

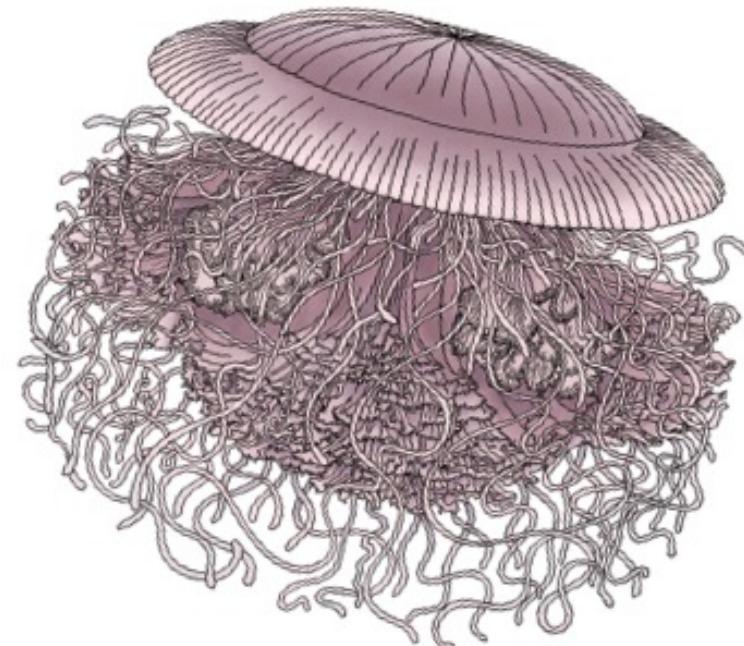
# AWS X 마이크로서비스



O'REILLY®

# Monolith to Microservices

Evolutionary Patterns to Transform  
Your Monolith



Sam Newman

With examples in Java

# Microservices Patterns

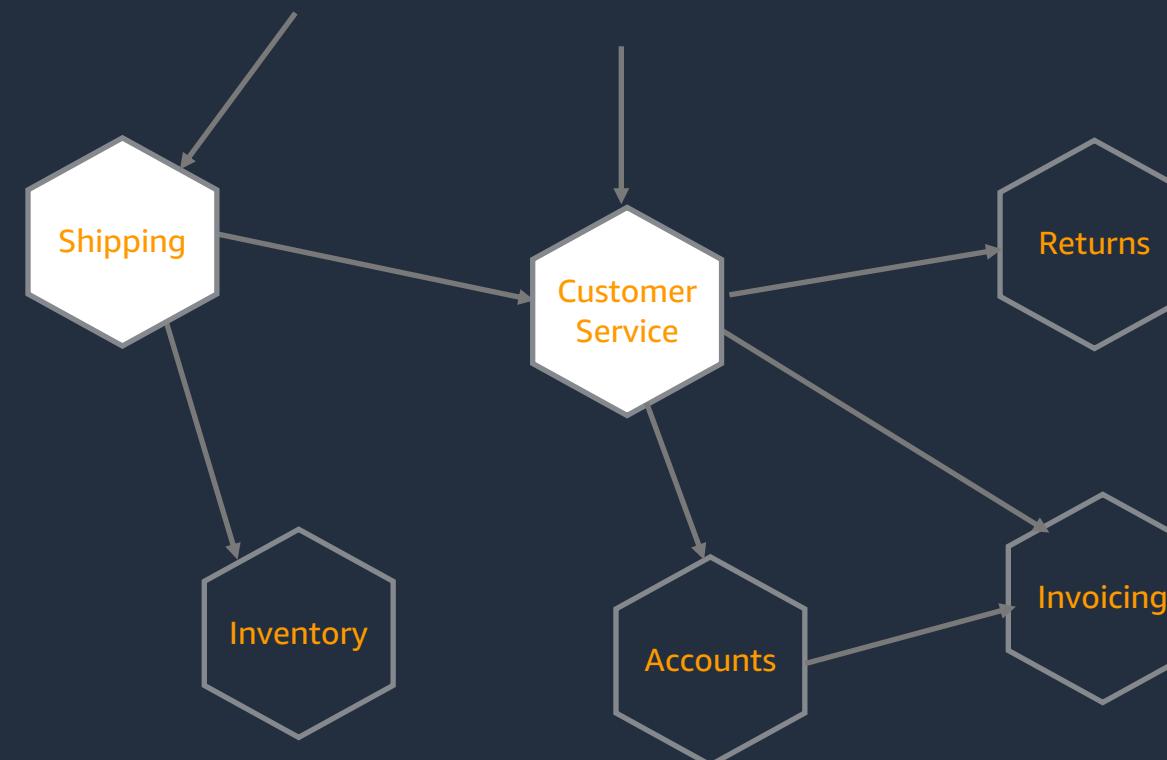
Chris Richardson

MANNING



# マイクロ서비스는 무엇입니까?

- 중요! 함께 작동하는, 독립적으로 배포 가능한 서비스는 비즈니스 도메인을 중심으로 모델링됩니다.
- 네트워크를 통해 서로 통신합니다.
- 데이터베이스는 서비스 경계 내부에 숨겨져 있습니다.
- 정보 은폐의 원칙.



# 비즈니스 도메인을 중심으로 하는 모델링



UI Team

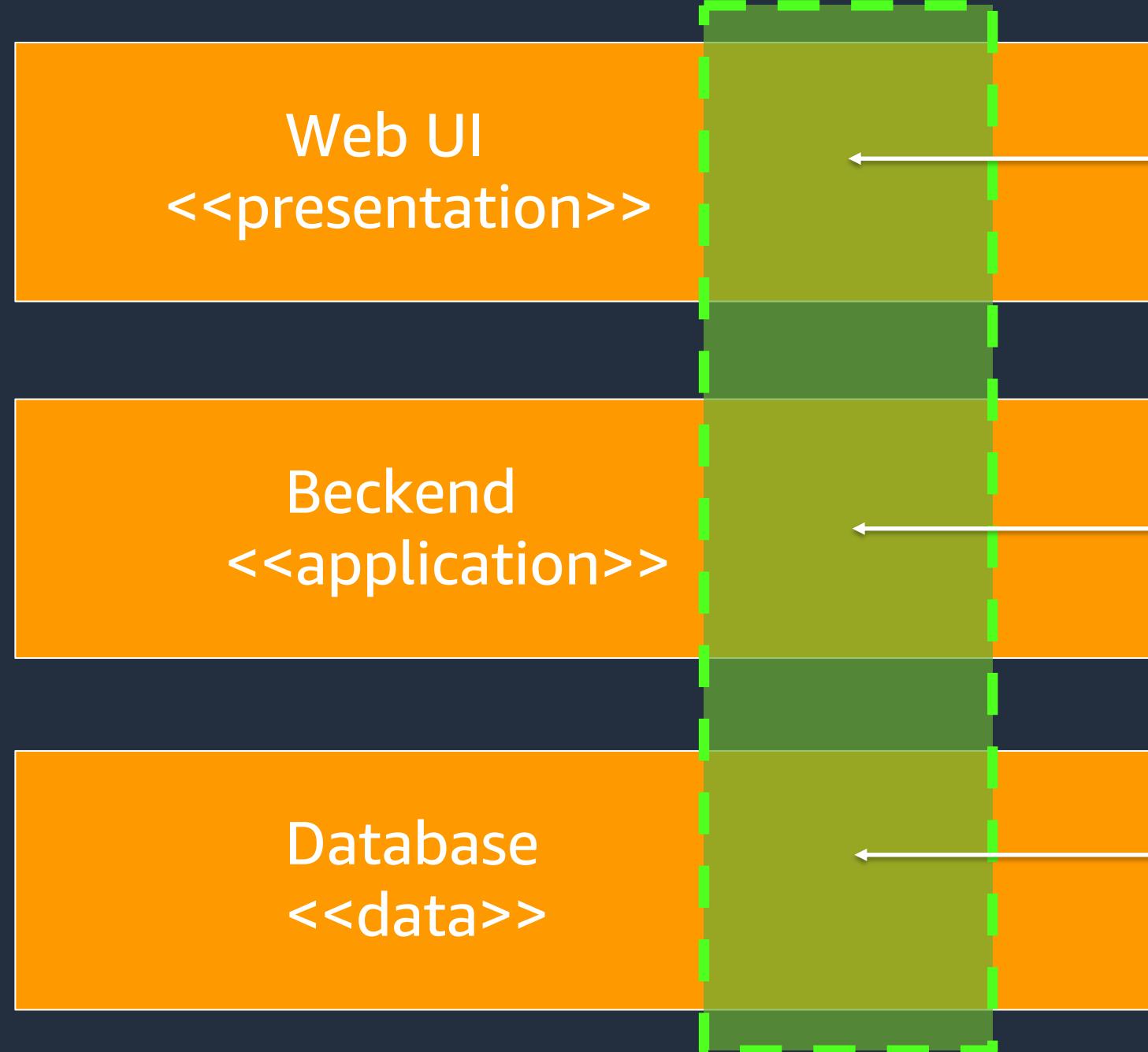


Backend Team



DBAs

Scope of Change

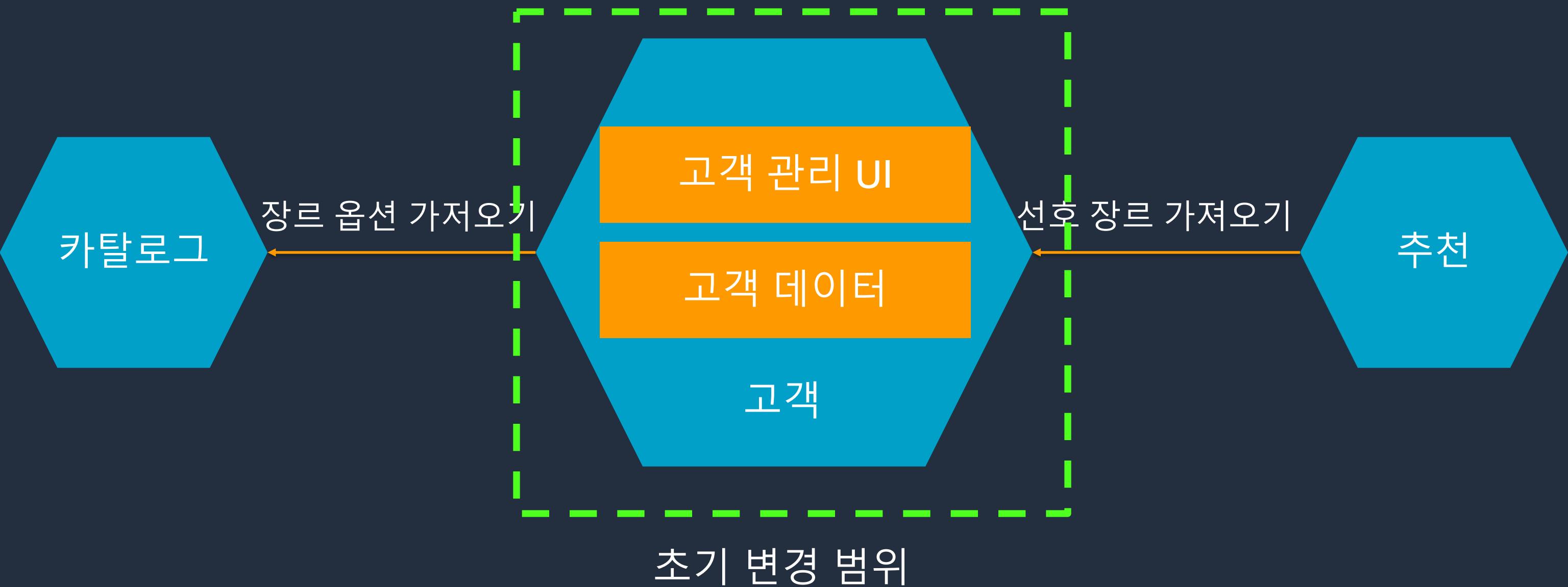


추천 UI 컨트롤 표시

추천 API 생성 및 노출

추천 선택 저장

# 비즈니스 도메인을 중심으로 하는 모델링



# 독립적인 배포와 데이터 소유권 문제

- 변경 사항을 독립적으로 변경하고 배포합니다.
- 느슨한 결합:
  - 서비스 간의 잘 정의되고 안정적인 계약.
  - 일부 구현 선택은 이를 어렵게 만듭니다. 예: 데이터베이스 공유.
- 자신의 데이터를 소유하십시오.

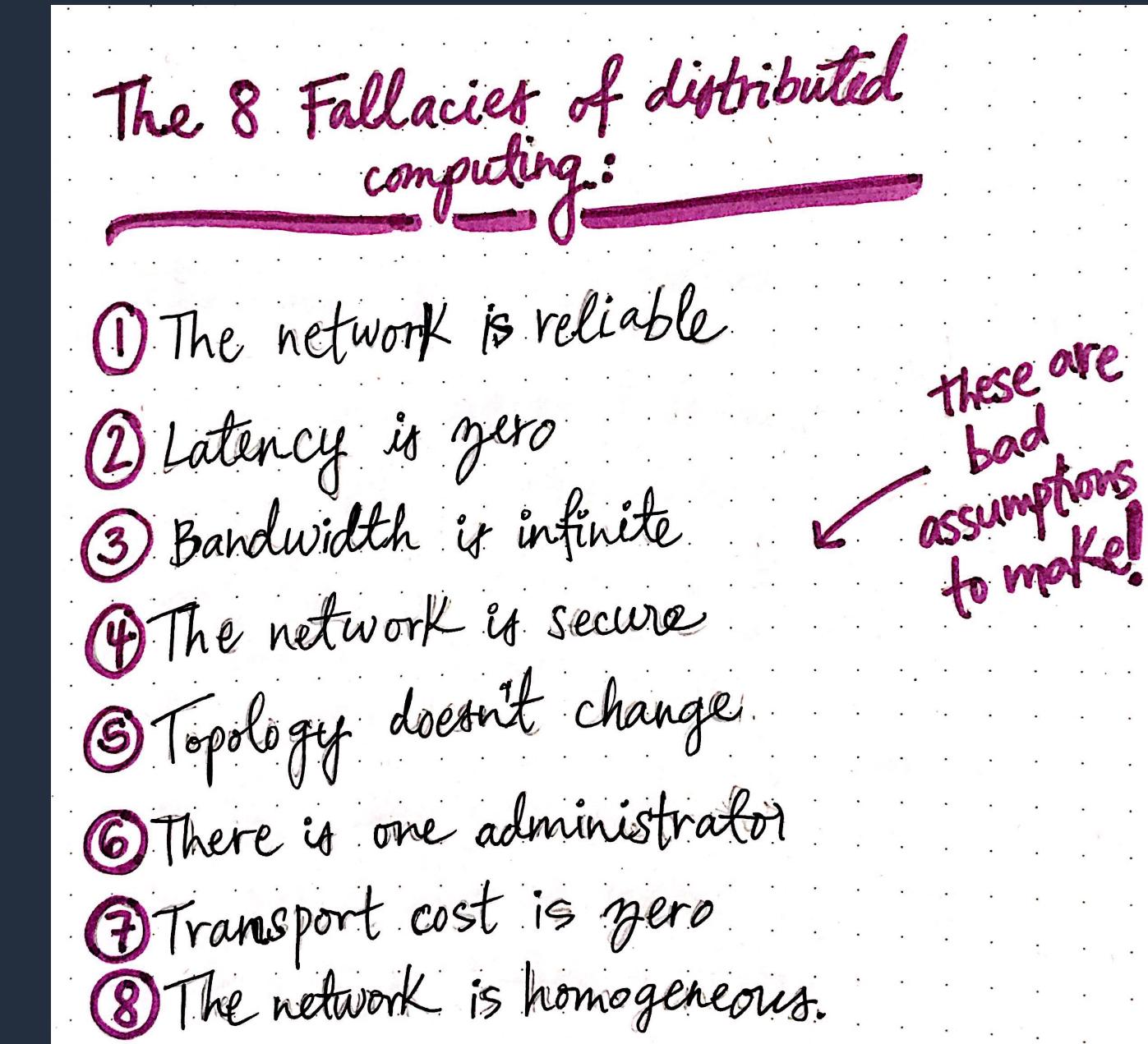


# マイクロ서비스의 장점

- 기술 믹스 앤 매치
- 독립적인 확장 가능
- 하이퍼 포커스 2-pizza 팀
- 더 많은 개발자 모시기
- 더 간단한 워크플로, 모니터링, 문제 해결 및 e2e 테스트

# マイクロサービスが や기하는 문제점

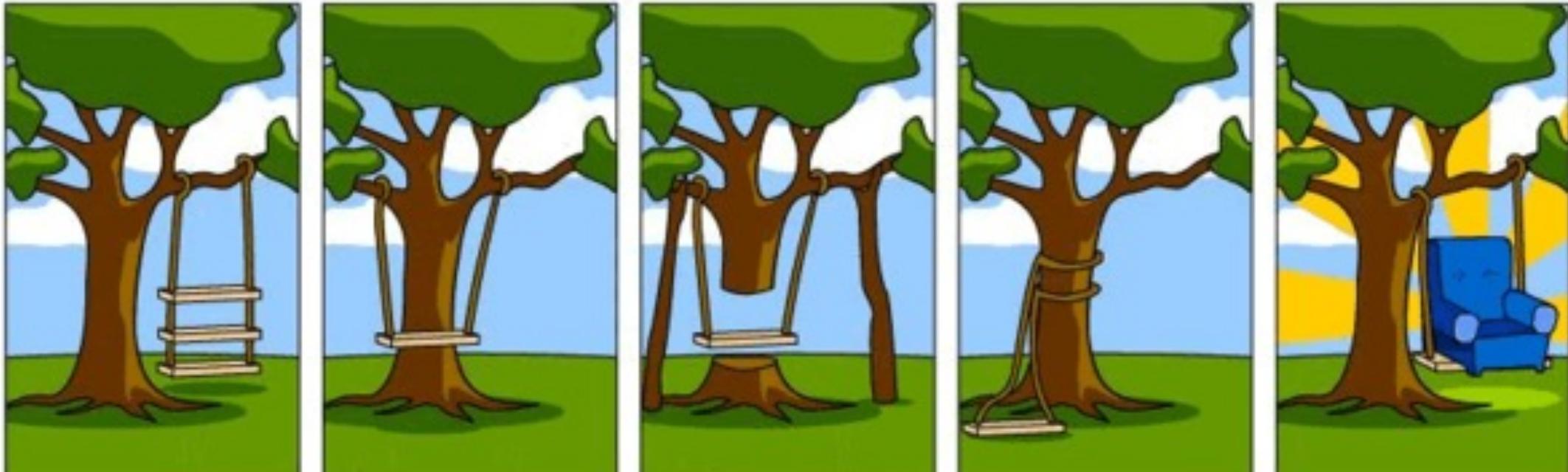
- 네트워크는 신뢰할 수 있습니다.
- 지연 시간은 0입니다.
- 대역폭은 무한합니다.
- 네트워크는 안전합니다.
- 토폴로지는 변경되지 않습니다.
- 한 명의 관리자가 있습니다.
- 전송 비용은 0입니다.
- 네트워크는 동질적입니다.
- 우리는 모두 서로를 신뢰합니다.



Source: <https://medium.com/baseds/foraging-for-the-fallacies-of-distributed-computing-part-1-1b35c3b85b53>

# 소유권

기술  
기능



고객이 설명한 요건

프로젝트리더의 이해

매널리스트의 디자인

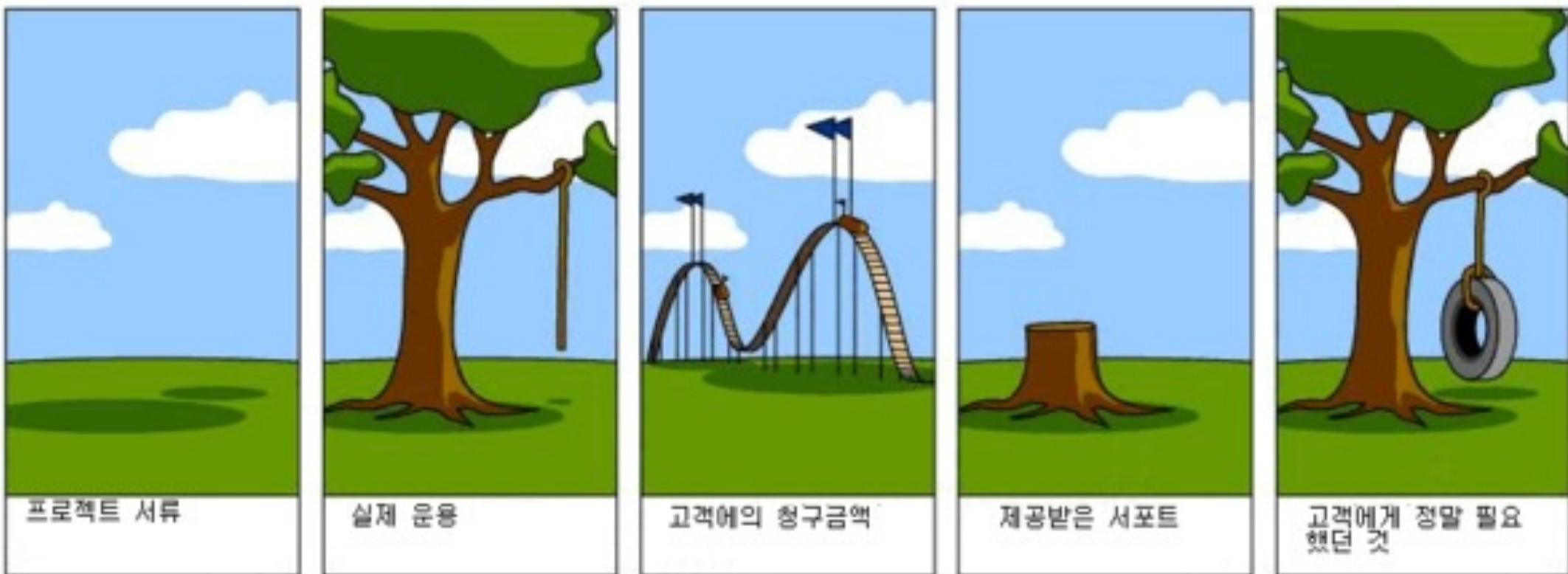
프로그래머의 코드

영업맨의 표현, 약속

관리  
(PMO etc)



배포 팀



프로젝트 서류

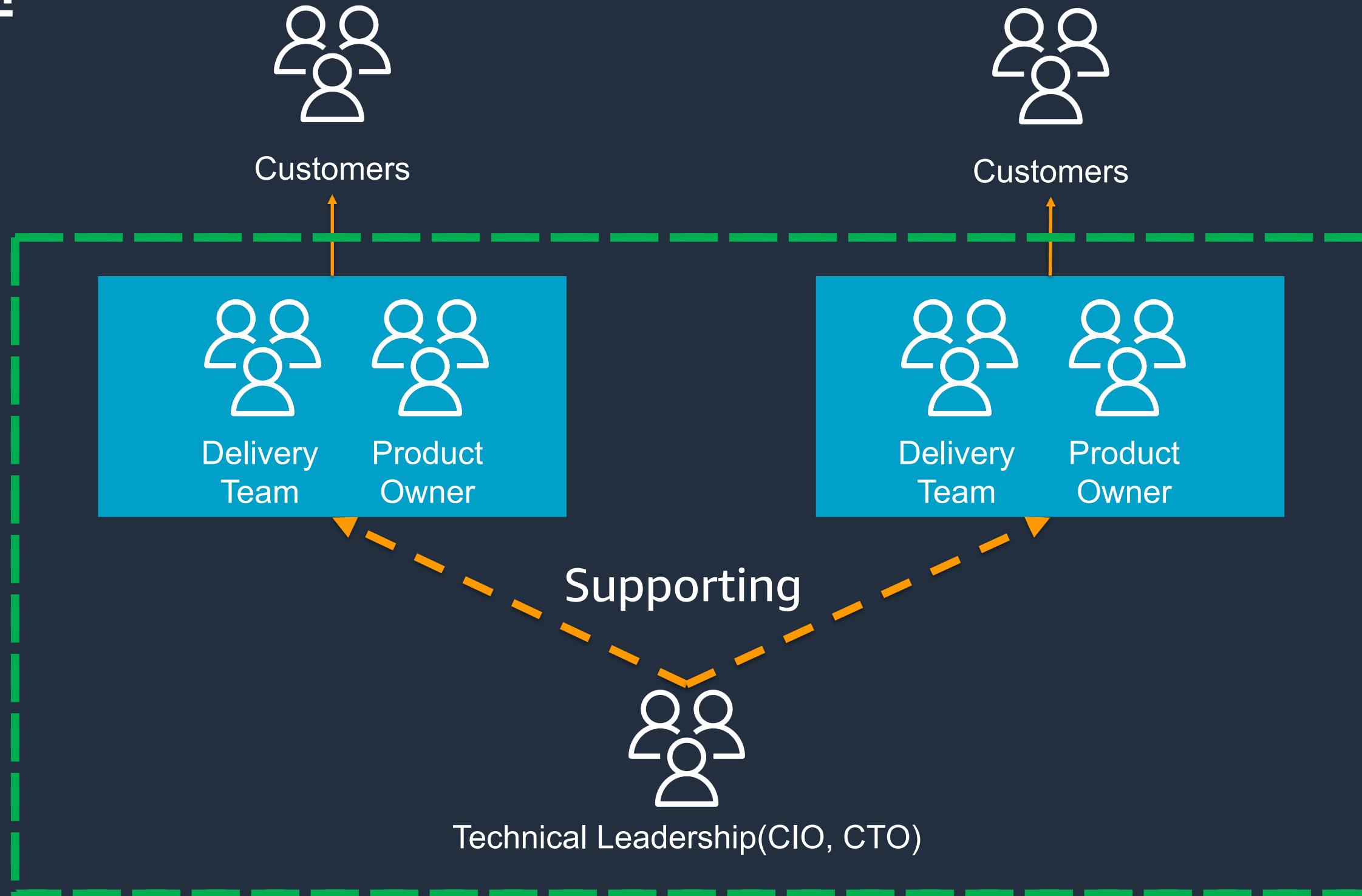
실제 운용

고객에의 청구금액

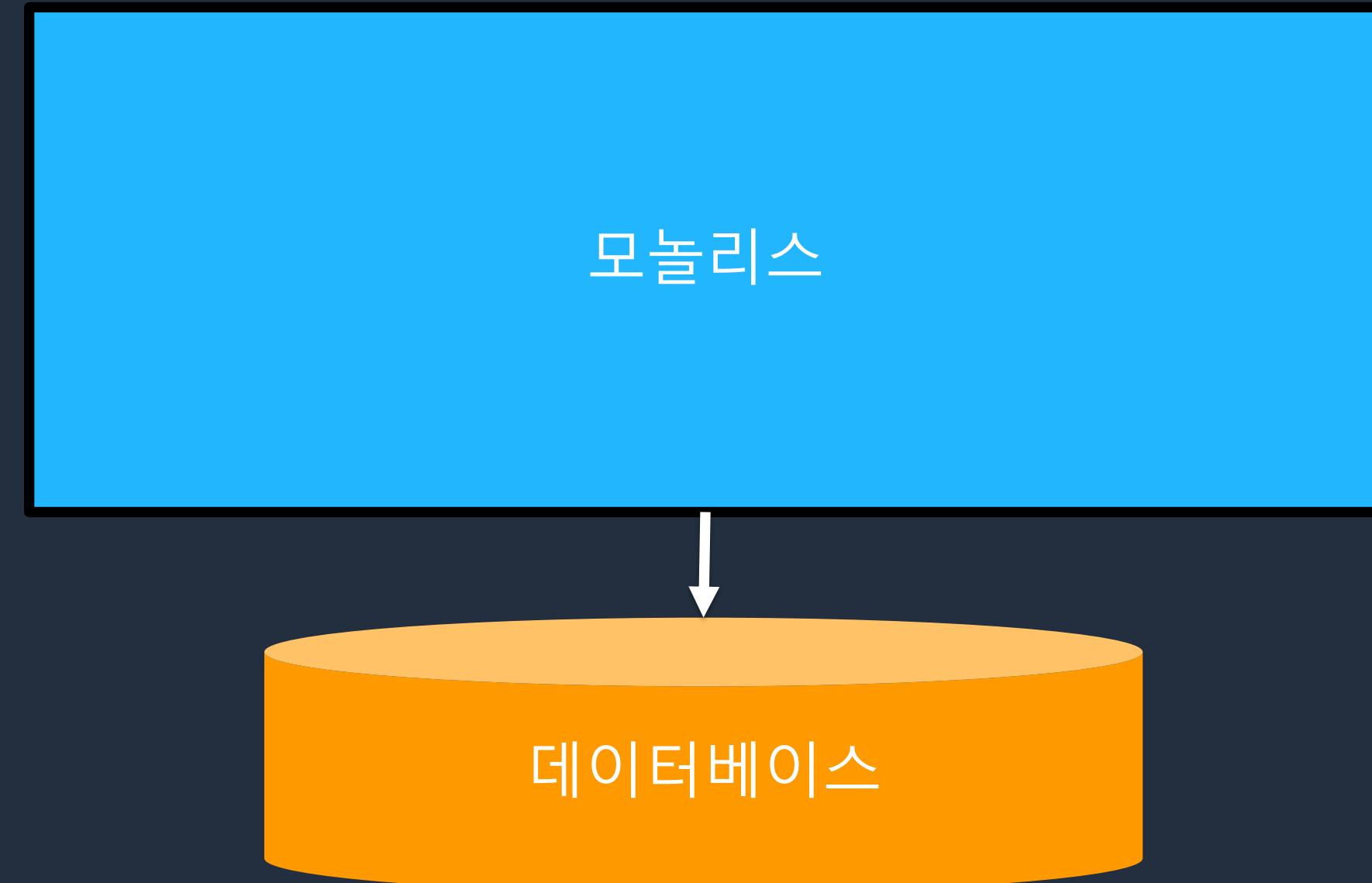
제공받은 서포트

고객에게 정말 필요  
했던 것

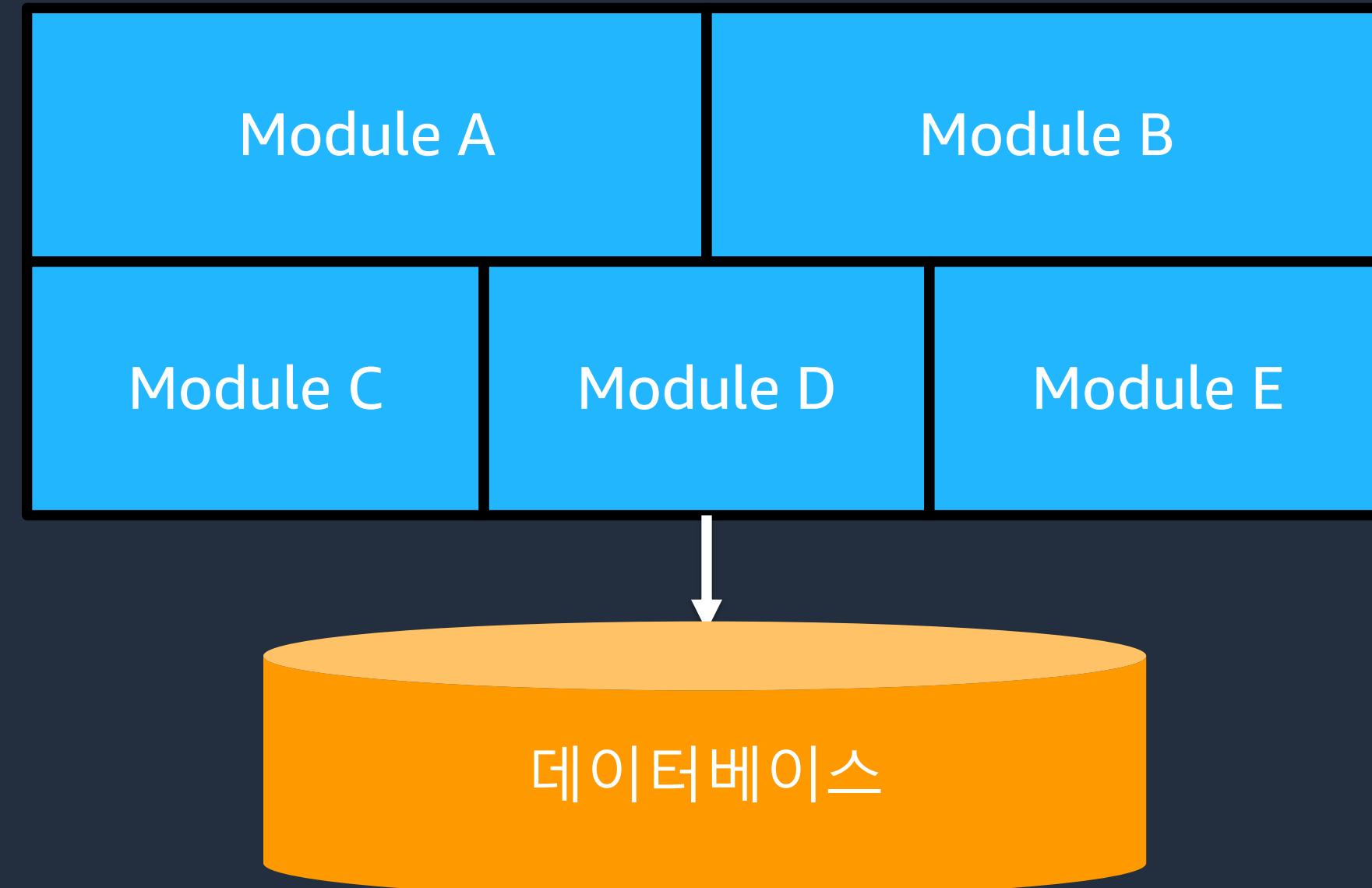
# 소유권



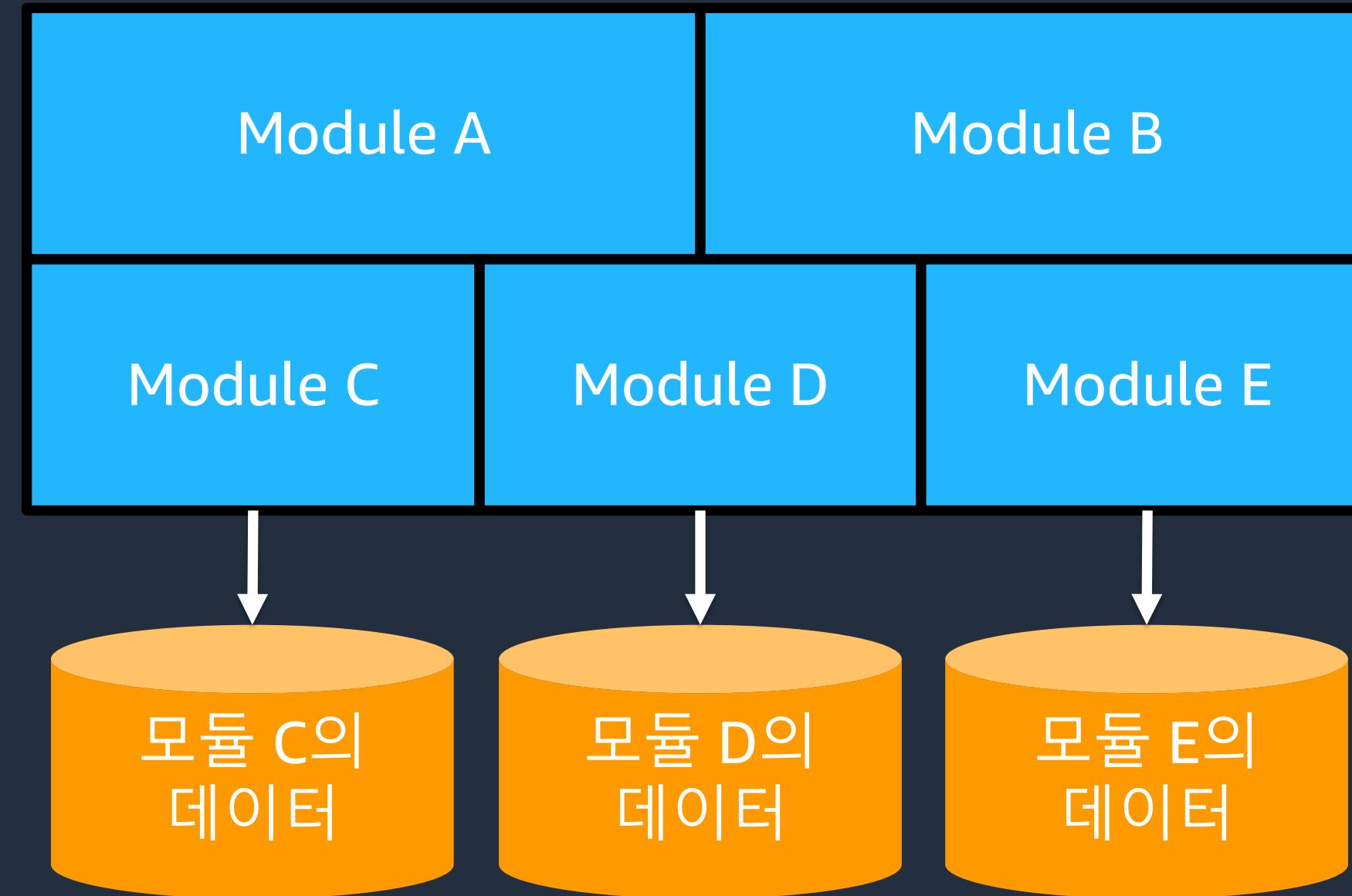
# 모놀리스의 종류 - 단일 모놀리스



# 모놀리스의 종류 - 모듈식 모놀리스



# 모놀리스 - 분산 모놀리스



# 모놀리스의 장점과 문제점

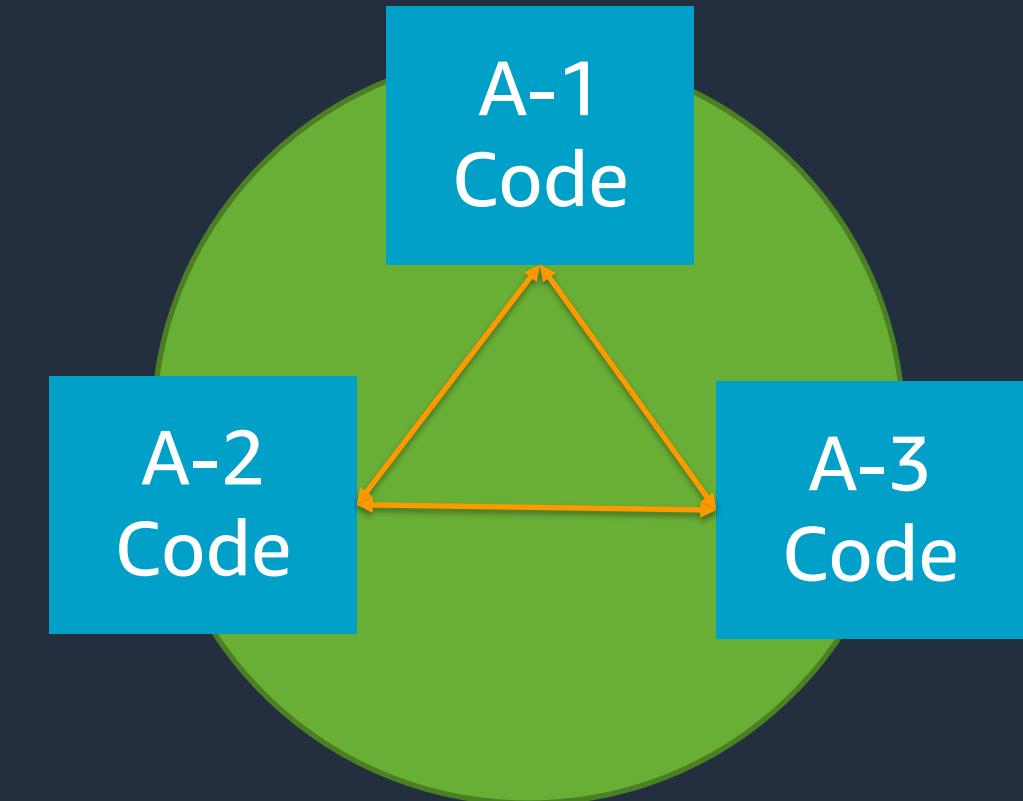
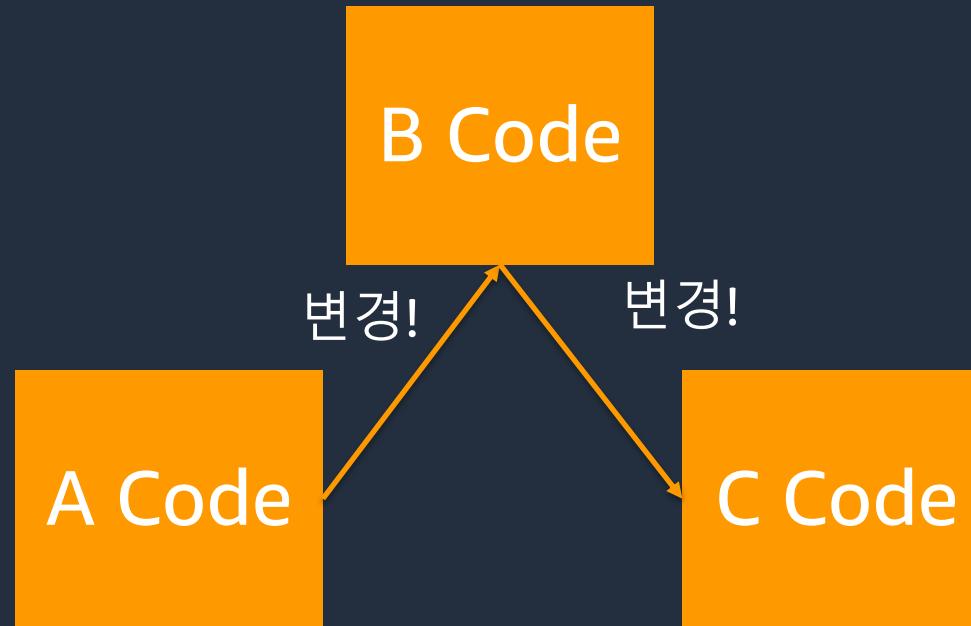
## 장점

- 간단한 배포
- 간단한 개발자 워크플로우
- 간단한 모니터링과 테스트
- 분산 시스템의 단점을 피함

## 문제점

- 높은 결합도
- 혼란스러운 소유권의 경계선
- 배포 경합

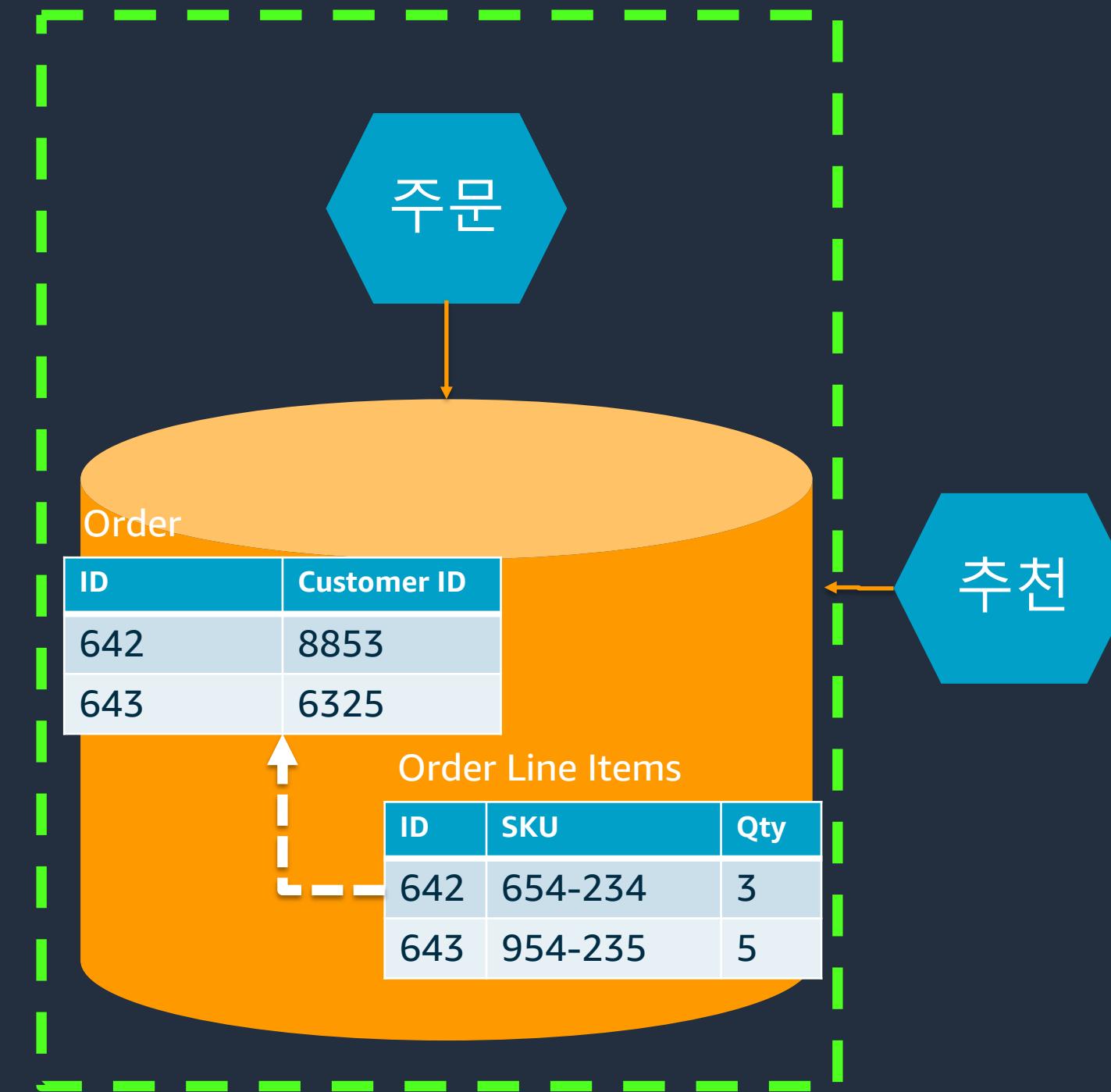
# 결합도(Coupling)와 응집력(Cohesion)



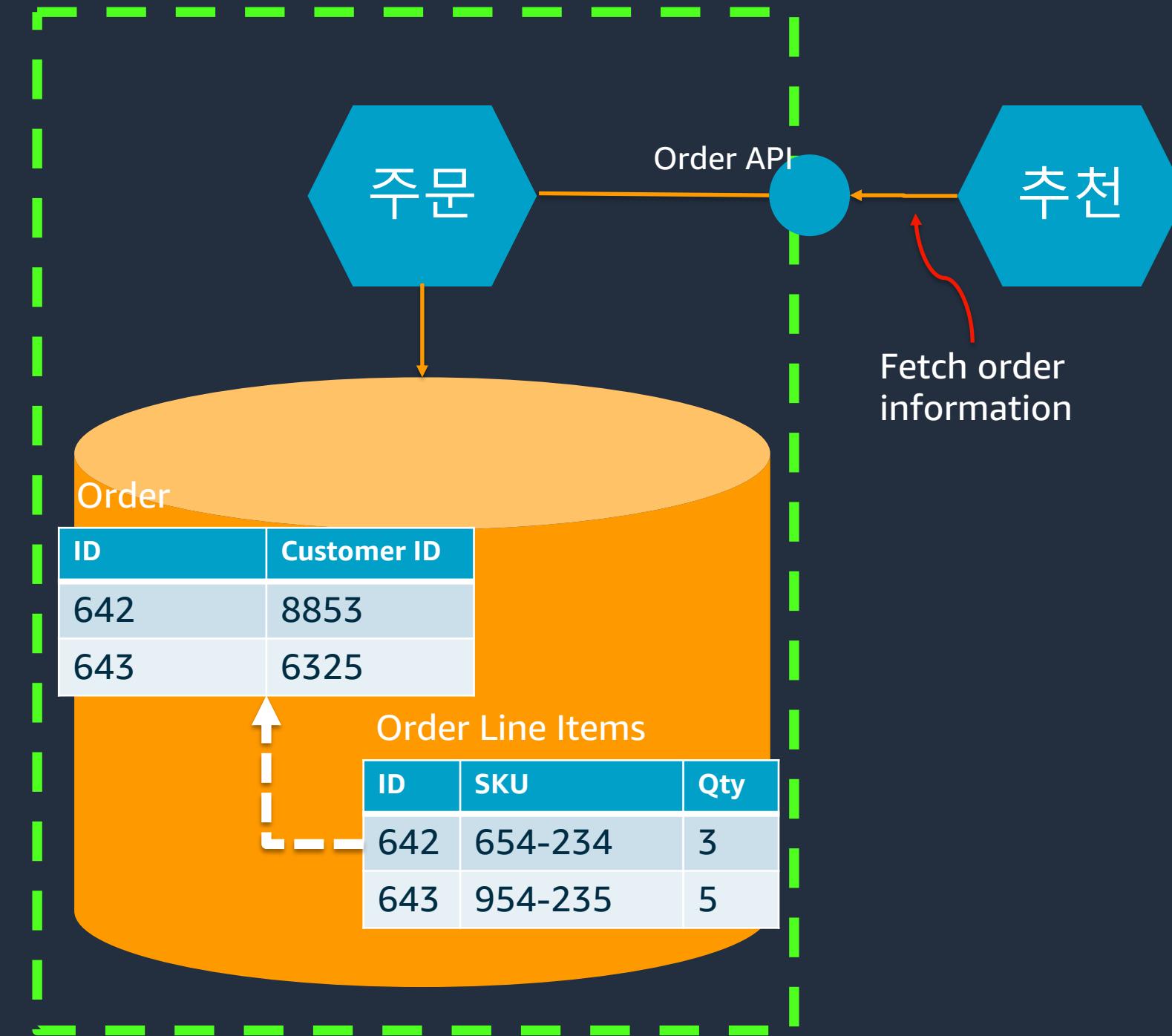
구조는 응집력이 높고 결합도가 낮을 때 안정적이다

- 레리 콘스탄틴

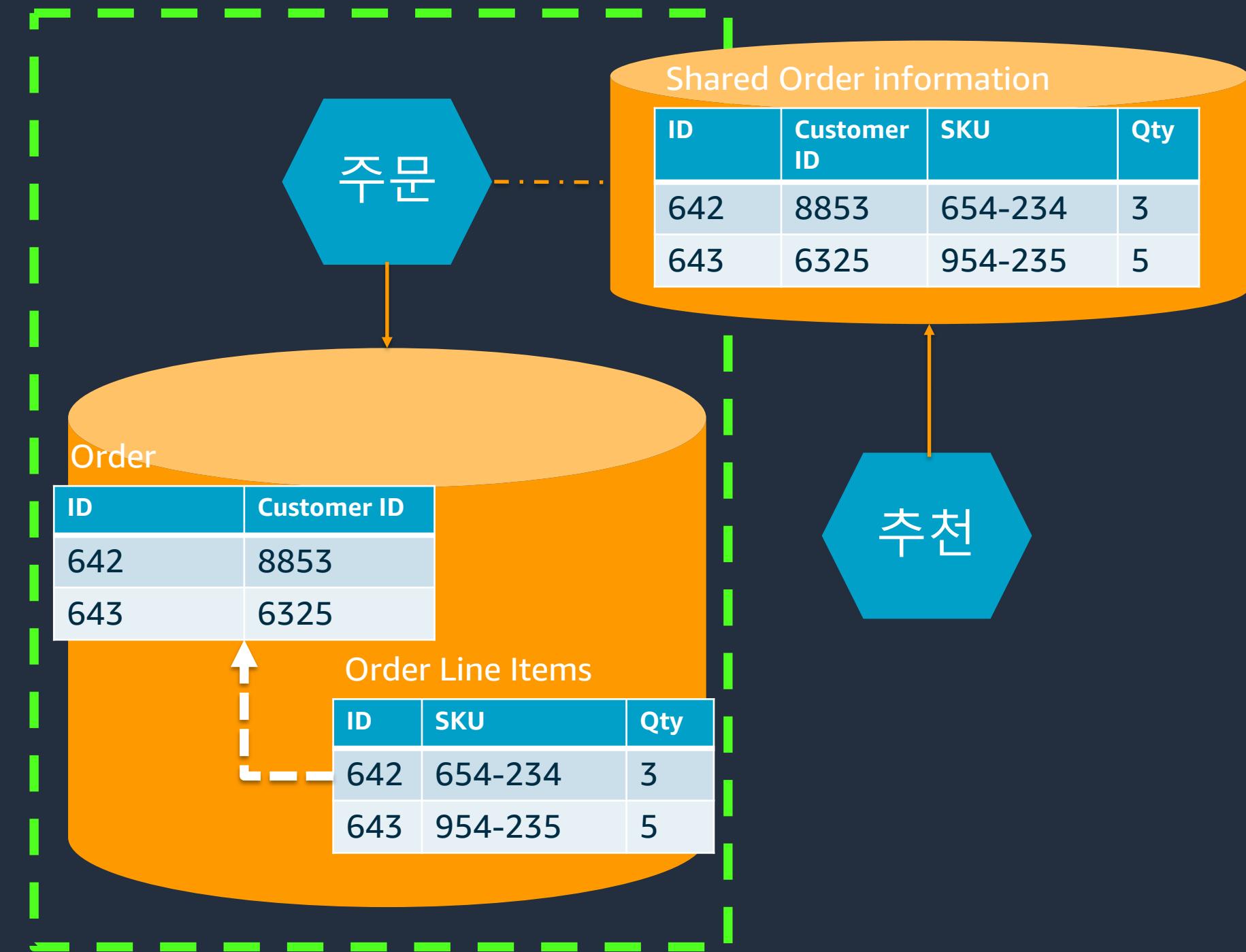
# 결합도의 종류 - (1) 구현 결합도(implementation coupling)



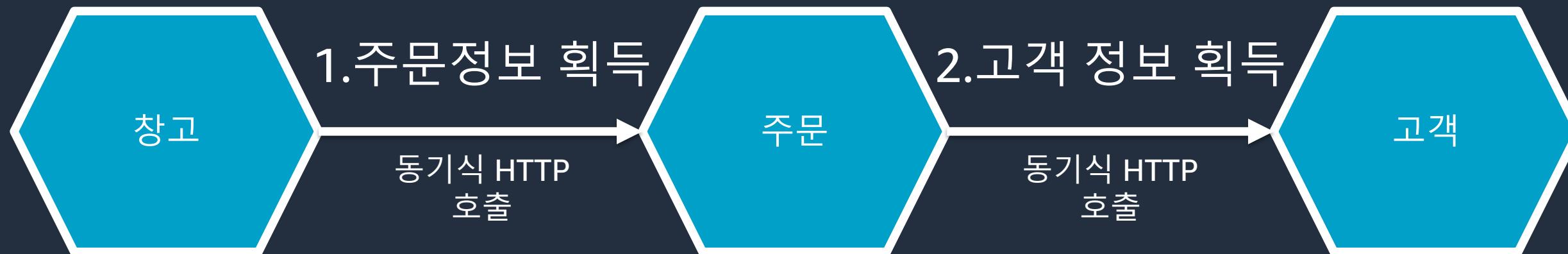
# 커플링의 종류 - (1) 구현 커플링



# 커플링의 종류 - (1) 구현 커플링(implementation coupling)



# 결합도의 종류 - (2) 일시적 결합도(temporal coupling)

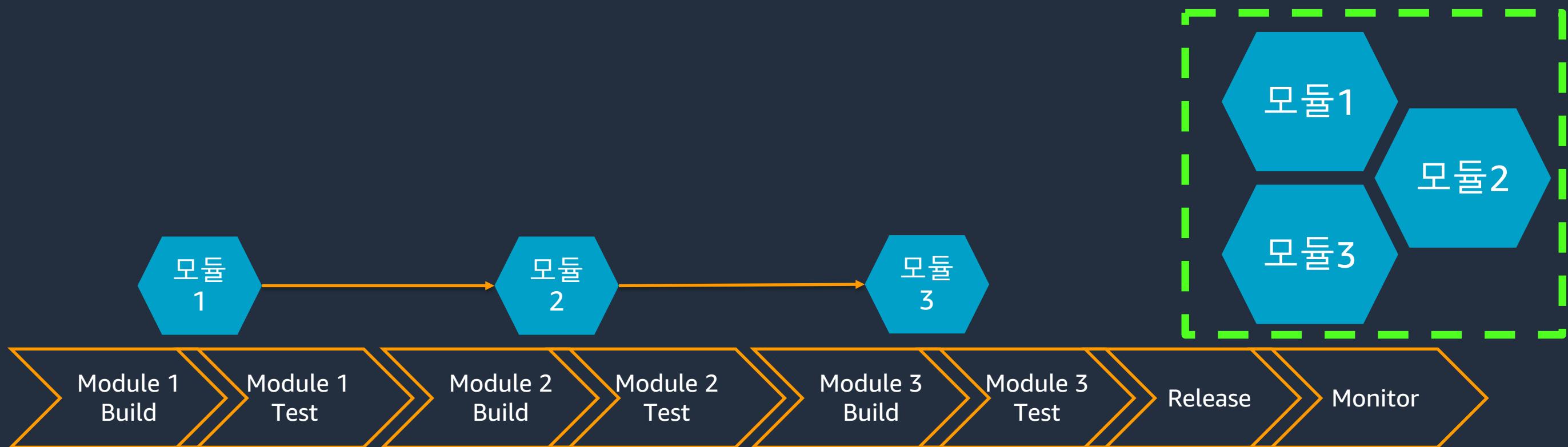


$$SLA \text{ Total} = SLA1 * SLA2 * SLA3$$

$$SLA \text{ Total} = 0.99 * 0.99 * 0.99 = 0.97$$

# 결합도의 종류 - (3) 배포 결합도(Deployment Coupling)

- 단일 프로세스는 정적으로 연결된 여러 모듈로 구성됩니다.



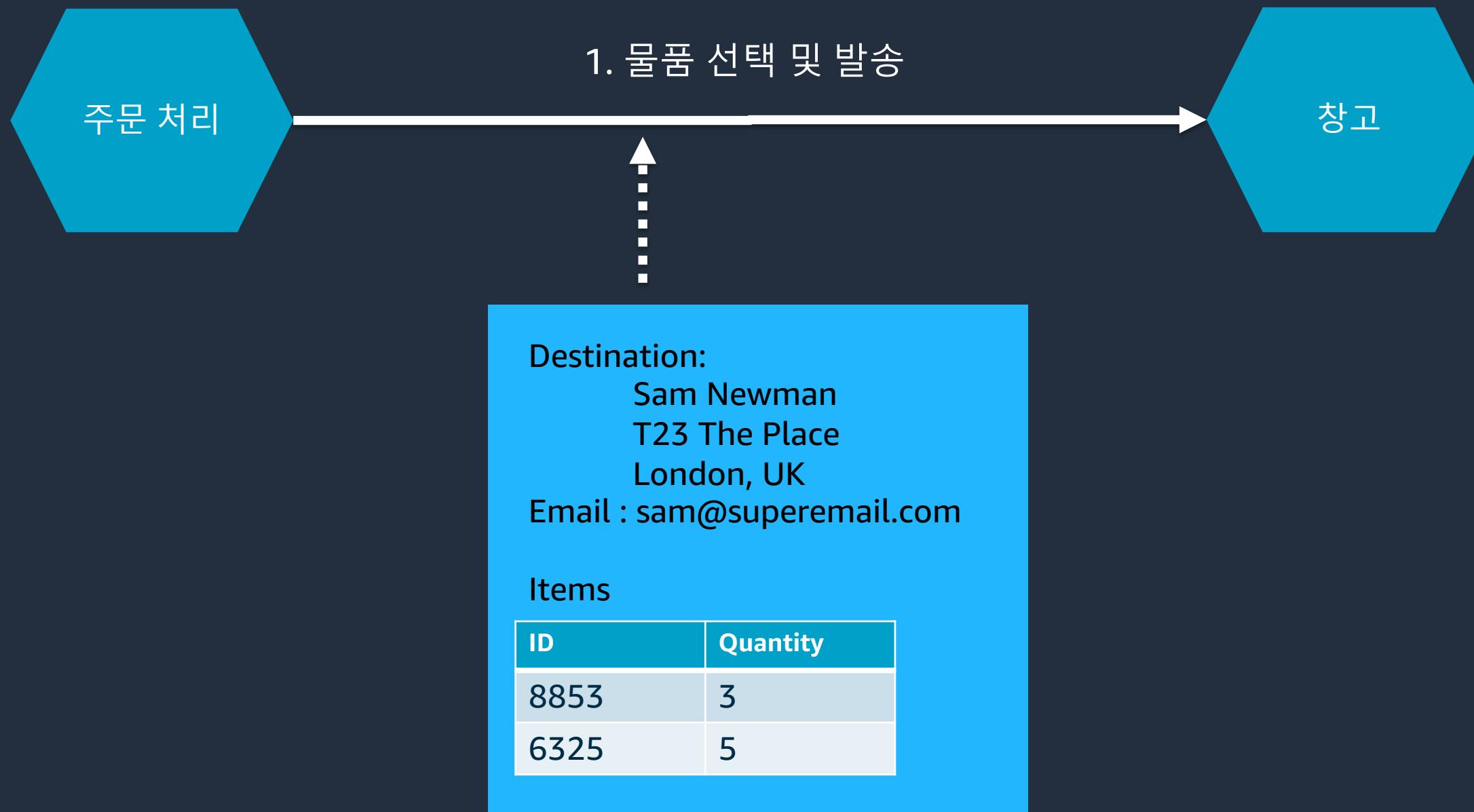
# 커플링의 종류 - (4) 도메인 결합도(Domain Coupling)



# 커플링의 종류 - (4) 도메인 결합도(Domain Coupling)



# 커플링의 종류 - (4) 도메인 결합도(Domain Coupling)

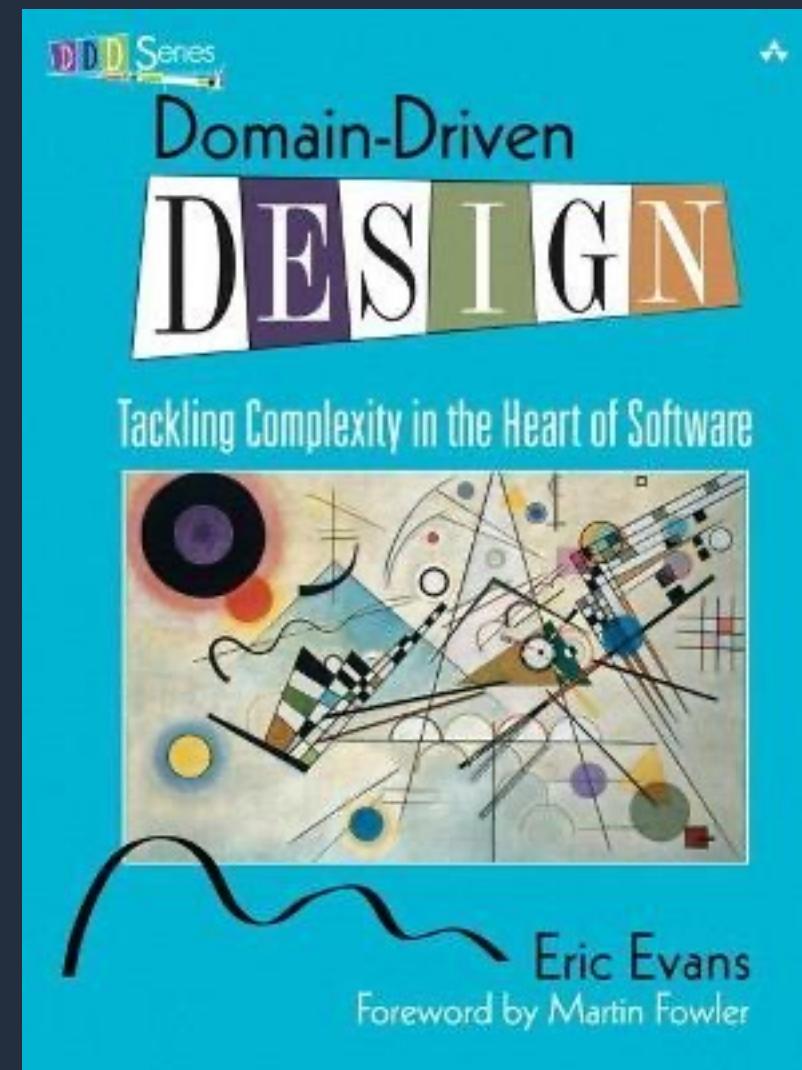


물품 선택 명령

# 커플링의 종류 - (4) 도메인 결합도(Domain Coupling)



# 도메인 주도 설계( Domain Driven Design)



# マイグレーション 계획



# マイクロサービス를 선택하는 이유는 무엇입니까?

- 팀 자율성 향상
- 출시 시간 단축
- 대량의 트래픽에 대한 비용 효율적인 확장
- 견고성 향상
- 개발자 수 늘리기

# マイクロ서비스를 선택하는 이유는 무엇입니까?

- 불분명한 도메인
- 스타트업
- 고객 설치형 소프트웨어, 관리형 소프트웨어
- 좋은 이유를.. 딱히..찾을 수 없다...

# 점진적인 마이그레이션의 중요성

“빅뱅 방식으로 백날 해봐야 얻는 것은  
빅뱅뿐이다..”

– Martin Fowler

# 변경 비용

호스팅 회사 변경

공개 API 변경

전체 조직에 걸쳐  
새로운 프로그래밍 언어 채택

데이터베이스 변경

새로운 프로그래밍  
언어 실행

오픈소스 라이브러리 사용

비가역적

전반적인 승인 필요

의사결정을 변경하기가 거의 불가능

신중한 생각이 필요

가역적

더 적은 승인

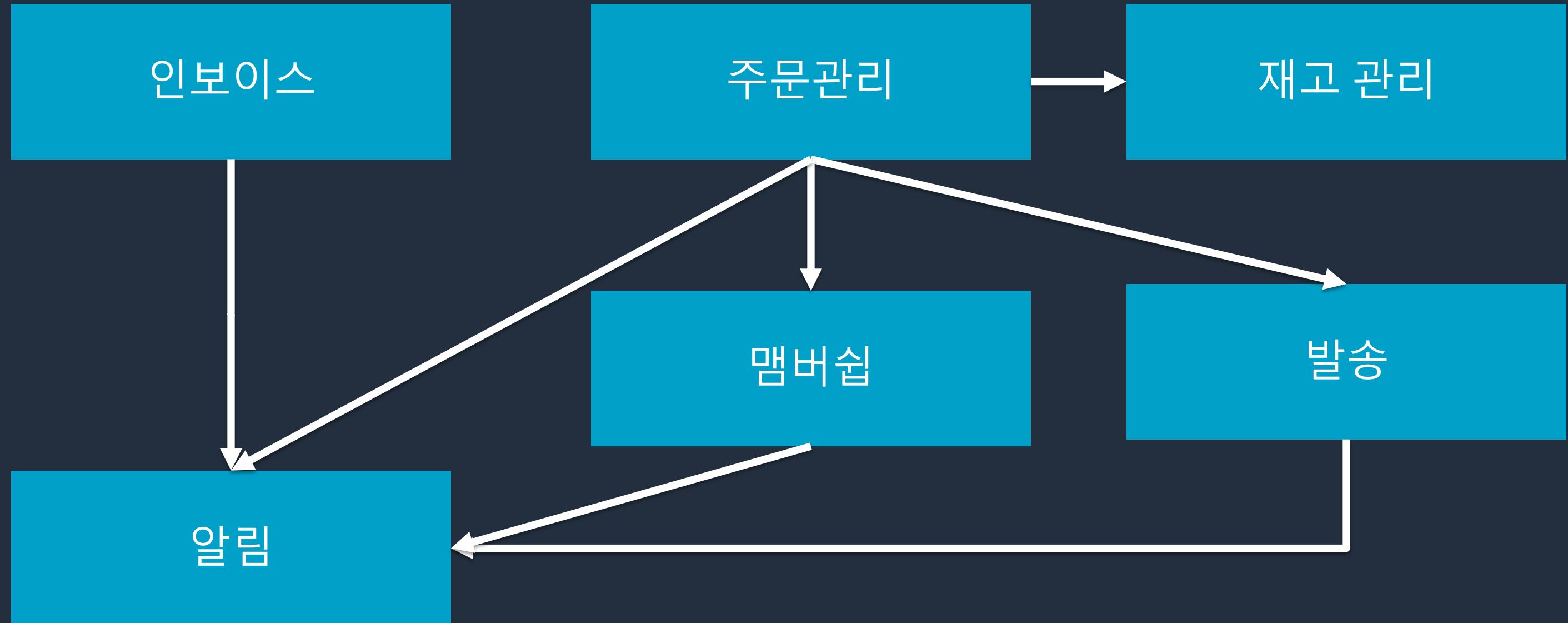
의사결정을 번복하는 비용이 적음

즉흥적인 의사결정도 가능

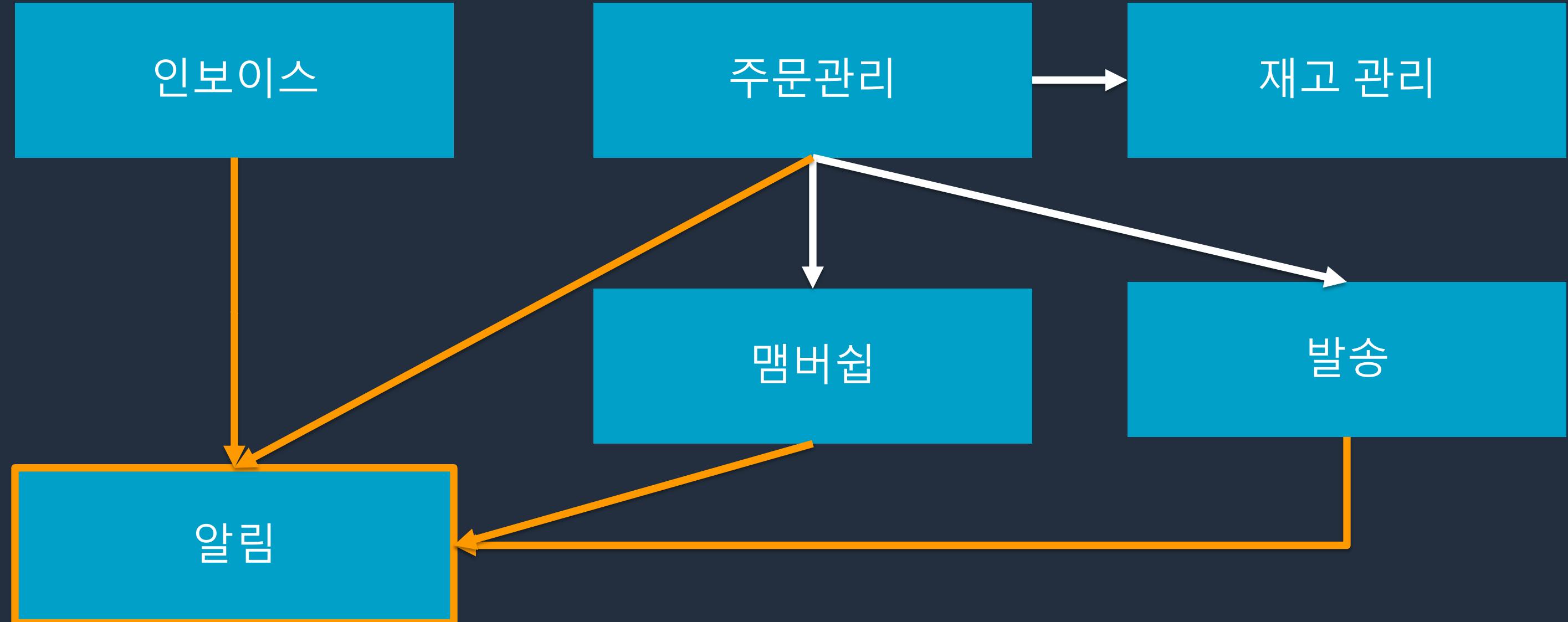
1-way door

2-way door

# 도메인 주도 설계



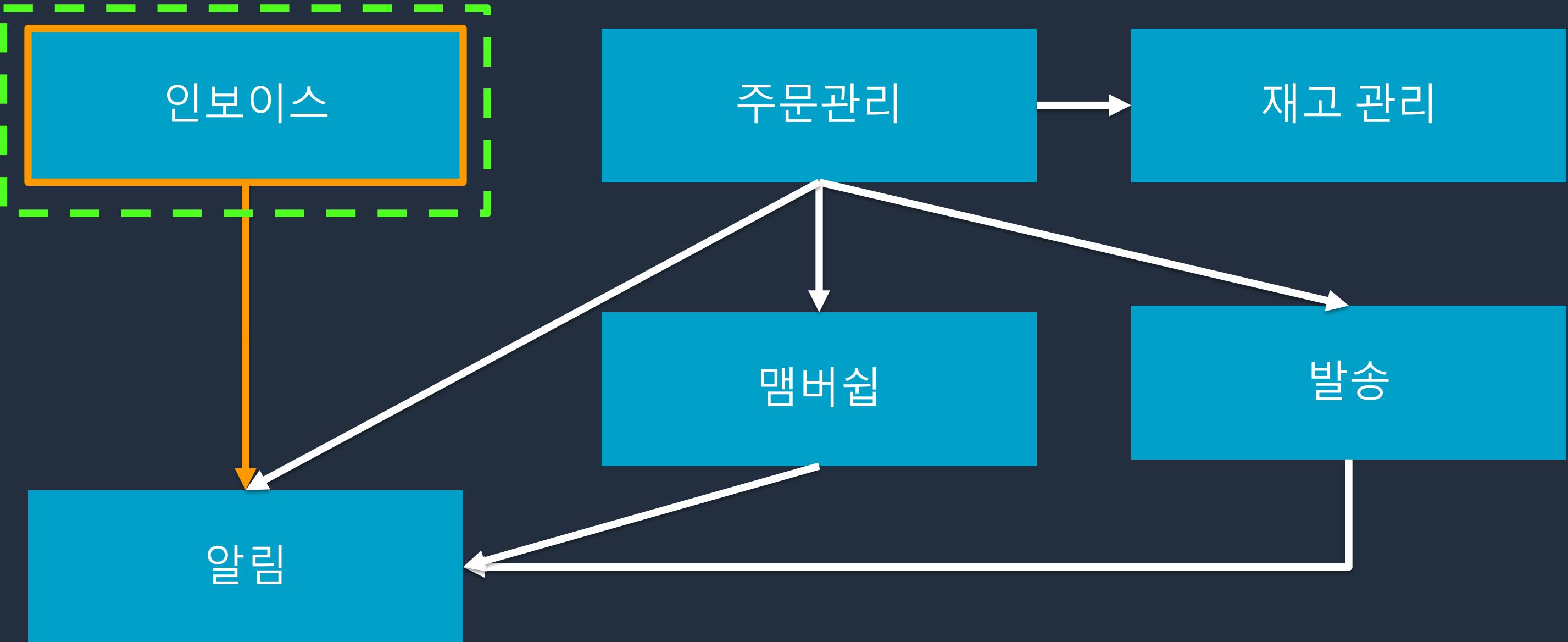
# 도메인 주도 설계



알림을 일종의 서비스로 호출하는 작업은 업스트림의 의존성과  
코드 변경이 필요할지도 모릅니다.

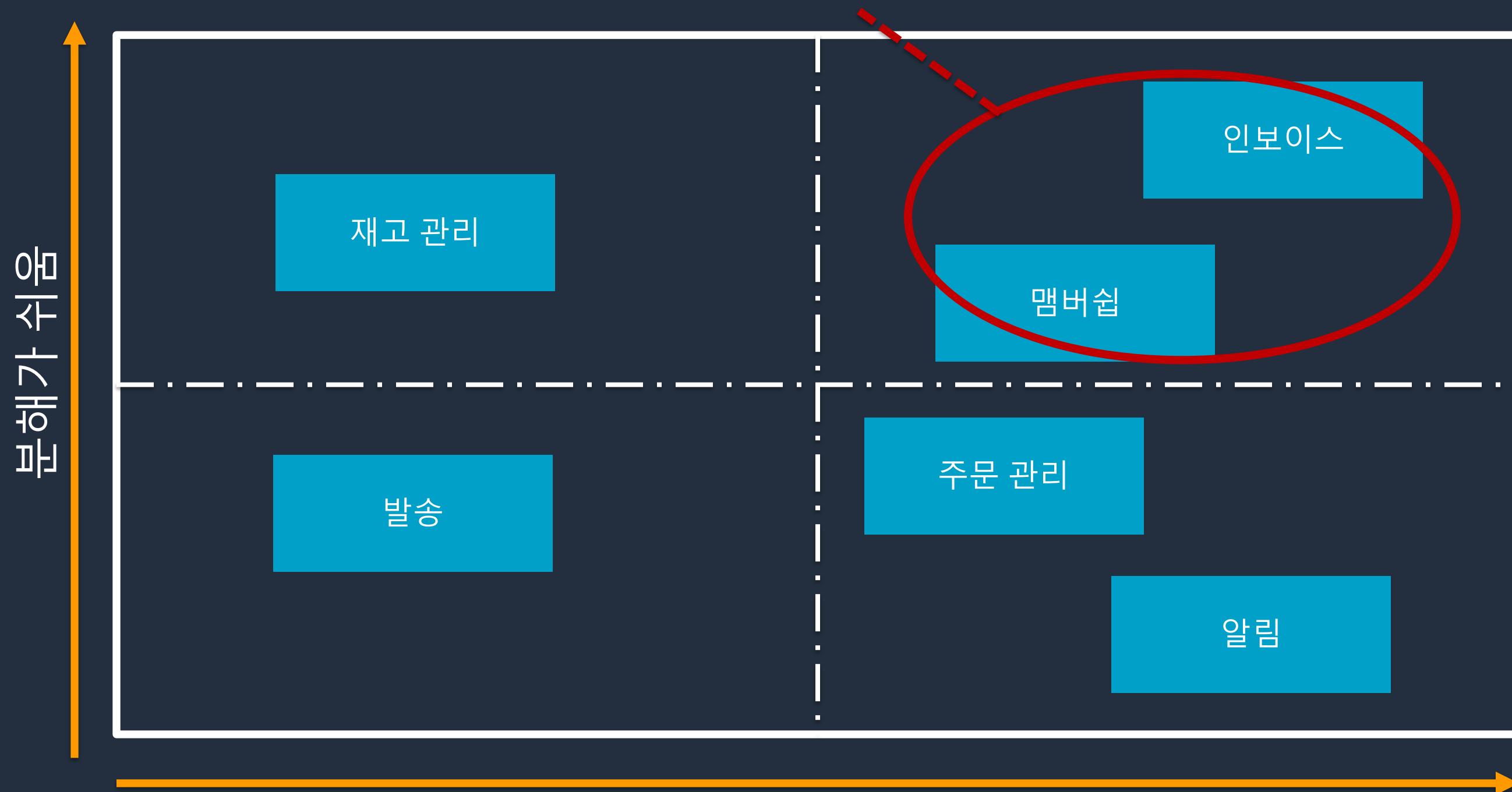
# 도메인 주도 설계

Potential for use of a strangler fig pattern to aid decomposition



# 결합 모델

## 분해하기 좋은 서비스 후보들



# 팀 재구성

비  
중  
씨  
비

GUI 프로토타이핑  
프론트엔드 개발

비  
중  
씨

백엔드 개발  
자동화된 기능 테스트

보  
야

테스트 환경 프로비저닝  
부하 테스트  
24/7 사고 대응  
운영환경 배포

팀 통합

비  
중  
씨

GUI 프로토타이핑  
프론트 엔드 개발  
백 엔드 개발  
자동화된 기능 테스트  
테스트환경 프로비저닝  
근무시간 내 사고 대응

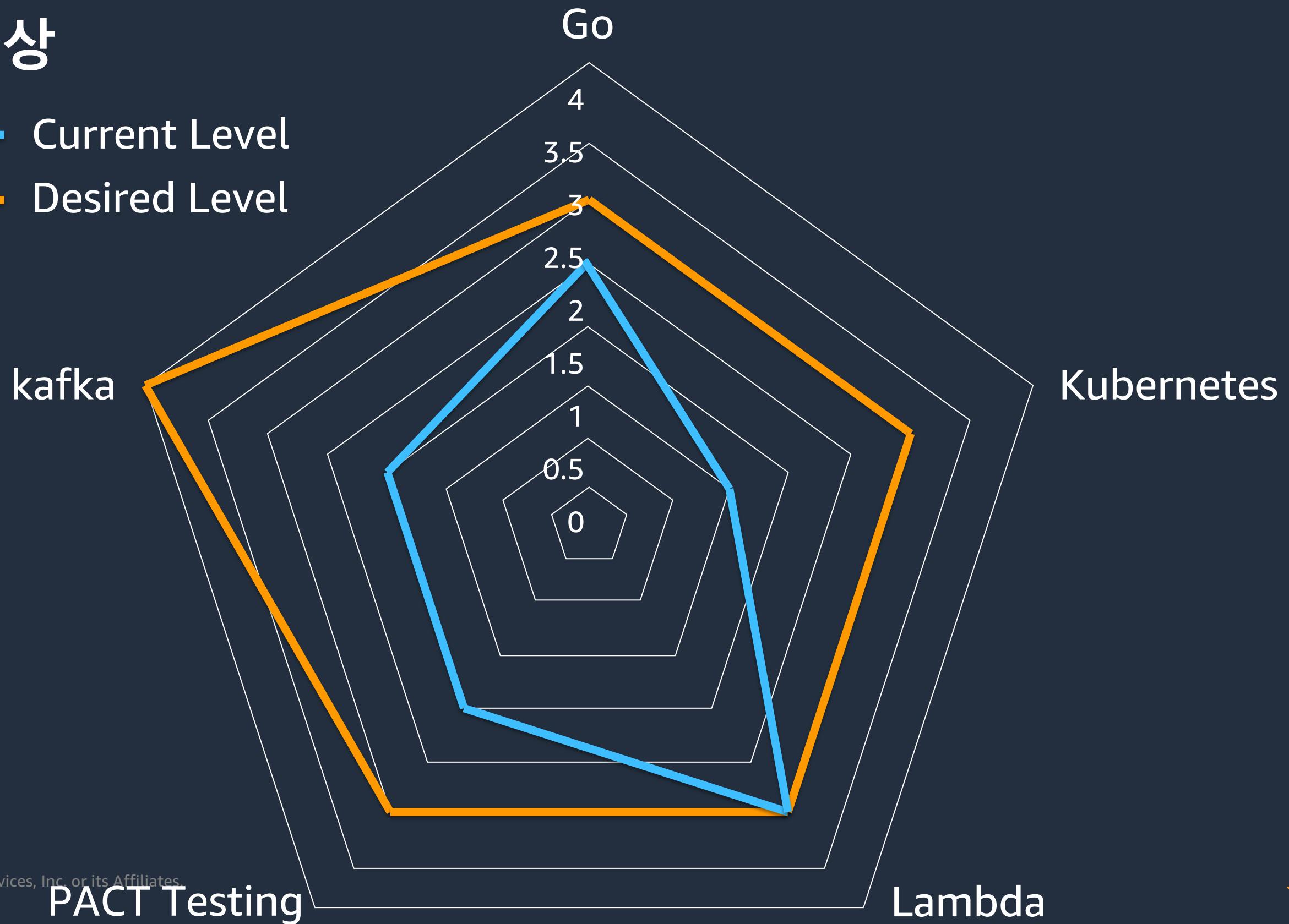
보  
야

マイ그레이션  
시작

일과 시간 외 사고 대응  
부하 테스트  
운영 환경 배포  
셀프 서비스 플랫폼

# 기술 향상

Current Level  
Desired Level



# マイクロサービス로의 전환이 잘되는지 어떻게 알 수 있습니까?

- 정기적인 점검사항 보유
- 정량적 측정
- 정성적 조치
- 매몰비용의 오류 피하기
- 새로운 접근 방식에 개방적

# マイクロサービス パターン



# Strangler Fig 패턴



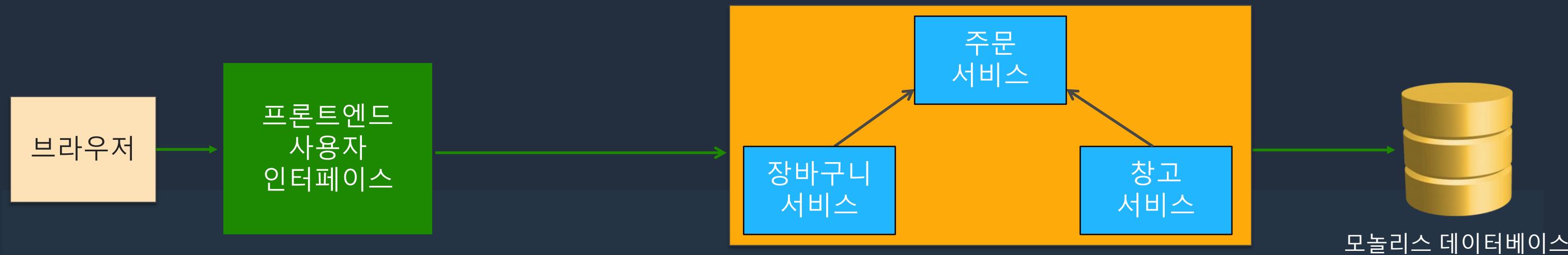
Source: [https://en.wikipedia.org/wiki/Strangler\\_fig](https://en.wikipedia.org/wiki/Strangler_fig)



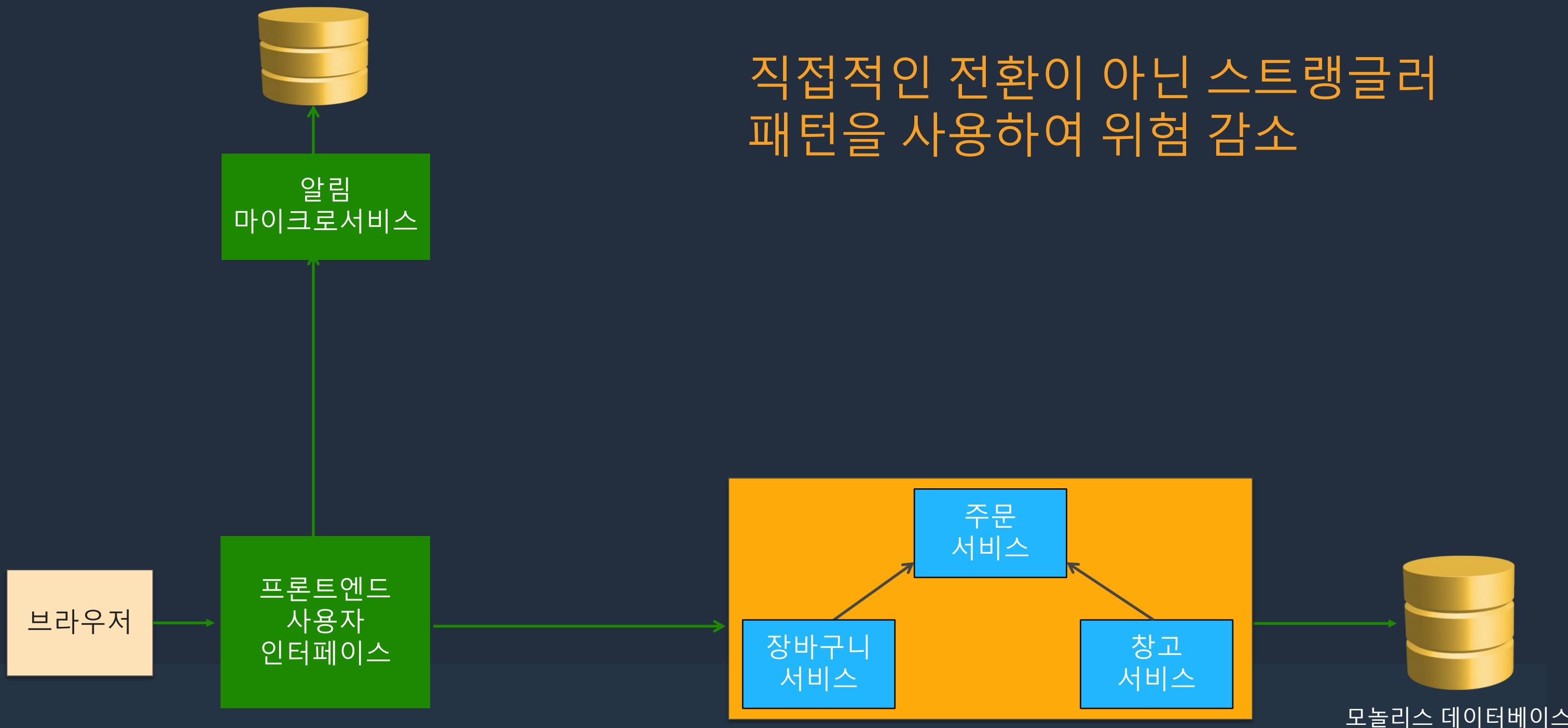
Source: <https://martinfowler.com/bliki/StranglerFigApplication.html>

# 모놀리스에 적용한 스트랭글러 패턴

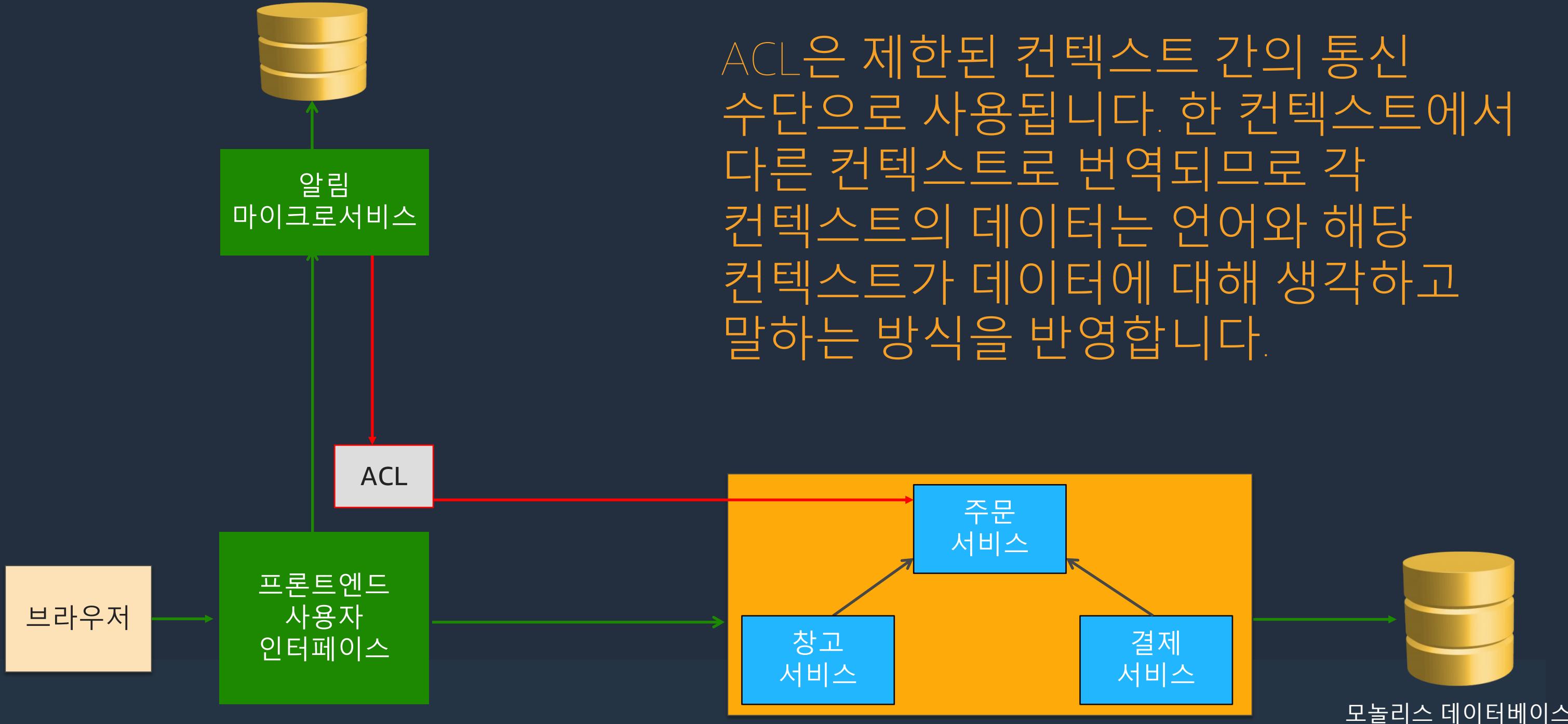
직접적인 전환이 아닌 스트랭글러 패턴을 사용하여 위험 감소



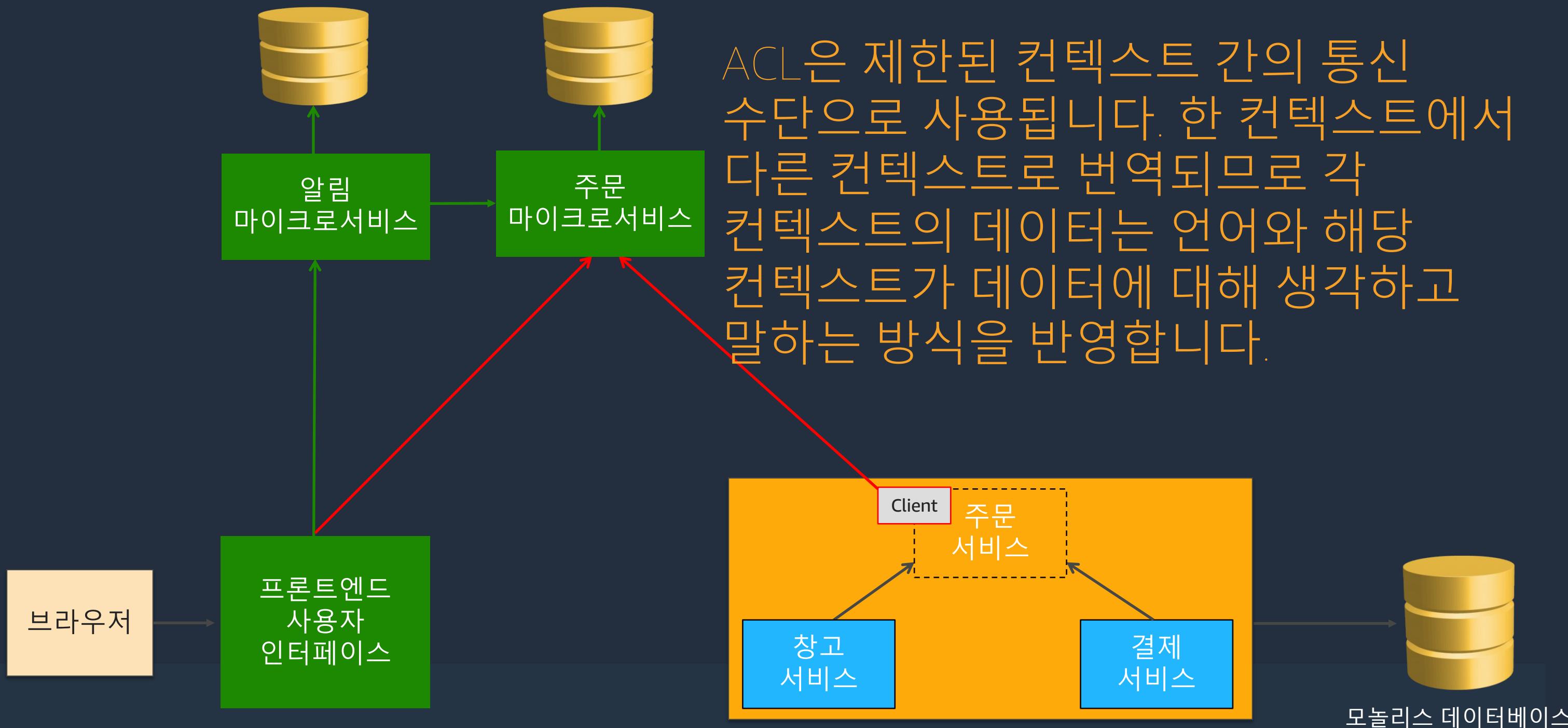
# 모놀리스에 더 이상 새로운 기능을 추가하지 않습니다.



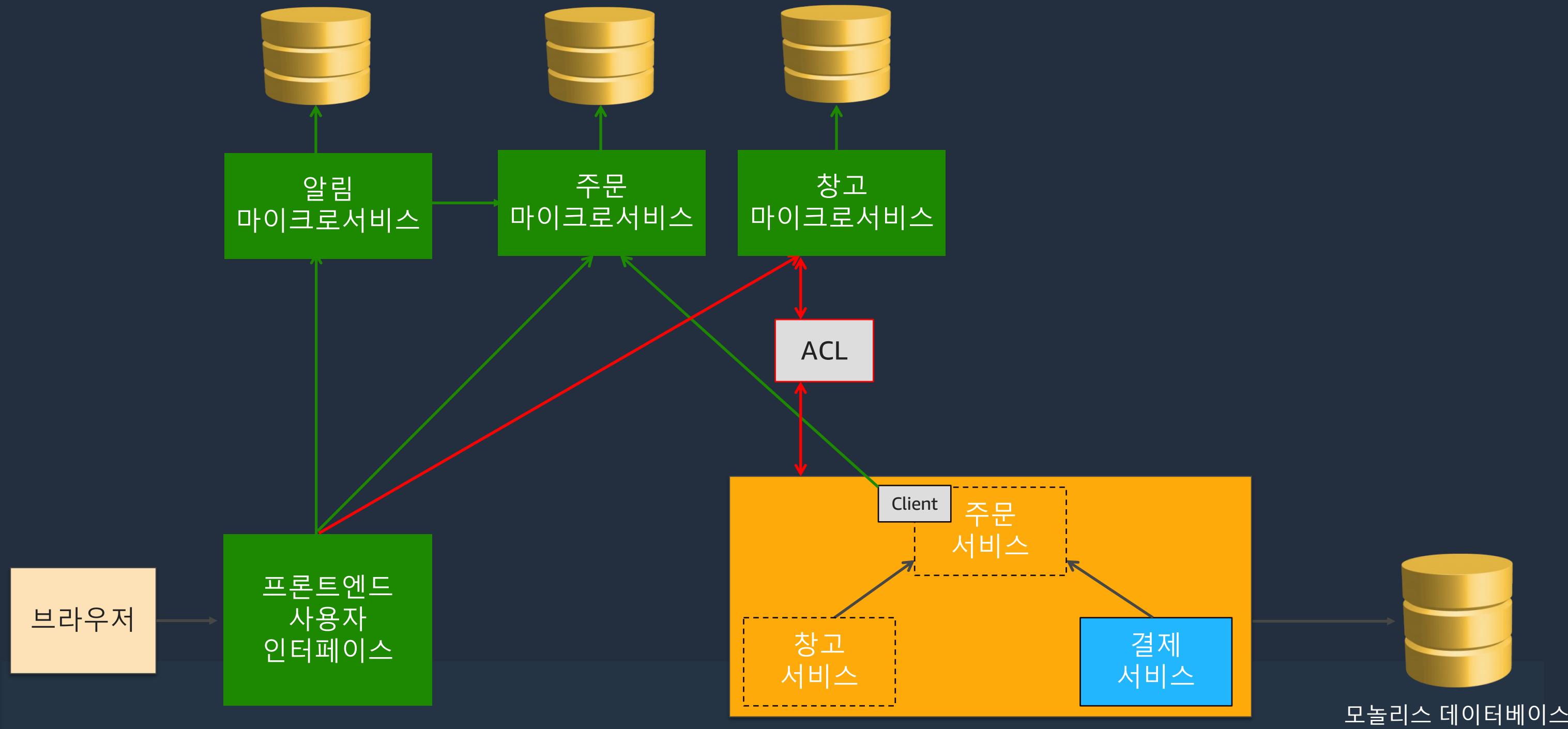
# 통합을 위한 손상 방지 계층(Anti Corruption Layer)



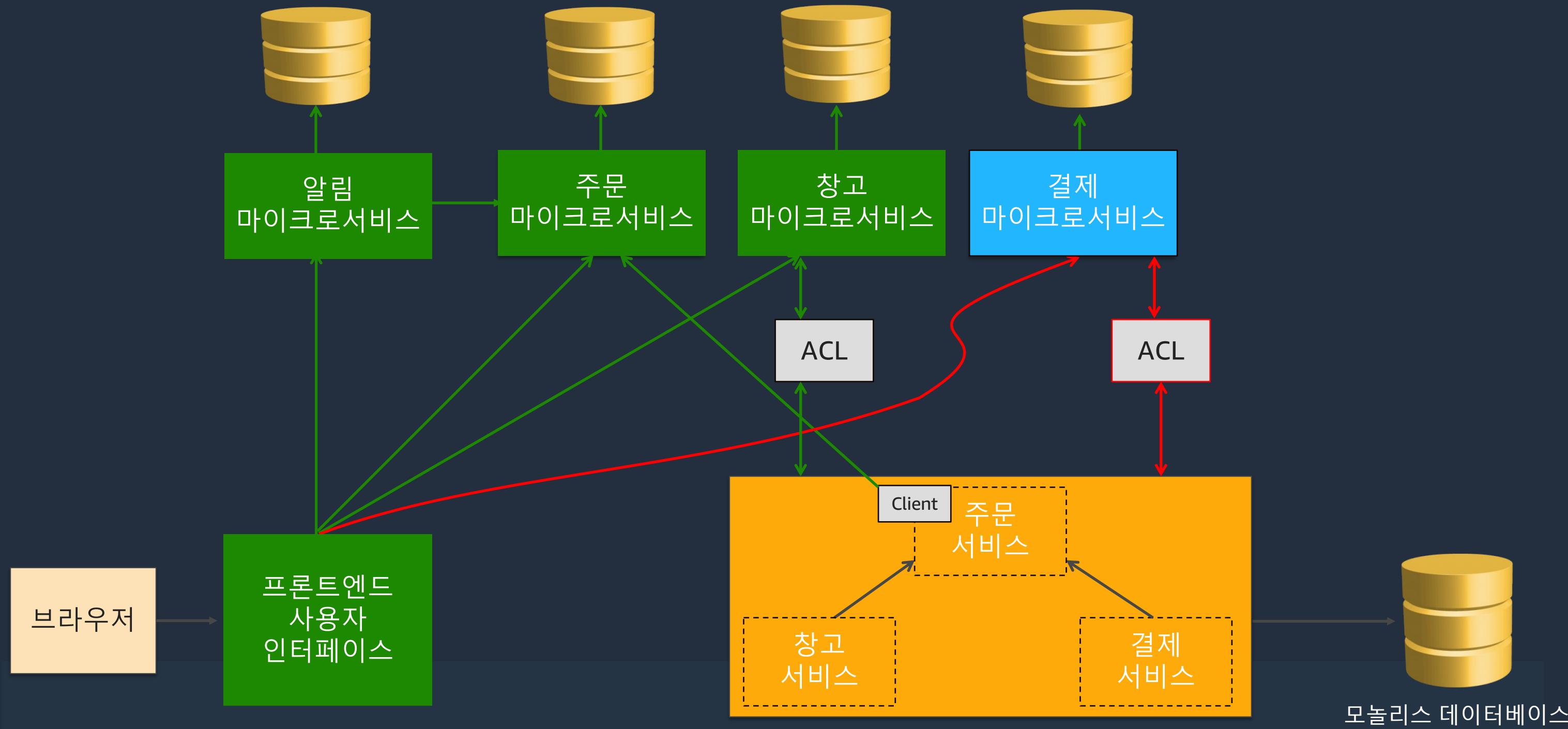
# 간단한 기존 서비스로 시작



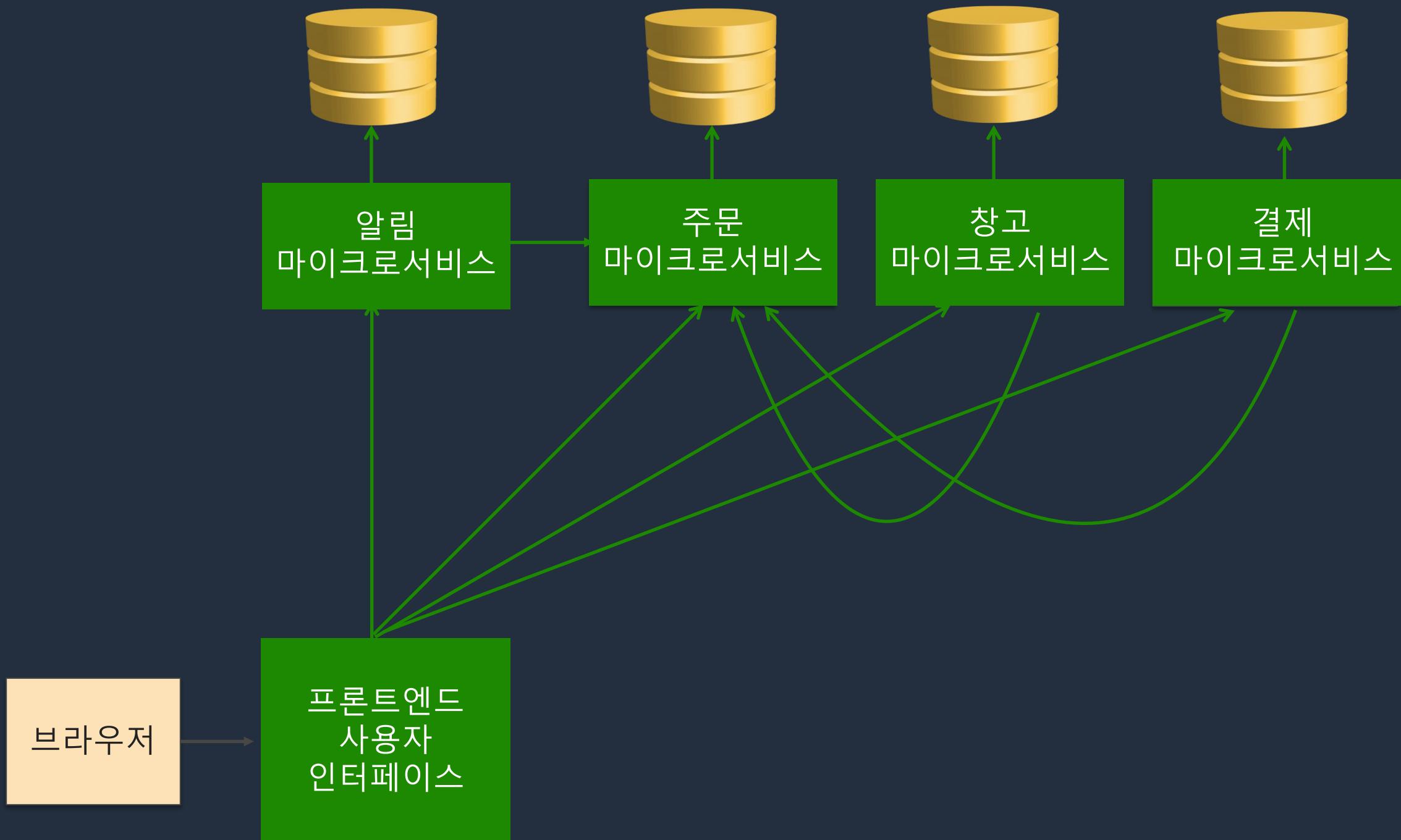
# 더 복잡한 서비스로 이동



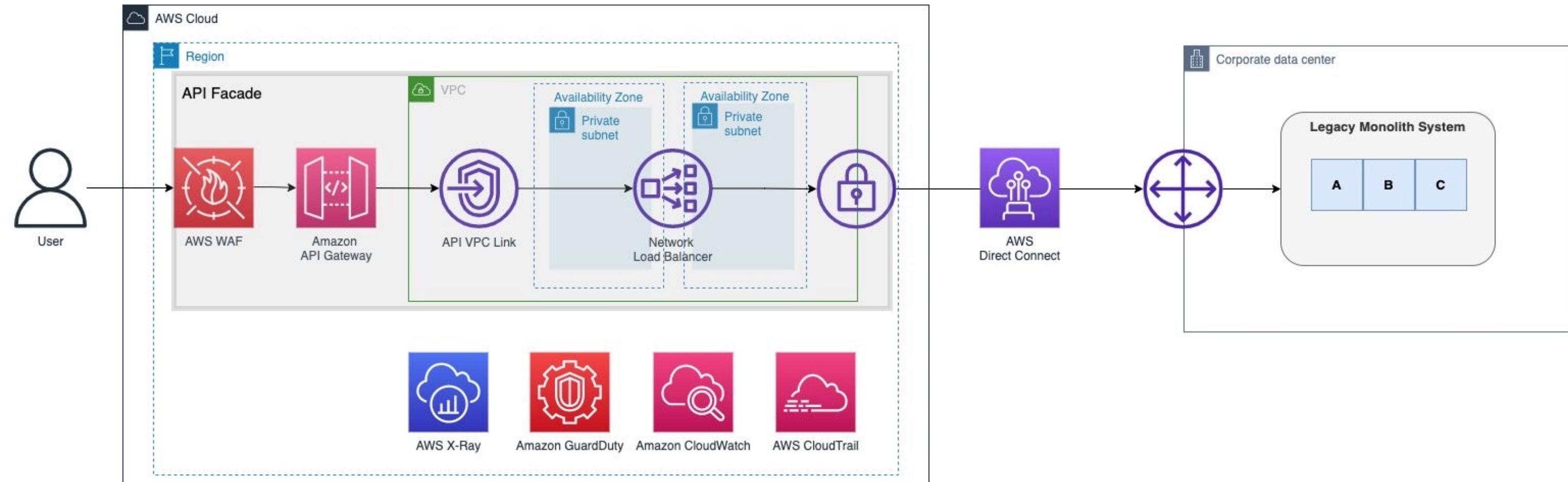
# 더 복잡한 서비스로 이동



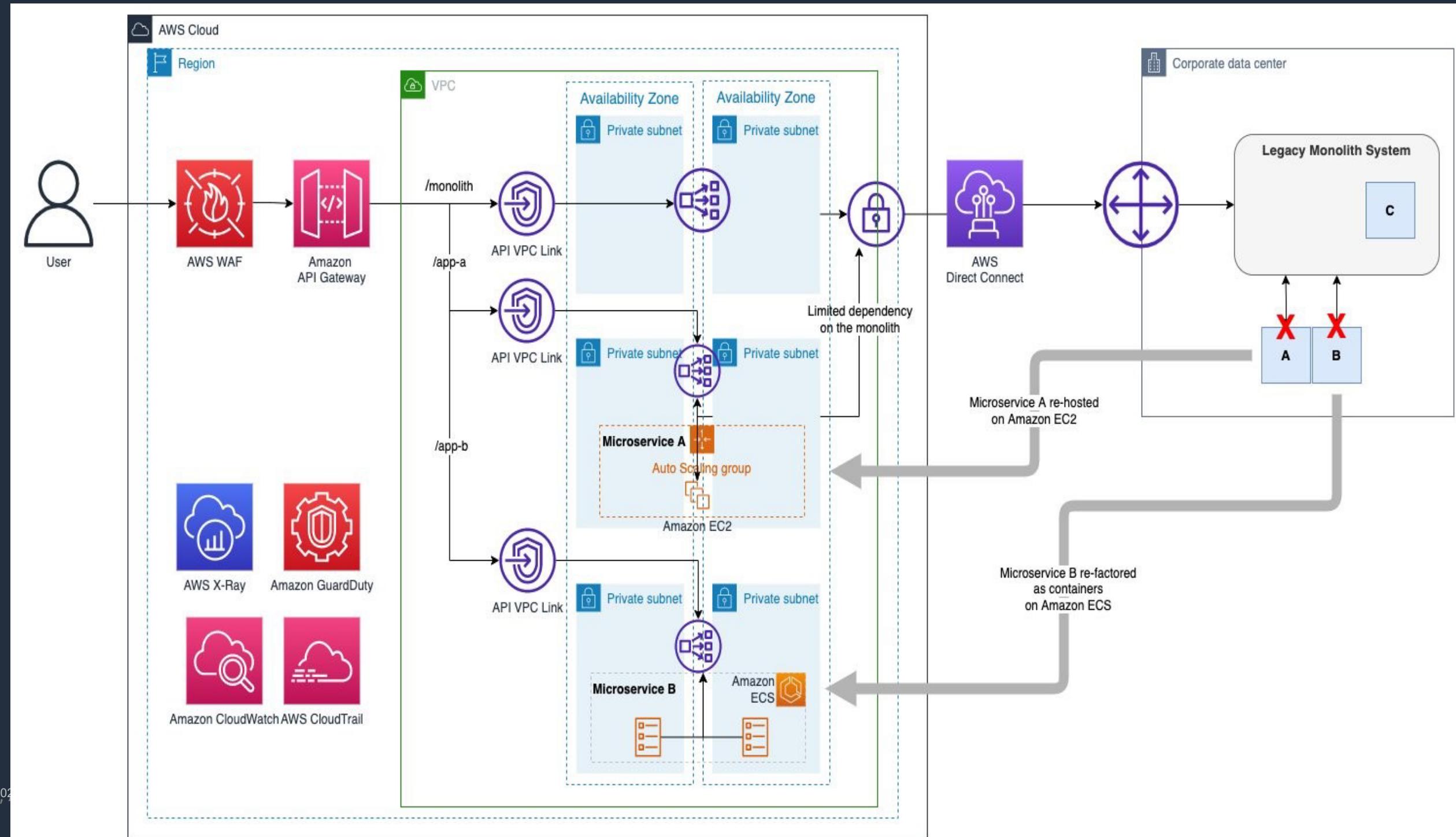
# 종료 상태



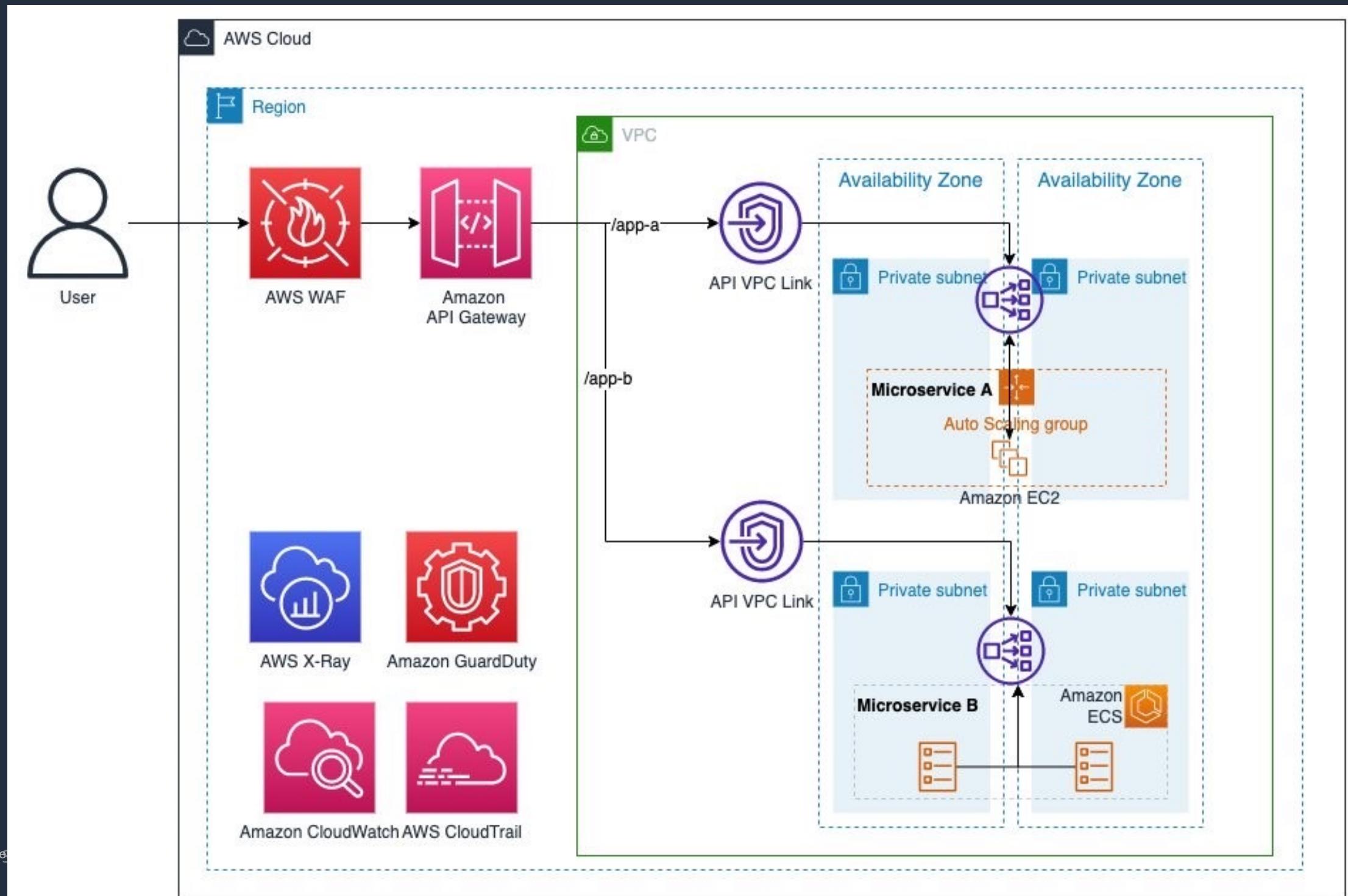
# AWS 위에서는 어떻게 동작할까요?



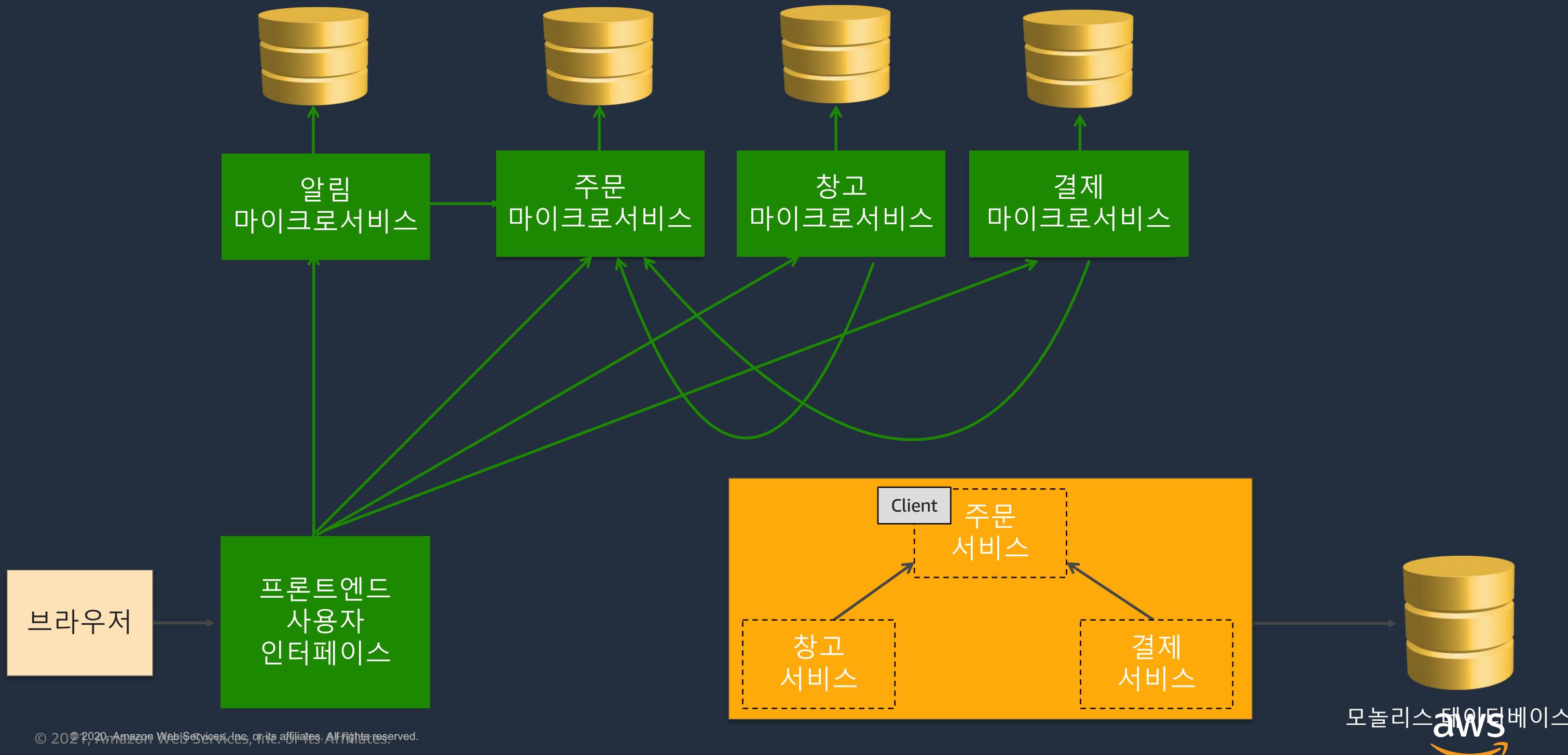
# AWS 위에서는 어떻게 동작할까요?



# AWS 위에서는 어떻게 동작할까요?



# 분리는 했는데 그럼 뭐가 또 문제일까?

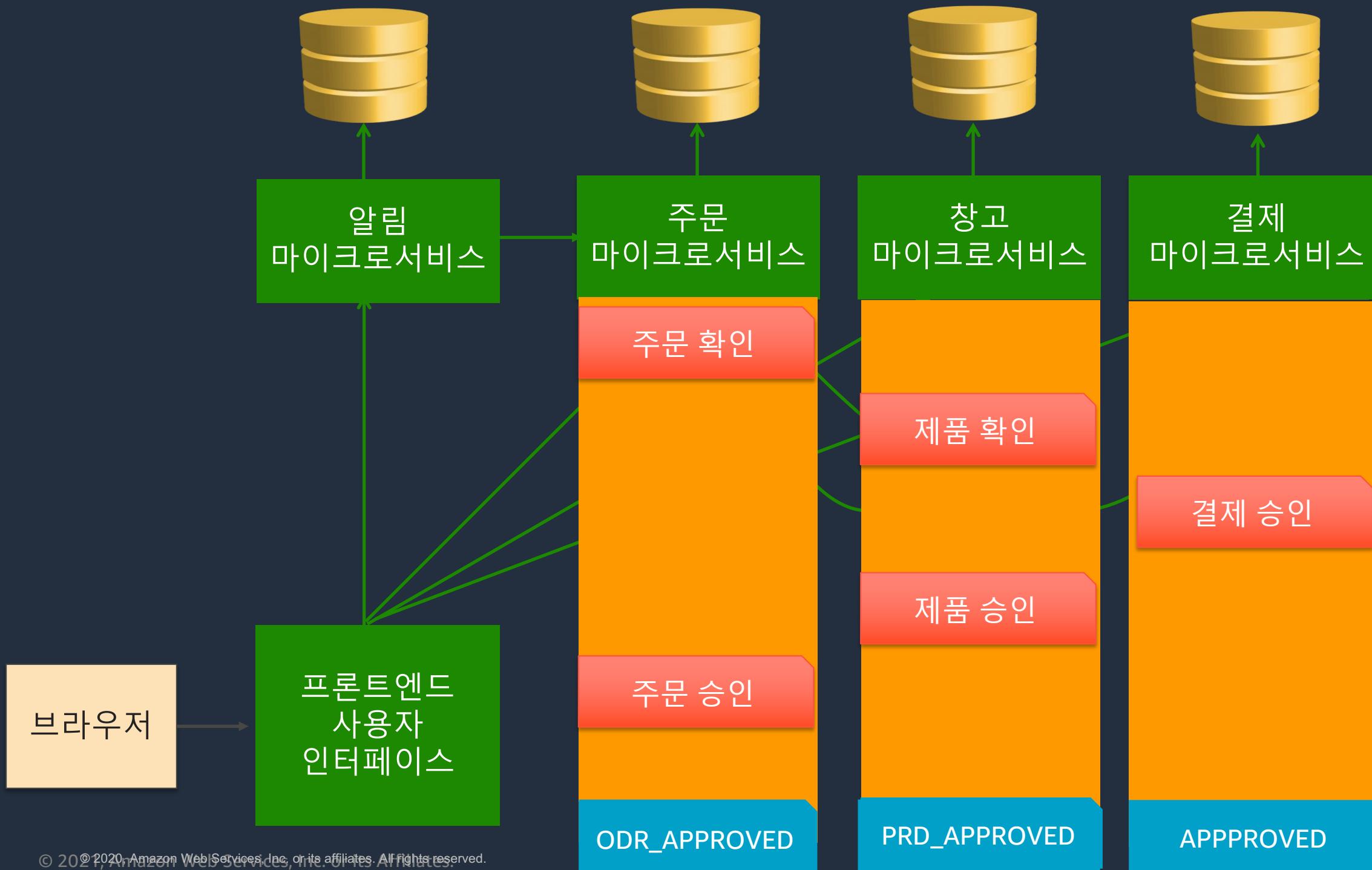


# SAGA 패턴(트랜잭션 관리)

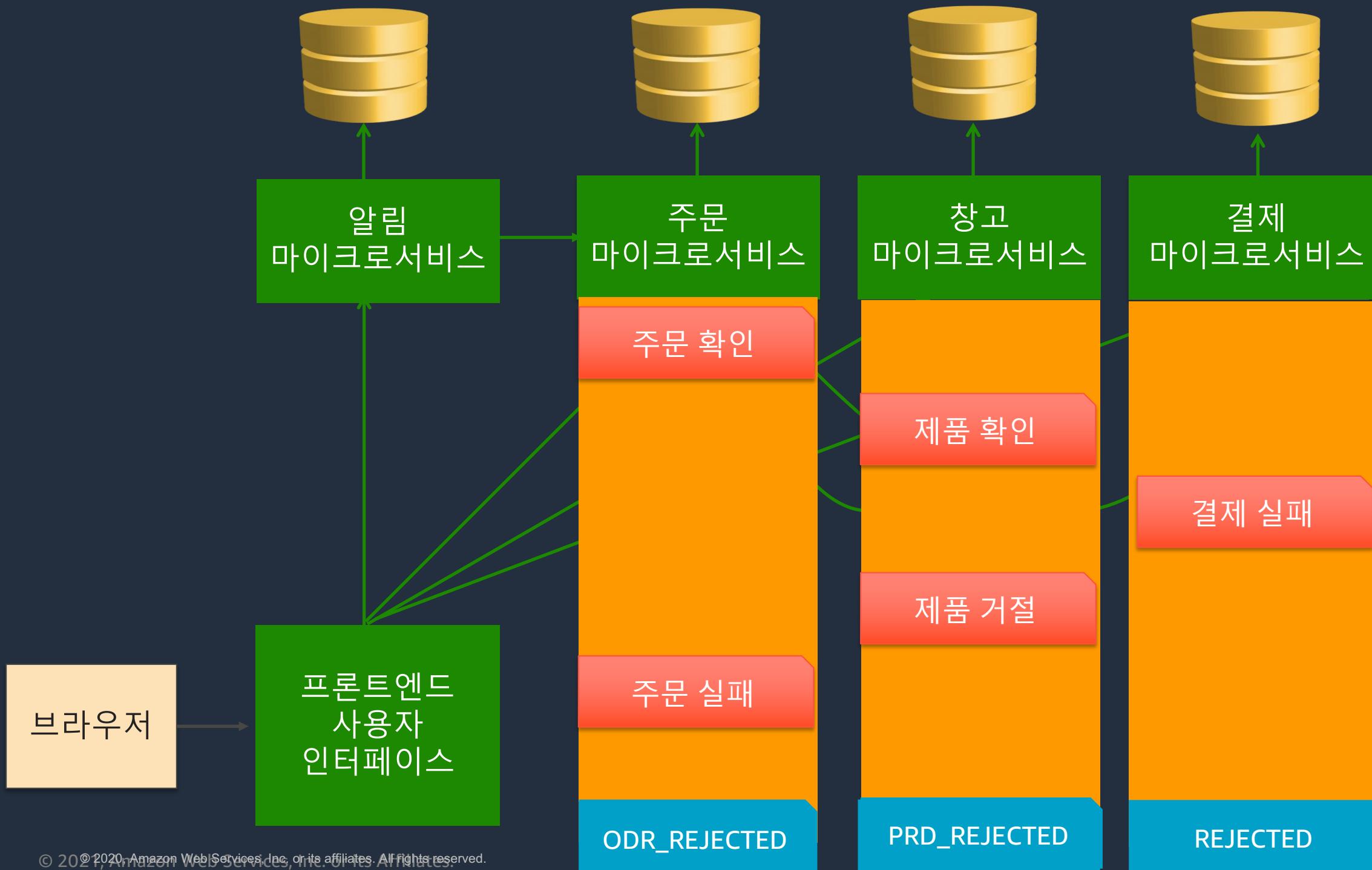


기존엔 단일 데이터베이스에서 ...  
하지만 나눠진 데이터베이스에서 트랜잭션은?

# SAGA 패턴(트랜잭션 관리 - 성공)



# SAGA 패턴(트랜잭션 관리 - 실패)

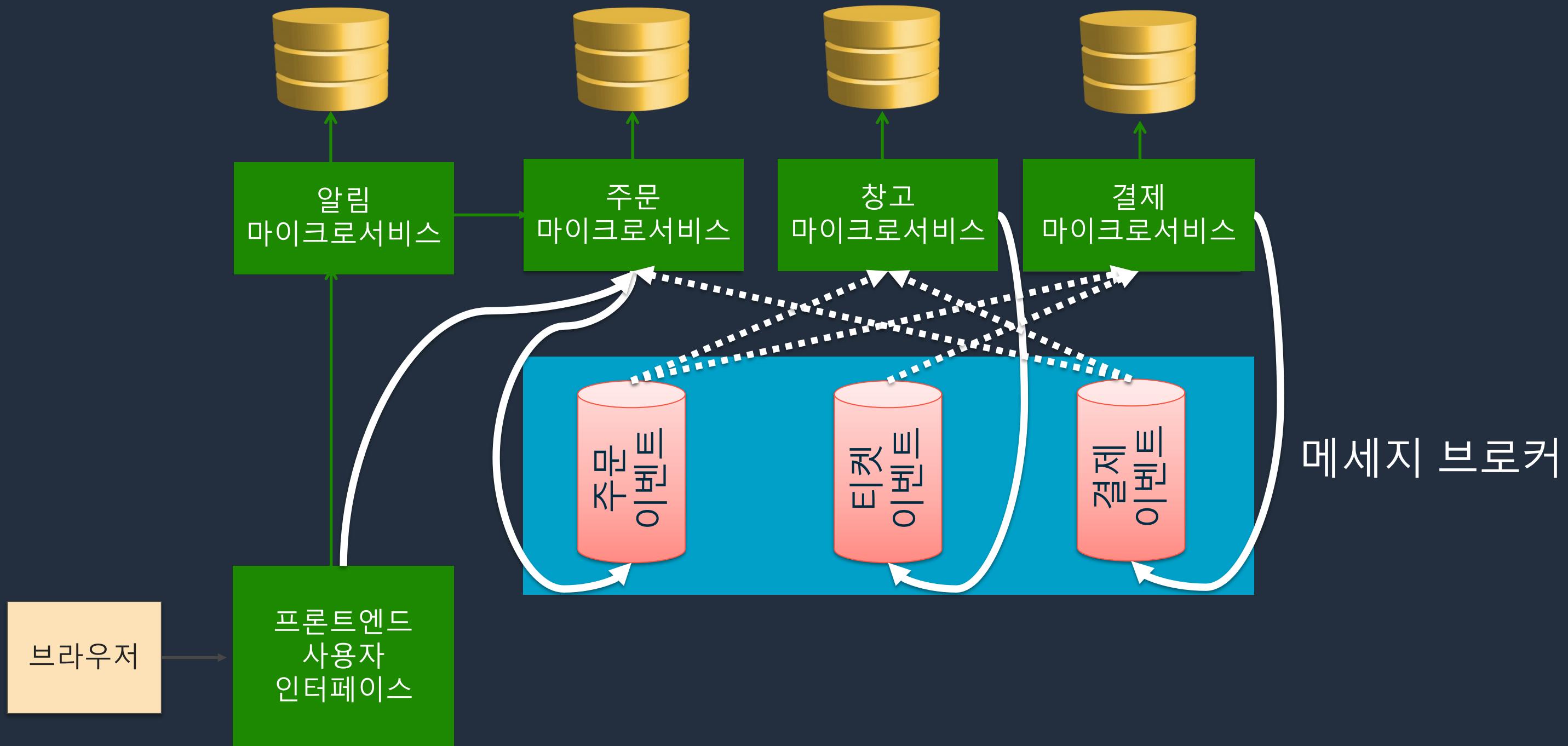


# SAGA 패턴의 종류

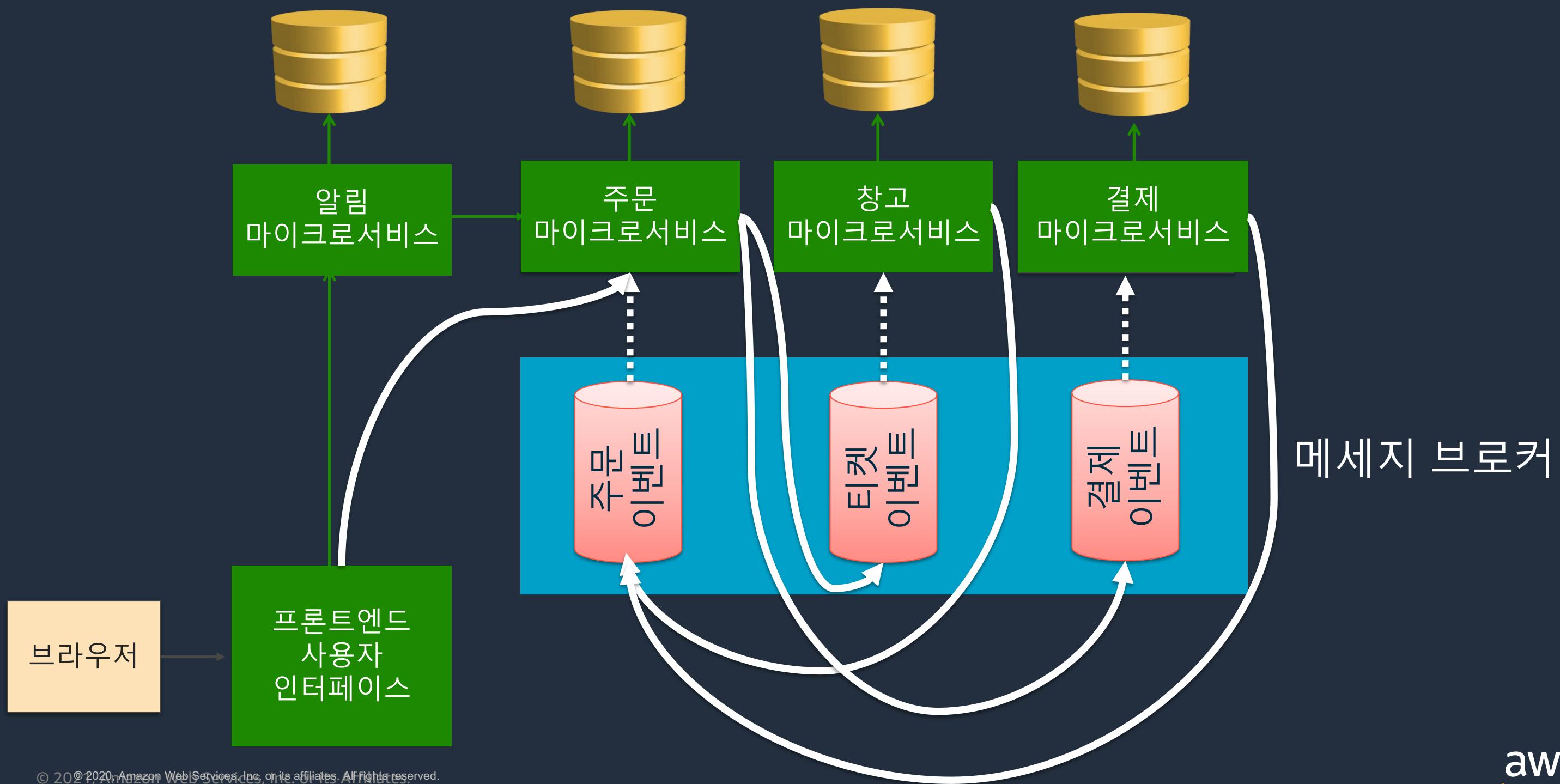
코레오그래피(Choreography)  
의사 결정과 순서를 SAGA 참여자에게 맡김

오케스트레이션(Orchestration)  
SAGA 편성 로직을 오케스트레이터에 중앙화

# SAGA 패턴(코레오그래피)



# SAGA 패턴(코레오그래피)



# SAGA 패턴의 상태머신



# AWS Step Functions

Fully-managed state machines on AWS

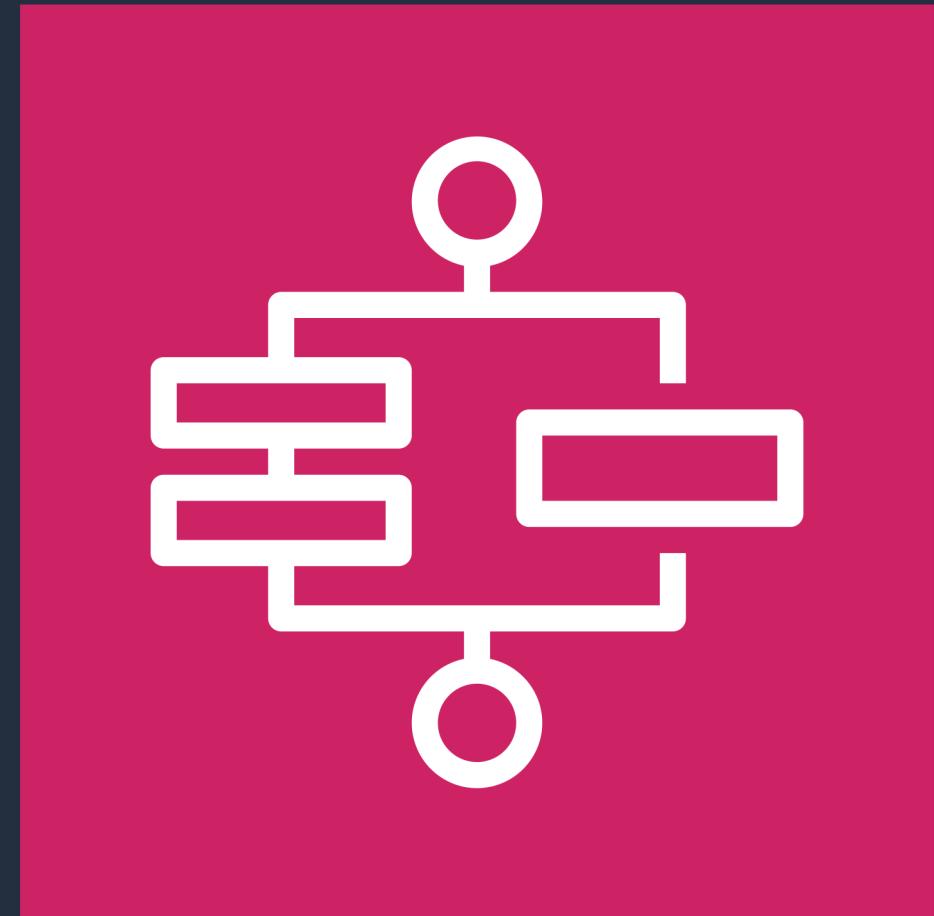
탄력적인 워크플로 자동화

내장 오류 처리

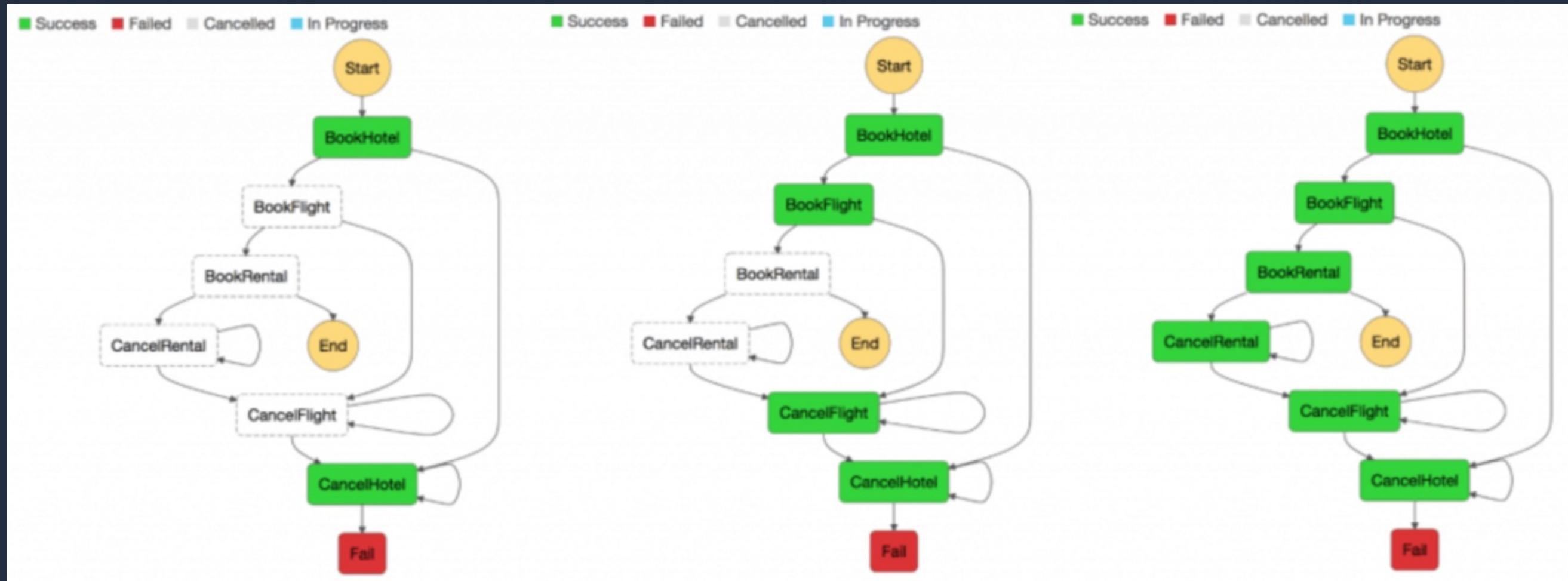
강력한 AWS 서비스 통합

자체 서비스와의 통합을 위한 최고 수준의 지원

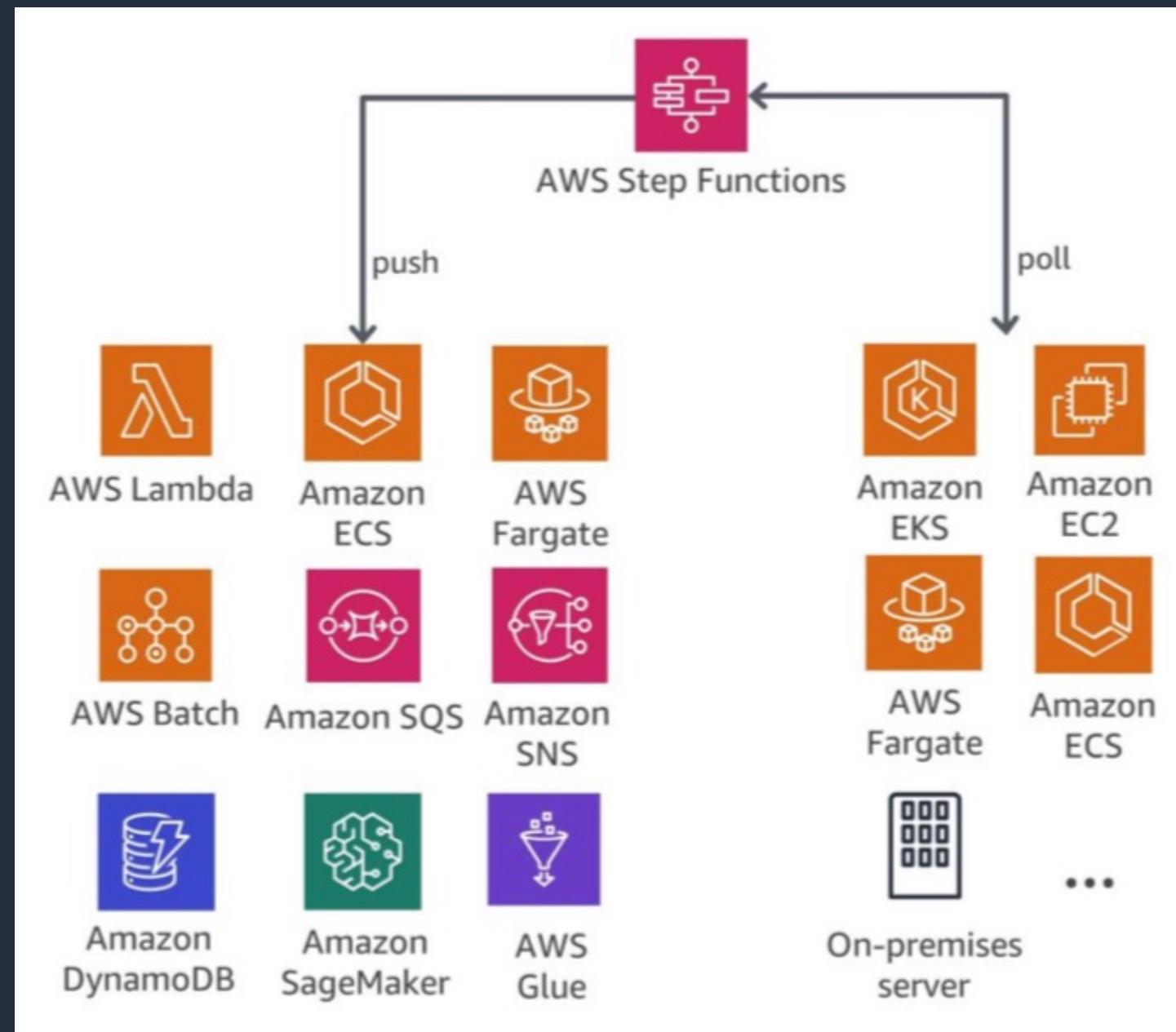
감사 가능한 실행 이력 및 시각적 모니터링



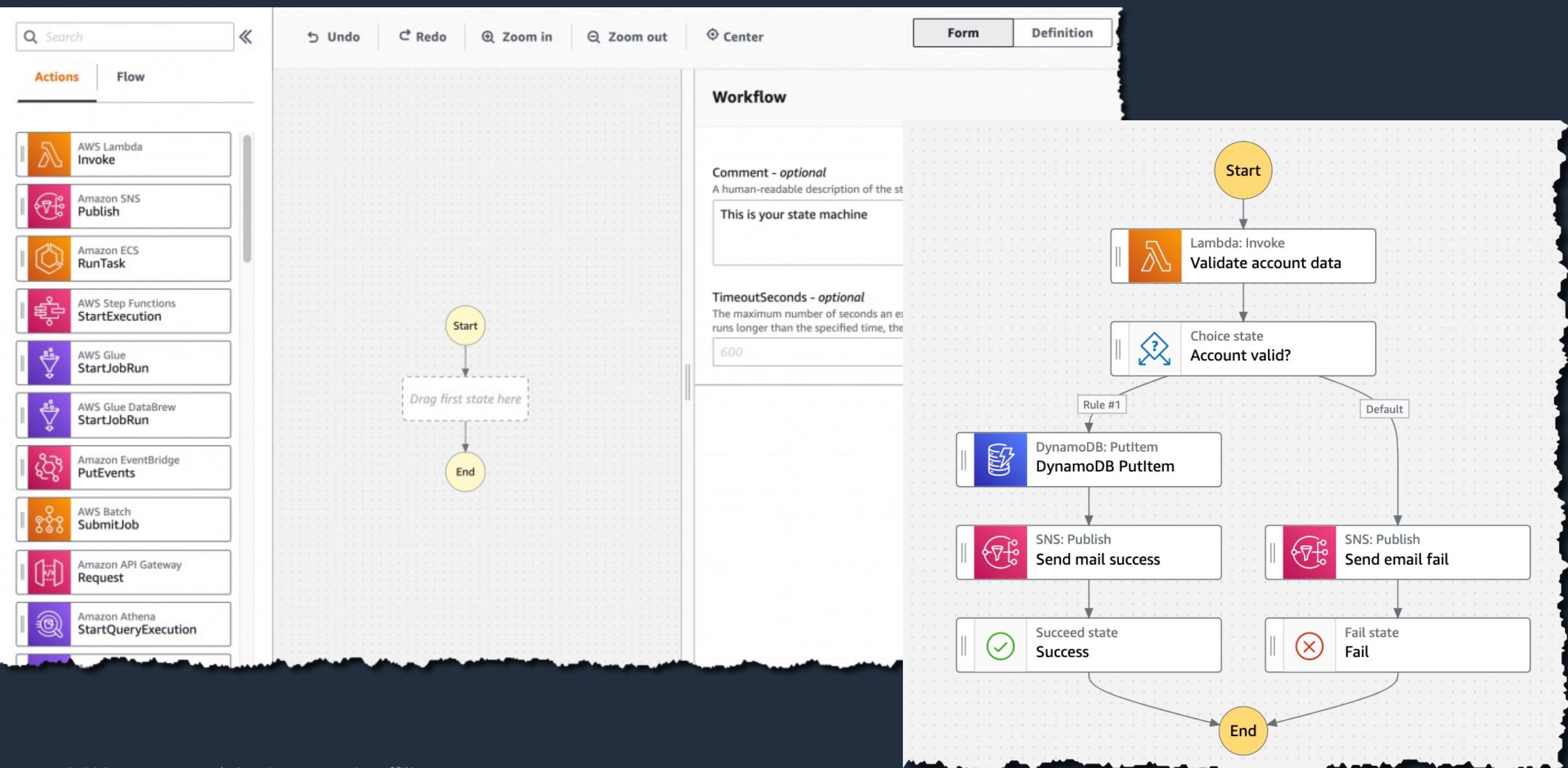
# Saga coordinator – AWS Step Functions



# Saga coordinator – AWS Step Functions



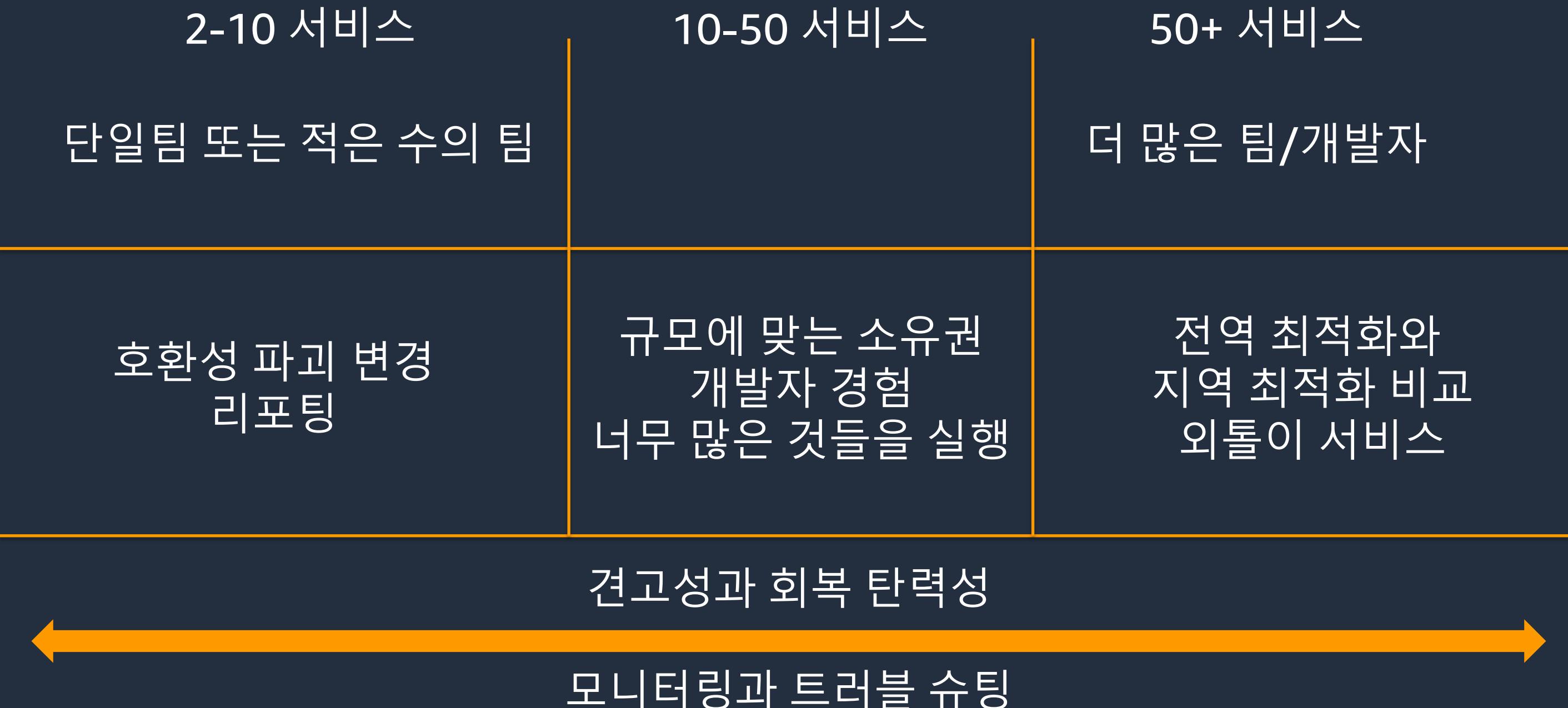
# AWS Step Functions 작업



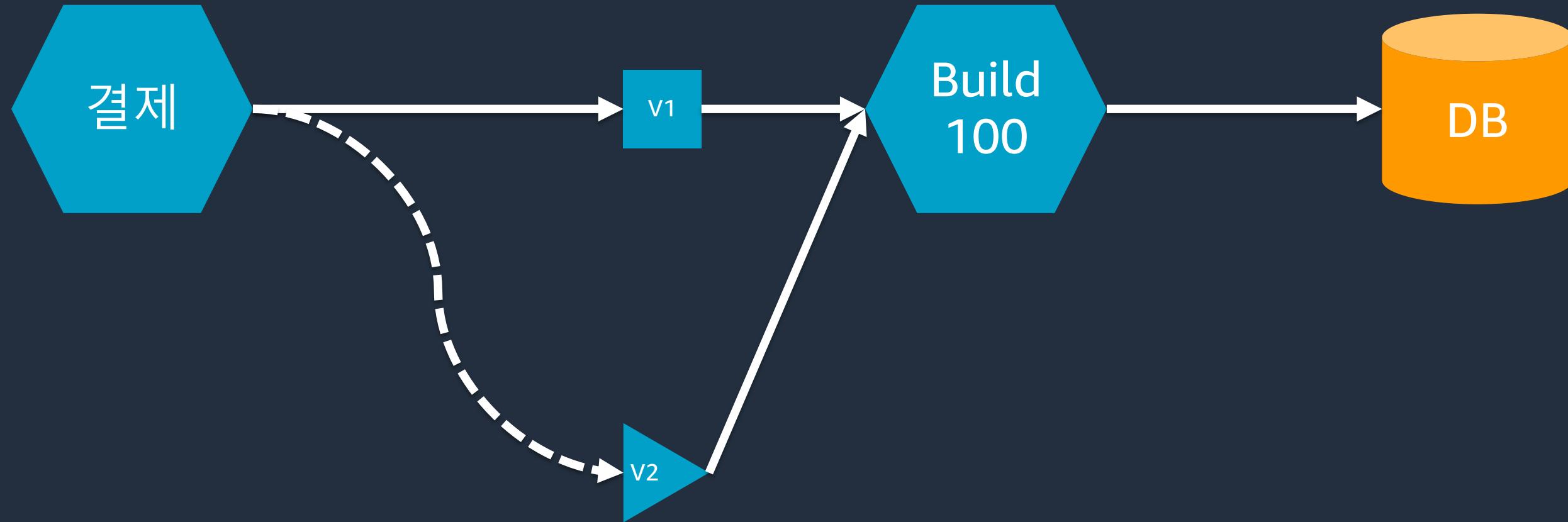
# 서비스가 늘어날 수록 커지는 고통



# 더 많은 서비스와 더 많은 고통

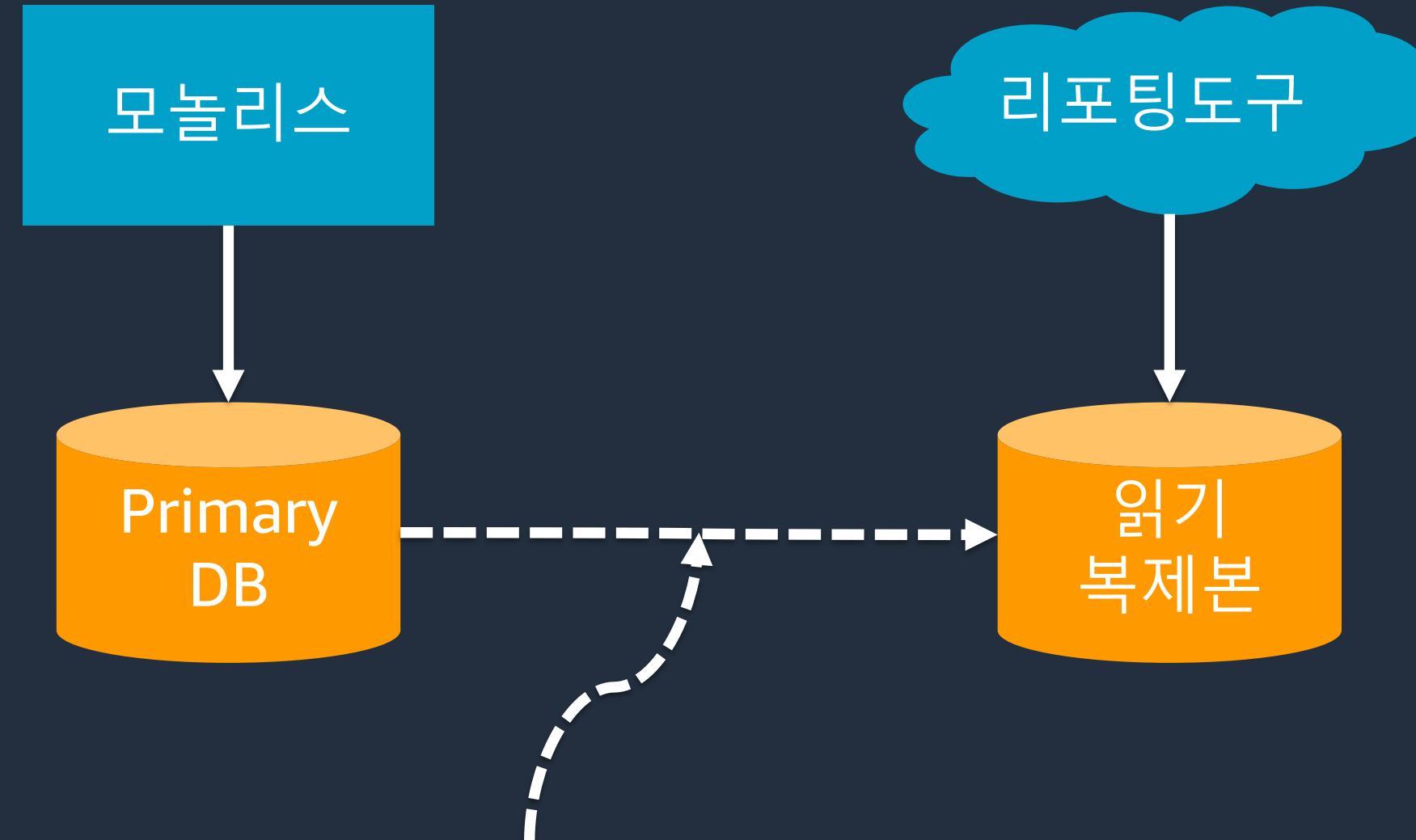


# 기존의 호환성을 깨뜨리는 변경

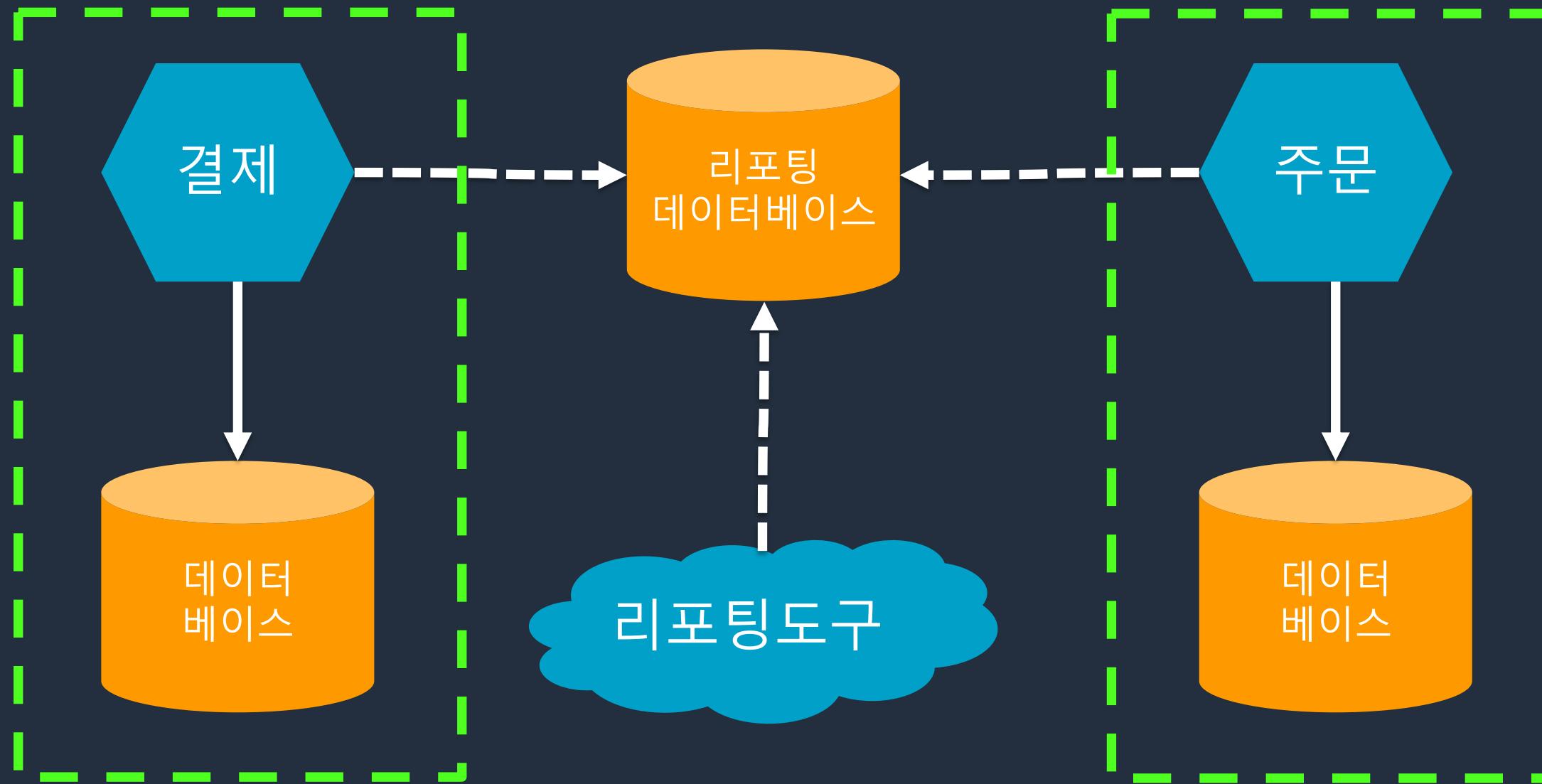


결제를 업그레이드 할 준비가 되면 새 앤드포인트 사용.  
DB는 서비스간 데이터 일관성을 허용하기 위해 서로 다른 서비스 버전에 공유.

# 전체 데이터를 활용한 보고서



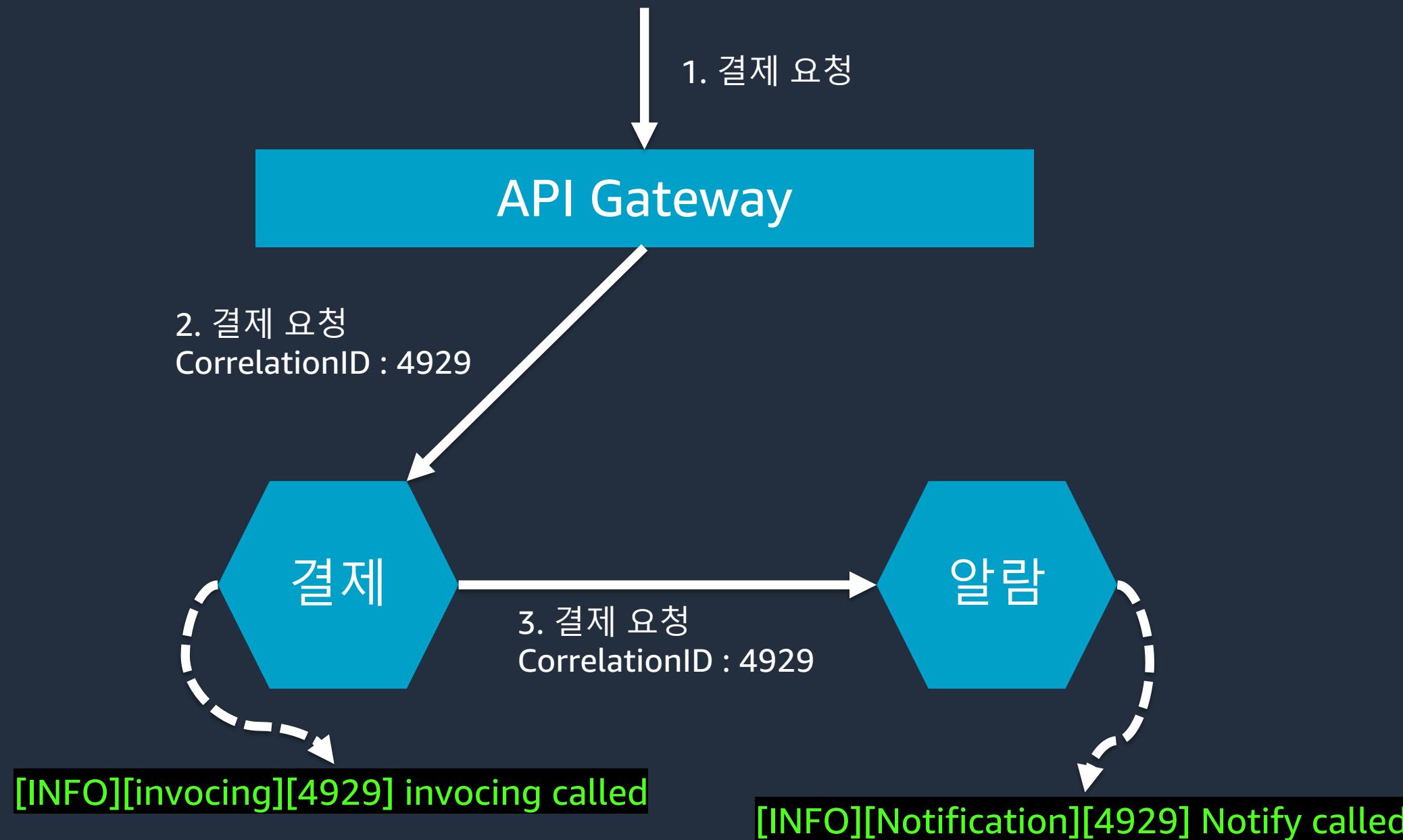
# 전체 데이터를 활용한 보고서



# 규모에 따른 소유권

- 강력한 코드 소유권
  - 모든 서비스에는 소유자가 있습니다.
- 약한 코드 소유권
  - 대부분의(또는 모든) 서비스가 누군가의 소유입니다.
- 공동 코드 소유권
  - 누구도 소유하지 않습니다.

# 관찰 가능성: 모니터링, 로깅 및 추적 그리고 디버깅



# AWS에서 구현하는 Microservice



# 서비스 컴퓨팅 옵션



서비스 기술을 적극 활용하십시오.

# 적절한 컴퓨팅 선택이 변화와 혁신의 핵심입니다.



AWS Lambda

## 서비스 함수

- 이벤트 알람
- 많은 언어 지원
- 데이터 통합
- 서버 관리 없음



AWS Fargate

## 서비스 컨테이너

- 긴 실행 시간
- OS 추상화
- 완전 관리형 오케스트레이션
- 완전 관리형 클러스터 확장

# 서비스 어플리케이션

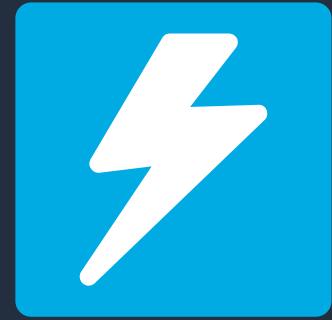
## 함수



Node.js  
Python  
Java  
C#  
Go  
Ruby  
Runtime API

# 서비스 어플리케이션

이벤트 소스



데이터 변경



엔드포인트  
요청



리소스 상태  
변경



함수



Node.js  
Python  
Java  
C#  
Go  
Ruby  
Runtime API

# 서비스 어플리케이션

이벤트 소스



데이터 변경



엔드포인트  
요청



리소스 상태  
변경



함수



Node.js  
Python  
Java  
C#  
Go  
Ruby  
Runtime API

서비스 (무엇이든)



# AWS 컨테이너 서비스

## Management

컨테이너화된 애플리케이션의 배포,  
스케줄링, 확장 및 관리



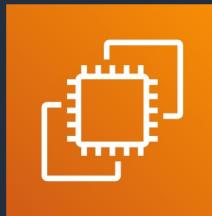
Amazon Elastic  
Container Service  
(Amazon ECS)



Amazon Elastic  
Kubernetes Service  
(Amazon EKS)

## Hosting

컨테이너가 실행되는 곳



Amazon Elastic  
Compute Cloud



AWS Fargate

## Image registry

컨테이너 이미지 저장소

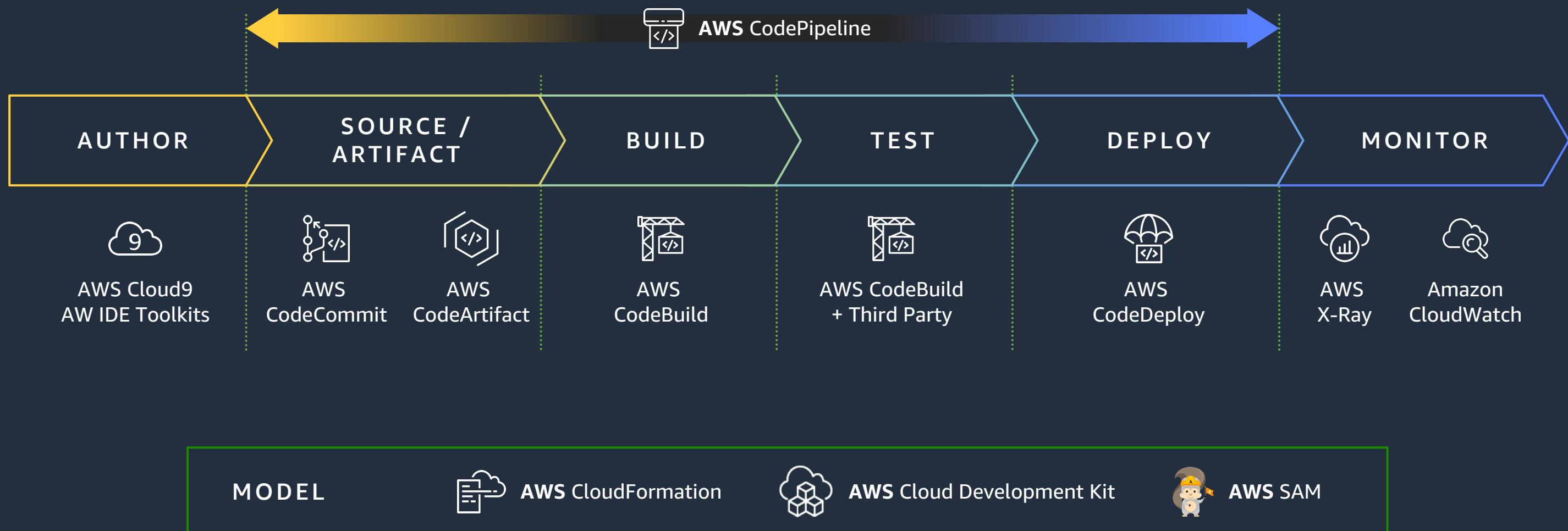


Amazon Elastic  
Container Registry  
(Amazon ECR)

# 개발자 도구



# 최신 소프트웨어 제공을 위한 AWS 개발자 도구는 표준화된 도구로 워크플로를 자동화하는데 도움이 됩니다.



# AWS 개발자 도구 포트폴리오에는 대부분의 작업을 위한 도구가 있습니다.

CI/CD Tools



AWS  
CodeStar



AWS  
CodeArtifact



AWS  
CodeBuild



AWS  
CodeCommit

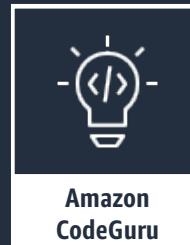


AWS  
CodeDeploy



AWS  
CodePipeline

ML DevTools



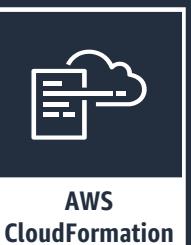
Amazon  
CodeGuru

Container Registry



Amazon ECR

Infrastructure as Code



AWS  
CloudFormation



AWS Cloud Dev.  
Kit (CDK)

Web Apps



AWS Elastic  
Beanstalk

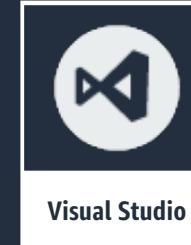
IDE



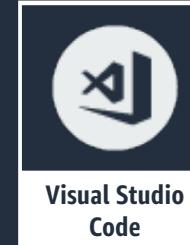
AWS Cloud9



JetBrains:  
IntelliJ, IDEA,  
etc.



Visual Studio



Visual Studio  
Code

CLI and Scripting Tools



AWS CLI



Tools for  
PowerShell

Monitoring



AWS X-Ray



Amazon  
CloudWatch



AWS Chatbot

Mobile



AWS Amplify

Modernization Tools



AWS  
App2Container



Porting  
Assistant for  
.NET

SDKs



JavaScript



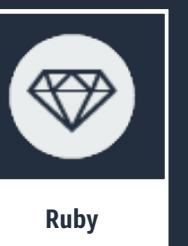
Python



PHP



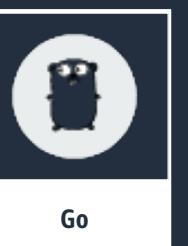
.NET



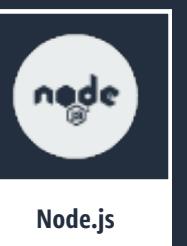
Ruby



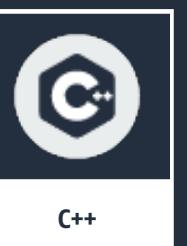
Java



Go



Node.js

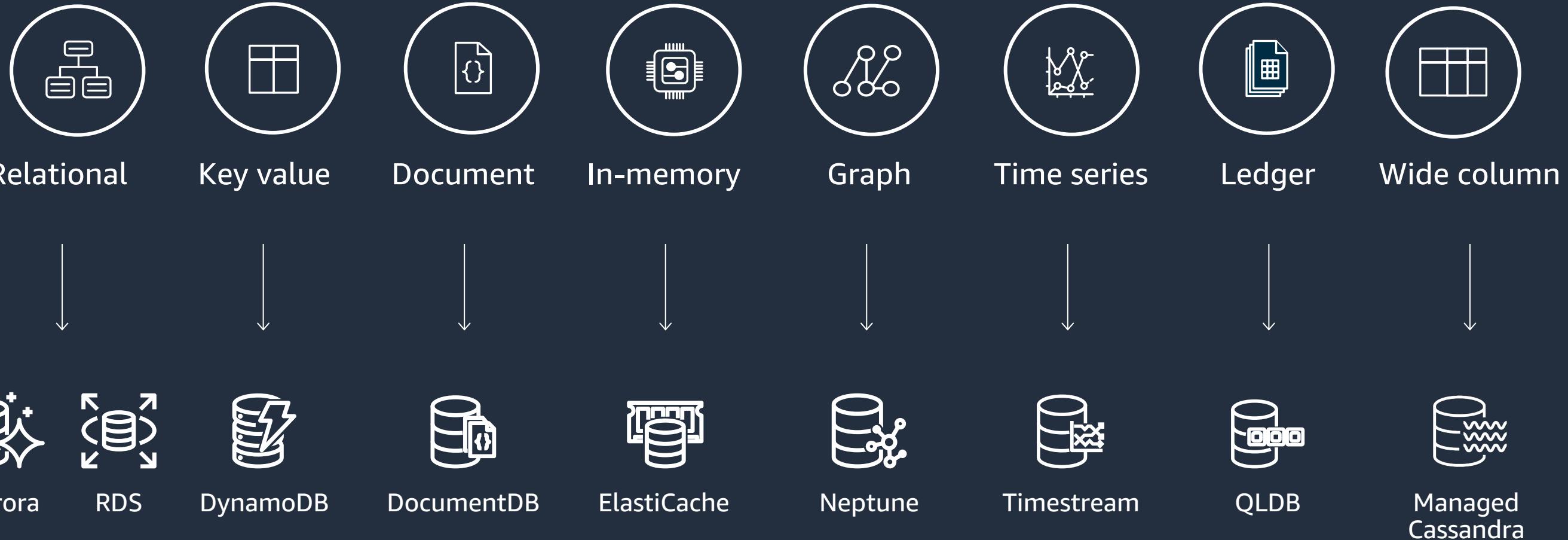


C++

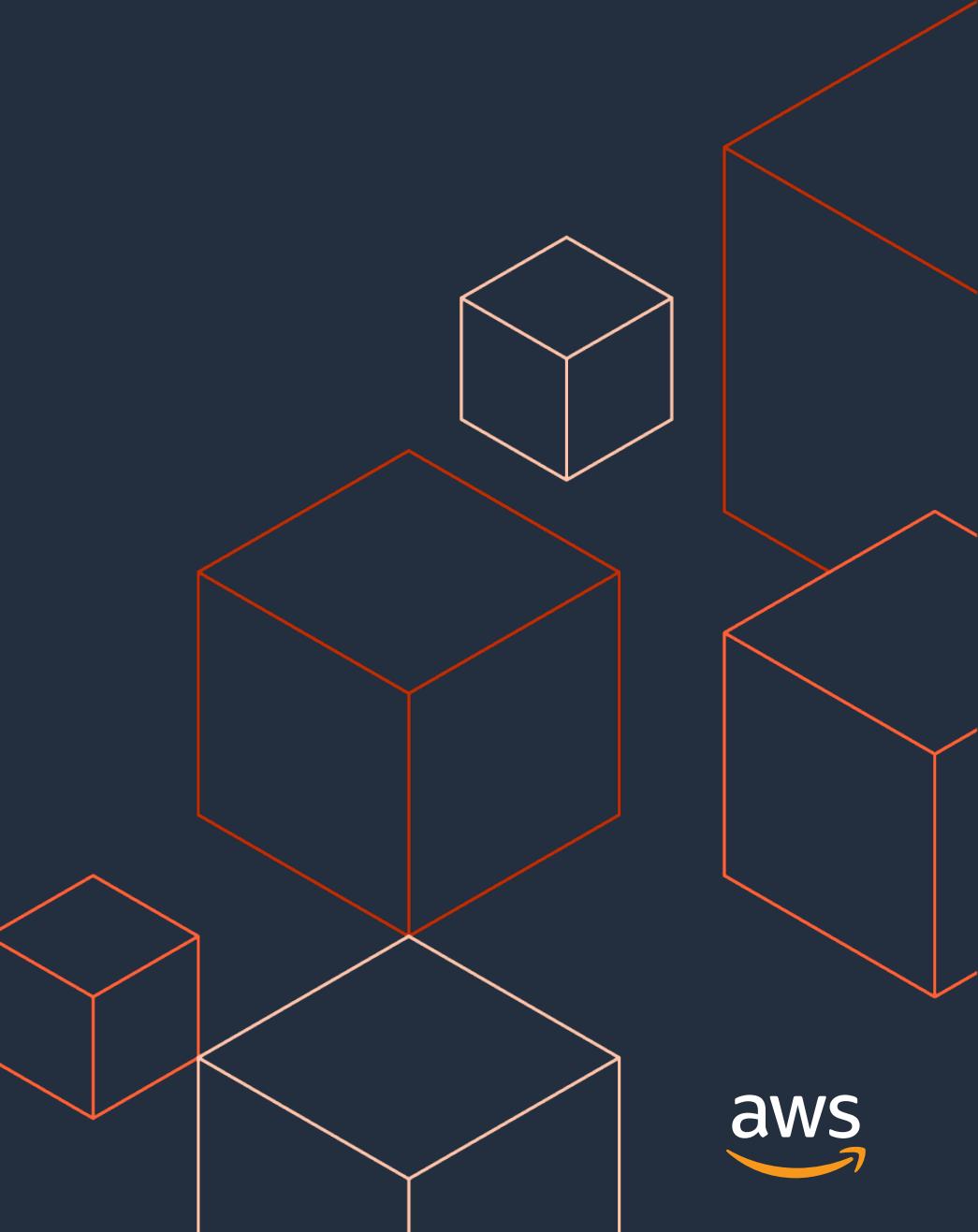
# 목적에 맞게 구축된 데이터베이스



# AWS에서 목적에 맞게 구축된 데이터베이스



# 분산 시스템 구성 요소



# DNS 기반 서비스 검색 – AWS Cloud Map

## 클라우드의 동적 지도 구축



모든 클라우드 리소스에 대해  
친숙한 이름 정의



특정 속성을 가진  
리소스 검색



동작하는 리소스만  
검색되도록 합니다.



고가용성 리전 API 및  
글로벌 DNS 사용

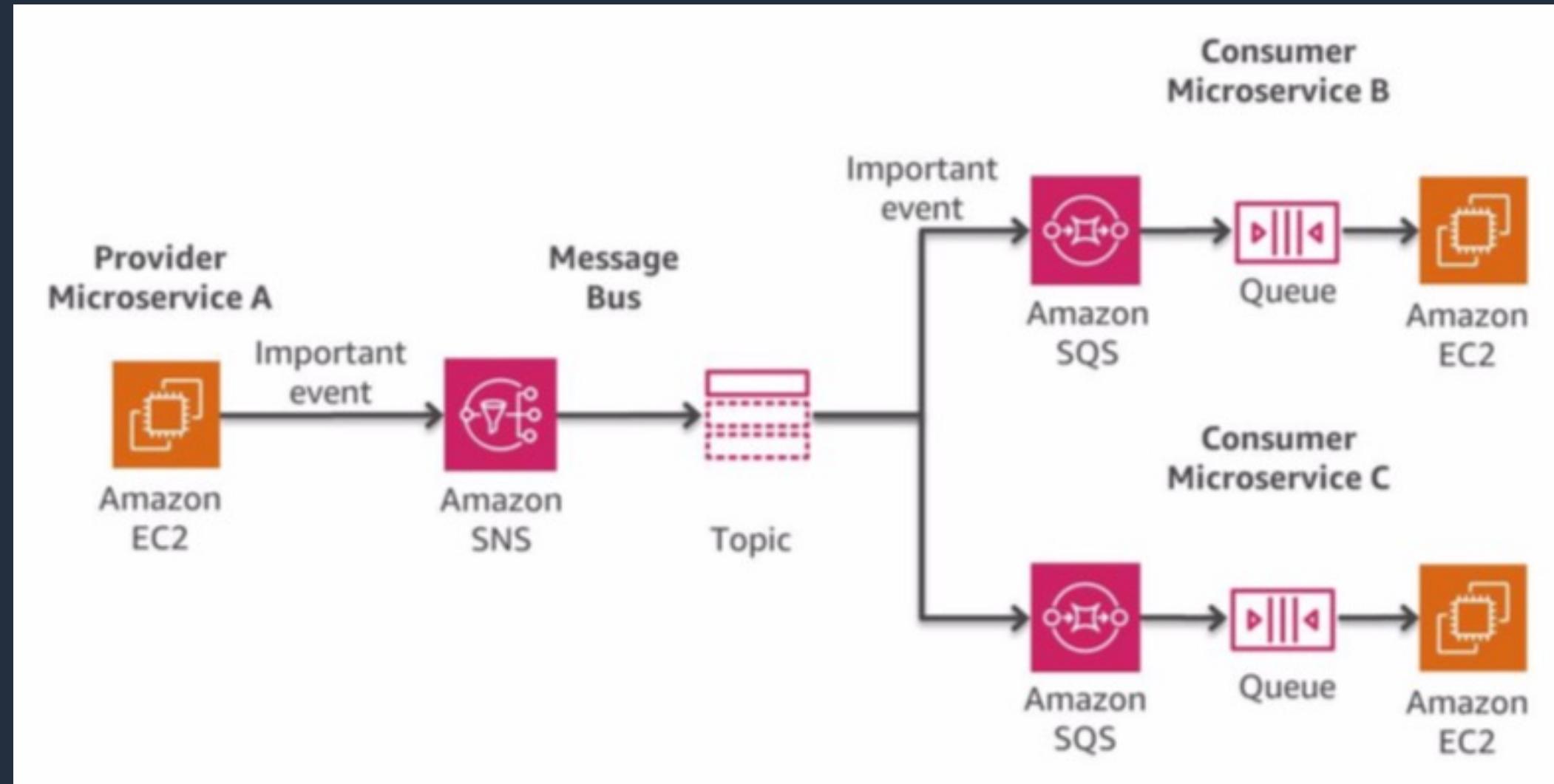
# 서비스 메시- AWS App Mesh



AWS App Mesh

AWS의 마이크로서비스에 대한 가시성 및 제어  
앱 메시는 서비스 메시입니다.

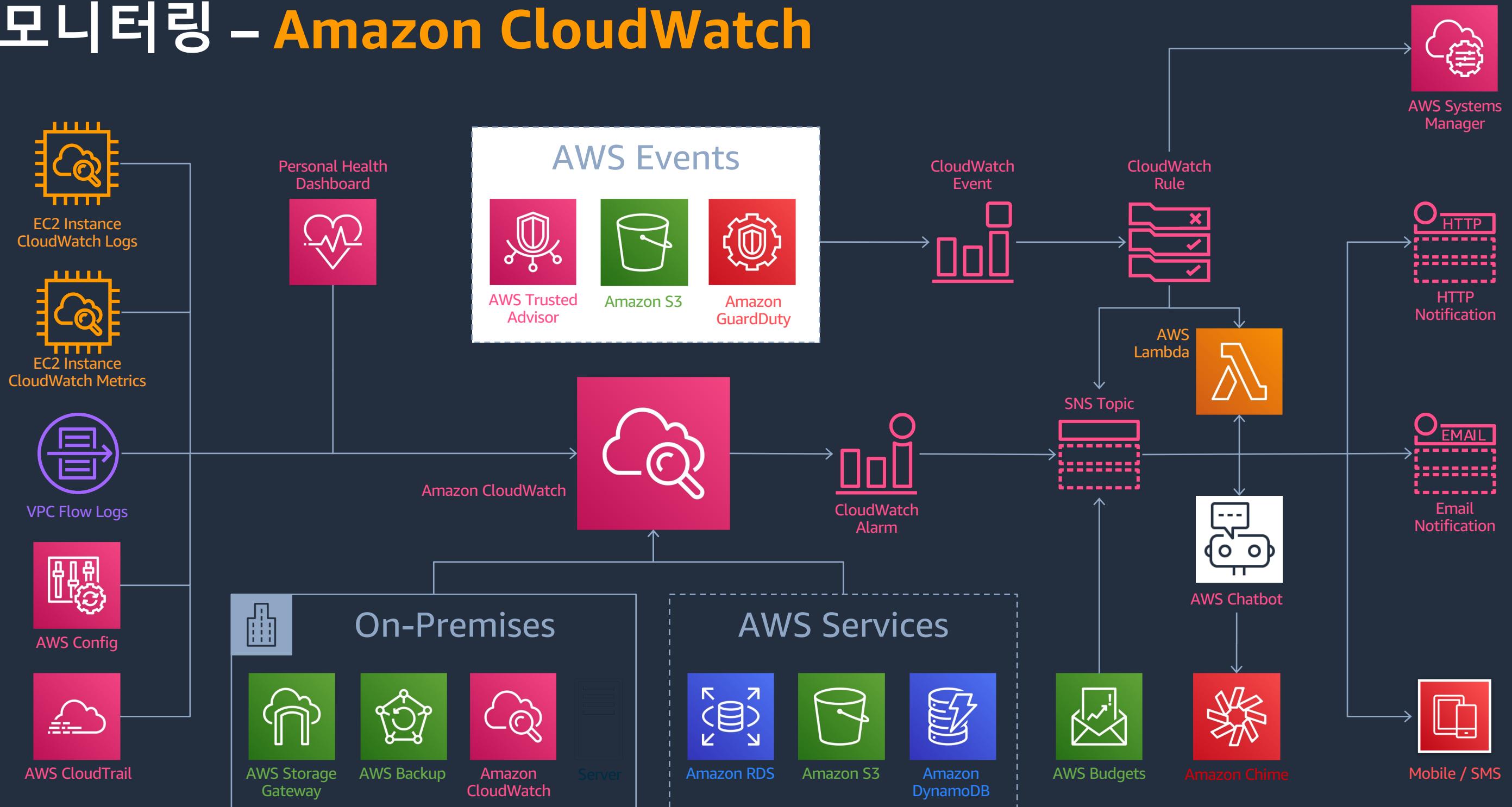
# 비동기 통신 및 이벤트 전달 – SQS & SNS



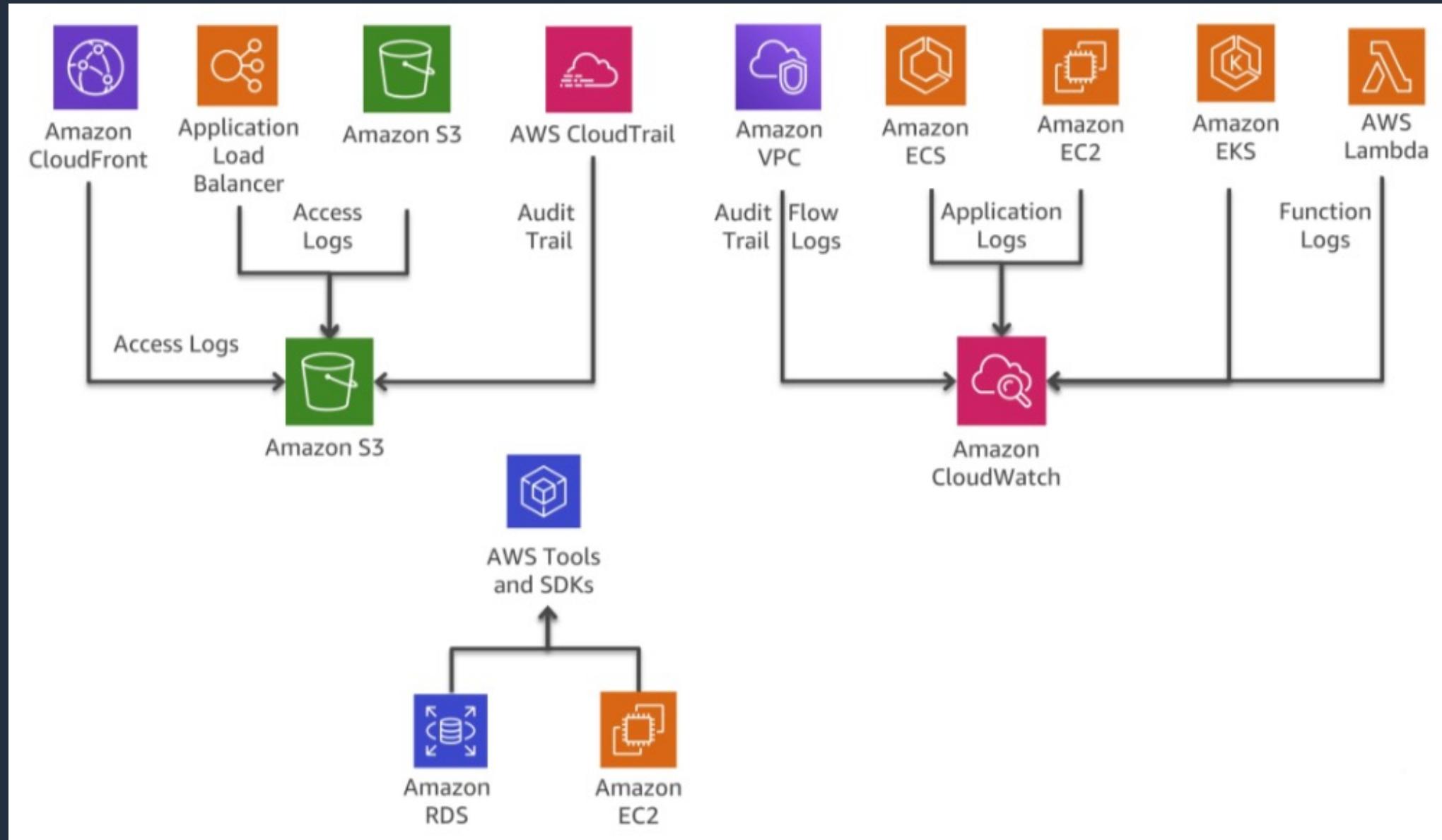
# 관찰 가능성



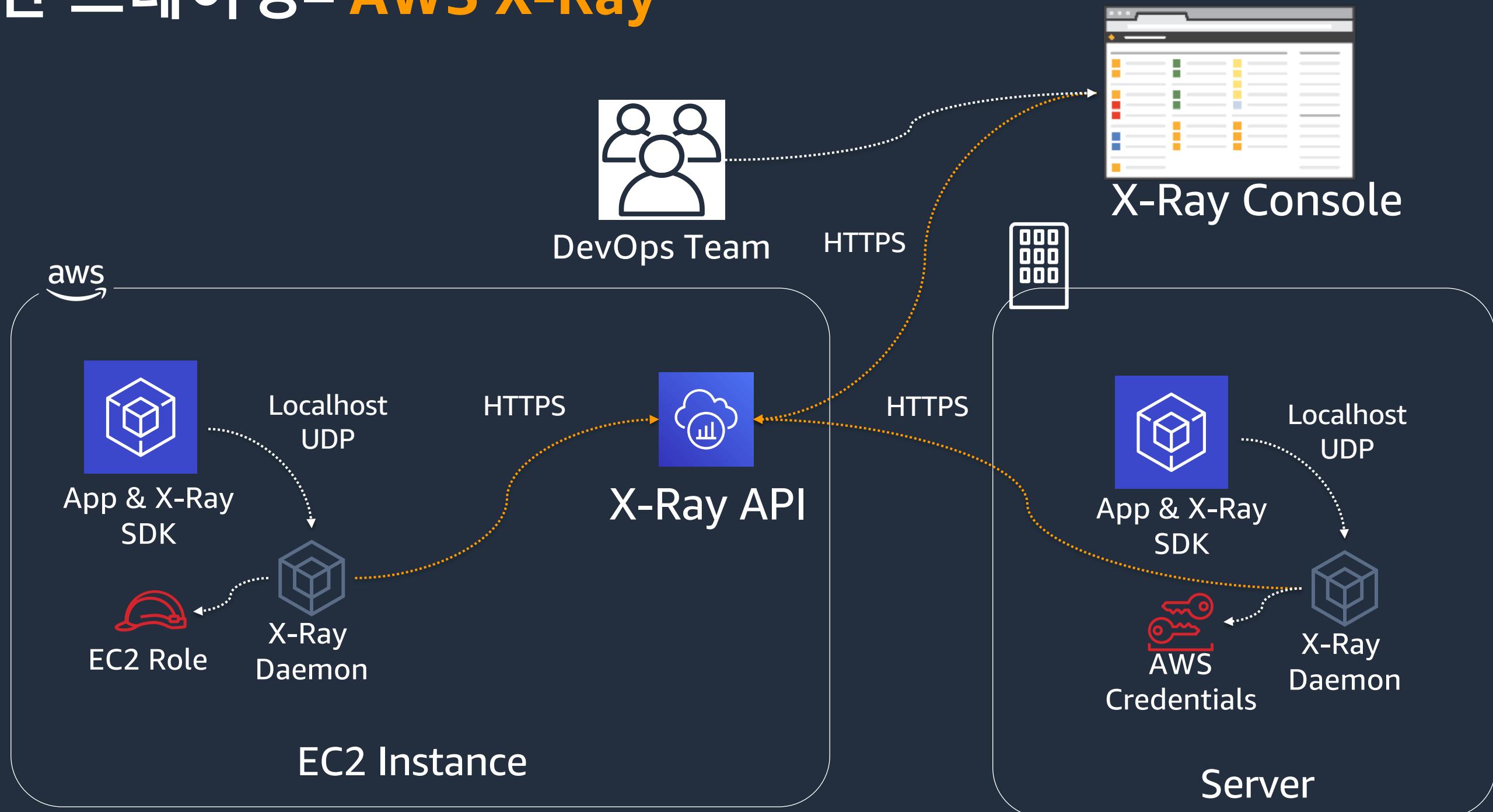
# 모니터링 - Amazon CloudWatch



# 중앙화 로깅 - Amazon CloudWatch Logs



# 분산 트레이싱- AWS X-Ray



# 분산 트레이싱- AWS X-Ray



# Hands on lab :

<https://bit.ly/3QyMG30>

# EventEngine :

<https://bit.ly/3AulIVr>

# 감사합니다.



# Let `s Design Social Network Service



# Challenge : AnyCompany : SNS 추가하기



UI Team

Web UI  
=>presentation



Backend Team

Backend  
=>application



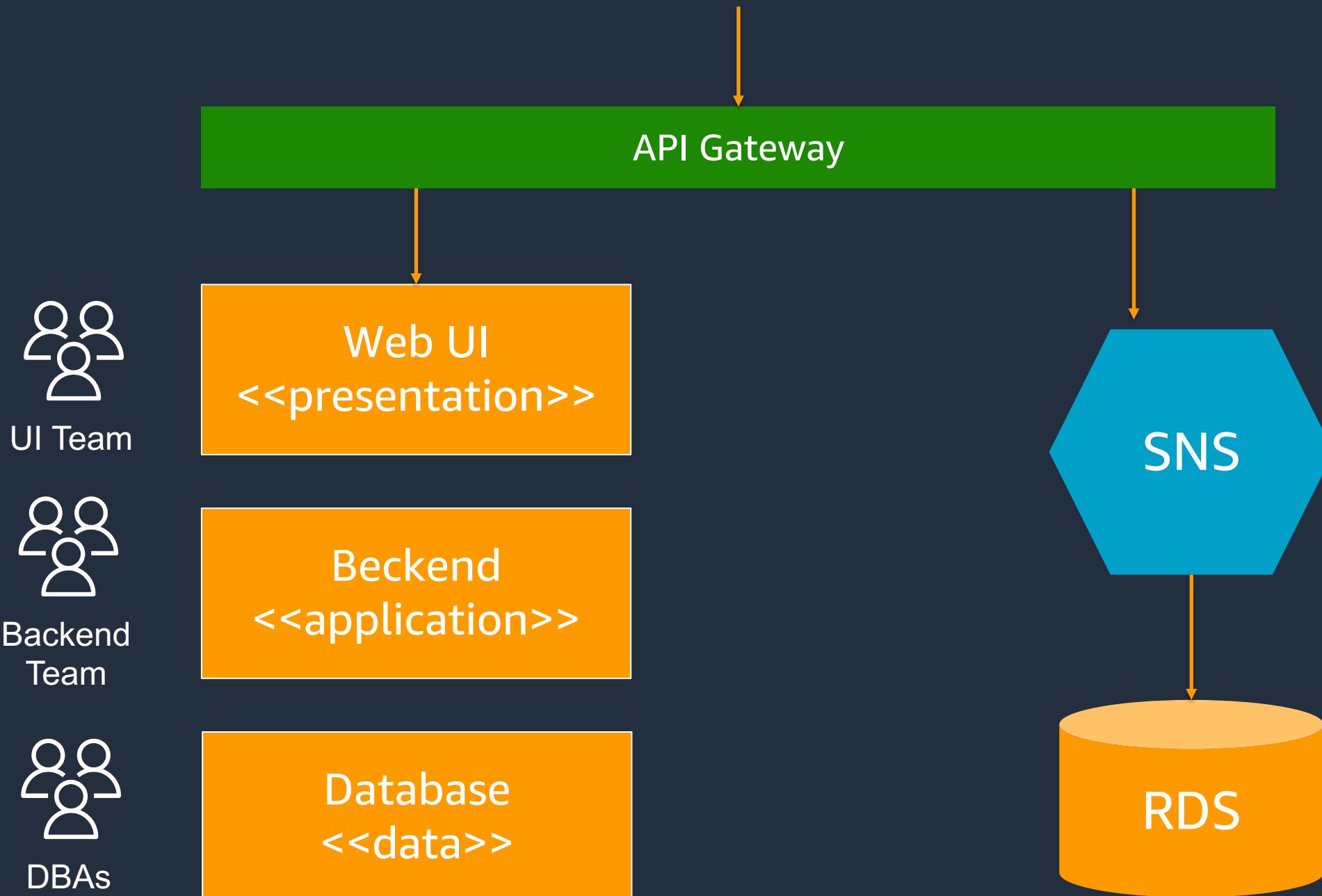
DBAs

Database  
=>data

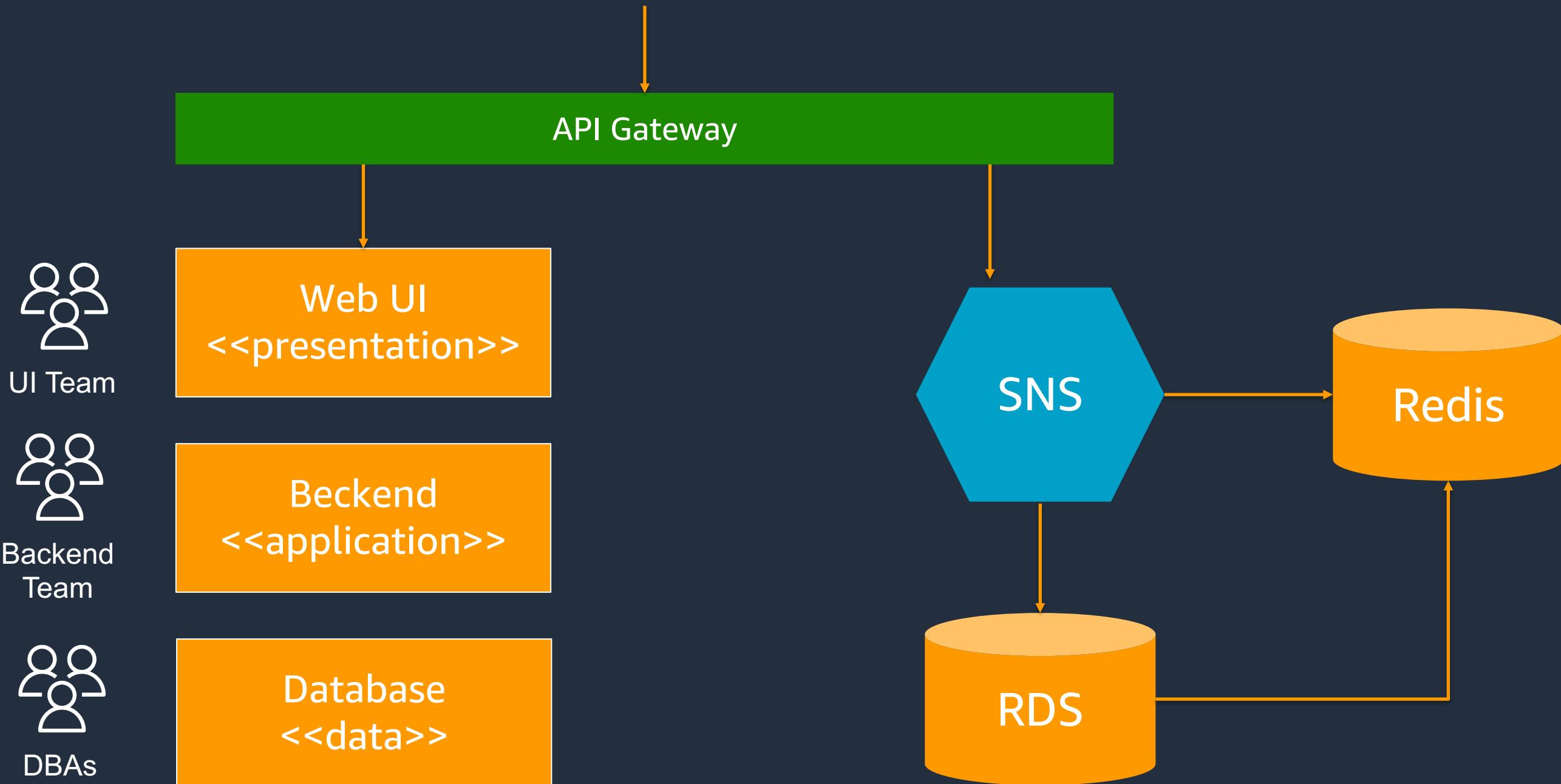
글로벌 e-Commerce 회사에서 새로운 소셜 네트워크 기능을 추가하려고 합니다. 하루 사용자는 1000명정도입니다. 당신의 새로운 팀은 아래와 같은 기능을 구현하고자 합니다.

1. 기존의 사용자 데이터베이스 이용
2. 친구추가, 친구 삭제 기능.
3. 글쓰기, 좋아요, 물건 홍보 기능
4. 기존의 개발자와 운영자들은 기존 서비스를 최대한 건드리고 싶지 않습니다.

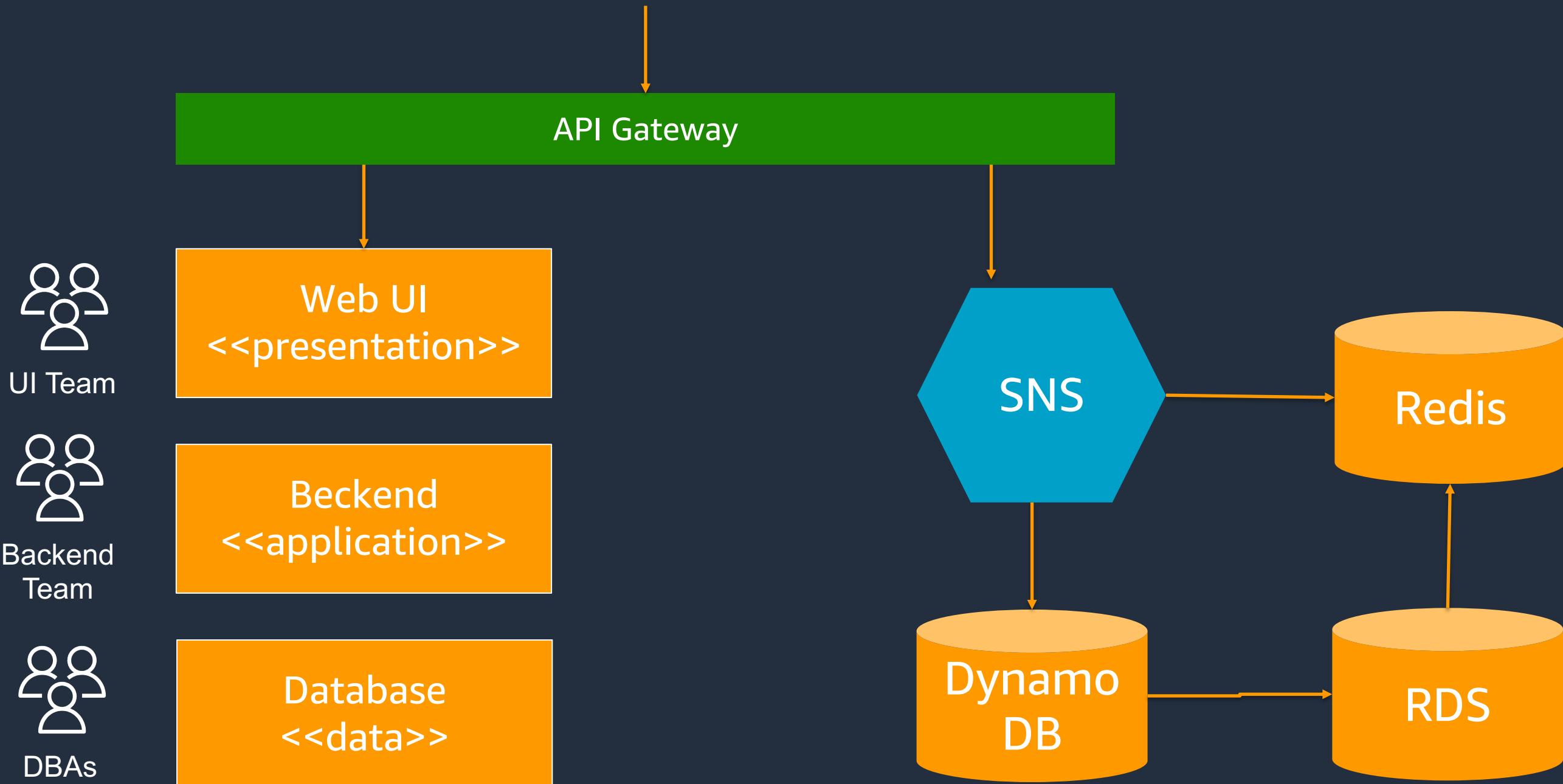
# Challenge : AnyCompany : SNS 추가하기



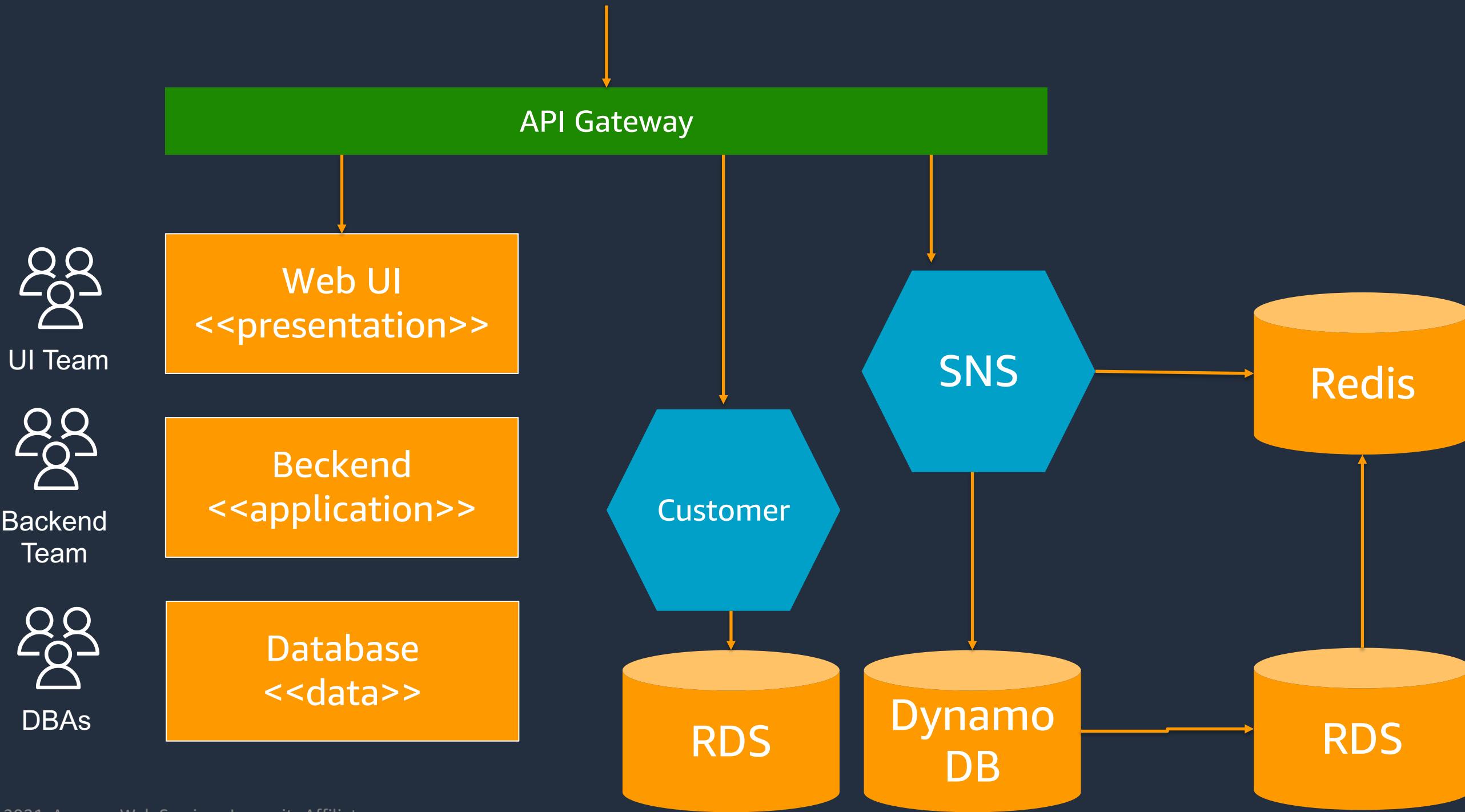
# Challenge : AnyCompany : SNS 추가하기



# Challenge : AnyCompany : SNS 추가하기



# Challenge : AnyCompany : SNS 추가하기



# Challenge : AnyCompany : Hero-Lim의 이벤트 페이지



UI Team

Web UI  
=> presentation



Backend Team

Backend  
=> application



DBAs

Database  
=> data

Hero-Lim이 콘서트를 합니다.  
평범한 e-Commerce인 여러분의 회사가  
Hero-Lim 콘서트 티켓을 팔게 되었습니다.  
사장님은 신났지만, 여러분은 큰일이  
났습니다!

1. 기존의 사용자 데이터베이스 이용
2. 기존의 E-commerce에 영향을 주면 안됩니다.
3. 좌석 선택 기능이 필요합니다.
4. 고객은 자신이 얼마나 기다려야하는지 궁금해합니다.

서비스가 중지되면 큰 손실이 생깁니다.

# Thank you

