 등은 무료로 사용할 수 있는 오픈소스이기 때문에, 상용 솔루션 대비 라이선스 비용이 없다.
- 단, 인건비(기술자 확보), 구축 시간, 유지보수 비용은 따로 고려해야 하며, 상용 솔루션 대비 초기 진입 장벽이 존재한다.

출처:

https://www.openstack.org/software/start/ (OpenStack 공식 소개 문서)

"Total cost of ownership for open-source cloud software" 보고서 등

2025-05-18

유연성

"조직의 요구에 맞게 시스템을 자유롭게 커스터마이징 할 수 있다."

✓ 정확하다.

- 오픈소스는 소스코드 접근이 가능하므로, 필요 시 기능 수정이나 구성 변경이 자유롭다.
- 예를 들어, OpenStack에서는 Neutron(네트워크 서비스)을 Calico, OVN, Open vSwitch 등으로 교체 가능하다.
- 커스터마이징이 쉬운 만큼, 기업 내부 정책/보안 규정에 맞춘 설계도 용이하다.

기술 독립성

"특정 벤더에 종속되지 않고 자체 기술 역량을 강화할 수 있다."

✓ 사실이다.

- 벤더 종속(lock-in)을 피할 수 있다는 점은 오픈소스 도입의 핵심 이유 중 하나다.
- 다만, OpenStack의 경우 일부 기업은 Red Hat OpenStack Platform, Canonical 등 상용 배포판을 사용하며 이때는 간접적인 벤더 종속이 발생할 수 있음.
- 실제로 NAVER Cloud는 자체 OpenStack/리눅스 배포판 개발, KT Cloud는 KOSMOS 프로젝트를 통해 기술 독립 추진.

커뮤니티 지원

"활발한 오픈소스 커뮤니티를 통해 지속적인 기술 지원과 업데이트를 받을 수 있다."

✓ 사실이다.

- OpenStack: Foundation 주도, 각 릴리즈에 수천 명의 개발자 기여.
- Kubernetes: CNCF 산하 프로젝트, 전 세계 대규모 커뮤니티 참여.
- 보안 패치, 버그 수정도 빠른 편이나, 기업은 내부적으로 적용 책임이 있음 (지원 계약은 별도 필요할 수 있음).

자체 기술 개발 가능성

"충분한 인력과 시간이 있는 경우, 상용 제품과 동일한 기능 및 아키텍처를 자체적으로 구현이 가능하다."

✓ 조건부 사실이다.

상용 서비스 수준에 도달하기 위해서는 운영 인프라 경험, DevOps 자동화, 고가용성 설계 역량이 요구된다.

- **국내 사례:**
  - KT Cloud: KOSMOS 플랫폼 (OpenStack 기반)
  - NHN Cloud: OpenStack 기반 클라우드 플랫폼 자립화
  - NAVER Cloud: 자체 OpenStack 기반 + Hypercloud 기술

관련 기사:

전자신문: 네이버·KT, 오픈스택 기반 클라우드 독립 추진

ZDNet Korea: NHN, 자체 클라우드 기술 내재화

2025-05-18

오픈소스 클라우드의 과제

오픈소스 기반으로 인프라를 운영하는 경우, 다음과 같은 과제가 항상 따른다.

항목	검증 결과	설명 및 근거
전문 인력 부족	✓ 사실	OpenStack, Kubernetes, Ceph 등은 복잡한 구조로 인해 네트워크/스토리지/보안 등 전반적 전문성 요구. 국내·국외 모두 인력 부족은 주요 장애 요인. 교육 비용도 지속적으로 발생.
보안 및 안정성	✓ 사실	오픈소스는 빠르게 패치가 이뤄지지만, 보안 적용 책임은 사용자에게 있음. 예: Log4j, DirtyPipe 등의 취약점 대응은 자체 역량에 의존. 고가용성, 백업 설계도 별도 필요.
라이선스 준수	✓ 사실	GPL, AGPL, Apache 등 각기 다른 조건 존재. 상업적 사용 시 오픈소스 컴플라이언스 위반 발생 가능. 예: Redis, Elastic 사례. 기업은 내부 감사 또는 OSS 검토 체계 필요.

2025-05-18

오픈소스 클라우드의 과제

위의 내용 계속 이어서…

항목	검증 결과	설명 및 근거
오픈소스 벤더 도입 시 메리트 저하	✓ 조건부 사실	Red Hat, Canonical, SUSE 등은 오픈소스를 엔터프라이즈 패키징하여 제공하므로, 기술 지원 비용 발생. 결과적으로 가격이 퍼블릭 클라우드와 유사해지는 경우도 많음.
오픈소스 락인 발생 가능	✓ 조건부 사실	이론상 자유롭지만, 실무에서는 담당자 기술 스택 선호, 운영 방식 고착, 유료 기술지원 의존으로 lock-in 발생. 예: RHEL, OpenShift, 특정 Helm/Kustomize 운영 패턴 고착 등.

전문 인력 부족

"오픈소스 클라우드 플랫폼을 효과적으로 운영하기 위한 전문 인력이 부족. 또한, 지속적인 교육을 통해 기술 업데이트가 필요함."

✓ 사실이다.

- 특히 **OpenStack, Kubernetes, Ceph 등은 구조가 복잡하고**, 제대로 운영하려면 인프라, 네트워크, 스토리지, 인증까지 전반적 지식이 필요함.
- 2023년 Linux Foundation 보고서에 따르면, **클라우드 네이티브 인력 수요는 높지만 공급은 부족**한 상태.
- 기술 진화가 빨라서 지속적인 교육/러닝 커브가 필요함.
- 국내에서는 **공공기관/중소기업에서 이 인력 부족이 특히 큰 진입장벽**이 되고 있음.

보안 및 안정성

"보안 취약점 관리와 시스템 안정성 확보에 대한 지속적인 노력이 필요."

✓ 정확하다.

- 오픈소스는 공개되어 있어 취약점이 빠르게 알려지는 반면, **적용 및 대응은 각 기업 책임**이다.
- 예: Log4Shell, Linux Kernel Dirty Pipe 등은 오픈소스 기반 시스템에 직접 영향.
- 안정성 확보를 위해선 **CI/CD 기반 테스트, 보안 스캐닝(Snyk, Trivy 등), 멀티노드 DR 설계** 등이 필요.
- **상용 서비스 수준(SLA)**에 도달하려면 상당한 모니터링 및 자동화 시스템이 필요함.

라이선스 준수

"오픈소스 라이선스 조건을 정확히 이해하고 준수."

✓ 법적 책임이 따르므로 매우 중요한 사실이다.

- GPL, AGPL, Apache 2.0, MIT 등 **각 라이선스마다 소스코드 공개 범위가 다르며**, 특히 **상업적 배포 시 오용 우려가 큼**.
- 예: Redis는 오픈소스지만, 특정 클라우드에서 상업적으로 재판매되며 라이선스 논쟁 발생 (→ Redis License로 전환).
- 기업은 **오픈소스 컴플라이언스 정책**을 마련하거나, 전문 컨설팅을 받기도 함.

오픈소스 벤더 도입 시 가격 및 아키텍처 메리트 약화

"벤더를 통해 오픈소스를 도입하면 상용 소프트웨어와 가격/구조 메리트가 떨어진다."

✓ 조건부 사실이다.

- 예: Red Hat OpenStack, SUSE Rancher, Canonical 등은 기술 지원 계약이 존재하며, 이 경우 **라이선스는 무료지만 기술지원 계약으로 비용 발생**.
- 이 경우 가격이 **상용 클라우드 서비스와 유사하거나 높아질 수 있음.**
- 또한 **패키징된 오픈소스의 아키텍처가 벤더에 따라 lock-in**될 수도 있음.
  - 예를 들어서, 레드햇 경우에는 업 스트림에서 다운스트림으로 내려오는 경우, 무조건 재설치가 요구
- 다만, **코드를 소유하고 벤더 없이 직접 운영하는 경우** 이러한 비용은 줄일 수 있음.
  - 수세 리눅스 및 수세 계열의 제품은 상용버전과 오픈소스 버전을 동일하게 운영하고 있음
  - 컴플라이언스 위반에 대해서는 수세 리눅스가 상대적으로 느슨함

2025-05-18

유료 기술 지원을 전제로 제공한다.

✓ 정확하다.

이들 벤더(SuSE/RedHat/Canonical)는 오픈소스를 기반으로 한 엔터프라이즈 배포판을 제공하고, 이에 대해 기술지원(Support Subscription)을 판매한다.

Red Hat은 RHEL, RHOSP, OpenShift 등 모두 서브스크립션 기반이고, Canonical도 Ubuntu Pro나 LTS 지원은 유료다.

SUSE Rancher도 공식 배포판은 무료지만, 엔터프라이즈 기능(예: Rancher Prime, SUSE Manager 등)은 지원 계약이 필요하다.

Red Hat은 호환성 제한이 있다.

✓ 사실이다.

예: RHEL 계열은 CentOS 종료 후 **`Rocky Linux, AlmaLinux`**가 분기되었고, RHEL 내부 커널/라이브러리와의 완전한 호환을 보장하지 않는다.

Red Hat 제품(RHEL, OpenShift 등)은 특정 버전에 tightly coupled된 구조로, 업스트림 버전으로 마이그레이션이 쉽지 않음 (재설치, 설정 변경 필요).

Red Hat은 정책적으로 소스코드는 공개하되, **바이너리/문서/CI 결과물은 유료 사용자에게만 배포**하는 전략을 택함.

SUSE는 거의 동일하게 운영된다.

✓ 상대적으로 사실이다.

SUSE는 **오픈소스 커뮤니티 버전(예: openSUSE Leap, openSUSE MicroOS, Harvester)**과 **상용 버전(SLES, SUSE Rancher)** 간의 기능 차이가 다소 적다.

특히 Harvester는 GitHub에서 전체 코드와 운영 매뉴얼이 공개되어 있고, 커뮤니티 사용자도 실질적 운영이 가능하다.

단, Prime Subscription 사용자에 한해 보안 패치 선적용, 긴급 지원, SLA, 인증 지원 등이 제공되며 이것이 주요 차이점이다.

SUSE의 컴플라이언스 정책은 상대적으로 느슨하다.

✓ 상대적인 표현으로 볼 수 있다.

SUSE는 오픈소스 철학을 강조하는 만큼, GPL/LGPL 컴플라이언스를 엄격하게 관리하긴 하지만, 사용자에게 강제적인 검증 절차 요구가 적고, 상업적 활용에 대해 RHEL보다 유연한 편이다.

예: SLES 운영 중 패키지 일부를 교체하거나 커스터마이징해도 기술 지원 거절 사례가 적음 (반면 RHEL은 바이너리 변경 시 기술 지원 제한이 있음).

오픈소스 인력 환경

451 Research의 보고서에 따르면, **OpenStack 기반 프라이빗 클라우드**는 **라이선스 비용이 없고 초기 비용이 낮지만, 전문 인력 확보의 어려움**으로 인해 **운영 비용이 증가**할 수 있다.

- **인력 비용**: OpenStack 전문가의 수요가 높아 인건비가 상승하고 있다.
- **운영 복잡성**: OpenStack의 복잡한 구성과 관리로 인해 운영 비용이 증가할 수 있다.

자료: datacenterknowledge.com

오픈소스 프라이빗 클라우드

항목	퍼블릭 클라우드 (AWS, Azure 등)	오픈소스 프라이빗 클라우드 (OpenStack 등)
초기 투자 비용	낮음 (CapEx 없음)	중간~높음 (서버, 스토리지, 네트워크 장비 등)
운영 비용	사용량 기반 변동 (예측 어려움)	고정 비용 (예측 가능)
데이터 전송 비용	높음 (예: AWS egress $0.09/GB)	낮음 또는 없음
라이선스 비용	있음 (예: VMware, RDS 등)	없음 (오픈소스 사용)
인건비	낮음 (벤더 관리)	중간~높음 (전문 인력 필요)
5년간 TCO	높음 (사용량 증가 시 급격히 상승)	중간~높음 (초기 투자 후 안정적인 비용 구조)
비용 예측성	낮음 (사용량 변동에 따라 비용 변동)	높음 (고정 비용 구조)
확장성 및 유연성	높음 (글로벌 리전, 다양한 서비스 제공)	중간 (내부 인프라 확장 필요)
보안 및 규제 준수	중간 (벤더에 의존)	높음 (자체 통제 가능)

결론

오픈소스는 자유롭게 사용할 수 있다는 장점이 있지만, 벤더를 통해 배포된 상용 오픈소스는 실질적으로 **기술 지원 및 운영 편의를 제공하는 대가로 비용이 발생**하며, 구조적으로도 **벤더 종속(lock-in)** 위험을 동반할 수 있다.

특히 Red Hat은 업스트림-다운스트림의 경계를 분명히 하며 상용 고객 중심의 배포 전략을 강화하고 있는 반면, SUSE는 상대적으로 **커뮤니티와의 호환성, 컴플라이언스 요구 수준이 낮고 유연하게 운영되고 있다.**

따라서 도입 조직은 비용과 유연성, 기술 지원 수준을 종합적으로 고려하여 적절한 벤더 및 배포판을 선택해야 한다.

AWS

기업의 IT인프라 변화에 대한 배경

최근 AWS의 공공기관

AWS는 한국 공공 부문에 진출했지만, 제공 서비스는 제한적.

- CSAP 인증 등급: AWS는 2025년 3월, **CSAP 'Low' 등급(Group C)** 인증을 획득. 이는 공공기관 중에서도 **기초 지자체, 산하기관, 공공학교 등**에 해당하며, **민감한 데이터를 다루지 않는 시스템**에만 적용 및 사용이 가능.

마이크로소프트, 아마존

- 인증 범위: 현재 CSAP 인증 범위에 포함된 AWS 서비스는 **191개**로, 이는 전체 AWS 서비스 중 일부에 해당. 공공기관은 CSAP 인증 범위 내의 서비스만 사용이 가능.

최근 AWS의 공공기관

2. NCP, NHN 등 국내 사업자와의 비교: 서비스 범위와 등급에서 차이가 있다.

- 국내 사업자들의 CSAP 등급: 네이버클라우드(NCP), NHN, KT 등은 **CSAP 'High' 또는 'Medium' 등급**을 보유하고 있어, **중앙정부, 주요 공공기관, 민감 정보 시스템** 등 **보다 높은 보안 수준이 요구되는 분야**에 사용 가능.
- 서비스 제공 범위: 이러한 국내 사업자들은 **IaaS, PaaS, SaaS** 등 다양한 클라우드 서비스를 제공하며, **공공기관의 다양한 요구사항**을 충족.

최근 AWS의 공공기관

3. AWS의 향후 계획과 시장 전망

- **중장기 전략:** AWS는 향후 **CSAP 'Medium' 또는 'High' 등급** 획득을 통해 **중앙정부 및 민감 정보 시스템**에도 서비스를 제공할 계획을 가지고 있음. 그러나 이를 위해서는 **물리적 망 분리** 등 추가적인 보안 요건을 충족이 되어야 됨.

Microsoft LearnKorea Joongang Daily

- **시장 경쟁 구도:** AWS의 진출로 인해 국내 클라우드 시장에서의 경쟁이 더욱 치열해질 것으로 예상되며, 이는 **공공기관의 클라우드 도입 확대**와 **서비스 품질 향상**에 긍정적인 영향.

최근 AWS의 공공기관

항목	AWS (2025년 기준)	NCP / NHN / KT 등 국내 사업자
CSAP 인증 등급	Low (Group C)	Medium / High
제공 대상 기관	기초 지자체, 공공학교 등	중앙정부, 주요 공공기관 등
서비스 제공 범위	제한적 (191개 서비스)	광범위 (IaaS, PaaS, SaaS 등)
망 분리 방식	논리적 망 분리	물리적 망 분리 지원
향후 확장 가능성	Medium / High 등급 획득 계획	이미 다양한 등급 보유

정리

항목	온프레미스	프라이빗	퍼블릭	하이브리드	멀티
인프라 소유	자체 구축 및 소유	자체 또는 호스팅 기반	클라우드 사업자	프라이빗 + 퍼블릭 조합	여러 퍼블릭 사업자 혼합
운영 위치	내부 데이터센터	내부 또는 전용 호스팅 환경	외부 사업자 데이터센터	혼합: 내부 + 외부	퍼블릭 클라우드 다중 분산
운영 방식	수동 또는 로컬 자동화	자동화된 클라우드 방식 포함	완전 자동화, API 중심 운영	자동화 + 연동	자동화 + 분산관리
기술 구성	물리 + 하이퍼바이저 중심	하이퍼바이저 + 클라우드 스택	컨테이너 중심, 서버리스 포함	VM/컨테이너 혼합 가능	클라우드별 컨테이너 혼합
자원 프로비저닝	수동 또는 단순 스크립트	템플릿 기반 프로비저닝	API, CLI, UI로 자동 배포	이기종 자원 간 통합 관리	클라우드 간 독립적 배포

2025-05-18

정리

항목	온프레미스	프라이빗	퍼블릭	하이브리드	멀티
오픈소스 활용 예시	KVM, Libvirt, Cobbler 등	OpenStack, Harvester, oVirt 등	Terraform, Kubernetes, ArgoCD	OpenShift, KubeVirt, Rancher 등	KubeFed, Crossplane 등
보안 및 통제 수준	최고 (물리적 통제 가능)	매우 높음 (논리적 격리)	낮음~중간 (공유 환경)	민감정보는 내부, 일반은 외부 배치	CSP간 분산, 벤더 회피 용이
사용 목적	규제 환경, 고정 워크로드	내부 SaaS, 유연한 개발 환경	확장성, 비용 효율적 서비스 운영	연속성 확보, 혼합 워크로드 대응	벤더 락인 회피, 고가용성

정리

1. **예측 가능한 워크로드**: 장기적으로 안정적인 리소스 사용이 예상되는 경우, 프라이빗 클라우드가 비용 효율적입니다.
2. **탄력적인 워크로드**: 수요 변동이 큰 경우, 퍼블릭 클라우드의 유연한 확장성이 장점입니다.
3. **인력 확보**: OpenStack 기반 프라이빗 클라우드를 고려할 경우, 전문 인력 확보 및 교육이 중요합니다.
4. **하이브리드 전략**: 워크로드 특성에 따라 퍼블릭과 프라이빗 클라우드를 혼합하여 사용하는 하이브리드 클라우드 전략이 효과적일 수 있습니다.

클라우드 서비스 모델

이론

서비스 설명

클라우드 서비스 모델은 사용자가 IT 자원을 어떻게 소비하느냐에 따라 구분되며, 일반적으로 세 가지 핵심 모델로 분류된다. **IaaS(Infrastructure as a Service)**, **PaaS(Platform as a Service)**, **SaaS(Software as a Service)**이다. 각각은 사용자에게 제공되는 자원의 범위와 관리 책임 범위에 따라 차이가 있다.

1. IaaS (Infrastructure as a Service) – 인프라 서비스형

IaaS는 가장 기본적인 형태의 클라우드 서비스 모델로, 가상화된 컴퓨팅 자원(서버, 저장소, 네트워크 등)을 서비스로 제공한다. 사용자는 운영체제부터 애플리케이션까지 설치 및 구성할 수 있으며, 물리적인 자원 관리는 클라우드 제공자가 담당한다.

- **예시**: OpenStack, AWS EC2, Microsoft Azure VM, Google Compute Engine
- **사용 사례**: 시스템 관리자나 인프라 엔지니어가 서버를 직접 구성하거나, 기존 온프레미스 환경을 클라우드로 이전할 때 사용된다.
- **근거 자료**: NIST SP 800-145

서비스 설명

2. PaaS (Platform as a Service) – 플랫폼 서비스형

PaaS는 개발 환경을 클라우드 상에 제공하는 서비스로, 개발자는 서버나 네트워크, 운영체제 관리 없이 애플리케이션만 개발하고 배포하면 된다. 미들웨어, 데이터베이스, 런타임 환경 등이 포함되어 있어 개발 생산성을 높이는 데 유리하다.

- **예시**: Red Hat OpenShift, Google App Engine, Heroku, Cloud Foundry
- **사용 사례**: 애플리케이션 개발자가 소프트웨어를 빠르게 개발하고 배포할 수 있는 플랫폼을 요구할 때 유용하다.
- **근거 자료**: Gartner Glossary on PaaS

서비스 설명

3. SaaS (Software as a Service) – 소프트웨어 서비스형

SaaS는 최종 사용자에게 소프트웨어를 완전한 형태로 제공하는 모델이다. 사용자는 웹 브라우저 또는 클라이언트 앱을 통해 서비스를 이용하며, 설치나 유지보수는 모두 클라우드 제공자가 책임진다.

- **예시**: Google Workspace(Gmail, Docs), Microsoft 365, Salesforce, Zoom
- **사용 사례**: 이메일, 문서 작성, CRM 등 다양한 업무 도구가 필요할 때 간편하게 사용할 수 있다.
- **근거 자료**: McKinsey: The state of cloud computing

서비스 모델

이러한 서비스 모델은 단일하게 쓰이기보다는, 실제 기업 환경에서는 IaaS 기반 위에 PaaS를 얹고, SaaS를 병행하여 사용하는 **복합 아키텍처**로 구성되는 경우가 많다.

추가로 `BaaS(Backend as a Service)`, `FaaS(Function as a Service)` 등도 존재하지만, 이는 위 세 가지의 세부 확장 또는 특수화된 형태로 간주된다.

구분	IaaS	PaaS	SaaS
주요 대상	인프라 관리자, 시스템 엔지니어	개발자	최종 사용자
제공 항목	서버, 저장소, 네트워크 등	런타임, 미들웨어, DB 등	완성된 애플리케이션
관리 주체	사용자가 대부분 직접 관리	플랫폼은 제공자가 관리	제공자가 전부 관리
대표 예시	OpenStack, AWS EC2	OpenShift, Heroku	Gmail, Salesforce

2025-05-18

공유 책임 모델(Shared Responsibility Model) 개요

클라우드 컴퓨팅에서 **공유 책임 모델**은 클라우드 서비스 제공자(CSP)와 고객 간의 보안 및 운영 책임을 명확히 구분하는 프레임 워크. 이 모델은 사용 중인 클라우드 서비스 유형(IaaS, PaaS, SaaS)에 따라 책임 범위가 달라진다.

- **IaaS(Infrastructure as a Service)**: CSP는 물리적 인프라와 가상화 계층을 관리하며, 고객은 운영 체제, 애플리케이션, 데이터 등을 관리합니다.
- **PaaS(Platform as a Service)**: CSP는 인프라뿐만 아니라 운영 체제, 런타임 환경 등을 관리하며, 고객은 애플리케이션과 데이터를 관리합니다.
- **SaaS(Software as a Service)**: CSP가 대부분의 책임을 지며, 고객은 데이터와 사용자 접근 관리에 집중합니다.

이러한 모델은 고객이 보안 및 컴플라이언스 요구 사항을 충족하는 데 도움이 됩니다.

2025-05-18

오픈소스 프라이빗 클라우드와의 비교

오픈소스 프라이빗 클라우드는 조직이 자체적으로 클라우드 인프라를 구축하고 운영하는 모델로, `OpenStack/OpenNebula` 등이 대표적인 솔루션입니다.

책임 분담

- **공공 클라우드**: CSP와 고객 간의 책임이 명확히 구분되어 있으며, 고객은 CSP가 제공하는 보안 조치를 신뢰해야 합니다. Palo Alto Networks
- **프라이빗 클라우드**: 조직이 모든 인프라를 직접 관리하므로, 보안, 유지보수, 컴플라이언스 등의 모든 책임을 조직이 부담합니다.

2025-05-18

오픈소스 프라이빗 클라우드와의 비교

보안 및 컴플라이언스

- **공공 클라우드**: CSP는 다양한 보안 인증을 획득하여 고객에게 신뢰를 제공합니다.
- **프라이빗 클라우드**: 조직이 직접 보안 정책을 수립하고 적용해야 하며, 이는 높은 수준의 보안 제어를 가능하게 하지만, 더 많은 리소스와 전문 지식이 필요합니다.

정리

항목	공공 클라우드 (IaaS/PaaS/SaaS)	오픈소스 프라이빗 클라우드
인프라 소유권	CSP 소유	조직 소유
보안 책임 분담	CSP와 고객 간 공유	조직이 전부 책임
커스터마이징	제한적	높음
초기 투자 비용	낮음	높음
유지보수	CSP 담당	조직이 직접 수행

클라우드 보안 기준

이론

오픈소스 보안 인증의 중요성

오픈소스는 현대 클라우드 인프라의 핵심을 이루는 기술로 자리잡았다.

`Kubernetes`, `Ceph`, `OpenStack`, `Terraform`, `Ansible`등 클라우드의 거의 모든 계층은 오픈소스 기술에 의해 구성된다.

그러나 이처럼 유연하고 개방된 기술이 **실제 공공기관, 금융기관, 또는 보안이 중요한 산업 영역에서 채택되기 위해서는 반드시 '신뢰'라는 기준을 충족**해야 한다.

이 신뢰를 검증하고 외부에 입증하는 방법이 바로 **보안 인증(Security Certification)**이다.

오픈소스 보안 인증의 중요성

1. **보안 인증은 '신뢰할 수 있는 도구'임을 입증하는 공식적 근거다**

오픈소스는 누구나 열람하고 수정할 수 있지만, 이는 곧 **신뢰할 수 있는 보안 수준을 보장하지 않는다는 이중성**을 내포한다.

예를 들어 OpenSSL은 광범위하게 사용되지만, 내부 모듈이 FIPS 인증을 받았는지 여부에 따라 **정부 및 군기관에서는 사용 불가**로 분류되기도 한다.

즉, 오픈소스라고 해서 불안한 것이 아니라, **공식 인증을 받지 않은 상태에서는 법적 책임이나 위험 분석의 기준에 오를 수 없다.**

오픈소스 보안 인증의 중요성

2. **FIPS, NIST, ISMS 등은 클라우드 기반에서 신뢰할 수 있는 암호화 및 통제 기준을 제시한다**

클라우드 인프라는 이제 물리적으로 보이지 않는다.

이로 인해 데이터는 **'어디에 있는지'**보다 **'어떻게 보호되고 있는지'**를 중심으로 평가되며, 이때 핵심 평가 기준이 바로 FIPS 140-3, NIST SP 800-53, ISO/IEC 27001, ISMS-P와 같은 **보안 인증 프레임워크**다.

오픈소스 솔루션이 이러한 기준을 만족한다는 것은, 단순한 기능성 뿐 아니라 **암호화 수준, 접근제어, 로깅, 무결성, 보안 사고 대응체계까지 갖추고 있음을 의미**한다.

오픈소스 보안 인증의 중요성

3. **공공 및 민감 산업군에서는 '보안 인증 여부'가 채택 기준이 된다**

한국의 공공기관 및 금융/의료기관은 정보보호 관리체계(ISMS-P) 또는 클라우드 보안 인증(CSAP) 기준을 따르며, 이를 만족시키기 위해서는 구성된 소프트웨어와 플랫폼이 **보안 인증을 획득한 모듈로 구성되어야** 한다.

따라서 기업은 FIPS 인증된 암호 모듈을 사용하는 OpenSSL, NIST 기준을 따르는 컨테이너 보안 정책 도구(e.g., Kyverno, Open Policy Agent), 또는 한국인터넷진흥원(KISA)의 인증을 받은 오픈소스 기반 솔루션을 적극적으로 검토하게 된다.

오픈소스 보안 인증의 중요성

4. **보안 인증은 커뮤니티의 책임을 상업적/공공 영역까지 확장시키는 수단이다**

오픈소스는 기술적으로 자유롭지만, 보안 인증은 해당 소프트웨어가 **상업 환경이나 제도권 서비스에서 사용될 수 있도록 신뢰성을 구조화 하는** 장치다.

이는 단지 기술적인 검증을 넘어서, **운영자와 사용자 간 책임 분리를 명확히 하고, 커뮤니티의 신뢰성을 시장의 신뢰로 전환**하는 중요한 수단이 된다.

2025-05-18

FIPS 인증 현황 (미국 연방 정보처리 표준)

SuSE Rancher (RKE2)

- `Rancher Kubernetes Engine 2 (RKE2)`는 FIPS 140-2 인증을 받은 암호화 모듈을 사용하여 보안성 강화.
- Rancher Government Solutions는 RKE2에 대해 FIPS 140-2 인증을 획득하였으며, 이는 미국 국방부 및 민간 기관에 적합한 보안 표준을 충족.

FIPS 인증 현황 (미국 연방 정보처리 표준)

SuSE Harvester

- `Harvester Government`는 운영 체제 수준에서 FIPS 140-2 및 STIG(보안 기술 구현 가이드) 준수.
- 이는 정부 기관 및 군사용 사례에 적합하도록 설계.

FIPS 인증 현황 (미국 연방 정보처리 표준)

Redhat OpenShift Container Platform

- OpenShift Container Platform은 FIPS 모드를 지원하며, RHEL(Red Hat Enterprise Linux) 또는 RHCOS(Red Hat CoreOS)에서 FIPS 모드로 설치하여 운영 가능.
- FIPS 모드를 활성화하면 OpenShift의 구성 요소들이 FIPS 인증을 받은 암호화 라이브러리를 사용.

FIPS 인증 현황 (미국 연방 정보처리 표준)

Redhat OpenShift Virtualization

- OpenShift Virtualization은 OpenShift Container Platform의 확장 기능으로, FIPS 모드에서 운영 가능.
- 이는 가상 머신과 컨테이너 워크로드를 통합하여 보안성을 유지하면서 운영할 수 있도록 지원.

국내 인증

CSAP 인증은 한국 공공기관에서 민간 클라우드 서비스를 이용하기 위한 필수 절차로, 기본, 표준, 고급 3단계로 나뉘며, 인증 등급에 따라 사용할 수 있는 데이터의 민감도와 서비스 영역이 다르다.

특히 고급 인증은 개인정보 및 국가 주요정보를 다루는 시스템에서 요구되며, 기술적·관리적 요구사항이 가장 높다.

등급 구분	설명	대상 서비스 예시
기본 (IaaS 전용)	인프라형 클라우드 서비스(IaaS)에 대한 기본 보안 수준 인증	VM, 스토리지, 네트워크 등 기본 IaaS
표준 (IaaS, PaaS, SaaS)	공공기관이 실제로 활용할 수 있는 보안 수준 확보 (가장 많이 요구됨)	VM + DB + 개발플랫폼 + 메일, 협업 등
고급 (중요 정보처리 대상용)	개인정보/중요정보를 다루는 고위험군 업무 대상. 강화된 보안 요구사항 반영	복지, 의료, 금융 등 민감정보 처리 시스템

2025-05-18

CSAP 인증 구조 요약

1. **인증 대상:**
   - 민간 클라우드 서비스 제공자(CSP)
   - 인증 범위는 IaaS / PaaS / SaaS에 따라 다름
2. **인증 항목:**
   - 기술적, 관리적, 물리적 보호 조치 등 117개 통제 항목 (표준 기준)
   - 고급은 142개 이상 항목 (예: 내부망 단절, 망연계, 탐지 시스템 등 포함)
3. **인증 유효기간:**
   - 3년간 유효, 매년 사후점검(정기 심사) 필요
4. **주관 기관:**
   - KISA (한국인터넷진흥원)
   - 신청은 CSAP 홈페이지에서 가능

2025-05-18

CSAP 인증 구조 요약

현재 클라우드 업체의 인증 상태는 다음과 같다.

구분	KT Cloud	NHN Cloud	AWS	Naver Cloud
IaaS	기본/표준 인증 보유	표준 인증	표준 인증 (Seoul 리전 한정)	고급 인증
SaaS	일부 서비스 인증	일부 인증	없음 (별도 신청 필요)	일부 고급 서비스 인증
고급 인증	일부 인증	진행 중	해당 없음	보유 (의료/복지 서비스용)

2025-05-18

Container/Virtual Machine 기반 하이브리드 워크로드

이론

2025-05-18

설명

하이브리드 워크로드란, 서로 다른 인프라 환경에서 동시에 분산되어 실행되는 애플리케이션 구성 요소 또는 서비스 단위를 말한다.

이는 일반적으로 **가상머신(VM)**과 **컨테이너(Container)**, 또는 **온프레미스 인프라**와 **퍼블릭 클라우드 환경**이 함께 사용되는 구조에서 나타나며, **워크로드의 일관성과 유연성 확보를 동시에 추구**하기 위해 활용된다.

이러한 구조는 기업이 기존의 레거시 시스템을 유지하면서 점진적으로 클라우드 기반으로 전환하고자 할 때 유용하게 작용한다.

예를 들어, 핵심 금융 시스템이나 ERP와 같은 민감하거나 변경이 어려운 서비스는 여전히 온프레미스 VM에서 실행되고, 사용자 인터페이스나 외부 API 같은 상대적으로 유연한 구성 요소는 클라우드 컨테이너에서 운영되는 형태가 이에 해당한다.

2025-05-18

설명

하이브리드 워크로드는 단순히 인프라를 나누어 사용하는 것에 그치지 않고, **다양한 플랫폼 간의 연동과 일관된 배포 및 관리를 가능하게 하는 기술 스택의 통합**이 핵심이다.

이 과정에서 `Rancher`, `OpenShift`, `Anthos`, `Azure Arc` 등의 플랫폼이 자주 활용되며, 인프라 간 보안과 연결을 위해 `SD-WAN`, `VPN`, `Direct Connect`와 같은 네트워크 기술도 함께 사용된다.

이러한 하이브리드 구조는 **▲보안 요구사항이 상이한 워크로드의 분리, ▲클라우드 자원의 확장성 활용, ▲비용 효율적인 리소스 배분, ▲재해복구(Disaster Recovery) 및 클라우드 버스팅(Cloud Bursting) 전략 구현** 등에 매우 효과적.

따라서 하이브리드 워크로드는 단순한 기술 선택이 아닌, **기업 IT 전략의 유연성과 지속 가능성을 높이는 핵심 수단**으로 자리잡고 있다.

2025-05-18

실행 환경 혼합

가상머신 + 컨테이너 혼용

- 예: 백엔드는 VM에서 실행, 프론트엔드는 Kubernetes 컨테이너에서 실행
- 현재: 쿠버네티스에서는 컨테이너/가상머신 워크로드를 동시에 지원하고 있음
- VMware 경우에는 Tanzu를 통해서 하이브리드를 제공하고 있음

실행 환경 혼합

온프레미스 + 퍼블릭 클라우드

- 예: 내부 ERP 시스템은 온프레미스, 외부 API 서비스는 AWS에서 운영

클라우드 간 연계

- 예: GCP에서 AI 연산 → 결과를 Azure Blob에 저장

API 게이트웨이 워크로드

Container/VM 기반 하이브리드 워크로드는 다양한 애플리케이션 컴포넌트를 컨테이너 혹은 가상머신 위에서 동시에 실행시키는 아키텍처다.

컨테이너는 경량화된 실행 환경을 제공하며 빠른 배포와 확장이 가능하고, 가상머신은 보안 격리와 레거시 애플리케이션 호환성에 강점을 가진다.

이러한 이질적인 환경 위에서 동작하는 서비스들을 외부에 안전하게 제공하고, 경로를 통제하며, 인증/인가를 중계하고, 로드밸런싱을 수행하는 것이 API Gateway의 역할이다.

즉, 하이브리드 워크로드 상에서 돌아가는 각종 **마이크로서비스(컨테이너 기반)**와 **모놀리식 앱(가상머신 기반)**을 **API Gateway가 통합된 진입점(Entry Point)**으로 감싸게 되며, 내부 리소스 구조를 외부 사용자로부터 추상화 한다.

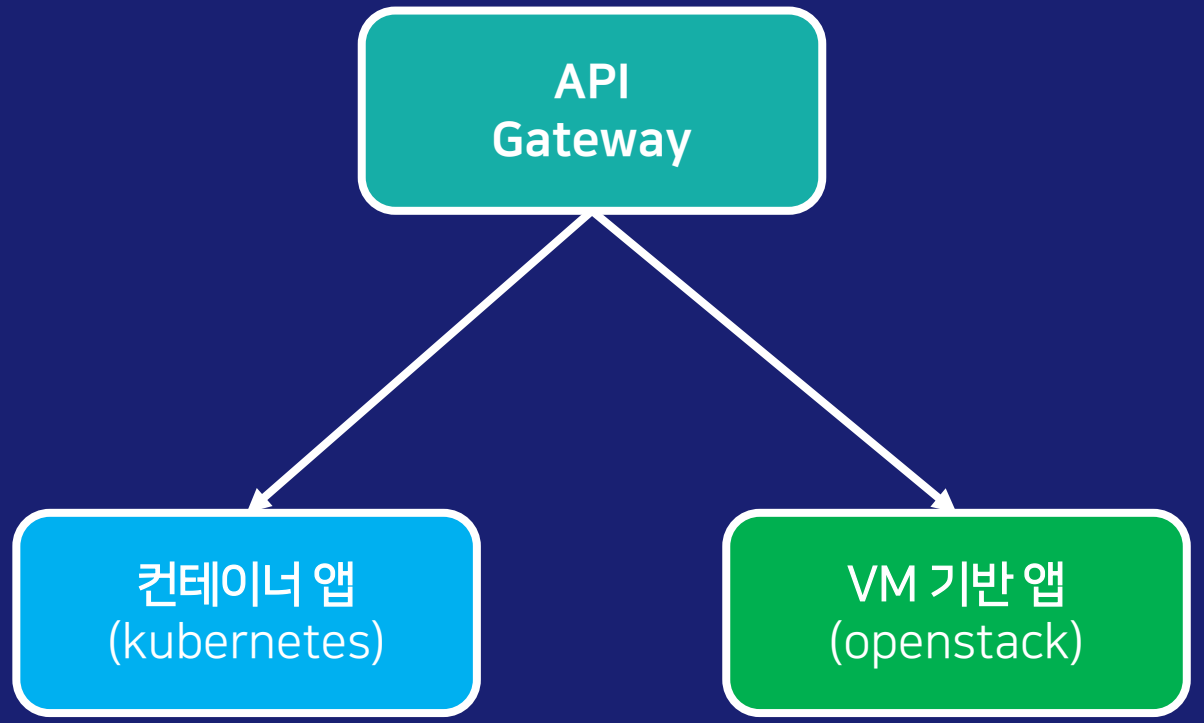
2025-05-18

API 게이트웨이 워크로드

Kubernetes 클러스터에서 돌아가는 컨테이너 앱과, OpenStack VM 위에서 돌아가는 레거시 웹 앱이 함께 운영되는 환경에서, 사용자는 하나의 API Gateway(예: Kong, NGINX, Istio, Ambassador 등)를 통해 두 서비스 모두 접근하게 된다.

이 Gateway는 내부 라우팅 규칙에 따라 컨테이너 서비스는 Service Mesh로, VM 서비스는 외부 네트워크로 프록시 해준다.

API 게이트웨이 워크로드



2025-05-18

하이브리드 워크로드

KubeVirt는 쿠버네티스(Kubernetes) 상에서 가상머신(VM)을 네이티브하게 운영할 수 있게 해주는 오픈소스 프로젝트다.

이 환경에서는 기존의 컨테이너 앱(Kubernetes Pod)과 더불어, VM 기반의 워크로드(VirtualMachineInstance, VMI)도 Kubernetes 리소스로써 배포·운영된다.

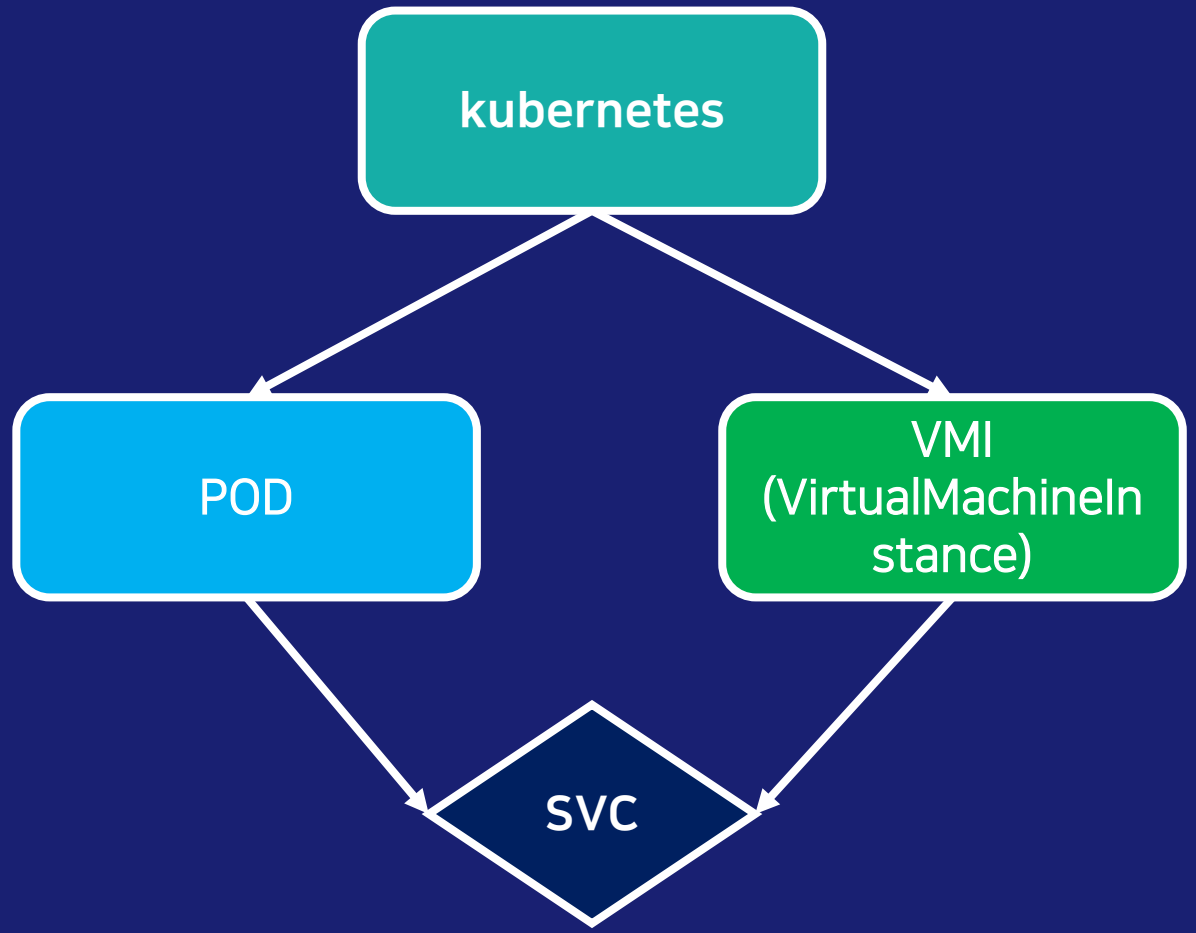
따라서 API Gateway를 사용하면, 단일 진입점 역할을 하면서, Kubernetes 내부의 SVC로 등록된 컨테이너와 가상머신 모두를 라우팅할 수 있게 된다.

하이브리드 워크로드

1. **UI / 클라이언트:** 최상위 사용자 인터페이스
2. **API Gateway:** 외부 요청을 Kubernetes SVC로 전달
3. **Kubernetes SVC:** 컨테이너 앱과 VMI 모두에 적용
4. **Pod (컨테이너 앱):** 전통적인 Kubernetes 서비스
5. **VMI (KubeVirt 기반 VM):** VM이지만 Pod와 같이 동작

항목	내용
VM 관리 방식	Kubernetes 리소스로 정의 (YAML)
배포 명령어	kubectl apply -f vm.yaml
VM 인터페이스 구성	Pod처럼 네트워크, 볼륨 구성 가능
API 통합	컨테이너와 VM 모두 SVC로 라우팅 가능

하이브리드 워크로드(kube-virt)



모놀릭 서비스

모놀리식 아키텍처는 하나의 응용 프로그램이 모든 기능을 하나의 코드베이스로 구성하여 하나의 단일 프로세스로 배포되는 구조를 따른다.

모든 기능(예: 사용자 인증, 상품 관리, 주문 처리 등)이 서로 강하게 결합되어 있고, 동일한 런타임 환경에서 작동한다.

이 방식은 초기 개발이 간단하고 배포도 쉬운 장점이 있지만, 규모가 커질수록 관리와 유지보수가 어려워지며, 부분적인 장애가 전체 시스템에 영향을 줄 수 있다.

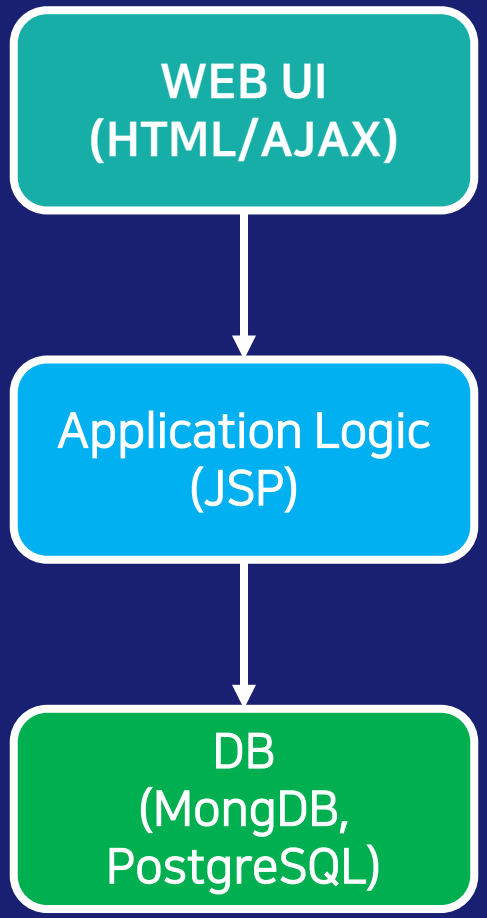
2025-05-18

모놀릭 서비스

특징

- 단일 코드베이스, 단일 배포 단위
- 기능 간 강한 결합(tight coupling)
- 특정 모듈의 오류가 전체 시스템에 영향을 줌
- 확장이 제한적 (주로 수직 확장)
- 기술 스택 통일이 요구됨

모놀릭 서비스


WEB UI
(HTML/AJAX)
Application Logic
(JSP)
DB
(MongDB,
PostgreSQL)

마이크로 서비스

마이크로서비스 아키텍처는 응용 프로그램을 여러 개의 독립적인 작은 서비스로 분리하여 구성한다.

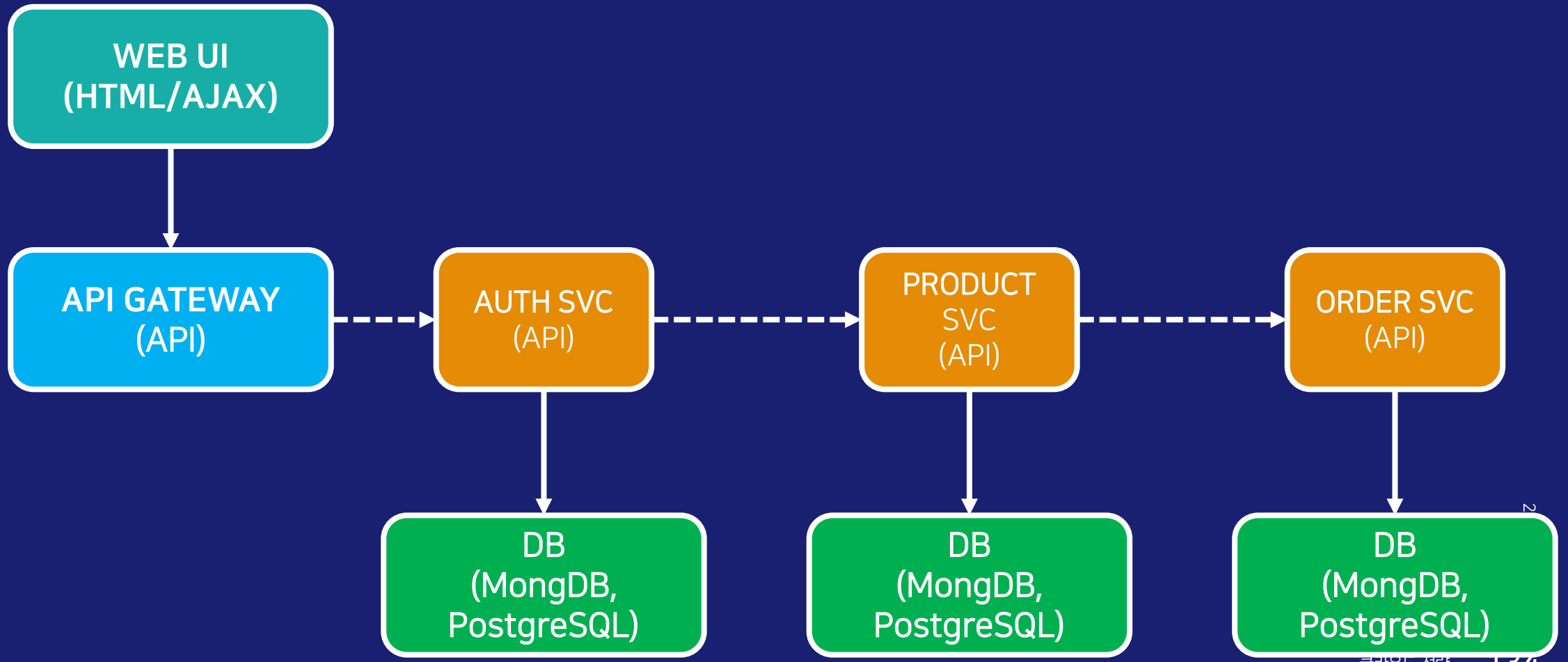
각 서비스는 고유한 기능을 담당하며, 독립적으로 개발, 배포, 확장할 수 있다. 서비스 간은 네트워크 기반의 API(주로 REST, gRPC 등)를 통해 통신하며, 각 서비스는 자체 데이터베이스를 가질 수도 있다.

이 구조는 복잡성은 높지만, 유연성과 확장성이 매우 뛰어나다.

특징:

- 독립적인 서비스로 기능 분리
- 느슨한 결합(loose coupling)
- 독립적인 배포와 스케일링 가능
- 장애 격리와 회복이 쉬움
- 서비스마다 다른 기술 스택 사용 가능 (Polyglot)

마이크로 서비스


WEB UI
(HTML/AJAX)
API GATEWAY
(API)
AUTH SVC
(API)
PRODUCT
SVC
(API)
ORDER SVC
(API)
DB
(MongDB,
PostgreSQL)
DB
(MongDB,
PostgreSQL)
DB
(MongDB,
PostgreSQL)

워크로드 혼합

상태 저장(Stateful) + 상태 비저장(Stateless)

- 예: 데이터베이스는 상태 저장, 웹서버는 상태 비저장 구조로 구성

마이크로서비스 + 모놀리식 통합

- 예: 일부 신규 기능은 마이크로서비스로 이관 중, 나머지는 기존 모놀리식 서버 유지

왜 하이브리드 워크로드를 사용하는가?

다음과 같은 목적으로 하이브리드 클라우드를 구성 및 사용한다.

목적	설명
비용 최적화	자주 사용하지 않는 서비스는 저렴한 퍼블릭 클라우드로 이관
보안 / 규제 대응	민감 데이터는 온프레미스에, 비민감 서비스는 외부로 운영
확장성과 유연성	트래픽이 많을 땐 클라우드로 확장 (Cloud Bursting 구조)
점진적 클라우드 전환	기존 시스템을 유지하며 일부만 클라우드로 이전 (Brownfield 전략)

대표 기술 예시

대표적인 하이브리드 기술을 제공하는 도구는 다음과 같다.

기술 영역	도구/플랫폼
VM 기반	VMware, KVM, Hyper-V
컨테이너 기반	Docker, Kubernetes, OpenShift
하이브리드 플랫폼	Rancher, Anthos, Azure Arc, Red Hat ACM
네트워크 연동	VPN, Direct Connect, SD-WAN
관리/자동화	Terraform, Ansible, ArgoCD, Crossplane

클라우드 주요 도구 및 오케스트레이션

이론/랩

설명

클라우드 오케스트레이션 도구를 이해하기 위해서 다음과 같이 분류한다. 여기서 클라우드 오케스트레이션에 포함되는 영역은 `API/CONTAINER/VIRTUAL MACHINE`이렇게 포함이 된다.

1. 컨테이너 도구
2. 컨테이너/가상머신 오케스트레이션 도구
3. 하이브리드 클라우드 도구

프라이빗/퍼블릭 클라우드는 일반적으로 API를 통해서 통합 및 운영이 된다. 역시, 컨테이너/가상화 오케스트레이션 도구도 대다수가 API를 통해서 외부에서 제어 및 접근이 가능하다.

이러한 이유로, 각 계층별 클라우드 도구를 확인하고 이들을 어떠한 오케스트레이션 도구를 통해서 API구성 및 관리를 해야 할지 확인해야 한다.

2025-05-18

설명

프라이빗 및 퍼블릭 클라우드 인프라는 일반적으로 `RESTful API`를 통해 상호 작용하며, 이러한 통합 구조는 오케스트레이션 도구의 사용을 전제로 한다.

컨테이너와 가상머신 역시 마찬가지로, 대부분의 관리 및 제어 작업은 API를 통해 외부에서 접근 가능하며, 이는 클라우드 자원의 선언적 정의와 동적 제어를 가능하게 만든다.

따라서, 각 계층, 즉 **컨테이너**, **가상머신**, 그리고 클라우드 자원을 이해하기 위해서는 단순히 구성 도구 자체만이 아니라, 이들이 어떤 API 기반 오케스트레이션 체계 위에서 작동하는가를 파악하는 것이 중요하다.

오늘날의 클라우드 인프라는 단순한 자원 배치의 문제를 넘어, 분산된 자원의 상호 연계, 일관성 유지, 상태 추적을 요구한다. 이 과정에서 핵심은 단순한 자동화가 아닌 **정책 기반의 오케스트레이션, 그리고 플랫폼 간의 상태 일치 `(infrastructure reconciliation)`**이다.

2025-05-18

설명

아름이는 특히 오케스트레이션 도구를 단순히 자원 배포 수단이 아닌, 조직의 클라우드 전략을 코드로 표현하는 인터페이스로 바라본다.

예를 들어, `Kubernetes`의 `CRD(Custom Resource Definition)`를 통해 컨테이너 뿐 아니라 클라우드 자원까지 정의하고 제어할 수 있는 `Crossplane`, 그리고 하이브리드 환경에서의 포털형 제어와 자동화를 제공하는 `ManageIQ` 같은 도구는 단지 기술 선택이 아니라 운영철학의 결정이라고 말할 수 있다.

또한, 오케스트레이션 도구가 API를 호출하는 구조라는 점은, 결국 API의 보안성, 연동 정책, 호출 제한 등의 측면까지도 오케스트레이션 전략에 포함되어야 함을 시사한다.

이는 클라우드 네이티브 환경에서 `"오케스트레이션"`이 단순한 인프라 자동화를 넘어서, 서비스 아키텍처 전체의 구조와 거버넌스를 반영하는 계층이라는 것을 의미한다.

2025-05-18

설명

CNCF(Cloud Native Computing Foundation) 및 **NIST**의 클라우드 오케스트레이션 정의에 따르면,

- 오케스트레이션은 클라우드 자원의 자동화된 조정(control) 및 운영(operation)을 포함하며, 이는 API를 통해 구현된다.(NIST SP 800-145, CNCF Glossary)

OpenStack, **Kubernetes**, **Terraform**, **Crossplane**, **Ansible** 등 거의 모든 현대 오케스트레이션 도구는 RESTful API or gRPC 기반 통신을 사용함 → 외부 제어 가능 구조.

2025-05-18

오케스트레이션 도구의 분류

1. **컨테이너 도구:** Docker, Podman(CNCF)
2. **컨테이너/VM 오케스트레이션 도구:** Kubernetes, OpenStack, Nomad
3. **하이브리드/멀티 클라우드 도구:** Rancher, Crossplane, Anthos

이들 모두는 API를 기반으로 외부에서 제어 및 접근이 가능함

오케스트레이션은 전략이다

- 단순한 자동화가 아닌 정책 기반 상태 관리
- 오케스트레이션 도구는 조직의 운영 철학을 코드로 표현하는 인터페이스
- 서비스 간 상태 동기화, 정책 적용, 연계성 관리가 핵심

정책 기반 오케스트레이션

- 인프라 변경은 반드시 정책과 조건에 따라 실행됨
- 예: 리소스 과부하 시 확장 정책, 보안 그룹 자동 지정, 태그 기반 배포 제한
- 활용 도구: OPA(Gatekeeper), Kyverno, Ansible 정책 템플릿

클라우드 API 게이트웨이 설계 개념

- **목적:** 마이크로서비스 및 오케스트레이션 API 보호 및 라우팅
- **구성요소:**
  - 인증(Authentication) / 권한(Authorization)
  - 속도 제한(Rate Limiting), 로깅 및 감사(Audit)
  - 서비스 디스커버리, 라우팅 정책
- **대표 도구:** `Kong, Apigee, Ambassador, Istio (Ingress Gateway)`

결론: 오케스트레이션은 API의 통제권을 가진다

- API는 자원의 제어 창구
- 오케스트레이션은 그 API를 전략적 관점에서 활용하는 도구
- 계층별 오케스트레이션 도구 + 정책 기반 통제 + 보안된 API = 완전한 클라우드 운영 구조

표준 컨테이너 도구 소개

이론+랩

Docker vs Moby vs CNCF

먼저, 해결해야 될 부분이 몇 가지가 있다. 왜냐면, 많은 사람들이 Docker를 표준이라고 생각하는 부분, 그리고 여전히 CNCF에서 여전히 도커를 지원한다는 부분이다. 실제로 이 부분은 다음과 같이 수정이 되어야 한다.

1. Docker는 Docker Inc.의 제품이며 CNCF 프로젝트가 아님
2. Moby는 Docker의 내부 컴포넌트화 프로젝트이며, 역시 CNCF와는 무관함
3. containerd는 CNCF가 주관하는 공식 런타임이고, Kubernetes에서도 이를 기본 런타임으로 채택하고 있어 CNCF Projects

Docker vs Moby vs CNCF

CNI 경우에는 다음과 같은 도구들이 CNCF에서 표준 네트워크 도구로 지원하고 있다.

1. Calico
2. Cilium
3. Flannel

CNI(Container Network Interface)는 Kubernetes에서 컨테이너 네트워크 설정을 표준화하기 위한 인터페이스이며, Calico, Cilium, Flannel과 같은 도구들은 이 CNI 사양을 구현하거나 기반으로 동작하는 네트워크 솔루션이다.

Kubernetes에서는 고수준, 저수준 네트워크 기능 모두 CNI 기반 플러그인을 통해 네트워크를 설정하고 연결한다.

2025-05-18

CNI?

Calico / Cilium / Flannel은 CNI 플러그인이다?

정확하게는 *CNI 사양을 구현한 네트워크 플러그인을 포함하는 네트워크 솔루션들*이다.

- **Calico**: 보안 정책과 L3 기반 라우팅을 지원하는 고성능 네트워크 솔루션 (CNI 지원)
- **Cilium**: eBPF 기반 고급 네트워크 정책과 관찰 기능 제공 (CNI 플러그인 포함)
- **Flannel**: 단순한 오버레이 네트워크 (vxlan 등)를 제공하는 경량 CNI 플러그인

CNI?

최근 오픈소스 기반의 Kubernetes 네트워크 환경은 **Cilium 중심으로 전환되는 추세**이며, Calico는 여전히 널리 사용되지만, 정책 구현 방식이나 고급 기능 면에서 Cilium에 비해 상대적으로 제한적이라는 평가를 받는다.

라이선스 측면에서는 **Calico 또한 오픈소스(Apache 2.0)**이지만, 일부 고급 기능은 **상용 구독 모델(Tigera Enterprise)**로 제공되기 때문에 **완전한 오픈소스 생태계를 지향하는 조직에서는 Cilium을 선호**하는 경우가 늘고 있다.

2025-05-18

CNI?

"CNCF에서 표준 네트워크 도구로 지원하고 있다"의 의미는?

Flannel은 CNCF Incubating Project였으나 아카이브됨 → 유지보수 중단.

- Calico, Cilium은 현재 CNCF 산하 프로젝트가 아니며, 각기 Tigera, Isovalent에 의해 개발됨. 하지만 Kubernetes 커뮤니티에서 널리 사용되는 사실상의 표준임.
- 진정한 의미의 "CNCF 표준"은 CNI 사양 자체와 이를 구현한 플러그인 구조에 있음.

Moby Project

특히, Moby는 도커의 오픈소스 프로젝트 이지만, 표준에는 크게 영향을 주지 않았다. Moby프로젝트는 여전히 Docker 라이브러리 영역에서 활용하고 있다.

1. **Docker:** Docker는 Docker Inc.에서 개발한 컨테이너 플랫폼으로, `CNCF(Cloud Native Computing Foundation)`의 공식 프로젝트가 아닙니다.
2. **Moby:** Moby는 Docker의 내부 컴포넌트화 프로젝트로, CNCF와는 무관합니다.
3. **containerd:** containerd는 Docker에서 분리된 컨테이너 런타임으로, CNCF의 공식 프로젝트입니다.

Moby와 containerd의 관계

Moby는 Docker Inc.가 Docker 엔진의 구성 요소를 오픈소스로 분리하여 만든 프로젝트로, 컨테이너 시스템을 구성하는 다양한 컴포넌트 별도로 사용이 가능하도록 라이브러리 형태로 제공.

`mobyproject.org`

containerd는 원래 Docker의 내부 구성 요소였으나, 이후 독립적인 오픈소스 프로젝트로 분리되었고, 현재는 CNCF(Cloud Native Computing Foundation)의 공식 프로젝트로 관리.

Moby는 containerd를 기본 컨테이너 런타임으로 사용하며, containerd는 Moby의 핵심 구성 요소 중 하나.

containerd와 Kubernetes의 CRI 연동

Kubernetes는 컨테이너 런타임과의 통신을 위해 `CRI(Container Runtime Interface)`를 사용합니다. 이 인터페이스를 통해 kubelet은 다양한 컨테이너 런타임과 통신.

containerd는 CRI를 지원하기 위해 자체적으로 CRI 플러그인을 제공하며, 이를 통해 Kubernetes와 통합.

과거에는 containerd와 Kubernetes를 연동하기 위해 별도의 어댑터인 cri-containerd가 사용되었으나, 현재는 containerd에 CRI 기능이 내장되어 있어 별도의 어댑터 없이도 Kubernetes와 통합이 되었음.

정리

1. Moby는 Docker의 오픈소스 프로젝트로, containerd를 포함한 다양한 컴포넌트로 구성됨.
2. containerd는 Docker에서 분리된 독립적인 컨테이너 런타임으로, 현재는 CNCF의 공식 프로젝트로 관리.
3. containerd는 Kubernetes와의 통합을 위해 CRI를 지원하며, 이를 통해 Kubernetes에서 직접 사용 가능.

표준 컨테이너 도구

표준 컨테이너 도구는 기본적으로 도커에서 많은 영향을 받았고, 이를 통해서 기능을 분리 및 루트리스 기반으로 기능 구현함.

1. buildah
2. podman
3. skopeo
4. kaniko

위와 같이 지원한다. 다만, 1, 4번은 엄연히 따지고 보면 CNCF의 표준 도구로 포함이 되어 있지는 않다.

하지만, 레드햇 계열 사용자는 대다수가 **buildah**기반으로 활용하고 있다. 좀 더 높은 활용성과 범용성을 원하는 경우, 특히 쿠버네티스에서 가급적이면 **kaniko**사용을 권장한다.

2025-05-18

이미지 빌드

표준 컨테이너 도구 소개

2025-05-18

이미지 빌드 도구 소개

아래 두 개의 도구는 쿠버네티스에서 사용이 가능. 다만, 쿠버네티스 기반에서는 Kaniko를 사용 하도록 권장한다.

항목	Buildah	Kaniko
개발 주체	Red Hat	Google
라이선스	Apache 2.0	Apache 2.0
CNCF 소속 여부	CNCF 프로젝트 아님	CNCF 프로젝트 아님
Docker 데몬 의존성	없음(완전한 데몬리스)	없음 (데몬리스)
루트 권한 필요 여부	루트리스 가능 (podman과 연계 시 완전 비루트 환경)	일부 루트 권한 필요 (보통 루트 컨테이너로 실행)
Kubernetes 친화성	직접 사용은 어려움, Podman이나 privileged 환경 필요	매우 높음 (Tekton, Argo 등과 잘 통합됨)
이미지 빌드 방식	명령형(buildah run, copy 등), Dockerfile, Containerfile 호환	완전한 Dockerfile, Containerfile 기반 빌드

이미지 빌드 도구 소개

앞에 내용 계속 이어서…

항목	Buildah	Kaniko
OCI 이미지 지원	완전 지원	완전 지원
기본 컨테이너 환경	RHEL/CentOS/Fedora 기반 CLI 도구	컨테이너로 제공되는 executor 이미지
속도 및 유연성	빠르고 유연함. 명령어 단위 스크립트화 가능	Dockerfile 기반으로 비교적 단순하지만 약간 느릴 수 있음
보안 및 정책 통합	SELinux/AppArmor/Rootless 환경 통합 용이	보통 컨테이너에서 루트로 실행됨 (보안 설정 필요)
대표 사용처	RHEL/Podman 환경에서의 CI, 폐쇄망 내 루트리스 빌드	Kubernetes 기반 CI/CD (Tekton, GitLab CI 등)
멀티 스테이지 빌드 지원	지원	지원

2025-05-18

빌드 도구 정리

모든 기능은 거의 동일하나, 다음과 같은 사항을 고려해서 도구를 선택한다.

상황	추천 도구
Kubernetes 기반 CI/CD 파이프라인	Kaniko, buildah 사용은 가능하나, SA 계정 구성이 필요
루트리스 보안 강화된 환경 (폐쇄망 등)	Buildah
스크립트 기반 세밀한 제어 필요 시	Buildah
단순 Dockerfile 기반 빌드 자동화	Kaniko

빌드 도구 정리

오픈소스에서 많이 사용하는 이미지 빌드 도구.

도구	CNCF 프로젝트?	용도	배포도	OCI 호환성
Buildah	(Red Hat 주도)	이미지 빌드	중간~높음	✅
Kaniko	(Google 주도)	CI/CD용 빌드	높음	✅
BuildKit	(CNCF 소속, Moby의 일부로 포함됨)	Docker 대체 빌더	중간	✅

랩

배포판에 표준 컨테이너 도구를 설치한다.

```
# dnf search container-tools.noarch
# dnf search kaniko
# podman search kaniko
# skopeo search kaniko
# podman pod ls
# podman container ls
# podman volume ls
# podman network ls
# podman images
# podman build
# buildah bud
```

런타임

표준 컨테이너 도구 소개

소개

런 타임은 보통 두 가지 형태로 지원하고 있다.

1. 고수준 컨테이너 런타임
2. 저수준 컨테이너 런타임

일반적인 저수준 런타임은 YAML이나 JSON 선언형을 직접 해석하지 않고, **로컬 Unix 소켓을 통한 gRPC 기반 API 호출로 컨테이너 생성을 수행한다.**

사용자는 직접 CLI나 선언형 파일을 다루지 않지만, **오케스트레이터(Kubernetes 등)는 CRI 프로토콜을 통해 소켓 기반으로 런타임과 통신한다.**

고수준 컨테이너 런타임

정의:

사용자 친화적인 CLI/API를 제공하며, 이미지 빌드, 레지스트리 통합, 컨테이너 실행, 네트워크 설정 등을 통합적으로 지원하는 런타임.

특징:

1. docker run, podman run, buildah처럼 직접 CLI로 컨테이너 조작 가능
2. 종종 이미지 빌드 기능도 포함
3. JSON/YAML 없이도 실행 가능

2025-05-18

고수준 컨테이너 런타임

예시:

1. Docker (Moby 기반)
2. Podman
3. Buildah (빌드 전용 도구지만 고수준 인터페이스)
4. Nerdctl (containerd + Docker 호환 CLI)

근거: Docker & Podman 공식 CLI 문서, Buildah CLI 문서

저수준 컨테이너 런타임

정의:

컨테이너 실행 자체에만 집중하는 런타임. 이미지 빌드나 CLI는 없거나 제한적이며, 보통 상위 레이어에서 제어됨.

특징:

1. 직접 컨테이너 실행은 가능하나 CLI 도구가 없거나 제한적
2. 보통 YAML/JSON 혹은 CRI (Container Runtime Interface)로 호출됨
3. Kubernetes 등 오케스트레이터가 사용하는 내부 런타임

저수준 컨테이너 런타임

예시:

1. runc (OCI 런타임 표준 구현체)
2. containerd (CRI 제공 포함)
3. CRI-O (Kubernetes 전용 경량 런타임)
4. gVisor, Kata Containers (보안 강화된 저 수준 런타임)

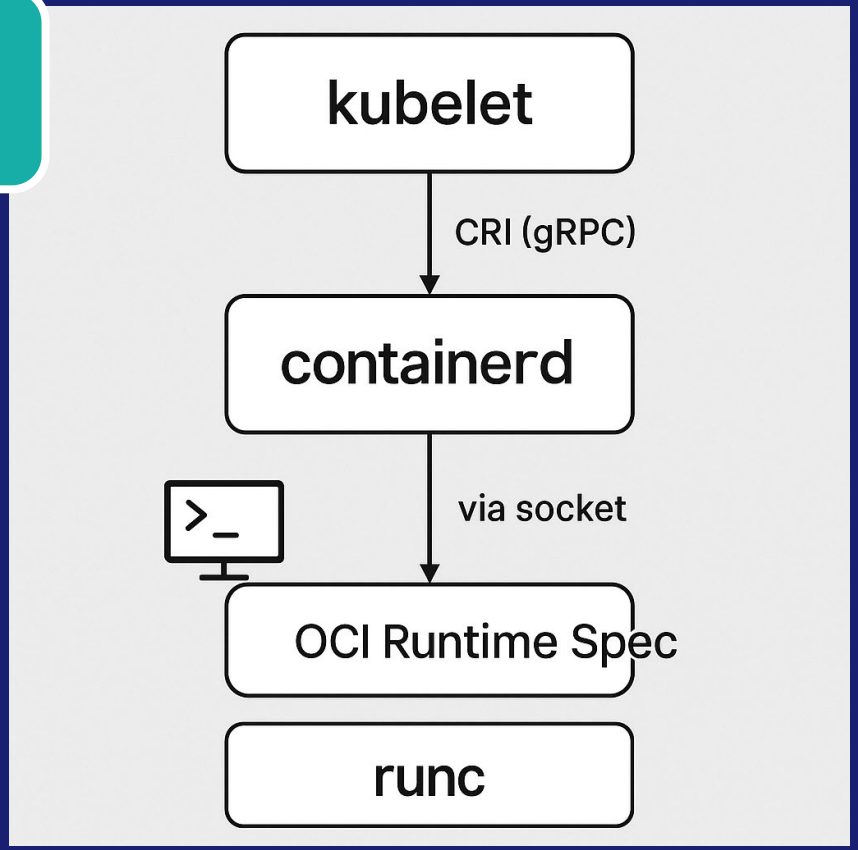
근거: OCI Runtime Spec, containerd.io

저수준

kubernetes

POD

VMI
(VirtualMachineIn
stance)

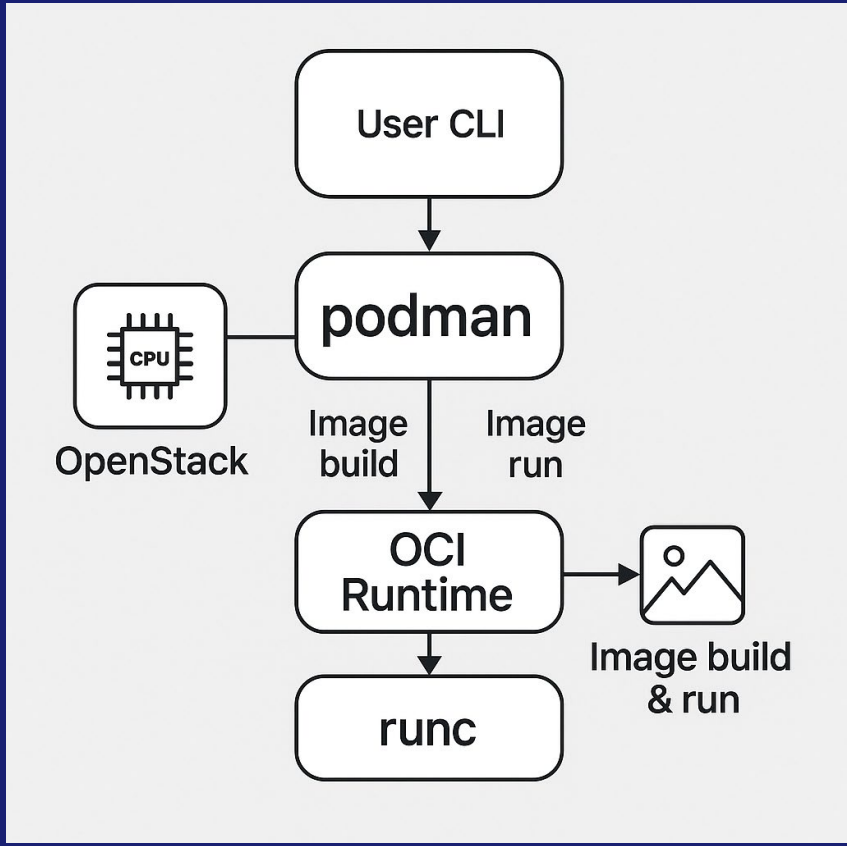


고수준

kubernetes

POD

VMI
(VirtualMachineIn
stance)



정리

런타임을 요약하면 다음과 같다.

계층	통신 방식	포맷	설명
kubelet → containerd	Unix Socket + gRPC	ProtoBuf	CRI 인터페이스
containerd → runc	fork + JSON Spec	OCI Runtime Spec	runc 실행
사용자 (CLI)	nerdctl, ctr CLI	옵션 기반	명령어 추상화 도구

랩

배포판에서 제공하는 기본적인 런타임에 대해서 확인한다.

```
# dnf search podman
# dnf search containerd
# dnf search docker
# dnf provides crun
# dnf provides runc
# ls -l /usr/share/containers/
```

표준 가상화 도구 소개

이론+랩

2025-05-18

하이퍼바이저 소개

가상화 기술은 하드웨어 자원을 추상화하여 논리적인 시스템을 구현하는 핵심 기반 기술이며, 이러한 가상화를 실현하는 도구들은 크게 **하이퍼바이저 계층**, **가상머신 관리 계층**, 그리고 **클라우드 통합 계층**으로 구분할 수 있다.

먼저, 가장 하위 계층에 해당하는 것은 **저수준 하이퍼바이저**이다. 이 계층에서는 **KVM(Kernel-based Virtual Machine)**과 **Xen**이 대표적인 도구로 사용된다. KVM은 리눅스 커널 자체에 통합되어 있으며, **QEMU**와 함께 가상머신을 생성하고 실행할 수 있다.

Xen은 초기 클라우드 환경에서 많이 사용되었으며, 하이퍼바이저와 게스트 OS 간의 분리된 구조로 보안성을 높인 것이 특징이다.

최근에는 **AWS의 Firecracker**와 같이 경량화된 마이크로VM을 운영할 수 있는 기술도 등장했는데, 이는 서버리스 환경이나 고속 부팅을 요구하는 워크로드에서 사용된다.

2025-05-18

하이퍼바이저 소개

이러한 하이퍼바이저를 기반으로 **가상머신을 생성, 제어, 관리**하는 API 및 CLI 계층이 존재한다. 여기에는 **libvirt**가 핵심 역할을 한다. libvirt는 KVM, QEMU, Xen과 같은 다양한 하이퍼바이저와 연동되는 통합 API이며, 이를 바탕으로 **virt-manager**와 같은 GUI 툴도 개발되어 사용자 친화적인 VM 관리를 가능하게 한다.

또한 **Buildah나 Podman**이 리눅스 컨테이너에 가까운 루트리스 환경을 제공하는 데에 비해, 이 계층은 **정통적인 VM 기반 워크로드**를 주로 다룬다.

HCI 환경

더 상위 계층에서는 **프라이빗 클라우드 또는 하이퍼컨버지드 인프라(HCI)** 환경에서 다수의 가상머신을 중앙에서 통합 관리할 수 있는 플랫폼들이 활용된다.

오픈소스 계열에서는, Proxmox VE, 오픈소스 기반으로 KVM과 LXC를 통합한 HCI 플랫폼.

oVirt는 Red Hat 계열의 기업 환경에서 사용되는 가상화 통합 관리 도구로, **RHV(Red Hat Virtualization)**의 업스트림이다. 현재 RHV는 더 이상 판매하지는 않으며, oVirt만 오픈소스 기반으로 제공하고 있다.

SUSE 계열에서는 Harvester가 등장하며, Kubernetes와 KVM 기반 가상화를 결합하여 클라우드 네이티브 기반의 VM 운영을 가능하게 하고 있다.

2025-05-18

쿠버네티스 가상화

최근에는 Kubernetes 환경과 VM을 통합하려는 시도도 활발하다. 대표적으로 KubeVirt는 Kubernetes 위에서 가상머신을 Pod처럼 실행할 수 있도록 하여, 컨테이너와 VM이 하나의 클러스터 내에서 공존하도록 만든다.

이러한 접근은 **전통적인 VM 워크로드와 클라우드 네이티브 워크로드를 동시에 수용**하고자 하는 조직에서 특히 주목받고 있다. Virtlet 역시 비슷한 목표를 가지고 개발되었으나, 현재는 KubeVirt가 사실상의 표준으로 자리 잡고 있다.

결론적으로, 현대의 가상화 환경은 단일 하이퍼바이저를 넘어 **API 기반의 가상머신 제어**, **클러스터 수준의 통합 관리**, 그리고 **컨테이너 플랫폼과의 융합**까지 고려해야 하는 복합 구조로 진화하고 있다.

각 도구는 이러한 계층적 구조 안에서 고유한 역할을 수행하며, 사용자의 요구에 따라 적절히 선택되고 조합되어야 한다.

2025-05-18

AWS Nitro System

하이퍼바이저: 경량화된 KVM 기반의 Nitro Hypervisor. 메모리와 CPU 할당을 관리하며, 대부분의 워크로드에서 베어메탈과 구분되지 않는 성능을 제공함.

하드웨어 오프로드: 전통적인 하이퍼바이저 기능을 Nitro 카드와 보안 칩으로 분리하여, 네트워킹, 스토리지, 보안 기능을 전용 하드웨어에서 처리함으로써 성능과 보안을 향상시킴.

보안: Nitro 보안 칩을 통해 하드웨어 수준의 신뢰 기반을 제공하며, 관리자 접근을 제한하여 보안성을 강화함 .

베어메탈 지원: Nitro Hypervisor 없이도 인스턴스를 실행할 수 있는 베어메탈 인스턴스를 제공하여, 특정 하드웨어 기능에 직접 접근이 필요한 워크로드에 적합함.

AWS Nitro System

Nitro System은 **EC2 인스턴스의 기반 인프라**로, 다음과 같은 주요 구성요소로 이루어진다.

Nitro Cards:

- 네트워킹, 스토리지, 보안, 모니터링 기능을 처리하는 전용 하드웨어 카드이다.
- 각각의 카드는 특화된 기능을 담당하며, 가상머신에 부하를 주지 않는다.
- 예: Nitro Security Chip, Nitro Card for VPC, EBS 등.

AWS Nitro System

Nitro Hypervisor:

- 최소한의 기능만을 갖는 경량화된 하이퍼바이저.
- Linux KVM 기반이지만, EC2 인스턴스에서 거의 native 수준의 성능을 제공.
- VM 내 커널에 접근하지 않음 (보안성 강화).

Nitro Security Chip:

- 하드웨어 루트 오브 트러스트(HWRoT)를 제공하며, 각 인스턴스를 부팅 시점부터 검증하고 보호한다.
- 각 인스턴스의 보안 상태가 항상 암호화로 검증되도록 보장한다.

OpenStack

하이퍼바이저: KVM, Xen, VMware ESXi 등 다양한 하이퍼바이저를 지원하며, Nova 컴포넌트를 통해 가상 머신의 생성과 관리를 담당함.

구성 요소: Nova(컴퓨트), Neutron(네트워킹), Cinder(블록 스토리지), Glance(이미지 서비스), Keystone(인증), Horizon(대시보드) 등 모듈화된 구조로 구성됨.

확장성: 대규모 데이터 센터에서 수천 개의 노드와 가상 머신을 관리할 수 있는 확장성을 제공함.

유연성: 다양한 하드웨어 및 네트워크 구성에 대한 유연한 지원을 통해, 맞춤형 클라우드 인프라 구축이 가능함.

Harvester

하이퍼바이저: KVM 기반으로, Kubernetes와 KubeVirt를 통합하여 가상 머신과 컨테이너 워크로드를 동시에 관리할 수 있음 .

통합 관리: Rancher와의 통합을 통해 단일 인터페이스에서 가상 머신과 컨테이너를 관리할 수 있으며, 멀티 테넌시와 RBAC를 지원함 .

스토리지: Longhorn을 기반으로 한 분산 블록 스토리지를 제공하여, 고가용성과 데이터 복제를 지원함.

배포 용이성: 베어메탈 서버에 직접 설치하여 간편하게 HCI 환경을 구축할 수 있으며, 엣지 컴퓨팅 및 중소 규모의 클러스터에 적합함.

OpenShift Virtualization

OpenShift Virtualization은 OpenShift Console에 완전히 통합된 관리 인터페이스를 제공하며, Web UI를 통해 VM 생성, 모니터링, 삭제뿐 아니라 Persistent Volume, 네트워크, 가상 GPU 등 자원을 Kubernetes 방식으로 관리할 수 있다.

KubeVirt 기반으로 동작하며, 컨테이너 중심의 CI/CD 환경 속에서도 레거시 시스템을 VM으로 유지해야 하는 조직에 현실적인 해법을 제공한다.

컨테이너와 VM을 동일 파이프라인에서 배포하거나, VM 워크로드를 점진적으로 컨테이너화하는 과도기 전략에도 적합하다.

다만 OpenShift 구독이 필요하고, 기술 복잡성과 자원 요구가 높으며, 최소 3노드 이상의 클러스터와 성능 튜닝, 정책 설정을 위한 OpenShift 생태계에 대한 이해가 요구된다.

2025-05-18

HCI 오케스트레이션

항목	Harvester	Rancher	OpenShift(OV)
기반 기술	KVM + Longhorn	Kubernetes 클러스터 관리자	OpenShift + CRI-O + OpenShift Virtualization
주요 기능	VM 생성/운영, HCI 인프라	K8s 클러스터 배포, 멀티 테넌시, RBAC	PaaS, GitOps, CI/CD + VM 통합 실행
컨테이너 지원	K8s 위에서 직접 실행은 아님	완전 지원	완전 지원
가상머신 지원	네이티브 VM 지원 (KVM)	직접 지원 아님 (Harvester 통해 가능)	Red Hat 기반 VM 실행
통합 관리	Rancher와 완전 통합	Harvester 및 K8s 통합	OpenShift Console 기반 통합 관리
대상 사용자	시스템 관리자, 엣지 관리자	인프라 관리자, DevOps 관리자	엔터프라이즈 인프라팀, 보안 중심 워크로드 운영자

2025-05-18

HCI 오케스트레이션

항목	Harvester	Rancher	OpenShift(OV)
설치 난이도	낮음 (ISO 설치)	중간	높음
리소스 요구	중간 (베어메탈 필요)	낮음 ~ 중간	높음
적합한 사용 사례	엣지, 경량 HCI, 단일 VM 관리	다중 클러스터 통합 운영	금융, 공공, 통신 등 보안 필수 VM+앱 운영 환경
오픈소스 여부	Apache 2.0	Apache 2.0	오픈소스 기반이나 Red Hat 구독 필수 (상용 모델)

2025-05-18

가상머신 오케스트레이션

항목	Harvester	OpenStack	Proxmox VE
개발 주체	SUSE / Rancher	OpenInfra Foundation	Proxmox Server Solutions GmbH
오픈소스 여부	완전한 오픈소스	오픈소스	오픈소스 (유료 지원 옵션)
기반 하이퍼바이저	KVM (libvirt)	KVM / Xen / 기타	KVM + LXC
컨테이너 통합	K3s 내장, Rancher 통합	별도 Kubernetes 연동 필요	Kubernetes 통합은 간접적
스토리지 구성	Longhorn 기반 내장 스토리지	Ceph, LVM, NFS 등 유연한 구성	ZFS, LVM, Ceph 연동 가능
네트워크 구성	Flannel + Multus + VLAN	Neutron (SDN, VLAN, VXLAN 등)	Linux bridge, OVS, VLAN

2025-05-18

가상머신 오케스트레이션

항목	Harvester	OpenStack	Proxmox VE
관리 인터페이스	웹 UI (Rancher 기반), API	Horizon 웹 UI, OpenStack CLI, API	웹 UI, CLI (qm, pct), API
클러스터 확장성	수평 확장 가능 (노드 추가 용이)	대규모 수직/수평 확장 가능	클러스터 구성 가능 (중소형 중심)
Kubernetes 통합	완전 통합 (Rancher/Fleet)	기본은 독립 구조, 연동은 가능	통합 아님, 별도 연동 필요
주요 사용 환경	엣지, 클라우드 네이티브 HCI	대기업, 통신사, 공공기관	중소기업, 교육기관, 개인 서버
라이선스 모델	Apache 2.0	Apache 2.0	AGPL v3

2025-05-18

QEMU: 에뮬레이터에서 가상화 핵심 엔진으로

QEMU (Quick Emulator)는 2003년 Fabrice Bellard에 의해 개발되었으며, 초기에는 **다양한 CPU 아키텍처 (x86, ARM, MIPS 등)를 에뮬레이션하는 목적으로 만들어진 범용 에뮬레이터**였다.

하지만 시간이 지나며, 하드웨어 가속 기반의 가상화(KVM)와 연계되면서 **풀 가상화 하이퍼바이저**로서도 자리잡게 된다. QEMU의 가장 큰 특징은 **유연성**이다.

순수한 에뮬레이션(속도 느림)뿐 아니라, KVM 모듈이 탑재된 커널에서 실행하면 **하드웨어 지원을 통해 네이티브 성능**에 근접한 가상화를 구현할 수 있다.

좀 더 오픈소스…

KVM: 리눅스를 하이퍼바이저로 만든 전환점

KVM (Kernel-based Virtual Machine)은 2006년 Avi Kivity가 개발하였고, 2007년에 **리눅스 커널 메인라인에 정식 통합**되었다. KVM은 리눅스를 하나의 하이퍼바이저로 전환시켜주는 모듈이며, x86 CPU의 하드웨어 가상화 지원(Intel VT-x, AMD-V)을 활용해 고속의 가상 머신 실행을 가능케 한다.

KVM은 자체적으로는 VM을 관리하거나 실행하지 않는다. 대신, QEMU 같은 상위 계층이 이를 호출해 **하드웨어 수준 가상화를 위임**하는 구조다. 즉, 현대의 대부분 리눅스 가상화는 **"QEMU + KVM"**을 기본 조합으로 사용한다.

libvirt: 통합 관리 API, 하이퍼바이저의 관리자

libvirt는 2005년 Red Hat 주도로 개발된 가상화 추상화 API이다. **QEMU/KVM**, **Xen**, **LXC**, **bhyve** 등 다양한 하이퍼바이저를 공통 인터페이스로 통합할 수 있도록 설계되었다.

libvirt는 QEMU 명령어를 직접 다루는 복잡성을 줄이고, **virsh**, **virt-manager**, **virt-install** 등의 CLI/GUI 툴을 통해 가상 머신, 스냅샷, 네트워크, 볼륨 관리를 직관적으로 수행할 수 있게 한다.

즉, libvirt는 QEMU/KVM을 좀 더 운영자 친화적으로 만드는 추상화 계층이라고 할 수 있다.

왜? QEMU/KVM는 느린가?

약간 주제는 다르지만, 많은 오픈소스 고객들은 다음과 같은 부분을 엔지니어나 혹은 세일즈에게 도전을 한다.

1. 디스크 기능은 왜 부족하고 느린가?
2. 메모리 기능은 왜 항상 수동적인가?
3. 네트워크 성능은 왜 항상 부족하고 느린가?

이 부분에 대해서 다음과 같이 아키텍처 이유로 설명이 가능하다.

왜? QEMU/KVM는 느린가?

QEMU/KVM 기반의 아키텍처는 다음과 같은 구조를 가지고 있다.

요소	VMware ESXi	QEMU/KVM
하이퍼바이저 구조	Type-1 (네이티브, 전용 OS 위)	Type-2 또는 내장형 (Linux 커널 내 KVM 모듈)
가상머신 런타임	VMkernel + vSphere / vCenter 관리	KVM 모듈 + QEMU (에뮬레이터 레벨)
디스크 처리	VMFS 또는 vSAN + Paravirtual SCSI (PVSCSI)	raw/qcow2 이미지 + virtio-blk/scsi
네트워크 처리	VMXNET3 paravirtual NIC + DVS/VSS	virtio-net, vhost-net, TAP, bridge 등

2025-05-18

디스크 (블록 장치) 아키텍처 차이(VMware)

- **디스크 포맷**: VMDK + VMFS 또는 vSAN을 통해 공유 스토리지 구성
- **드라이버 모델**:
  - 기본 IDE/SCSI
  - 고성능: **Paravirtual SCSI(PVSCSI)** – ESXi 커널에 최적화됨
- **I/O 경로**: VM → PVSCSI 드라이버 → VMkernel → 스토리지 계층(FC/iSCSI/NFS 등)

특징:

1. I/O 처리 시 오버헤드가 낮고,
2. vSphere Storage I/O Control(SIOC)로 VM간 IOPS 제어 가능
3. 고성능 환경에 적합 (특히 SSD, NVMe 환경)

2025-05-18

디스크 (블록 장치) 아키텍처 차이(QEMU/KVM)

- **디스크 포맷:** raw 또는 qcow2 이미지 사용 (qcow2는 스냅샷, 압축 기능 포함)
- **드라이버 모델:**
  - 기본: IDE, SATA
  - 고성능: **`virtio-blk`, `virtio-scsi`**
- **I/O 경로:** VM(QEMU) → virtio → vhost-user ↔ ioThread → Linux 커널 I/O stack → 물리 디스크

네트워크 아키텍처 차이(VMware)

- **네트워크 드라이버**: VMXNET3: VMware 전용 paravirtual NIC (드라이버 설치 필요)
- **가상 스위치 구성**: Standard vSwitch (vSS), Distributed vSwitch (vDS)
- **I/O 경로**: VM → VMXNET3 → vSwitch → 물리 NIC (uplink)

특징:

1. low-latency, high-throughput 환경에서 우수
2. VLAN, QoS, NSX 같은 기능과 통합
3. 관리/보안/모니터링 도구 연동이 뛰어남

2025-05-18

네트워크 아키텍처 차이(QEMU/KVM)

- **네트워크 드라이버**: virtio-net (기본 paravirtual), e1000, rtl8139
- **I/O 가상화 방식**: user-mode, TAP, bridge, macvtap, SR-IOV, vhost-net
- **I/O 경로**: VM(QEMU) → virtio-net → vhost-net (kernel bypass) → host NIC

특징:

1. 성능은 virtio + vhost-net 또는 DPDK 적용 시 우수
2. SR-IOV 사용 가능, 리눅스 친화적
3. 보안 및 트래픽 제어는 별도 방화벽/NAT 설정 필요

VMware, QEMU/KVM

위의 블록장치 아키텍처를 정리하면 다음과 같다.

구성 요소	VMware	QEMU/KVM
디스크 구조	PVSCSI + VMkernel	virtio + ioThread + Linux block layer
네트워크 구조	VMXNET3 + vSwitch	virtio-net + vhost-net or SR-IOV
성능 기본값	기본값도 고성능	튜닝 없이는 성능 저하 가능
아키텍처 복잡도	직관적, 통합 관리	유연하지만 구성 요소 다양, 튜닝 필요

guestfish-tools (libguestfs)

`guestfish`는 `libguestfs`라는 프로젝트의 일부이며, 가상 머신의 디스크 이미지를 직접 마운트하거나 파일을 수정하는 등의 오프라인 디스크 편집 기능을 제공한다.

특히, VM이 부팅되지 않아도 루트 파일시스템을 열어 설정을 변경하거나 로그를 조사할 수 있기 때문에, 운영 중인 VM에 문제가 발생했을 때 복구 및 포렌식 도구로 매우 유용하다.

`guestmount`, `virt-edit`, `virt-copy-in` 등의 도구도 같은 프로젝트에 속하며, 특히 Red Hat 계열의 클라우드 이미지 생성/검사 자동화에 널리 사용된다.

2025-05-18

가상화 기여도

가상화 기능에 대해서 다음과 같이 각 배포판 회사들은 기여하고 있다.

배포판	가상화 기술 기여도 / 특성
Red Hat	QEMU/KVM/libvirt/libguestfs의 주요 커미터 다수 보유. virt-manager, virt-install 도구의 원 개발자이자 유지보수자.
SUSE	Xen과 KVM 모두 오래 지원, libvirt 패치와 VM 변환 도구(zypper 기반)에 기여. Cloud용 이미지 제작 도구 통합.
Ubuntu	KVM을 기본 가상화 백엔드로 사용, 클라우드 이미지와 cloud-init 통합에 강점. libvirt는 보통 안정 패키징 수준 유지.
Debian	QEMU/libvirt 등 오픈소스 전반에 기여하지만, 개발보다는 안정화·패키징 위주. Upstream 기여율은 낮은 편.

랩

강사의 가이드에 따라서 도구를 설치하면서 랩을 진행한다.

```
# dnf grouplist
> Virtualization Hosts
# dnf group install "Virtualization Hosts"
# dnf search guest-tools
# dnf install guest-tools -y
# dnf install diskimage-builder
# systemctl enable --now libvirtd
# virsh
# virt-builder
# virt-sparsify
# virt-sysprep
```

API는 왜 중요한가? 시스템 엔지니어 입장에서 API

이론+랩

2025-05-18

시스템 엔지니어의 관점에서 본 API의 본질

과거의 시스템 엔지니어링은 물리적 서버 배치, OS 설치, 서비스 설정, 보안 정책 구성 등 "손을 움직이는 일"이 중심이었다.

하지만 지금의 인프라 환경은 명확히 달라졌다. API(Application Programming Interface)는 이 변화의 핵심이며, 시스템 엔지니어에게 있어 더 이상 "프로그래머의 영역"이 아니다.

오히려 API는 오늘날 인프라 자체의 언어가 되었다.

API는 현대 시스템의 "제어판"이다

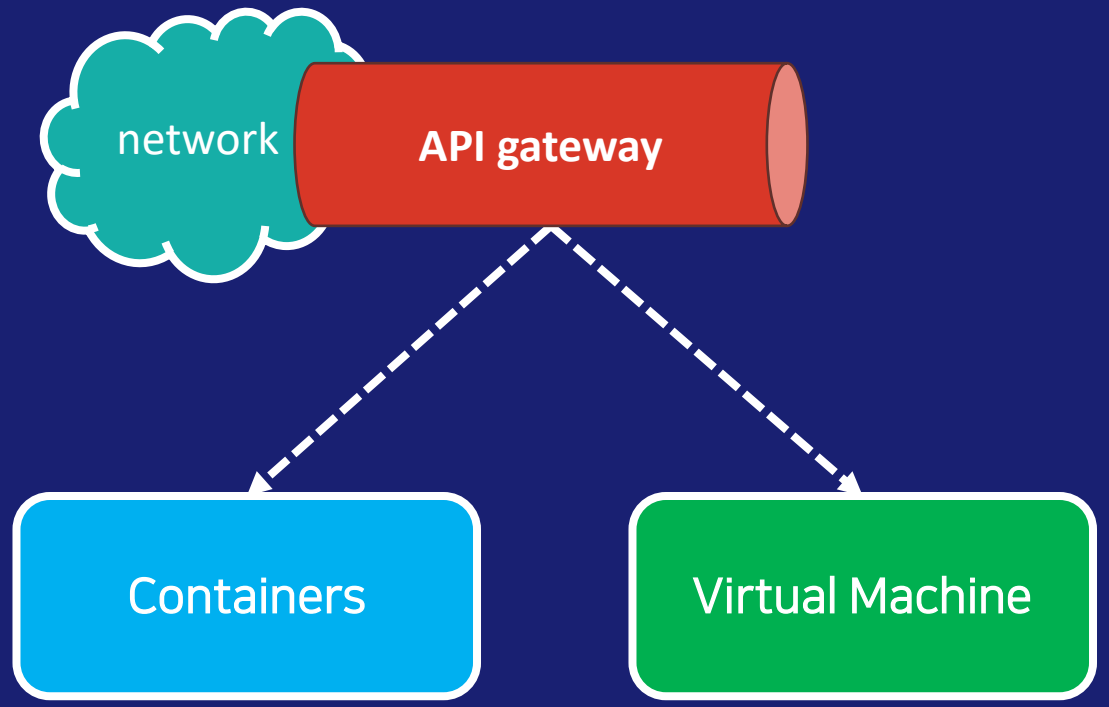
API는 단지 프로그래밍 인터페이스가 아니다.

오늘날의 **클라우드 플랫폼(OpenStack, AWS, GCP)**, **컨테이너 플랫폼(Kubernetes)**, **네트워크 컨트롤러(SDN)**, 스토리지, 심지어 하드웨어 `BMC(iDRAC, iLO)`까지 **모든 구성 요소는 API를 통해 제어**된다.

시스템 엔지니어가 인프라를 관리하고 자동화하려면, GUI나 CLI만으로는 한계가 있으며, **API를 통한 선언형 구성, 자동화, 통합 제어**가 요구된다. 이는 단순 편의성의 문제가 아니라, **속도와 일관성, 보안과 정책 적용의 문제**다.

2025-05-18

API는 현대 시스템의 "제어판"이다



API는 자동화와 GitOps의 전제조건이다

현대 인프라 운영의 핵심 전략은 `IaC(Infrastructure as Code)`, `GitOps`, `DevOps`다. 이 모든 전략은 **API가 존재할 때만 의미를 가진다.**

예를 들어 `Terraform`, `Ansible`, `Pulumi`, `Salt`, `ArgoCD`와 같은 도구들이 인프라를 코드로 정의하고 운영할 수 있는 이유는, **클라우드 자원, VM, 컨테이너, 네트워크 장비 등이 전부 API로 구성되어 있기 때문**이다.

즉, API는 **시스템 엔지니어가 자동화를 구현하는 유일한 접점**이다. 만약 관리 대상 시스템에 API가 없다면, 이는 자동화에 있어 "불통"과 같다.

HashiCorp State of Cloud Strategy 2023 보고서에 따르면, "전통적 수작업 운영 방식을 유지하는 조직은 평균 40% 이상의 운영 오류를 경험하고 있으며, API 기반 IaC 및 DevOps 자동화에 성공한 조직은 배포 속도와 보안 적용률이 평균 3배 이상 향상되었다"고 보고되었다.

2025-05-18

API는 '보안'의 새로운 경계다

시스템 엔지니어는 과거 방화벽, 계정 관리, OS 업데이트에 집중했다면, 지금은 **API 접근 통제**, **API 토큰 및 인증 방식**, **API Rate Limit**, RBAC(Role-Based Access Control)이 **실제 보안 정책의 중심**이 되었다.

즉, 시스템 엔지니어는 더 이상 단순히 OS 사용자 권한만 설정하는 것이 아니라, API를 통해 자원을 생성하고 삭제하는 사용자에게 **세밀한 접근 권한을 부여한다.**

API Gateway, **인증 서버(OAuth, OIDC)**, **시크릿 저장소(Vault, SealedSecrets)** 등과도 **연계된 보안 정책**을 설계해야 한다.

API를 이해하는 시스템 엔지니어는 '현대형 관리자'다

API를 이해하지 못하는 시스템 엔지니어는 점차 **클라우드** 시대의 인프라 관리자 역할에서 소외되고 있다.

반면, REST API 구조, 인증 토큰 관리, Swagger 문서 읽기, HTTP 통신 흐름에 익숙한 엔지니어는 DevOps, 클라우드 아키텍처, 자동화 관리자, GitOps 운영자로서 중요한 위치를 차지한다.

이는 단순히 프로그래밍 언어를 익히는 것과는 다르다. API를 통해 시스템이 **"무엇을 할 수 있고, 어떤 방식으로 통제되며, 어떤 정책이 적용되는가"**를 이해하는 것은 곧 **시스템 그 자체를 이해하는 것**과 같다.

대표적인 오픈소스 기업

시스템 엔지니어링의 중심이 점점 **수동 구성에서 API 중심의 선언형 자동화**로 이동하면서, Red Hat과 SUSE는 각기 다른 철학과 제품군을 통해 이 변화를 뒷받침하고 있다.

두 회사 모두 커널 기반 가상화(KVM), 시스템 관리 도구, 자동화 API에서 핵심 역할을 해왔지만, API 도입 방식과 생태계 통합 전략은 뚜렷한 차이를 보인다.

Red Hat vs SUSE의 시각과 전략 비교

시스템 엔지니어링의 중심이 점점 **수동 구성에서 API 중심의 선언형 자동화**로 이동하면서, Red Hat과 SUSE는 각기 다른 철학과 제품군을 통해 이 변화를 뒷받침하고 있다.

두 회사 모두 커널 기반 가상화(KVM), 시스템 관리 도구, 자동화 API에서 핵심 역할을 해왔지만, API 도입 방식과 생태계 통합 전략은 뚜렷한 차이를 보인다.

Red Hat: API는 표준화된 인프라 운영의 기반

Red Hat은 API를 인프라 자동화의 핵심 인터페이스로 보고, 이를 **플랫폼화(PaaS + IaC)**하는 데 주력해왔다.

`OpenStack`, `RHEL`, `OpenShift`, `Satellite` 등의 제품군은 전부 `RESTful API`를 통한 제어를 기본으로 하며, `virt-install`, `subscription-manager`, `cockpit`, `cloud-init` 등의 도구들도 모두 API 또는 D-Bus 기반 메시지 버스를 통해 작동한다.

특히 OpenShift에서는 모든 작업을 oc CLI 또는 GitOps 방식으로 선언하고, 실제로는 백엔드의 Kubernetes API를 통해 완전히 선언적, 자동화 가능한 구조를 지향한다.

2025-05-18

Red Hat: API는 표준화된 인프라 운영의 기반

즉, Red Hat은 **"API = 시스템의 추상화된 제어 권한"**이라는 철학을 실현하고 있고, 시스템 엔지니어가 API를 통해 보안, 자원 할당, CI/CD 연동까지 모두 통제할 수 있는 설계를 한다.

예시: OpenShift Virtualization의 kubevirt.io/v1 API 그룹 RHEL의 System Role을 Ansible 기반으로 통합한 방식

2025-05-18

SUSE: API는 오픈 생태계와 통합의 도구

반면 SUSE는 가시성과 상호운용성을 우선시하며, API를 타 도구와 통합 가능한 범용 인터페이스로 제공하는 전략에 초점을 맞춘다.

예를 들어 Harvester는 KVM 기반의 HCI 플랫폼이지만, Rancher를 통해 Kubernetes API와 완전히 통합되어, VM도 Kubernetes 리소스처럼 다룰 수 있고, Longhorn도 API 기반으로 외부에서 관리된다.

SUSE: API는 오픈 생태계와 통합의 도구

또한 YaST, AutoYaST, SUSE Manager, SaltStack 등 수세 기반의 구성 도구들도 모두 API 또는 모듈화된 인터페이스를 제공하여, 시스템 엔지니어가 다른 자동화 도구와 쉽게 연결할 수 있게 한다.

- 예시: Harvester는 Rancher 내에서 VM을 생성하는 UI 요청을 Kubernetes API로 변환 SUSE Manager는 REST API + SaltStack으로 서버 구성 전역 자동화 가능

SUSE의 전략은 Red Hat처럼 모든 계층을 API로 "추상화"하는 방향보다는, 사용자에게 친숙하고 투명한 방식으로 API를 노출하고, 선택 가능한 도구와의 유연한 연동성을 중시한다.

Canonical이 바라보는 API의 철학과 전략

반면 SUSE는 가시성과 상호운용성을 우선시하며, API를 타 도구와 통합 가능한 범용 인터페이스로 제공하는 전략에 초점을 맞춘다.

예를 들어 `Harvester`는 KVM 기반의 HCI 플랫폼이지만, Rancher를 통해 Kubernetes API와 완전히 통합되어, VM도 Kubernetes 리소스처럼 다룰 수 있고, Longhorn도 API 기반으로 외부에서 관리된다.

Canonical이 바라보는 API의 철학과 전략

또한, `YaST, AutoYaST, SUSE Manager, SaltStack` 등 수세 기반의 구성 도구들도 모두 API 또는 모듈화 된 인터페이스를 제공하여, 시스템 엔지니어가 다른 자동화 도구와 쉽게 연결할 수 있게 한다.

- 예시: `Harvester`는 `Rancher` 내에서 VM을 생성하는 UI 요청을 Kubernetes API로 변환 **SUSE Manager**는 `REST API + SaltStack`으로 서버 구성 전역 자동화 가능

SUSE의 전략은 Red Hat처럼 모든 계층을 API로 **"추상화"**하는 방향보다는, 사용자에게 친숙하고 투명한 방식으로 API를 노출하고, 선택 가능한 도구와의 유연한 연동성을 중시한다.

2025-05-18

핵심 비교 요약

정리하면 다음과 같다.

구분	Red Hat	SUSE
API 철학	선언형 제어, 완전한 자동화 중심	상호운용성과 오픈 인터페이스 중심
API 활용 사례	RHEL System Role, OpenShift REST API	Harvester K8s API, SaltStack API
자동화 구조	GitOps + Ansible 통합 자동화	SaltStack 기반 이벤트 중심 자동화
표준화 지향성	업스트림 중심 표준화 및 산업 확산	실용적 오픈소스 통합 및 커뮤니티 협력 강조
도구 통합성	tightly coupled 플랫폼 (OpenShift 등)	loosely coupled 모듈형 시스템 (Harvester 등)

랩

Hoppscotch 소개

Hoppscotch는 오픈소스 기반의 웹 API 테스트 도구이다.

Postman과 유사한 기능을 제공하면서도, 브라우저 기반으로 실행되기 때문에 별도의 데스크탑 설치 없이 누구나 사용할 수 있다. 컨테이너로 실행할 수 있어 Podman을 기반으로 손쉽게 시연 환경을 구축할 수 있다.

간단하게 Podman기반으로 실행한다.

```
# podman run -d --name hoppscotch -p 3000:3000 hoppscotch/hoppscotch:latest
# firefox http://localhost:3000
```

랩

테스트로 사용할 API는 아래 주소에서 사용한다. 여기서는 GET으로 API요청한다.

- **주소: `https://jsonplaceholder.typicode.com/users`**
- **메소드: `get`**

"send"버튼을 누르면, Hoppscotch에서 다음과 같이 정보를 가져온다.

```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "email": "Sincere@april.biz"
  },
   ...
]
```

랩

이번에는 POST를 통해서 요청한다. Hoppscotch에서 아래와 같이 작성한다.

- 주소: `https://jsonplaceholder.typicode.com/posts`
- 메소드: `post`
- body: JSON

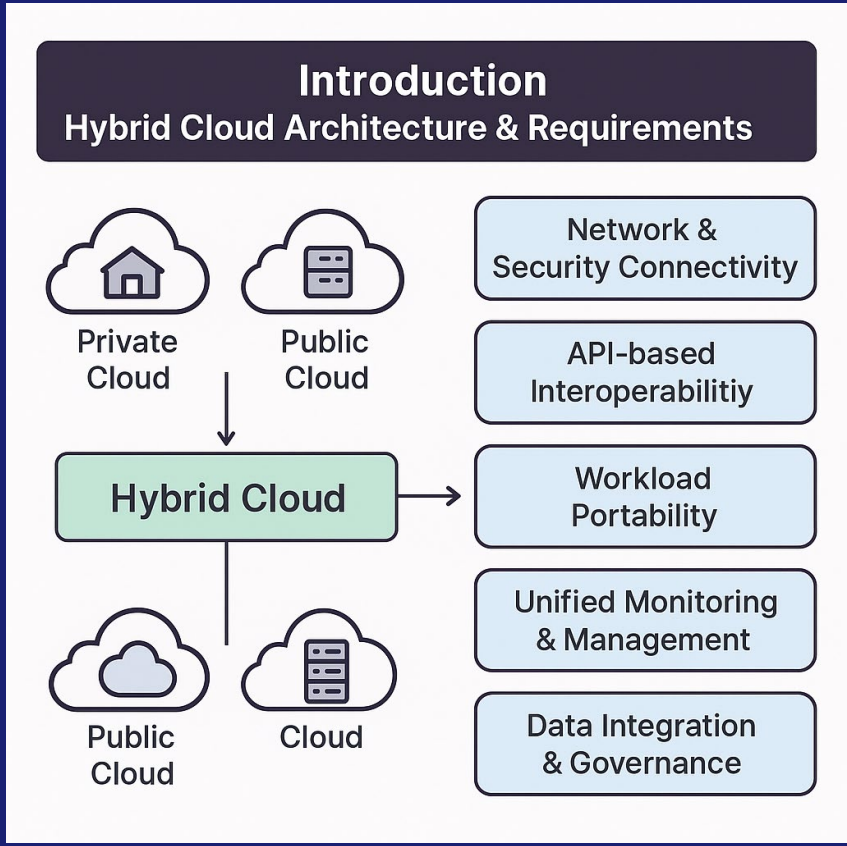
```
{
  "title": "API 데모",
  "body": "이것은 테스트입니다",
  "userId": 1
}
```

하이브리드 클라우드 아키텍처 소개 및 충족조건

이론+랩

2025-05-18

하이브리드 클라우드 전략



2025-05-18

하이브리드 클라우드 아키텍처 소개 및 충족 조건

하이브리드 클라우드 아키텍처는 **퍼블릭 클라우드와 프라이빗 클라우드(또는 온프레미스 인프라)**를 연결하여, **통합된 IT 서비스 환경을 제공하는 구조**를 의미한다.

이 아키텍처는 **두 환경 간의 워크로드 이동**, **데이터 연동**, **보안 정책 일관성**을 유지하면서도, **비용 효율성과 유연성**을 동시에 추구한다.

하이브리드 클라우드는 **단순히 이기종 인프라**를 **나란히 놓는 구조가 아니라, 서로 다른 환경이 단일 운영 체계처럼 작동하도록 설계된 구조**를 전제로 한다.

이로 인해 아키텍처에는 반드시 다음과 같은 **충족 조건**이 요구된다.

2025-05-18

조건

1. **네트워크 및 보안 연동성이 충족되어야 한다.**

하이브리드 클라우드 환경은 두 개 이상의 인프라를 연결하므로, VPN, 전용 회선, SD-WAN **등 안정적이고 보안이 강화된 네트워크 경로**가 확보되어야 하며, 동시에 **아이덴티티 및 액세스 관리 체계(IAM)** 또한 서로 연동되어야 한다.

2. **API 기반의 연계성과 표준화된 인터페이스가 확보되어야 한다.**

퍼블릭과 프라이빗 환경에서 워크로드를 이동하거나 동일한 방식으로 제어하려면, REST API, Terraform, Salt, Ansible **등으로 양쪽 환경을 공통된 방식으로 자동화**할 수 있어야 한다. 이는 DevOps 관점에서도 중요한 기준이 된다.

2025-05-18

조건

3. 워크로드 이식성(Portability)이 확보되어야 한다.

가상머신(VM)과 컨테이너 기반 워크로드는 **하이퍼바이저 호환성**, **이미지 포맷 통일**, Kubernetes 기반의 **컨테이너 오케스트레이션 구조**를 통해 이동 가능해야 한다. 이식성은 곧 **탄력성(Flexibility)**과 직결된다.

조건

4. **넷째, 통합된 모니터링과 정책 기반 관리체계가 필요하다.**

하이브리드 환경은 물리적, 가상화, 클라우드 기반 인프라가 혼재하기 때문에, **단일 관점에서의 가시성(Observability)**과 **정책 기반의 리소스 제어, 보안 제어(RBAC, 정책 컴플라이언스 등)**가 필수로 요구된다.
예: Rancher, Red Hat Advanced Cluster Management, Azure Arc 등

5. **다섯째, 데이터 연계성과 거버넌스 체계가 구현되어야 한다.**

데이터는 위치에 따라 법적, 기술적 제약을 받는다. **데이터 이동, 복제, 백업 정책, 분산 스토리지 연동, 데이터 암호화와 주권 확보**는 하이브리드 클라우드 아키텍처에서 반드시 고려되어야 할 조건이다.

만약, 하이브리드 클라우드 구현을 해야 한다면?

이론+랩

만약, 하이브리드 클라우드 구현을 해야 한다면?

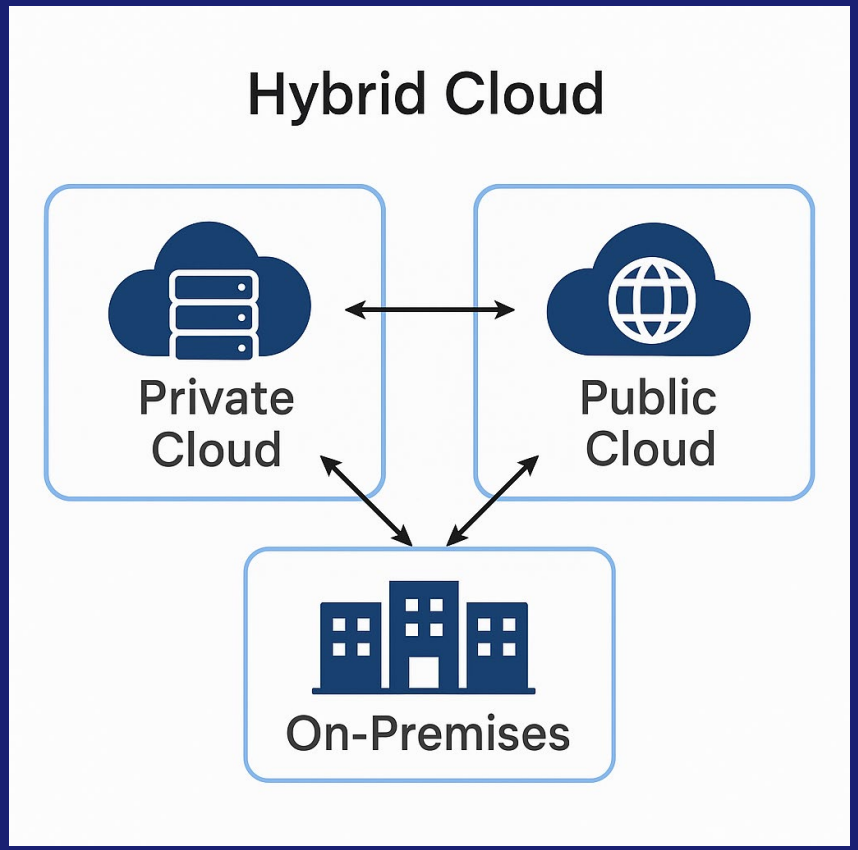
하이브리드 클라우드는 단순한 기술적 선택이 아니라, **비즈니스의 유연성과 제어권을 동시에 확보하려는 전략적 요구**에서 비롯된다.

퍼블릭 클라우드는 확장성과 민첩성을 제공하지만, **보안, 규제, 비용 통제, 레거시 호환성**은 온프레미스 인프라가 더 적합한 경우가 많다.

따라서 이 둘을 함께 묶어 일관된 방식으로 운영하는 하이브리드 아키텍처는,기업에게 **다양한 조건에서의 선택권과 최적화된 자원 활용**을 가능케 한다.

2025-05-18

하이브리드 클라우드



엔지니어의 전환

1. 온프레미스 인프라의 지속적인 감소 추세

Deloitte의 보고서에 따르면, 2025년까지 온프레미스 워크로드의 비중이 급격히 감소할 것으로 예상되며, 이는 클라우드로의 전환이 가속화되고 있음을 나타내.

2. 하이브리드 클라우드 환경에서의 기술 격차

NetworkWorld의 조사에 따르면, 기업의 62%가 클라우드 관리 기술 부족을 주요 성장 장애물로 인식하고 있어. 이는 온프레미스와 클라우드 기술을 모두 이해하고 운영할 수 있는 인재의 부족을 의미해. Network World

3. 전통적인 IT 역할의 변화

CBT Nuggets의 보고서에서는 시스템 관리자, 전통적인 DBA, 네트워크 엔지니어와 같은 전통적인 IT 역할이 감소하고 있으며, 이는 클라우드 중심의 기술 변화에 따른 것으로 분석돼. CBT Nuggets

2025-05-18

엔지니어의 전환

4. 하이브리드 클라우드의 중요성 인식

기업들은 퍼블릭 클라우드의 유연성과 온프레미스의 보안성을 결합한 하이브리드 클라우드 모델을 채택함으로써, 비용 효율성과 규제 준수를 동시에 달성이 가능. 이러한 모델은 특히 금융, 의료, 공공 부문에서 중요.

5. 온프레미스 기술자의 역할 재정의

온프레미스 기술자들은 클라우드 환경에서도 필요한 역할을 수행할 수 있도록, 클라우드 네이티브 기술과 자동화 도구에 대한 이해를 높이고, DevOps 문화에 적응.

6. 오픈소스 커뮤니티와의 협력 강화

오픈소스 기술은 클라우드 환경에서도 핵심적인 역할을 하므로, 기업들은 오픈소스 커뮤니티와의 협력을 통해 기술 지원을 확보하고, 내부 인재를 육성하는 전략을 고려.

왜 CSP는 온프레미스 기술자를 더 찾을까?

실제로 글로벌 기업은 점점 기존 온프레미스 엔지니어들을 자사의 핵심 엔지니어로 스카우트를 매우 적극적으로 하고 있다. 그 이유는 아래와 같다.

1. **하이브리드/멀티 클라우드 환경 통합 때문**

- CSP의 고객 대부분은 **온프레미스 시스템을 완전히 버리지 않음**. 예: ERP, 금융 시스템, 보안장비, 제조 자동화 시스템 등은 온프레미스 유지.
- 이들을 **클라우드와 연동, 마이그레이션, 보안 정책 적용**하려면 온프레미스에 정통한 인력이 반드시 필요함.

2025-05-18

왜 CSP는 온프레미스 기술자를 더 찾을까?

2. **에지 컴퓨팅 / 온사이트 리소스 수요 증가**

- 에지 클러스터 (ex. 제조, 의료, 군사 현장)는 여전히 물리적 인프라 기반이 많아.
- CSP는 해당 고객에 맞는 **온프레미스 장비 설계, 설치, 모니터링**까지 지원해야 함.

3. **오픈스택, 쿠버네티스 기반 프라이빗 클라우드 구축**

- Naver Cloud, NHN Cloud, KT Cloud, LG CNS 등 국내 CSP는 자체 클라우드 인프라를 온프레미스에서 운영 중.
- 이때 필요한 건 결국 **리눅스/네트워크/스토리지 기반의 온프레미스 인프라 엔지니어**임.

2025-05-18

왜 CSP는 온프레미스 기술자를 더 찾을까?

현재 채용 트렌드는 다음과 같다.

기업	채용 포지션	요구 기술 역량
NAVER Cloud	IDC 기반 오픈스택 운영자	Ceph, Libvirt, Ansible, Linux
KT Cloud	프라이빗 클라우드 엔지니어	On-prem HA 구성, 네트워크, 보안
AWS Outposts	하드웨어 설치 및 현장 유지관리 엔지니어	랙 구성, 라우팅, 보안 어플라이언스
NHN Cloud	MSP 고객 온프레미스 연동	VPN, VPC, H/W 장비 연동 경험

2025-05-18

왜 CSP는 온프레미스 기술자를 더 찾을까?

앞으로 엔지니어의 역할은 다음과 같이 변경이 될 예정이다.

구분	과거 역할	현재/미래 역할
온프레미스 인프라 엔지니어	장비 설치, OS 운영, 네트워크 구성	하이브리드 연동, 클라우드 마이그레이션, 보안 통합
채용 수요	점진 감소	CSP/MSP에서 고급 기술자로 수요 증가
필요 역량	리눅스, 스토리지, 장비 이해도	클라우드 네이티브 + 온프레미스 + DevOps 역량

왜 CSP는 온프레미스 기술자를 더 찾을까?

인력에 대한 전략은 최근에 다음과 같이 선언한다.

항목	현재 상황	미래 전략
온프레미스 인프라	지속적인 감소 추세	핵심 워크로드에 대한 전략적 유지
기술 인력	클라우드 및 온프레미스 기술 격차 존재	크로스 트레이닝 및 지속적인 교육 투자
오픈소스 기술 활용	기술 지원 및 유지보수에 어려움 존재	커뮤니티 참여 및 내부 기술 역량 강화

VM과 컨테이너를 동시에(하이브리드)

오늘날 많은 조직은 여전히 VM 기반 레거시 시스템과 컨테이너 기반 마이크로서비스를 동시에 운용하고 있다.

이 경우, OpenShift + KubeVirt, 또는 Harvester + Rancher 조합은 하이브리드 환경에서 **가상 머신과 컨테이너를 하나의 플랫폼에서 통합 운영**할 수 있는 가장 현실적인 선택지가 된다.

OpenShift Virtualization은 **KubeVirt를 기반으로 하면서도 Red Hat의 기술 지원과 보안 인증**을 제공하므로, 특히 **공공, 금융, 통신 분야의 하이브리드 구현에 적합하다**.

하지만, 밴더사 락인(Lock-In)를 피하기 위해서 Kubernetes + Kube-virt기반으로 구현이 가능하다.

2025-05-18

베어메탈 서비스

만약 내부에 서버 자원이 많고, 이를 **클라우드처럼 자동화된 방식으로 활용하고자 한다면**, MAAS(Metal as a Service)는 효과적인 해법이 된다.
MAAS는 REST API 기반으로 동작하며, **물리 서버의 자동 배포와 제어를 가능**하게 해주므로, 퍼블릭 클라우드와 유사한 환경을 내부에 구현할 수 있다.

MAAS는 **Harvester나 OpenStack 같은 상위 IaaS 구축의 전단 구성**으로도 매우 유용하게 사용된다. 지원하는 도구는 다음과 같다.

베어메탈 서비스

도구 이름	주요 기능	특징/비고	오픈소스 여부
Canonical MaaS	- 물리 서버 등록/관리 - PXE 부팅 및 자동 설치 - DHCP/TFTP 관리	Ubuntu와의 연동이 뛰어남. Web UI 제공. 대규모 자동화에 적합	✅ (GPL v3)
Foreman + Katello	- 설치 자동화 (Kickstart/Preseed) - DHCP/TFTP/DNS 연동- 패키지 관리	Red Hat 기반 시스템에 최적화. Satellite 대체로 사용됨	✅
Razor (Puppet)	- OS 자동 설치 - 정책 기반 프로비저닝	Puppet과 통합. 설정 복잡함. 현재 활발한 개발은 아님	✅
Cobbler	- OS 배포 자동화 - PXE 부팅 관리 - DHCP/TFTP 서버 내장	단순한 OS 설치 자동화 도구. 오래된 서버에도 적합	✅

베어메탈 서비스

도구 이름	주요 기능	특징/비고	오픈소스 여부
Tinkerbell	- 프로비저닝 워크플로우 정의 - 컨테이너 기반 - PXE + gRPC 활용	Equinix에서 개발. 모던한 아키텍처. Go 기반. 커스텀성 높음	✅
OpenStack Ironic	- Bare-metal Provisioning - PXE 및 IPMI 기반 설치 - 클라우드 연동	OpenStack 환경과 통합되어 사용됨. 클라우드 IaaS용으로 주로 활용	✅
netboot.xyz	- PXE 메뉴 기반 설치 이미지 선택 - 다양한 OS 지원	MaaS는 아님. 유틸리티 용도로 PXE 서버와 연동 가능	✅

2025-05-18

멀티 클러스터, 멀티 환경의 통합 관리가 필요할 때

하이브리드 클라우드의 핵심은 이질적인 환경을 **단일 정책 하에 운영할 수 있는 통합성**이다.

Rancher는 퍼블릭 클라우드(Kubernetes, EKS, GKE 등)와 온프레미스 클러스터(Harvester, MicroK8s 등)를 **단일 콘솔과 인증 체계로 통합 관리할 수 있는 능력**을 제공한다.

특히 Rancher는 **Harvester와의 긴밀한 통합**을 통해 Kubernetes 기반의 HCI까지 통제할 수 있으며, 이는 클라우드 네이티브 하이브리드 구현을 매우 직관적이고 가볍게 만들어준다.

선언형 기반의 서비스 구성 전략

단순한 VM 배포를 넘어서, **서비스 단위로 클러스터를 구성하고 운영하고 싶다면**, Canonical의 **Juju**는 훌륭한 선택이 된다.

Juju는 모델 기반의 오케스트레이션 프레임워크로, **퍼블릭/프라이빗 클라우드, 베어메탈, Kubernetes까지 통합한 선언형 관리**를 지원하며, 이 모든 과정을 **API 기반으로 자동화**할 수 있다.

2025-05-18

오픈소스 솔루션 기능

이러한 기능을 제공하는 오픈소스 솔루션은 다음과 같다.

항목	MAAS	Juju	Foreman	SUSE Manager
주요 목적	베어메탈 자동 배포	서비스 오케스트레이션	서버 설치 + 구성 관리	서버 구성 및 패치 관리
핵심 기술	PXE, IPMI, REST API	Charm, 모델 기반 제어	Kickstart, Ansible/Puppet	SaltStack, Uyuni 기반
대상 플랫폼	베어메탈 중심	클라우드, K8s, 베어메탈	베어메탈 + 일부 클라우드	SUSE, RHEL, CentOS 등
대체 가능성	Foreman 일부 기능 중첩	완전한 대체 불가	SUSE Manager와 유사	Foreman 완전 대체 가능

2025-05-18

오픈소스 솔루션 지원 기능 및 라이선스

구성 요소	Red Hat (RHEL)	SUSE (Rancher / SLE)	Canonical (Ubuntu)
가상화 하이퍼바이저	KVM (Apache 2.0)	KVM, Xen (GPLv2)	KVM (Apache 2.0)
컨테이너 플랫폼	OpenShift(Apache 2.0, 일부 LGPL)	Rancher, K3s (Apache 2.0)	MicroK8s (Apache 2.0)
멀티클러스터 관리	Advanced Cluster Management (상용 일부 Apache)	Rancher Manager + Fleet (Apache 2.0)	Juju, LXD 클러스터링 (AGPLv3 / Apache 2.0)
베어메탈 프로비저닝	Foreman (GPLv3), Red Hat Satellite (상용)	SUSE Manager + SaltStack (GPLv2 + Apache 2.0)	MAAS (GPLv3 + 일부 proprietary UI)

2025-05-18

오픈소스 솔루션 지원 기능 및 라이선스

구성 요소	Red Hat (RHEL)	SUSE (Rancher / SLE)	Canonical (Ubuntu)
서비스 오케스트레이션	Ansible Automation Platform (GPLv3 / 상용)	SaltStack (Apache 2.0)	Juju Charms (AGPLv3)
네트워크/스토리지 구성	OVN, Ceph, Rook (Apache 2.0)	Longhorn, Flannel, RKE2 (Apache 2.0)	Open vSwitch, Ceph (Apache 2.0 / LGPL)
자동화 및 IaC	Ansible, Terraform (GPLv3 / MPL-2.0)	SaltStack, Terraform, AutoYaST (혼합 라이선스)	Ansible, Cloud-init (GPLv3 / Apache 2.0)

제안

목표: 퍼블릭/프라이빗 클라우드, 베어메탈 및 VM/컨테이너 워크로드를 유연하게 통합하는 하이브리드 클라우드 환경 구현

핵심 조건

- 가상화와 컨테이너 기반 워크로드의 공존
- 멀티 클러스터 및 베어메탈 자동화
- API 기반의 오케스트레이션과 자동화 통합
- 오픈소스 기반 구성 및 라이선스 투명성

2025-05-18

제안

위의 조건으로 다음과 같이 소프트웨어 조합이 가능하다.

목적	Red Hat Stack	SUSE Stack	Canonical Stack
VM 기반 하이퍼바이저	RHEL + KVM	SLE + KVM/Xen	Ubuntu + KVM
컨테이너 오케스트레이션	OpenShift + ACM	Rancher + K3s/RKE2	MicroK8s
베어메탈 자동화	Foreman or Satellite	SUSE Manager + Salt	MAAS
서비스 단위 자동 구성	Ansible Roles / Playbook	Salt State + Formulas	Juju + Charms
클러스터 스토리지	Ceph, Rook	Longhorn	Ceph

하이브리드 클라우드에서 API 통합 운영

하이브리드 클라우드는 **서로 다른 환경(Public, Private, Edge, VM, Container 등)**이 혼합된 구조이기 때문에, **운영 일관성, 보안, 자동화, 확장성 확보**를 위해 반드시 **공통 제어 지점**이 필요하다.

이때 핵심이 되는 것이 바로 **API 기반 통합 운영**이다.

하이브리드 클라우드에서 API 통합 운영

이유 1: 이질적인 인프라의 표준화

- OpenStack, AWS, KVM, Harvester 등은 모두 제각기 다른 CLI와 대시보드를 제공한다.
- API를 사용하면 이들을 **공통된 스크립트, 자동화 도구, UI로 통합** 가능하다.
  → 예: Ansible, Salt, Terraform, Juju, Crossplane이 모두 REST API 기반

이유 2: 자동화의 전제조건

- IaC, GitOps, 워크플로우 자동화는 전부 API 호출로 실행된다.
- API가 없으면 **수동 조작 또는 불완전한 자동화**에 머물 수밖에 없다.

2025-05-18

하이브리드 클라우드에서 API 통합 운영

이유 3: 보안과 접근 통제

- API Gateway, RBAC, 토큰 인증(OAuth, JWT) 등은 **API 단위로 보안 정책 적용**이 가능하다.
- 사용자마다 API 권한을 세분화함으로써 **서비스 간 책임 분리** 가능

이유 4: 중앙 집중형 관제/정책 통제 가능

- Crossplane, Cloudify, ManageIQ, Mist.io는 다양한 자원을 **API 기반으로 감시하고 통제**할 수 있는 플랫폼이다.
- 예: Cloudify에서 AWS EC2를 API로 호출 → 상태 확인 후 자동 확장 → K8s 서비스 재구성까지 연동

ManageIQ

CMP기반으로 API통합 및 운영이 필요한 경우, 다음과 같은 조건으로 구성이 가능하다.

1. **ManageIQ (Red Hat 기반, 오픈소스)**

- **기능**: OpenStack, VMware, Azure, AWS 등 다양한 IaaS 플랫폼을 API로 통합 관리
- **특징**:
  - RBAC 기반 통합 사용자 관리
  - 자원 할당/요청 포털 (Self-service Portal)
  - 가상머신/스토리지/네트워크 통합 분석
  - 이벤트 기반 자동화 (Policy + Alert 기반)

ManageIQ

- **하이브리드 활용법:** OpenStack 프라이빗 클라우드와 AWS를 함께 쓰는 환경에서 **단일 통합 관제 플랫폼**으로 구성 가능
- **라이선스:** Apache 2.0
- **비고:** Red Hat CloudForms의 업스트림 프로젝트

Mist

2. Mist.io (클라우드 통합 포털 중심 CMP)

- **기능**: 온프레미스, 퍼블릭 클라우드, 컨테이너까지 통합 GUI 관리
- **특징**:
  - 웹 기반 인프라 제어판 (multi-cloud dashboard)
  - 템플릿 기반 VM/컨테이너 생성
  - Web SSH, 정책 기반 통제
- **하이브리드 활용법**: 사내 KVM + 클라우드(K8s/GCP) 자원을 **한 화면에서 관리하고 사용자별 자원 제한 가능**
- **라이선스**: 오픈코어 (MIT for community, 유료 버전 존재)
- **비고**: DevOps/SMB 친화적, Rancher 대안으로 가볍게 활용 가능

Crossplane

3. Crossplane (Kubernetes 기반 하이브리드/멀티 클라우드 제어)

- **기능**: Kubernetes 내에서 AWS, GCP, Azure 리소스를 CRD 형태로 선언하고 관리
- **특징**:
  - 인프라를 Kubernetes 리소스로 추상화 (XRD, Composition)
  - GitOps 친화적 (FluxCD, ArgoCD 통합)
  - Provider 모델로 온프레미스 확장 가능 (vSphere, RDS, S3 등)
- **하이브리드 활용법**: K8s 클러스터 내에서 VM, DB, S3를 하나의 YAML 리소스로 선언적 관리
- **라이선스**: Apache 2.0
- **비고**: Platform Engineering에서 인기 상승 중

Cloudify

4. Cloudify (워크플로우 중심의 인프라 자동화 및 CMP)

- **기능**: Terraform, Ansible, REST API 등 다양한 인프라 도구를 연결하는 **워크플로우 중심 CMP**
- **특징**:
  - Service Blueprint 기반 선언형 배포
  - 멀티 클라우드 워크플로우 연결
  - REST API로 확장 가능, 외부 시스템 연계 강력
- **하이브리드 활용법**: Terraform + Ansible + vSphere + AWS 등 **인프라 자동화를 구성**할 수 있음
- **라이선스**: Apache 2.0
- **비고**: DevOps와 SRE 팀에 적합, GUI + CLI + API를 모두 제공

정리

도구	주요 특징	하이브리드 활용 포인트	라이선스
ManageIQ	통합 인벤토리/VM/스토리지 관리	OpenStack + AWS 통합 관제	Apache 2.0
Mist.io	웹 기반 멀티 클라우드 제어판	가볍게 KVM + GCP 관리	MIT (community)
Crossplane	K8s 기반 인프라 선언 + GitOps	K8s로 AWS/RDS/S3 포함 전체 리소스 선언	Apache 2.0
Cloudify	복합 워크플로우/외부 도구 통합	Ansible + Terraform + Cloud 연결 자동화	Apache 2.0

랩