Whatap DevOps Day

구준한

개요

WhaTap DevOps Day

- Observability Practices on AWS
- 클라우드와 개발자, 모놀리틱부터 오케스트레이션까지
- 롯데ON MSA 모니터링 최적화 사례
- 금융권 퍼블릭 클라우드, DevOps 구축 여정
- 성장하는 엔지니어 학습 문화
- 와탭랩스 DevOps 이야기

개요

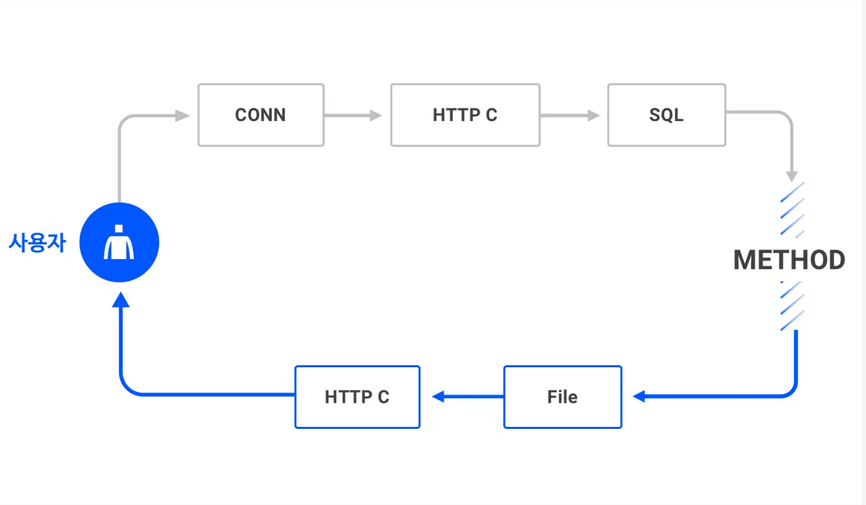
- 대한민국 SaaS 모니터링(APM) 서비스 기업
- Server, Application, Database, Kubernetes 성능 모니터링 솔루션 개발

- IaaS(Infrastructure-as-a-service): IT 인프라 제공
- PaaS(Platform-as-a-service): IaaS + 개발툴, 기능, 애플리케이션 배포 제공
- SaaS(Software-as-a-Service): 서드파티, 호스팅 방식으로 소프트웨어 제공

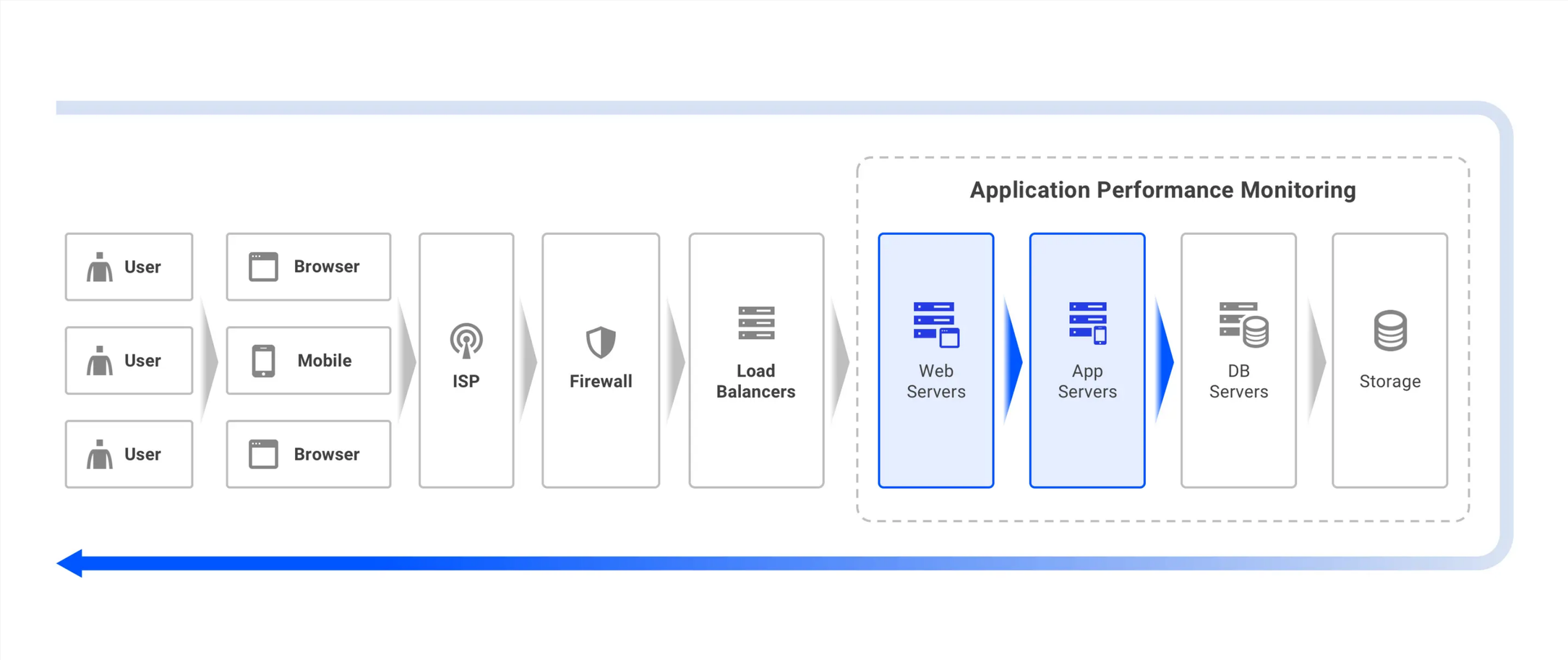
APM

- Application Performance Management의 약자
- A : Web Application을 말하며 기업의 웹 서비스 성능을 관리하는 서비스
- P : Application의 성능(Performance)을 의미

  주로 Web Service의 응답속도를 통해 측정
- M : 웹 서비스의 성능을 관리하기 위한 Management 또는 Monitoring

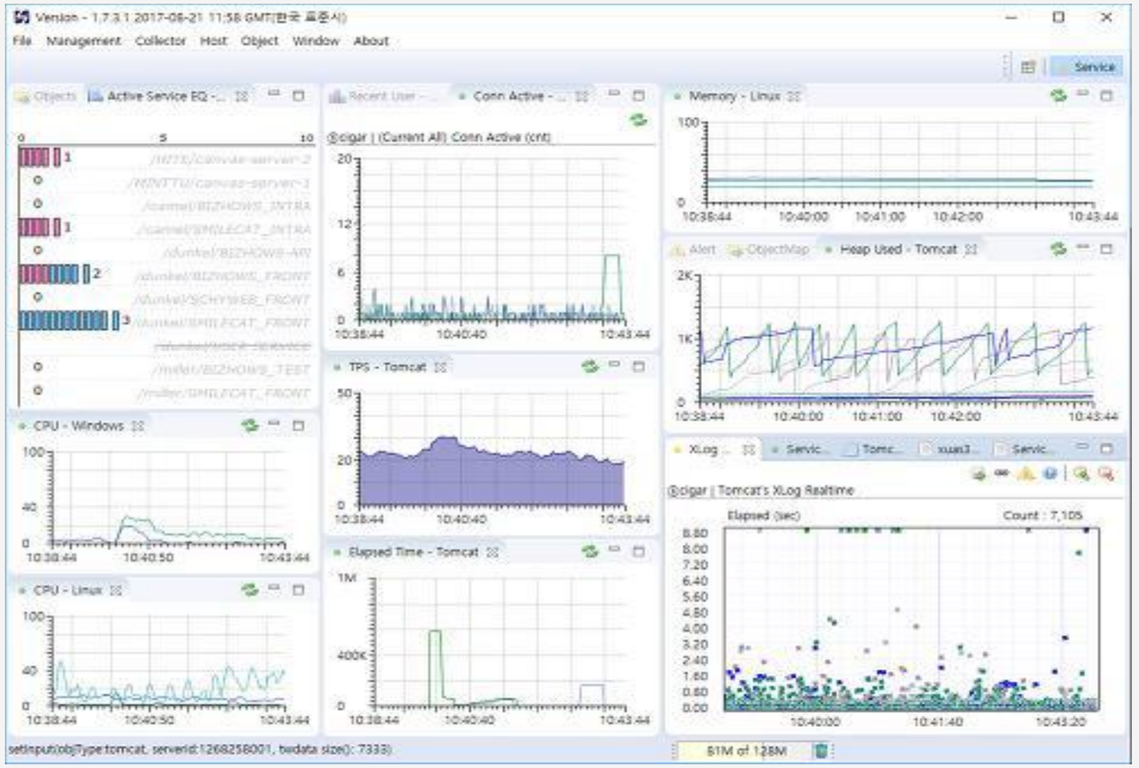


APM


Application Performance Monitoring
User
User
User
Browser
Mobile
Browser
ISP
Firewall
Load Balancers
Web Servers
App Servers
DB Servers
Storage

개요

Jennifer



Scouter

Observability (관찰 가능성)

- Observability : 오직 시스템의 외부 출력만을 이용해서 시스템의 현재 상태를 이해할 수 있는 능력

(Rudolf E. Kálmán)

- Monitoring : 시스템이 잘 동작하고 있는지 말해준다.

  → Observability를 통해 시스템이 동작하지 않는 이유를 이해할 수 있게 해준다.

Spinnaker

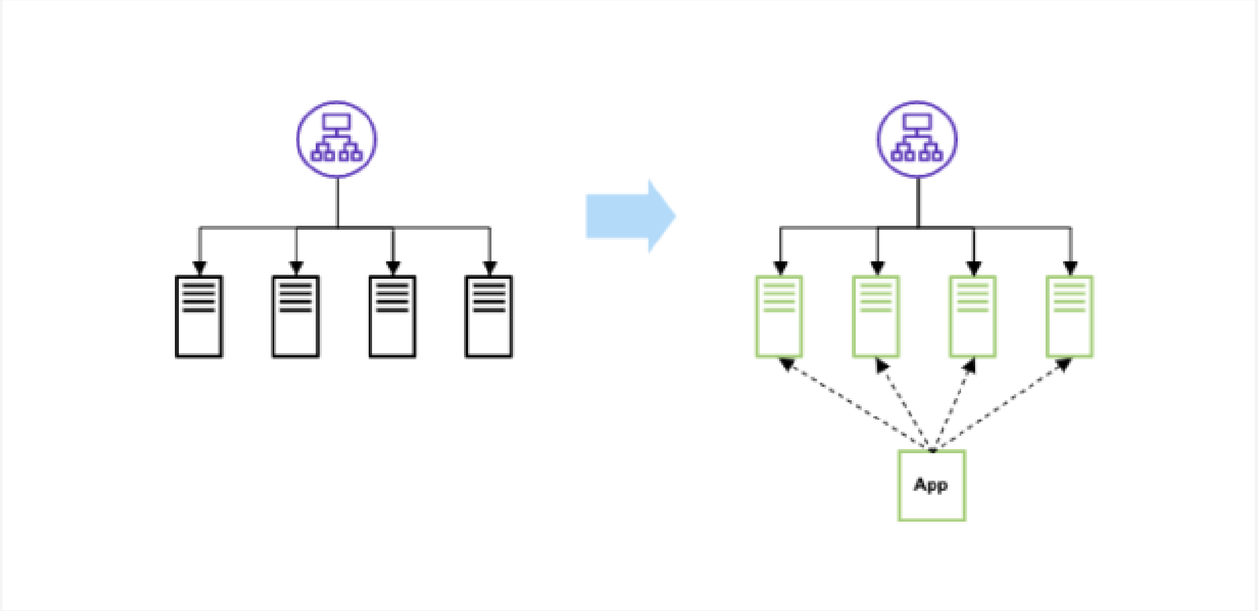
- Netflix에서 개발 한 지속적 배포 소프트웨어 플랫폼
- 현재는 Google에서 확장하여 오픈 소스로 개발 중
- AWS, GCP, Azure 등 Public Cloud 및 Private Cloud 지원
- 카나리 배포와 같은 고급 배포 방법 구현 가능

배포 전략 (Deployment Strategy)

- 인플레이스 배포 (In-place Deployment)
- 롤링 배포 (Rolling Update Deployment)
- 블루/그린 배포 (Blue/Green Deployment)
- 카나리 배포 (Canary Deployment)

인플레이스 배포 (In-place Deployment)

- 사용중인 환경에서 새로운 변경사항이 있는 Application만 반영하는 방법 (가장 전통적인 방식)
- Application을 일시정지한 후 새로운 버전을 배포(Deploy)한 후 다시 실행
- 무중단 배포 가능

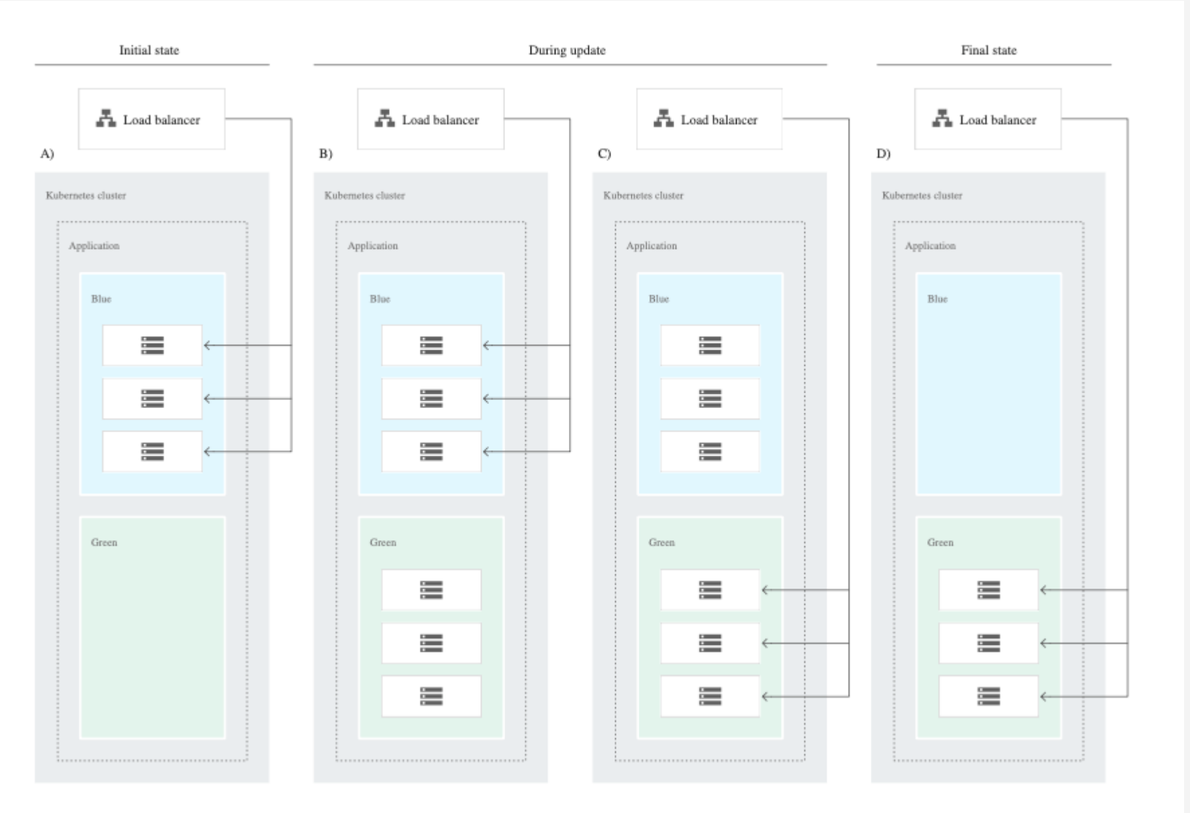


롤링 배포 (Rolling Update Deployment)

- 무중단 배포 기법 중 하나
- 구 버전에서 신 버전의 인스턴스로 트래픽을 점진적으로 전환하며 구 버전의 인스턴스는 점차 삭제
- 사전에 서버 처리 용량을 고려 필요

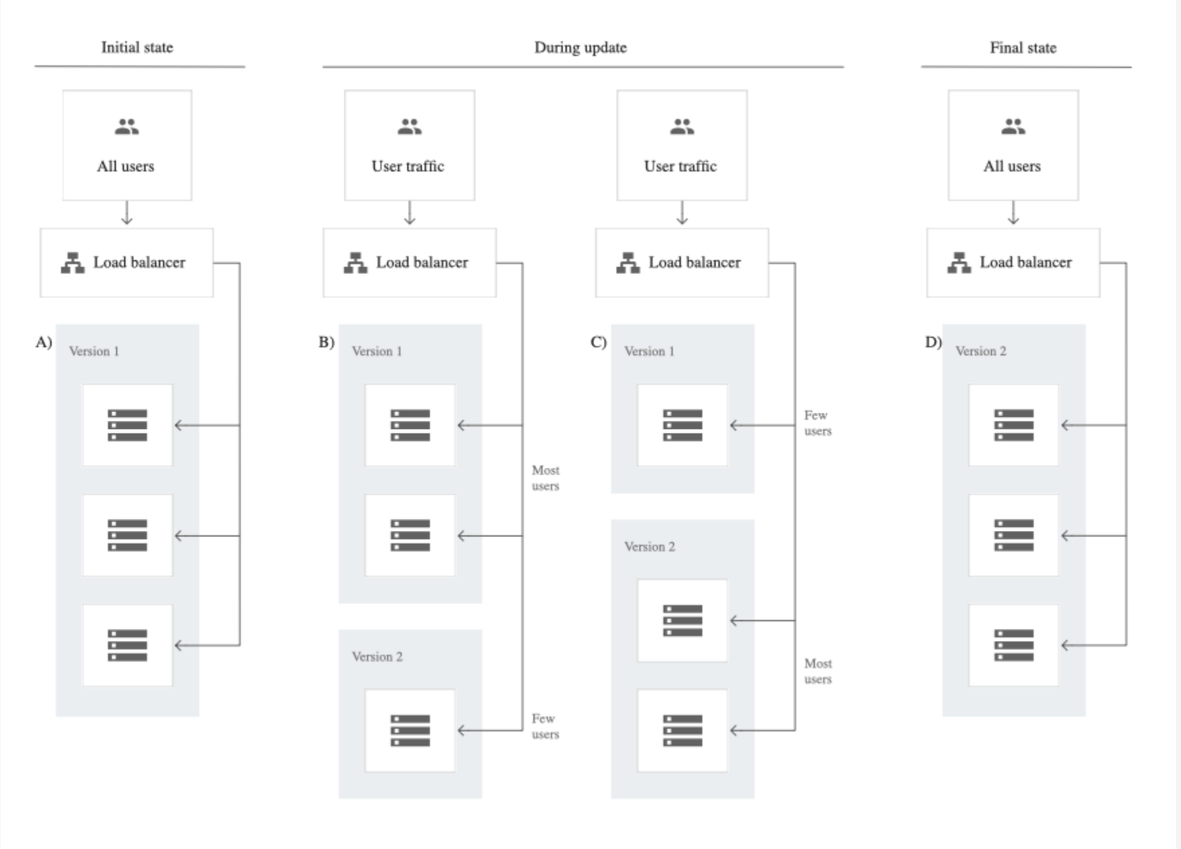
블루/그린 배포 (Blue/Green Deployment)

- 무중단 배포 기법 중 하나
- 두 개의 그룹으로 구성하여 배포 진행
- 신 버전을 배포하고 일제히 모든 트래픽을 전환
- 빠른 롤백 가능



카나리 배포 (Canary Deployment)

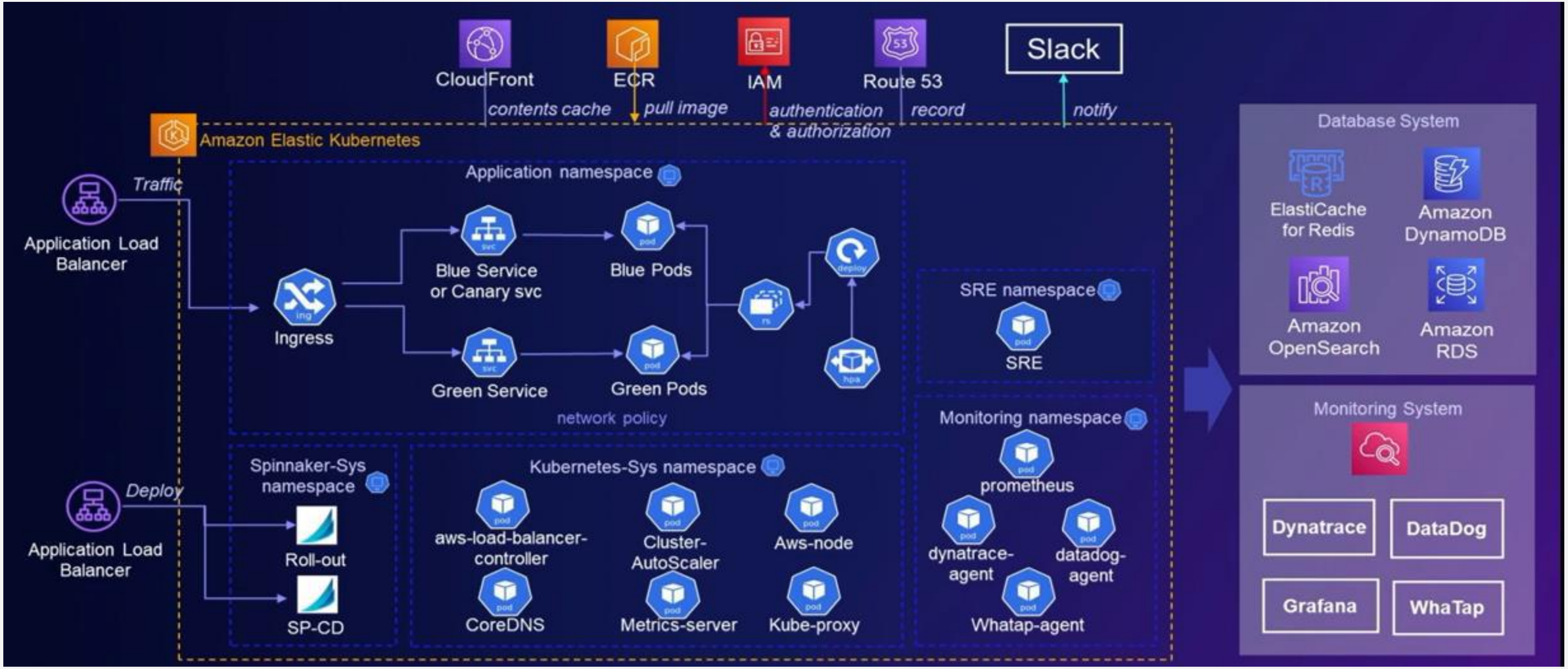
- 변경사항을 부분적으로 출시한 후 기준 배포와 비교하여 성능 평가
- 점진적 출시로 인해 일부 기간 모니터링을 통해 안정성을 검증하기에 전체 출시가 지연 될 수 있음



롯데ON MSA 모니터링 최적화 사례

HPSC LAB
High Performance & Secure Computing

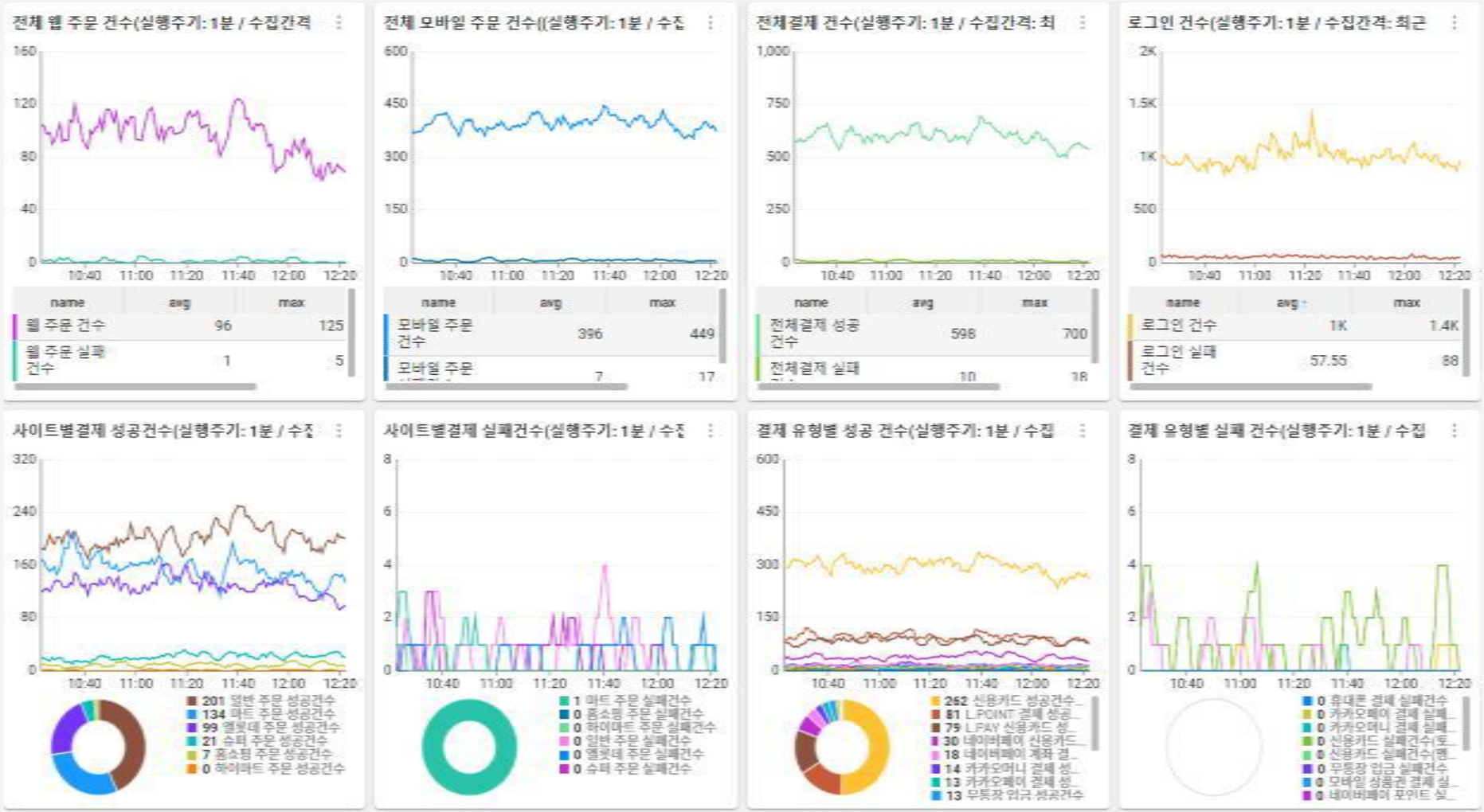
롯데ON 아키텍처



롯데ON MSA 모니터링 최적화 사례

HPSC LAB
High Performance & Secure Computing

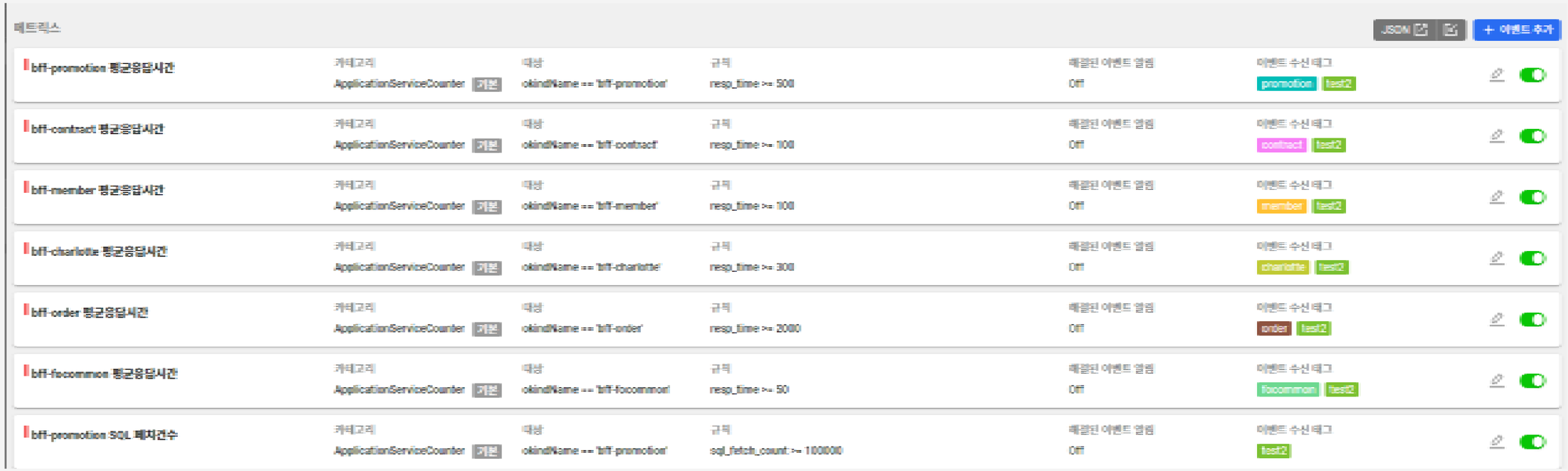
모니터링 과제



롯데ON MSA 모니터링 최적화 사례

HPSC LAB
High Performance & Secure Computing

모니터링 과제



롯데ON MSA 모니터링 최적화 사례

HPSC LAB
High Performance & Secure Computing

모니터링 과제

▪ 프로젝트 에러현황

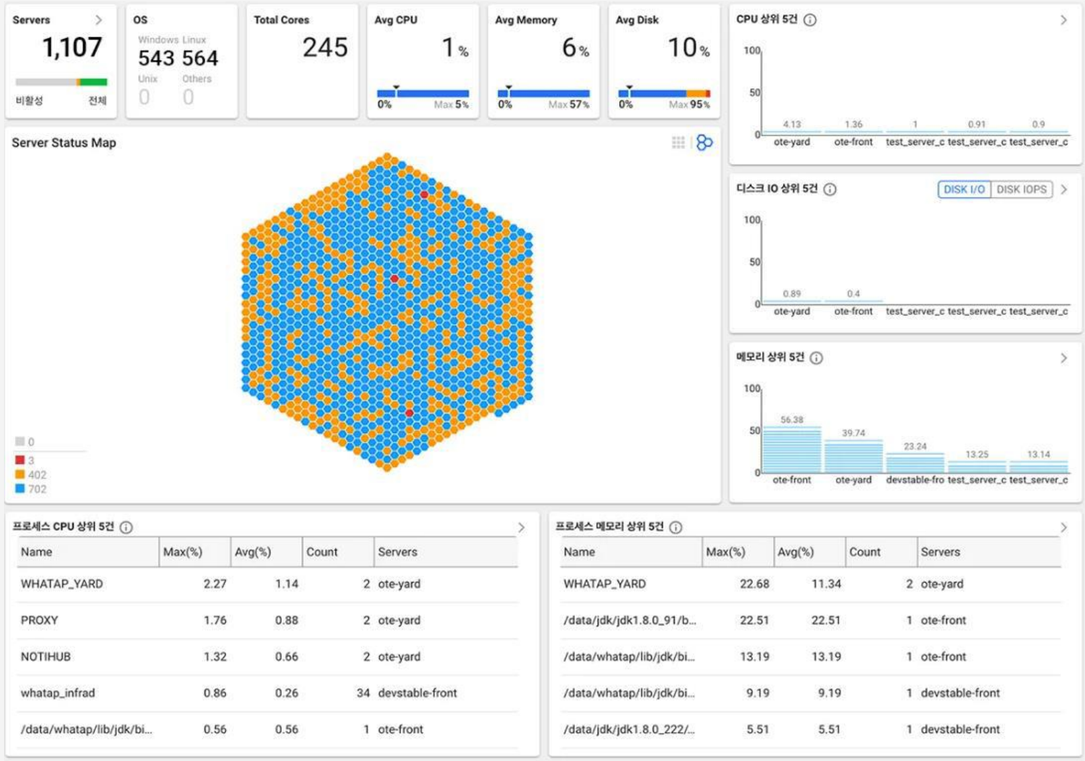
프로젝트명	에러 클래스	2주전 05/19 ~ 05/25	지난주 05/26 ~ 06/01	증감건	증감률
fo	whatap.error.STATUS_ERROR	509,611	408,825	-100,786	-19.78%
	java.net.SocketTimeoutException	8,428	9,109	681	8.08%
	java.io.IOException	3,641	4,600	959	26.34%
	org.springframework.web.HttpRequestMethodNotSupportedException	4,227	4,218	-9	-0.21%
	org.springframework.web.util.NestedServletException	2,600	1,828	-772	-29.69%
	whatap.error.SLOW_HTTPC	1,052	1,277	225	21.39%
	io.netty.handler.timeout.ReadTimeoutException	6,862	761	-6,101	-88.91%
	com.lotteon.focommon.v1.config.FOMessageException	513	698	185	36.06%
	org.springframework.web.reactive.function.client.WebClientResponseException$InternalServerError	781	270	-511	-65.43%
	java.io.FileNotFoundException	279	180	-99	-35.48%
	org.springframework.web.HttpMediaTypeNotAcceptableException	50	71	21	42.0%
	java.lang.NullPointerException	97	58	-39	-40.21%
	org.apache.http.NoHttpResponseException	47	32	-15	-31.91%
	org.springframework.http.converter.HttpMessageNotReadableException	134	27	-107	-79.85%
	org.eclipse.jetty.http.BadMessageException	11	20	9	81.82%
	java.net.SocketException	5	12	7	140.0%
	java.lang.IllegalArgumentException	3	9	6	200.0%
	java.util.concurrent.TimeoutException	40	6	-34	-85.0%
	org.springframework.web.multipart.support.MissingServletRequestPartException	1	5	4	400.0%

incoming-webhook 앱 오후 8:09
프로젝트 이름 : bfb
프로젝트 코드 : 19429
애플리케이션 이름 : ip-10-194-14-55.ap-northeast-2.compute.internal
이벤트 메시지 : bfb-product-prd-tag-20220616-120000-dcdf7d8b6-j4jmq-10.194.14.55 메모리 사용률 98%이상
이벤트 시작 시간 : 2022-06-19 20:09:16 +0900

롯데ON MSA 모니터링 최적화 사례

HPSC LAB
High Performance & Secure Computing

MSA 모니터링 과제



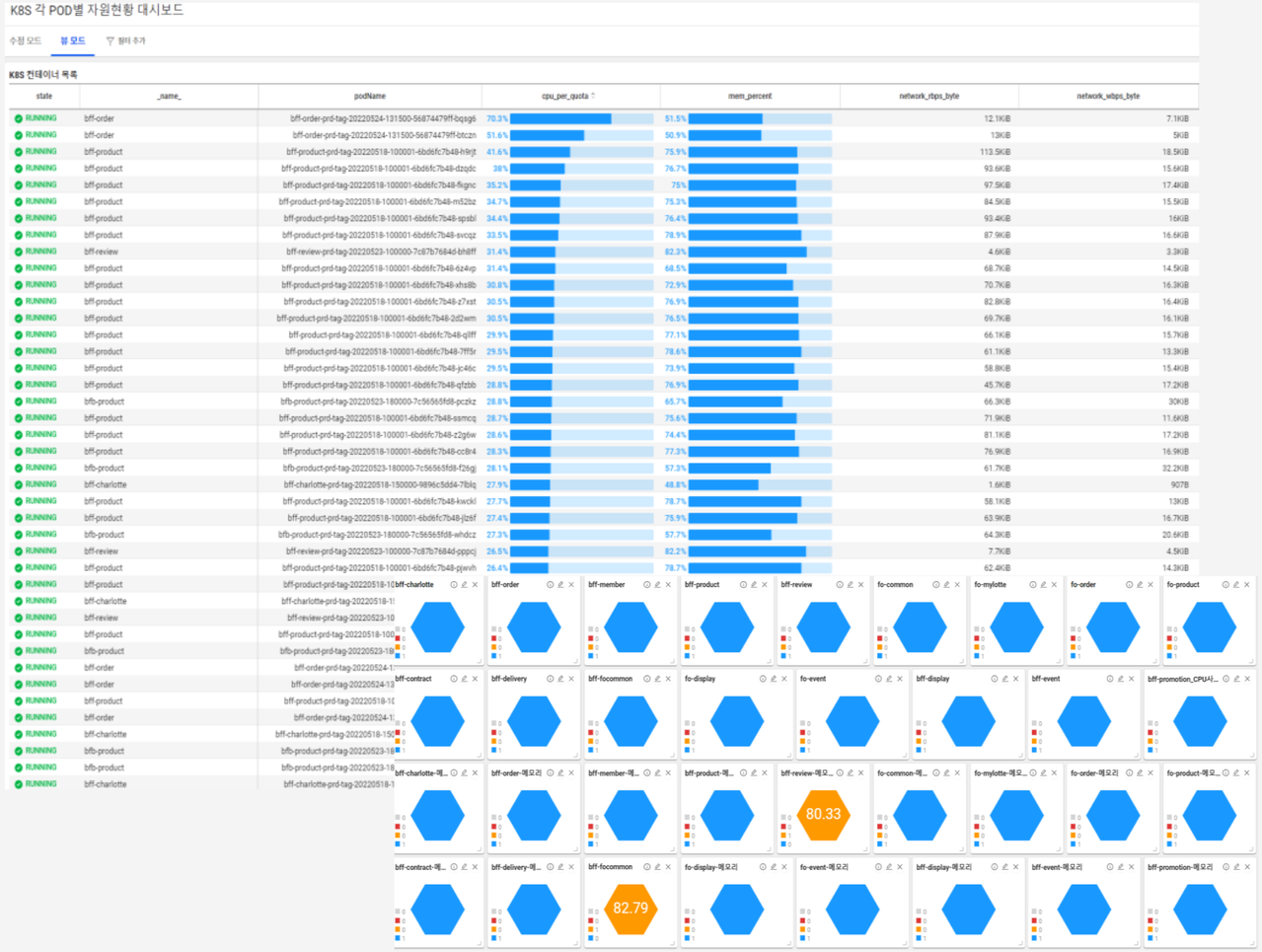
롯데ON MSA 모니터링 최적화 사례

HPSC LAB
High Performance & Secure Computing

MSA 모니터링 과제

- 한 트랜잭션에도 여러 개의 서비스를 거침
- 복잡한 아키텍처 구조로 잠재된 위험 요소 산재
- 서비스는 계속 쪼개짐 → 관리해야 할 서비스 증가

Events + Logs + Traces = Observability



롯데ON SRE (사이트 신뢰성 엔지니어링) 원칙

`확인이 필요한 알림만 발생시킨다`

금융권 퍼블릭 클라우드, DevOps 구축 여정

HPSC LAB
High Performance & Secure Computing

왜 클라우드인가

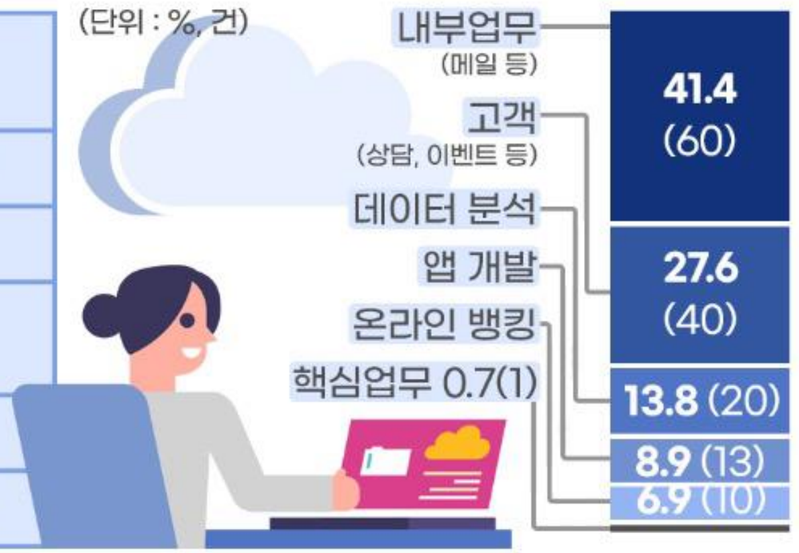
금융회사들의 클라우드 도입 계획

KB 국민은행 신한은행	2024년까지 그룹 공동 클라우드망 조직
우리카드	내년 2월 그룹 공동 클라우드 시스템 구축
NH농협은행	네이버와 손잡고 외부 클라우드 도입
하나은행	인천 청라데이터센터 개설 자체 개발 내부 클라우드망 사용
KB 국민카드	KB페이에 외부 클라우드(AWS) 도입
LOTTE CARD	내부 고객 금융이력 관리에 클라우드 도입

* 신한 삼성카드 퍼블릭 클라우드 검토 중

자료 : 각 사

클라우드 이용분야는 주로 내부업무



자료 : 금융감독원

금융권 퍼블릭 클라우드, DevOps 구축 여정

왜 클라우드인가

글로벌 금융 기업의 클라우드 여정 사례

	Phase 1	Phase 2	Phase 3
BNP PARIBAS	• 격리된 Dedicated Public Cloud 인프라 환경 구축	• 공유 서비스 센터(SSC)를 통해 그룹사 내 다양한 IT 서비스 제공	• 금융 전용 Cloud 솔루션 및 인프라를 타행까지 확대 예정
Santander	• Cloud Competence Center설립 : Co-Management Cloud 거버넌스 체계 수립	• 모든 비즈니스 영역에 Cloud 서비스를 제공하기 위한 Marketplace 제공	• 디지털 서비스 및 Ecosystem 구축을 위한 Cloud Platform 'Serenity' 구축
브라질 대형 은행	• 코어 뱅킹에 API Integration 실행 • 컨테이너 Platform 기반 MSA/모듈화 적용	• Modernization을 위한 Hybrid Cloud 환경 구축 • 컨테이너 Platform/ Public Cloud 환경 및 코어	• 마켓/비즈니스 프로세스/ 기술 Platform 구축을 위한 파트너십 확보

출처 : 2021 IBM Industry Insight Finance

나스닥, AWS로 증권거래 시스템 이전.."100% 클라우드 전환 목표"

발행일 : 2021-12-01 13:58

금융권 퍼블릭 클라우드, DevOps 구축 여정

왜 클라우드인가

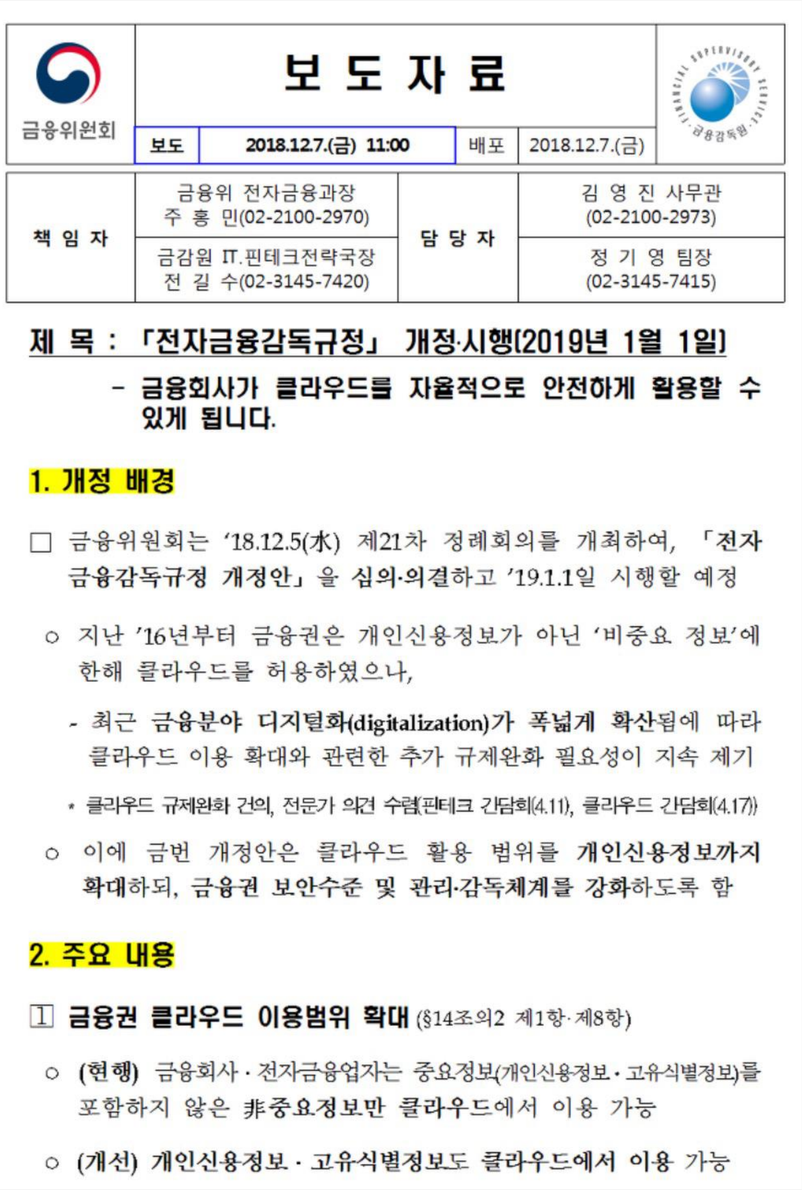
- 2019년 이전엔 전자금융감독법에 따라 퍼블릭클라우드 사용 제한

  → 19년 법 개정을 통해 사용 가능해짐

- 맞물려 인프라 노후화와 대외요건 등으로 클라우드 도입 가속화

- 인프라 노후화
  - U2L (Unix to Linux)
  - X86, 가상화 기술 발전
  - 벤더 종속성 탈피
  - 라이선스 비용 과다
  - SW EOS

- 대외 요건
  - 빅테크 기업 경쟁
  - 예측 어려운 사업 요건
    (오픈API, 마이데이터, 재난지원금)
  - 기술 트렌드 변화
  - 비용 증가



금융위원회

보 도 자 료

보도	2018.12.7.(금) 11:00	배포	2018.12.7.(금)

책 임 자	금융위 전자금융과장 주 홍 민(02-2100-2970)	담 당 자	김 영 진 사무관 (02-2100-2973)
	금감원 IT.핀테크전략국장 전 길 수(02-3145-7420)		정 기 영 팀장 (02-3145-7415)

제 목 : 「전자금융감독규정」 개정·시행(2019년 1월 1일)

- 금융회사가 클라우드를 자율적으로 안전하게 활용할 수 있게 됩니다.

1. 개정 배경

□ 금융위원회는 '18.12.5(水) 제21차 정례회의를 개최하여, 「**전자금융감독규정 개정안**」을 **심의·의결**하고 '19.1.1일 시행할 예정

○ 지난 '16년부터 금융권은 개인신용정보가 아닌 '비중요 정보'에 한해 클라우드를 허용하였으나,

- 최근 **금융분야 디지털화(digitalization)가 폭넓게 확산**됨에 따라 클라우드 이용 확대와 관련한 추가 규제완화 필요성이 지속 제기

* 클라우드 규제완화 건의, 전문가 의견 수렴(핀테크 간담회(4.11), 클라우드 간담회(4.17))

○ 이에 금번 개정안은 클라우드 활용 범위를 **개인신용정보까지 확대**하되, **금융권 보안수준 및 관리·감독체계를 강화**하도록 함

2. 주요 내용

1 금융권 클라우드 이용범위 확대 (§14조의2 제1항·제8항)

○ **(현행)** 금융회사·전자금융업자는 중요정보(개인신용정보·고유식별정보)를 포함하지 않은 **非중요정보만 클라우드**에서 이용 가능

○ **(개선) 개인신용정보·고유식별정보도 클라우드에서 이용** 가능

금융권 퍼블릭 클라우드, DevOps 구축 여정

HPSC LAB
High Performance & Secure Computing

왜 클라우드인가

- 그룹 클라우드 표준을 토대로 Legacy 분석
- 사업 요건을 반영하여 자체 표준 수립

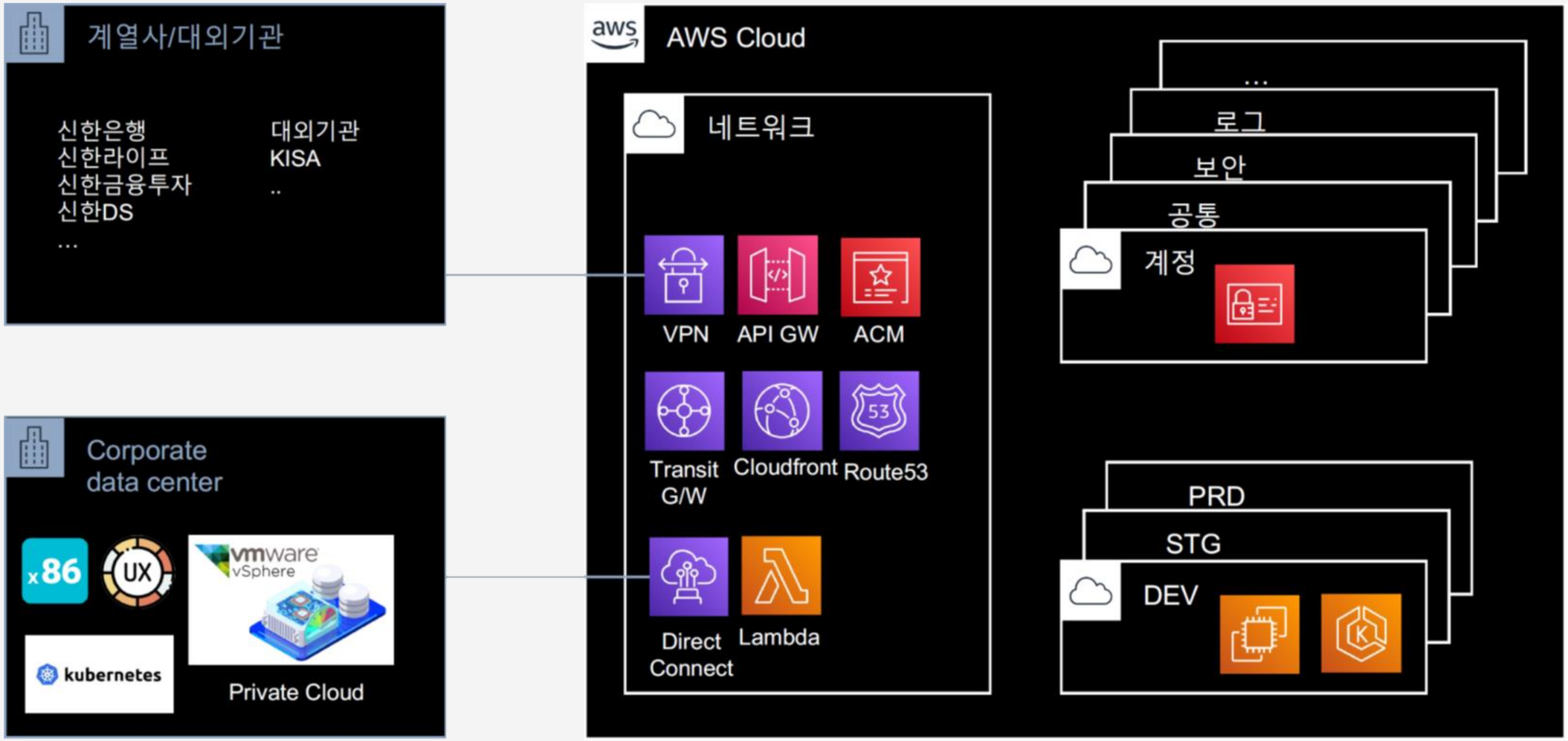
금융권 퍼블릭 클라우드, DevOps 구축 여정

HPSC LAB
High Performance & Secure Computing

왜 클라우드인가

금융권 퍼블릭 클라우드, DevOps 구축 여정

왜 클라우드인가



금융권 퍼블릭 클라우드, DevOps 구축 여정

팀내 정책 상, 상용S/W 사용을 지양하고 오픈소스 기반으로 구축

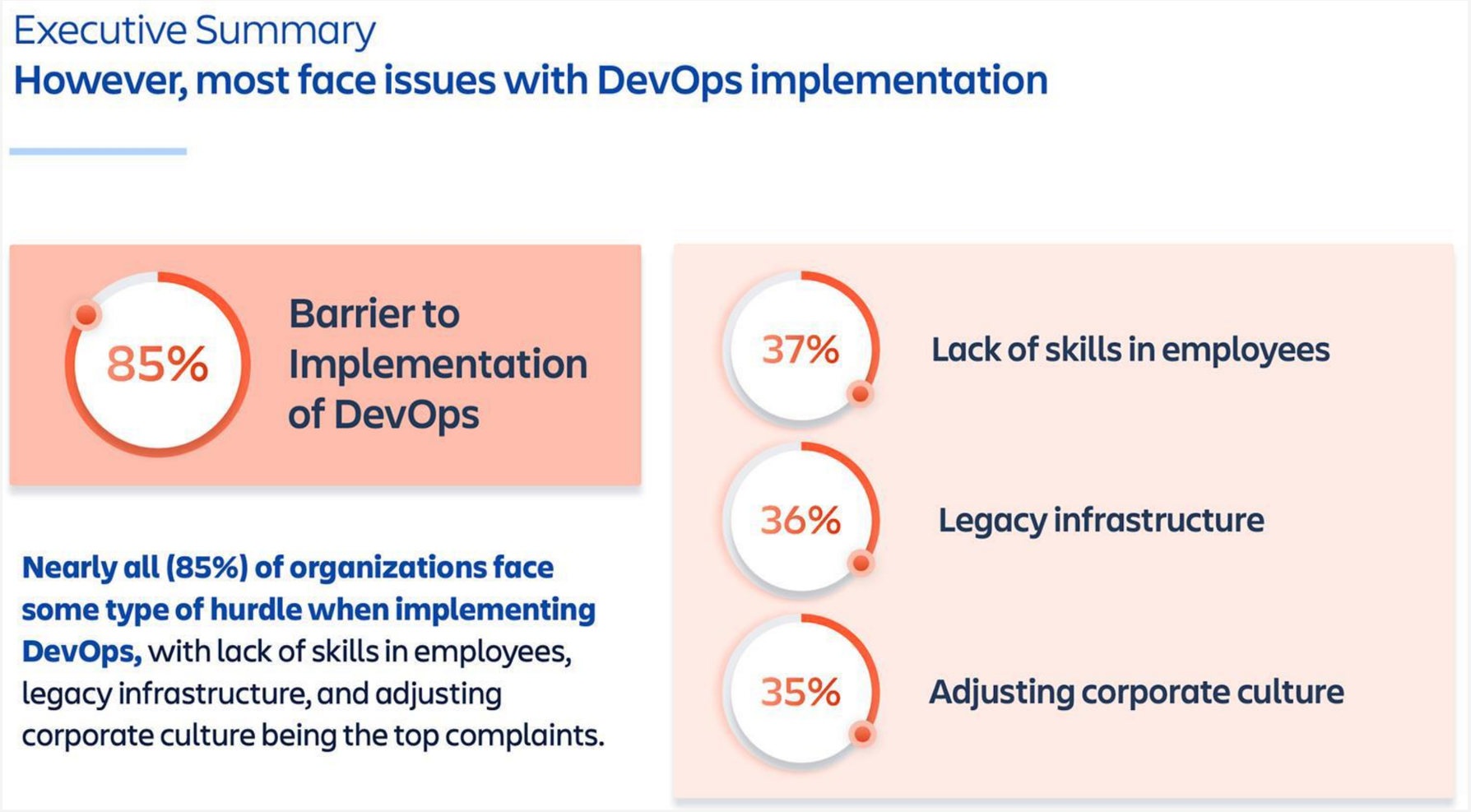
DevOps는 점진적 변화

DevOps 안티 패턴

- 이번에 대대적으로 CI/CD를 재구축했으니 DevOps?
- 제일 잘나가는 비싼 Observability 도구를 도입했으니 DevOps?
- 개발팀과 운영팀을 합쳐 DevOps 팀을 만들었으니 DevOps?

→ 특정 도구, 서비스를 도입했다고 DevOps라 할 수 없다.

85%는 DevOps에 장벽을 느낀다고 한다.



출처: Atlassian DevOps Trends Survey 2020

구성 요소를 상시 업데이트

Framework /Library

- SpringBoot
  - 1.4 → 2.7 (6년치, 매우 어려움)
  - 1.4 → 1.5 → 2.0 → ... → 2.7 (해 볼만 함)

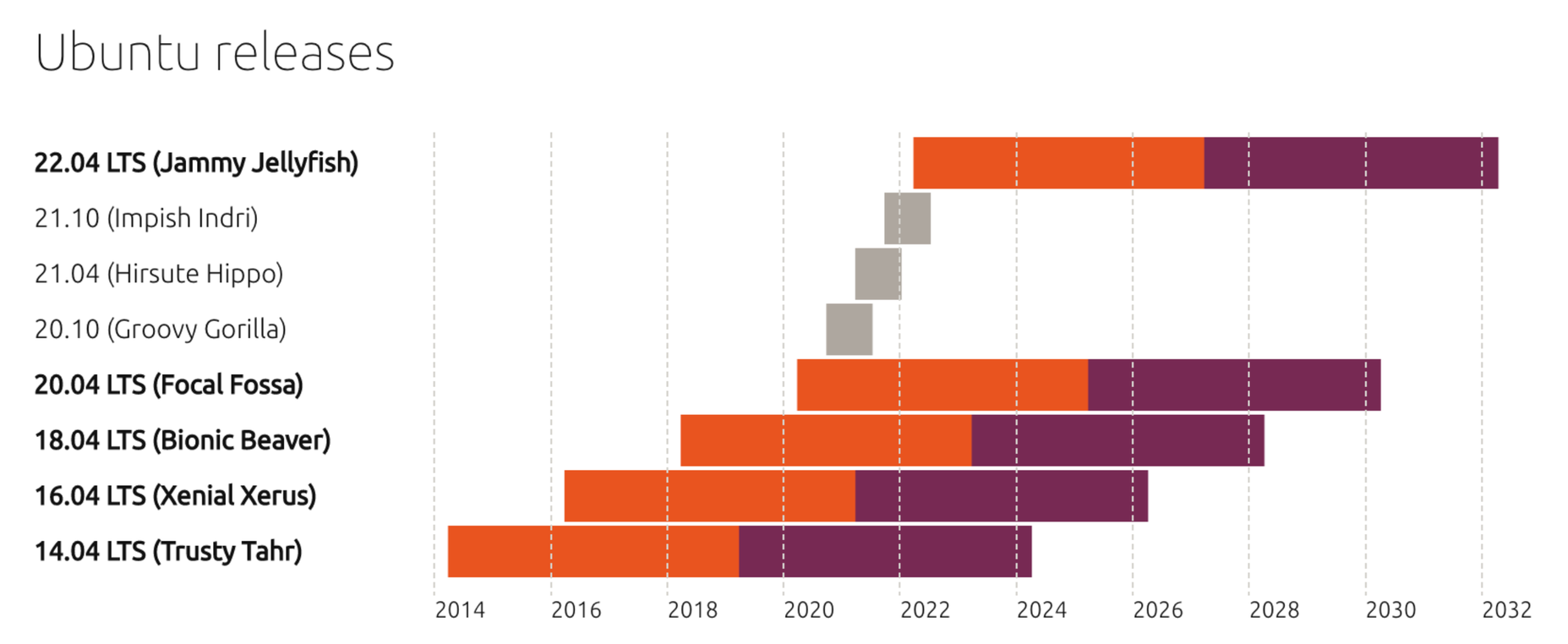
JDK

- JVM
  - Java8 → Java17 (7년)

Linux Kernel

- Ubuntu
  - 16.04 → 22.04 (Kernel 4.10 → 5.15)
  - 16.04 → 18.04 → 20.04 → 22.04

OS의 유통기한은 5년


Ubuntu releases
22.04 LTS (Jammy Jellyfish)
21.10 (Impish Indri)
21.04 (Hirsute Hippo)
20.10 (Groovy Gorilla)
20.04 LTS (Focal Fossa)
18.04 LTS (Bionic Beaver)
16.04 LTS (Xenial Xerus)
14.04 LTS (Trusty Tahr)
2014
2016
2018
2020
2022
2024
2026
2028
2030
2032

안티 패턴

- 이슈가 없다면 손대지 않는다.
- 이슈가 생기면 최소한의 수정을 시도한다.
- 이렇게 누적되어 수 년 간의 버전 차이를 업그레이드 하는 것은 대 공사일 확률이 크다.

와탭랩스 절차

- 상시 업데이트를 수행한다.
- 이슈 요소가 업데이트 과정에서 식별된다.
- 업데이트 전/후 비교로 unknown unknowns를 known unknowns로 바꾼다.

구성 요소를 상시 업데이트: 사례 1

점진적인 변화는 이해하며 할 수 있다

- Unknown unknowns
- Known unknowns
- Known known

구성 요소를 상시 업데이트: 사례 2

```
top - 08:08:18 up 16 days,  3:01,  4 users,  load average: 0.16, 0.70, 0.77
Tasks: 171 total,   2 running, 168 sleeping,   0 stopped,   1 zombie
%Cpu(s):  1.4 us,  0.8 sy,  0.0 ni, 97.5 id,  0.1 wa,  0.0 hi,  0.3 si,  0.0 st
KiB Mem : 16432232 total,  3022812 free,  8383916 used,  5025504 buff/cache
KiB Swap: 16776700 total, 16776700 free,        0 used.  7515432 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4406 whatap_+  20   0 5791128 838980  24696 S   3.7  5.1   1:45.76 java
  381 root      20   0   52080  14020  11556 S   2.0  0.1  10:42.79 systemd-journal
 3653 whatap_+  20   0 5779728 469368  20016 S   2.0  2.9   1:08.81 java
 4525 root      20   0  477740  18300   7484 S   1.0  0.1   0:06.99 whatap_infrad
 3465 whatap_+  20   0 2475116 108772  15764 S   0.3  0.7   0:08.29 java
 4764 whatap_+  20   0 9554920 3.721g  26576 S   0.3 23.7   2:44.37 java
 5932 whatap_+  20   0 7638668 2.066g  22632 S   0.3 13.2   2:38.93 java
```

```
Native Memory Tracking:

Total: reserved=2074311KB, committed=819171KB
-                 Java Heap (reserved=524288KB, committed=524288KB)
                            (mmap: reserved=524288KB, committed=524288KB)
```

SpringBoot 1.4 : 2GB

SpringBoot 2.2 : 800MB

이해하면 새로운 시도

Image	2022년 6월 17일, 10:49:36 (UTC+09)	4.20	URI 복사
Image	2022년 5월 31일, 17:23:41 (UTC+09)	593.99	URI 복사
Image	2022년 5월 31일, 16:05:25 (UTC+09)	593.99	URI 복사
Image	2022년 5월 04일, 16:32:41 (UTC+09)	593.99	URI 복사
Image	2022년 2월 07일, 15:15:31 (UTC+09)	593.25	URI 복사
Image	2021년 12월 17일, 09:49:23 (UTC+09)	593.11	URI 복사

Go Lang : 4.2MB

Java : 593MB

Java → Go Lang 전환 후 Container Image 크기 차이 : 140배

복잡도 줄여 나가기 → The Simple is the best

- 리소스가 자산이라면 많을 수록 좋겠지만 리소스는 부채

  → 빠르고 과감한 폐기

사용량 없는, 용도가 확인되지 않은 것은 삭제

= 복잡도를 줄일 수 있는 최고의 전략

지속적 대체제 전환

- 신식 대체제를 사용하고 사용하기로 결정했다면 퇴행 방지를 노력 → 대체 과정에서 기술의 질 향상

대체제로 전환

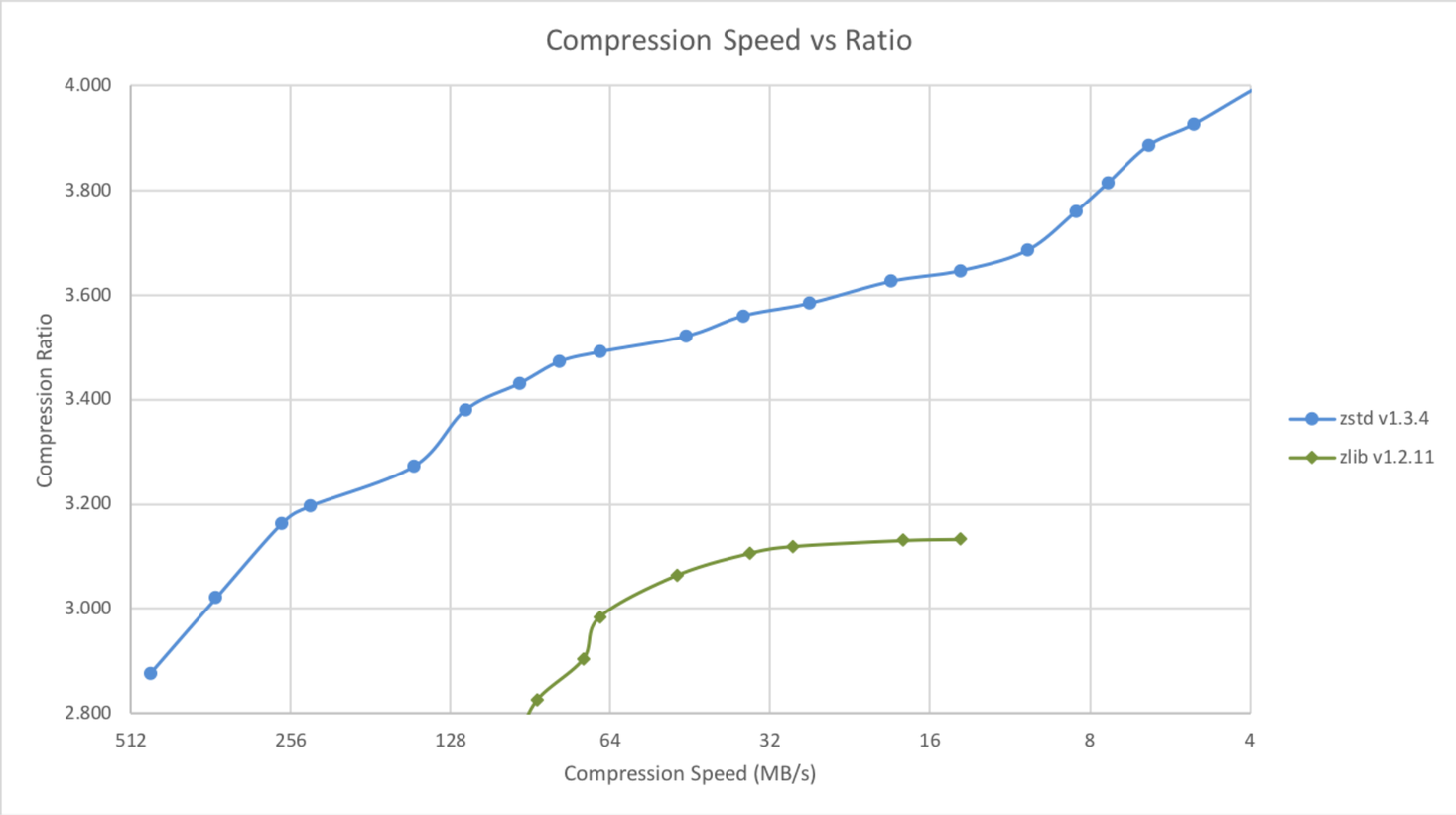
- 대체제가 있다면 신식으로 이주

퇴행 방지

- 폐기를 결정했다면 신규 기능이 폐기 결정 리소스를 사용하지 않도록 관리
- 새로운 기술을 쓰기로 했다면 과거의 기술은 더 이상 쓰지 않는다
    - Ex) 신규 서비스 모듈은 Container와 Serverless 만 사용한다.

신식 대체제 사용: 사례

- 대체 : gzip → zstd **5x faster**



최신 버전 :zstb 옵션만 추가

https://facebook.github.io/zstd/

목차

성장하는 엔지니어 학습 문화

걸어가는 속력과 방향을 인지하자

속도 = 속력 * 방향 $v_{\mathrm{av}} = \frac{\Delta \mathbf{x}}{\Delta t}$

자신의 성장 속도를 인지하는 단계 ➔ 성장 속도를 차츰 높이기

- 학습해야 하는 지식 분량
- 해결해야 하는 문제 방향
- 생산해야 하는 코드 분량
- 공유해야 하는 설계 방향
- 반복해야 하는 학습 분량
- 반복해야 하는 개발 흐름

성장하는 엔지니어 학습 문화

익숙한 것보다, 낯선 방식으로 해결하라

- 주어진 문제에 대해 자신만의 해결 방법 고민하기
- 필요한 지식을 찾아서 자기만의 방식으로 구성하기
- 동료들과 피드백을 교환하면서 점진적으로 개선하기

학습 = 지식 * 연습 * 습관 * 인지 * 사회적 비용

성장하는 엔지니어 학습 문화

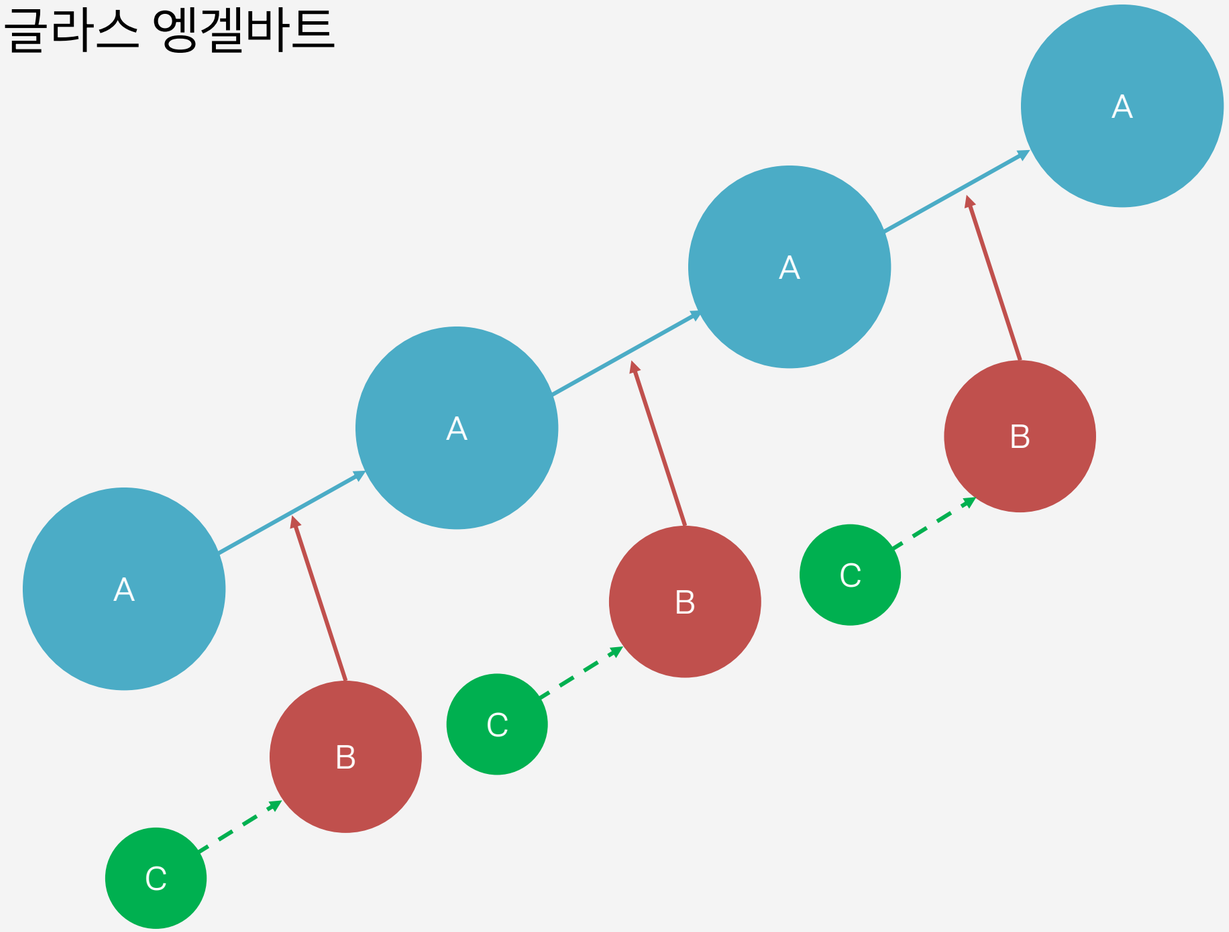
자존심을 버리고, 자존감을 키우자

- 천재를 이기는 방법은 없다
- 성장할 때는 자신과 비교
- 어제의 나와, 일주일 전의 나와 비교
- 큰 덩어리로 정의를 정의할 때는 애매모호해짐
- 계획을 작게 하는 것을 추천

A-B-C 작업

제록스 연구소에서 “일 잘하는 사람들 " 분석 – 더글라스 엥겔바트

- **A 작업 : 반복해서 해야 하는 일**
- **B 작업 : A 작업을 개선하는 것**
- **C 작업 : B 작업을 개선하는 것**



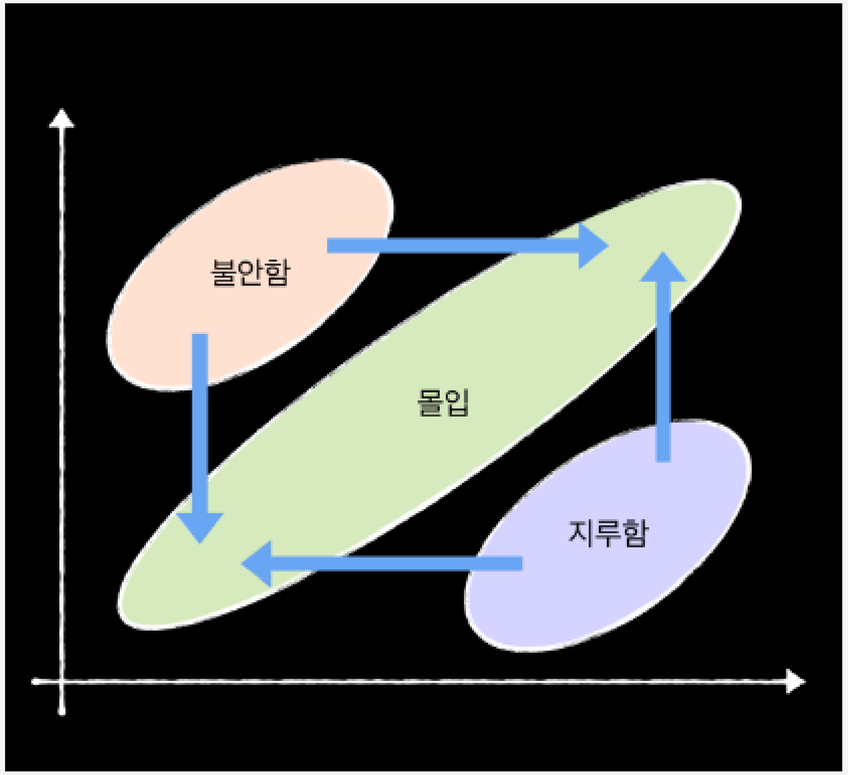
성장하는 엔지니어 학습 문화

몰입과 난이도 조정하기: 즐거워야 몰입할 수 있고, 몰입해야 즐겁다.

지루함 극복하기

방법1: 실력 낮추기 (다른 도구, 낯선 환경)

방법2: 난이도 높이기 (제약사항 추가하기)



불안감 극복하기

방법1: 실력 높이기(전문가, 도구 등)

방법2: 난이도 낮추기 (작문 문제로 분리)

Thank You