Techtonic 2018

-

Thu . Nov 15

-

SAMSUNG SDS Tower
West Campus B1F
Magellan Hall /Pascal Hall

SAMSUNG SDS
Realize your vision

To Microservices and Beyond

Agile팀의 MSA 적용 고군분투기

삼성SDS 이기영 프로

Techtonic 2018

Agenda

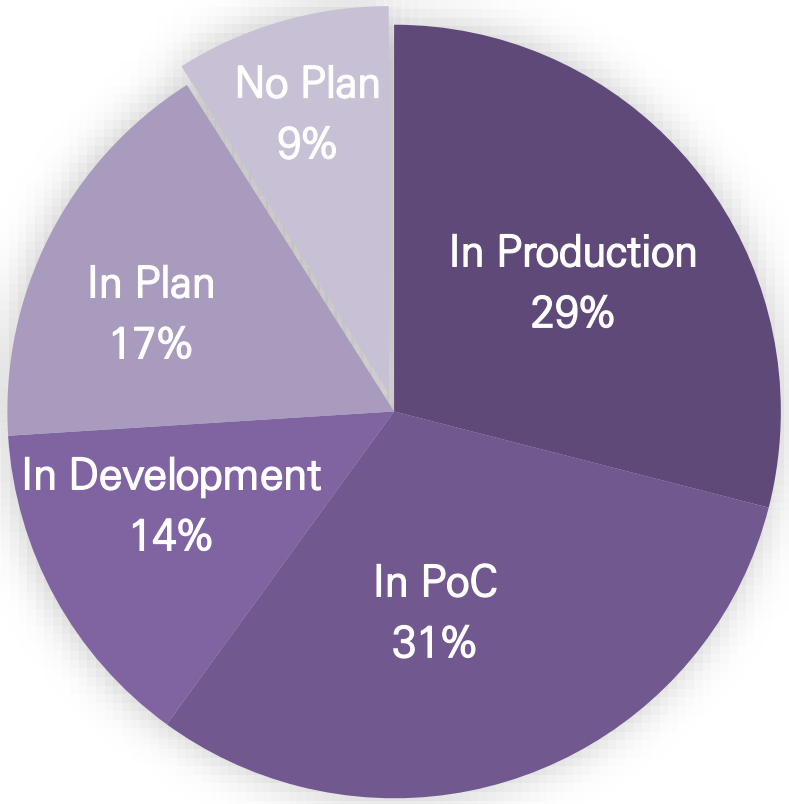
- About Microservices
- Transition to MSA
- Lessons learned
- To Microservices and Beyond

Agile팀의 MSA 적용 고군분투기 – To Microservices and Beyond

About Microservices

Microservices are popular

Microservices have become **mainstream** in the **enterprise**



〈Microservice 적용 현황〉

91% 사용 중이거나 사용할 계획

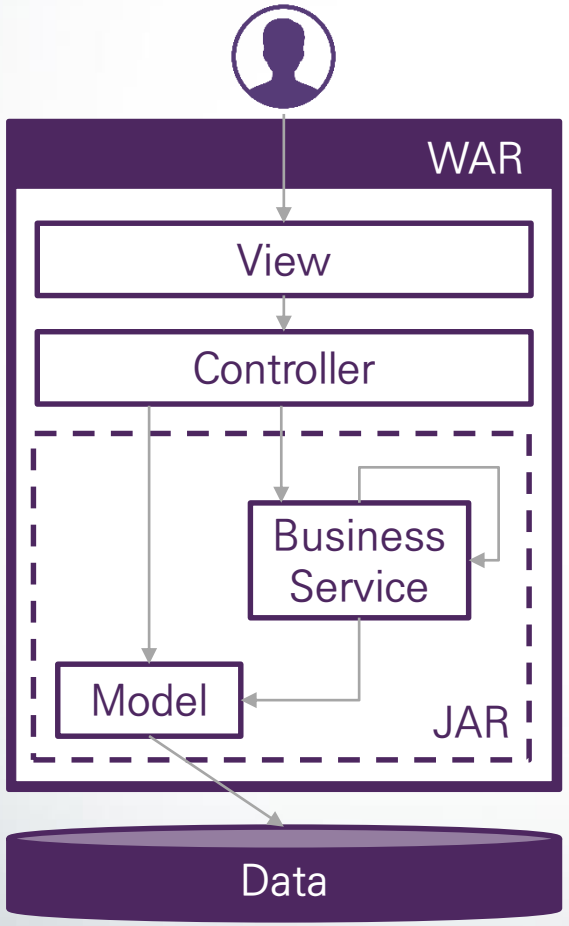
86% 향후 5년 이내 기본 아키텍처로 예상

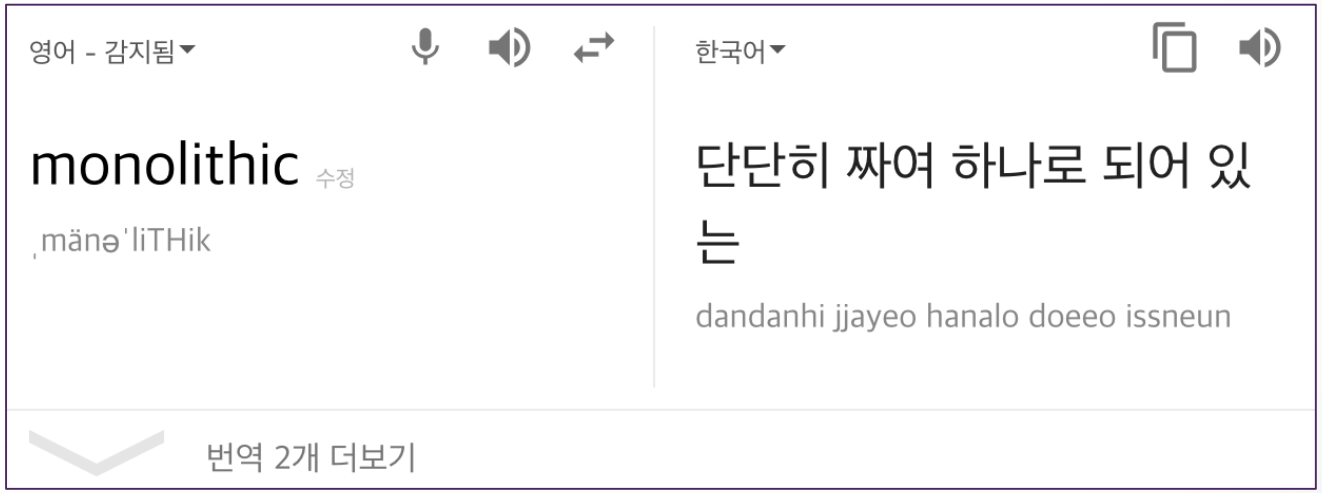
92% 작년 대비 마이크로서비스 수가 증가

Microservices Global Trends Report
2018 sponsored by LightStep

Monolithic Architecture

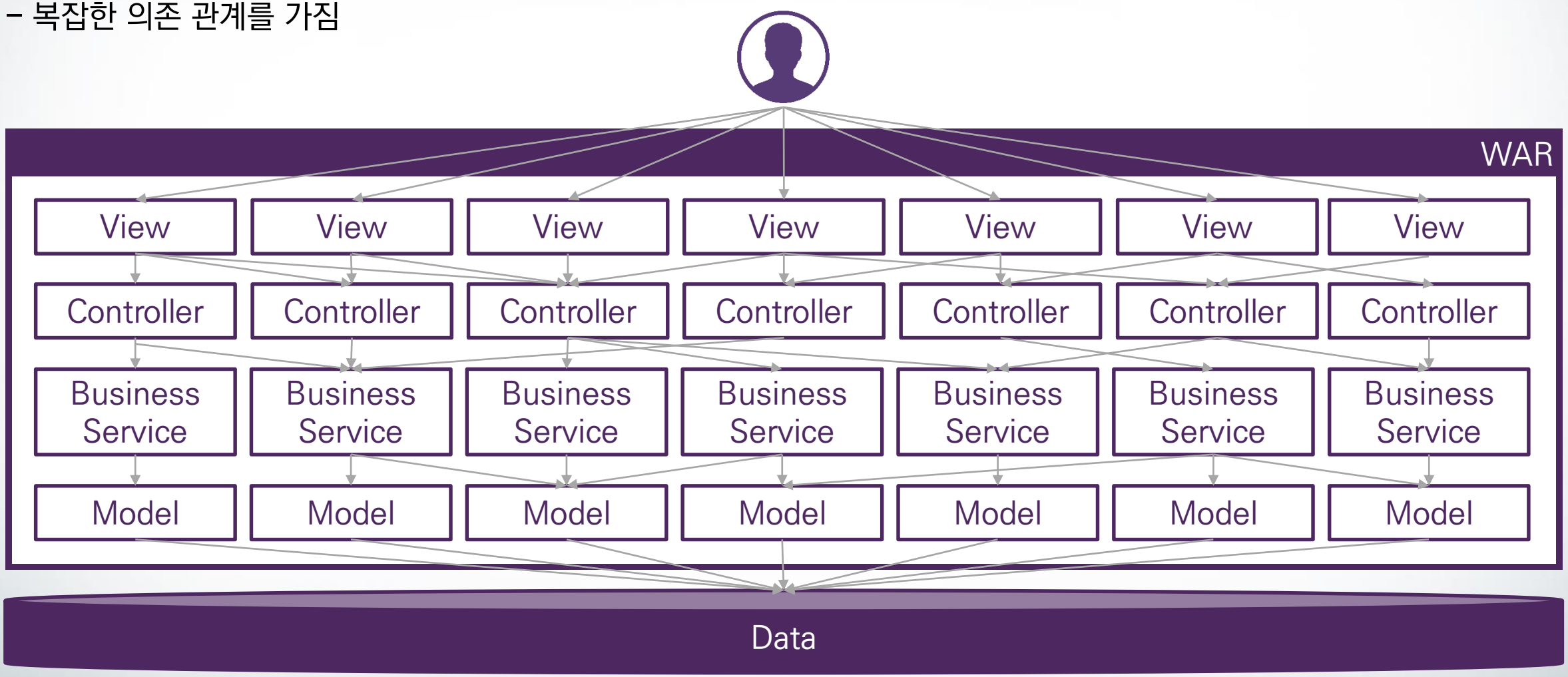
- 전통적 웹 시스템 개발 스타일
- 모든 구현 로직이 **단일 코드베이스 기반으로 하나의 애플리케이션**에 모두 들어있는 구조





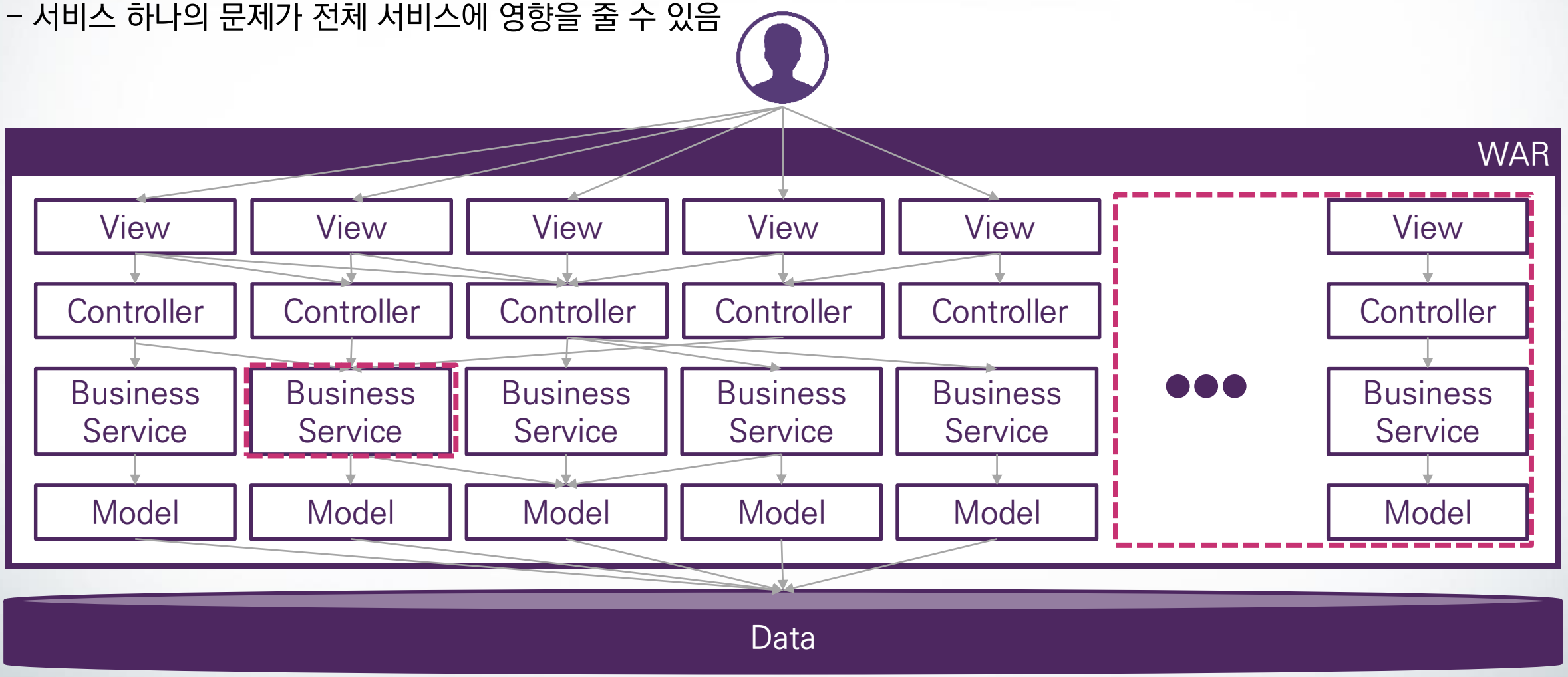
Monolithic Architecture

- Enterprise 환경에서 Monolithic Architecture는 많은 기능들이 복수의 모듈들로 구성됨
- 복잡한 의존 관계를 가짐



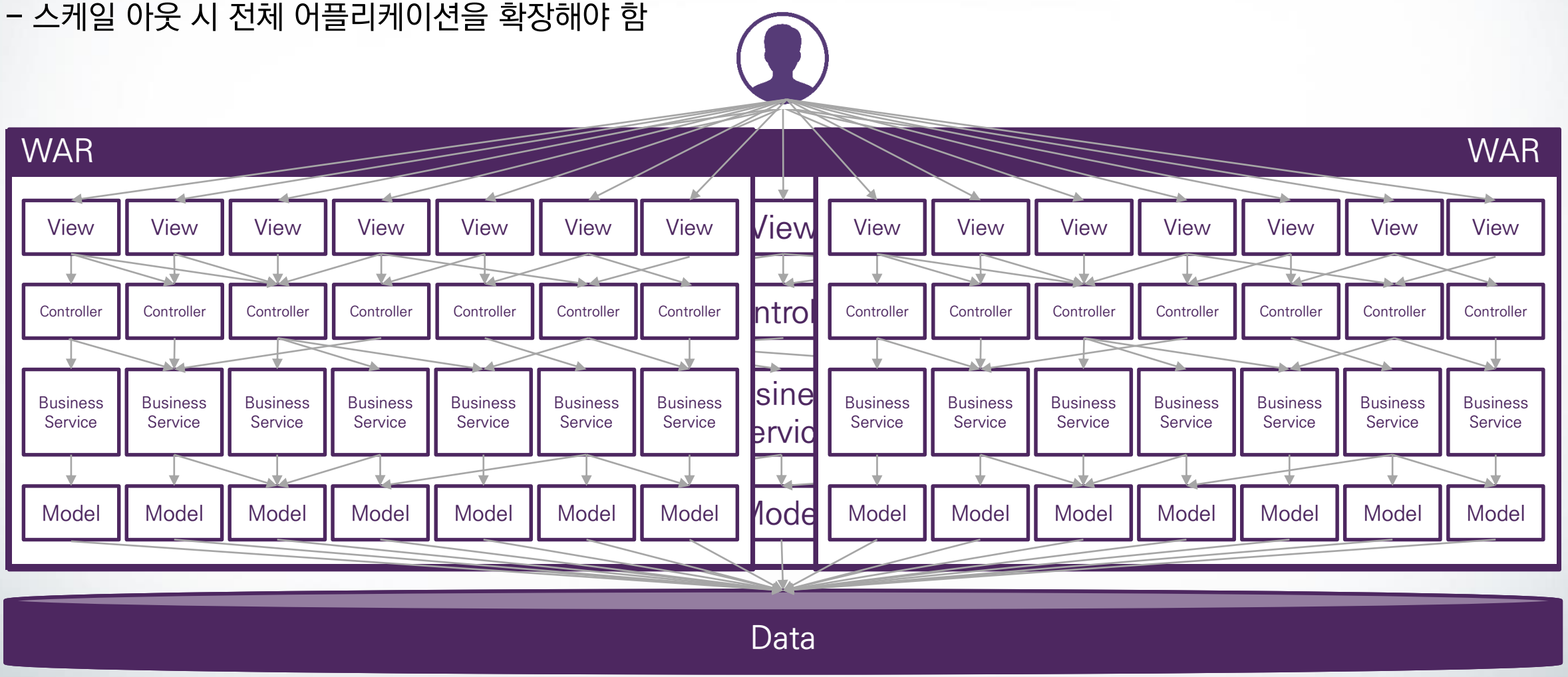
Monolithic Architecture – Problem

- 서비스(기능)가 증가할수록 코드복잡도, 빌드/테스트/배포/기동 시간이 점점 증가함
- 서비스 하나의 문제가 전체 서비스에 영향을 줄 수 있음



Monolithic Architecture – Problem

- 부분적 스케일 아웃이 불가능함
- 스케일 아웃 시 전체 어플리케이션을 확장해야 함



Microservices Architecture

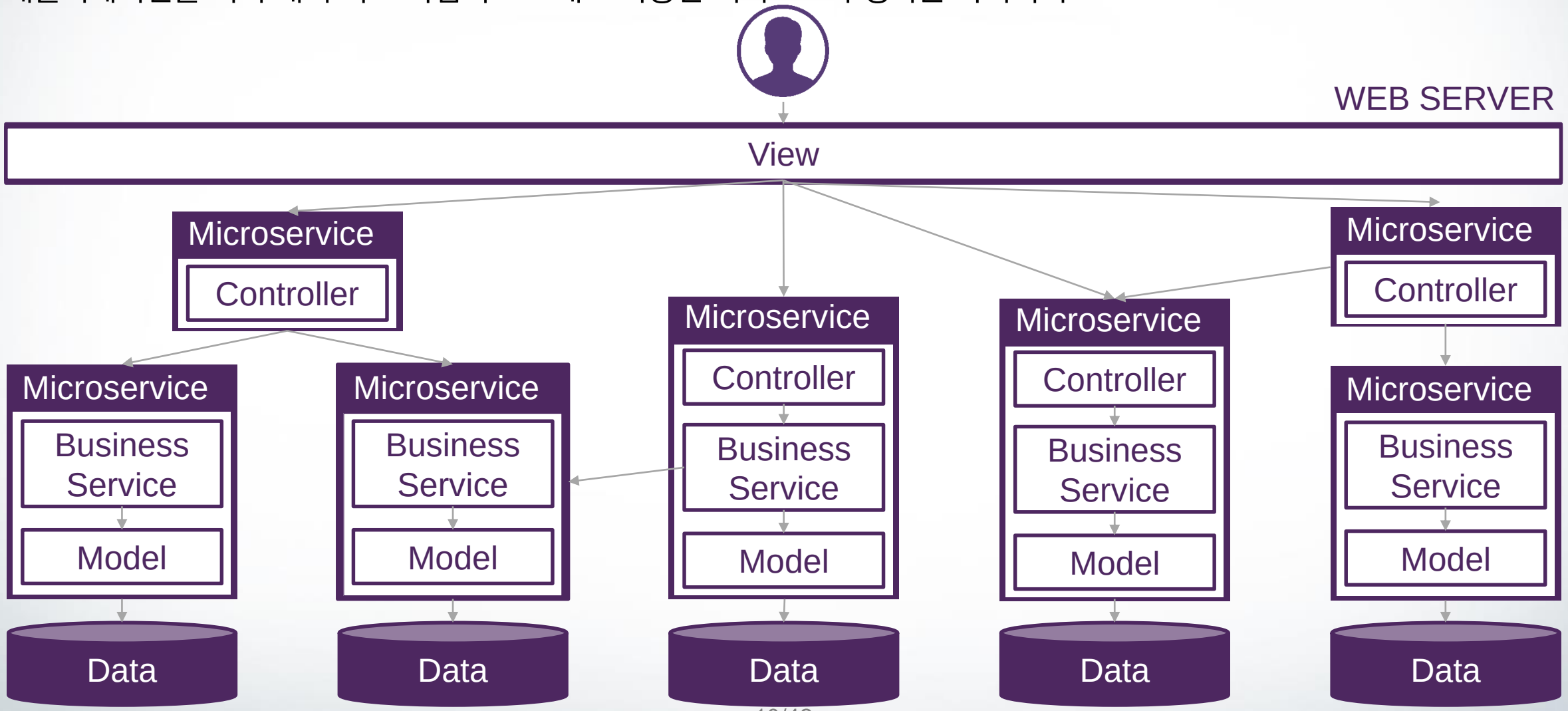
마이크로서비스 아키텍처(Microservices Architecture) 라는 용어는 소프트웨어 응용 프로그램을 독립적으로 배치 가능한 서비스 조합(suite)으로 설계하는 방식으로서 지난 몇 년 동안 빠르게 자리잡아 가고 있습니다. 본 아키텍처 스타일에 대해 아직 명확한 정의는 없지만, 비지니스 수행과 관련된 조직, 배포 자동화, (서비스) 엔드 포인트의 지적 능력(intelligence), 그리고 프로그래밍 언어와 데이터의 분산 제어에 관한 일반적인 특징들을 지닙니다.

2014년 3월 25일
James Lewes, Martin Fowler

http://channy.creation.net/articles/microservices-by-james_lewes-martin_fowler

Microservices Architecture

애플리케이션을 여러 개의 작고 독립적으로 배포 가능한 서비스로 구성하는 아키텍처



Microservices Architecture – 장/단점

일반적으로 MSA는 Monolithic Architecture와 비교하여 다음과 같은 장/단점을 지님

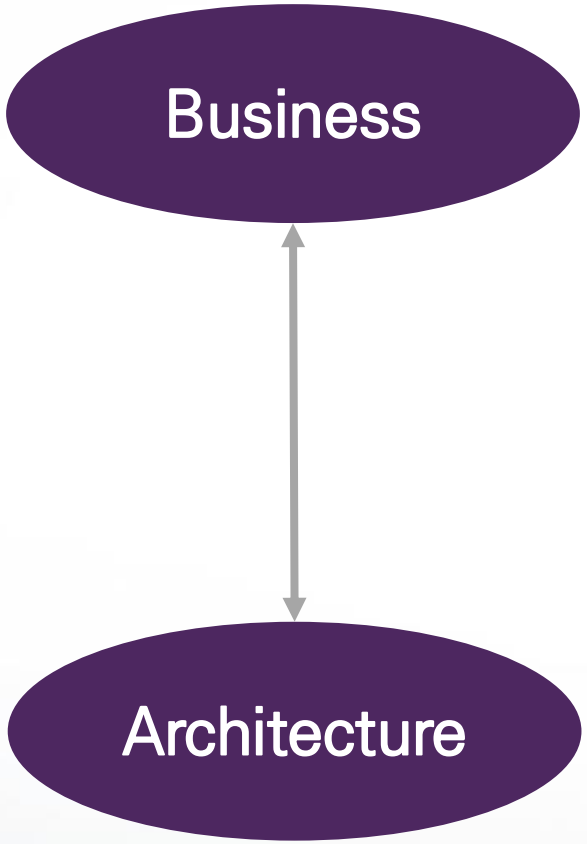
장점

- 빌드/배포/테스트 시간 단축
- Polyglot 아키텍처 구성 가능
- 서비스 별 선택적인 확장 가능
- 타 서비스 영향 최소화

단점

- 성능 이슈
- 서비스 간 트랙잭션 처리 (부분적 실패)
- 관리 포인트 증가 (모니터링, 로깅, 분산된 데이터)
- 테스트 복잡도

Why Microservices Architecture

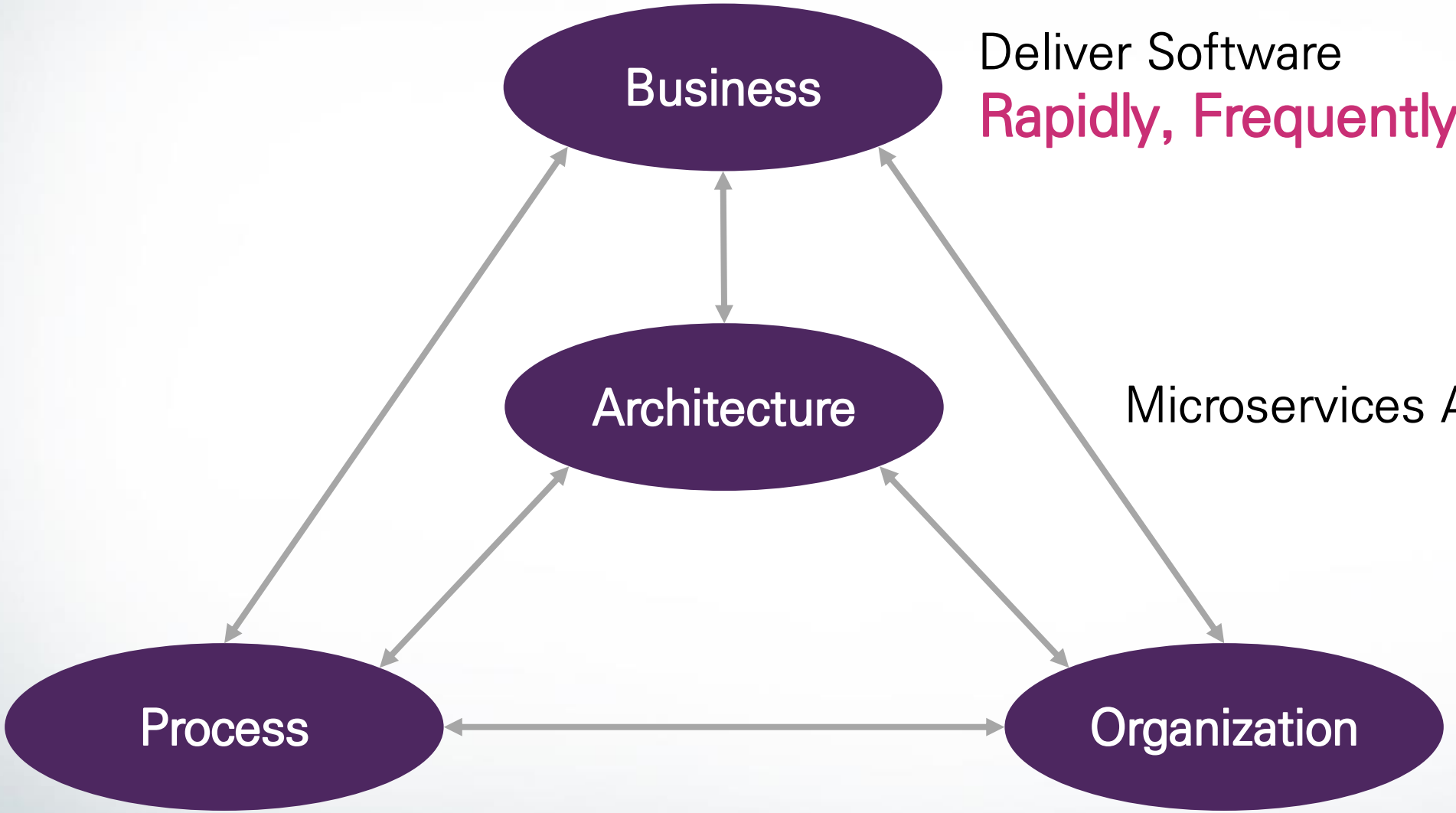


Deliver Software

Rapidly, Frequently and Reliably

Microservices Architecture

Why Microservices Architecture



Deliver Software
Rapidly, Frequently and Reliably

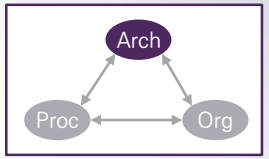
Microservices Architecture

Agile팀의 MSA 적용 고군분투기 – To Microservices and Beyond

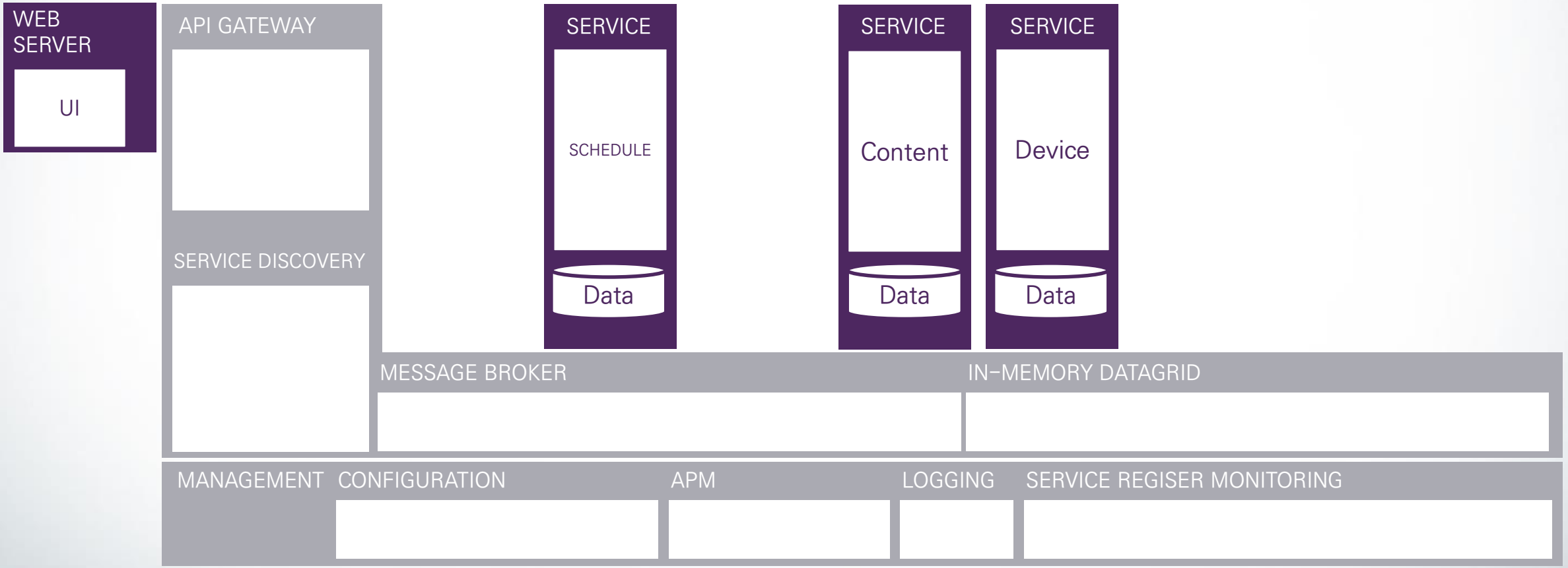
Transition to MSA

PHASE #1

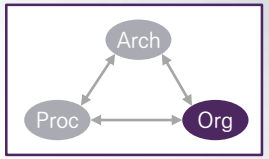
Phase #1 Architecture



정해진 Delivery 일정 고려하여 MSA 구성 요소에 대한 검토 비즈니스 분석, 서비스 설계/개발이 동시 수행

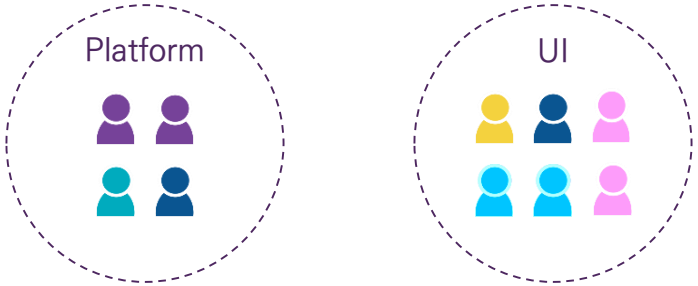


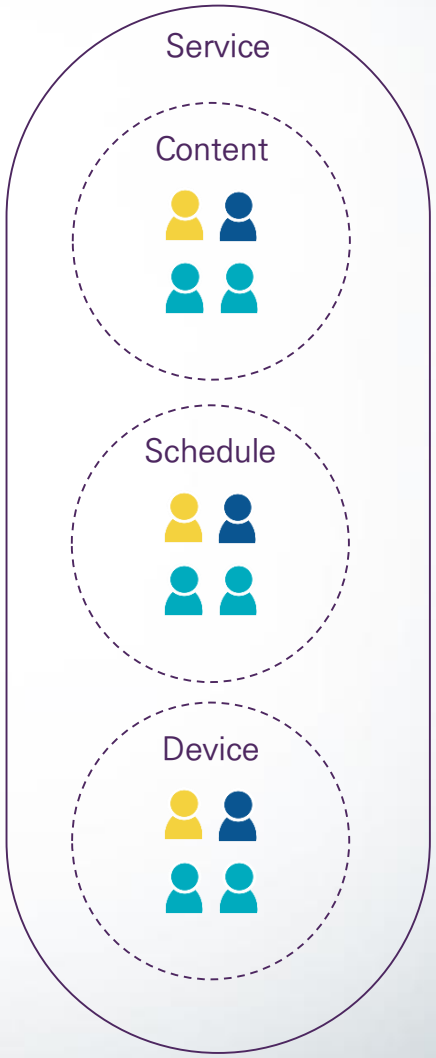
Phase #1 Organization



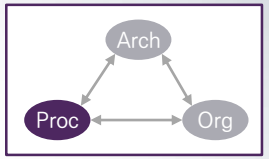
Platform팀, UI팀, Service팀(3개)으로 시작

- **Platform팀** : MSA Stack, DevOps 지원
- **UI팀** : CX설계, UI 개발
- **Service팀** : API 개발(unit/functional test)





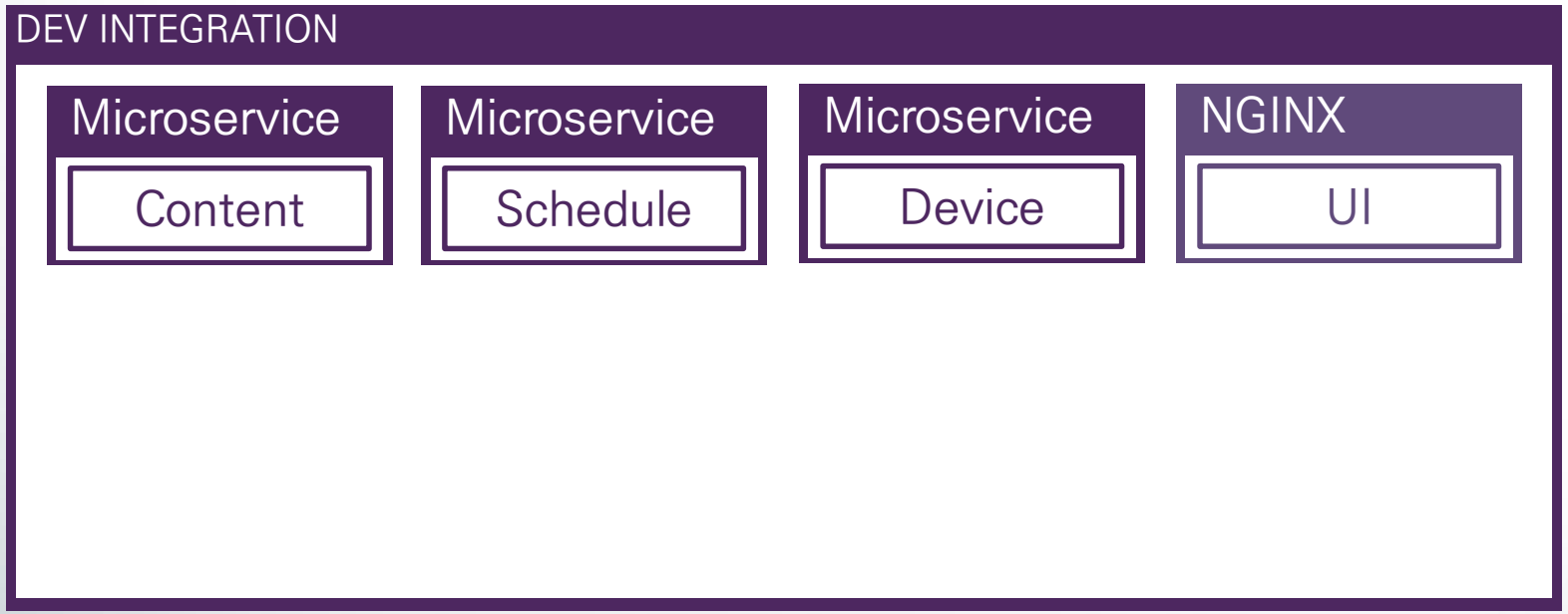
Phase #1 Process



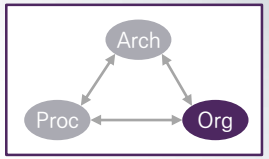
API 사용자 스토리의 Acceptance Criteria 확인을 위한 개발환경 구성

- CI/CD Pipeline

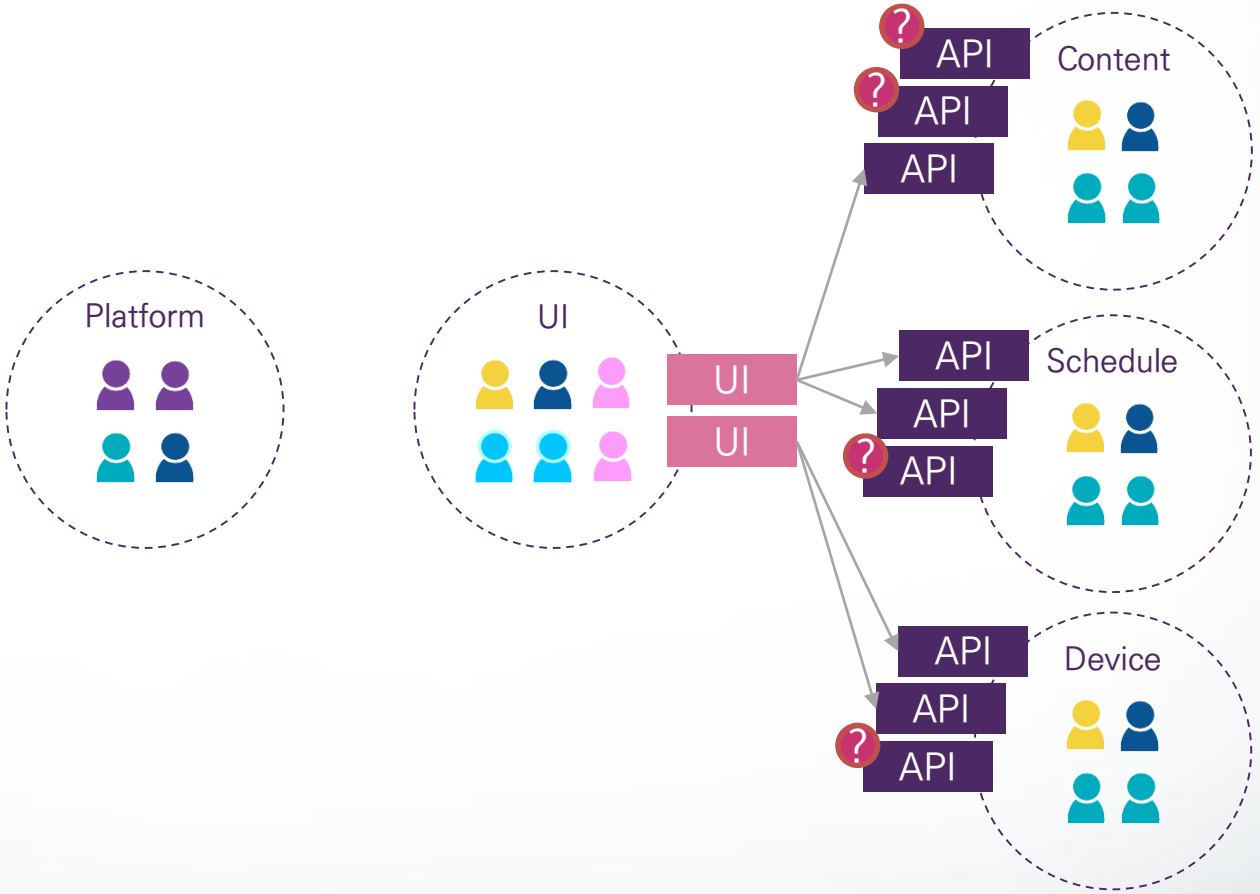
- Deploy Environment



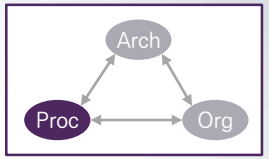
Phase #1 Organization Problem



- Service API 작업의 높은 우선 순위로 인해 실제 UI에서 사용하지 않는 API 개발
- Service팀에서 개발 완료된 API 공유 어려움



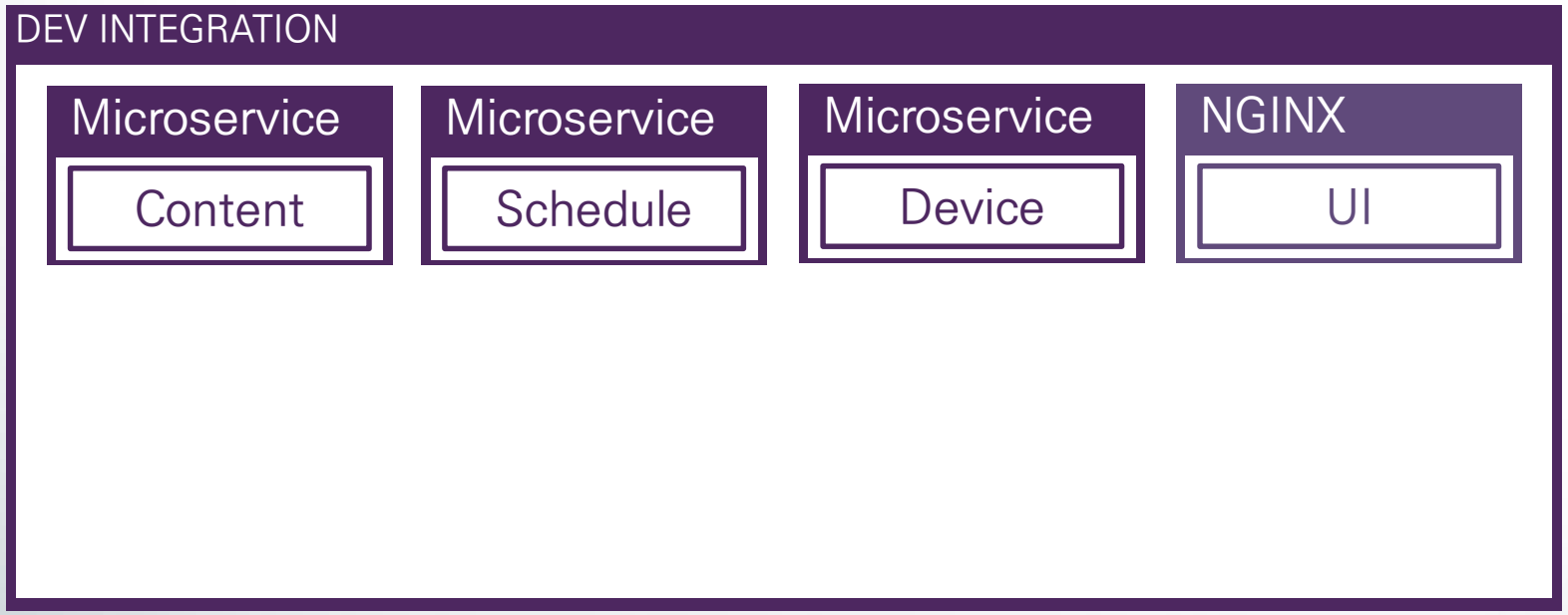
Phase #1 Process Problem



특정 서비스 Deploy 실패 시, 해당 서비스를 사용하는 UI 개발 진행 불가능

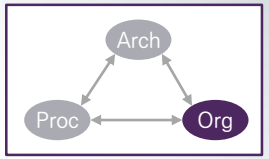
- CI/CD Pipeline

- Deploy Environment

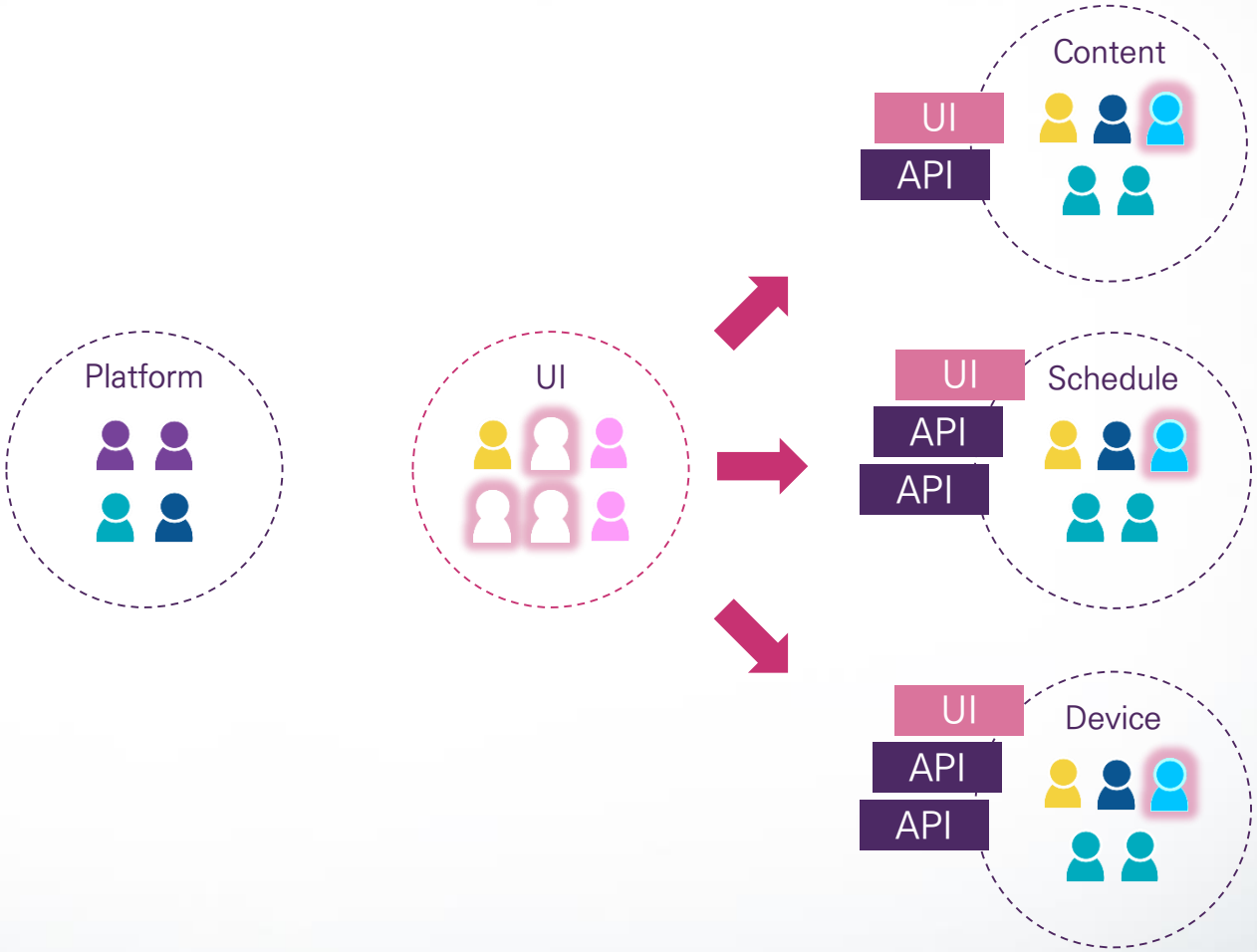


PHASE #2

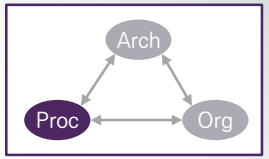
Phase #2 Organization



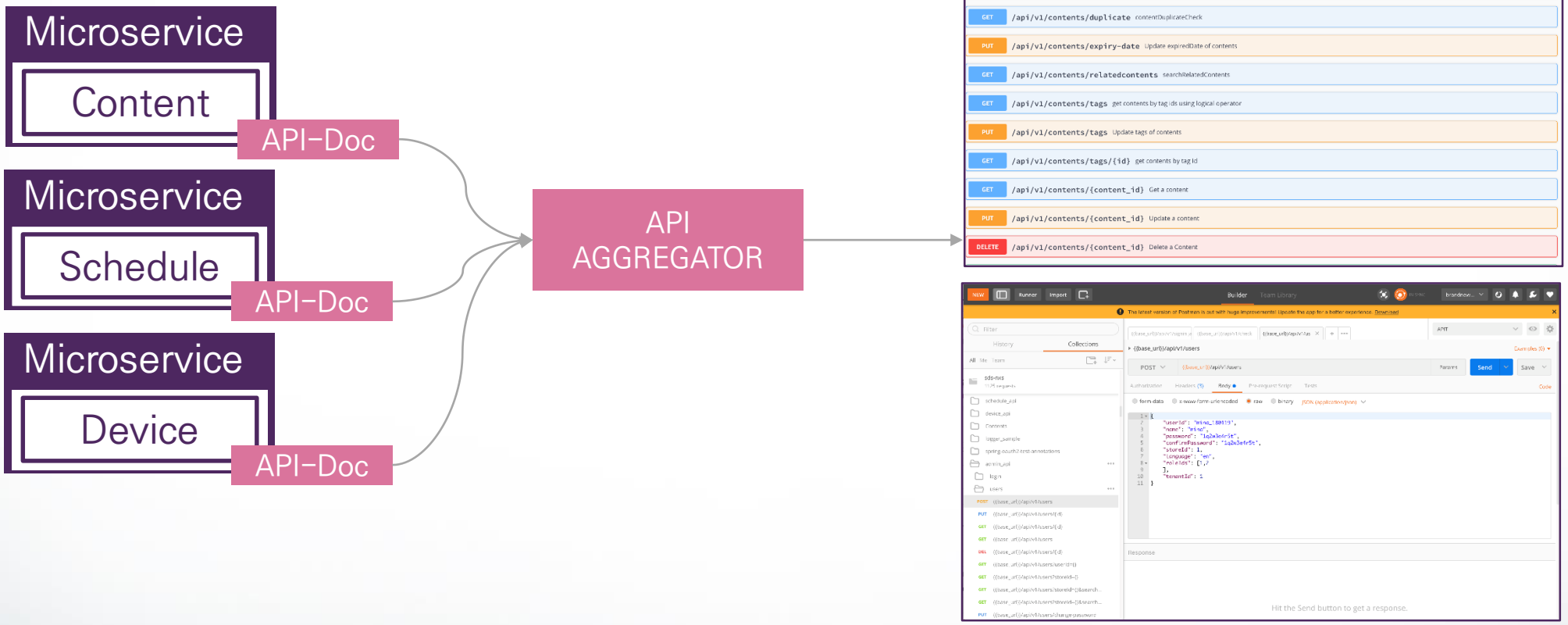
- UI팀에 소속된 Front-End개발자들을 각 서비스팀으로 이동시켜 팀 재구성
- 미사용 API 정리



Phase #2 Process

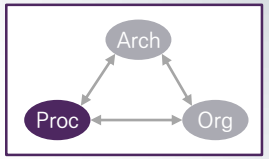


서비스 별 API정보를 통합하여 Swagger와 Postman 제공 → API 상세정보 공유 및 테스트 가능 환경 구축



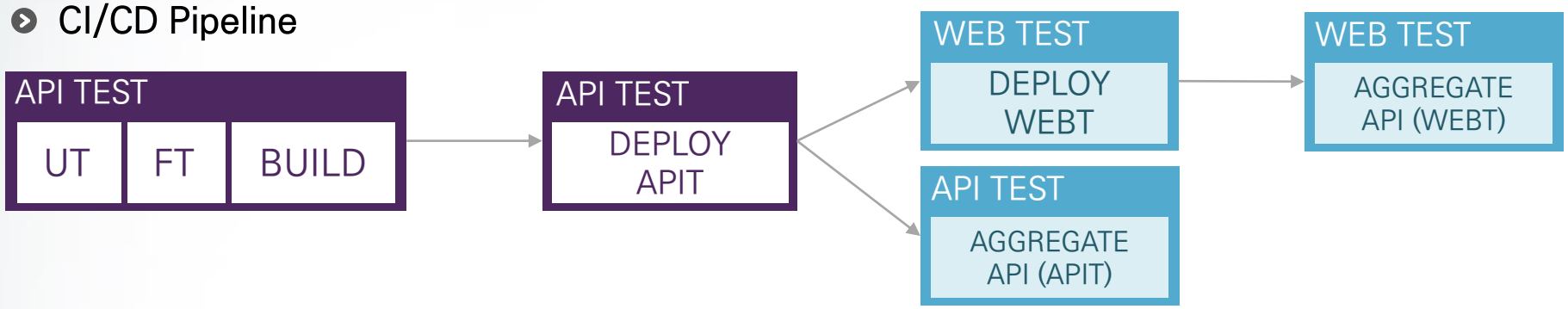
〈REST API 조회 및 테스트〉

Phase #2 Process

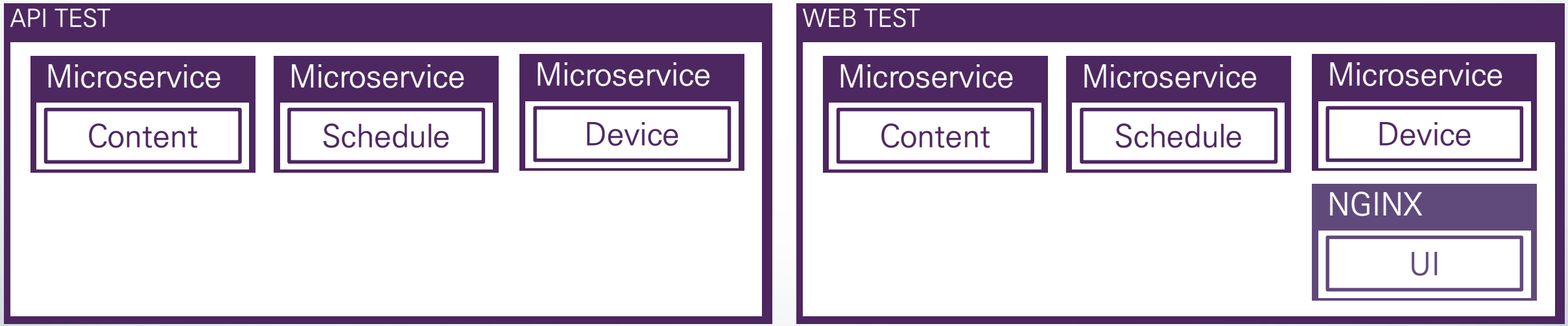


API 스토리와 UI 스토리의 Acceptance Criteria 확인을 위한 개발환경 분리

- CI/CD Pipeline

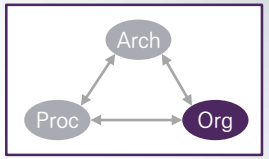


- Deploy Environment

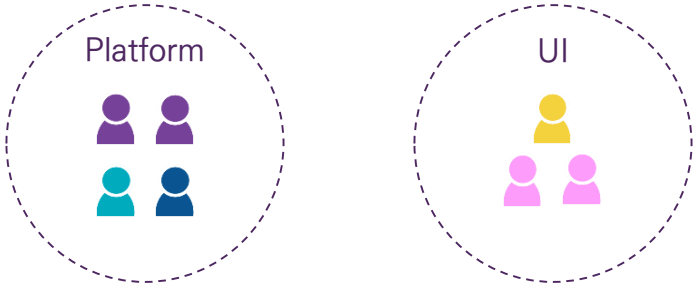


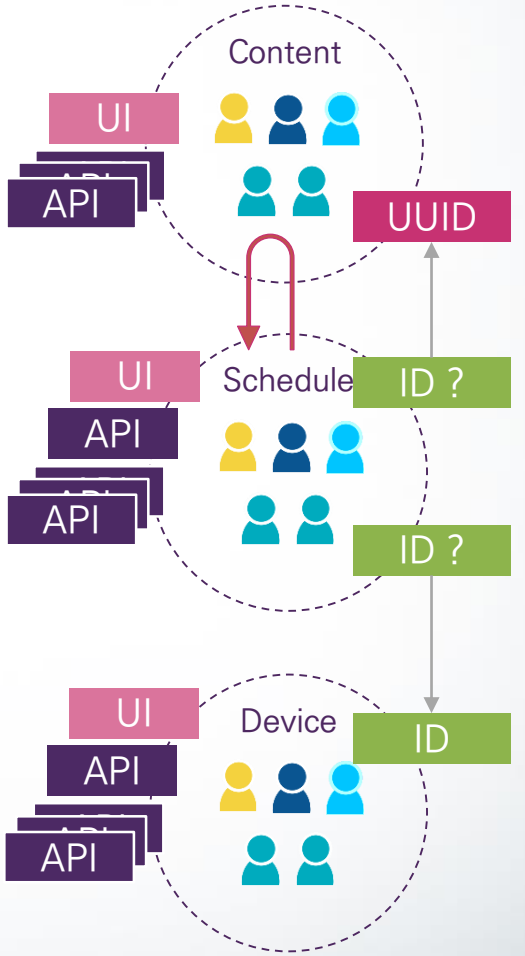
UT: Unit Test / FT : Functional Test

Phase #2 Organization Problem

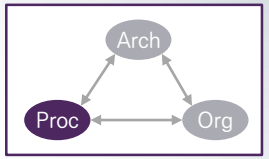


- 서비스 간 호출 증가로 API 제공 서비스가 약속된 기능을 제공하는지 검증 필요
- 기능 변경 후 기존 기능들의 정상 동작 여부 검증 필요



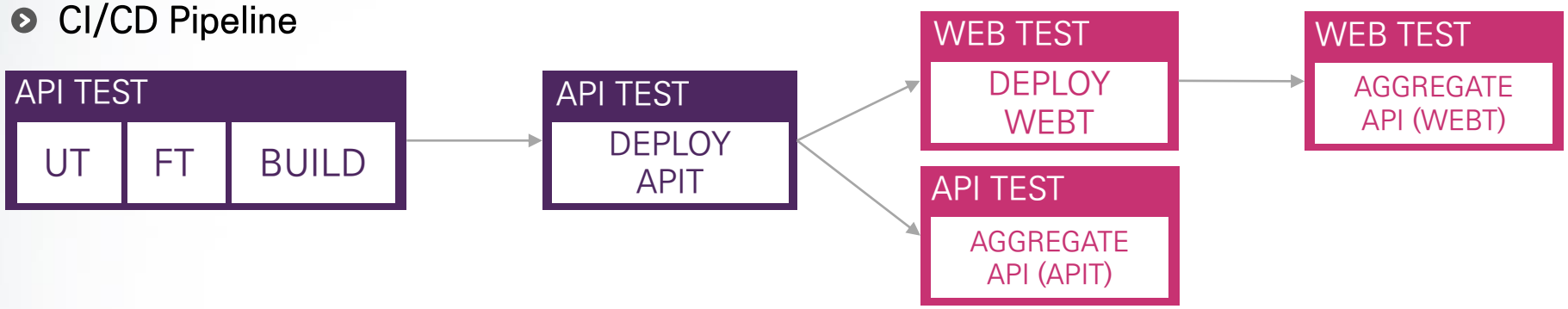


Phase #2 Process Problem

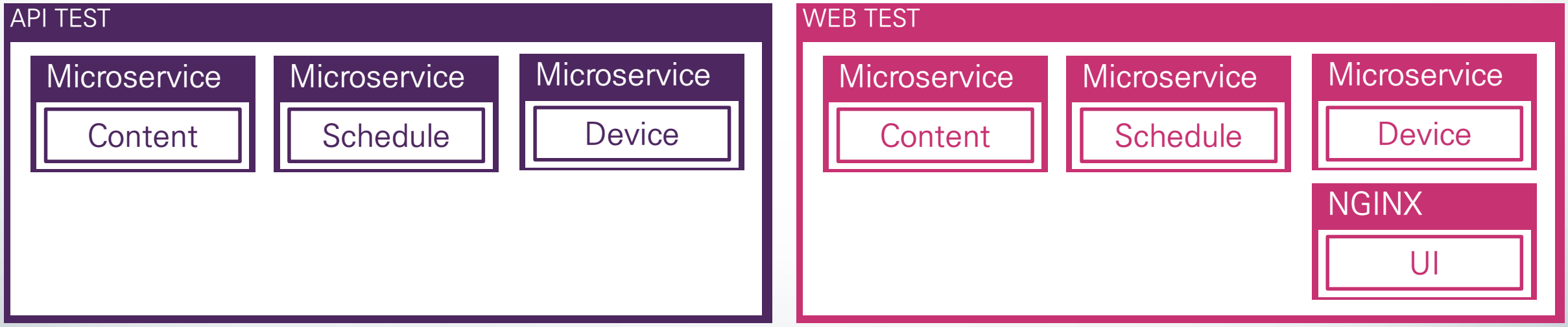


UI 개발 지연 발생 : 서비스 배포 후 오동작 및 Iteration Stakeholder 리뷰 시 Web Test 환경 프리징(2~3일)으로 인한 UI개발 수행 어려움

- CI/CD Pipeline



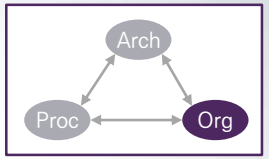
- Deploy Environment



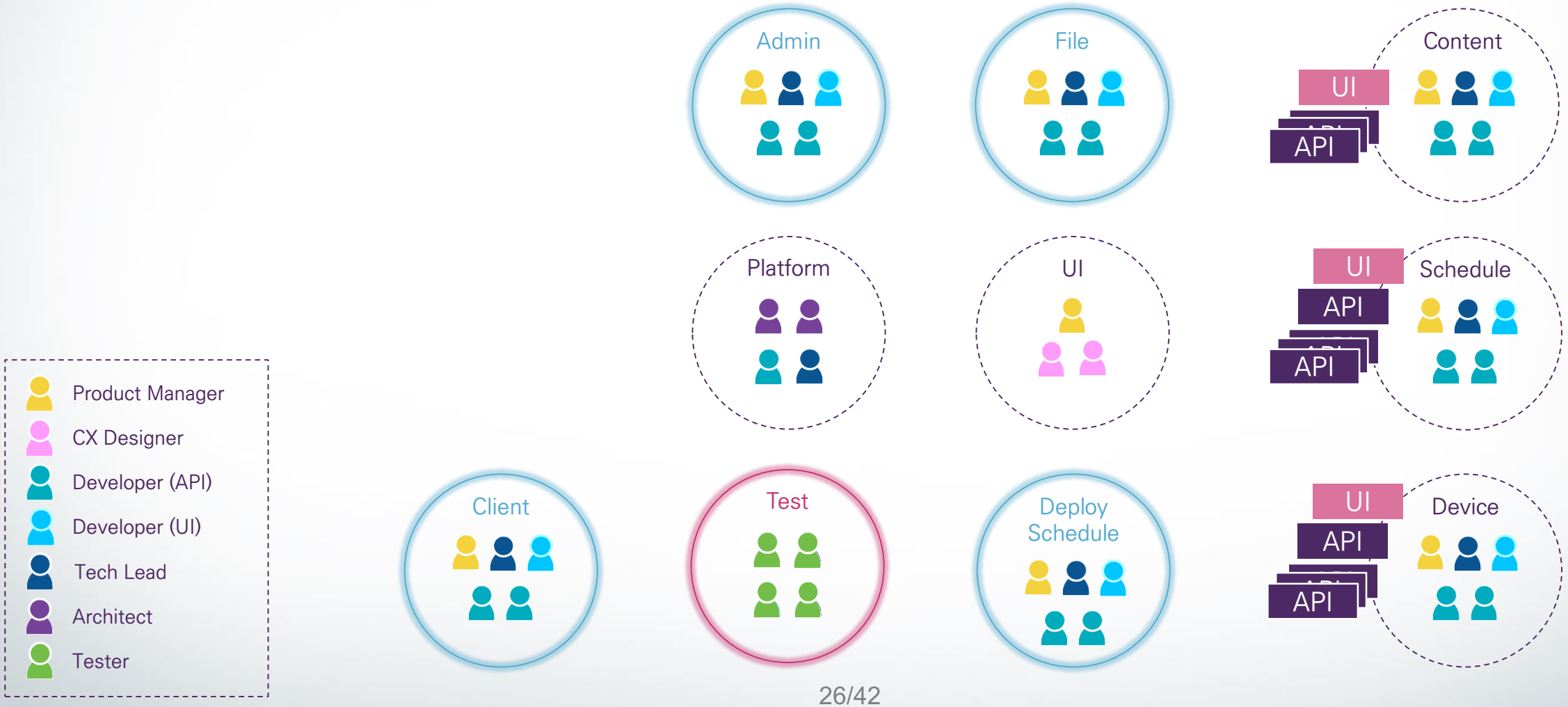
UT: Unit Test / FT : Functional Test

PHASE #3

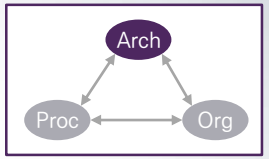
Phase #3 Organization



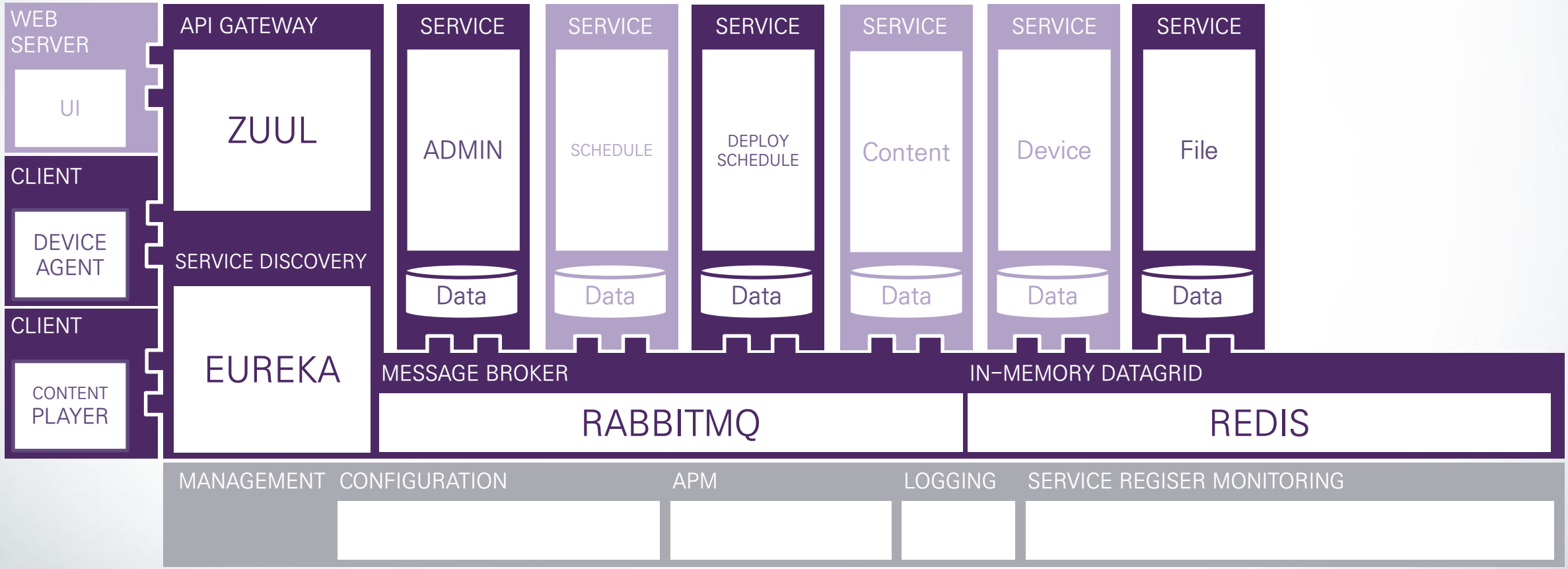
- 도메인 지식 축적으로 Happy Path가 확장되어 서비스의 성격이 명확히 구분됨
- 신규 Service/Client팀과 Regression Test를 전담하는 Test팀 신설



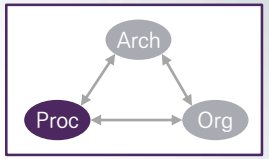
Phase #3 Architecture



서비스 간 호출 증가와 새로운 클라이언트 추가로 End-Point 단일화 필요
→ API GATEWAY & SERVICE DISCOVERY 추가

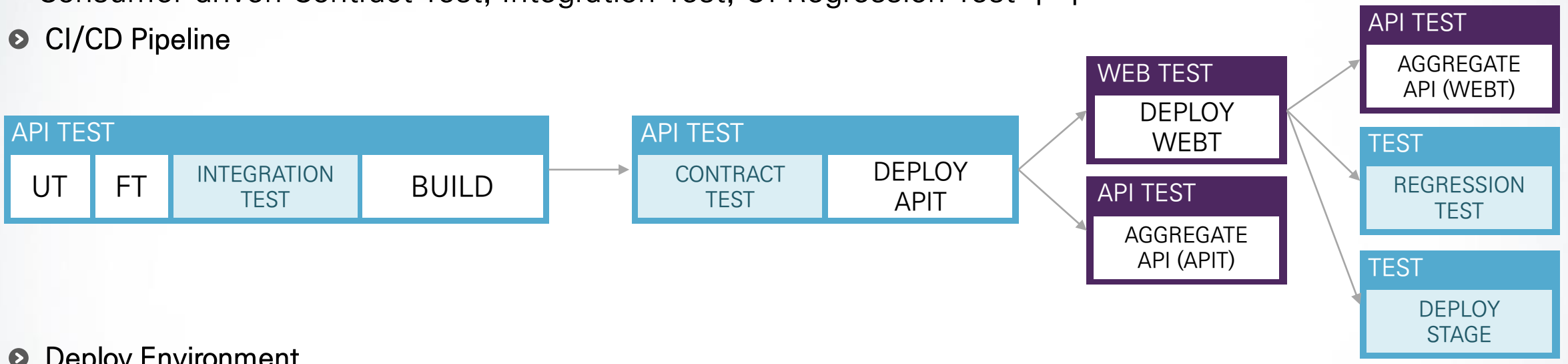


Phase #3 Process

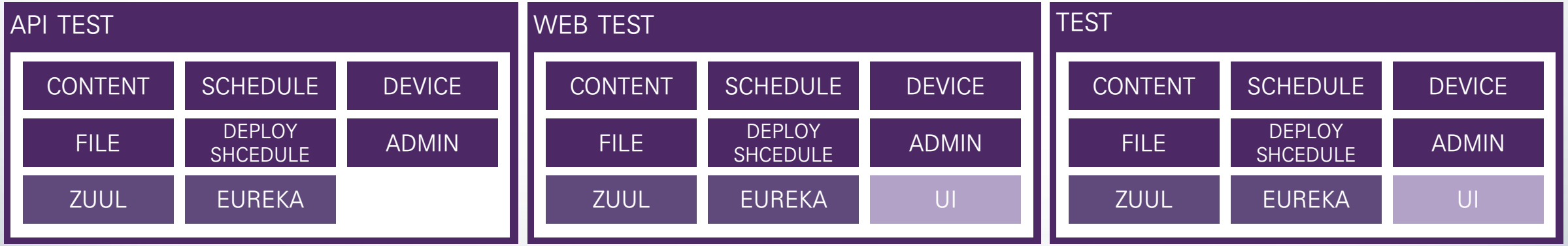


- 상시 시연과 클라이언트팀의 안정적 API 사용을 위한 TEST 환경 추가
- Consumer driven Contract Test, Integration Test, UI Regression Test 추가

- CI/CD Pipeline

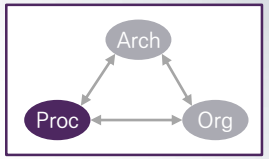


- Deploy Environment

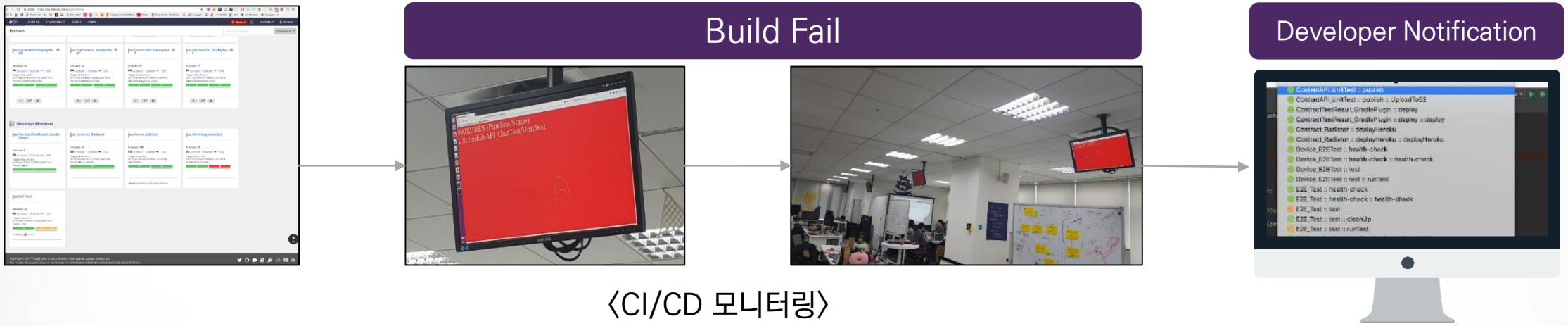


UT: Unit Test / FT : Functional Test

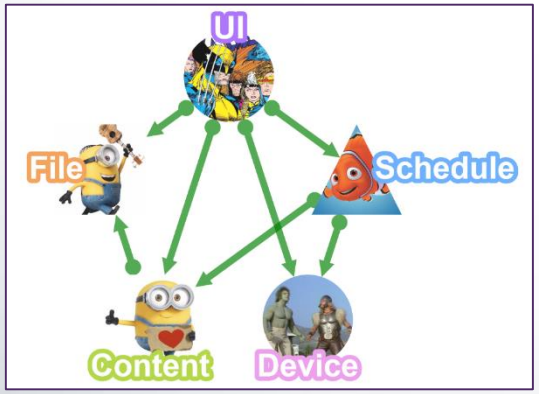
Phase #3 Process



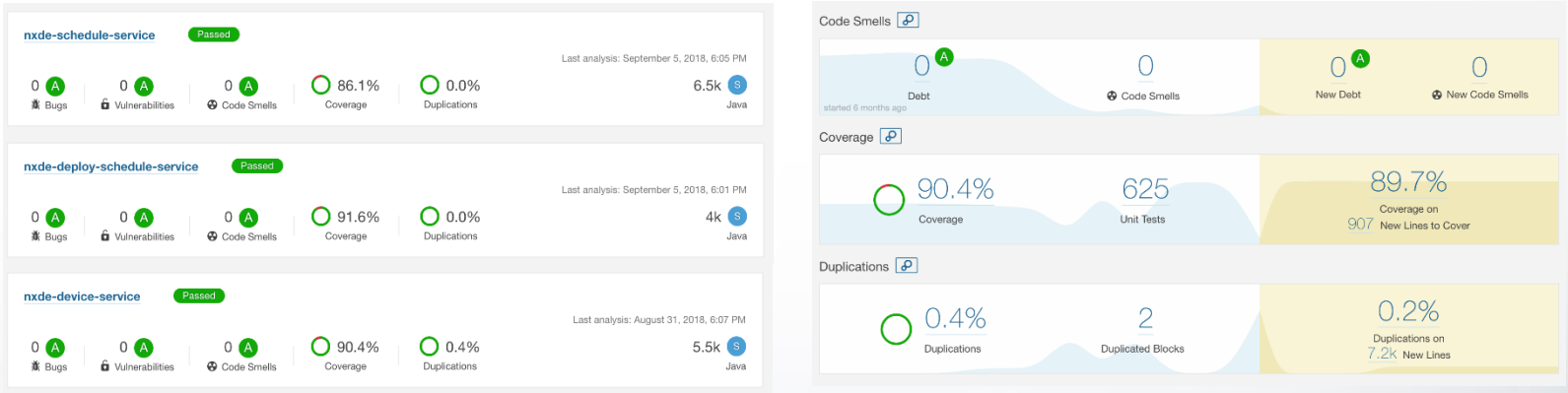
CI/CD 모니터링 환경 및 개인화된 배포 파이프라인 현황 알림 지원
→ 5천 여개의 자동화 테스트 중 실패 발생 시 전체 팀이 즉시 인지하고 조치 가능



〈CI/CD 모니터링〉

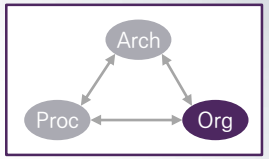


〈Contract Test 모니터링〉

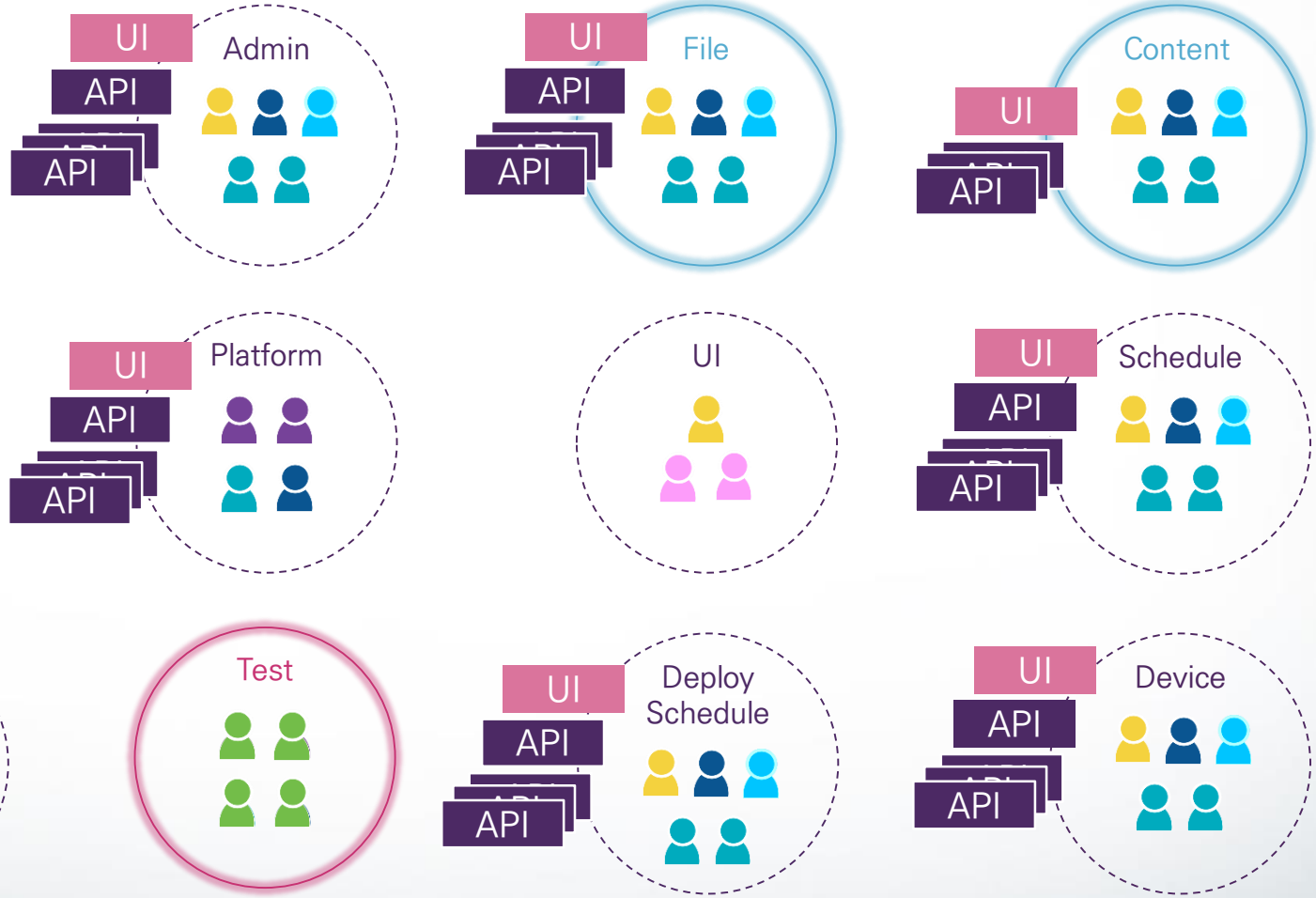


〈Code Smell, Test Coverage, Duplication Check〉

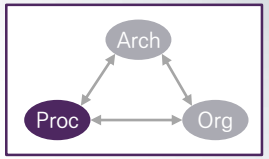
Phase #3 Organization Problem



- 개발 진행되며 역할이 모호해진 Service들이 나타남
- 신규/변경 기능에 한 Contract Test, Regression Test 반영 지연

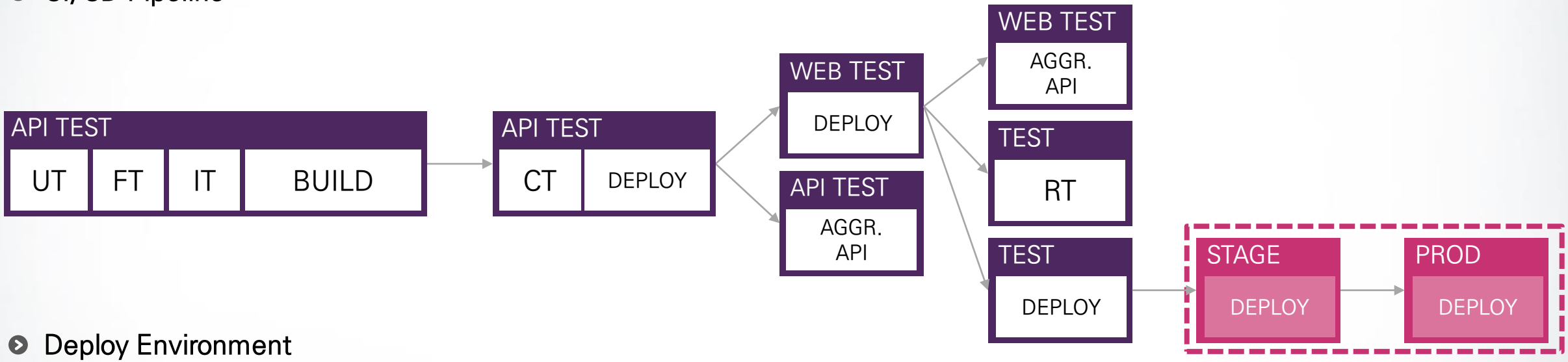


Phase #3 Process Problem

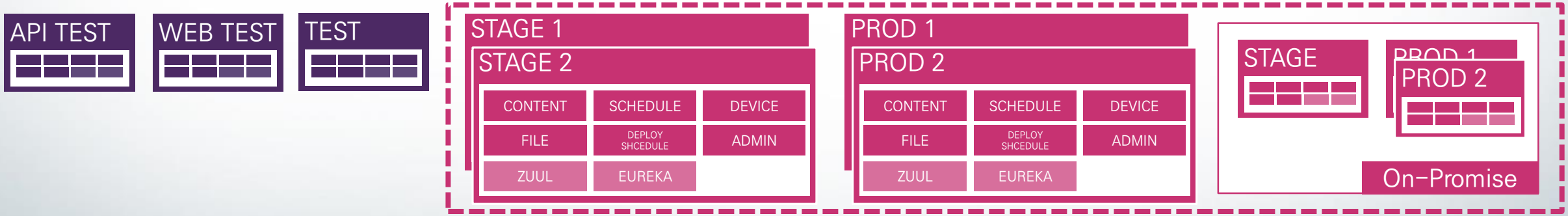


- 향후 운영 위한 검증/운영 환경과 On-Promise 판매 고려한 환경 구성 고려 필요
- 원활한 운영 위한 Monitoring, Logging , Configuration 등의 필요성 대두

CI/CD Pipeline



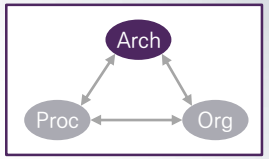
Deploy Environment



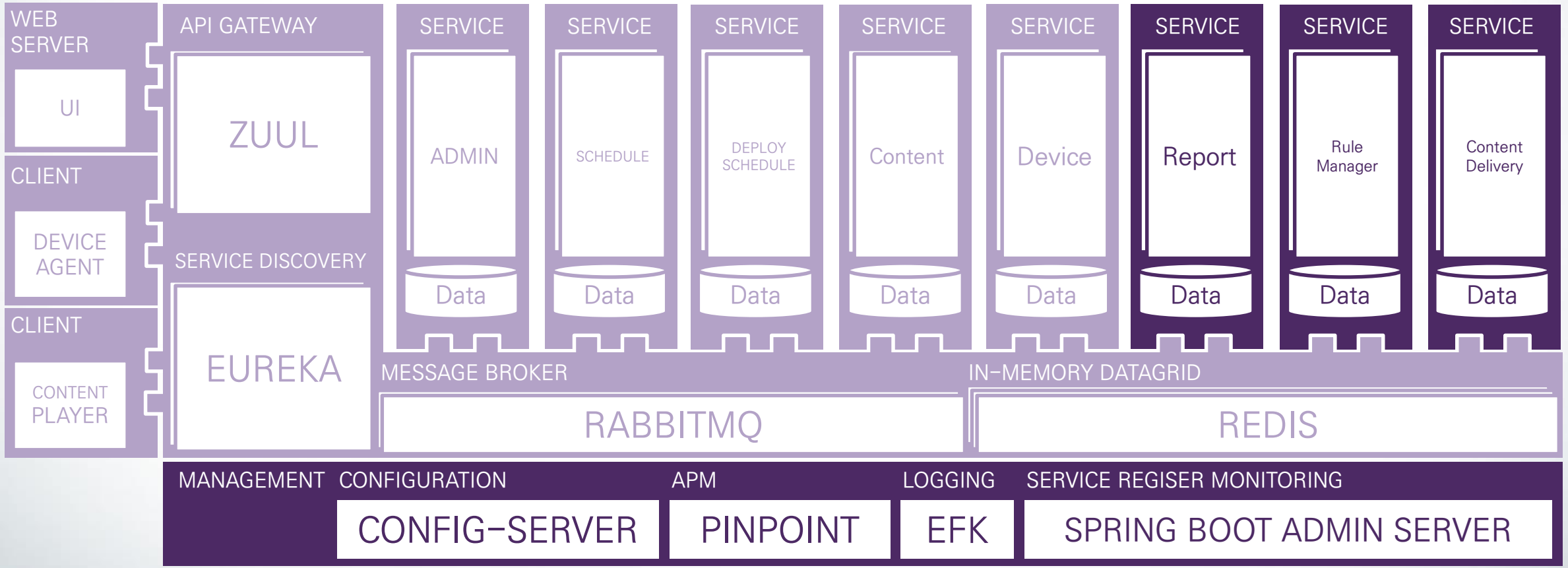
IT : Integration Test / CT : Contract Test / RT : Regression Test

PHASE #4

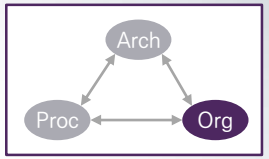
Phase #4 Architecture (Final)



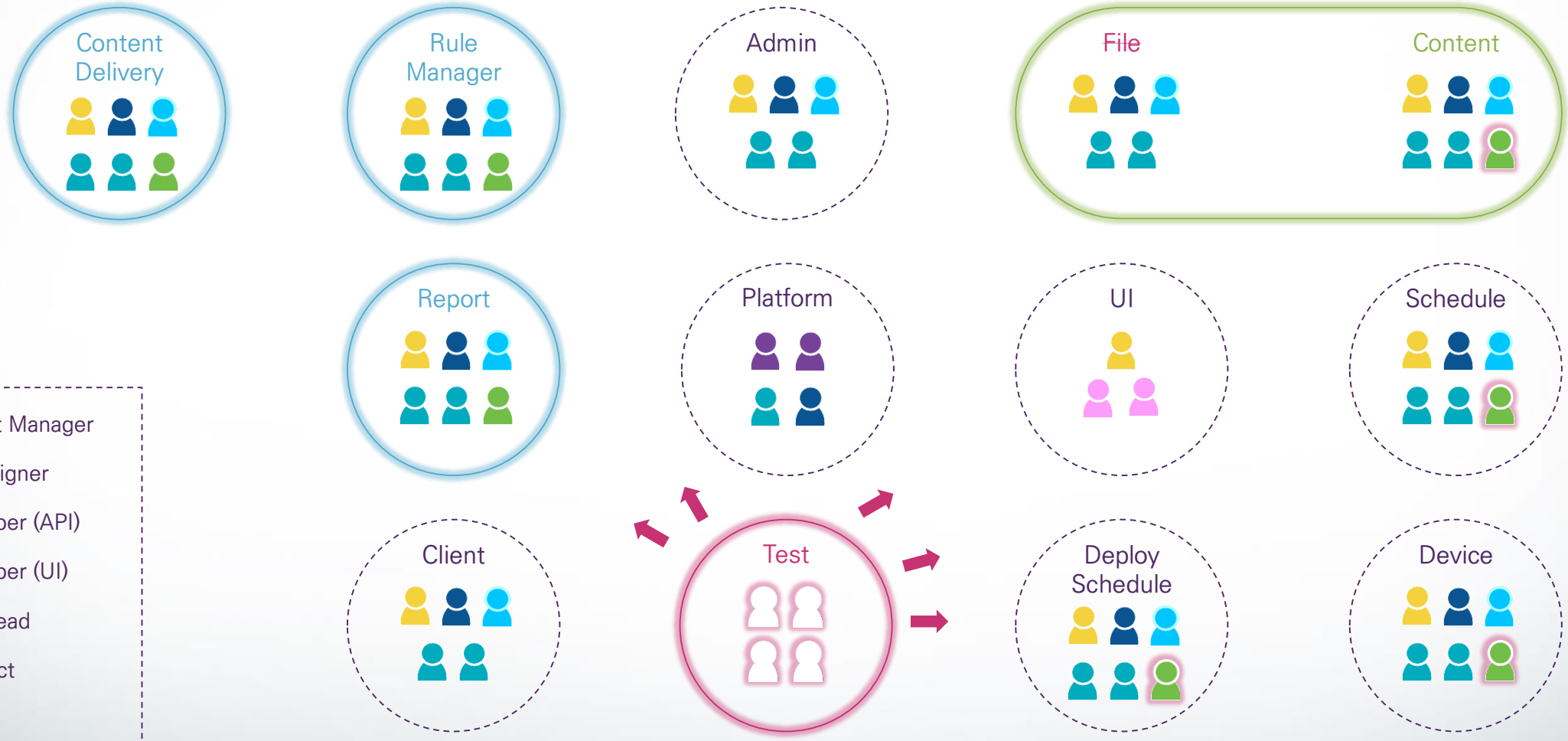
- 운영 위한 SRM, APM, Logging, Configuration 요소들 추가
- Ansible 도입으로 Provisioning 가능하도록 환경 구성 (Prod 환경 기준 VM 27)



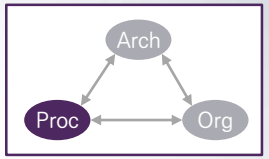
Phase #4 Organization (Final)



- 테스터 인력들을 각 Service 팀으로 재배치
- 역할이 모호한 서비스는 정리하고, 비즈니스 요건 대응을 위한 신규팀이 추가됨

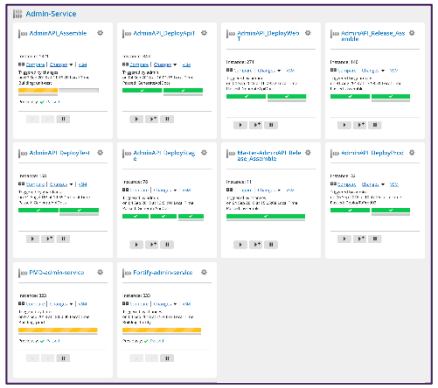


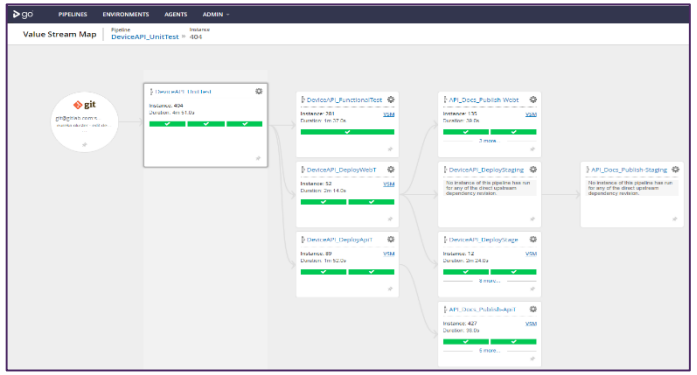
Phase #4 Process (Final)



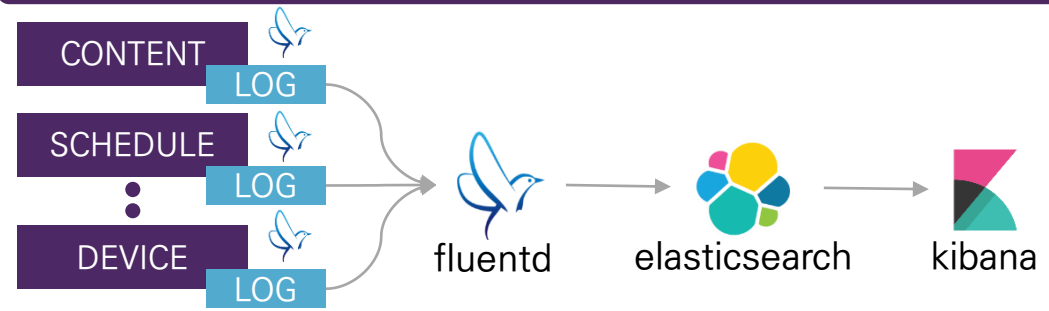
- 개발/테스트/운영을 위한 5개 환경과 210개 파이프라인 구성
- Centralized Logging 위한 EFK 및 Centralized Monitoring 위한 Pinpoint 적용

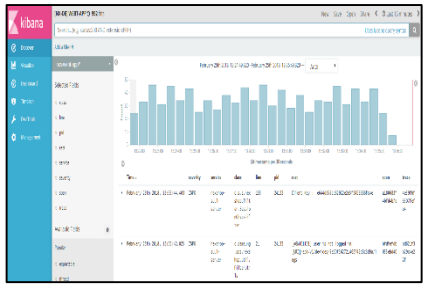
CI/CD Pipeline



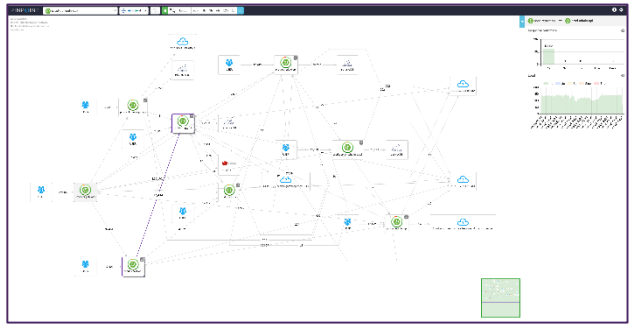


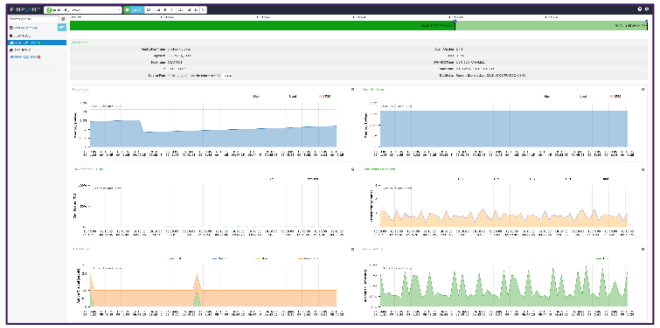
Centralized Logging



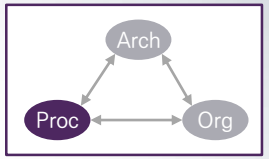


Centralized Monitoring



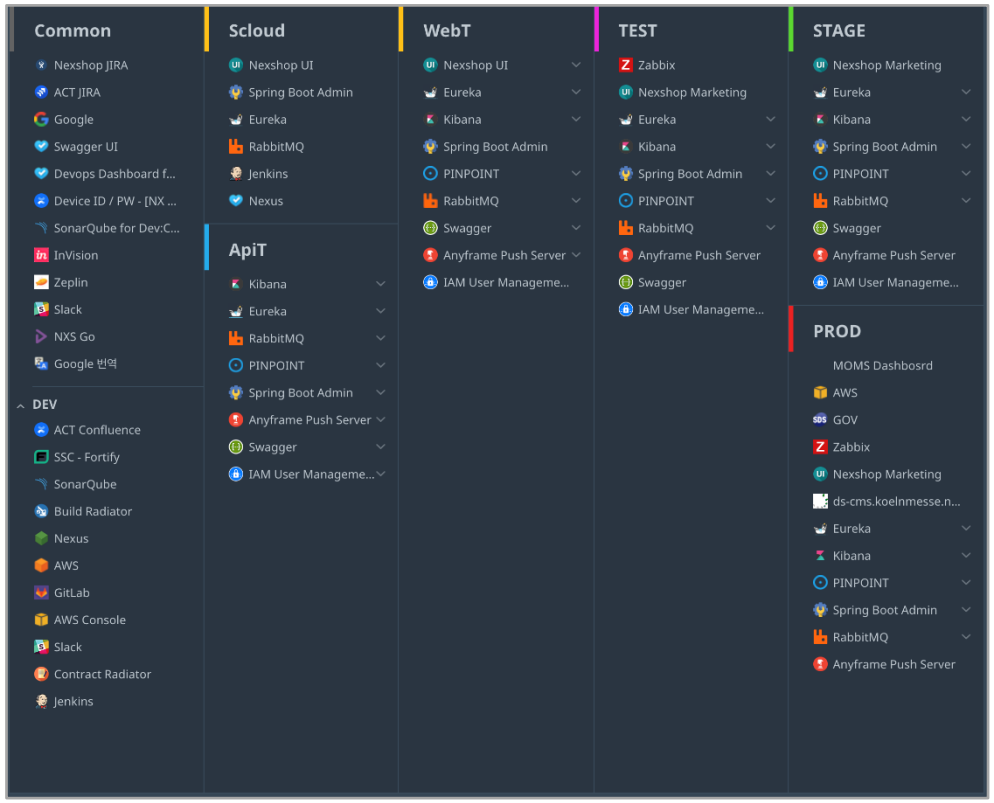


Phase #4 Process (Final)

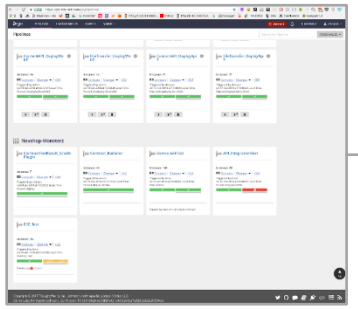


- 환경 별 개발자 북마크 도입으로 개발/운영 편의성 제공
- Service Health(On/Off) 상태 알림 제공으로 장애상황의 빠른 인지 가능

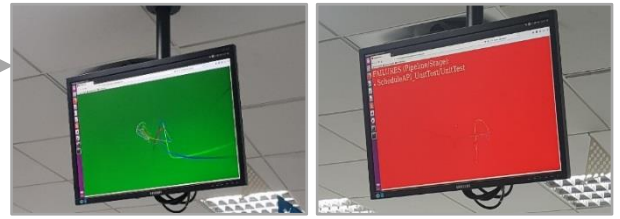
Environment Support



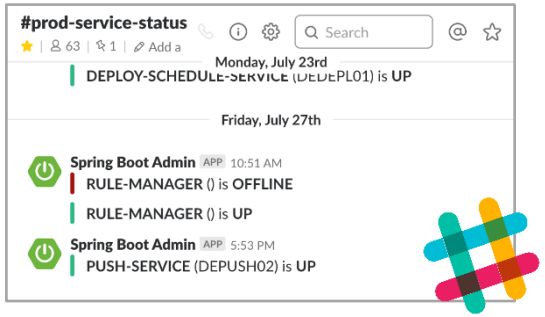
Centralized Logging



Deploy



Status Notification



Agile팀의 MSA 적용 고군분투기 – To Microservices and Beyond

Lessons Learned

Lessons learned

Product 팀이 직접 고객 채널에 참여하며 프로젝트 수행 필요

Dynamic Service Discovery

MSA 수행 방식

도메인 지식 및 아키텍처 준비 상태에 따라 다른 접근이 필요

ORGANIZATION & PROCESS 〉 **ARCHITECTURE**

Domain Driven Design

Centralized Monitoring

Centralized Logging

Database per Service

Iteration #0 의 필요성

Strangler Pattern VS. Big-Bang Rewriting 중 비즈니스 상황에 따라 선택

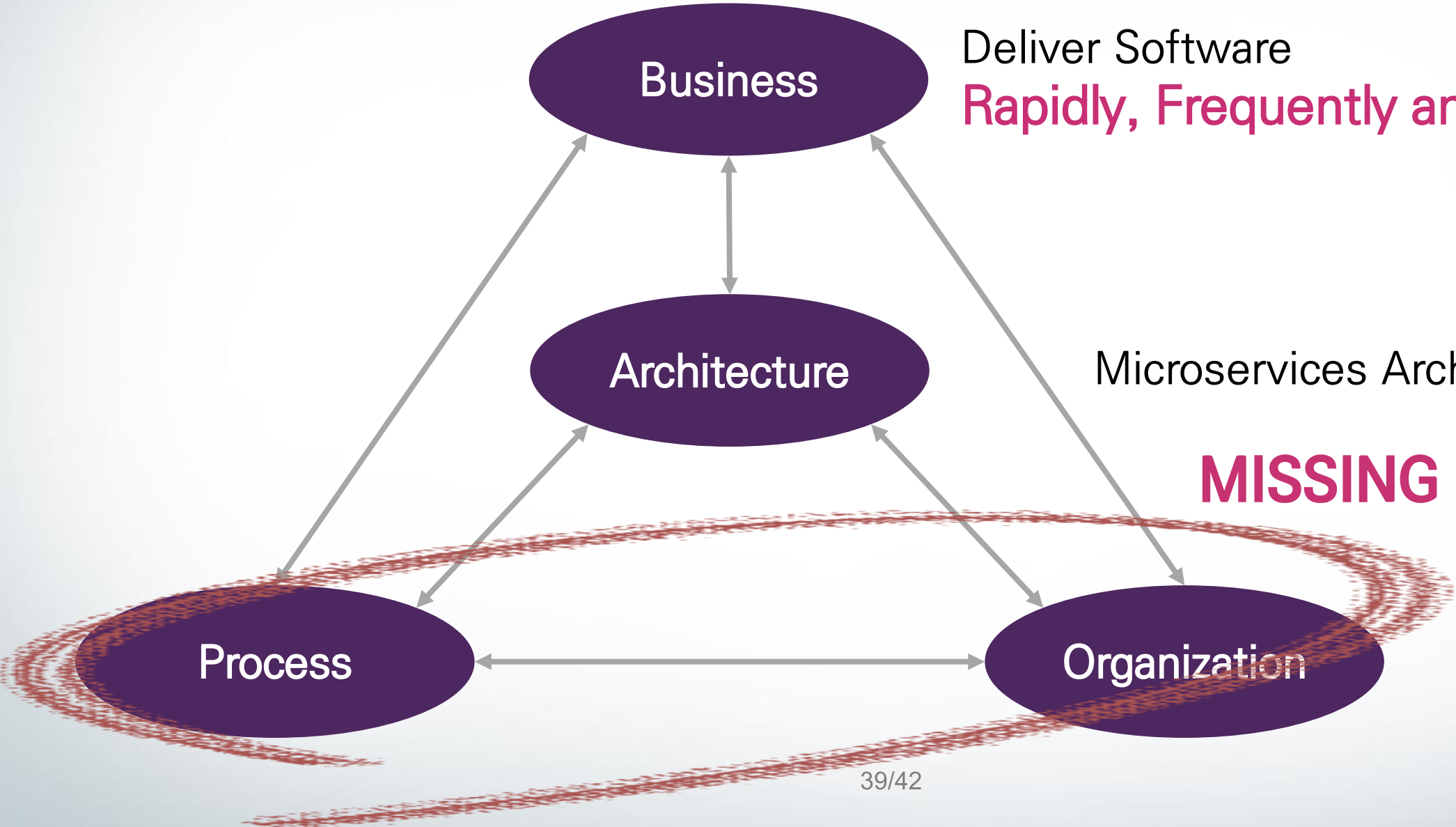
대형 Product팀에 맞는 R&R 이 사전 정리 필요

Service Integration Contract Test

Decompose by business capability

심각도와 우선순위의 부여기준 사전 마련

Lessons learned



Deliver Software
Rapidly, Frequently and Reliably

Microservices Architecture

MISSING PART

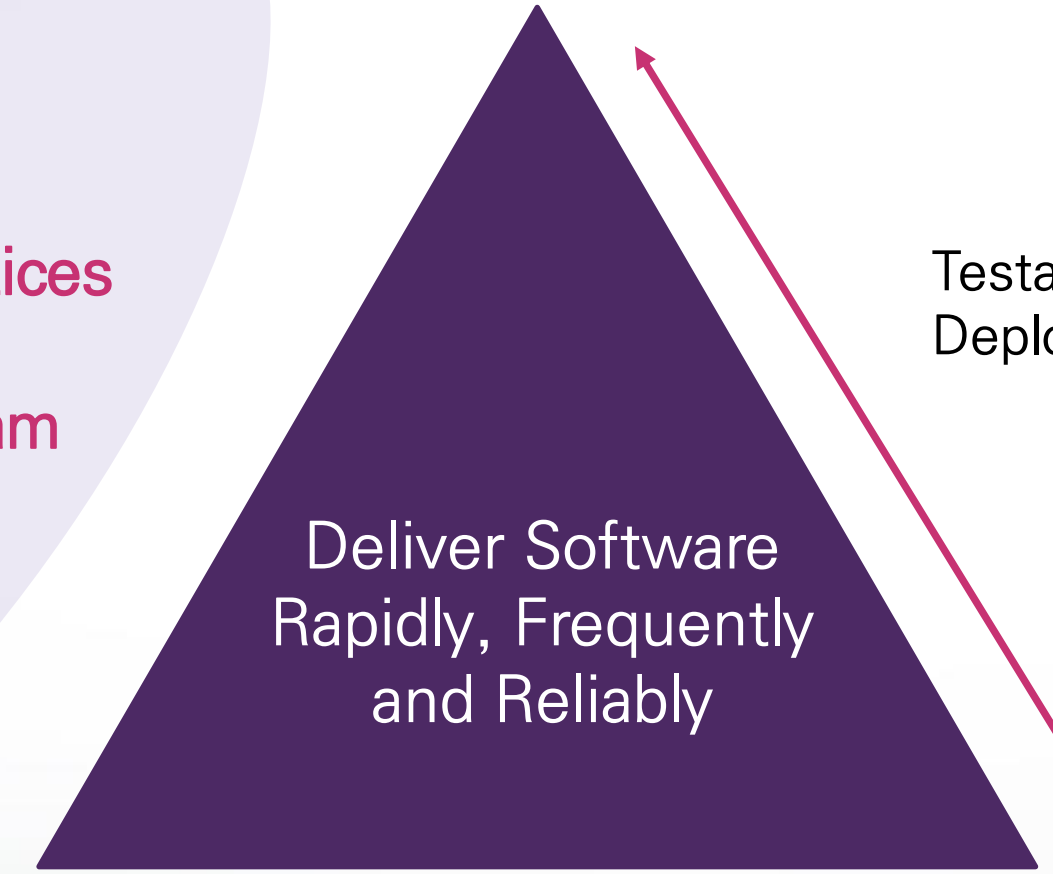
Agile팀의 MSA 적용 고군분투기 – To Microservices and Beyond

To Microservices and Beyond

To Microservices and Beyond

PROCESS: DevOps/Continuous Delivery & Deployment

Agile Practices
&
Agile Team



Testability
Deployability

ORGANIZATION :
Small, Cross-Functional,
Autonomous team

Enable Autonomy

ARCHITECTURE :
Microservices

To Microservices and Beyond in SDS

Architecture
MSA CoE

Organization
Agile Core Team

Process
개발 플랫폼

DEP, CMP를 활용한
MSA 전환

엔터프라이즈 Agile
Transformation

adPaaS 를 통한
DevOps 체계

DEP: Digital Enterprise Platform / CMP: Cloud Management Portal / adPaaS : Application Development Platform as a Service

Q & A

Thank you