마이크로서비스 아키텍처 이해하기

- 운영의 입장에서 -

클라우드 서비스팀 남세현

여러분이 만나는 시스템은 어떤 시스템입니까?

- 운영/관리가 편한 시스템
- 스케일링이 편한 시스템
- 발 뻗고 편안히 잠을 잘 수 있는 시스템
- 새로운 실험, 테스트를 편히 할 수 있는 시스템

혹은

- 운영/관리가 힘든 시스템
- 자동화가 안되어있는 시스템
- 스케일링이 힘든 시스템
- 빌드/테스트/릴리즈 사이클이 느린 시스템

그런 여러분의 시스템은

퍼블릭 클라우드에 적합한 시스템입니까?

퍼블릭 클라우드 세상에 적응할 수 있습니까?

왜 퍼블릭 클라우드로 가야하나요?

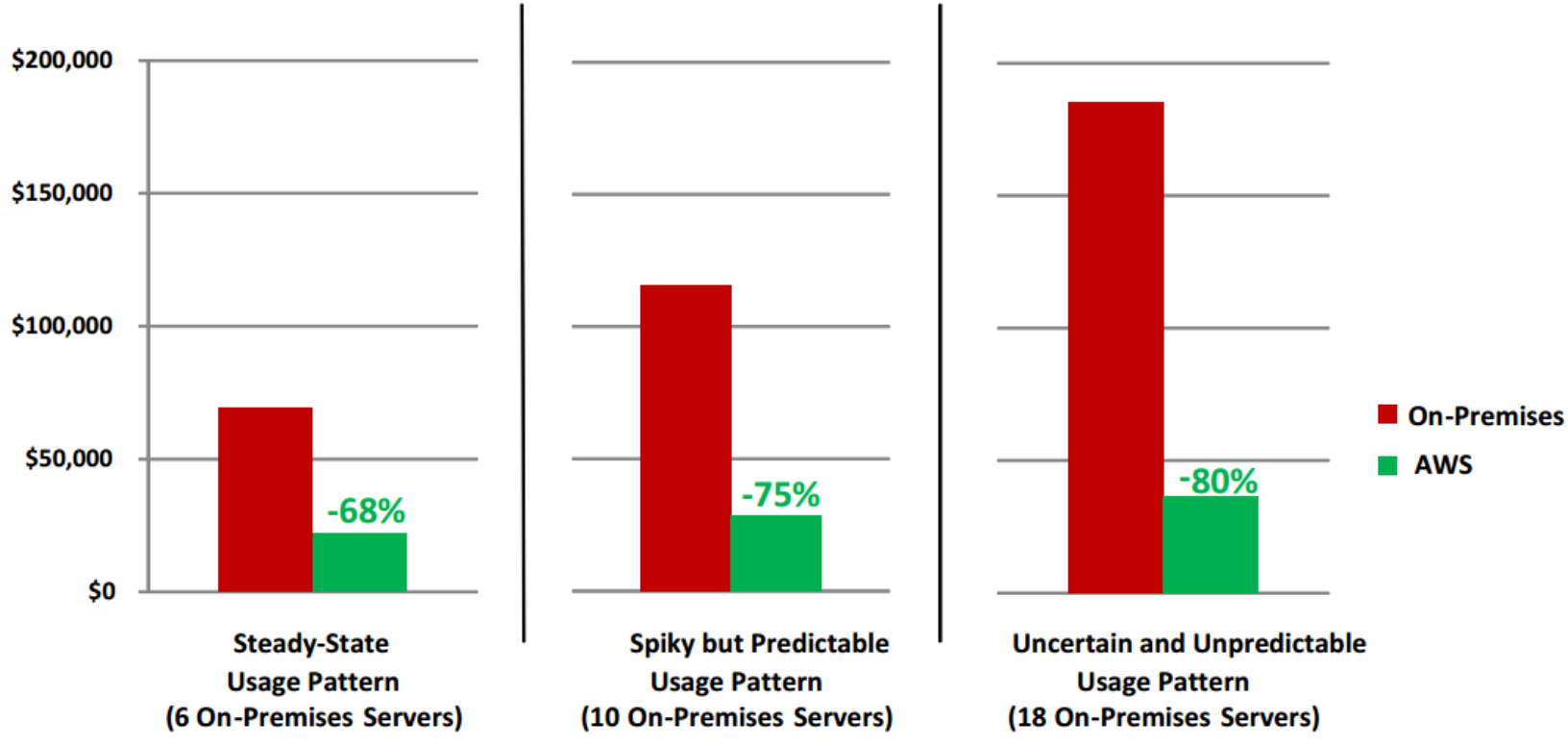
Q. 퍼블릭 클라우드를 사용하는 이유는?

퍼블릭 클라우드를 사용하는 이유

- 잘 만들어진 Managed Services
  - "바퀴를 다시 발명하지 말라"
  - Computing, Storage, 자동화, Log, etc...
- 인프라의 관리, 보안, 공급(provision) 제공
  - 장비 공급에 한달 기다릴 필요가 없음
  - 은행권도 사용할 정도의 보안레벨
  - 장비 망가져서 교체하러 출장 갈 필요 X
- 업계 최고 레벨의 기술력 제공

퍼블릭 클라우드를 사용하는 이유

- 높은 퀄리티를 적은 비용으로
  - 99.95 Availability, 99.999999999 Durability (11 9's)
    - SLA(Service Level Agreement)
  - 규모의 경제, 유연한 인프라 증감(Pay as you go)



TCO of Web Applications (Compute and Database) for 3 Years

Q. 그렇다면 우리의 시스템도 AWS의 장점을 누릴 수 있는가?

- 운영/관리 힘든 시스템 + 운영/관리 편한 인프라 (?!)
- 스케일링 힘든 시스템 + 스케일링 편한 인프라 (?!)
- 탄력성이 적은 시스템 + 탄력적인 비용의 인프라 (?!)

오늘은 우리의 시스템이 어떻게 변해야 하는지에 대해 이야기 해볼까 합니다.

우리의 시스템 - Monolithic 이란?

"단단히 짜여 하나로 되어 있는"

하나의 프로세스에 모든 기능이 들어있는 Architecture

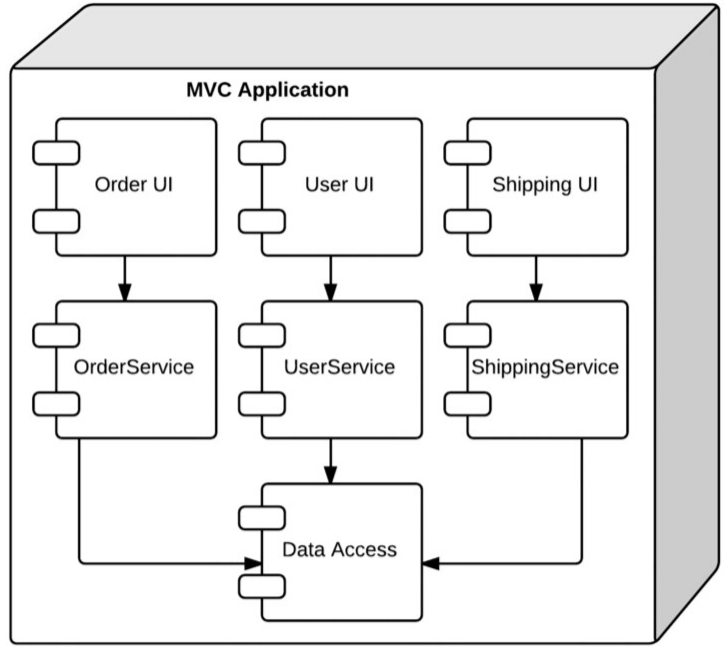


Figure 4: Monolithic Architecture

Monolithic의 문제점

- 빌드/테스트/릴리즈 사이클이 너무 느림
  - 빌드만 수 시간 이상 걸림
  - 하나의 코드베이스 = 많은 커밋 충돌
  - 한달 1~2번 밖에 릴리즈 못함
  - 실험, 혁신은 존재할 수 없는 환경
- 기능 추가가 너무 힘듬
  - 강한 소프트웨어 결합성
  - 개발을 위해 알아야 할게 너무 많음
    - 어디에 어떻게 영향을 미칠지 몰라서...

Q. 근데 우리에게 이게 중요한가? (개발파트이신분?)

Monolithic의 문제점

이것이 우리에게 중요한 것!(운영/관리 입장)

- 스케일링이 힘듬
  - 기능마다 필요한 리소스 종류와 용량이 다름
  - 유연성이 떨어짐
  - 비용 효율화 급감
- 운영과 관리가 힘듬
  - "문제 생겼다. 누구의 문제이지!?"
  - 높은 확률로 자동화가 안되어있음
    - 자동화의 가성비가 없어서
    - "한달에 1~2번 빌드에 무슨 자동화야~ 사람 써!"

"새 업데이트가 있으면 10개 넘는 서버를 직접 수동으로 업그레이드 시켜야해요"

"서버에 문제가 생기면 일일이 다시 복구시켜야해요"

"워크로드를 AWS로 옮기면 오히려 가격이 비싸요"

비슷한 문제를 경험하신 적 있으신가요?

이런 문제 해결하고자
Microservices 언급되기 시작

> (Microservices는 예전부터 논문으로 제시되었던 아키텍처입니다)

새로운 아키텍처 - Microservices

Microservices 도입의 배경

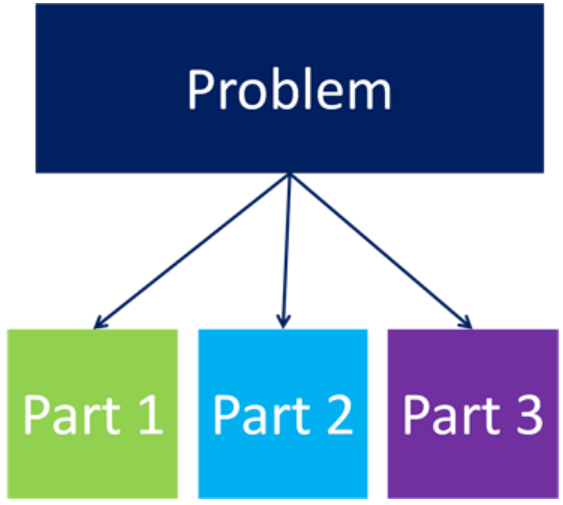
- 빠르게 대응하려는, 혁신을 원하는 서비스 업체에서 시작
  - SI, 치고빠지는 프로젝트형과는 거리가 멀 수도 있음
- 모놀리식은 개발 속도가 너무 느려요.
  - 빠른 세상의 변화, 느린 개발 사이클
  - 코드 결합도가 너무 강해요
    - 기능 개발이 점점 더 복잡해짐
- 큰 서비스를 만들기 힘들어요
  - 큰 CPU, 큰 OS를 모놀리식으로 못만드는거랑 비슷

그러면 모놀리식을 쪼개자(!?)

새로운 아키텍처 - Microservices

쪼개서 문제를 해결하자

Computer Engineering에서는 흔한 문제 해결 방식
(분할 정복 알고리즘-Devide and Conquer)



-> 거대한 모놀리식 서비스를 잘개 잘라진 마이크로서비스로

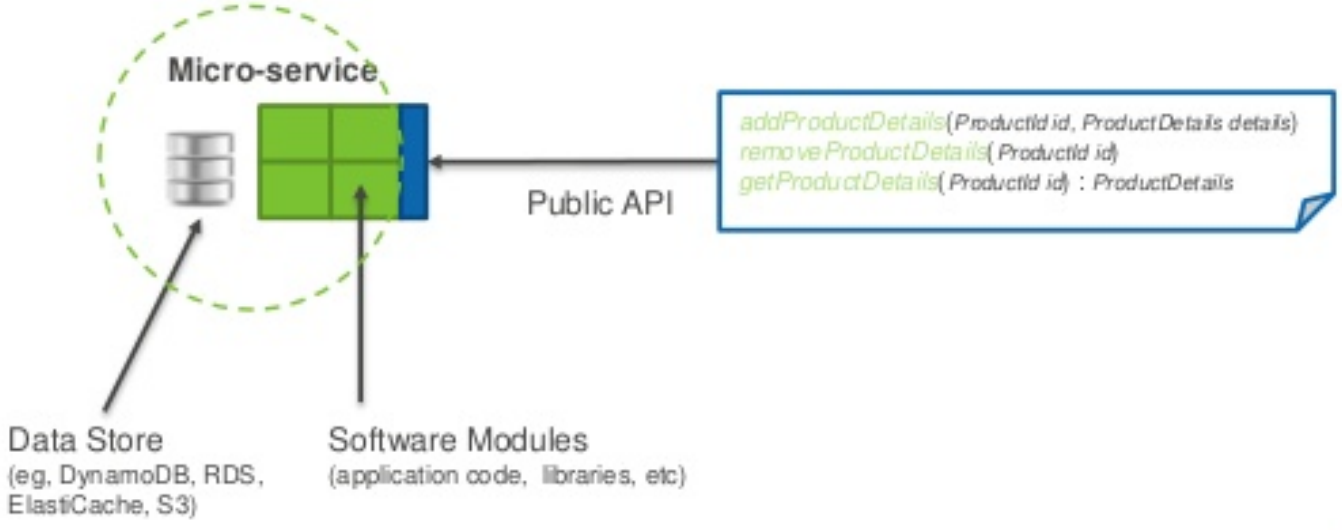
새로운 아키텍처 - Microservices

마이크로서비스의 정의

"복잡한 어플리케이션이 **작고, 독립적인** 프로세스들로 이루어진 소프트웨어 아키텍처.
이 서비스들은 **작고 매우 낮은 결합도**를 가지고 **하나의 작은 작업**에 집중한다. 이로서 시스템 개발에 모듈화된 접근을 가능하게 한다."

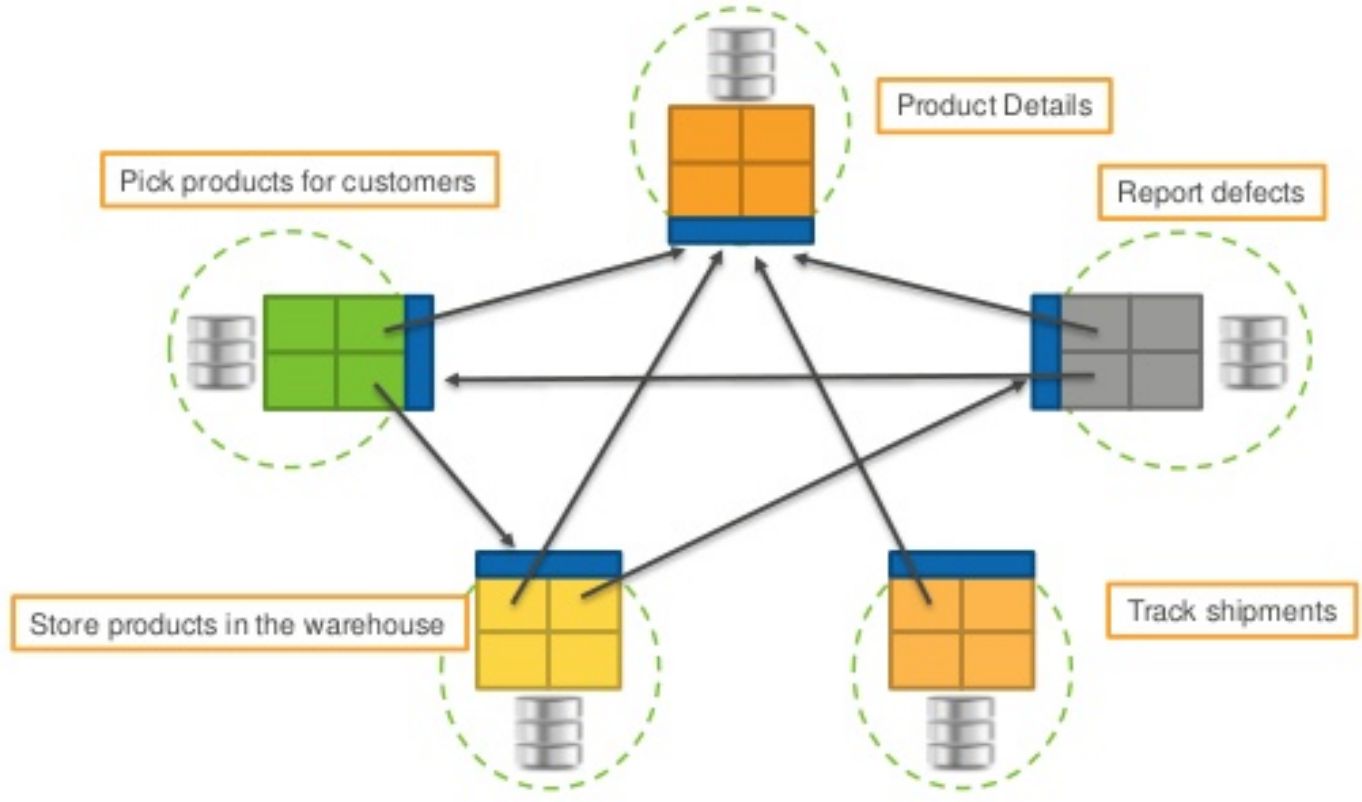
The Anatomy of a Microservice

Micro-service = Fine-grained Service-oriented architecture + “small” public API

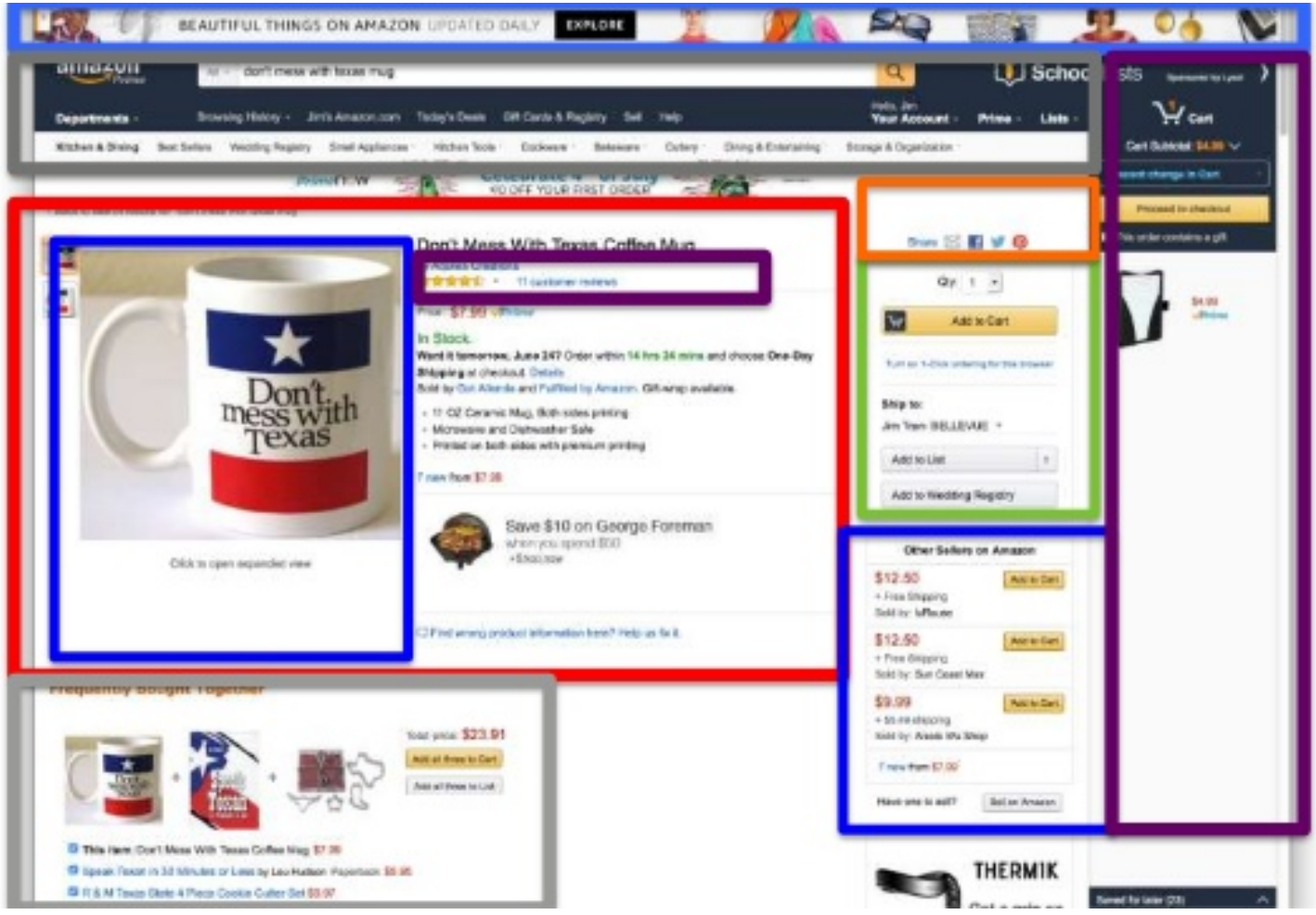


amazon web services

Web of Microservices



amazon web services



새로운 아키텍처 - Microservices

마이크로서비스의 정의

- 분산되어 있는
- 독립적인
- 하나의 일을 잘하는
- 적합한 도구(언어)를 골라 사용하는
- 내부를 몰라도 상호작용할 수 있는
- 한 팀이 직접 개발과 운영을 동시에 하는

새로운 아키텍처 - Microservices

모놀리식 vs 마이크로서비스

모놀리식	마이크로서비스
중앙집중	분산
1 프로세스 - N 서비스	1 프로세스 - 1 서비스
1 언어, 공유 자원	적합한 언어, 독립 자원
에러 = 전체 다운	에러 = 부분 다운
한달 1-2 빌드 사이클	지속적 빌드-배포
비효율 스케일링	효율적 스케일링

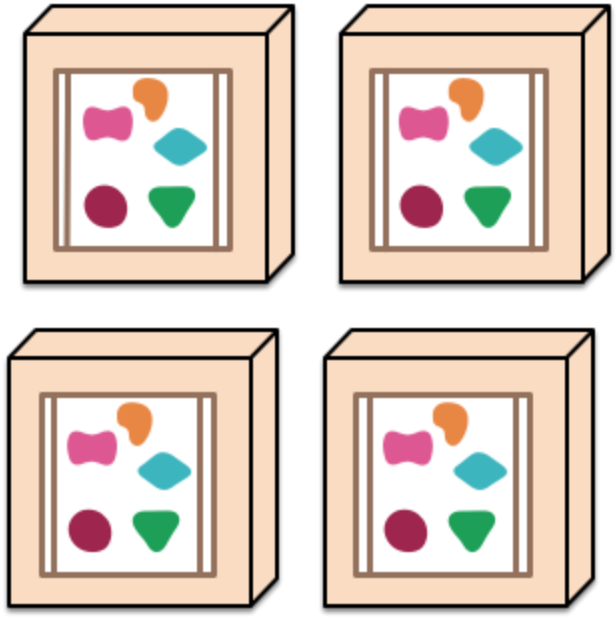
정리하면

“Do one thing, and do it well”

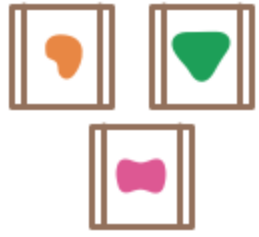
“Do one thing, and do it well”

A monolithic application puts all its functionality into a single process...

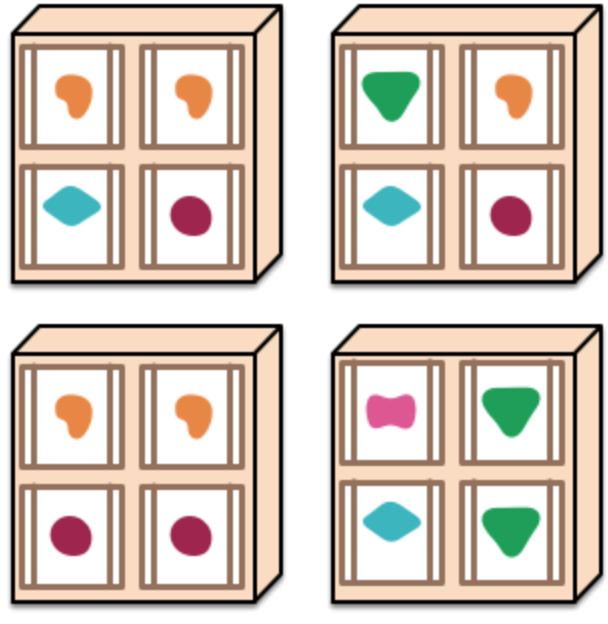
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



정리하면

마이크로서비스 도입하면

개발] 개발은 용이해져요, 쉽지만은 않아요

사업] 많은 실험을 빠르고 리스크 적게 할 수 있어요

설계] 비용, 기능 최적화 가능해요, 복잡하긴 해요

+ 이렇게 안하면 큰 서비스 못해요

...

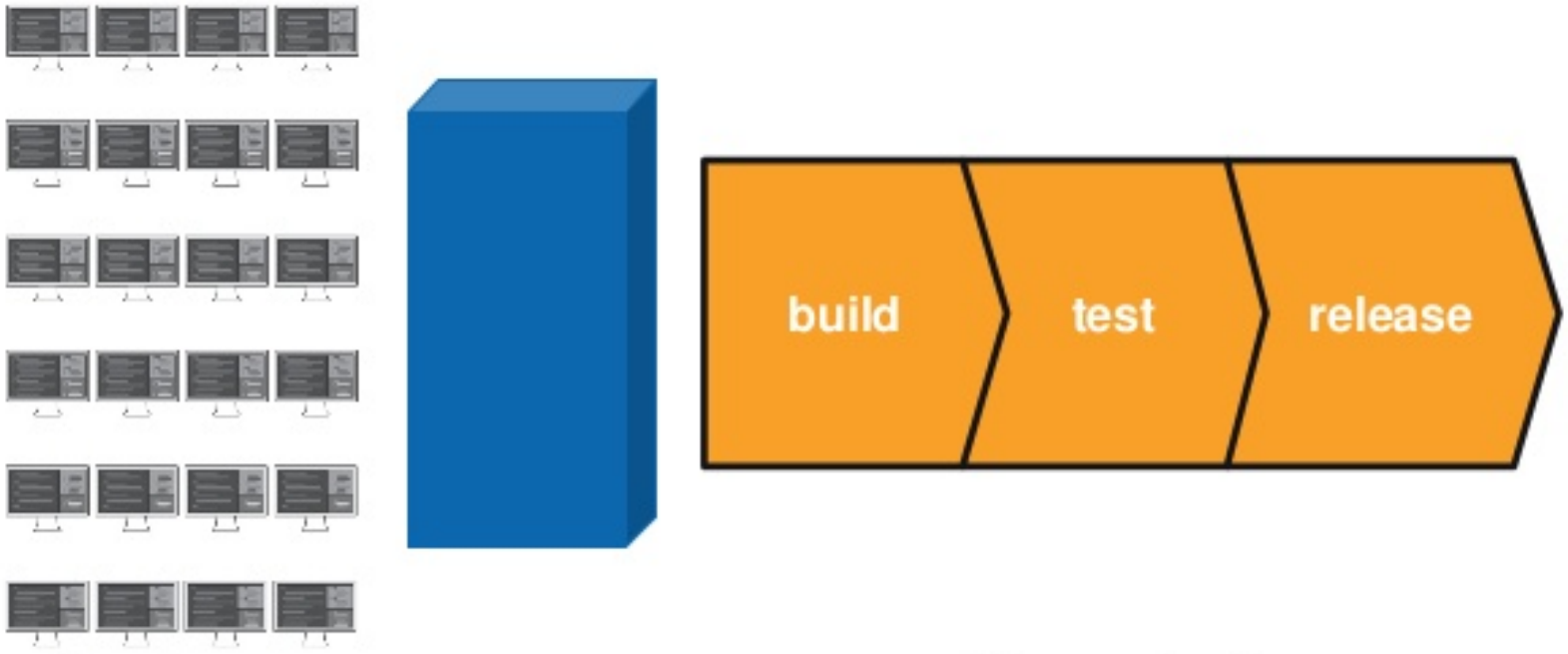
운영] 일이 많아져요!!!

왜 운영 일이 많아지는가?

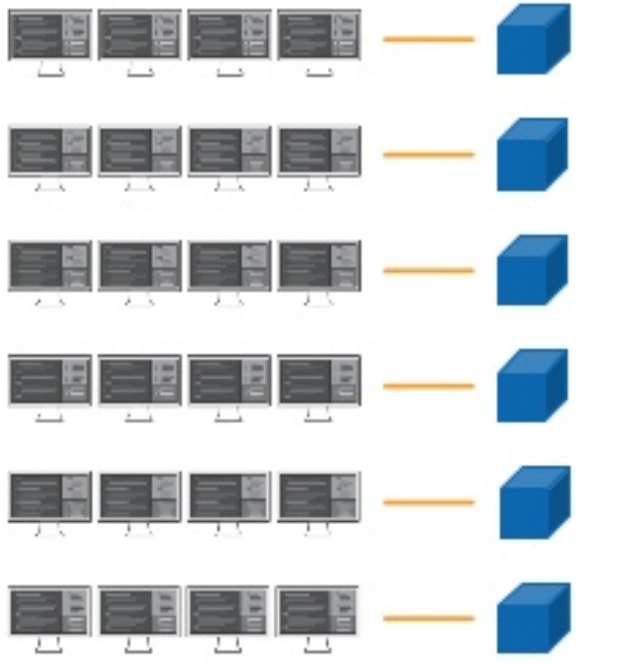
작은 서비스
=> 빠른 개발속도
=> 많은 빌드
=> 많은 테스트
=> 많은 배포
=> 많은 업무량(?!)

안그래도 빡샜는데, 업무량이 더 늘어나는거 아닙니까?

Monolith development lifecycle



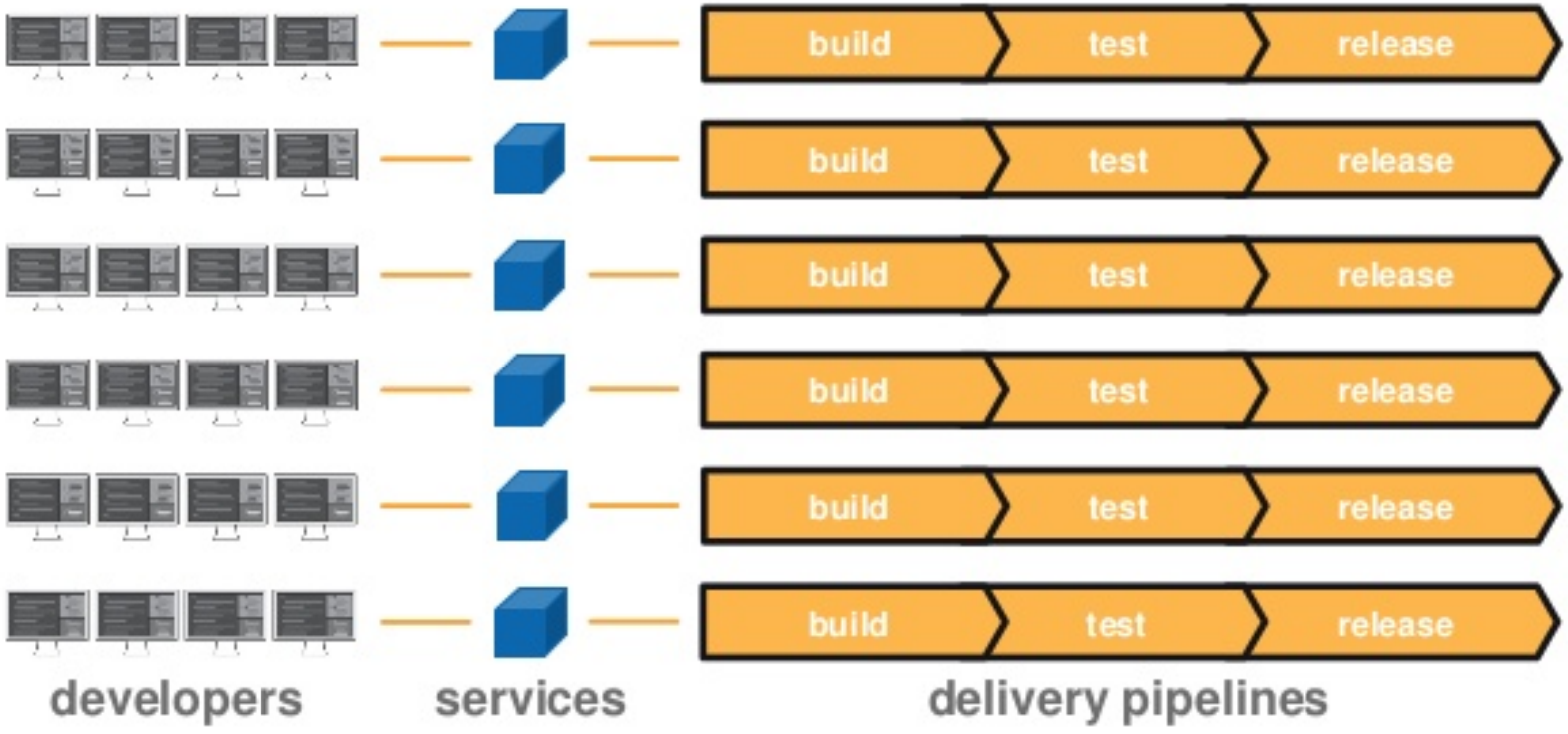
We found out that we had a tooling gap – Missing Tools



So….We decided to built tools to automate our software release process

delivery pipeline

Microservice development lifecycle



자동화의 필요성

그전까진 어떻게든 사람을 갈아 넣으면(?) 됐지만

그러기엔 엄청난 양의 배포가 쏟아짐

=> Key는 바로 Delivery Pipeline

배포 문화의 변화

- 일반적인 배포 --> Continuous Delivery
  - 계속 전달, 전달, 전달
- 지속적인 전달을 위해 자동화
  - 테스트 자동화
  - 배포 프로세스 자동화
- 좋은 자동화 도구들이 많이 나옴

Pipeline 자동화, 실질적인 업무량은 줄어듬

Delivery Pipleline만 최적화 되었나?

- 다른 운영 툴들도 자동화, 최적화
- 운영 업무의 양이 사람의 힘으로는 커버 불가
  - 사람이 하더라도 100배, 10000배 더 편해진 도구 발전
- 자동화는 거대해진 서비스 유지 필수요소

반대로 말하면

운영 조직의 의미가 사라지기 시작함

- 다 자동화 되니...
- 하지만 운영 조직이 사라진다는 의미는 아님
  - 작아지거나
  - 꼭 필요한 Ops만 두거나, DevOps로

Q. 그러면 우리는 어디로 가야하나요?

지난 슬라이드에

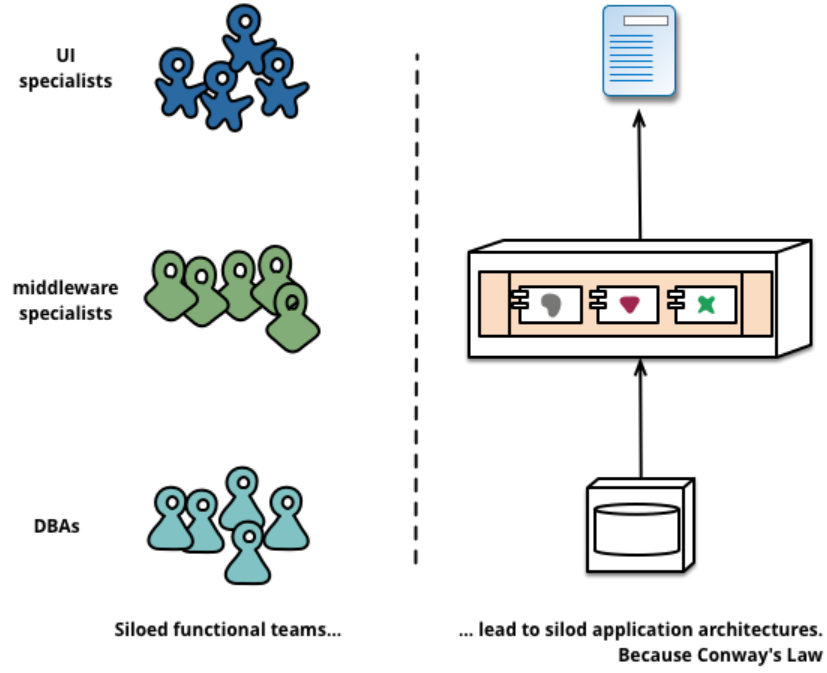
> # 새로운 아키텍처 - Microservices
>
> ## 마이크로서비스의 정의
>
> - 분산되어 있는
> - 독립적인
> - 하나의 일을 잘하는
> - 적합한 도구(언어)를 골라 사용하는
> - 내부를 몰라도 상호작용할 수 있는
> - **한 팀이 직접 개발과 운영을 동시에 하는** <-- 여기를 보면

Conway의 법칙

> Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.
>
> -- Melvyn Conway, 1967

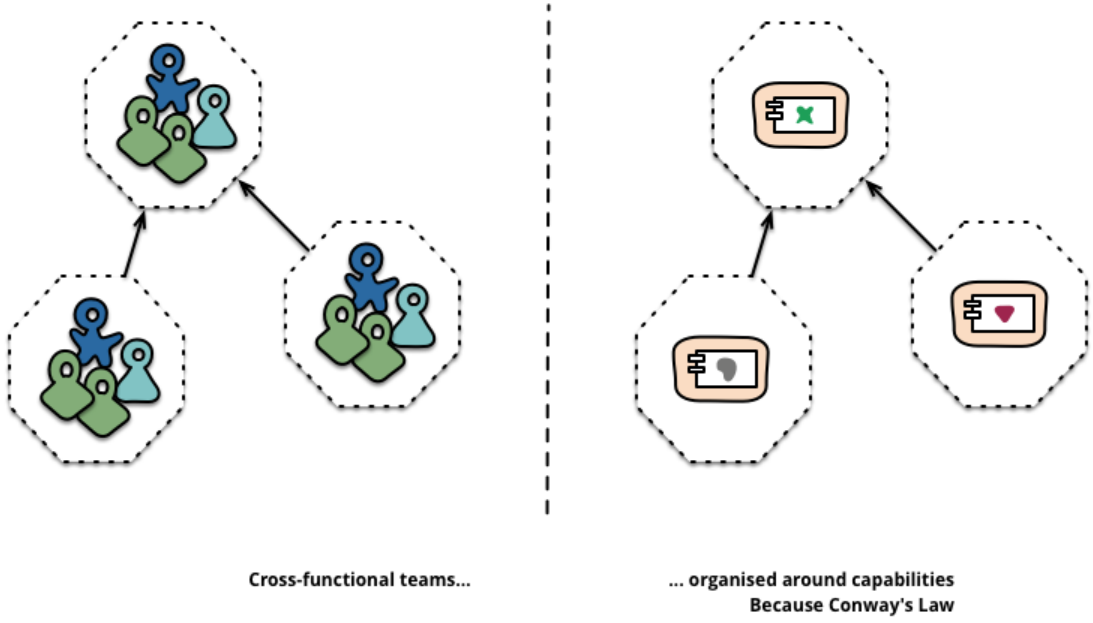
요약 : 그 팀에 그 아키텍처

Conway의 법칙 - 기존의 팀 구성



- 팀간 낮은 이해도 & 전체를 이해하기 쉽지 않음
- 책임 회피 및 전가 하기에 바쁜...

Conway의 법칙 - MSA의 팀 구성



- 작은 서비스 집중 = 작지만 확실한 목표 = 높은 업무 이해도
- Q. 그럼 한 팀에 사람은 어느정도가 적합한가요?

혁신 : 더이상 예전 방식은 싫어요

- 필요 이상으로 오래걸리는 의사결정
  - 장비 추가하는데 결재하고, 기다리고...
- 비전공자에게 설명하는게 쉽지 않음
  - 서로 다른 Domain 지식
- 사용자가 너무 많아졌어요!
  - Human Power론 커버가 안됨
- 그럴바엔 최대한 다 자동화해버리고
  - 개발과 운영을 합쳐버리자

이것이 DevOps

운영 조직의 운명은?

- AI가 나오지 않는 이상 안사라짐
- 대신, 효과적인 도구로 더 많은 리소스를 관리
  - 반대로 말하면, 운영팀 숫자가 작아지게 됨
- SI, 단기 프로젝트 위주만 하면...
  - 나쁜 것은 아니라고 생각함. (사실은 쫌 나쁘다고 생각)
  - 하지만 세상은 변하는데...

계속 필요한 운영 Role

- Public Cloud 지식이 풍부한 사람
  - 더 효과적인 '선택'을 위해
  - 레고처럼 조합하는 세상
    - 어떤 블럭이 좋은 블럭인가? 대답할 수 있는 사람
- 자동화 시스템 개발에 관심있는 사람
  - 비교적 난이도가 낮은 개발
  - 무언가를 만드는 일에 관심이 없다면 비추천
  - 우리가 더 좋은 업무 환경에서 일할 수 있게 만드는 것에 보람을 느낀다면 추천

etc...

계속 신경 쓰셔야 합니다.

- 테스트의 자동화
- 배포 프로세스의 자동화
- 인프라 운영은 AWS가
  - 이젠 서버 죽어도 새벽에 뛰쳐나가지 않아도 돼요
  - Auto recovery
  - fault tolerance (장애 허용 시스템)
- 개발자들이 운영/관리 할 수 있는 세상이 오는 중
  - 코딩, Computer Engineering에 지속적인 관심 필요

Microservices는 특별한 것이 아니에요.

"We do not claim that the microservice style is novel or innovative, its roots go back at least to the design principles of Unix."

> https://martinfowler.com/articles/microservices.html

- 항상 중요하게 여겨야 했던 것
- 하지만 왠지 우리의 업무/일상과는 멀어보이는 이 느낌은?

MSA가 시사하는 바는

- 그들은 계속 혁신하려고 노력한다는 점
  - 코드 레벨뿐만 아니라
  - 조직 레벨에서부터 노력하는 중
- 우리도 그런 정신이 필요하지 않은가?
  - 변화하려는 마음도 좋지만 따라하는것을 넘어서,
  - 우리 조직의 문제점과 불편함을 해소하려는 의지와 용기가 우리에게 있는가?
  - 나는?

기술은 문화입니다.

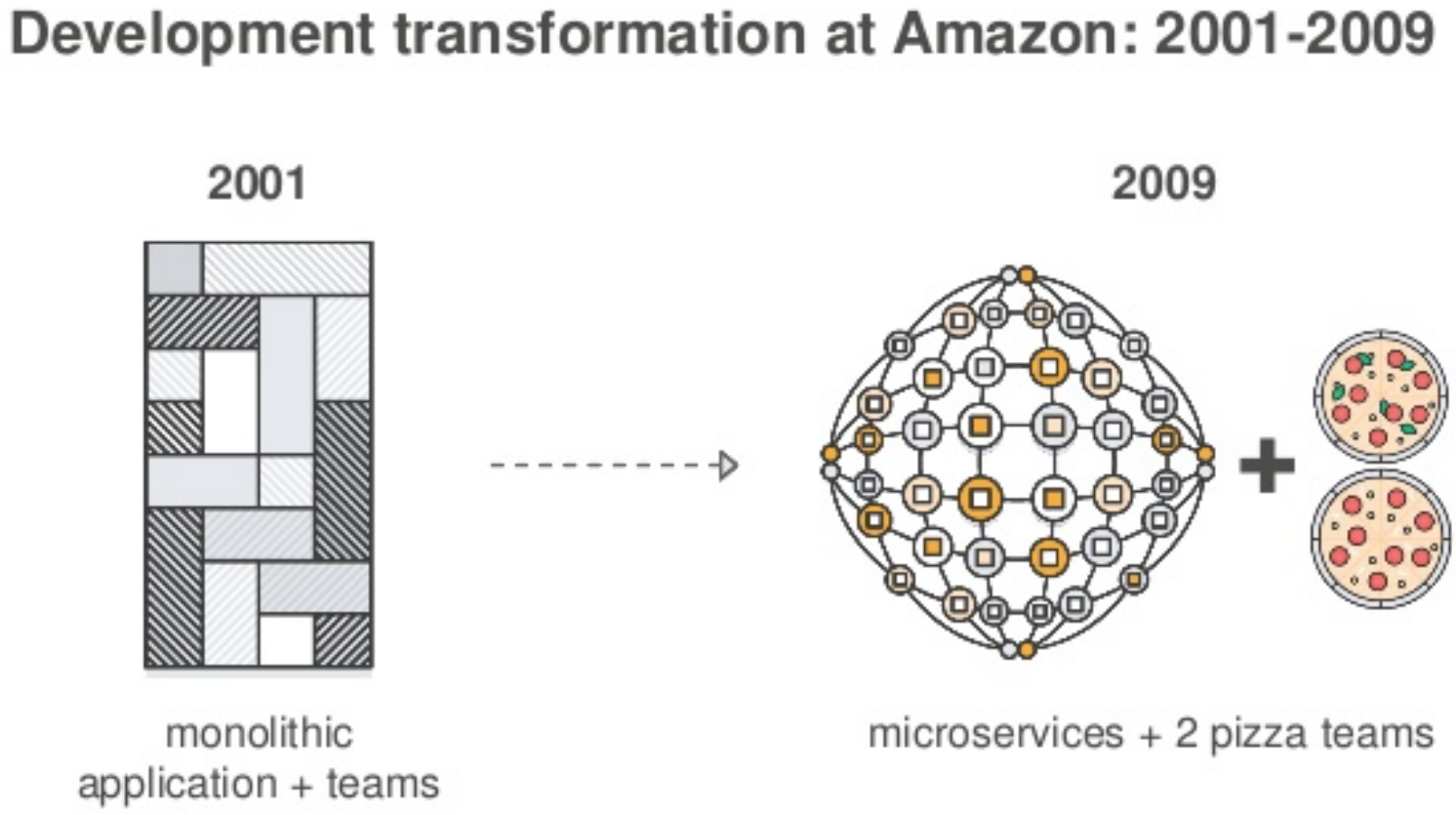
또한 조직의 문제입니다.

> 조직 혁신은 관심 없으면서 남들의 새로운 아키텍처만 따라가 다가는
> 영원한 후발 주자가 될 수 밖에 없다.
> 진정한 의미의 기술의 혁신이 있을 수 없다.

다시 Microservices로 돌아가서..

AWS와는 무슨 관계인가요?

Amazon.com


Development transformation at Amazon: 2001-2009
2001
2009
monolithic
application + teams
microservices + 2 pizza teams

AWS is Microservices Architecture

AWS가 MSA에 적합한 것은 우연이 아님

> AWS has several characteristics that directly address the most important challenges of microservices architectures

- On-demand resources
- Experiment with low cost and risk
- Programmability
- Infrastructure as code
- Continuous delivery
- Managed services
- Service orientation
- Polyglot

MSA를 써야 AWS를 제대로 쓴다

- Monolithic은 애초에 한계가 있다
  - Trade-off 감안?
- Cost Optimization
- 자동화의 세계에 옛날 방식으로 접근하지 마세요.

정리

- Monolithic의 불편함 --> MSA로 해결
- MSA도 불편함 있음 --> 여러 도구로 해결 (AWS, CI/CD)
  - 가장 중요한 것은 조직 문화의 변화로 해결에 도전
- MSA 잘 쓰려면 AWS, AWS 잘 쓰려면 MSA
- 운영 작아지거나 흡수되거나 사라진다
  - 자동화, 효율화, 조직의 변화
- 새로운 인재상
  - 개발 할 줄 아는 운영 (최소한 관심이라도)
  - AWS 잘 알면 금상첨화

이제야 기술적인 이야기를 할 수 있습니다.

- 처음부터 기술 얘기해봤자...
  - '나랑 관계 없는 일인데?'
- 하지만 지금은?
  - '저걸 아는게 힘이구나'
  - '저걸 왜 알아야 하는지 알겠다!'
  - 동기부여는 최고로 중요

앞으로 할 교육들

- 개발, 사업, 아키텍처링 관점의 MSA
- Continuous Integration / Delivery
- AWS Specific한 이야기들
  - Storage의 종류와 특징들
  - EC2 Instances
- Well-Architected Framework
- Domain Driven Design

앞으로 진행할 교육들

> "언어(파이썬) 교육 받고 와도 다음 단계로 공부할 것 찾기가 어려워서 실력이 안늘어 업무에 사용할 자신이 없어요."

- 실습 위주의 AWS 인프라 운용 교육 (코딩위주)
  - Lambda, Cloudformation
  - "이런 Architecture의 Cloudformation Template를 만드시오"
  - "매일 05시에 EC2를 백업하는 자동화를 Lambda 코드를 이용하여 만드시오"
  - 과제(숙제) -> 채점(피드백) -> 모범답안(강의)

단계별 학습 + AWS 실습위주 = 업무 연관성 Up