컨테이너와 마이크로서비스를 통한

클라우드 네이티브 구현 전략

윤석찬

아마존웹서비스 테크에반젤리스트

오늘의 주제

1. 애플리케이션 로직 진화와
   마이크로서비스 아키텍처의 출현

2. 컨테이너 기반 클라우드 네이티브
   마이크로서비스 구축 패턴

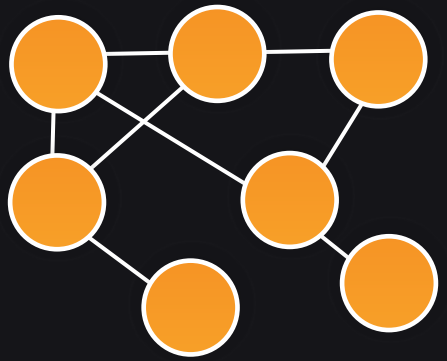
3. 컨테이너 기반 마이크로서비스
   삼성전자 및 폭스바겐 구축 사례

1. 애플리케이션 로직 진화와
마이크로서비스 아키텍처의 출현

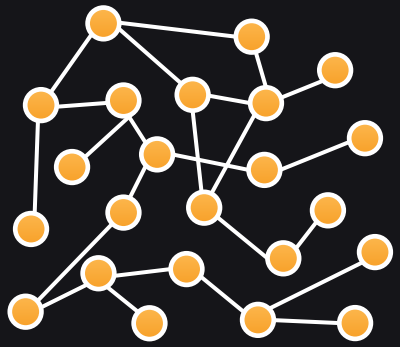
애플리케이션 비즈니스 로직의 진화

Monolith

Microservices

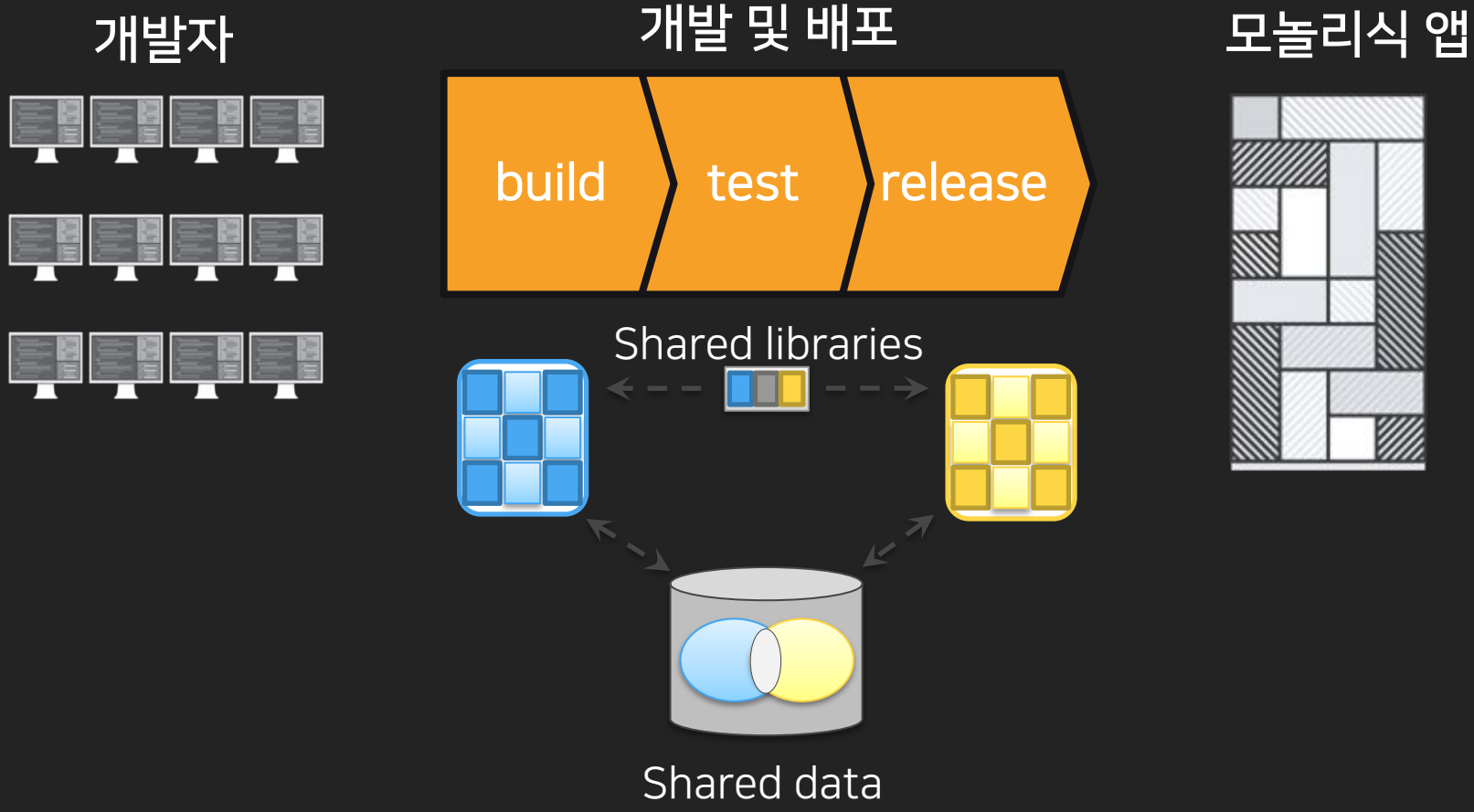


Functions

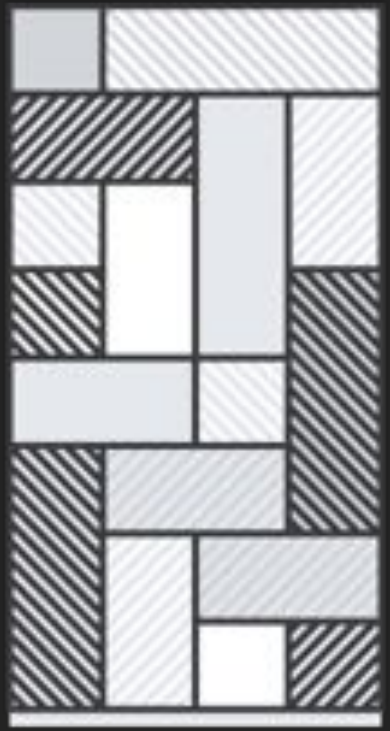


애플리케이션 성장에 따른 **민첩한 배포 및 관리 시간**의 변화

전통적인 “Monolith 앱” 개발 배포 및 아키텍처

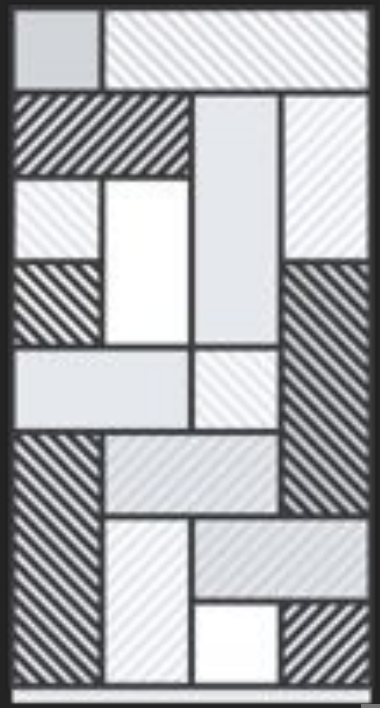


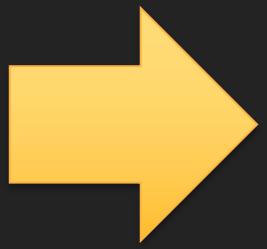
성장하는 조직 및 서비스에서 “Monolith 앱”의 단점



애플리케이션 확장성 애로	아키텍처 유지 진화의 어려움	**민첩성 저해**
장기 개발 사이클 (다수개발자 공동 참여)	신규 출시에 몇 달이 걸림	**혁신 저해**
운영의 어려움 (특정 모듈 장애시)	신규 기능 추가에 어려움	**고객 불만족**

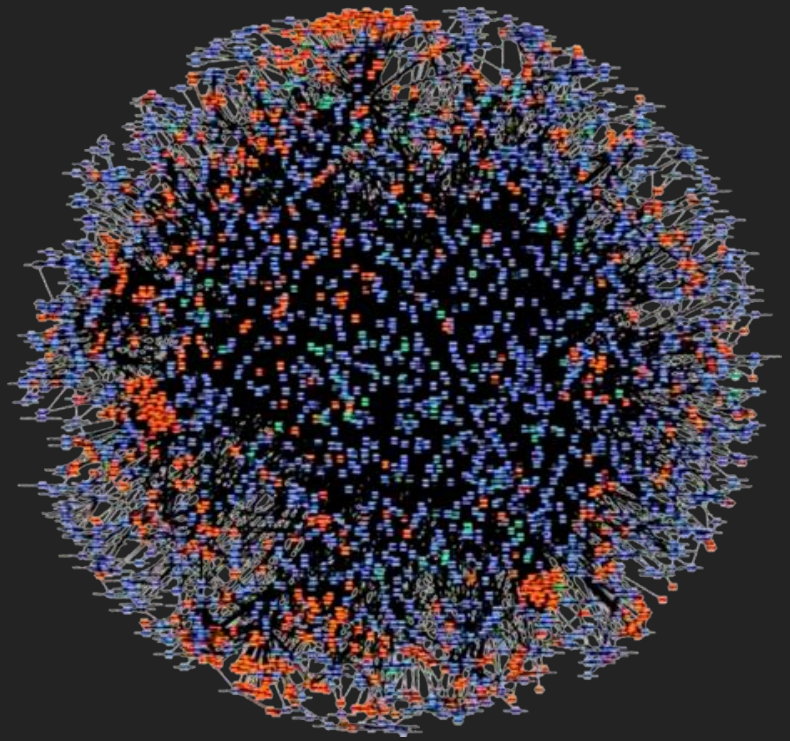
마이크로서비스로의 전이





많은 기업들이 마이크로 서비스
아키텍처를 선택하고 있습니다!

Amazon.com의 사례



수 천개 팀 (자율적 DevOps팀)

× 마이크로서비스 아키텍처

× 지속적 배포 (CD)

× 다양한 개발 환경

= 연간 5천만회 배포

(시간당 5708 회, 또는 0.63 초)

(c) Werner Vogels, The Story of Apollo - Amazon's Deployment Engine

http://www.allthingsdistributed.com/2014/11/apollo-amazon-deployment-engine.html

아마존은 수 많은 스타트업의 조합이다

NETFLIX

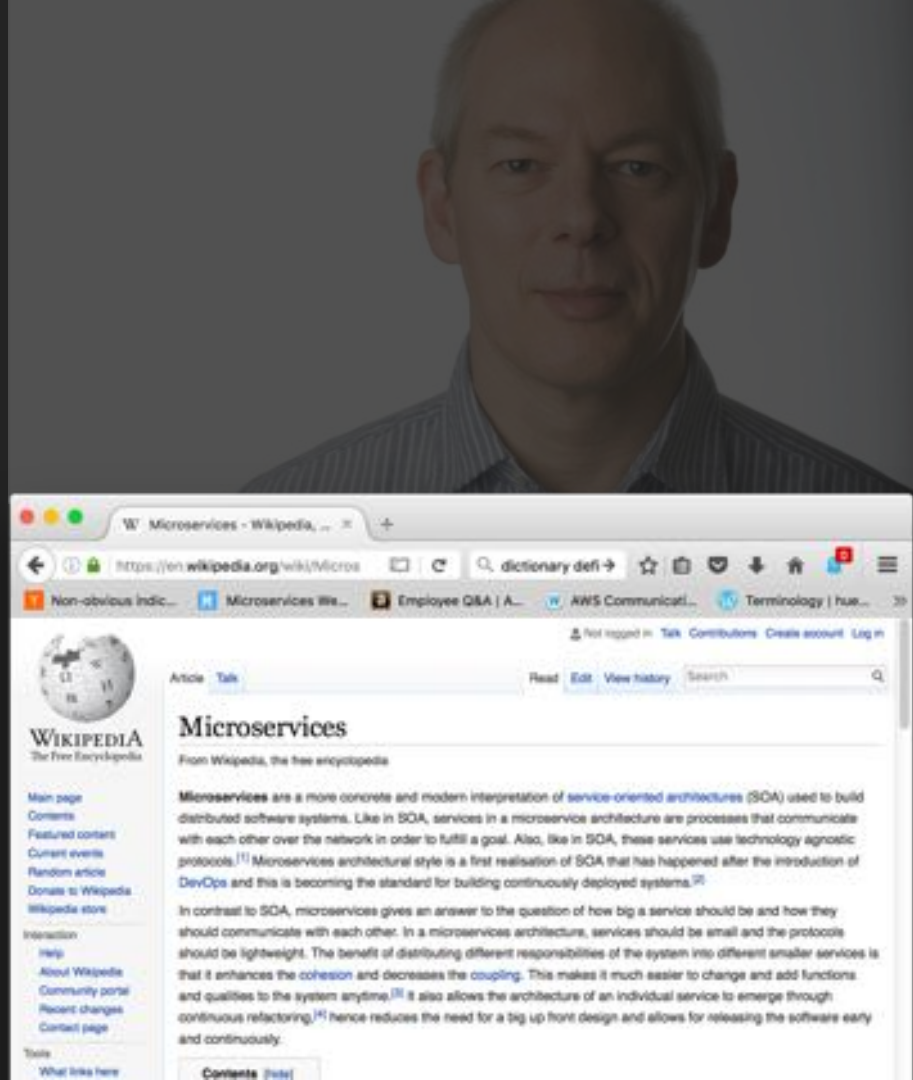
"넷플릭스는 수백개의 **마이크로서비스**를 AWS 클라우드 기반으로 운영하고 있는 것으로 유명하다. 또한, 인터널 API를 기반으로 가벼운 REST 프로토콜을 활용하여 서비스 통신을 하고 있으며, Netflix Internal Web Service Framework(NIWS) 그리고 이러한 다양한 서비스를 발견하기 위한 목록 관리를 위한 Eureka, 서비스간 유연한 소통을 위한 Ribbon 등 클라우드 내 서비스 운영을 위한 수십개의 오픈 소스 프로젝트를 http://netflix.github.io/ 에 공개하고 있다."

http://techblog.netflix.com/2013/01/announcing-ribbon-tying-netflix-mid.html

마이크로 서비스란?

"service-oriented architecture composed of loosely coupled elements that have bounded contexts"

Adrian Cockcroft (VP of Cloud Architecture @ AWS, former Cloud Architect at Netflix)



"service-oriented architecture composed of loosely coupled elements that have bounded contexts"

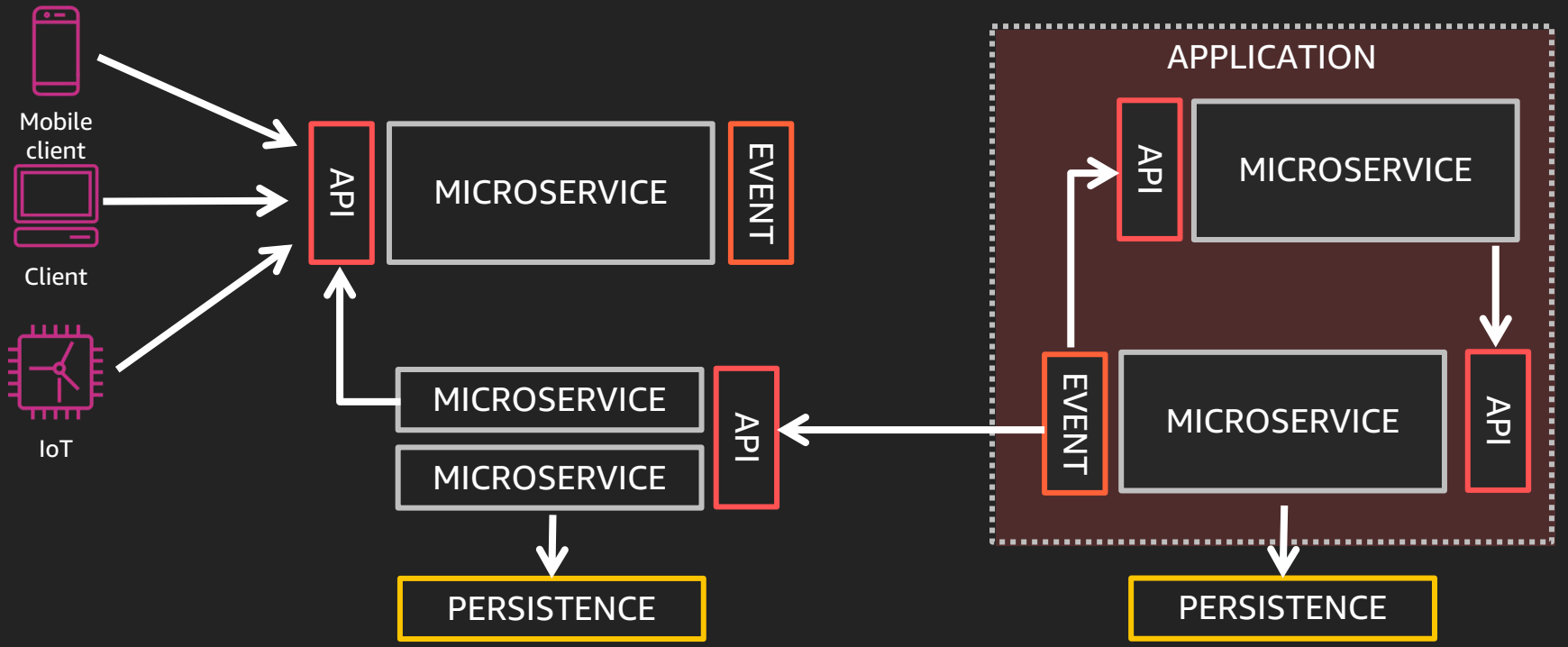
Adrian Cockcroft (VP of Cloud Architecture @ AWS, former Cloud Architect at Netflix)

서비스들이 네트워크를 통해 서로 API로 통신한다.

서비스는 독자적으로 업데이트하며, 서로 영향을 주지 않는다.

다른 서비스의 내부 구조를 알지 못해도, 내 서비스 코드를 업데이트 할 수 있다.

마이크로 서비스(Microservices) 아키텍처



AWS 기반 마이크로 서비스 아키텍처 사례

Nike's Journey into Microservices

Developing Mobile Apps and Serverless Microservices for Enterprises using AWS

Building a Microservices Gaming Platform for Turbine Mobile Games

마이크로서비스 기반 모바일 서비스 마이그레이션

삼성전자 IoT 서비스 마이크로서비스 구축 사례

A Journey to Microservices

Pure Play Video OTT- A Microservices Architecture

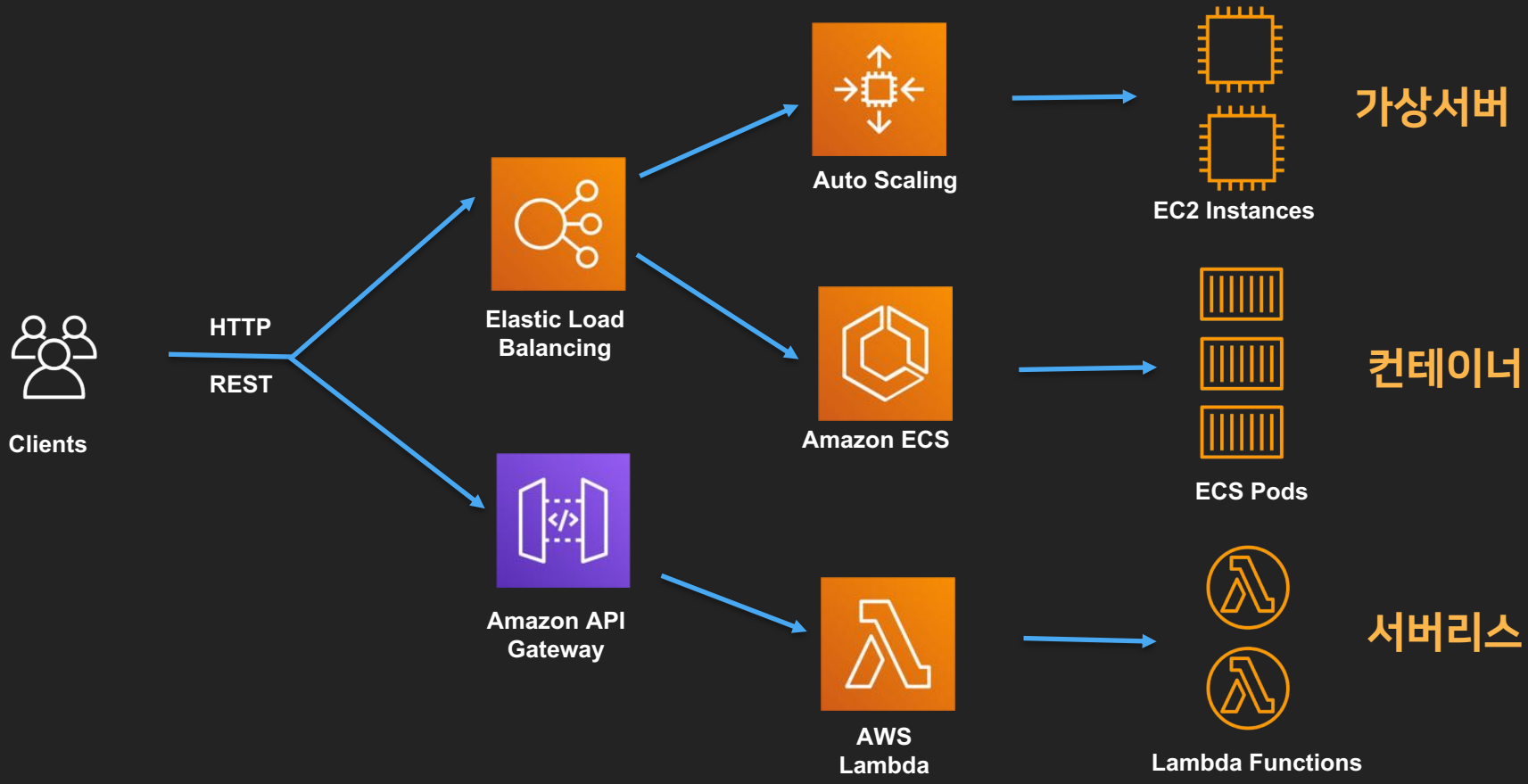
From Monolithic to Microservices Evolving Architecture Patterns in the Cloud

마이크로 서비스 아키텍처로 방송 서비스 진화

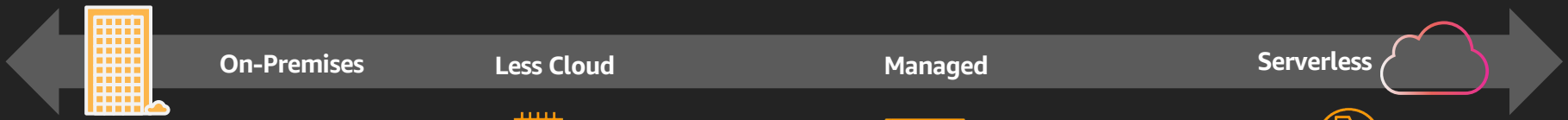
서버리스 마이크로 서비스 전환 사례

2. 컨테이너 기반 클라우드 네이티브
마이크로서비스 구축 패턴

클라우드 기반 마이크로서비스 아키텍처 패턴



클라우드 기반 마이크로서비스 아키텍처 패턴



Kubernates	Kubernetes on EC2	EKS	Lambda
애플리케이션 코드	애플리케이션 코드	애플리케이션 코드	애플리케이션 코드
코드 배포 관리	코드 배포 관리	코드 배포 관리	확장/증설 관리
고 가용성 구성	고 가용성 구성	고 가용성 구성	고 가용성 구성
보안 및 네트워크 관리	보안 및 네트워크 관리	보안 및 네트워크 관리	보안 및 네트워크 관리
클러스터 관리	클러스터 관리	클러스터 관리	클러스터 관리
플랫폼 설치 및 구성	플랫폼 설치 및 구성	플랫폼 설치 및 구성	플랫폼 설치 및 구성
OS 패치 및 업그레이드	OS 패치 및 업그레이드	OS 패치 및 업그레이드	OS 패치 및 업그레이드
OS 설치 및 구성	OS 설치 및 구성	OS 설치 및 구성	OS 설치 및 구성
서버 운영 관리	서버 운영 관리	서버 운영 관리	서버 운영 관리
전원/공조/랙 관리	전원/공조/랙 관리	전원/공조/랙 관리	전원/공조/랙 관리

개발자

aws

다양한 컨테이너 선택 옵션

마이크로 서비스를 위한 선택 | 민첩한 개발 및 테스트 가능 | 분 단위 배포 가능

Amazon ECS

높은 확장성 및
고성능 콘테이너
관리 플랫폼 제공

AWS Fargate
for ECS

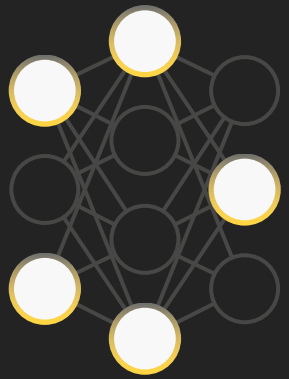
클러스터 관리 불필요
서버리스 기반 콘테이너
배포에만 집중

Amazon EKS

Kubernetes 기반
정식 서비스 가능한
기업형 플랫폼 제공

AWS Fargate
for EKS

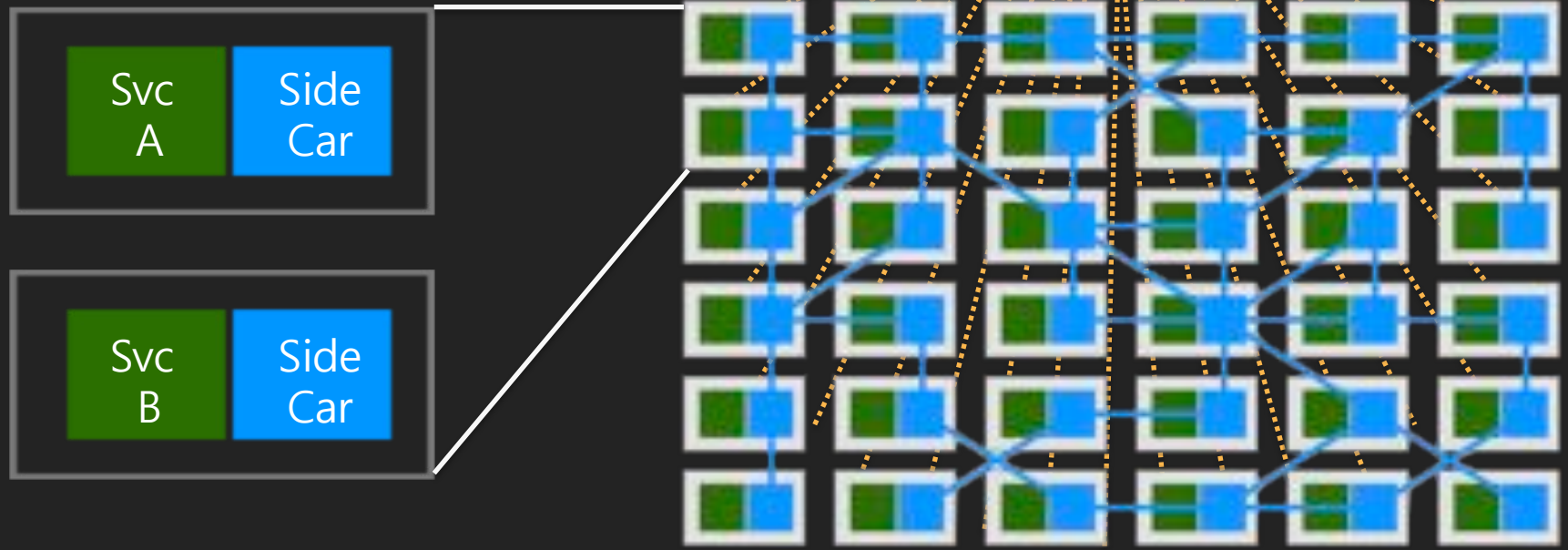
서버리스 기반 K8s
서비스 지원 예정



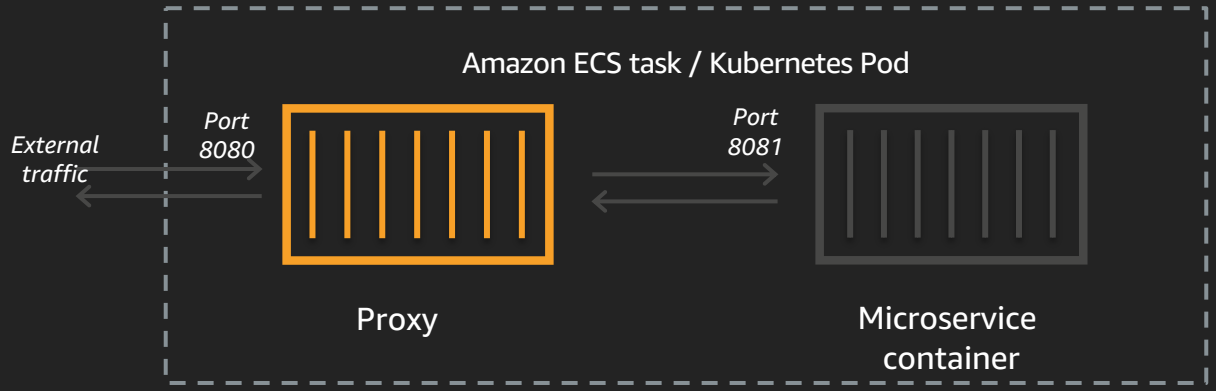
어떻게 마이크로서비스를 관리할 것인가 ?

Microservices to Service Mesh

라이브러리 방식을 벗어나 경량 프록시로
비지니스 로직과 내부 네트워크 분리



SideCar Proxy for Microservices



envoy

- 오픈 소스로 제공되는 프록시
- 커뮤니티 지원 및 다양한 통합 제공
- 안정적으로 검증되어 CNCF 졸업
- 2016년 Lyft에서 시작됨

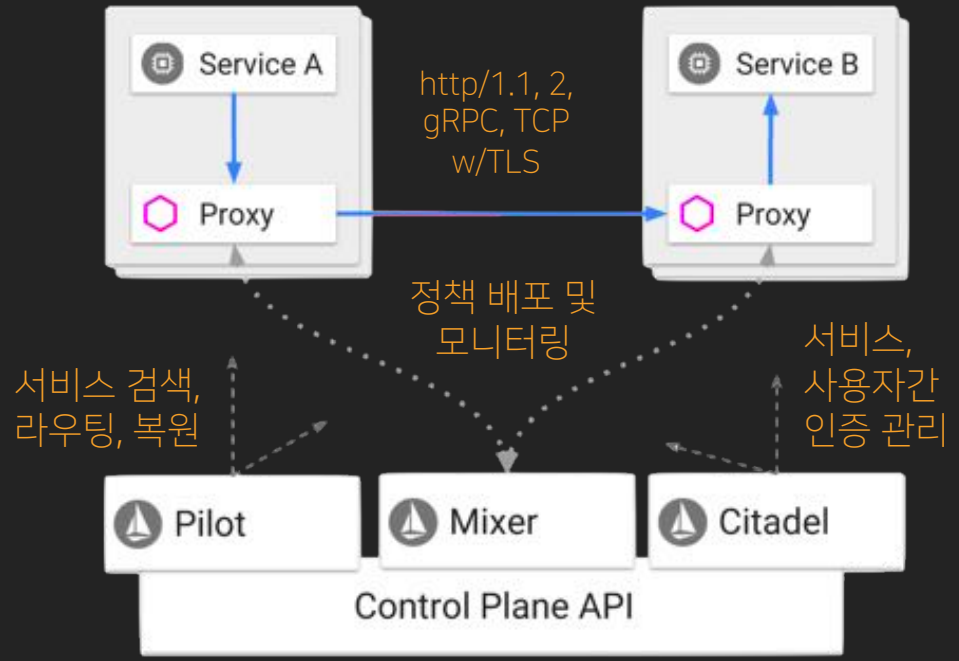
https://github.com/envoyproxy/envoy

Istio – Service Mesh용 오픈 소스

마이크로서비스의 서비스간 통신, 운영 및 배포 복잡성을 해결하기 위한 분산 서비스 관리 도구

주요 기능

- 서비스간 통신은 Envoy라는 오픈 소스 사용
- 지능적 라우팅 및 로드밸런싱 가능
- 언어 및 프레임워크와 독립적인 복원성 제공
- 전체 서비스에 라우팅 정책 적용 가능
- 심층적인 서비스간 트래픽 모니터링



https://github.com/istio/istio

Istio를 통한 트래픽 관리

Traffic Management

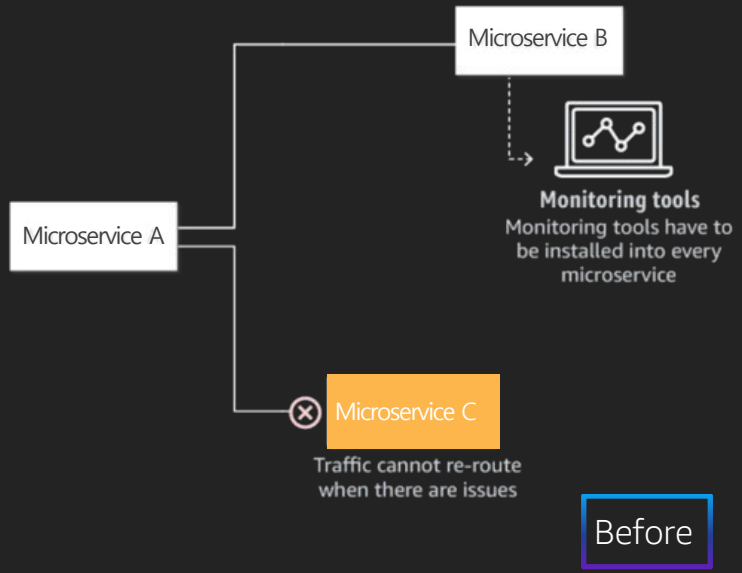
```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
      weight: 75
    - destination:
        host: reviews
        subset: v2
      weight: 25
```

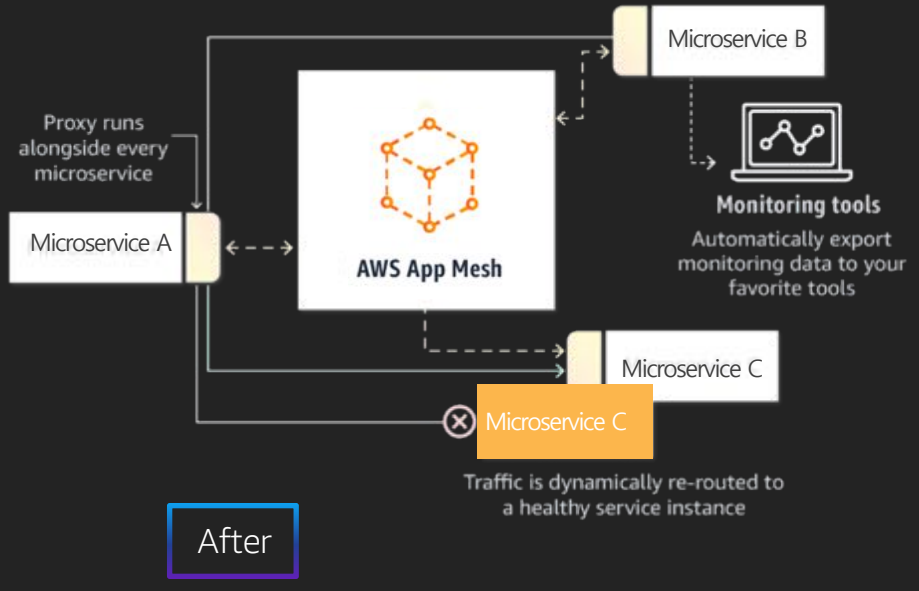
Circuit breaker

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: ratings-destination-rule
spec:
  host: ratings
  subsets:
  - name: ratings
    labels:
      ratings: hello
    trafficPolicy:
      connectionPool:
        tcp:
          maxConnections: 100
```

AWS AppMesh

Amazon ECS 및 **Amazon EKS** 기반 마이크로 서비스에 대한 서비스 메쉬 모니터링 및 제어 가능한 매니지드 서비스

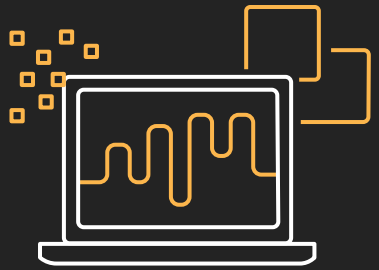




Why AWS App Mesh

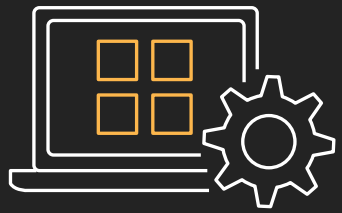
Libraries or application code vs. mesh

SDK나 코드를 통한
개발자들의 직접 관리
작업 필요 없음



애플리케이션
로직과 통신 및
운영단 분리 가능

모든 플랫폼 및
프로그램 언어
사용 가능

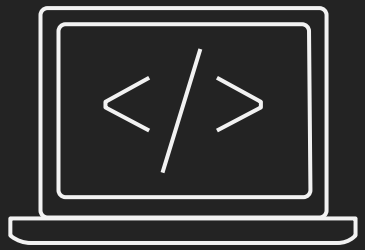


가시성, 문제점 해결 및
배포 등의 복잡한 문제
해결 가능

클라우드 네이티브 마이크로서비스로 이전 쉬움

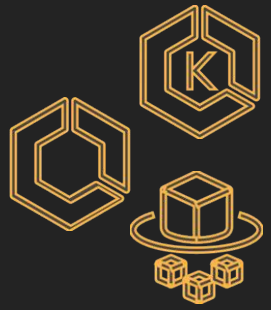
Why AWS App Mesh

vs. building or running your own mesh



개발 및 배포 시스템
구축에 드는 노력
필요 없음

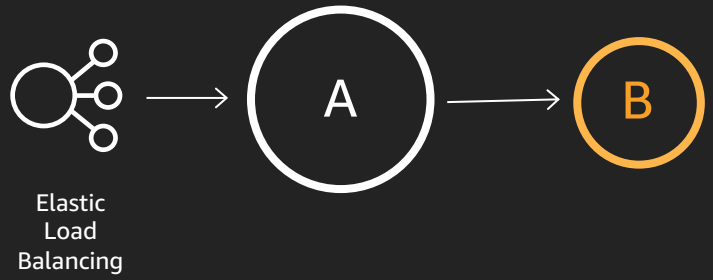
컨테이너 배포
시스템에 대한
종속성 없음

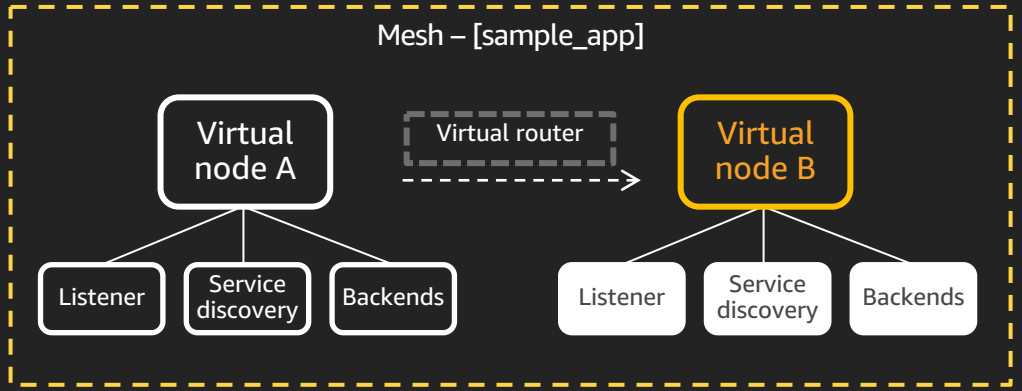


모든 클라우드 시스템
적용 가능
(EC2, ECS, EKS, Fargate)

한번에 또는 점진적
마이크로 서비스
이전 가능

AWS AppMesh 동작 방법





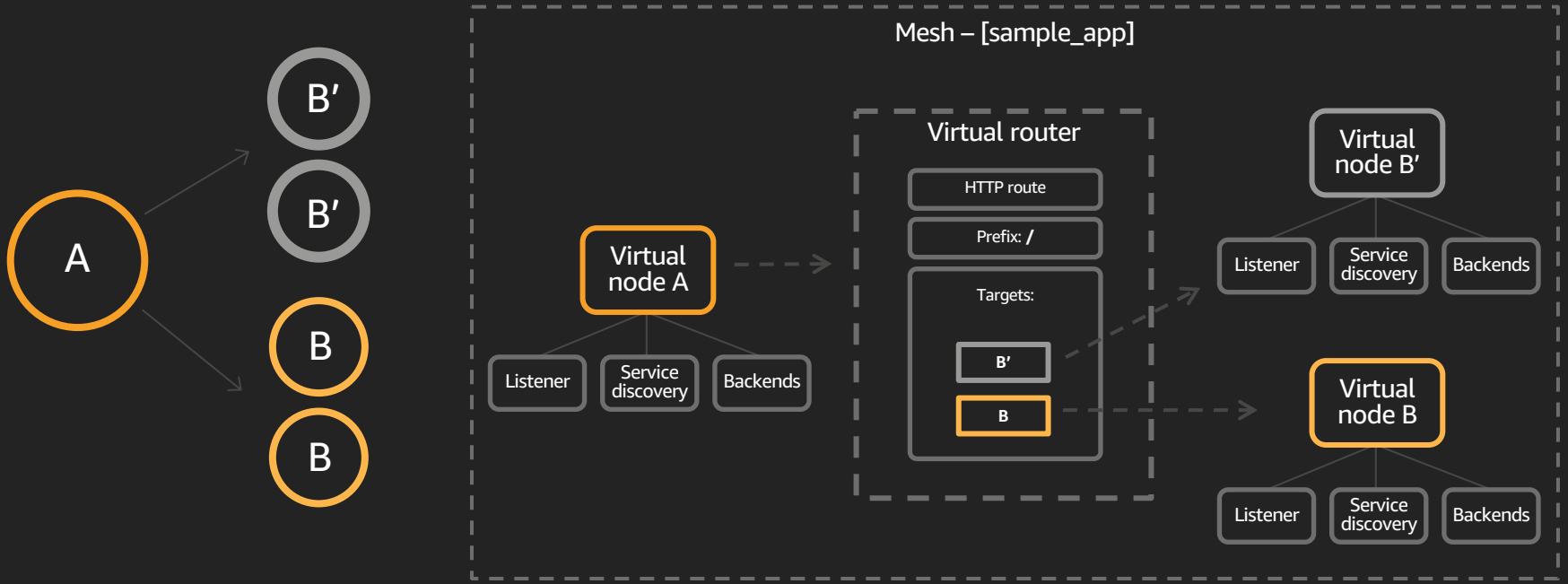
App Mesh API, CLI, SDK 및
AWS 관리 콘솔

AWS Cloud Map – 서비스
디스커버리

Microservices

App Mesh

AWS AppMesh 동작 방법 – 서비스간 전이



Demo – EKS 및 App Mesh 기반 서비스 메시

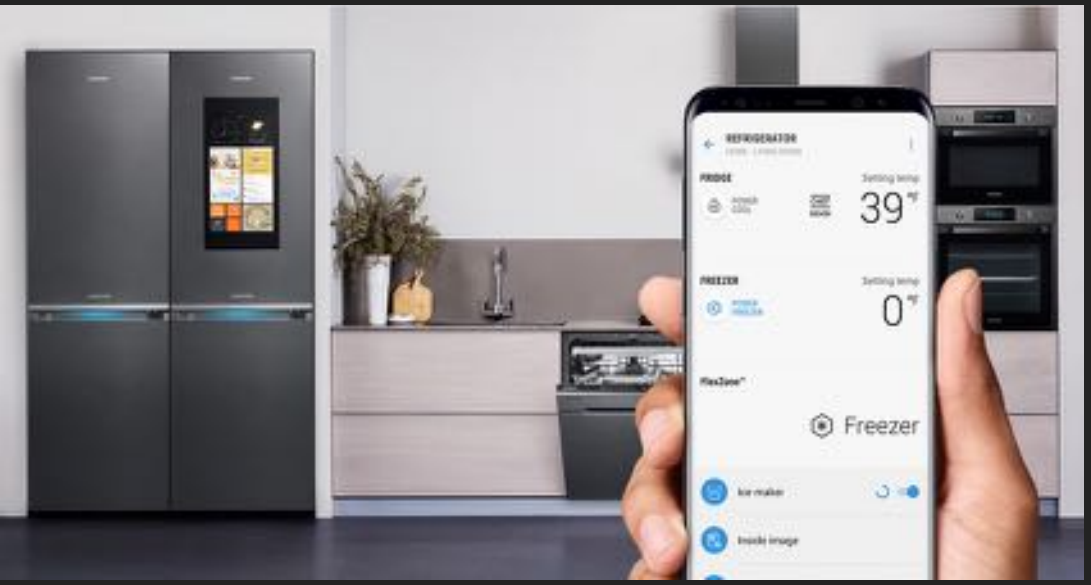
https://youtu.be/u1XmPdc3rc4?t=2106

https://github.com/awslabs/aws-app-mesh-examples

3. 컨테이너 기반 마이크로서비스

삼성전자 및 폭스바겐 구축 사례

Samsung Connect 마이크로 서비스 사례



Connect even more with

SAMSUNG SmartThings™

Samsung Connect 마이크로 서비스 사례

직면한 문제

대용량 트래픽

4개 지역에 나눠진 팀 구성

60+ 애플리케이션 구성 모듈

보안, 확장성 및 신뢰성

솔루션

EC2 Containers & Auto Scaling

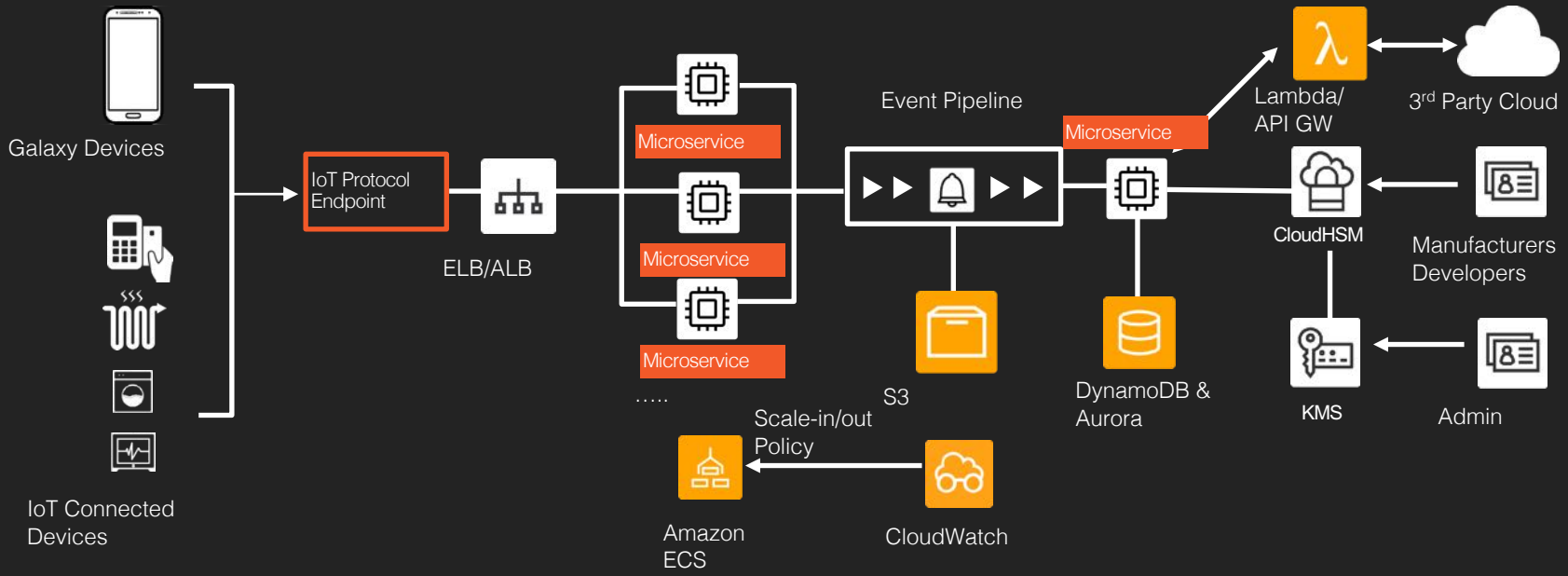
Autonomous/Decentralized

Monolithic to Fine-grained

Automation and
Independent deployment

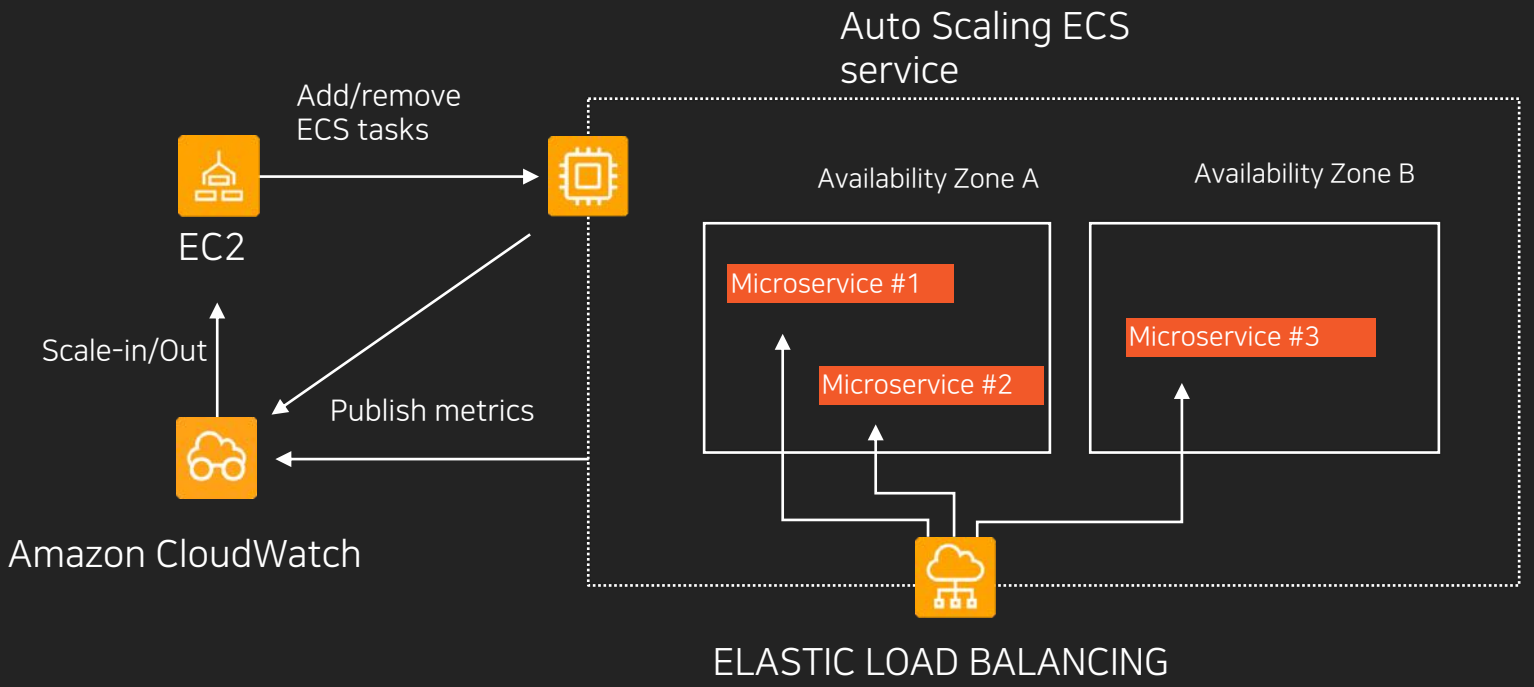
Microservices Architecture on AWS

Samsung Connect 서비스 아키텍처

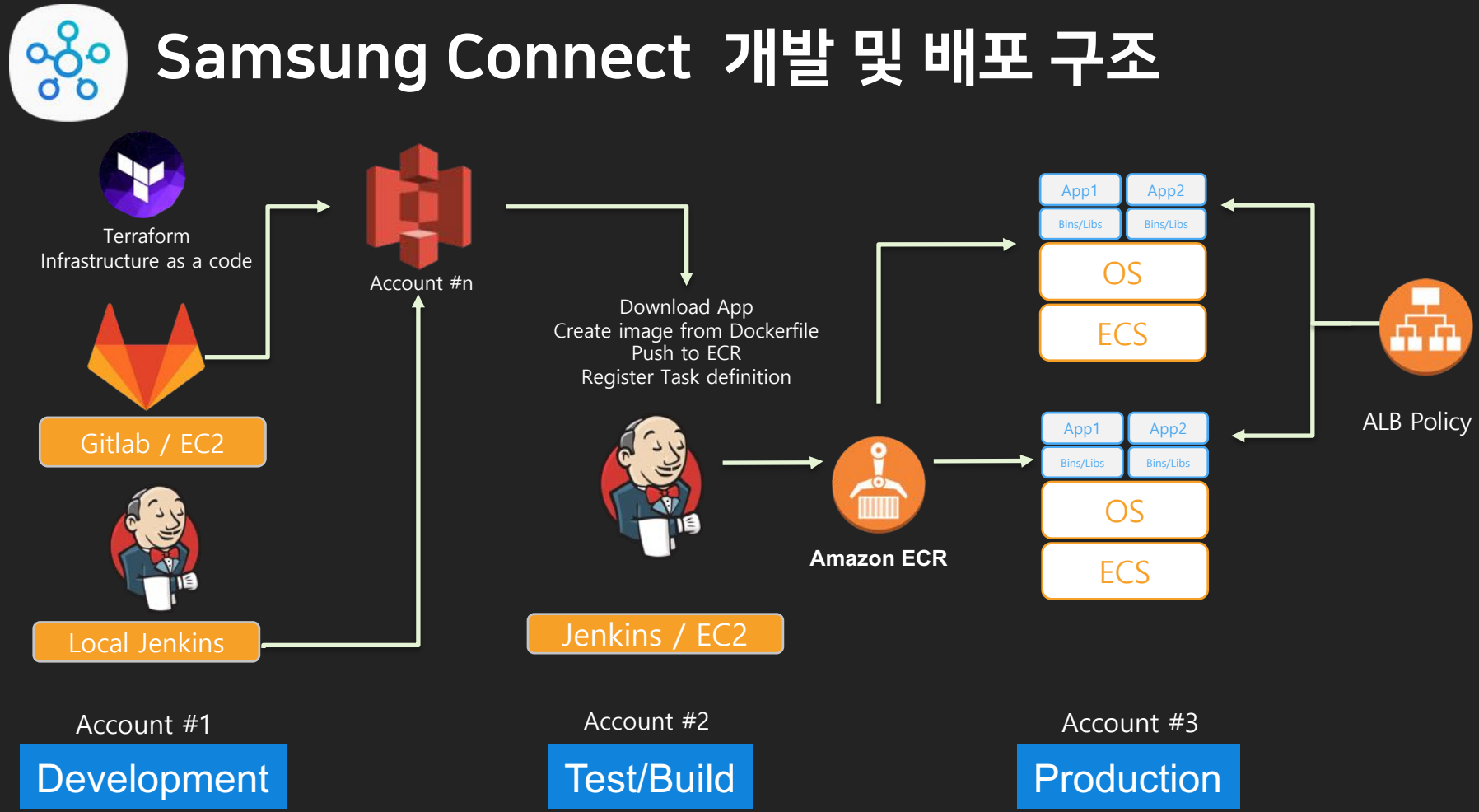


(c) 장수백(삼성전자), Samsung Connect 마이크로서비스 도입 사례, AWS Summit 2016

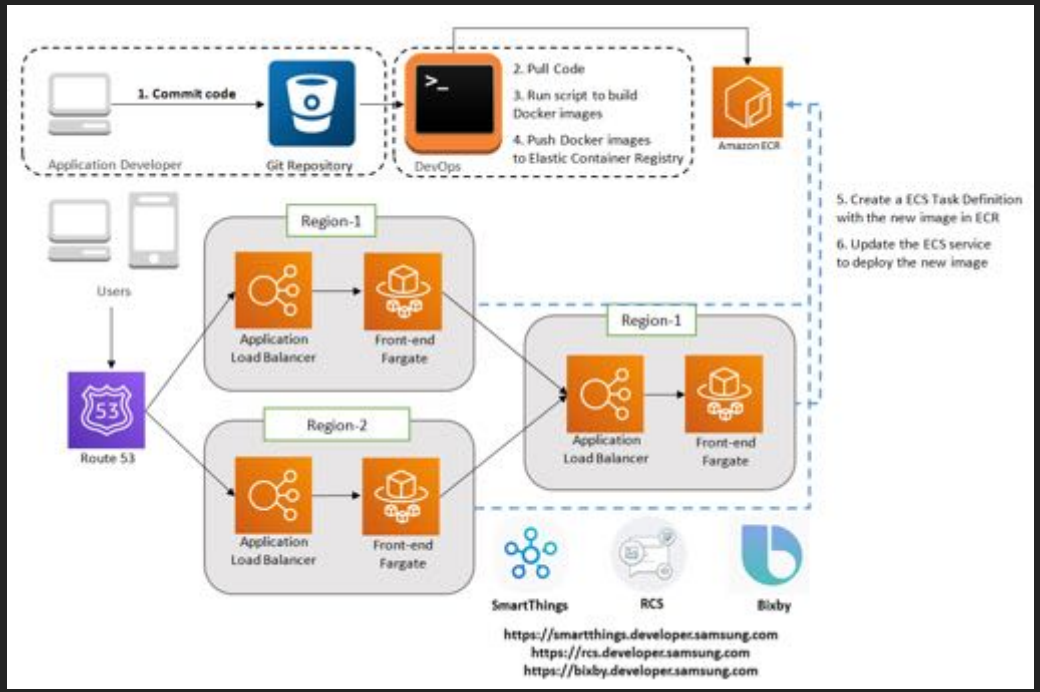
Samsung Connect 컨테이너 관리 구조



Samsung Connect 개발 및 배포 구조



Samsung Developer Portal

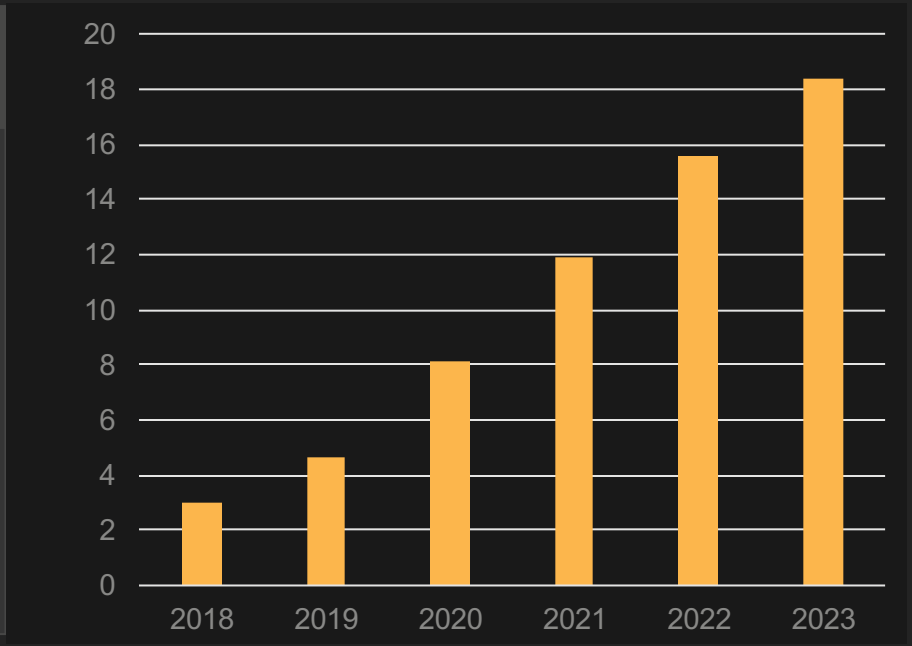


https://aws.amazon.com/ko/blogs/korea/samsung-builds-a-secure-developer-portal-with-fargate-and-ecr/

폭스바겐 커넥티트 카 플랫폼 전환 사례

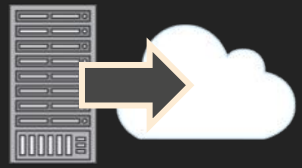
"Modularer Backend Baukasten" (MBB)

- 전 세계 3백 50만대 이상의 자동차 데이터 수집 시스템
- 2011년에 프로젝트 시작
- 100+ 이상의 애플리케이션
- 기존 데이터 센터 기반 구축
- x86 서버 Java/Tomcat/Oracle RAC DB 기반으로 구축



2019년에 한계 용량에 이를 것으로 예상되어 **클라우드 네이티브 플랫폼으로 전환 필요**

MBB 플랫폼 클라우드 전환 프로젝트 개요



Scope

100+ 개 이상의
애플리케이션
클라우드 전환

Timeline

새 모델 출시 전까지
14개월 안에 확장
가능한 플랫폼으로 전환

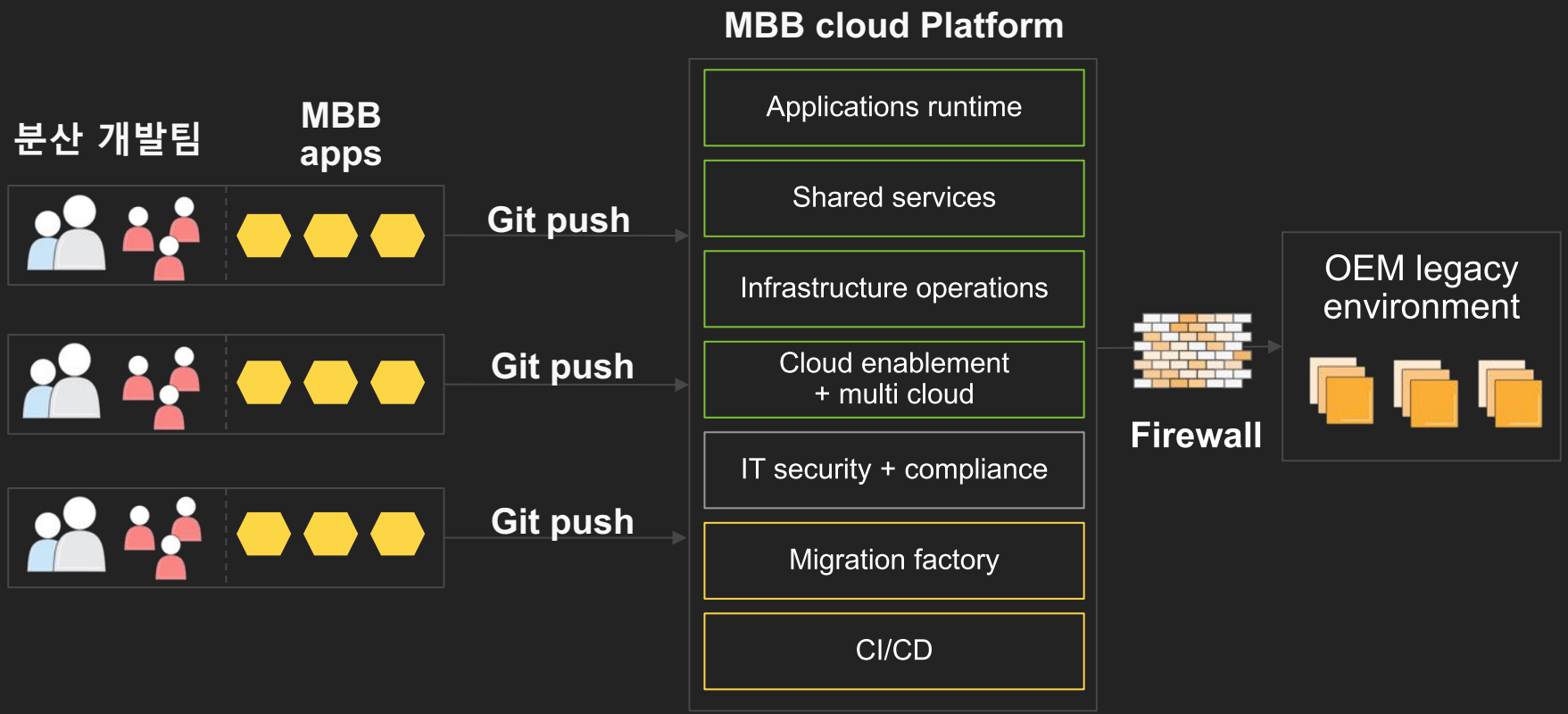
Approach

컨테이너 기반으로
리팩토링 및 Oracle
DB 전환 계획 수립

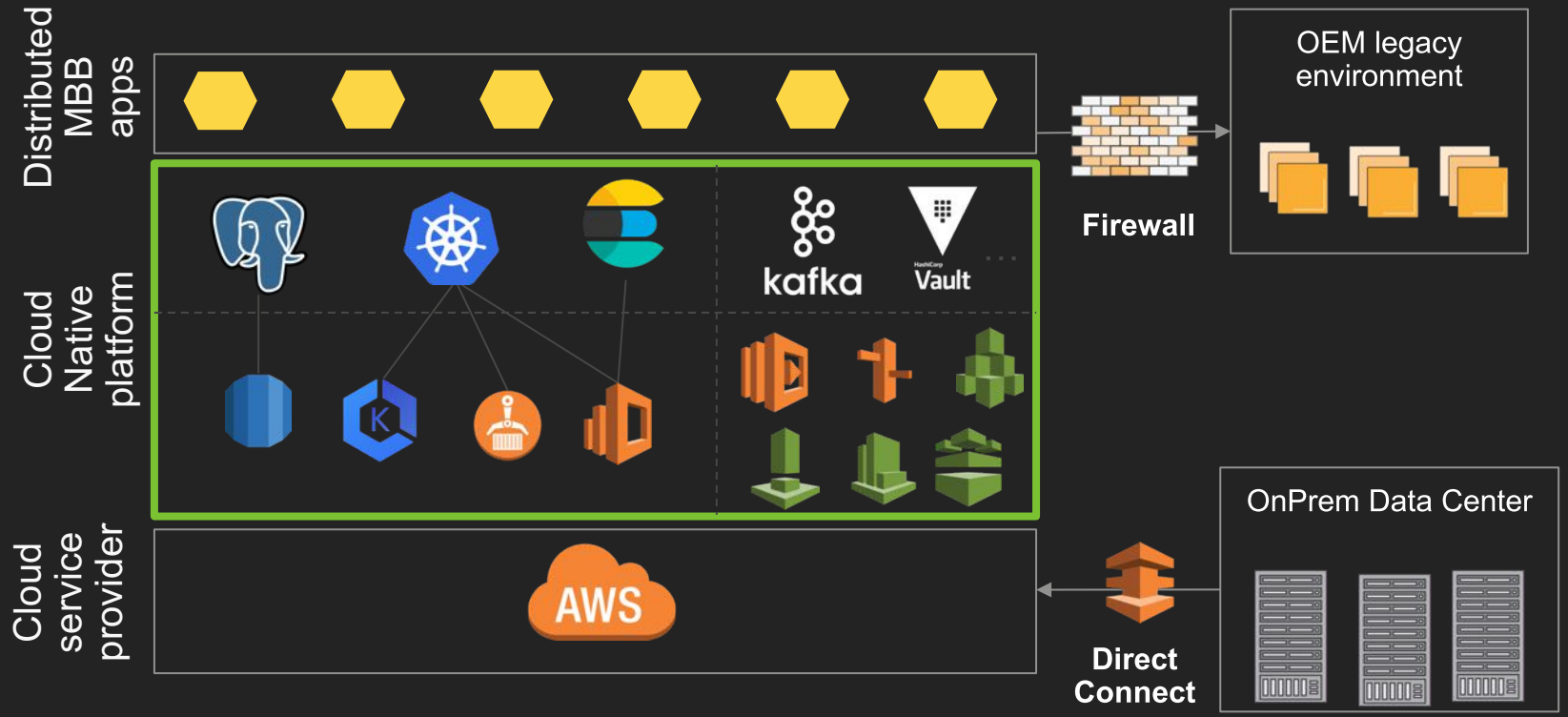
Team

120+ 개발자 투입
(Volkswagen, Audi,
Porsche 팀)

마이크로서비스 모범 사례 기반 플랫폼 설계



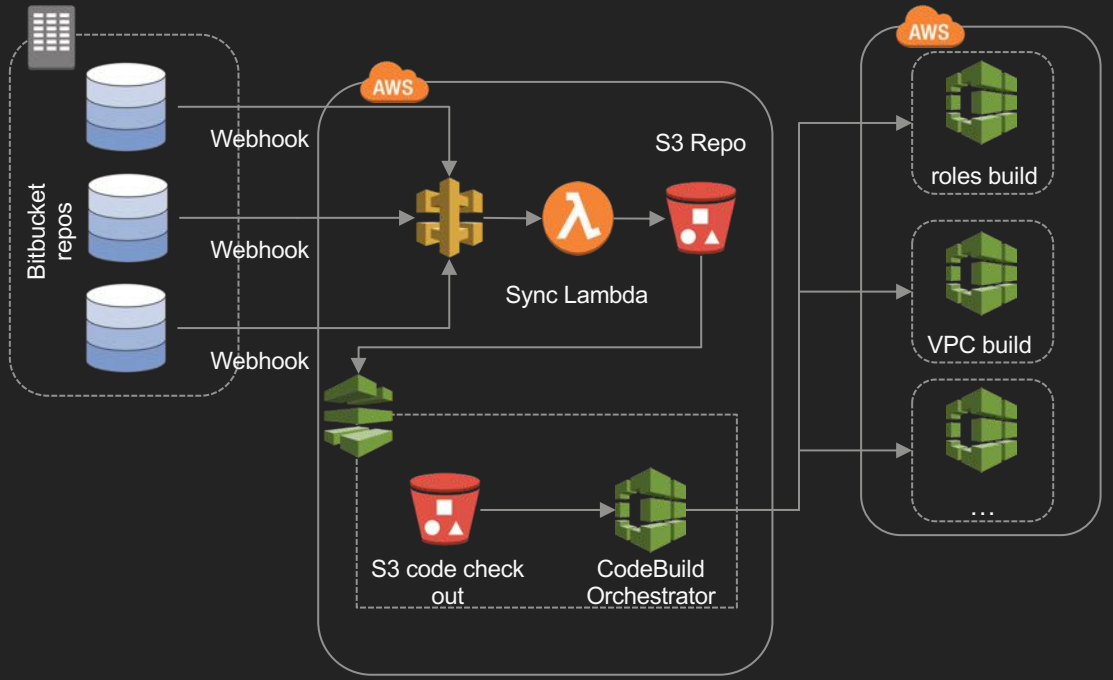
쿠버네티스 기반 컨테이너 및 클라우드 플랫폼 활용



서버리스를 활용한 멀티 리전 배포 플랫폼 구축

Bootstrap

Automate new environment bootstrapping by implementing a standardized modular pipeline



마이크로 서비스의 이점

개별 마이크로 서비스 확장 가능	빠른 운영 및 개선	높은 민첩성
빠른 빌드/테스트/배포 가능	몇 분만에 배포 가능	빠른 혁신
명확한 오너쉽 및 자율적 운영	신규 기능 빠르게 추가 가능	고객 만족

어디서부터 시작할까요?

- **마이크로서비스로 전환해야 하는가?**
  - 모놀리식 아키텍처의 단점이 보이는가?
  - 자율성 및 다양성에 대한 개발 플랫폼 변화가 가능한가?
  - 기업 내에 개발 문화 및 팀 조직 개선의 의지가 있는가?

- **새로운 변화를 선택했다면?**
  - 기존 클라우드, 데브옵스, 마이크로서비스의 모범 사례 활용
  - 작은 것 부터 마이그레이션하고, 실험을 반복
  - AWS 클라우드 전문가의 아키텍처 도움 & 교육 지원 요청

Q&A

channyun@amazon.com
https://aws.amazon.com/ko/contact-us/

 channyblog  @channyun