안녕하세요. 반갑습니다.
저는 클라우드 서비스팀의 남세현이라고 합니다.
이번 교육을 시작으로 아마존웹서비스에 관련된 여러 교육을 진행하려고 합니다.
여러분들이 듣고 싶은 교육이 있으시다면 꼭 저에게 알려주시길 부탁드리겠습니다.

오늘은 여기 써져있는 마이크로서비스 아키텍처 뿐만 아니라 우리가 다루는 시스템과 우리의 조직에 대해서 이야기를 해보는 시간을 가지겠습니다.

여러분, 여러분이 업무에서 만나는 시스템은 어떤 시스템입니까?
여기 나와있는 것 처럼 운영과 관리, 스케일링이 편하고 우리의 귀중한 휴식을 방해하지 않는 시스템입니까? 다양한 실험을 위험성 없이 할 수 있는 시스템인가요?

혹은
운영과 관리, 스케일링이 힘든 시스템입니까? 자동화가 안되어있고, 빌드릴리즈 사이클이 느린 시스템입니까?
한번 거수해보겠습니다. 어느쪽에 더 가깝습니까? 편한 전자인가요, 약간 불편한 후자인가요?
사실 이렇게 완벽하게 흑백으로 나뉘는 시스템은 없을 것입니다. 어느 부분은 편하고, 어느 부분은 불편하겠죠.

근데 저는 이런 질문을 추가로 얹고 싶습니다. 그런 여러분의 시스템은 퍼블릭 클라우드에 적합한 시스템입니까?
쉽지 않은 질문입니다. 또한 이 질문에 대답하기 위해서는 다음의 질문에 먼저 대답할 수 있어야 합니다.

왜 퍼블릭 클라우드를 사용합니까?
회사에서 사용하라고 해서? 우리의 고객사가 아마존 쓰고싶다고 해서? 맞습니다. 사용하라면 사용해야죠. 틀린게 아닙니다.
하지만 저는 퍼블릭 클라우드를 사용하는 이유가 오직 그것 하나뿐이라면, 저는 잘못되었다고 생각합니다. 프로라면 선택과 행동에 대한 분석과 논리에 의한 납득이 있어야 하기 때문입니다.

제가 생각하는 퍼블릭 클라우드를 사용하는 이유는 다음과 같습니다.
퍼블릭 클라우드는, 아마존 웹 서비스는 잘 만들어진 매니지드 서비스들을 제공합니다.
바퀴를 다시 발명하지 마라는 이야기가 있습니다. 세상에 이미 잘 만들어진 도구가 있다면, 처음부터 우리가 그 도구부터 새로히 만들 필요가 없습니다. 적절히 가져다 사용하는 것도 훌륭한 지혜입니다.
아마존 웹서비스는 컴퓨팅, 스토리지, 자동화된 인프라, 로그시스템, 보안시스템, 수많은 매니지드 서비스를 제공합니다. 표준화되어 만들어져있고, 서비스에 대한 관리도 그들이 합니다.

뿐만 아니라 인프라에 대해서도 관리와 보안, 공급을 제공합니다. 쉽게 말하면 데이터센터를 직접 운용해준다는 이야기입니다. 근데 그 데이터센터가 그냥 데이터센터가 아니라 수많은 표준과 보안레벨을 만족시킨다는 것입니다.

퍼블릭 클라우드의 기술력은 업계 최고 레벨입니다. 왜 그럴까요? 업계 최고 일인자들이 퍼블릭 클라우드를 하고 있기 때문입니다. 아마존, 마이크로소프트, 구글. 이름만 들어도 딱 감이 옵니다.

가장 큰 장점은 이 높은 퀄리티의 인프라 시스템을 굉장히 적은 비용으로 사용할 수 있다는 것입니다. 아마존 웹서비스 AWS는 99.95%의 가용성, 그리고 저기 9가 11개나 있는 99.99점점점 퍼센트의 내구성을 제공합니다.
가용성이란 말은 저 퍼센트만큼 컴퓨터가 켜져있는 걸 보장하겠다는 이야기입니다.
내구성은 여러분이 저장한 데이터가 사라질 가능성입니다. 저 정도면 만개의 오브젝트를 저장하면, 하나가 사라질 가능성이 1천만 년에 한번입니다.

이런 인프라를 굉장히 많은 사용자들이 함께 사용합니다. 따라서 그 규모의 경제에 따라 합리적인 가격에 인프라를 이용하실 수 있습니다.
가격만 싼게 아니라 필요하면 늘리고 필요 없으면 줄일 수 있습니다. 사용한 만큼만 돈을 낼 수 있습니다. 서버를 살 때 발생하는 초기비용이 없어지는 것입니다. 그래서 아래 그래프를 보면 온프레미스에서 서비스를 구축했을 때에 비해 최소 68%, 최대 80% 비용을 절감할 수 있는 것을 볼 수 있습니다.

좋습니다. 이게 AWS을 사용하는 이유입니다. 그렇다면 우리의 시스템도 이런 장점을 누릴 수 있습니까?

음... 운영과 관리가 힘든 시스템을 운영관리가 편한 인프라에 올리면?
스케일링이 힘든 시스템을 스케일링이 편한 인프라에 올리면?
과연 그렇게 올린다고 관리가 편해지고 비용이 줄어들까요?
아닙니다. 두 조건이 모두 만족되어야만 우리가 저런 장점을 누릴 수 있는 것입니다.

그래서 오늘은 우리의 시스템이 어떻게 변해야 하는지에 대해 이야기해보자 합니다.

먼저 우리 시스템은 어떤 모양새인지 보겠습니다. 대부분 우리가 만나는 시스템들은 모놀리식 이라는 아키텍처를 가지고 있습니다. 모놀리식이란 것은 단단히 짜여 하나로 되어있다는 것입니다. 그림처럼 하나의 프로세스에 모든 기능이 들어가 있습니다.

이 모놀리식에 어떤 문제점들이 있는지 한번 보겠습니다.
빌드 테스트 릴리즈 사이클이 너무 느립니다. 빌드, 즉 개발자들이 만들어내는 소스코드에서 실제로 돌아가는 프로그램을 만들어내는 과정인 빌드가 수 시간 이상 걸립니다. 왜냐면 한 소스코드에 모든 기능을 다 넣어야 하기 때문에 소스코드의 크기가 굉장히 커지기 때문입니다.

이 하나의 코드 베이스에 많은 개발자들이 접근해서 코드를 수정하고 합니다. 이러면 그 수정사항들, 커밋들이 자주 충돌합니다. 여러분, 엑셀이나 워드를 하나 켜놓고 수십명이 함께 수정한다고 생각해보세요. 내가 5번 페이지 수정중인데, 다른 사람도 그 페이지 같이 수정하면 충돌이 날 수 밖에 없겠죠? 이게 스트레스가 장난이 아닌데, 그 뿐만 아니라 그 충돌을 해결하기 위해 시간이 엄청 많이 소요됩니다.

그러다보니 완성된 프로그램을 서비스로 올려놓는 릴리즈가 한달에 한 두번밖에 할 수 없습니다.
한 달에 한번 올릴 수 있는 상황에서 실험적인 기능을 넣을 수 있을까요? 그거 올렸다가 잘못되면 수정하는데 한달 기다려야합니다. 이러니 보수적인 입장이 될 수 밖에 없습니다. 사용자 입장에서는 항상 진부한 서비스를 볼 수 밖에 없는 것입니다.

기능 추가가 너무 힘듭니다. 방금 말한 것 처럼 시간이 오래걸리는 것도 있지만, 애초에 뭔가를 새로 넣기가 쉽지가 않습니다. 스파게티 코드라는 말을 들어보신 적이 있으십니까? 개발을 하면 시간이 지날수록 내부가 끈적끈적하게 됩니다. 코드가 엉키고, 한 부분이 다른 한 부분에 영향을 미치고, 그게 엄청 많아지게 됩니다. 그러면 뭘 하나를 만들더라도 이게 어디에 어떻게 영향을 미칠지 알기가 굉장히 힘들어집니다.

근데 이건 사실 개발자가 아니라면 공감이 안되는 내용일지도 모릅니다. 여기 개발자이신 분은 없으시잖아요?
제가 오늘 교육은 여러분들의 초점에 맞추기 위해 최대한 개발의 입장이 아닌 운영과 관리의 입장에서 이야기하고자 합니다.

그러면 운영과 관리 입장에서는 어떤 문제점을 가지고 있는지 보겠습니다.
스케일링이 힘듭니다. 자원을 늘리고 줄이고 하는게 힘들다는 이야기입니다.
모놀리식은 한 프로세스에 모든 기능이 들어있다고 했습니다. 하지만 기능마다 필요한 리소스의 종류와 용량이 다릅니다. 프로세스는 하나이고, 필요한 리소스는 다양한데, 어느 기능에 장단을 맞춰서 인프라를 제공해줘야 할까요?
이러니까 리소스에 대해 유연성이 굉장히 떨어지게 됩니다. 리소스를 전부 다 활용하지 못하고 그에 따른 비용이 새어나가게 됩니다. 비용의 효율화가 안되는 것입니다.

문제가 생겨도 어디가 문제인지 알아내기가 쉽지 않습니다. 그리고 흔히 말하는 귀찮은 작업들은 전부 인력을 통해서 해결됩니다. 자동화의 필요성을 느끼지 못하기 때문에, 속된 말로 사람으로 떼우기 때문입니다.

실제로 겪고 계신 문제점들을 모아보았습니다.
-
-
-
비슷한 문제를 경험하신 적이 있으신가요? 한번 공유해주실 수 있으신가요?

자, 다양한 집단에서 이런 문제를 해결하려고 노력하기 시작합니다. 그리고 그 속에서 마이크로서비스라는걸 사용해보자는 이야기가 나오게 됩니다.

마이크로서비스는 트위터, 아마존닷컴, 넷플릭스같은 서비스 업체에서 주로 사용이 시작됩니다.
이 세 회사의 공통점은 사회의 변화에 빠르게 대응하려는 회사이고, 혁신을 원하고, 사용자가 굉장히 많다는 점입니다.
그들은 새로운 것들을 사용자에게 가능하면 더 빨리, 더더욱 빨리 전달하고 싶어합니다. 하지만 모놀리식은 신 기능 개발 속도가 너무 느리고 어렵습니다. 그래서 모놀리식을 쪼개기 시작합니다.

이 쪼갠다는 개념은 컴퓨터 공학에서 굉장히 흔한 문제 해결 방식입니다. 디바이브 앤 컨쿼, 분할 정복한다 라고 하는데, 그 방식을 아키텍처에 적용한 사례라고 볼수도 있겠습니다.

그러면 마이크로서비스를 어떻게 정의내릴 것인가를 보니
(읽으면 됨)

무슨 말인지 그림으로 한번 보겠습니다. 이 작은 원이 하나의 마이크로서비스입니다. 이 마이크로서비스는 하나의 일에 집중합니다. 데이터 저장소도 자기가 들고 있고, 코드, 라이브러리를 들고 있습니다. 그리고 바깥과 통신하는 인터페이스를 API로 제공합니다.

이 그림을 보시면 더 이해가 잘 되실겁니다.
이 전체가 하나의 서비스입니다. 서로 다른 기능들이 모여서 하나의 큰 서비스가 이루어지는 것입니다.
마치 우리가 어떤 업무에 대해서 A 부서에 가서 처리하고, B 부서 가서 마무리하는 것 처럼 여기도 일의 순서에 따라 필요한 마이크로서비스와 상호작용하면서 문제를 해결합니다.

더 쉽게 말하면

- 사진을 보여주는 서비스
- 리뷰 서비스
- 장바구니 서비스
- 추천 서비스
- 검색서비스
- 로그인 서비스
  이런 여러 서비스들이 모여서 하나의 거대하고 복잡한, 아마존 닷컴이라는 어플리케이션이 완성된다는 의미입니다.

마이크로서비스의 특징을 정리해보면
분산되어 있습니다.
각각이 독립적으로 돌아가고요
각각의 마이크로서비스는 자기에게 주어진 하나의 일만 합니다.
독립되어 있기 때문에 자기의 임무에 더 효과적인 도구를 골라서 사용합니다. 기존엔 이게 안되었습니다. 하나의 코드베이스에서 하기 때문에, 여러 언어를 사용하는게 거의 불가능에 가까웠습니다.
잘 정의된 API로 외부에서 요청을 받아들입니다. 규격화가 잘 되어있는 만큼 요청을 하는 쪽에서는 그 내부를 몰라도 편하게 요청을 날리고, 그것에 대한 응답을 기대할 수 있습니다.
마이크로서비스는 한 팀이 직접 개발부터 운영까지 동시에 하게 됩니다.

모놀리식과 비교해보겠습니다.

- 모놀리식은 중앙 집중식이라면 마이크로서비스는 분산형입니다.
- 한 프로세스에 여러 서비스가 돌아갔던 것에 비해 한 프로세스에 한 서비스입니다.
- 한 언어로 모든걸 다 만들고, 자원들도 다 공유되었습니다. 하지만 마이크로서비스에서는 자기가 맡은 임무에 적합한 언어, 적합한 라이브러리를 사용합니다. 자원을 독립하여 서로에게 영향이 안가도록 합니다. 여기서 말하는 자원이란 것은 데이터베이스나 스토리지 같은걸 전부 포함합니다.
- 어느 한 부분이 에러가 나면 모놀리식은 전체가 다운됩니다. 하나의 프로세스에서 돌아가기 때문입니다. 하지만 마이크로서비스는 그 한 부분에 관련된 서비스만 다운됩니다.
- 모놀리식은 한달에 한두번 빌드 사이클이 돌아갑니다. 하지만 마이크로서비스는 지속적으로 빌드와 배포가 이루어집니다.
- 뭉쳐있기 때문에 스케일링이 비효율적이었던 모놀리식과는 다르게 마이크로서비스는 효율적인 스케일링이 가능합니다.

정리하면

이게 모놀리식입니다.

이게 마이크로서비스입니다. 규격화 되어있는게 API같고, 하나의 일을 잘하고, 적합한 재질을 사용하고.

스케일링이야 해봤자 똑같은 걸 여러개 증설하는 것이라면
마이크로서비스는 램이 더 필요한 서비스엔 램이 많은 컴퓨터를, 스토리지가 필요한 서비스엔 스토리지 가득한 컴퓨터를 달아둘 수 있습니다. 인스턴스 갯수도 커스터마이징이 가능합니다.

마이크로서비스를 도입하면
개발팀에서는 신 기능 개발이 용이해진다고 합니다. 하지만 이 새로운 아키텍처가 쉽지만은 않습니다. 알아야 하는게 여러개 있어서 학습을 해야합니다.
사업쪽에선 좋습니다. 예측을 통해 결정하지 않고 이제 실험을 통해 나온 데이터로 결정합니다. 결정에 대한 리스크

가 더 적어집니다.
비용이 줄어들고 기능 최적화가 가능해집니다. 또한 가용성이 늘어나도록 설계할 수 있게 됩니다. 하지만 조금 복잡해지긴 합니다.

그리고 가장 중요한건, 이렇게 안하면 큰 서비스를 못만들고 운영을 못합니다. 트위터가 괜히 모놀리식에서 마이크로서비스로 옮긴 게 아닙니다.

근데... 운영은 일이 많아집니다. "그냥 마이크로서비스만 도입하면" 일이 많아집니다. "그냥 마이크로서비스만 도입하면"이 중요한 포인트입니다.

< 이쯤에서 쉬는게 좋을듯 >

왜 일이 많아집니까?
작은 서비스는 작으니까 개발 속도가 빠릅니다. 개발 속도가 빠르니 빌드 수가 많아지고 그럼 테스트 수가 많아지고 배포 수가 많아집니다. 이거 다 운영업무이지 않습니까?
안그래도 빡샜는데, 업무량이 더 늘어나는거 아닙니까?

기존엔 많은 개발자들이 하나의 어플리케이션을 만들고, 그것을 거대한 하나의 파이프라인이 돌아갔었습니다. 저 파이프라인이 운영팀의 업무라고 볼 수도 있겠습니다.

근데 이제는 작은 팀들이 작은 서비스들을 찍어내기 시작합니다. 그럼 여기 파이프라인은 어떻게 되어야 할까요?

당연한 얘기로, 서비스 만큼 많아지게 됩니다.

얼마나 많아지나면, 아마존닷컴 2014년 기준으로 일년에 5천만건입니다. 한시간에 6천건의 배포가 이루어집니다. 이제까지 사람이 하던 일을 지금은 사람이 하기엔 너무 많은 양입니다. 그래서 이 배포 파이프라인의 자동화가 이루어집니다.

정확하게는 이 배포 파이프라인의 변화, 배포 문화의 변화는 Continuous Delivery 라는 영역에서 다룹니다.
쉽게 얘기하면 유저에게 계속 전달하는 것입니다. 한달 빌드 돌려서 전달하는게 아니라, 부분적으로 완성되면 바로 바로 전달하는 것입니다.
그럴러면 자동화가 필요합니다. 테스트의 자동화도 필요하고, 배포 프로세스의 자동화가 필요합니다.
니즈가 확고하니 좋은 자동화 도구가 많이 나오게 됩니다.
그래서 이 파이프라인의 자동화를 통해, 많아질 뻔 했던 운영의 업무량은 실질적으로 줄어들게 됩니다.

근데 이 배포 파이프라인만 최적화 된게 아닙니다. 다른 운영 툴들도 자동화가 되기 시작합니다.
이유는 사람이 하기엔 양이 너무 많고, 자동화 하면 사람을 안써도 되기 때문입니다. 그리고 무엇보다 굉장히 귀찮은 반복작업이기 때문입니다.

반대로 말하면 운영 조직의 의미가 점점 희미해지기 시작합니다.
운영조직이 사라지는건 아니지만, 굉장히 작아지거나 아니면 데브옵스로 합병이 됩니다.
그럼 마이크로서비스에서 운영과 관리의 운명은 어떻게 되는 것인가요?

지난 슬라이드에 맨 아래에 이런 내용이 있었습니다. 마이크로서비스의 특징 중 '한 팀이 직접 개발과 운영을 동시에 한다'. 이것에 대해 이야기해보겠습니다.

콘웨이의 법칙이란 것이 있습니다.
시스템을 설계하는 조직은 그 조직의 의사소통 방식과 똑같은 설계을 만들어낸다.
즉, 그 팀에 그 아키텍처가 나온다는 것입니다.

기존 우리의 조직은 어땠습니까?
모든게 분리되어있었습니다. 프론트 엔드팀, 백엔드 팀, DBA, 운영, 테스트, 인프라, 모든게 분리되어 있습니다.
그럴 수 밖에 없는게 개발해야 하는게 너무 컷기 때문입니다.
하지만 이러다보니 팀 간에 서로가 뭘 하는지 잘 모르게됩니다. 그러니 전체를 이해하기가 쉽지 않습니다.
이런 상황에서 문제가 발생하면 각 팀은 책임을 회피하거나 책임을 전가합니다. 왜냐하면 전체를 모르기 때문에 어디가 문제인지 정확하게 모르기 때문입니다.

마이크로서비스에서 추구하는 조직은 다음과 같은 모양새입니다.
하나의 팀은 작은 서비스에 집중합니다. 작고 확실한 목표를 가지고 있습니다. 그러니 이 바운더리에서 뭘 해야하는지가 너무 명확합니다. 높은 업무 이해도를 가질 수 밖에 없습니다.
이런 팀들이 서로 상호작용하면서 거대한 문제를 해결해갑니다. 내부를 알 필요도 없습니다. 내부는 숨겨진 채 왜냐하면 그들이 만들어내는 API 문서만 보면 되기 때문입니다.
문제가 발생되면 어느 API에서 문제가 터졌는지만 보면 됩니다. 서로가 결합이 안되어있는, 커플링이 적기 때문에 대부분의 경우 문제가 여러곳에 걸쳐서 나타나지 않고, 확연하게 한 서비스에 나타납니다.
이런 마이크로서비스의 작은 팀은 몇명이 적당할까요?

팀이 이렇게 구성이 될 수 있어야 MSA를 만들어낼 수 있다는 것을 내포하기도 합니다.

어떻게 보면 여러 부분에 걸쳐 모놀리식 조직 구조에 대한 반발심이 생긴거라고 볼 수도 있겠습니다.
왜냐하면 그런 구조에서는 의사결정이 너무 오래 걸리기 때문입니다. 또한 서로 다른 도메인에서 업무를 보기 때문에 문제를 바라보는 시선이 달라지게 됩니다. 이러면 어떻게 말로 표현하기 참 애매한 골치 아픈일이 일어나곤 합니다.
본질적인 문제 중 하나로 사용자의 수가 늘어나니 휴먼파워로 많은 운영업무를 보기 힘들게 된 것도 있습니다.
그래서 다 자동화해버린 다음에 거대한 팀을 쪼개고 개발과 운영을 합쳐서 아까처럼 조직을 형성하게 됩니다.

이것이 DevOps입니다. DevOps는 개발과 운영을 동시에 하는 문화를 의미합니다. 개발자들이 운영도 하게 된다는 의미입니다.

그럼 운영 조직의 운명은 어떻게 됩니까?
사라지진 않습니다. 하지만 효과적인 도구로 더 많은 리소스를 관리하게 됩니다. 반대로 말하면 사람 수가 적어진다는 것을 의미합니다.
물론 이것은 아까 말씀드린 트위터, 넷플릭스, 아마존닷컴같은 회사의 이야기입니다. SI나 단기 프로젝트 위주에서는 이런 변화와는 상관이 없을 가능성이 더 클 것입니다. 왜 별로 변화가 없을거라 생각하는지는 여러분들도 잘 아실

거라 생각합니다.
반대로 말하면 거기에 머물러 있는 것이 얼마나 치명적인 것일지 여러분들도 잘 알고 계실거라 생각합니다.

하지만 계속 필요한, 운영에 좀 더 가까운 롤들이 있습니다.
첫째로는 퍼블릭 클라우드에 대한 지식이 풍부한 사람이 필요합니다. 더 효과적인 선택을 하기 위해서 필요합니다.
잘 만들어진 매니지드 서비스들을 레고처럼 조합하는 세상에서 어떤 블럭이 더 좋은건지 대답할 수 있는 사람이 필요합니다.

혹은 자동화 시스템 개발에 관심있는 사람이 필요합니다.
자동화 AI같은게 있어서 모든걸 처음부터 자동으로 다 셋팅이 되는게 아닙니다. 결국은 처음엔 이니시에이팅, 셋팅을 사람이 해줘야합니다.
이 작업은 비교적 난이도가 낮은 개발입니다.
저는 무언가를 만드는것에 관심이 없으시다면 추천하지 않습니다. 하지만 우리가 더 좋은 업무환경에서 일할 수 있게 만드는 것에 보람을 느끼시는 분이라면 추천합니다.

여러분, UX라는 단어에 대해서 아십니까? 유저 익스피리언스라는 뜻인데, 우리는 우리의 서비스에서 사용자들이 더 좋은 경험을 할 수 있도록 노력합니다. 그렇죠? 근데, 사용자의 경험이 좋아지는거 말고 혹시 우리는 우리의 일하는 경험이 좋아지도록 노력하고 있습니까? 왜 사용자만 편하고 행복해져야 하나요? 일하는 우리도 행복해야합니다.
자동화 개발이란 그런 일입니다. <플레이팅때 이야기>

자 다시 돌아가서, 이런 작업들은 단순히 운영의 경험으로는 할 수 없습니다. 어느정도 개발에 대해 관심을 가지고 공부하신다면 결코 뒤쳐지지 않습니다.

자, 우리는 계속 신경써야합니다.
테스트의 자동화, 배포의 자동화가 오고 있습니다.
심지어 인프라의 운영도 AWS가 해줍니다. 서버 관리, 자동 복구, 장애 허용 시스템의 세상입니다.
저게 뭔말이냐면, <장애 허용 시스템, 자동 복구 설명>

개발자들이 운영과 관리를 할 수 있는 세상이 오고 있습니다. 모든게 다 프로그래밍 가능하고 모든 자원들이 가상화되고 추상화되어 있기 때문입니다.
그래서 우리도 코딩과 컴퓨터 엔지니어링에 지속적인 관심을 가질 필요가 있습니다.
개발자들이 운영을 알아가는 과정처럼, 운영, 관리자들이 개발을 알아가는 과정을 거친다면 결코 꿀릴 게 없습니다.
개발자들이 가장 무서워하는 사람들이 누군지 아세요? <타 전공에서 개발하는 사람 이야기>

<언제 쉬지?>

마이크로서비스는 특별한 것이 아닙니다.
마틴 파울러라는 굉장히 유명한 프로그래머가 블로그에 마이크로서비스에 관련된 글을 올리면서 한 말이 있습니다.
마이크로서비스는 그닥 혁신적인게 아니고, 유닉스 정신, 유닉스 철학을 기반으로 한 것일 뿐이다.
유닉스 정신이라는 것은 개발자들이라면 항상 중요하게 생각하고 있는 것입니다. 여러가지 있는데, 그 중 가장 유명한게 "하나를 잘하는 것을 만들어라"라는게 있습니다.
컴퓨터 공학을 한다는 사람이라면 항상 중요하게 여겨야 하는 것인데, 왠지 이런 것들이 우리의 업무와 일상과는 멀어보이는 느낌을 가지고 있습니다. 왜 그럴까요?

저는 이런 점에서 부터 지금 마이크로서비스나 기술의 흐름을 이끌어가는 기업과의 차이점을 가지고 있다고 생각합니다.
그들은 계속 혁신하려고 노력합니다. 코드 레벨뿐만 아니라 조직 레벨에서부터 노력하고 있습니다.
저는 우리도 그런 정신이 필요하지 않을까 라는 질문을 던져봅니다.
변화하려는 마음, 좋습니다. 하지만 단순히 따라하는 것을 넘어서야 합니다. 근본적인 문제가 무엇인지 찾으려고 노력하고 그 불편함을 해소하려는 의지와 용기가 우리에게 있습니까?
본인에게 있습니까?

시스템 디자인은 조직의 커뮤니케이션 방식과 동일하게 나온다고 말씀드렸습니다. 콘웨이의 법칙 말입니다.
저는 여기서도 콘웨이의 법칙 비스무리하게 적용된다고 생각합니다. 우리 조직의 커뮤니케이션 방식, 우리 조직이 문제를 해결하는 방식, 내가 우리 조직에 접근하는 방식이 곧 우리의 시스템 형식이 되고 우리의 혁신의 모습이 됩니다. 내가 혁신적이지 않으면, 조직이, 시스템이 혁신적일 수 없습니다.

솔직히 우리는 굉장히 조심스럽지 않습니까? 그런 입장을 취하게 만드는 문화가 있는게 아닐까요?

저는 기술은 문화라고 생각합니다. 또한 조직의 문제라고 생각합니다.

마이크로서비스만 도입해서는 문제가 해결되지 않습니다.
조직이 변하지 않으면 그다지 변하는게 없을 것입니다.
조직은 실체가 아니고, 우리가 실체입니다. 개개인이 변해야 조직이 비로소 변하고, 그래야 진정한 의미의 기술의 혁신이 있을 수 있다고 생각합니다.

가장 쉬운 것 부터 변하면 좋을 것 같습니다.
기술에 더 관심을 가지고, 엔지니어링에 관심을 가져보면 좋을 것 같습니다.
어렵다고 도망치지 마시고, 다음 교육들 계속 들으면서 함께 전진했으면 좋겠다는 이야기입니다. 다음 차시부터는 기술적인 이야기가 많이 들어갑니다. 일부러 이번 차시에서는 기술적인 내용을 최대한 배제했습니다.

자 다시 돌아가서 아마존 얘기를 좀 해보겠습니다.
