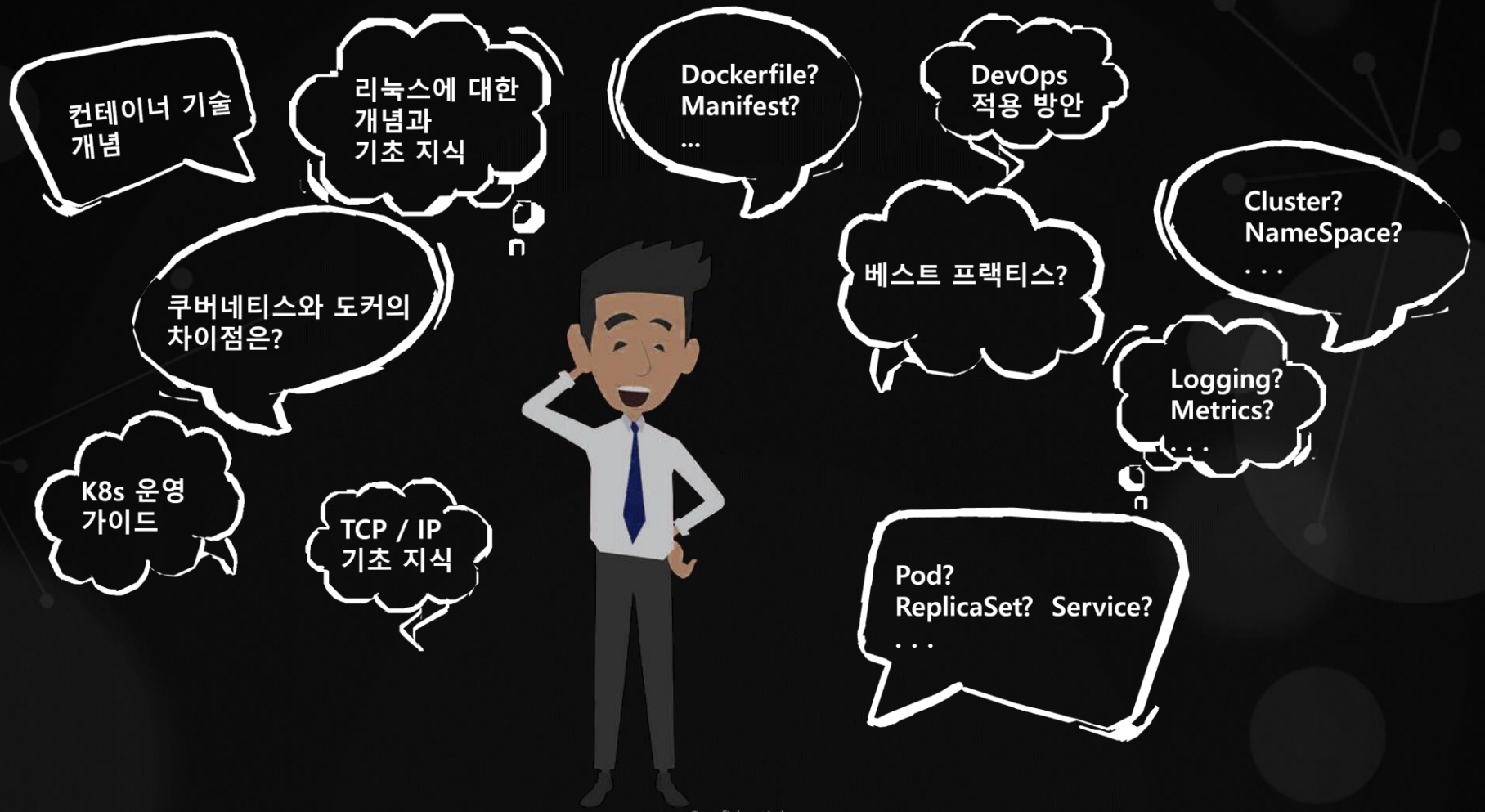

openmaru
APM
Kubernetes 소개

Kubernetes 는 학습장벽이 높은 기술

- 쿠버네티스를 유용하게 활용하기 위해서는 다양한 지식이 요구



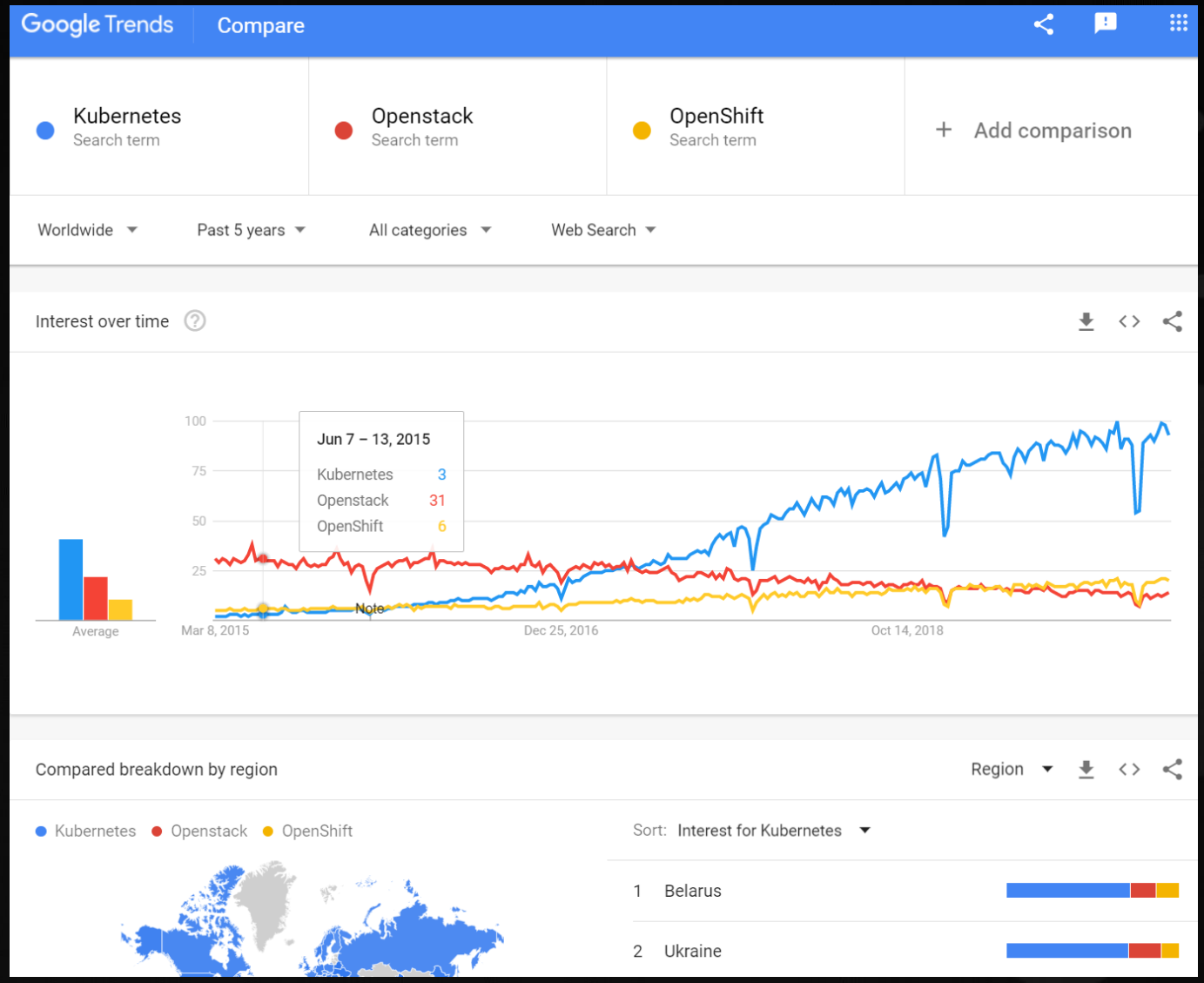
Kubernetes는 누구에게 필요한 도구 일까요?

- IT 운영 부담을 줄이려고 하는 개발팀,

  시스템 운영팀, 네트워크 운영팀, DEVOPS 팀

  - 애플리케이션 변경 요건이 많고,하루에 여러 번 배포
  - 이벤트로 인한 부하에 대응할 수 있는 시스템 요구
  - 빈번한 설정 변경과 시스템 작업으로 인한 이력 관리
  - 복잡한 작업 절차서
  - 배포 정책 요구와 롤백 방안

Google Trends –
Kubernetes vs. OpenStack vs. OpenShift




가상화 기술과 컨테이너 기술의 차이점
Google & Kubernetes

GOOGLE 과 컨테이너

- **Google의 업무 방식**

Gmail에서 YouTube, 검색에 이르기까지 Google의 모든 제품은 컨테이너에서 실행됩니다.

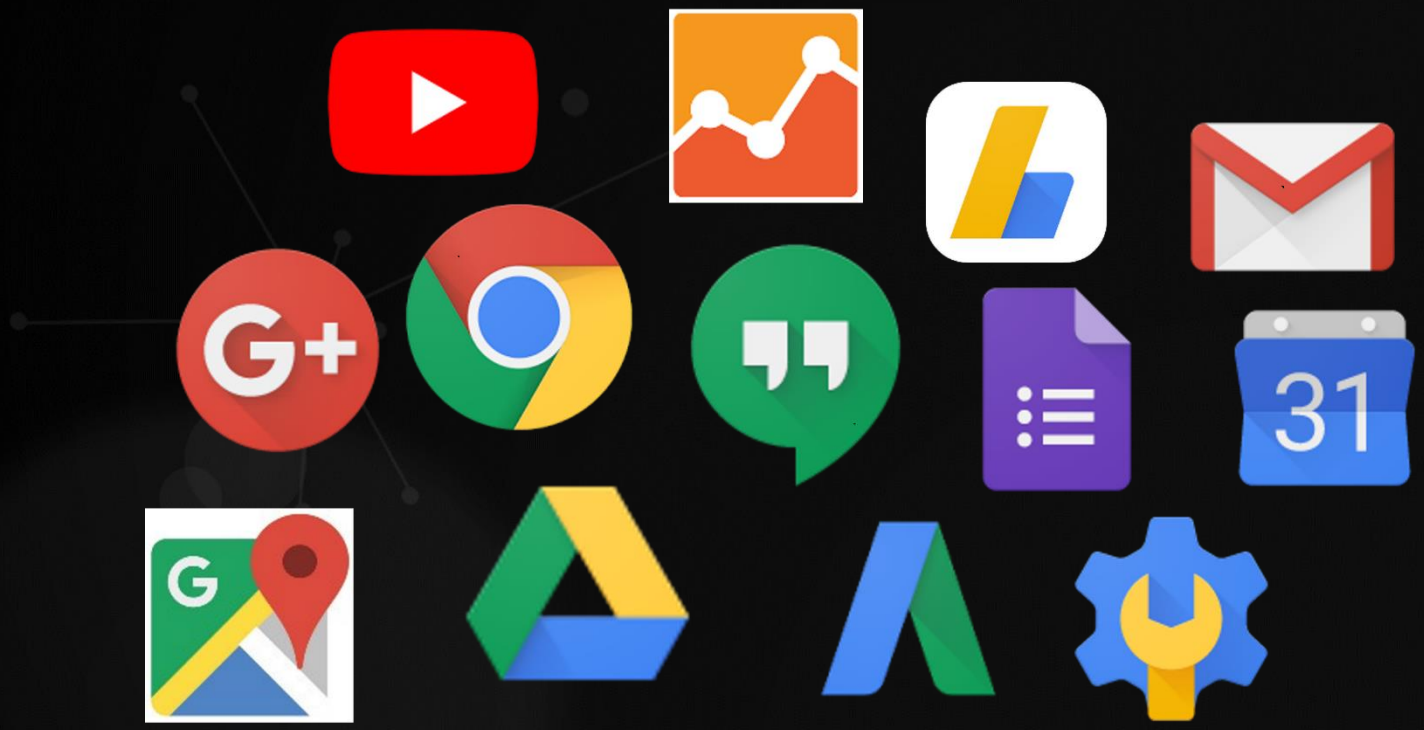
개발팀은 컨테이너화를 통해 더욱 신속하게 움직이고, 효율적으로 소프트웨어를 배포하며 전례 없는 수준의 확장성을 확보할 수 있게 되었습니다. **Google은 매주 수십억 개가 넘는 컨테이너를 생성**합니다. **지난 10여 년간 프로덕션 환경에서 컨테이너화된 워크로드를 실행하는 방법에 관해 많은 경험**을 쌓으면서 Google은 커뮤니티에 계속 이 지식을 공유해 왔습니다.

초창기에 cgroup 기능을 Linux 커널에 제공한 것부터 **내부 도구의 설계 소스를 Kubernetes 프로젝트로 공개**한 것까지 공유의 사례는 다양합니다. 그리고 이 전문 지식을 Google Cloud Platform으로 구현하여 개발자와 크고 작은 규모의 회사가 최신의 컨테이너 혁신 기술을 쉽게 활용할 수 있도록 하였습니다.

Source - https://cloud.google.com/containers/

Google 는 모두 컨테이너에서 실행

- Gmail , 검색, 지도 ...
- MapReduce , GFS , Colossus ...
- Google Compute Engine 가상 머신도 컨테이너에서 실행!
- 매주 20 억개 이상의 컨테이너를 실행 중

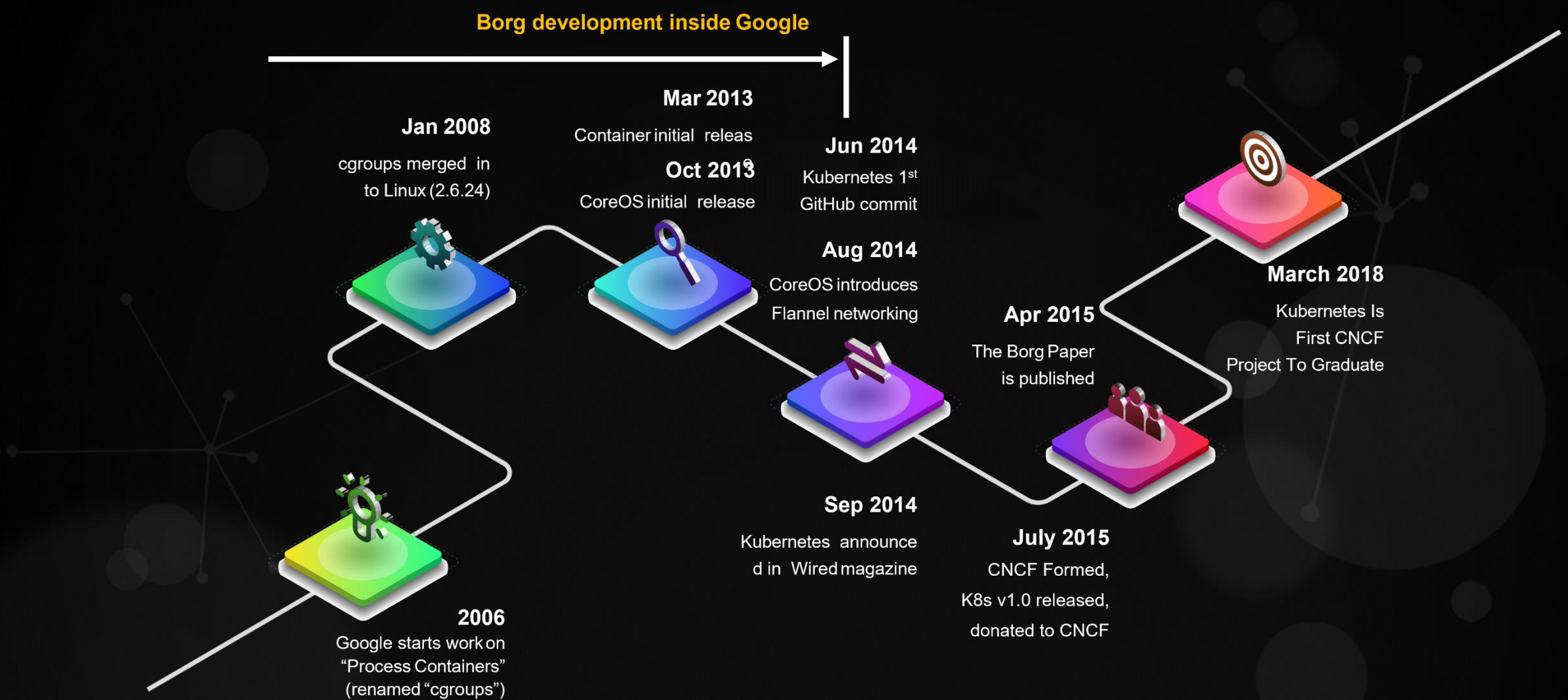


About Kubernetes

- 쿠버네티스(K8s)는 컨테이너화된 애플리케이션을 자동으로 배포, 스케일링 및 관리해주는 오픈소스 소프트웨어
- 쿠버네티스”, “쿠베르네테스”,“K8s”, “쿠베”, “쿠버”, “큐브”라고 부르며 Apache License 2.0 라이선스로 리눅스 재단 (Linux Foundation)산하 Cloud Native Computing Foundation (CNCF) 에서 관리
- Go로 작성된 오픈 소스 , OSS (Apache License 2.0) 라이선스
- 구글에서 개발하고 설계한 플랫폼으로서 사내에서 이용하던 컨테이너 클러스터 관리 도구인 “Borg”의 아이디어를 바탕으로 개발

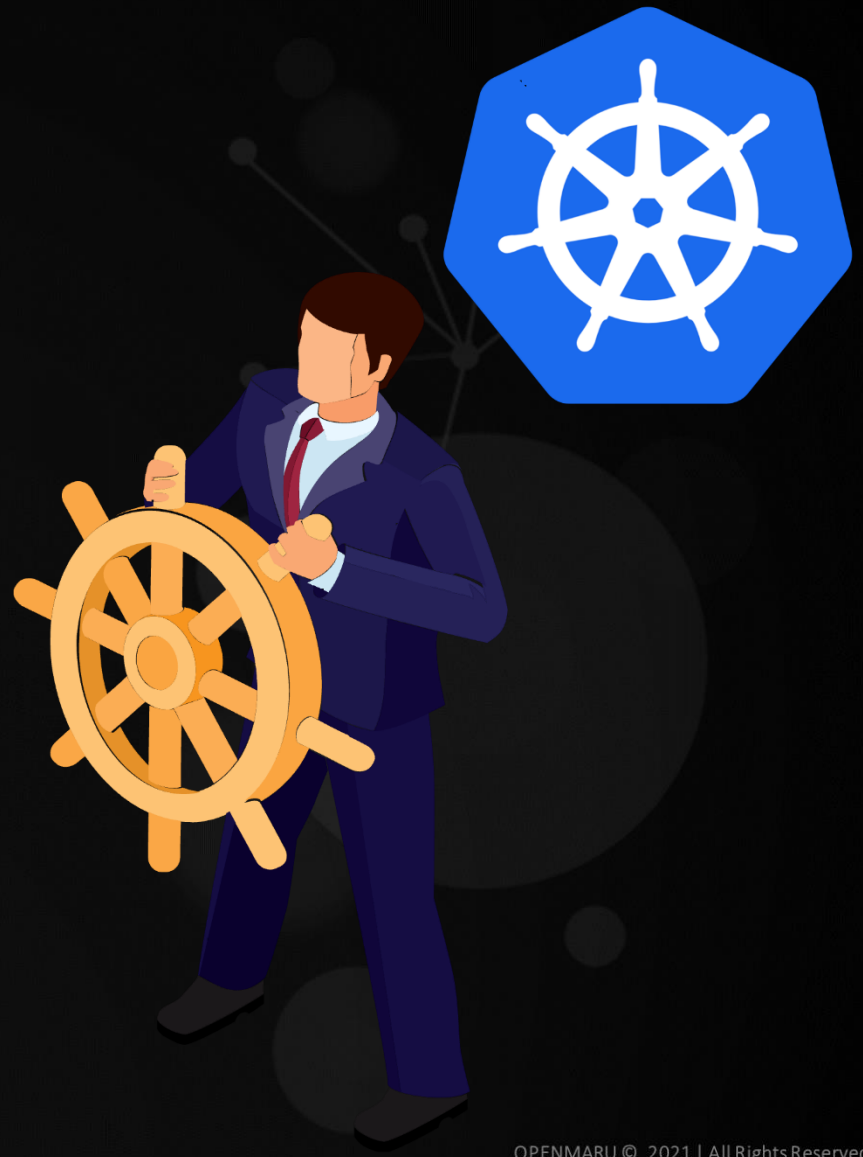
> "Kubernetes is open source-a contrast to Borg and Omega, which were developed as purely Google-internal systems. "
>
> *- Borg, Omega, and Kubernetes*

Kubernetes History



Kubernetes는

- **Kubnetes는 Open Source Software**
  - github.com/kubernetes/kubernetes (Apache License 2.0)
  - **오픈 디자인**
    - github.com/kubernetes/community
  - **오픈 커뮤니티**
    - Cloud Native Computing Foundation
    - Special Interest Groups (SIGs)
- **Kubernetes 는 Portability**
  - **cloud-controller-manager (클라우드 작업)**
    - L4 LB 블록 스토리지 등의 작업을 추상화
    - GCP, AWS, Azure, OpenStack 등
  - **CNI (Container Network Interface)**
    - 컨테이너 네트워크의 표준화
    - Calico, Flannel, Weave Net 등
  - **CRI (Container Runtime Interface)**
    - 컨테이너 작업의 표준화
    - Docker (containerd) cri-o, rkt



Container & Kubernetes

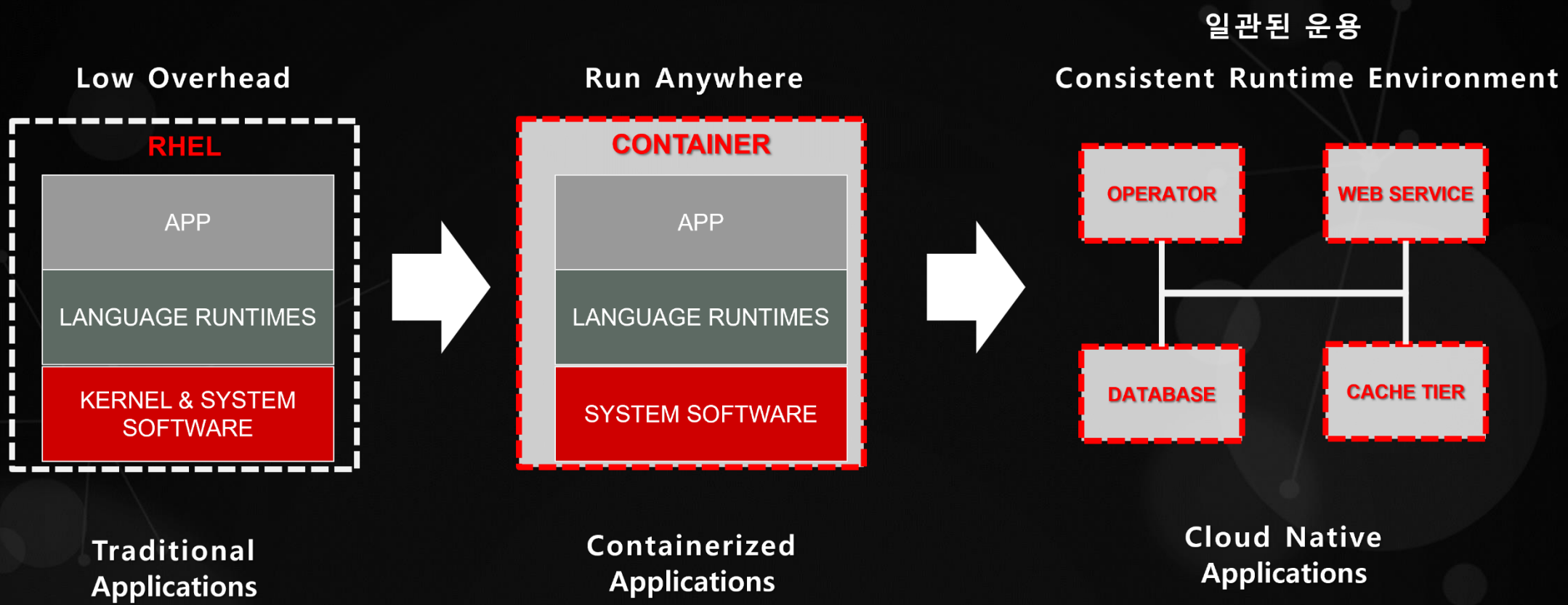
Container & Kubernetes

Kubernetes 설치 뿐만이 아니라 인프라 전반에 대해 검토

- Kubernetes 와 네트워크 구성
- Kubernetes용 스토리지
- Kubernetes 에 적합한 업데이트 방법
- Kubernetes 에 적합한 모니터링
- Kubernetes 에 대한 개발팀 적응
- Kubernetes 에 적합한 운영 방안
- Kubernetes 용 로그 통합 방안
- Kubernetes Troubleshooting 방안
- 애플리케이션에 대한 컨테이너화
- 각종 구성과 설정을 위한 YAML 작성
- Kubernetes 을 이용한 운영 자동화
- Kubernetes 기반 백업/이중화/네트워크/스토리지 통합

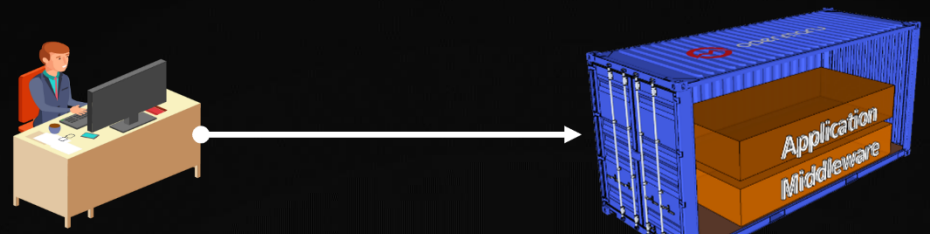
왜 지금 컨테이너화 하는가?

- 컨테이너화의 이유는 가상화 하는 이유가 같다.



컨테이너의 과제

- 여러 대의 컨테이너 플랫폼을 관리하는 번거로움

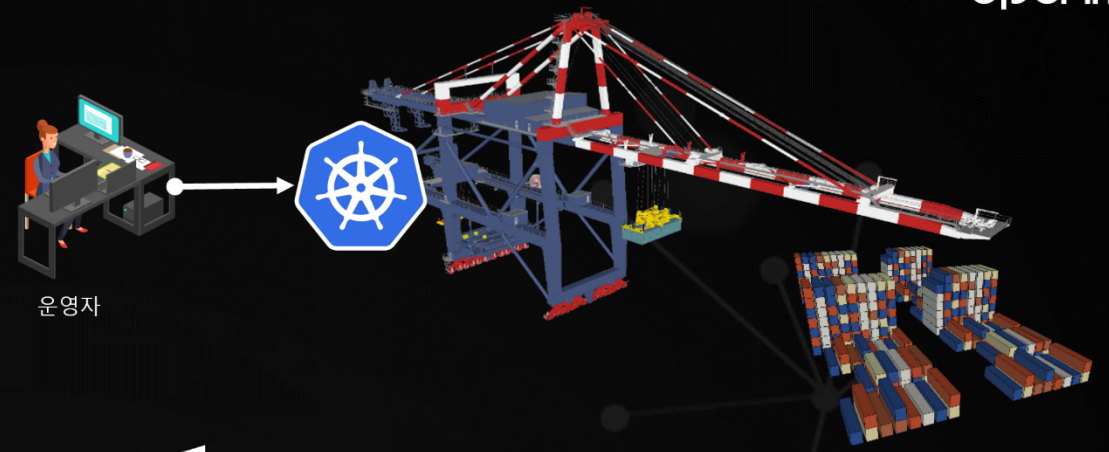


인프라 관리자의 컨테이너 운영 관리 이슈

작업	주요 작업 사항
컨테이너 배포	컨테이너를 배포할 호스트 결정과 배포
	컨테이너에 할당할 MAC 주소, IP 주소, 호스트 이름 관리
	애플리케이션에 대한 라우팅
	애플리케이션이 사용하는 데이터를 유지하는 영구 저장소 할당
장애 발생	노드 장애 시 : 다른 노드로의 재배치
	중복 결함시 : 복제 수 확보
부하 변동 부하에 맞게	스케일 아웃 / 인

Kubernetes 특징과 컨테이너 과제에 대한 해결책

- **Kubernetes 특징**
  - 여러 컨테이너 호스트를 중앙에서 관리
  - Pod 예약 사활 감시, 시작 / 정지
  - Pod 네트워크 설정
  - Pod 중복 보장
  - 부하에 따른 자동 확장/축소



컨테이너의 과제	Kubernetes 의한 해결
컨테이너 배포	**설정 파일에 따라 컨테이너를 배치하고 배포** · 배포 대상 호스트의 결정 Pod 단위에서의 IP 주소 부여 및 연결 경로 설정, 영구 저장소 연결을 설정 파일에 따라 Kubernetes가 자동으로 실시
장애 발생	**자동 복구에 의한 가용성** 확보 · 설정 파일에 따라 Kubernetes가 실행 복제본 수를 일정하게 유지 노드 장애와 지정 수량에 맞게 자동으로 복구
부하 변동	**업무 절정의 스케일 조작과 자원 관리의 부하 경감** · 접속량 리소스 사용량을 모니터링 부하에 따라 자동으로 추가 배치

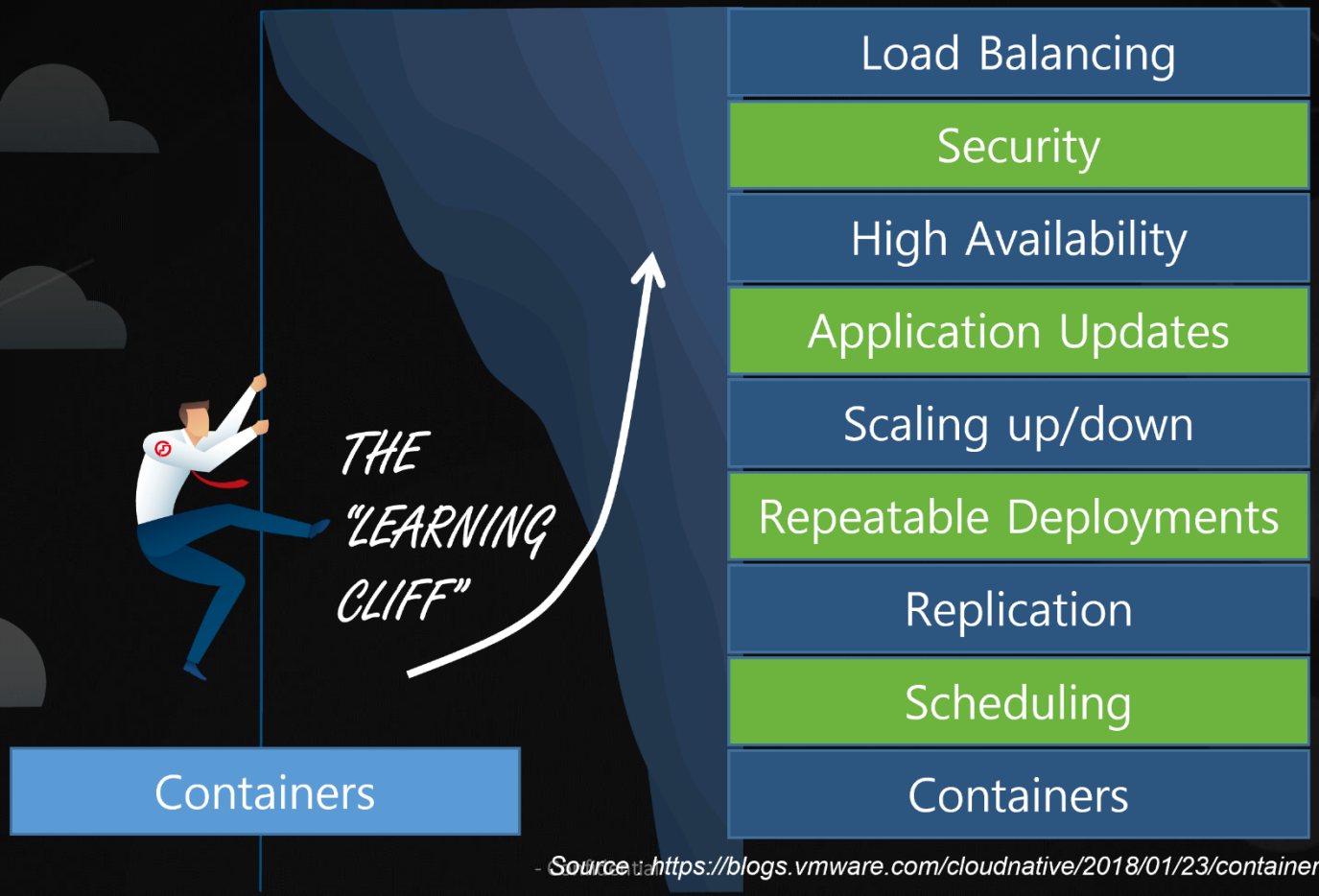

Container Orchestration
Kubernetes 소개
openmaru
APM

Containers and Kubernetes: The Time Is Now

openmaru

CONTAINERS IN DEVELOPMENT

CONTAINERS IN PRODUCTION

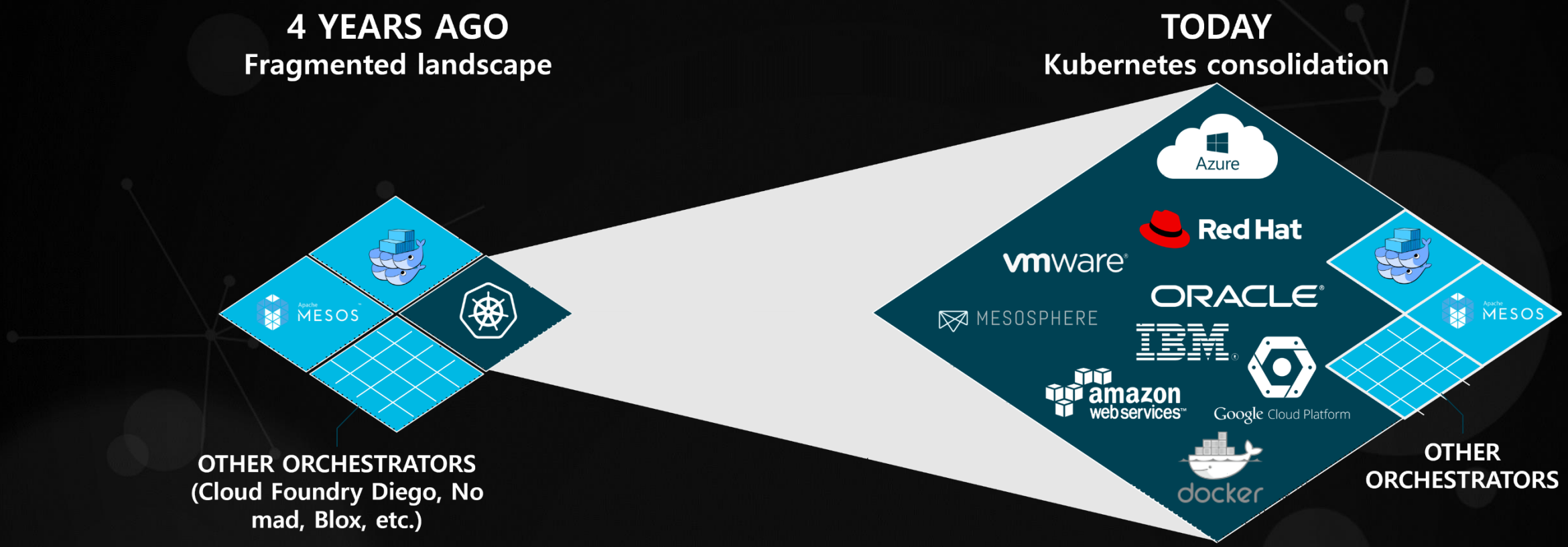


Source: https://blogs.vmware.com/cloudnative/2018/01/23/containers-kubernetes-benefits/

Kubernetes는 컨테이너 오케스트레이션의 표준

openmaru

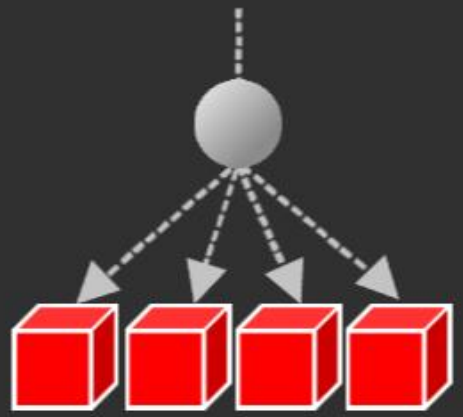
Kubernetes = Container Orchestration Standard



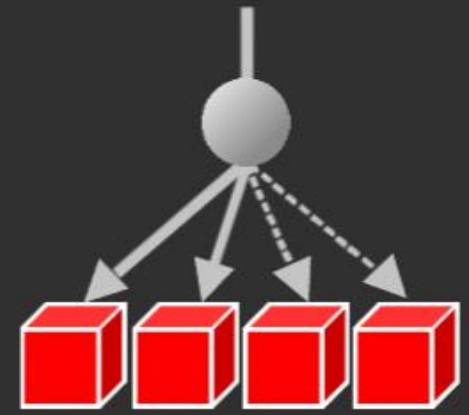
openmaru

Auto Scale Out / In

Load Balancer

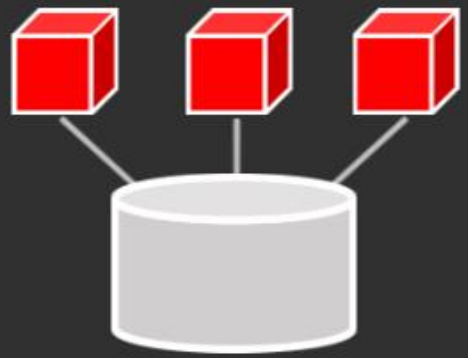


Auto Rolling Update

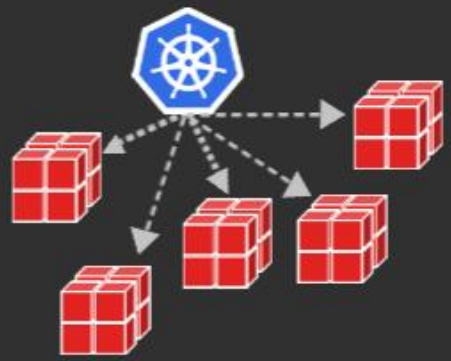


Auto Healing

Persistence Volume

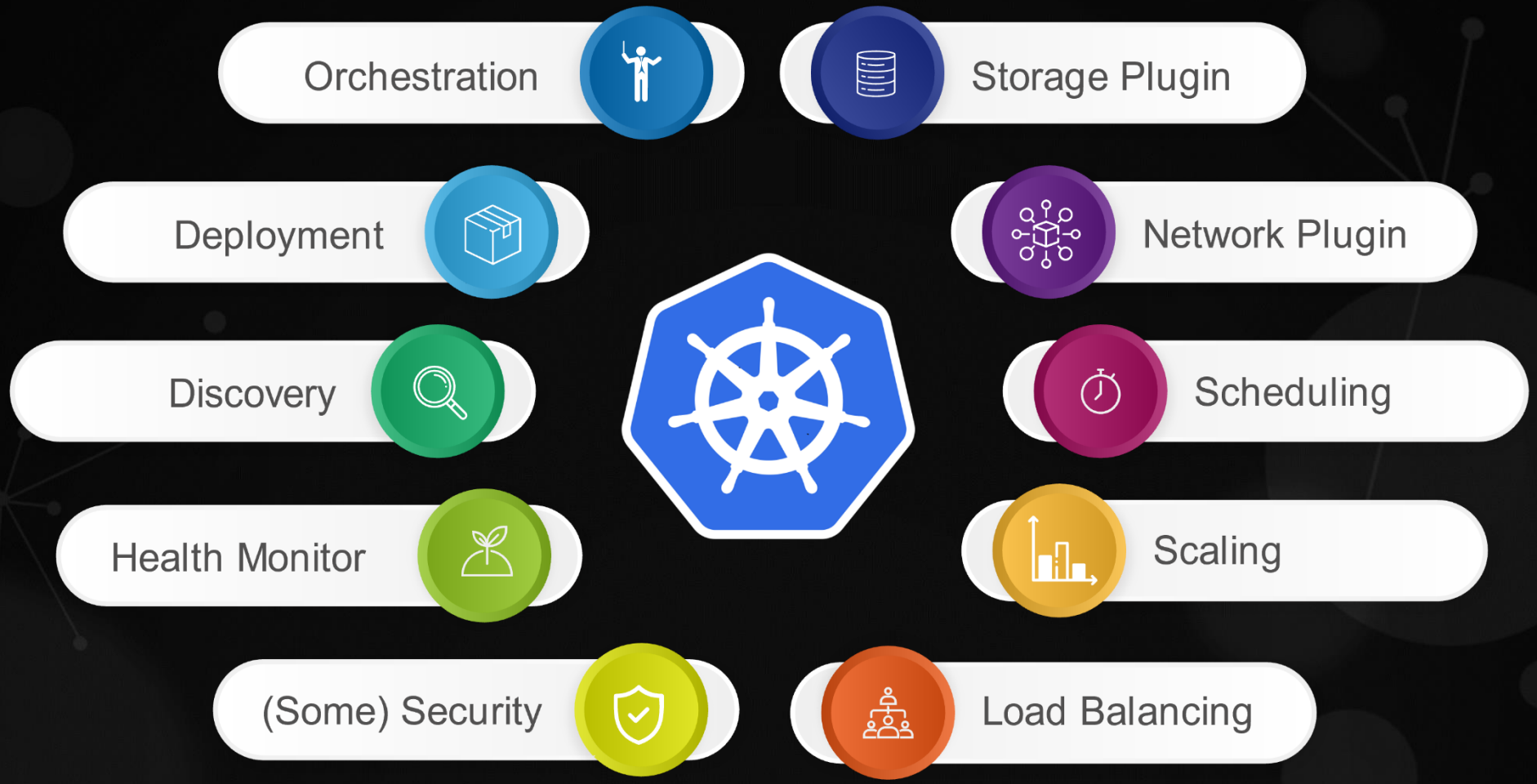


Container Orchestration

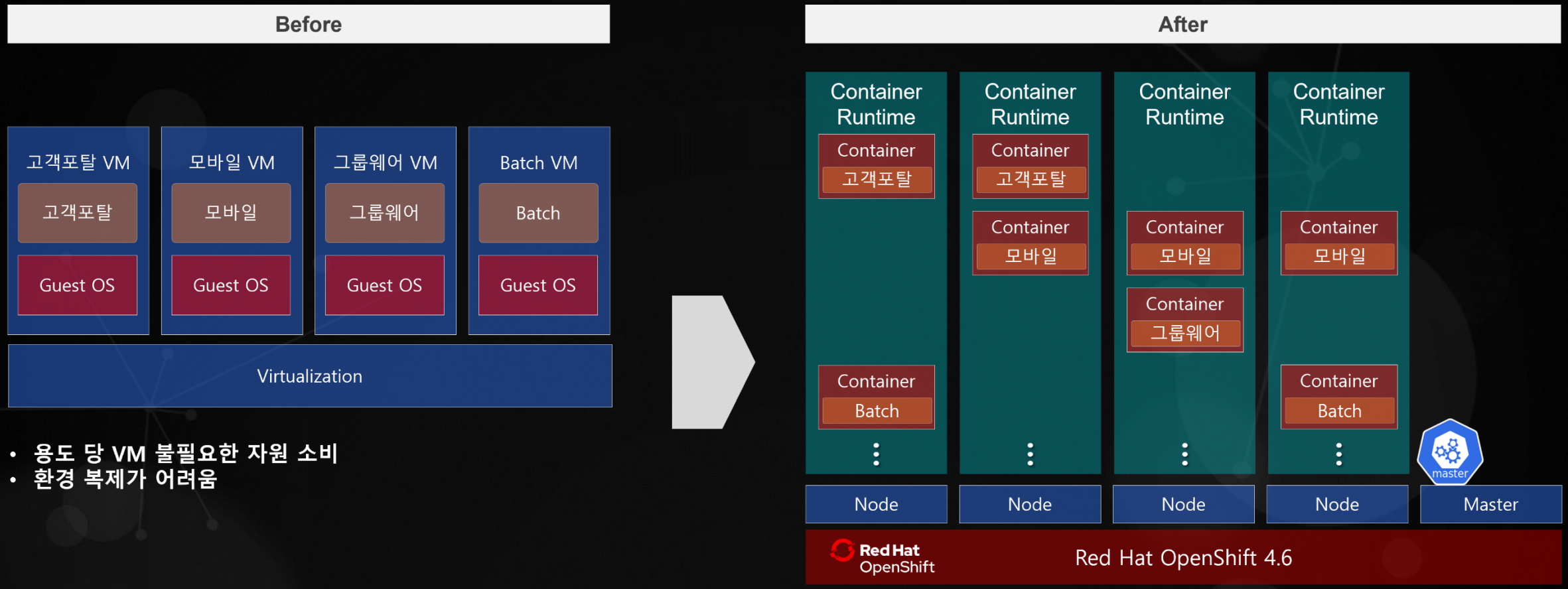


KUBERNETES DOES A LOT FOR YOU

openmaru



Kubernetes 의한 컨테이너 오케스트레이션의 실현



- 용도 당 VM 불필요한 자원 소비
- 환경 복제가 어려움

- Kubernetes이 Node의 자원 상황을 보고 적절한 컨테이너를 배치
- 손쉬운 개발환경 구축 과 관리자의 개입을 최소화 하여 자동 확장/복구 실현

컨테이너 운영 환경의 발전

- 초기에는 개별 프로젝트팀이나 R&D 팀에서 컨테이너를 구성
- 컨테이너 이용이 확대됨에 따라 관리프로세스 확립과 함께 운영팀에서 통합
  - 환경 구축 작업의 공수 절감, 작업 부하 감소
  - 프로덕션에서 환경적인 문제 감소 (운영 및 개발을 동일한 환경으로 구축)
  - 장애 시 대체 환경의 전환 시간의 단축 및 작업 부하의 감소

가상화 기술과 컨테이너 기술의 차이점

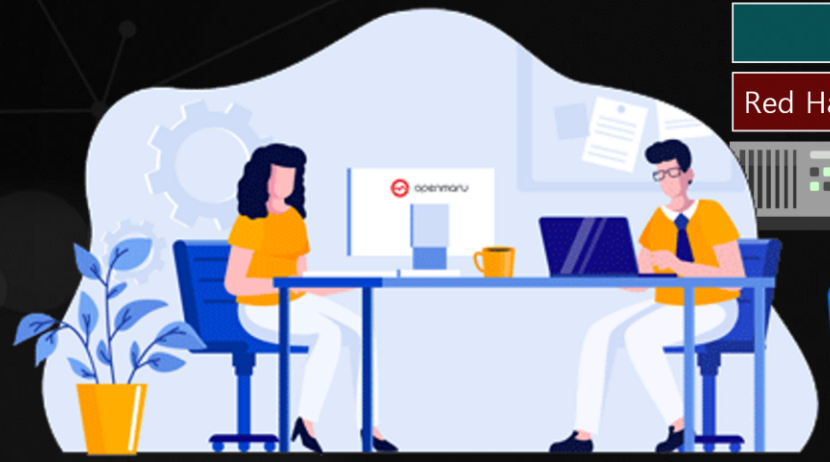
인프라에 대한 고민과 Kuberntes

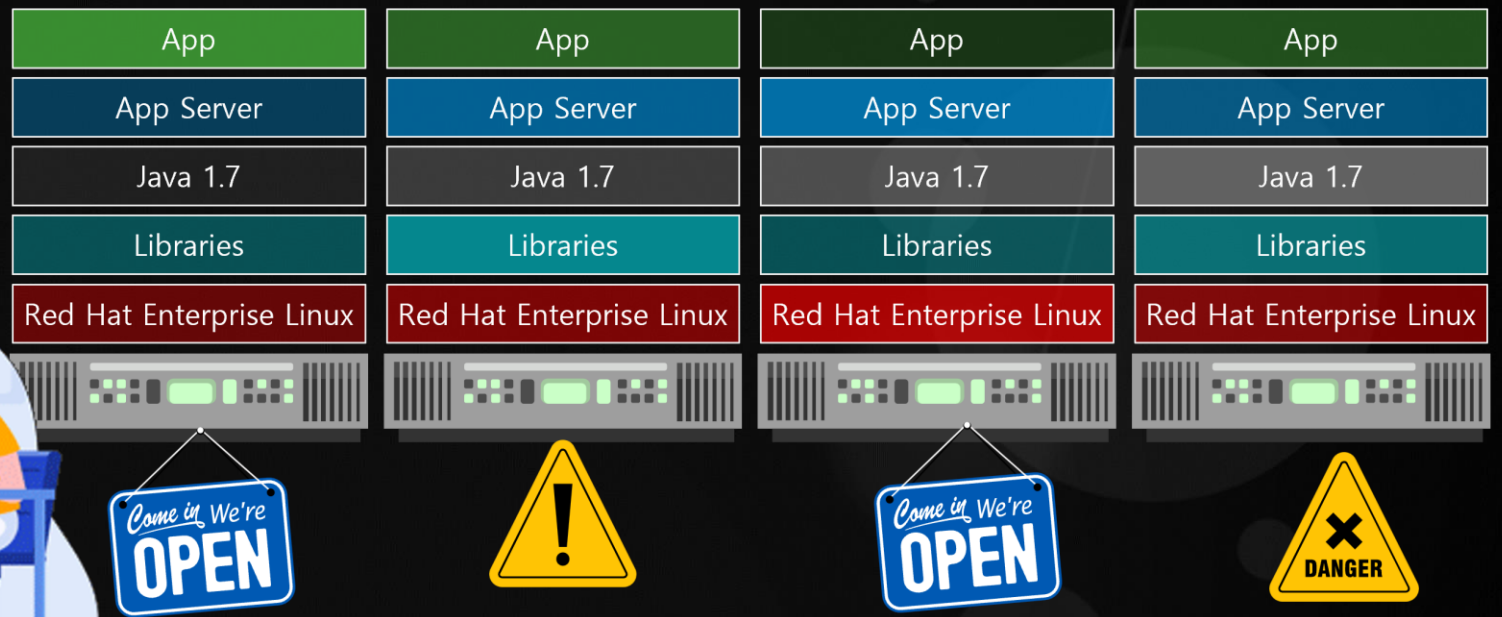
openmaru APM

- Configuration Drift
- Mutable Infrastructure vs. Immutable Infrastructure
- Reconciliation
- 멱등성

Configuration Drift

- 컨피그레이션 드리프트
  - 손으로 직접 수정한 임시 수정/업데이트와 전반적인 엔트로피(entropy) 증가로 인해 인프라의 서버들이 시간이 갈수록 점점 서로 다른 상태가 되는 현상
  - 장비의 라이프사이클 동안 초기 설정으로부터 멀어지고(drift) 다른 장비들 과도 서로 달라짐



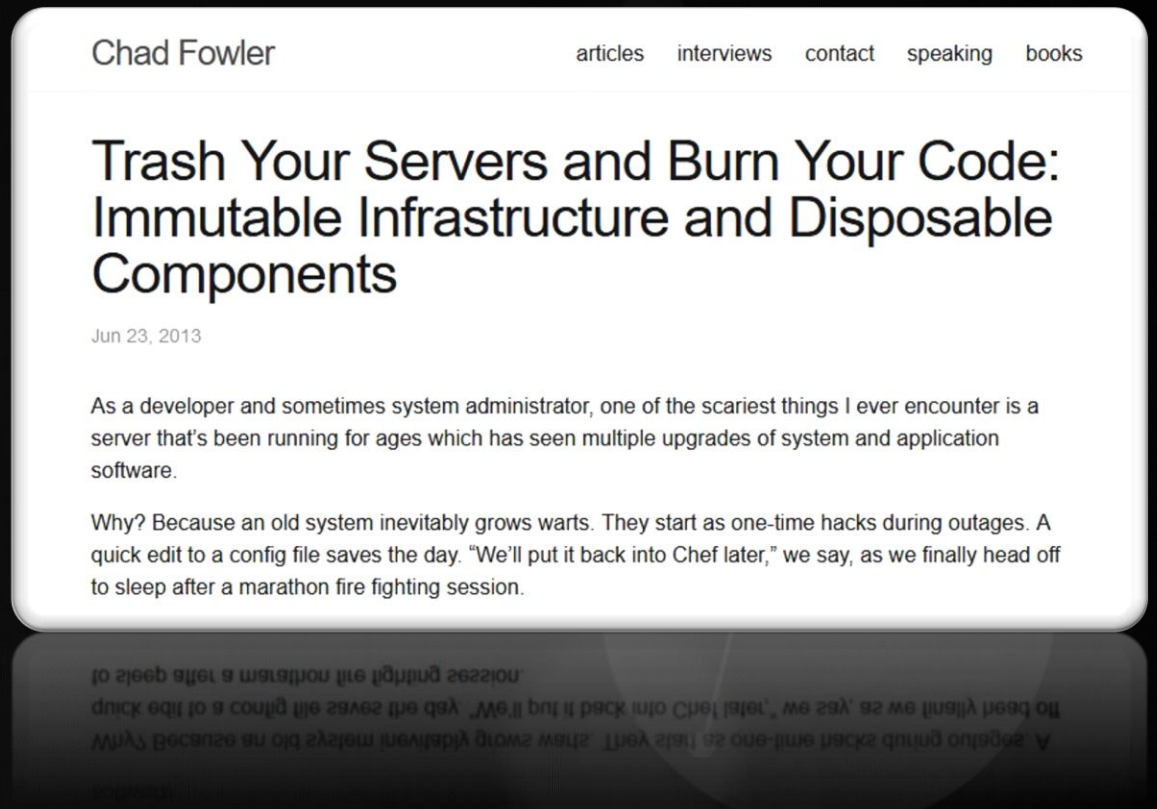


Trash Your Servers and Burn Your Code: Immutable Infrastructure and Disposable Components

시스템 관리자로서 내가 가장 무서워하는 것 중 하나는 오랜 기간 운영된 서버, 특히 시스템과 응용 프로그램을 여러 번 업그레이드한 서버입니다.
왜일까요? 오래된 시스템은 필연적으로 보이지 않는 문제를 키우기 때문입니다.

...

업그레이드가 필요한가요? 문제 없습니다. 업그레이드된 신규 시스템을 도입하고 이전 것을 버리세요.
애플리케이션의 신규 버전을 배포해야 한다고요? 마찬가지입니다. 새 버전의 애플리케이션이 탑재된 서버를 생성하고 이전 것을 제거하세요.



http://chadfowler.com/2013/06/23/immutable-deployments.html

Immutable Infrastructure (불변의 인프라스트럭쳐)

- 불변의 인프라스트럭쳐 정의
  - 서버가 배포된 이후 절대 변경되지 않는 형태의 인프라 패러다임
    - 업데이트는 덮어 쓰는 것이 아니라 버리고 새롭게 만드는 것 즉 변경하지 않는 것
    - "No Upgrade needed"
  - Disposable Components "폐기 가능한"
- 멱등성 법칙
  - 같은 작업을 여러 번해도 결과가 동일
  - 한번 설정된 서버는 수정없이 파기되므로 멱등성 보장
- 장점
  - 서버를 쉽게 삭제하고 늘릴 수 있음.
  - 인프라 환경의 변경이 쉬워짐.
  - 서버를 깨끗이 유지할 수 있음. (Configuration Drift 현상 걱정 노노)

Pets vs Cattle

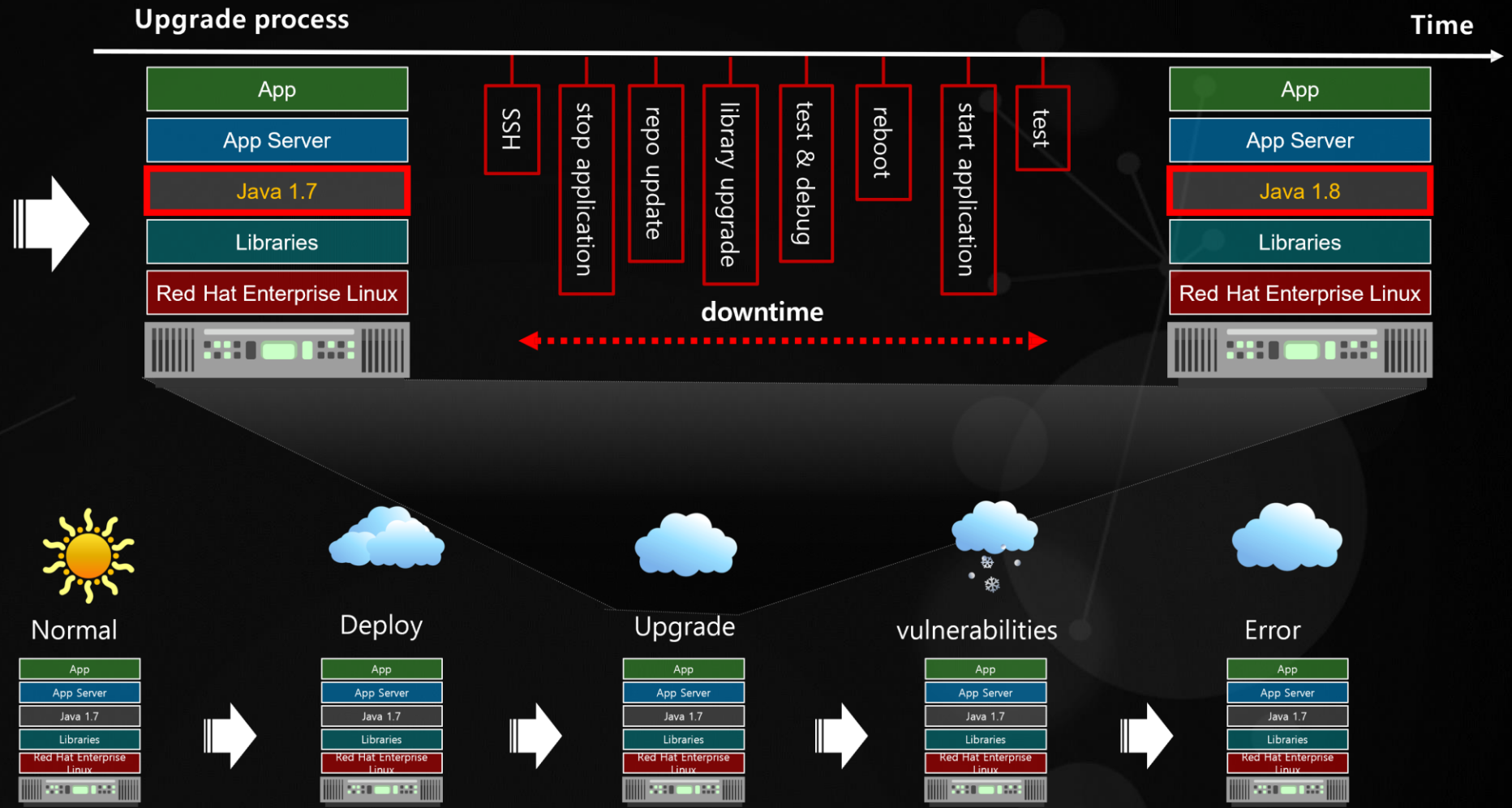
openmaru

Pets

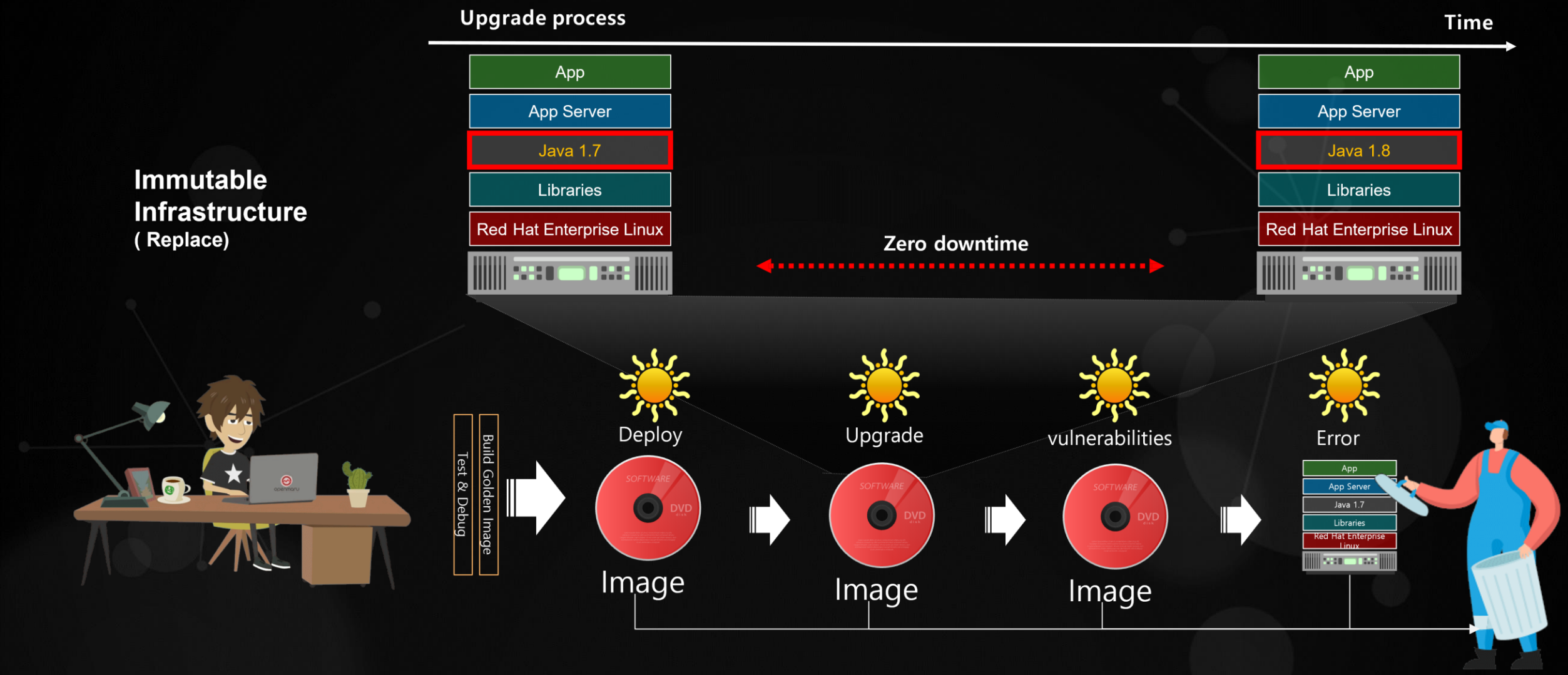
Cattle

In-place Server Upgrade

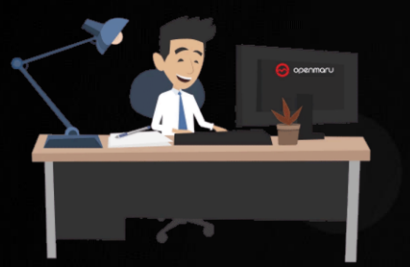
Mutable Infrastructure
(In-Place)

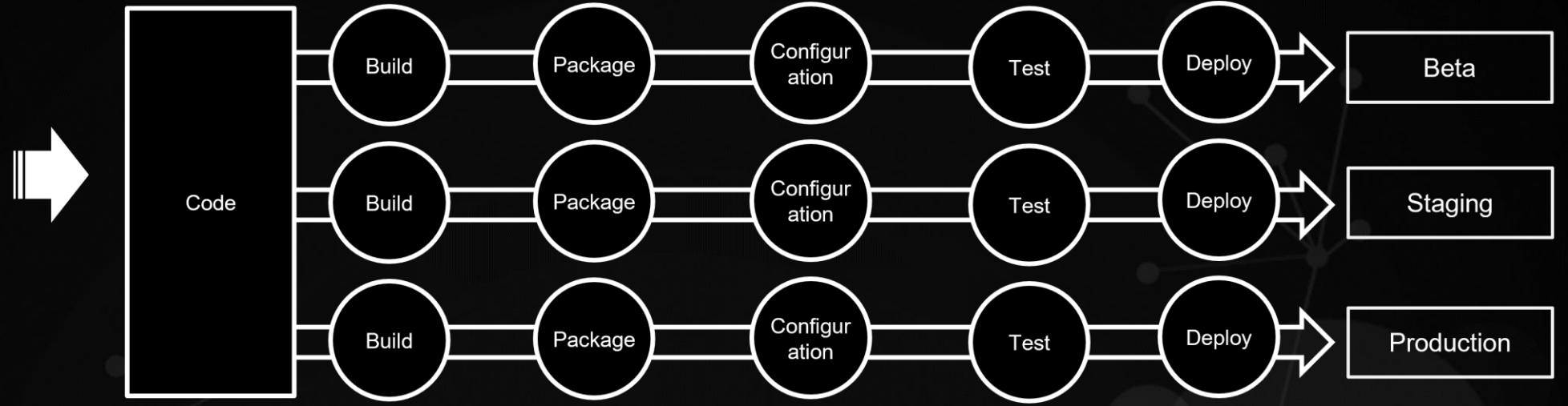


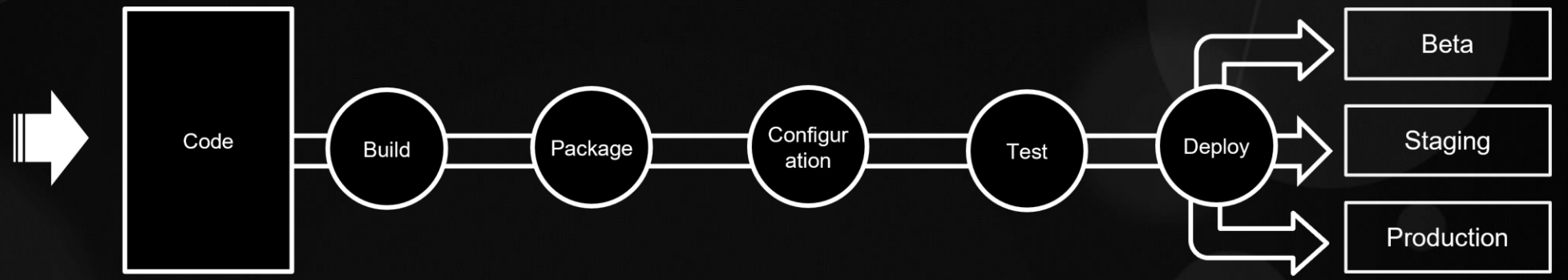
Replace Server Upgrade



Mutable vs. Immutable deployments pipelines





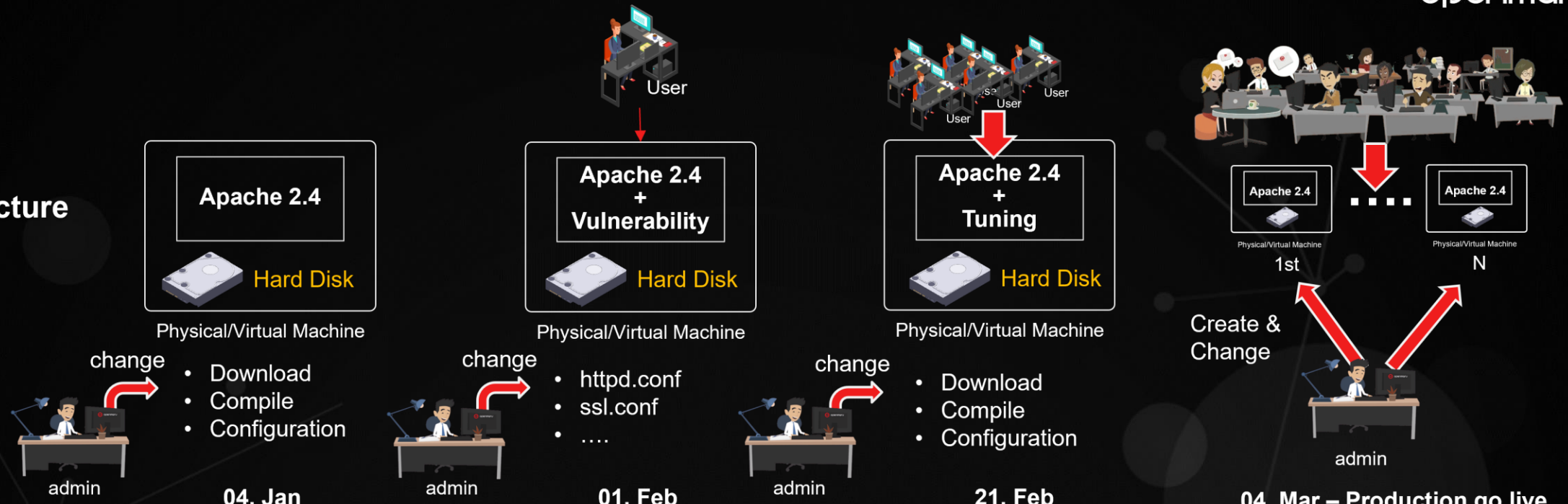


Source : Reliability, consistency, and confidence through immutability - Adrian Hornsby Principal Developer Advocate Amazon Web Services

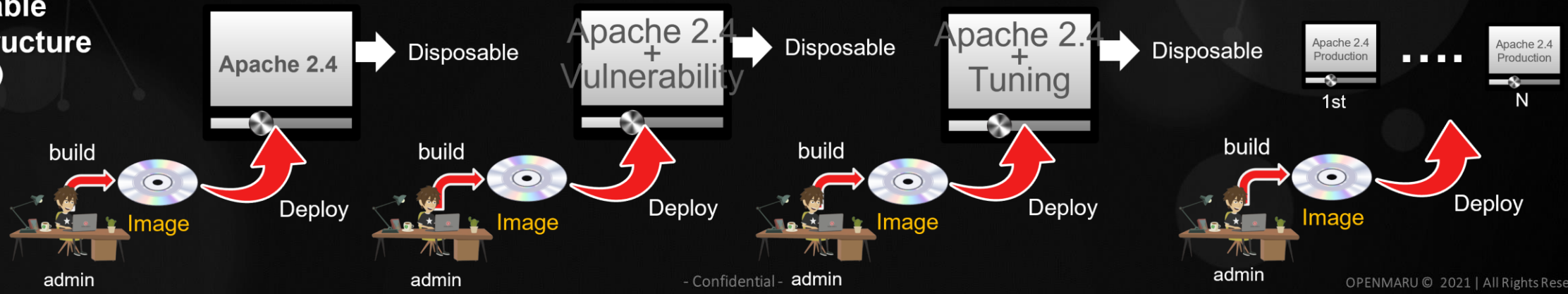
가변 vs. 불변 인프라스트럭쳐 운영 방법 비교

openmaru

Mutable Infrastructure (In-Place)

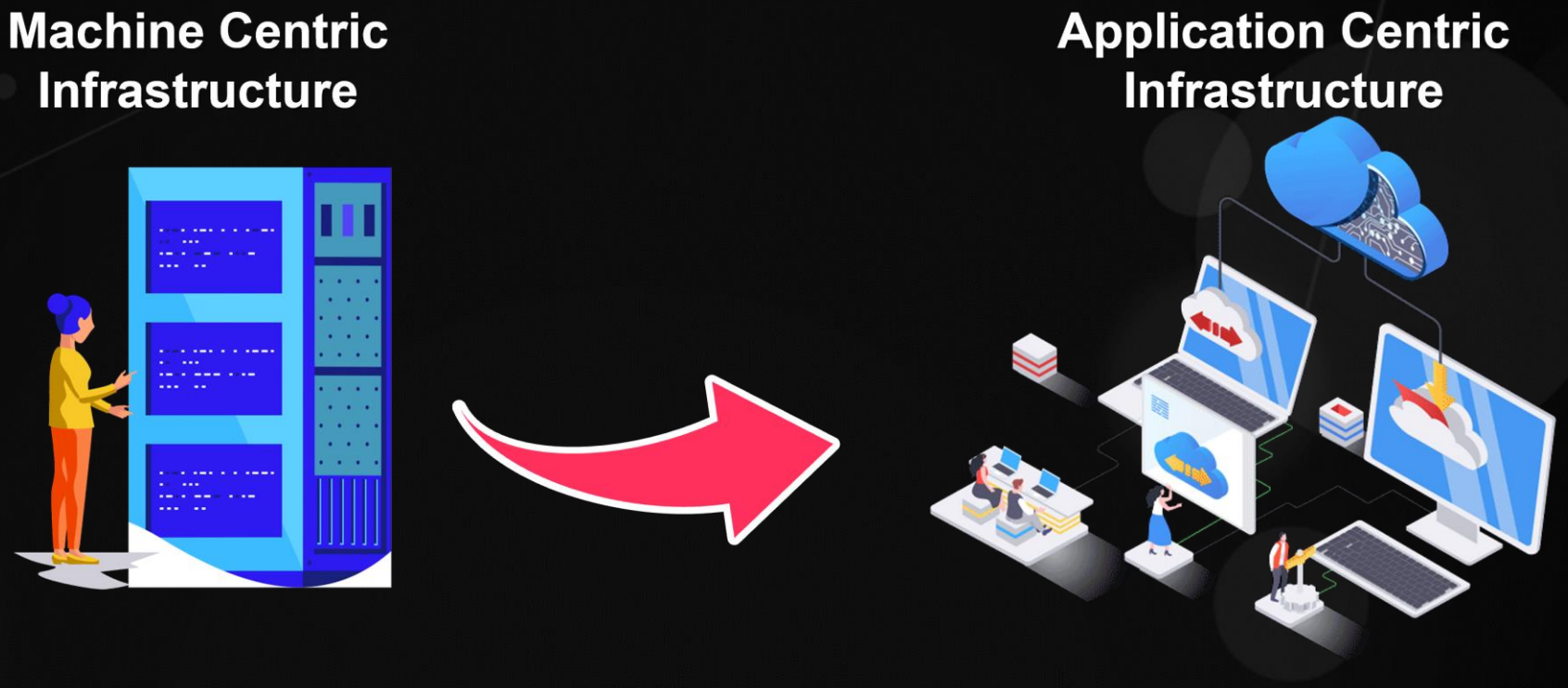


Immutable Infrastructure (Replace)



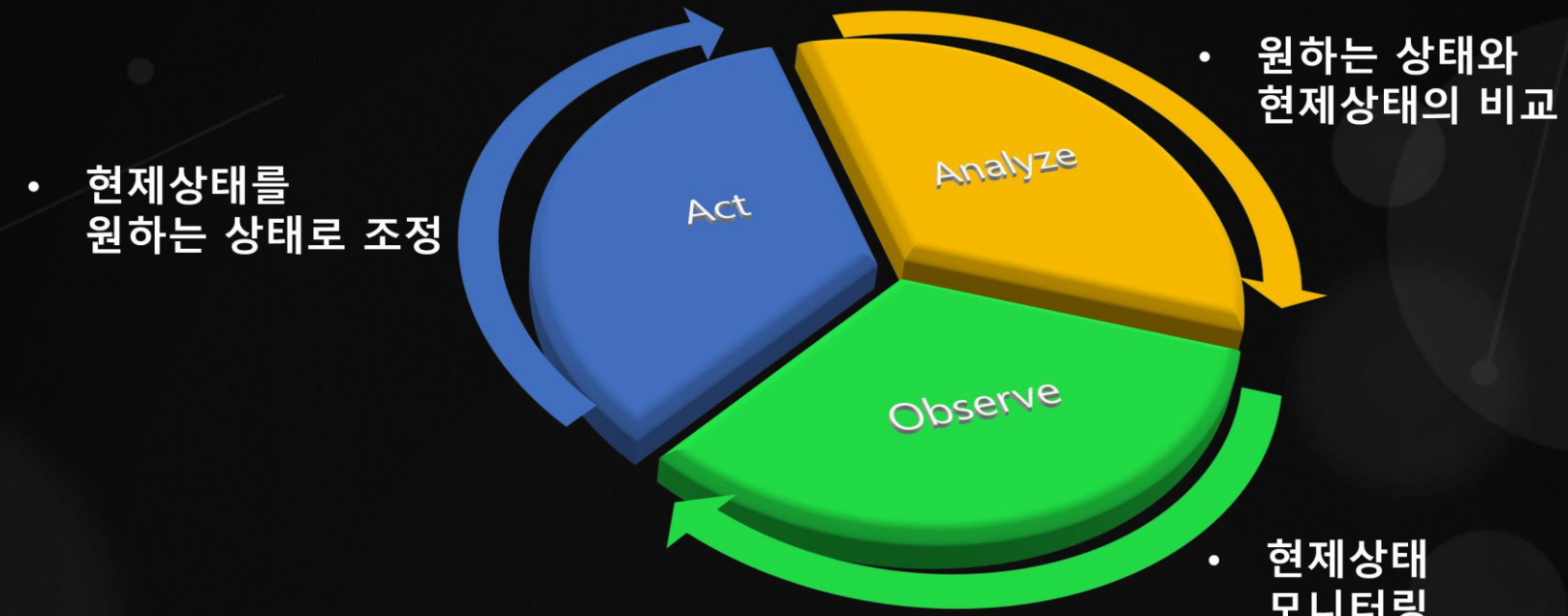
컨테이너 - 애플리케이션 지향 인프라

- 컨테이너화는 데이터 센터를 머신 지향에서 애플리케이션 지향으로 전환
  - 애플리케이션 개발자와 운영팀에게 서버와 운영 체제에 대한 세부 사항을 추상화
  - 실행중인 애플리케이션과 개발자에 미치는 영향을 최소화하면서 새로운 하드웨어 지원과 운영 체제를 업그레이드하여 인프라 팀의 유연성 제공
  - 서버 CPU와 메모리 사용량과 같은 메트릭 정보 뿐만 아니라 애플리케이션에 연결하여 스케일 업, 머신 장애 또는 유지 보수 시에 애플리케이션 모니터링

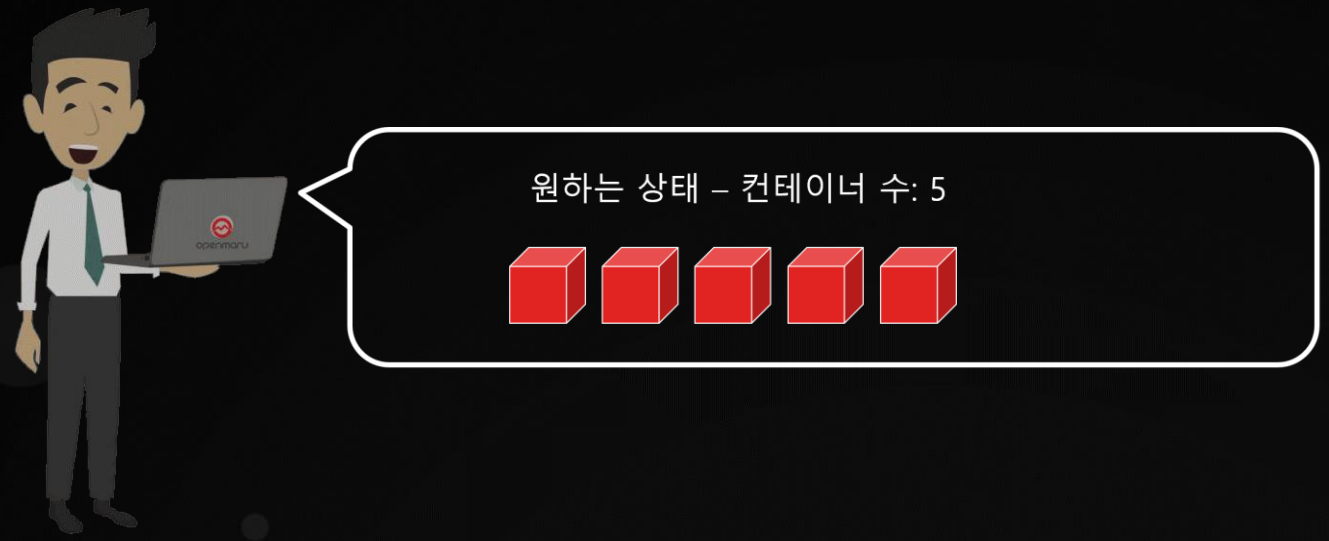


Kubernetes reconciliation model – Control loop

- 쿠버네티스에서 컨트롤러는 클러스터 의 상태를 관찰 한 다음, 필요한 경우에 생성 또는 변경을 요청하는 컨트롤 루프 구조
- 쿠버네티스는 pod 등의 resource 를 관리 할 때 desired state (원하는 상태)과 actual state (현제상태) 를 가지고 있으며, actual state 를 desired state 유지하기 위해 반복함



컨테이너 수를 원하는 상태로 조정하는 방법



현재 상태 확인
(Observe)

현재 상태 – 컨테이너 수: 2

원하는 상태 – 컨테이너 수: 5

원하는 상태와
현재 상태 비교
(Analyze)

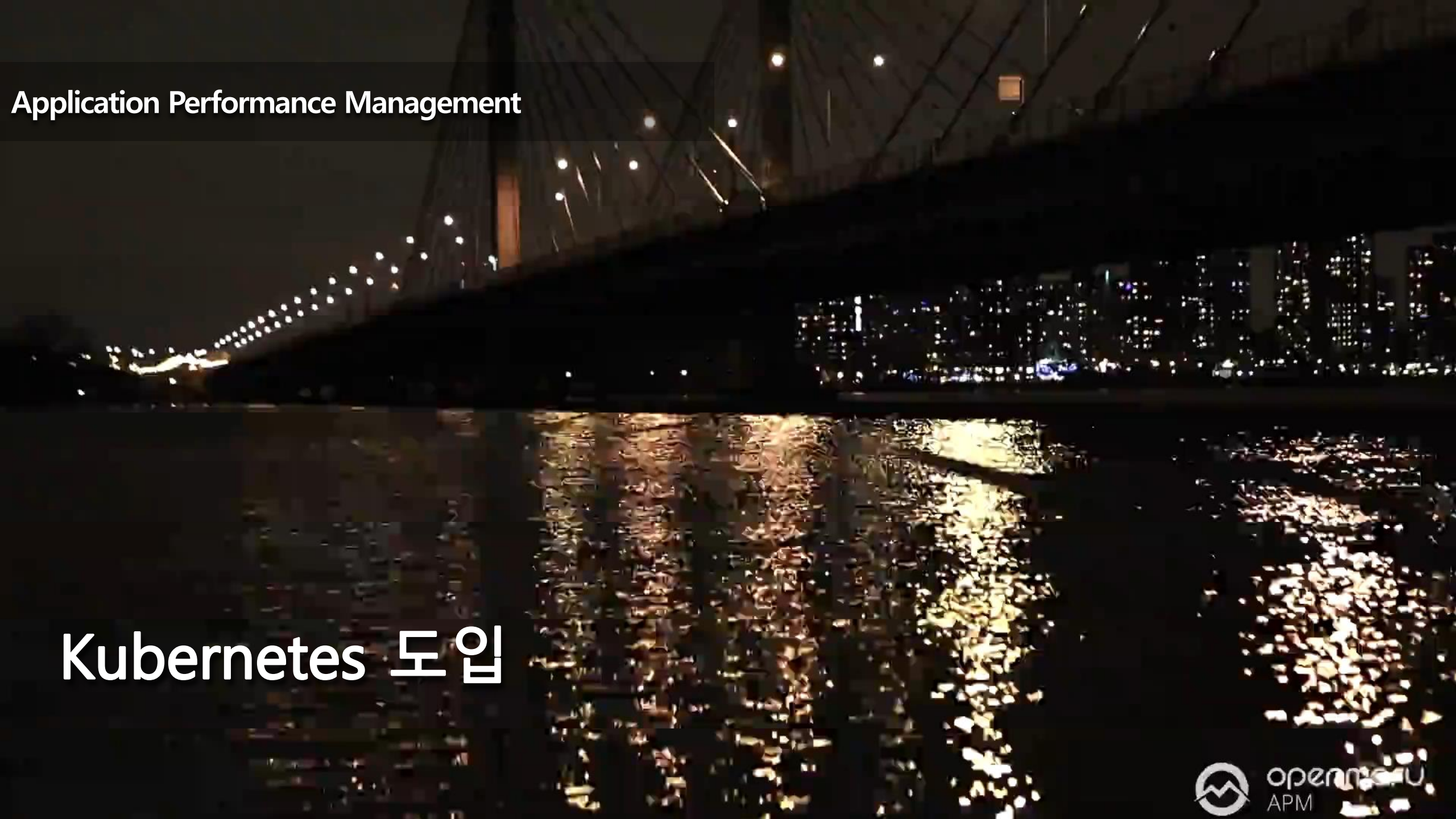
현재 상태 – 컨테이너 수: 2

원하는 상태 – 컨테이너 수: 5

원하는 상태와
현재 상태 일치
(Act)

현재 상태 – 컨테이너 수: 5

원하는 상태 – 컨테이너 수: 5

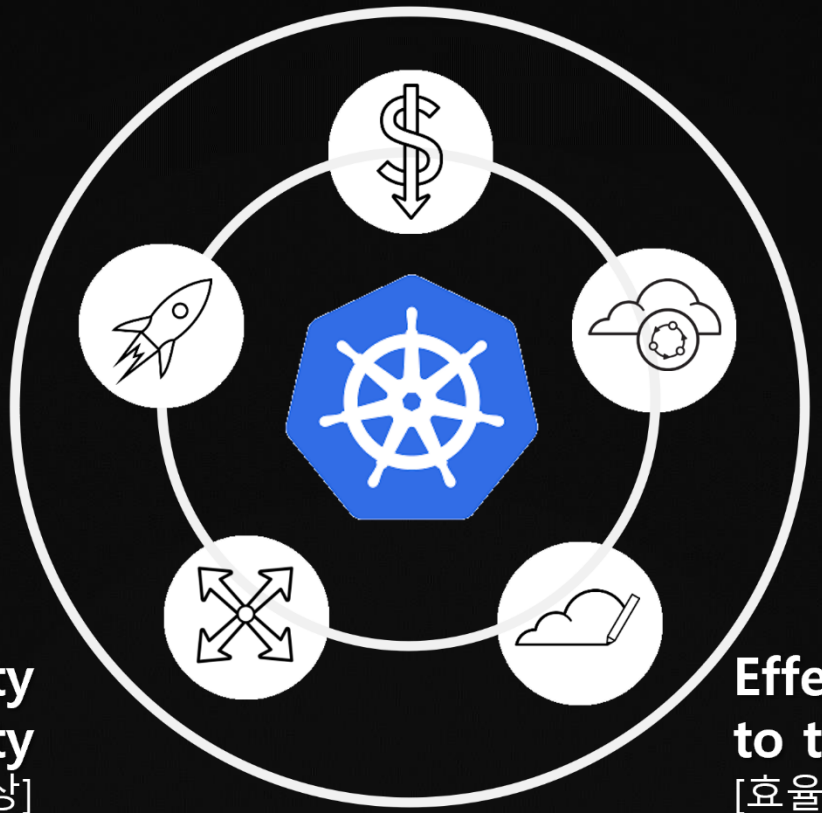

Application Performance Management
Kubernetes 도입
APM

CIO가 Kubernetes 를 고려해야 하는 이유 5가지

openmaru

IT cost optimization
[IT 비용 최적화]

Faster time to market
[신속한 애플리케이션 개발]



Multi-cloud flexibility
[멀티 클라우드에 유연한 지원]

Improved scalability and availability
[확장성과 가용성 향상]

Effective migration to the cloud
[효율적인 클라우드 전환]

Kubernetes 도입 로드맵

openmaru

01 Select an Implementation Target

도입 대상
애플리케이션 식별

02 Choose a Kubernetes Vendor Platform

Kubernetes 플랫폼
벤더 선택

03 Carefully Plan Your Success Criteria

Kubernetes 도입
성공 기준 설정

04 Align with Corporate Culture

기업문화와 융합

1. Select an Implementation Target

- Kubernetes(컨테이너) 특징에 적합한 비즈니스 애플리케이션 식별

- 애플리케이션 또는 프로세스가 느슨하거나 또는 느슨한 구성으로 재설계를 검토 중인 경우
- 애플리케이션이 마이크로 서비스로 개발되어 있거나 또는 마이크로 서비스로 변경하는 경우
- 애플리케이션의 리소스 요구 사항을 충분히 이해
- 애플리케이션에 대한 비즈니스 영향이 측정 가능한 상태

2. Choose a Kubernetes Vendor Platform

- **Kubernetes 관리 및 운영을 핵심 사업으로 하는 공급 업체를 선정**

- 호스팅, 클라우드 기반, 온프레미스 또는 멀티 클라우드 지원 및 스케일링 지원
- 이미 운영 중인 애플리케이션과 상호호환성 및 이용하고있는 서비스와의 연계 운영에 필요한 기능 확인 (RBAC 관리, UI , Active Directory 통합 등)
- 애플리케이션을 컨테이너 환경에서 운영하는 것을 전제로한 비용 및 지급 모델 타당성 검토

3. Carefully Plan your Success Criteria

- **Kubernetes 도입의 전반적인 성공을 결정하는 메트릭을 정의하고 사전에 충분한 이해**

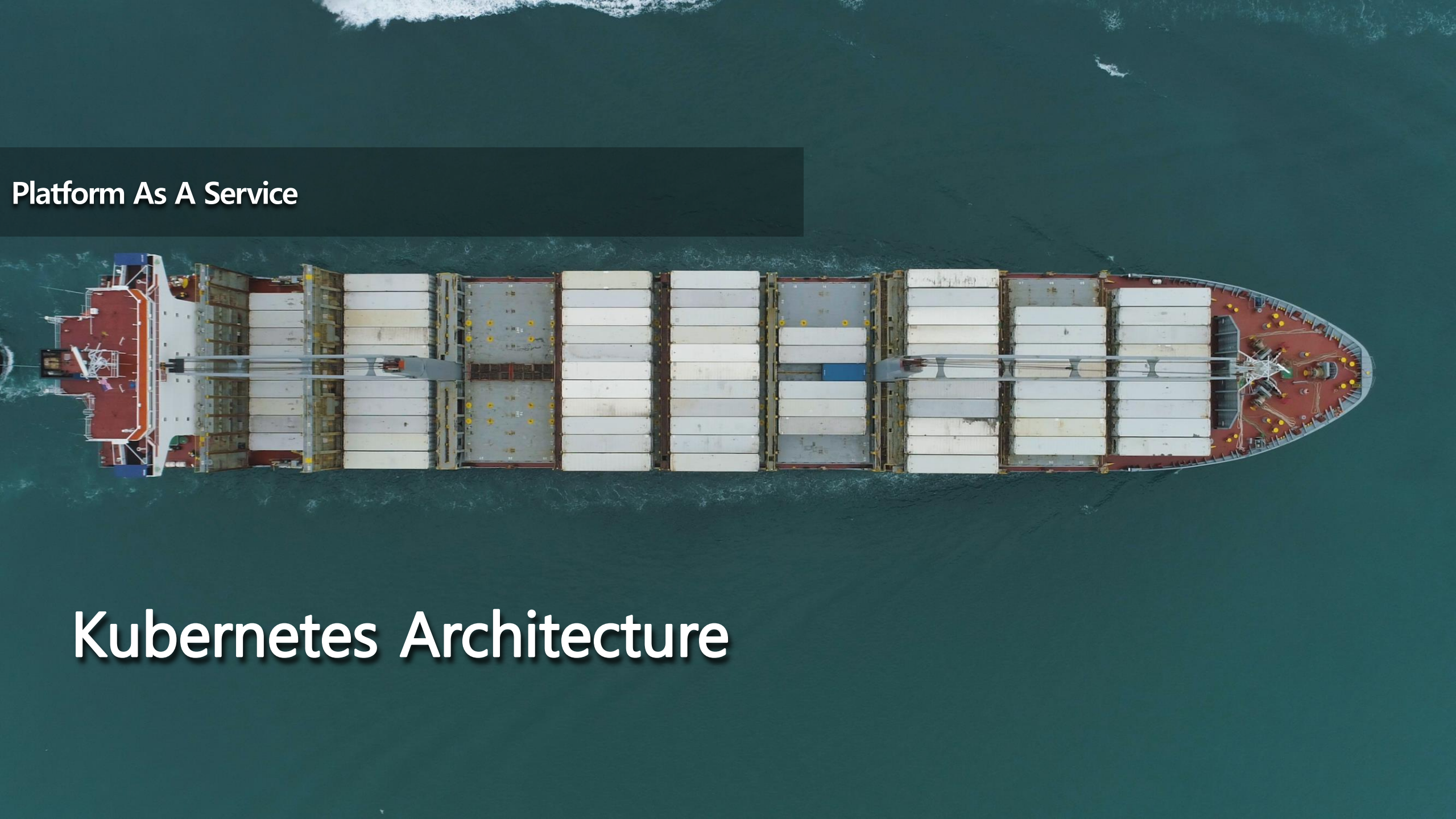
- 서비스의 성능이 기존 애플리케이션과 동일한 지와 차이점이 어떤 부분에서 있는지 확인
- 서비스 및 기능을 개선하기 위해 릴리즈 주기를 단축 할 수 딜리버리 파이프 라인을 표준화 하여 전반적인 품질을 개선
- 전체적인 안정성과 유연성 개선
- 오케스트레이션 된 애플리케이션의 예측할 수 없는 부하에 대한 전반적인 성능과 안정성 개선

4. Align with Corporate Culture

- 기업의 비즈니스 요구에 맞게 확장 할 수 클라우드 네이티브 플랫폼 구축

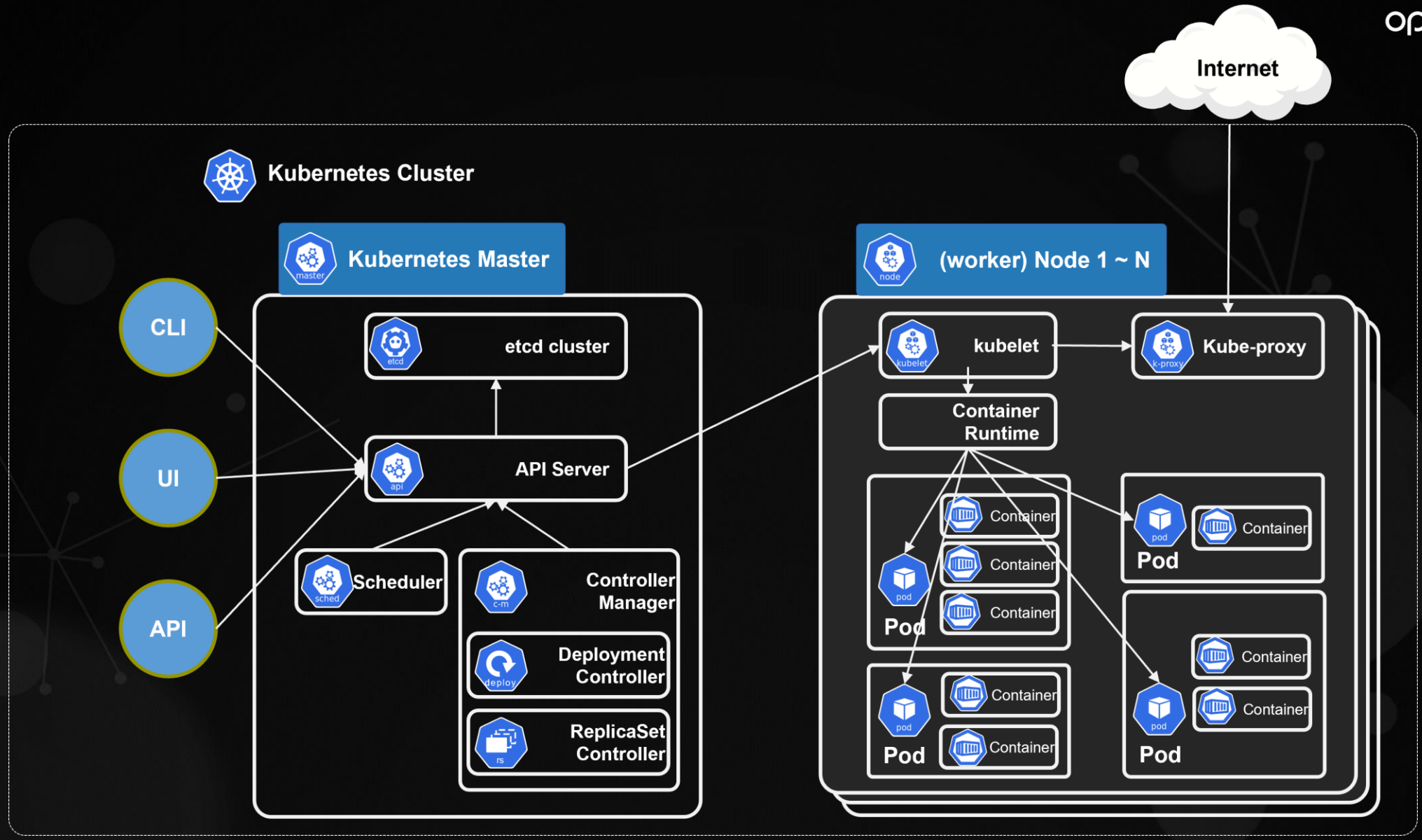
- 새로운 도구 사용과 변화에 대하여 잘 대응할 수 있는 방법을 찾아 가야함.
- 애플리케이션 배포와 릴리스 방법의 변화에 대한 자동화 추진

Platform As A Service

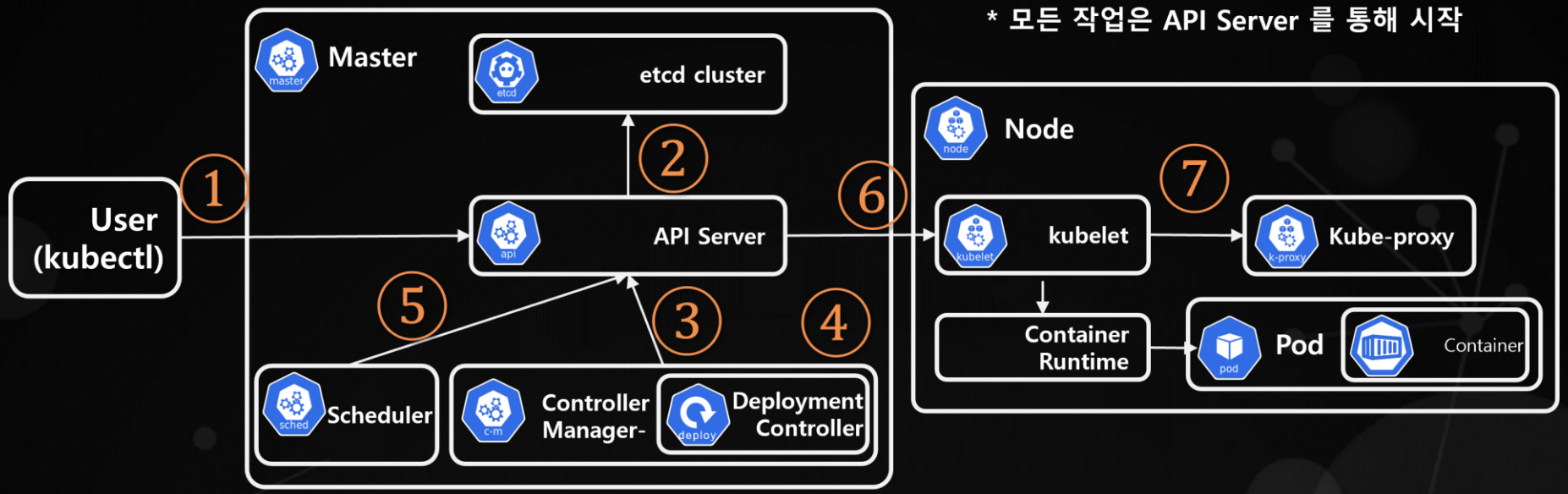


Kubernetes Architecture

Kubernetes Architecture



Kubernetes Architecture



① 사용자는 "kubectl"를 사용하여 컨테이너 배치에 필요한 정보 (컨테이너 이미지 개수 등)를 전달

② API Server는 받은 내용을 etcd cluster (DB)에 저장

③ Controller는 자원의 변화를 감지하면 변경 내용에 상태가 되도록 실행

④ 배포 일 경우 Deployment Controller는 새로운 Pod 정보를 API Server를 통해 DB (etcd)에 저장

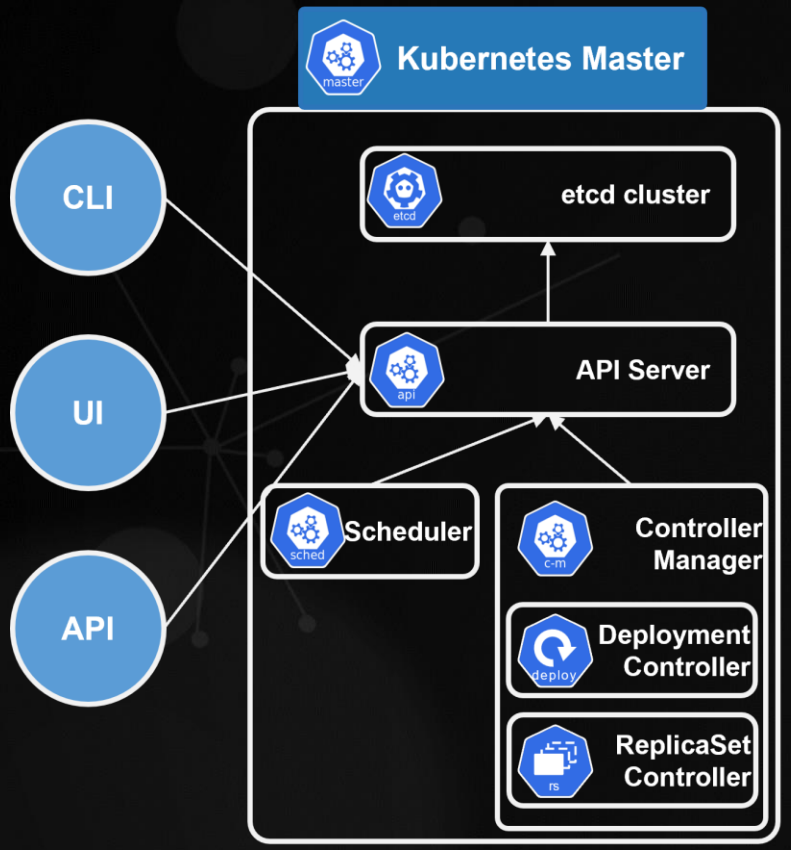
⑤ Scheduler는 정보에 따라 해당 Node 결정

⑥ Kublet 는 Container Runtime 를 사용하여 Pod를 생성

⑦ Kube-Proxy는 클러스터의 내부 또는 외부에서 Pod에 대한 연결을 포워딩

Kubernetes Master

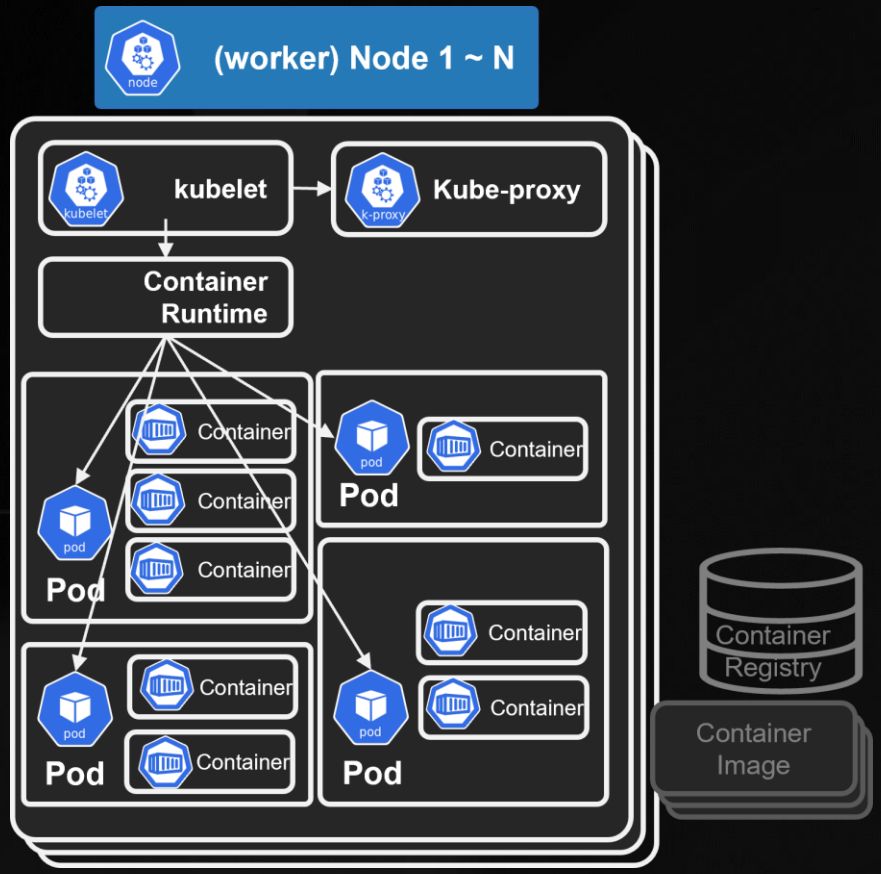
- 마스터는 컨테이너를 관리하는 역할로 "컨트롤 플레인" 이라고 불림
- Master는 각 Node 에 배포, 업데이트, 스케일링 등의 지시를 내리는 역할
- 3개 이상의 서버에서 마스터를 구성 할 것을 권장



컴포넌트	설명
API Server	• 쿠버네티스 클러스터의 모든 작업을 제어하는 RESTful API • 쿠버네티스 컨트롤 플레인에 대한 프론트엔드 • API Server 운영자 및 내부 노드와 통신하기 위한 인터페이스 / HTTP(S) RestAPI로 노출되어 있고 모든 명령은 여기를 통함
Scheduler	• 서비스를 리소스 상황에 맞게 적절한 노드에 배치하는 역할 / predicates와 priorities (LeastRequestedPriority, BalancedResourceAllocation, • ServiceSpreadingPriority, EqualPriority) 설정 • 노드가 배정되지 않은 새로 생성된 Pod를 감지하고 그것이 어느 Node에서 실행될 지를 선택 • Pod 를 어떻게 Node 에 배치할지를 예약
Controller-Manager	• 다양한 컨트롤러 (복제/배포/상태/크론/..)를 관리하고 API Server와 통 신하여 작업을 수행 • 백그라운드에서 클러스터 컨트롤러를 관리 . API 서버를 이용하여 클러스터 상태 모니터링 • 클러스터를 설계한 상태로 유지
etcd	• 가볍고 빠른 분산형 key-value 저장소 / 설정 및 상태를 저장 • 클러스터의 모든 정보를 저장하는 데이터 저장소 • 고가용성을 가진 Key-Value 저장소 구조

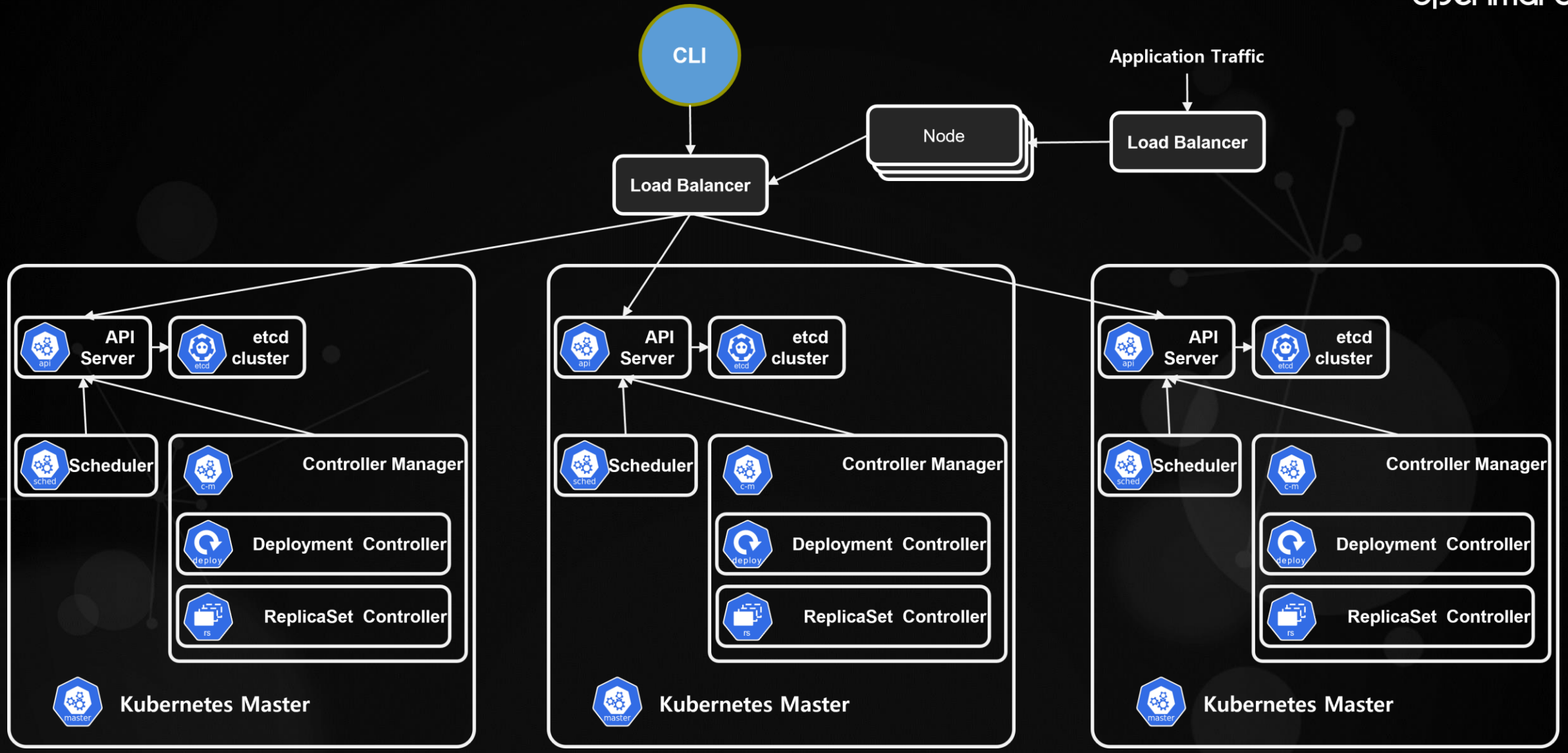
Kubernetes (worker) Node

- 노드는 마스터의 지시로 컨테이너를 실행하는 역할을 하며 “데이터 플레인” 이라 불림
- 노드가 마스터뿐만 아니라 최소 1 개로 구성 할 수 있지만 Kubernetes는 2 대 이상의 서버 노드를 구성 할 것을 권장 하고 있습니다.

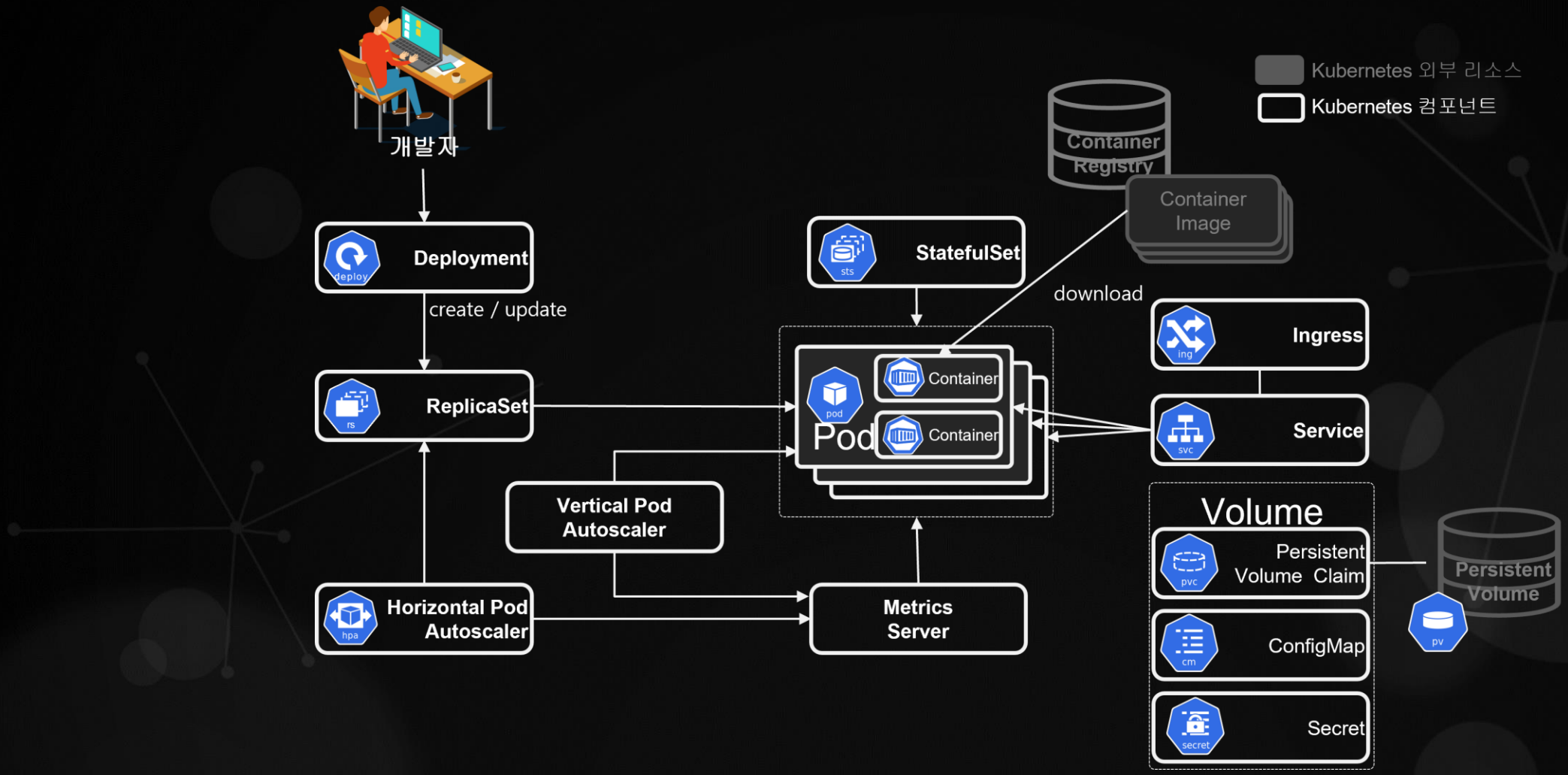


컴포넌트	설명
kubelet	• 서비스(컨테이너)를 실행/중지하고 상태를 체크하여 계속해서 살아있는 상태로 관리 • 노드에서 실행되는 에이전트로 컨테이너가 Pod에서 실행 중인지 확인 • kubelet은 다양한 메커니즘을 통해 제공되는 PodSpec을 가져와 해당 요구사항으로 상태를 유지
pod	• container 의 그룹 k8s 의 최저 배포 단위 • Pod 중 컨테이너 Storage, namespaces , port 공유
kube-proxy	• 네트워크 프록시와 로드 발란서 역할 (creates a iptable rule/...) • kube-proxy는 호스트 상에서 네트워크 규칙을 유지하고 연결에 대한 포워딩을 수행함 • 쿠버네티스 서비스를 추상화
Container runtime	• 작성된 이미지를 가져와 컨테이너를 실행 • Kubernetes 는 컨테이너 런타임 교체가 가능하기 때문에, Container , containerd , rkt , cri-o 같은 컨테이너 런타임을 사용

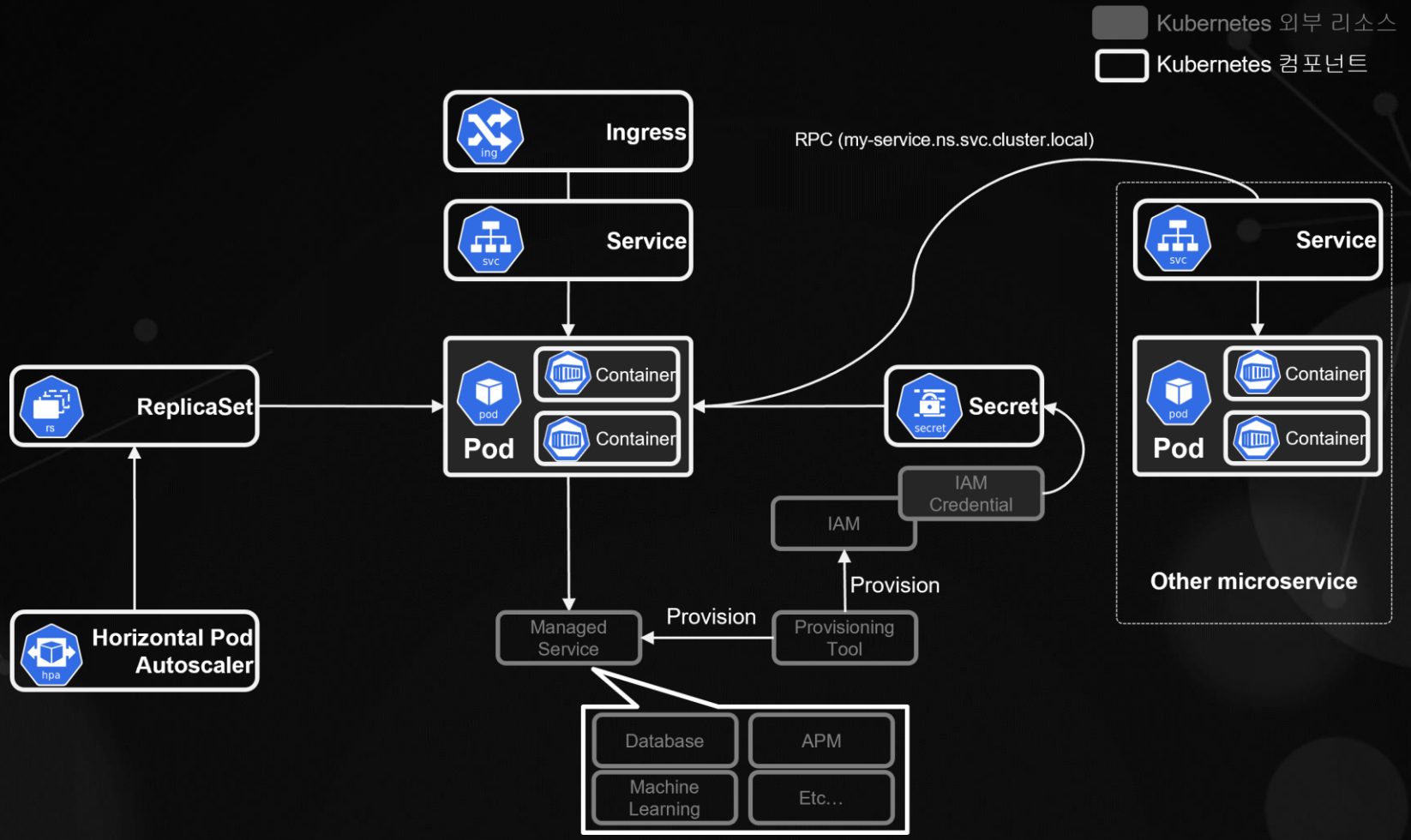
Kubernetes Master 고가용 구성



Kubernetes – 배포 프로세스

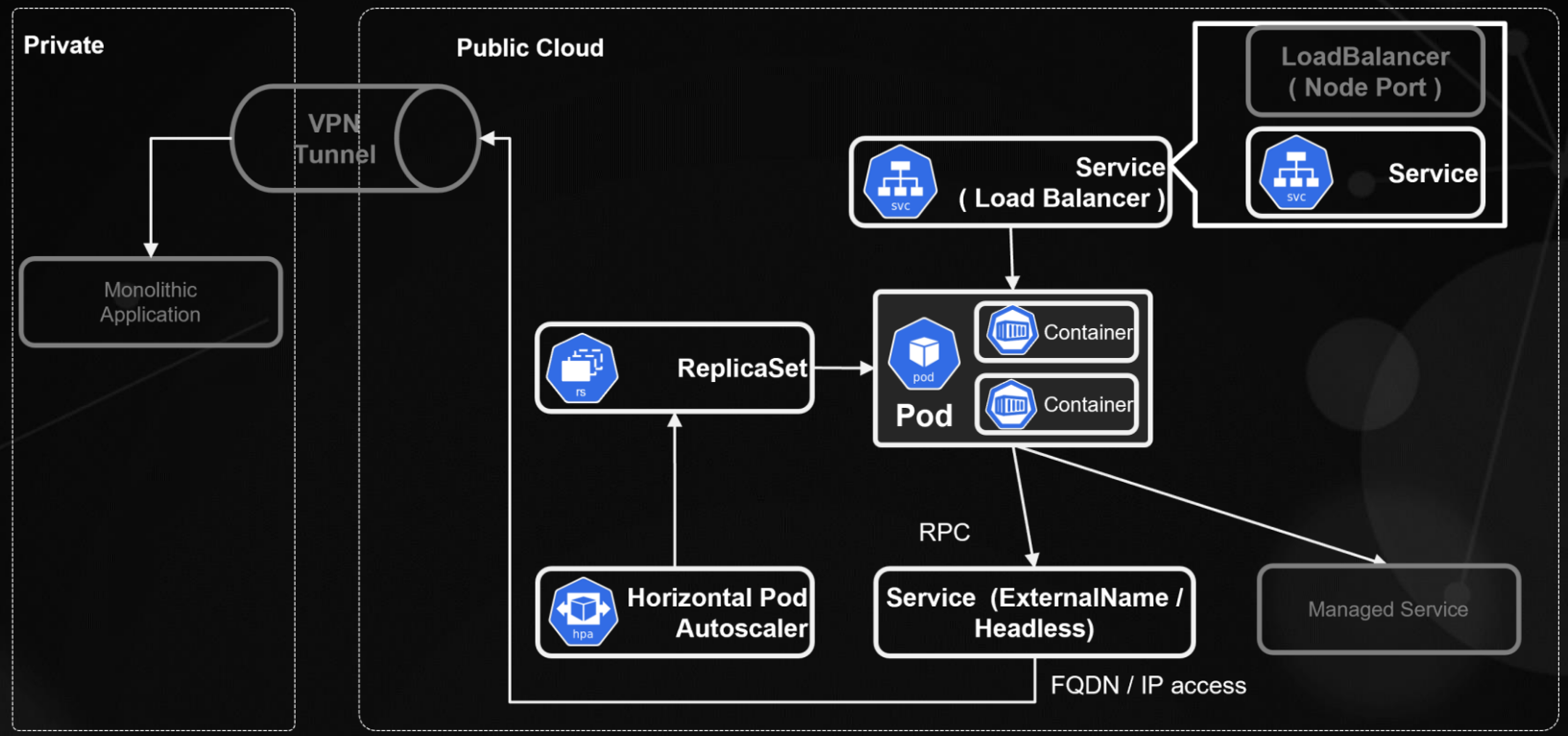


Kubernetes – 서비스 호출 프로세스



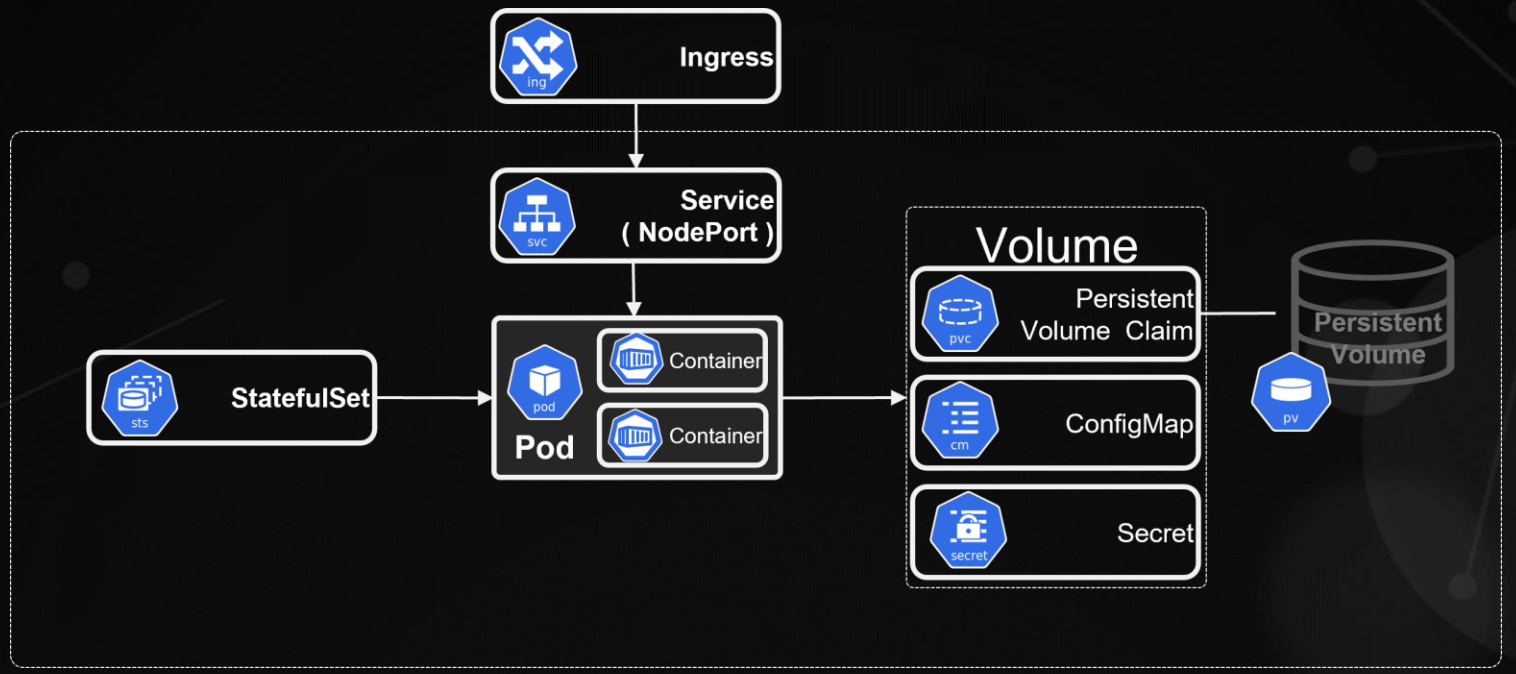
Kubernetes – 하이브리드 클라우드 예시

- 기존 시스템 (Monolithic) 에서 Agility를 확보하기 위해 서비스를 클라우드에 배포



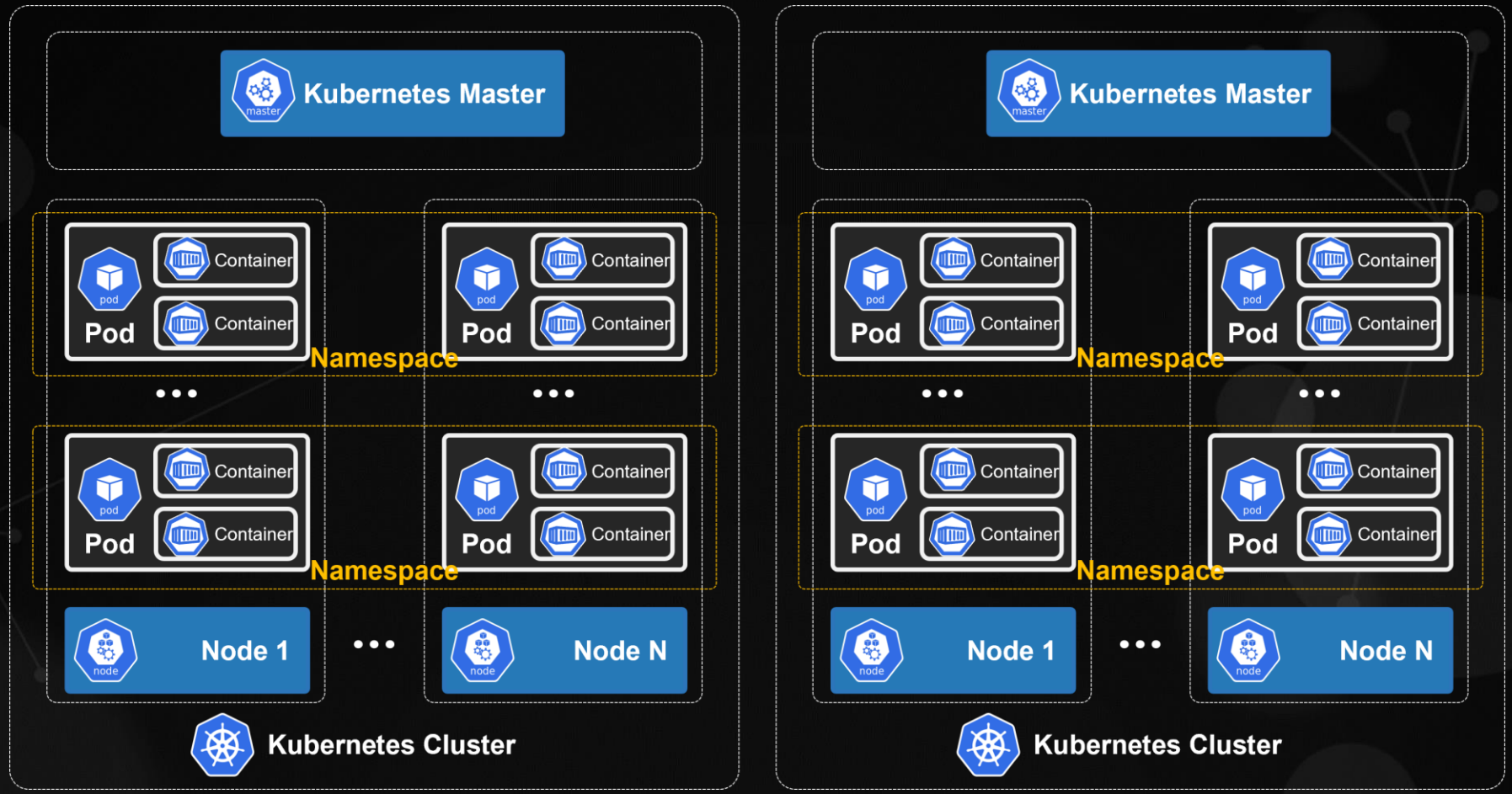
Kubernetes – Application 구성 단위

- 애플리케이션 구성 단위



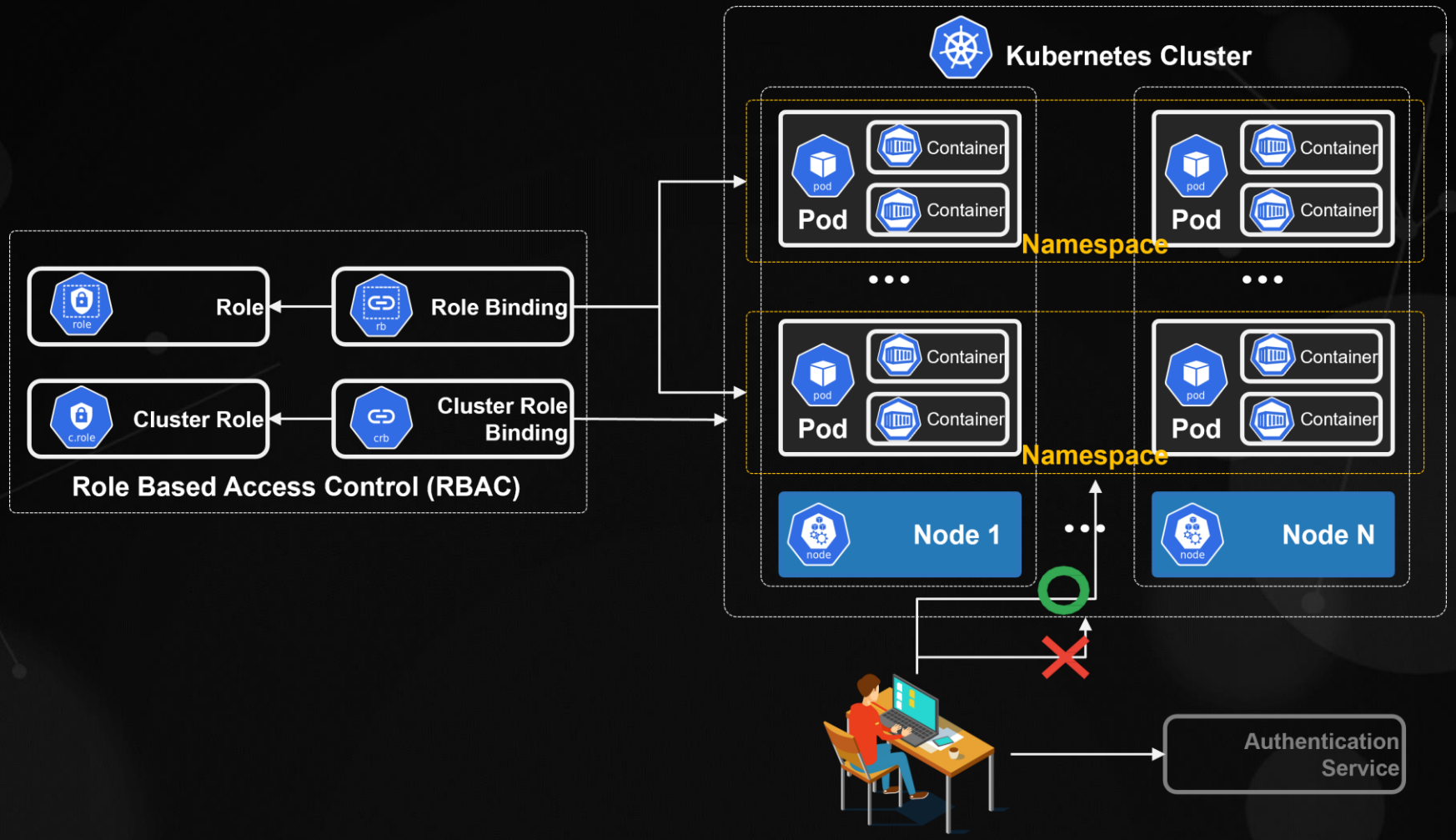
Kubernetes – Master/Node

- Kubernetes Cluster 단위



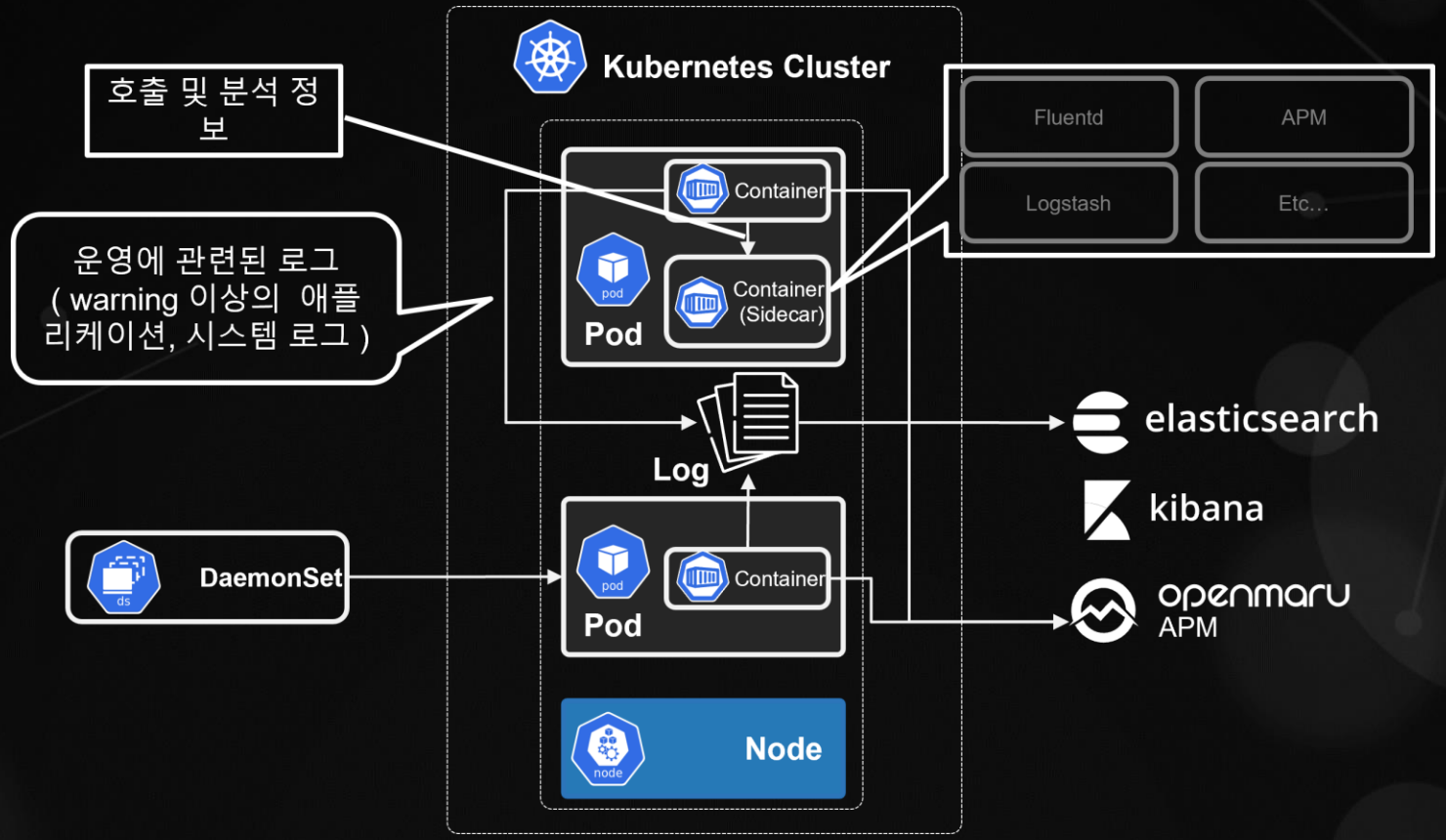
Kubernetes – 권한 관리

- Role Based Access Control (RBAC)

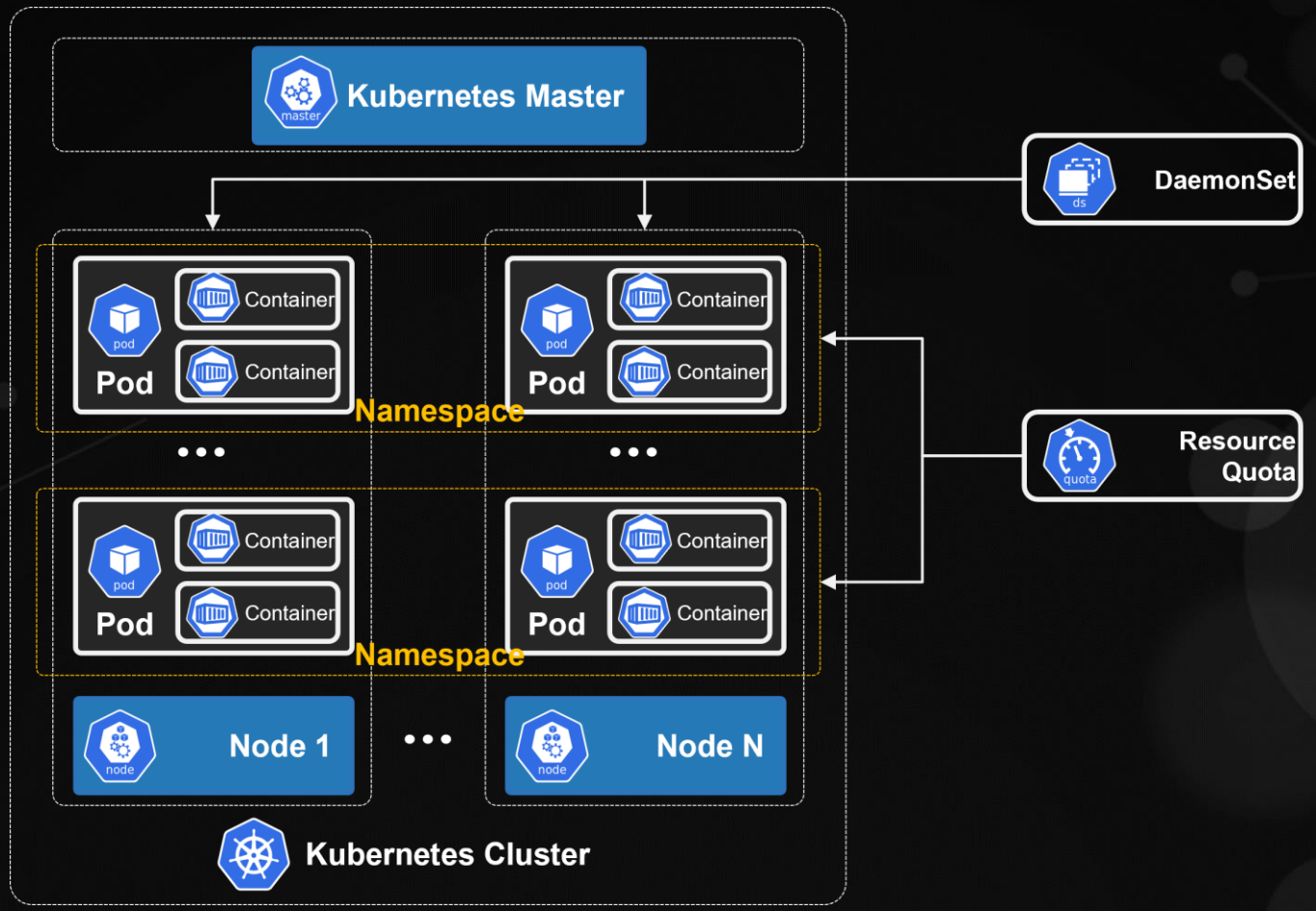


Kurnetes – Log 통합과 모니터링

- 간단한 설정을 통한 로그 통합 및 모니터링
- 로그의 과다 전송으로 인한 시스템 장애를 방지하고 안정적인 운영을 실시한다.

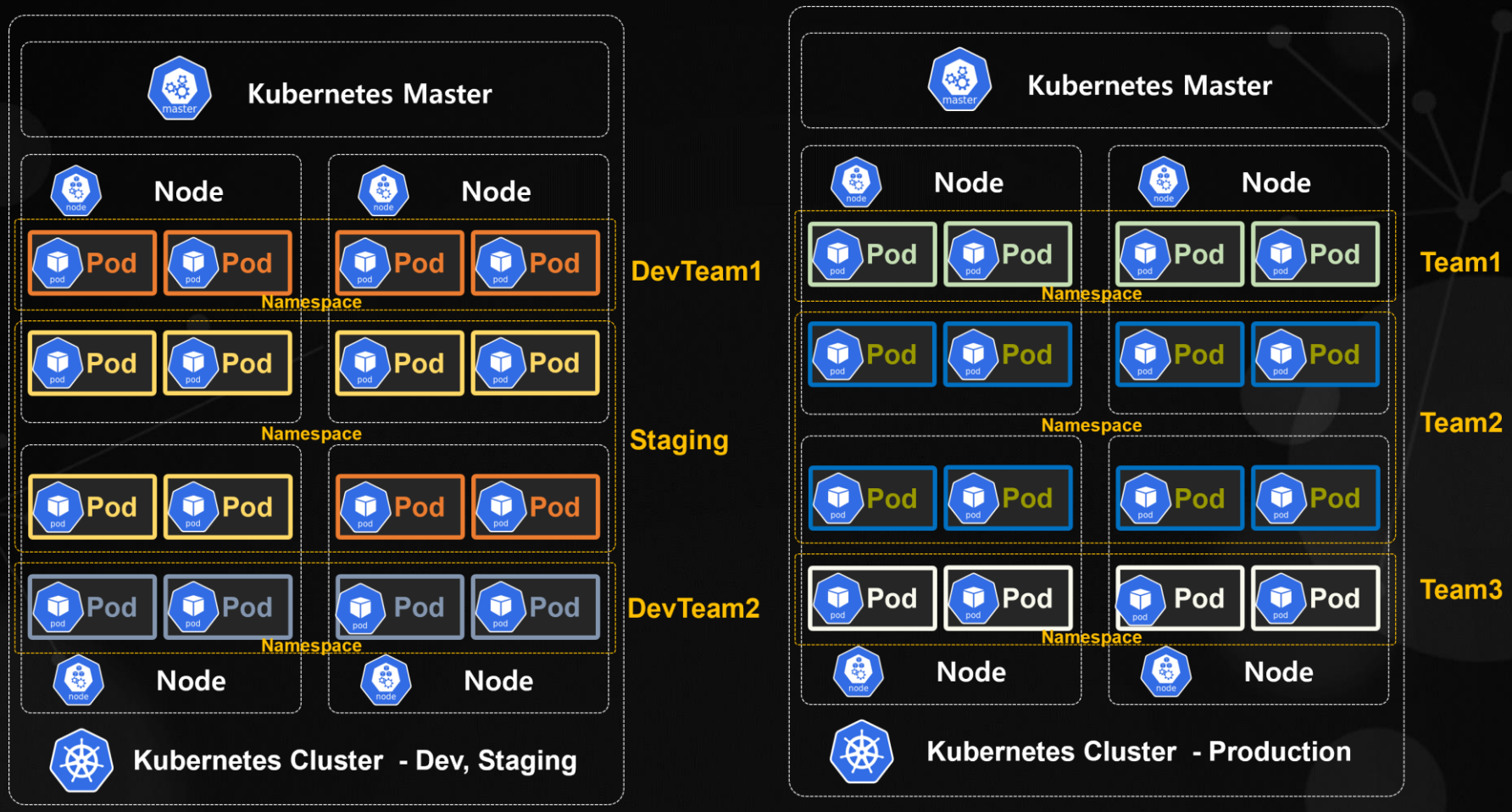


Kubernetes - Quota



Kubernetes – 업무별 구분

- Kubernetes Cluster 단위




openmaru

Application Performance Management

Kubernetes

openmaru APM