# 데브옵스 구축

데브옵스(DevOps)구축에 틀이 정해진 것은 아니다.
좋은 건 알겠는데 개발팀과 운영팀의 업무를 어떻게 나누고, 그 업무를 잘 나누도록 결정하는 사람은 누가될 것인가?
데브옵스의 핵심이자 가장 어려운 부분이다.

---

### 데브옵스는 팀을 만들자는 것이 아니다.

데브옵스 팀은 1인 2역(개발+운영)을 담당하는 사람들의 모임이 아니다. 운영팀과 개발팀은 같은 제품에 관여하지만, 입장과 역할이 다르다.
데브옵스는 이들의 일을 합해서 한 부서에 몰아주는 것이 아니라, 각자의 역할에 집중하도록 만드는 일에 가깝다.

### 소통과 협업을 통해 각자의 업무에 집중할 수 있게 하는 것

조직 내 데브옵스를 구축하는 것은 개발자와 운영자 중간에 존재하는 모호한 업무 영역을 어떻게 관리할지 정하는 일이다.
소통과 협업, 투명하고 솔직한 문화를 통해, 개발하는 사람은 개발에만 집중할 수 있고 운영하는 사람은 운영에만 집중할 수 있는 환경을 만들어야 한다.

---

### 문화를 바꿔보자

데브옵스 문화는 개발하는 서비스, 회사 규모, 조직 구성원에 따라 다를 수 있다.
따라서 완벽한 데브옵스의 기준도 명확히 정의할 수 없으며, 서비스 개발에 관여한 모두가 데브옵스 문화에 기여해야 한다. 언제나 그렇듯, 모두의 생각이 한 번에 바뀔 것이라고 기대하면 안 된다.
여러 번 시행착오도 겪어야 하고, 다시 원점으로 돌아가기도 하면서 마침내 데브옵스가 우리 회사에도 정착하는 것이다.

이 문화를 만들기 위한 방법론으로

1. 애자일 방식을 차용
2. 지속적 배포를 위한 파이프라인을 구축
3. 자동화 도구를 조직에 도입

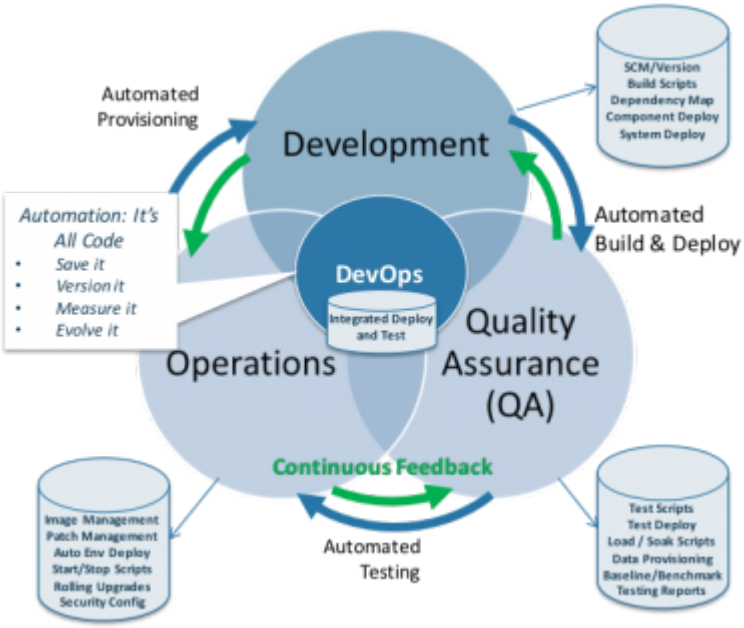
개발과 운영 과정을 모두가 공유함으로써 개발자는 자신의 코드가 어떻게 테스트 되고, 검증되고, 운영되고 있는지 투명하게 알 수 있으며, 운영자는 개발 일정과 과정을 파악할 수 있다.

---

### 문화를 만든다고 데브옵스 인가?

당연히 No
문화와 사고방식의 변화는 데브옵스를 적용하기 위한 가장 기본적인 조건이다.
모두 한 마음 한 뜻으로 데브옵스를 적용하기로 했다면, 소통과 협의를 통해 어떤 부분을 자동화할 것인지, 어떤 도구를 사용하면 효율적인지 끝없는 협의와 조율의 시간을 갖자.

# 올바른 데브옵스 구축을 위한 7가지 고려 사항 (Feat. 툴 선택을 잘 하자)



그림_데브옵스 모범 모델

1단계: 개발, QA 및 인프라 자동화 팀의 협업 및 공유 툴 전략을 이해하기.

- 개발팀과 운영팀 모두를 위한 DevOps 공통 전략 수립
  1. 프로세스 전략
  2. 커뮤니케이션 및 협업 계획
  3. 지속적인 개발 도구
  4. 지속적인 통합 툴
  5. 지속적인 테스트 툴
  6. 지속적인 구축 틀
  7. 지속적인 운영 및 CloudOps 툴

2단계: 툴을 사용하는 이유를 알자. 모든 요청은 기록되어야 한다.

- 데브옵스 프로세스 외부에서 임시 작업이나 변경이 발생해서는 안 된다
- 데브옵스 프로세스 내부에서는 새로운 소프트웨어나 변경된 모든 요청은 기록(공유) 되어야 한다

3단계: 툴을 사용함에 따라 처리할 수 있는 자동화 및 데브옵스 요청.

- 유연한 업무 계획
- 더 빠른 산출
- 명확한 목표와 투명성 제공

4단계: 툴을 사용하여 수동 및 자동 프로세스 모두 로그를 기록.

- 자동화 및 수동 DevOps 프로세스의 생산성을 이해
- 구축 혹은 발견된 테스트 오류와 같은 DevOps 프로세스와 관련된 매트릭스 정리
- 자동화된 프로세스를 정의, 사람의 개입 없이 문제를 해결할 수 있도록 한다

5단계: 테스트 자동화 및 데이터 프로비저닝 도구 선택.

- 테스트 자동화는 단순히 '자동화' 만을 의미 하지 않음
- 코드의 품질, 데이터 및 전체 솔루션을 보장할 수 있어야 함

6단계: 각 배포에 대한 테스트.

- 테스트에 대한 기준을 정의하고 결과물이 기준에 충족하는지 확인한다
- 새로운 변경 사항을 적용하며 변경된 부분에 대해 세분화하여 테스트 한다

7단계: 팀 간의 지속적인 피드백을 통해 정보 격차 최소화.

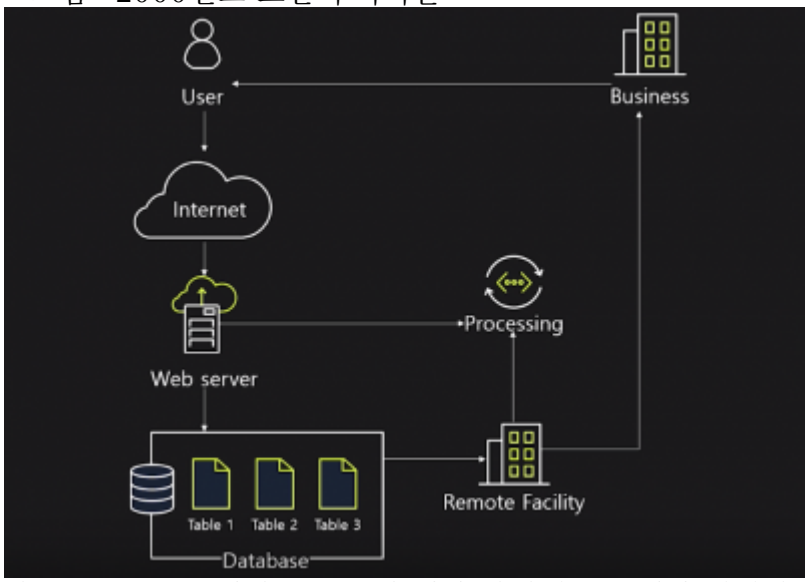
- 적절한 피드백 루프 정하기
- 예시)
  1. 수동 또는 자동화된 매커니즘을 사용하여 문제를 식별하고 있는가?
  2. 개발자나 운영자가 무엇이, 왜, 어디에서 발생했는지 이해할 수 있도록 태그를 적절히 사용하는가?
  3. 정보를 격차가 없도록 도움을 주고 있는가?
  4. 자동화된 시스템 운영을 하며 발생된 에러 사항을 보고하기 위한 툴은 제대로 된 역할을 하고 있는가?
  5. 팀원 전원이 공동으로 문제를 해결하기 위한 접근법, 적용해야 할 해결방식에 대한 합의, 필요한 추가 코드 또는 기술의 목록이 포함되어야 한다

**목표는 모든 것을 자동화 하며 이를 통해 DevOps를 더 잘 수행하는 방법을 지속적으로 실험해보고 DevOps 운영을 해야 하는 것이다!**

# DevOps 사례

## Amazon

<그림> 2000년도 초반의 아마존

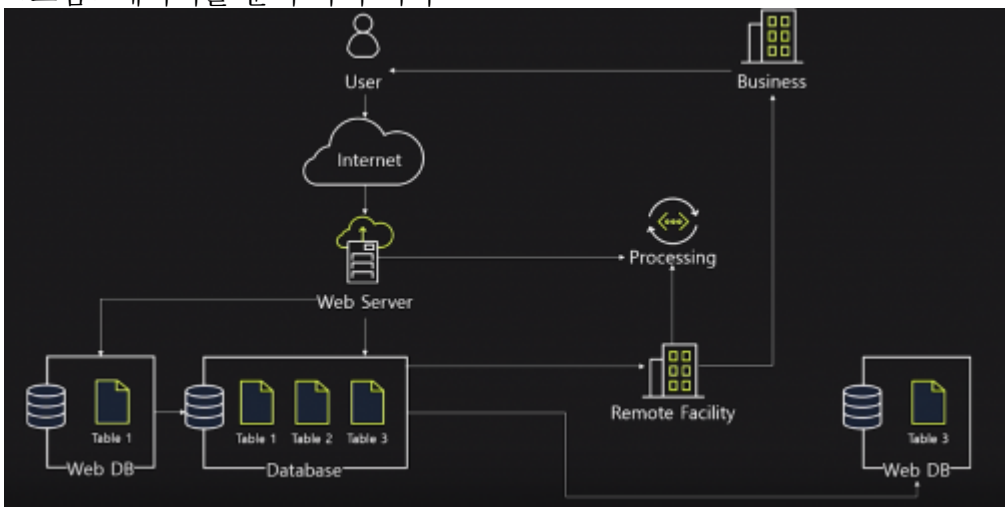



작은 application이나 소규모의 개발 환경에는 문제가 없지만, 빠르게 성장하고 있는 Amazon에게는 문제가 되었다.

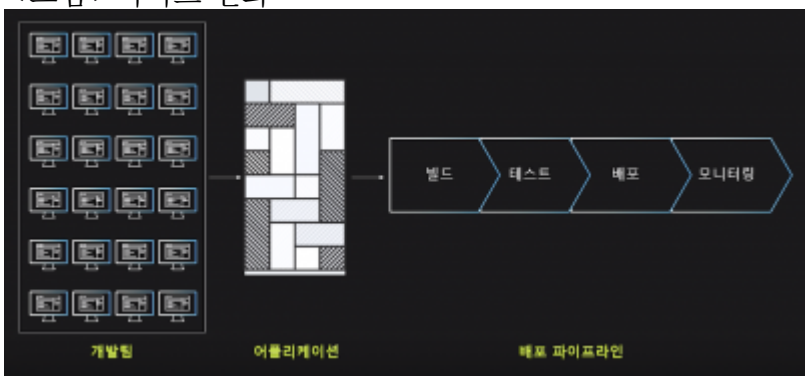
- 확장성 부재
- 컴포넌트 문제로 전체 시스템 장애
- 느린 배포 속도

- 다양성 부족

<그림>데이터를 분리 하기 시작



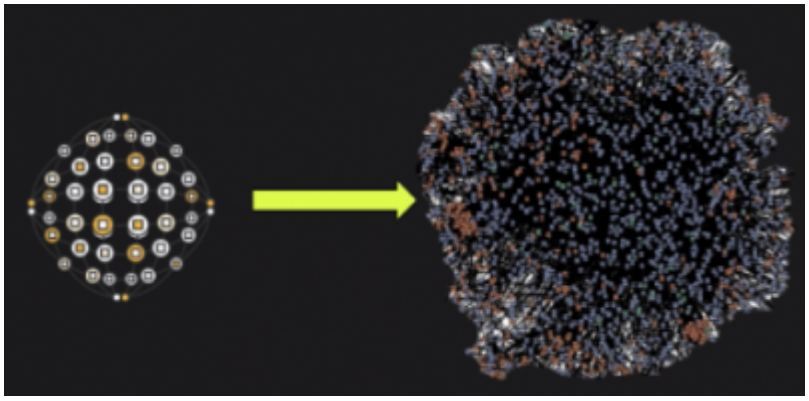

<그림>서비스 진화




- 개발에서 배포까지의 일련의 과정이 수일~수주까지 걸리는 어려움이 있었음.



- 서비스 지향 아키텍처의 도입으로 서비스를 분리
- 애자일 개발 방식 도입
- 개발 속도 향상
- 작은 장애 범위 구성



- application을 단위별로 쪼개고, 분해하기 시작
  - 고객 지향적
  - 서비스에 대한 책임 부여
  - 빠른 혁신
  - 빠른 움직임



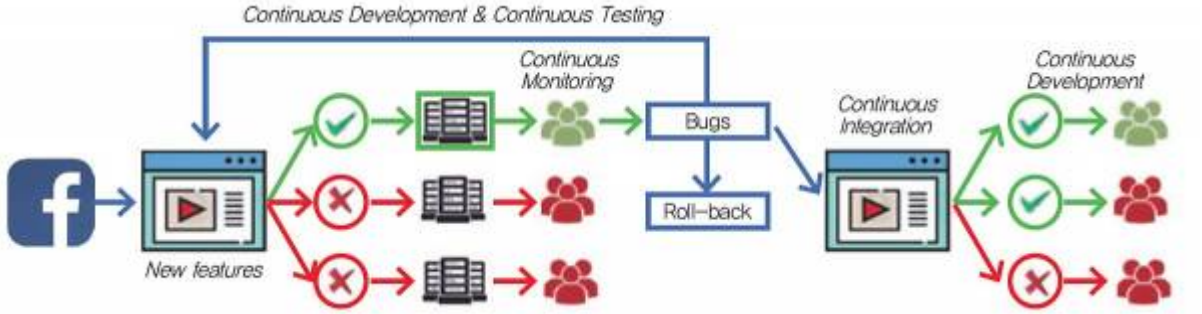

- 각 서비스를 담당하는 팀에 그들만의 tool을 제공
  - self-service 환경 제공
  - 각 팀에게 자체 배포 process를 소유, 관리하게 함
- 서비스간의 통신은 모두 API로 표준화
  - backend 간의 dependency를 없애게 됨
- 서비스를 개발한 팀이 해당 서비스를 가장 잘 안다는 기준 아래서 bug가 발생하면 바로 수정을 하는 형식
  - 운영 권한과 모니터링 권한을 줌
  - 서비스 빌드 / 테스트를 하도록 함

## Facebook

페이스북은 사용자 증가와 새로운 서비스 오픈에 따라 엄청난 트래픽이 발생했으며, 이로 인해 시스템이 중단되었다.
시스템에 영향이 가지 않으면서도 새롭게 출시하는 기능을 테스트할 수 있는 방법으로 다크 론칭(Dark Launching) 기법을 고안했다.
다크 론칭 기법은 특정 사용자에 한정해 새로운 기능을 배포하고 테스트한 뒤 피드백을 반영하여 새 기능이 안정화되면 전체 배포하는 방식이다.
또한 CD(Continuous Delivery) 도입을 통해 새로운 서비스 배포가 기존 애플리케이션 성능에 영향을 미

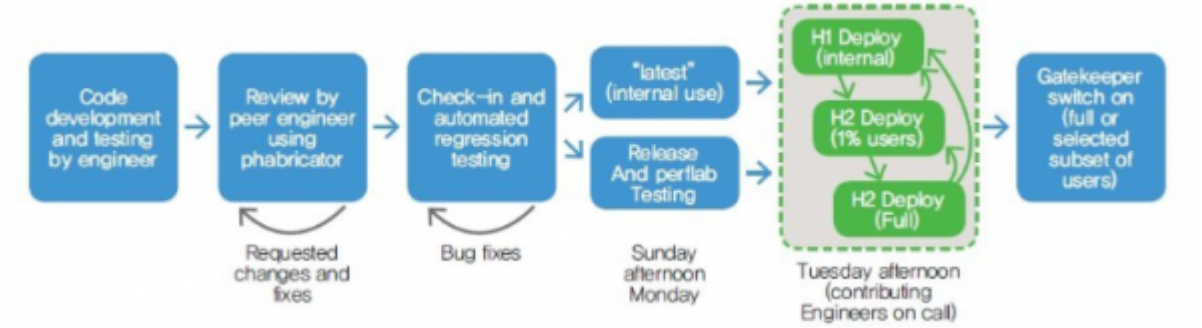
치지 않고, 더욱 빠르고 편리하게 서비스를 릴리즈한다.

다크 론칭 기법에서는 전용 배포 파이프라인을 통해 새로운 기능을 일부 사용자에게 배포한다.
아래 그림처럼 선택한 사용자 그룹에 해당하는 파이프라인만 동작하고 그 외 파이프라인은 모두 꺼진다.




새 기능이 배포된 특정 사용자 그룹을 지속적으로 모니터링하고 피드백을 반영하여 동일한 사용자 기준으로 안정성이 확보되면 꺼져있는 다른 배포 파이프라인을 켜서 나머지 사용자를 대상으로도 점진적으로 배포한다.

만약 테스트에서 문제를 발견하면 롤백한다. 이는 페이스북이 데브옵스를 기반으로 CI/CD 도입함으로써 가능하게 되었다.




페이스북에는 별도 QA팀이 없다.
개발자는 새롭게 개발한 모든 코드를 직접 테스트하며, 코드는 커밋과 푸시 작업의 일부로 자동화된 테스트를 모두 통과해야 한다.
개발자는 DevOps를 위해 구성된 조직을 통해 개발 외에 소프트웨어 운영 및 사용에도 동참하며, 이는 개발 시점부터 좋은 코드를 쓰고 철저하게 테스트를 진행하는 동기가 된다.
이러한 책임 문화는 심각한 버그 발생을 막는 데 도움이 됐다.

# Ref

https://cloudmt.co.kr/?p=3436
https://naon.me/posts/til137
올바른 데브옵스 구축을 위한 7가지
Amazon's DevOps
Facebook's DevOps

밤즌, 데브옵스, DevOps, 데브옵스구축, 데브옵스사례

From:
http://125.132.25.164/dokuwiki/ - 문제를 잘 정의하는 것은 문제를 절반 해결한 것이다. - 2023.12

Permanent link:
http://125.132.25.164/dokuwiki/doku.php?id=wiki:pm:devops:%EB%8D%B0%EB%B8%8C%EC%98%B5%EC%8A%A4_devops_%EA%B5%AC%EC%B6%95%EC%97%90_%EA%B4%80%ED%95%B4&rev=1649229585

Last update: 2022/04/06 16:19