마이크로서비스 아키텍처 이해하기

클라우드 서비스팀 남세현

저는

- 이것 저것 다양한 경험을 한 프로그래머
- Client, Front-end, Mobile Application
- Server, Back-end, Engine Programming, etc...
- Even low-level programming!
- 개발자의 시점을 더 잘 볼 수 있는 사람
- 클라우드 관련 업무에 어려움이 계시다면 연락주세요!

MSA 이해하기 교육의 대상

- MSA에 대해 들어보지 못한 분
- MSA가 중요하다곤 하는데 왜 그런지 모르는 분

Q. 마이크로서비스에 대해서 들어보신 적이 있습니까?

교육 진행

MSA의 필요성이 왜 나타났는가?

- 기존 시스템에 무슨 문제점이 있었는가?
- MSA가 어떻게 그 문제점을 해결하는가?
- MSA의 도입이 가져온 새로운 특징들은 무엇인가?

MSA는 좋다, 무조건 좋다-같은 이야기를 하지 않습니다.

교육 진행 방향

기술을 이해 할 수 있는가?

- 배경, 숨겨진 것들을 알고 있어야 한다.
- 단순 암기로 당장은 뭐라도 할 수 있겠지만...
  - 응용을 할 수 없다.
  - 기술의 흐름을 탈 수 없다.

실습용 프로젝트 - Twitter Clone

트위터를 따라 만들어졌습니다.

- 로그인
- 글(트윗)쓰기

오늘 여러분들은 Twitter Clone의 개발자입니다.

실습용 프로젝트 - Twitter Clone

프로젝트 환경

- Javascript / NodeJS + MongoDB
- 잘 몰라도 괜찮습니다!
  - 저를 따라하시기만 하면 됩니다.
- 이 프로젝트는 **허구로, 의도된** 프로젝트입니다.
  - 교육을 위해 특정 부분을 과장한 면이 있습니다.

실습용 프로젝트 - Twitter Clone

필요한 소프트웨어 설치하기

- NodeJS
- Atom

NodeJS 설치

Chrome V8 javascript 엔진으로 만들어진 JS 런타임

- https://nodejs.org/en/ 접속 및 설치
- 아무 버전이나 괜찮습니다. (가능하면 6.10.x LTS로)

Atom 설치

Electron(NodeJS + Chromium) 기반의 코드에디터
Github이 주도적으로 개발

사용하시는 코드 에디터가 있는 경우 설치X

- https://atom.io/ 접속 및 설치

실습용 프로젝트 - Twitter Clone

프로젝트 동작 확인하기

실습용 프로젝트 - Twitter Clone

오늘 여러분들은 Twitter Clone의 개발자입니다.

TODO:

- 로컬 로그인 기능 <-- 이것 먼저..!
- 로그 수집 시스템

실습용 프로젝트 - Twitter Clone

TODO : 로컬 로그인 기능

As is (현재)

- Github을 통한 Third Party Login

To be

- 아이디/패스워드를 이용한 Local Login도 가능하도록

실습용 프로젝트 - Twitter Clone

TODO : 로컬 로그인 기능

- 로그인 페이지에 추가
  - 아이디/패스워드 입력창
  - 회원가입 버튼
- 회원가입 페이지

실습용 프로젝트 - Twitter Clone

프로젝트 실행 방법

- cmd 실행 (윈도우키 + R / cmd)
- 소스코드 폴더까지 이동 (cd 명령어)
- cmd에 `npm start` 입력
- **코드 수정시 자동으로 재빌드 실행됨**

실습용 프로젝트 - Twitter Clone

코딩을 할 줄 알아야 하나요?

아니요. 전부 제공됩니다.

디렉토리: TODO/ 안에 있습니다.

실습용 프로젝트 - Twitter Clone

TODO : 로컬 로그인 기능

- 로그인 페이지에 추가
  - 아이디/패스워드 입력창 <--처리중
  - 회원가입 버튼
- 회원가입 페이지

실습용 프로젝트 - Twitter Clone

TODO : 로컬 로그인 기능

- 로그인 페이지에 추가
  - 아이디/패스워드 입력창
  - 회원가입 버튼 <--처리중
- 회원가입 페이지

실습용 프로젝트 - Twitter Clone

TODO : 로컬 로그인 기능

- 로그인 페이지에 추가
  - 아이디/패스워드 입력창
  - 회원가입 버튼
- 회원가입 페이지 <--처리중

개발 완료

그 이후 프로세스는

- 중앙 코드 관리소에 코드합병
- QA팀에서 테스트
- 운영팀에서 배포

빌드 진행중...

...

4시간 후...

... 문제 발생!

옆 팀에서 연락이 왔어요!

코드베이스에 문제가 생김

- 공유 데이터가 꼬임
- 원래 기능이 제대로 동작하지 않는다고 함

누가 잘못했는가?

- 총 5개의 새로운 Commit(코드 수정 단위)이 올라옴
- 그 커밋을 올린 모든 팀들을 다 불러모아야 하나?
- 우린 잘못이 없는 것 같은데...ㅠㅠ

Q. 여러분들이라면 어떤 반응이 나올 것 같습니까?
어떻게 문제를 해결하시겠습니까?

결국 고쳐서 올리는데...

- 커밋 충돌남
  - 서로 같은 파일을 수정해서, 충돌된 파일을 풀어줘야함
- 내가 고치는 사이에 다른 개발자들이 수정사항을 계속 올리기 때문
- 충돌 해결하고 올리는데 또 충돌나고...

어떤 문제들이 있었습니까?

이것이 모놀리식의 일상인가요?

- 개발 업무에 있어서 당연한 것인가?
    - 어쩔 수 없으니 꾹 참고 넘어가야 하는 것인가?

https://www.slideshare.net/AmazonWebServices/introduction-to-microservices-74259780

4~10페이지 -> 13 ~ 16페이지 -> 20 ~ 33페이지

MSA의 정의는 내리기 어렵습니다.

사람마다 생각하는 정의가 다르기 때문입니다.

대신 공통의 특징은 다음과 같습니다.

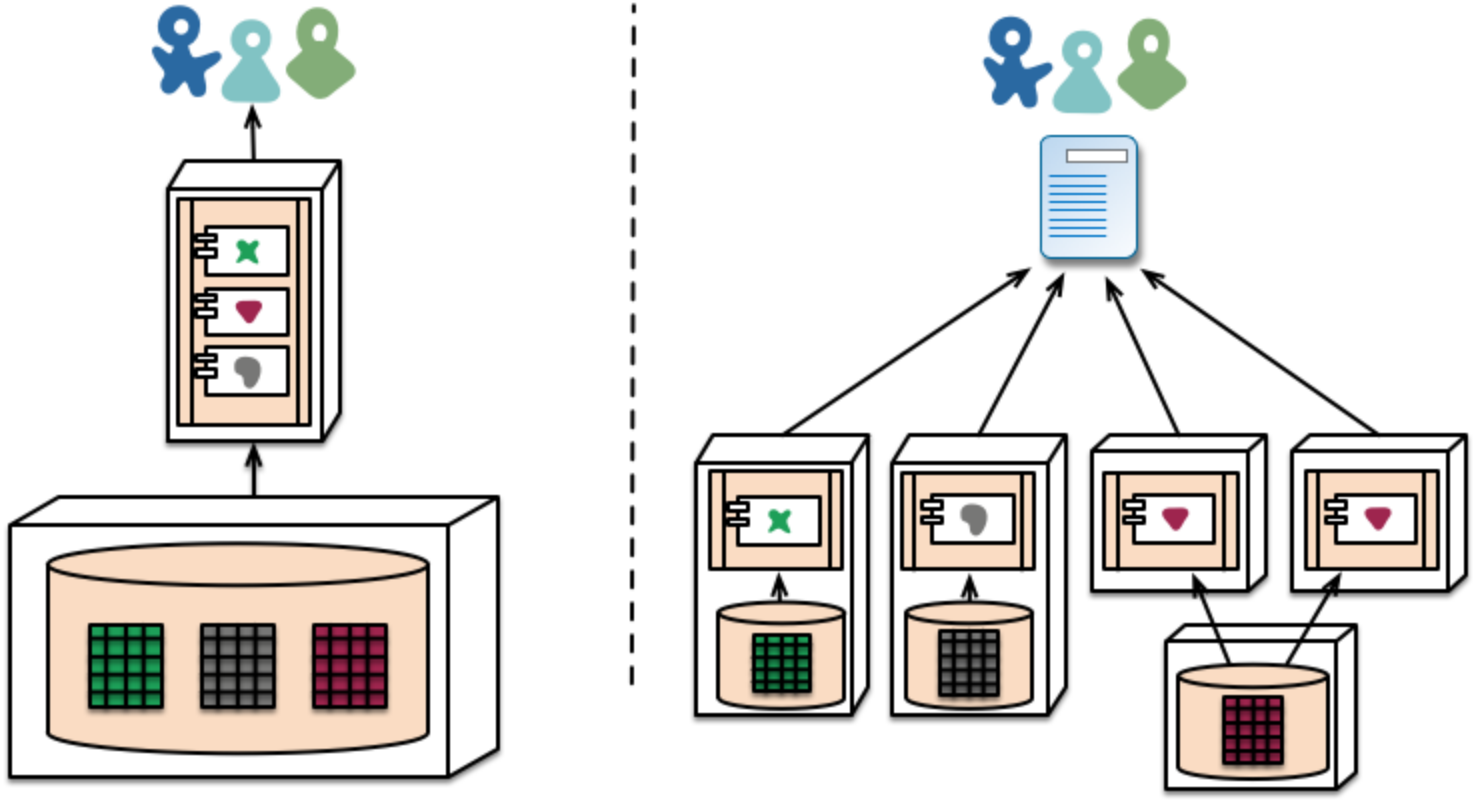
MSA의 특징

- Decentralized
- Independent
- Do one thing well
- Polyglot
- Black box
- You build it, you run it

> 출처 : AWS 백서 / Microservices on AWS

Decentralized

- 분산 시스템
- DB도 분리
  - 전사적 DB 정규화의 고정관념 탈피

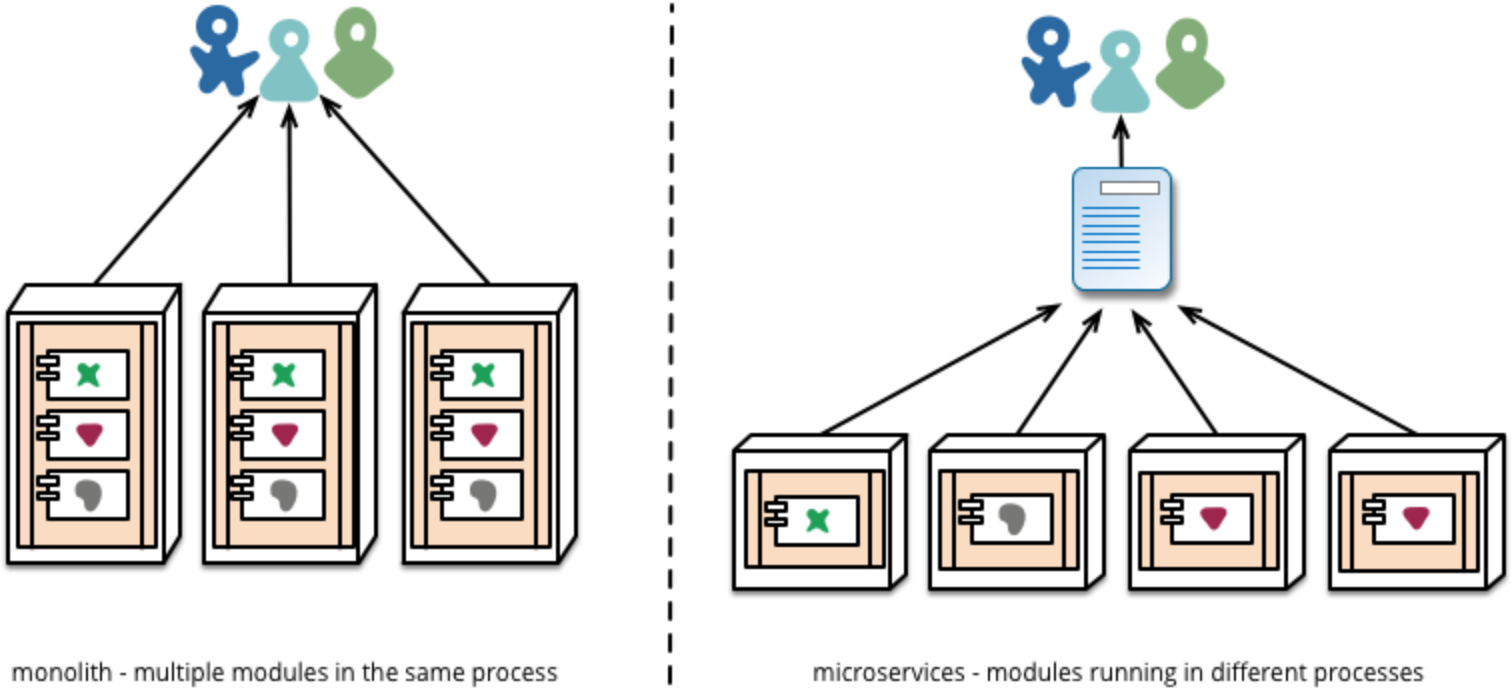


monolith - single database

microservices - application databases

Independent

- 수정, 변경, 교체에 있어 독립적



monolith - multiple modules in the same process

microservices - modules running in different processes

Do one thing well

- Unix 정신
- "하나만 하고, 그리고 그것을 잘 해라"
- 강력한 소프트웨어는 대체로 Simple & Powerful

Polyglot

- 사용하고 싶은 언어를 선택 가능
  - 프로그래밍 언어별 특징을 최대한 살릴 수 있다.
  - Java가 필요하면 Java를, NodeJS가 필요하면 NodeJS를...
- 필요한 DB를 선택해서 사용
  - RDBMS, NoSQL, Columnar-DB, ...

Black box

- 내부가 어떻게 돌아가는지 알아야 할 필요가 없습니다.
    - 잘 정의된 API를 사용하면 됨

Q. 이것이 가능한 이유는 무엇이라 생각합니까?

You build it, you run it

- 서비스 만든 팀이 운영과 관리를 직접 합니다.
  - DevOps (Development + Operation)
- 개발자가 유저와 가까울수록, 서비스 개선이 더 용이해진다.

Q1. 조직의 형태가 아키텍처에 중요하게 영향을 미칩니까?
Q2. 팀의 크기는 얼마정도가 적당합니까?

> ...Conway's Law...

그러면 MSA는 어떻게 문제를 해결합니까?

실습을 통해 알아봅시다.

Twitter Clone에 마이크로서비스 적용하기

...

Q. 모놀리식 서비스에 MSA를 어떻게 적용합니까?

....

Law of Holes

If you find yourself in a hole, stop digging.

--> 새 기능을 마이크로서비스로 개발

새 기능만 마이크로서비스로 개발

주의해야 하는 점

- 기존 모놀리식 서비스와는 독립적일 수 있는 기능인가?
- etc...

실습용 프로젝트 - Twitter Clone

오늘 여러분들은 Twitter Clone의 개발자입니다.

TODO:

- 로컬 로그인 기능
- 로그 수집 시스템 <-- 이제 이것

실습용 프로젝트 - Twitter Clone

TODO : 로그 수집 시스템

As is (현재)

- 로그를 딱히 모으고 있지 않음

To be

- 로그를 모아 분석할 수 있도록 수집
- **모놀리식은 최대한 수정 안하는 식으로**

모놀리식 <--> 마이크로서비스

- 모놀리식의 로그를 마이크로서비스로 보내려면?
- 필요에 따라 적합한 프로토콜을 골라 이용
- HTTP
  - 가장 대중적인 프로토콜
  - 언어와 무관하게 사용 가능한 API 인터페이스
  - TCP based, Securable(SSL)

HTTP의 구성

- Method(Action)
  - GET / POST / PUT / DELETE
- URL or URI
  - `scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]`
  - `protocol://domain/path?query`

`POST | http://log.yourdomain.com/log`

토막상식 : RESTful API

Representational state transfer API

- 인터넷을 통해 컴퓨터 시스템이 상호작용하는 방법 중 하나
- HTTP를 이용하여...
- 예시
  - `http://api.example.com/resources`
  - GET : 리소스를 가져오고
  - POST : 리소스를 만들고
  - PUT : 리소스를 교체하고
  - DELETE : 리소스를 제거하고
- "로그를 쌓으려면 URL(`..com/log`)에 POST로 데이터 보내세요"

토막상식 : URL vs URI

- URL : Uniform Resource Locator
- URI : Uniform Resource Idenfitfier
- All URLs is also a URI
- `Roger Pate` is URI, not URL
- `4914 West Bay Street, Nassau, Bahamas` is URL, also URI

프로젝트 생성하기

1. 폴더 생성

```
mkdir log-analytics
```

2. 패키지 추가

```
yarn add express mongoose morgan body-parser
```

3. `index.js`, `model.js`를 복사/붙여넣기

4. 실행

```
node index
```

로그 수집 시스템

- DB에 잘 쌓이는지 확인해보기
  - Postman
    - HTTP Request를 쉽게 날릴 수 있도록 만들어진 툴
  - cURL
    - Postman이 GUI라면, cURL은 CLI

```
POST /log HTTP/1.1
Host: 127.0.0.1:12345
Content-Type: application/json
Cache-Control: no-cache
Postman-Token: e7d36dc7-0578-491f-50cc-0f500c648f03

{
  "originalUrl": "test",
  "req": "test",
  "user": 123
}
```

모놀리식 수정하기

- 간단하게 API 호출만 하면 끝

```
exports.list = (req, res) => {
  logAnalytics(req);            // 한줄 추가로 끝!
  const page = (req.param('page') > 0 ? req.param('page') : 1)
  const perPage = 5;
  const options = {
    perPage: perPage,
    page: page
  };
  return User.list(options, (err, users) => {
  ...
```

```
function logAnalytics(req) {
  return fetch('http://127.0.0.1:12345/log', {
    method: 'POST',
    body: JSON.stringify({
      ip: req.ip,
      originalUrl: req.originalUrl,
      userID: req.user._id,
    }),
  });
}
```

로그가 쌓이는지 확인해보기

실습해보니, 모놀리식 vs 마이크로서비스

- 기능 수정을 위해 4시간씩 기다릴 필요가 없습니다.
- 코드의 복잡성이 줄어들었습니다
  - (업무의 난이도와 복잡성도...)
- Q. 여러분의 생각은 어떻습니까?

마이크로서비스의 장점

- Agility
- Innovation
- Quality
- Scalability
- Availability

마이크로서비스의 장점

Agility

- 작아진 팀
- 높은 업무 이해력
- 빠른 의사결정
- 빠른 개발 사이클

마이크로서비스의 장점

Innovation

- 한 팀이 개발-운영(DevOps)
- 불필요한 마찰이 없어짐
- 새로운 아이디어를 테스트하기 좋은 환경이 만들어짐
- 빠르게 실패하세요! : 더 적은 부담, 더 다양한 실험

마이크로서비스의 장점

Quality

- Simple is Best
- Unix Philosophy
- "Do One Thing and Do It Best!"
- AWS S3가 잘되는 이유도 이 때문이 아닐지...

마이크로서비스의 장점

Scalability

> "Fine-grained decoupling of microservices is a best practice for building largescale systems"

- 효율 최적화의 전제조건
  - 부분별로 잘 나뉘어진 시스템 = 마이크로서비스
  - 해당 기능에 최적화된 기술, 언어, 장비, etc...
- Decoupling = Easy to scale horizontally!
  - Vertical Scaling의 한계 극복
  - 자연스럽게 장애 회복 탄력성도 증가

마이크로서비스의 장점

Availability

- 한 부분이 죽어도, 전체가 다운되지 않음
- 기술이 점점 발전
  - health-checking, caching, bulkheads, circuit breakers
  - 이쪽은 Netflix가 최고라고 '카더라' - 여러 OSS 공개
  - Availability 개선을 위해 임의로 서버를 죽이는 Bot도 운용
  - Chaos Monkey - http://github.com/netflix/chaosmonkey

마이크로서비스는 장점밖에 없나요?

MSA는 Silver Bullet이 아닙니다.

분명한 Trade-offs와 Challenges를 가지고 있음

- Distributed system -> Fallacies of Distributed Computing
- Migration from Monolithic to microservices isn't Easy
  - 분리 가능한 경계선에 대한 이해, DDD에 대한 이해
  - 이어폰 100개가 엉켜있는 것 같은 코드를 풀어내야 함
  - 조직과 관련된 문제 - 우리 조직은 변화할 수 있는 조직인가?

모 회사 전 CTO의 말씀

- MSA를 권장하지 않습니다.
  - 실력이 전체적으로 좋은 팀만이 운영 가능
  - 좋은 커뮤니케이션이 전제
  - 서비스 개발에 유용, SI나 프로젝트식의 만들고 빠지는 식엔 부적합

https://www.slideshare.net/Byungwook/micro-service-architecture-52233912

Conway's Law

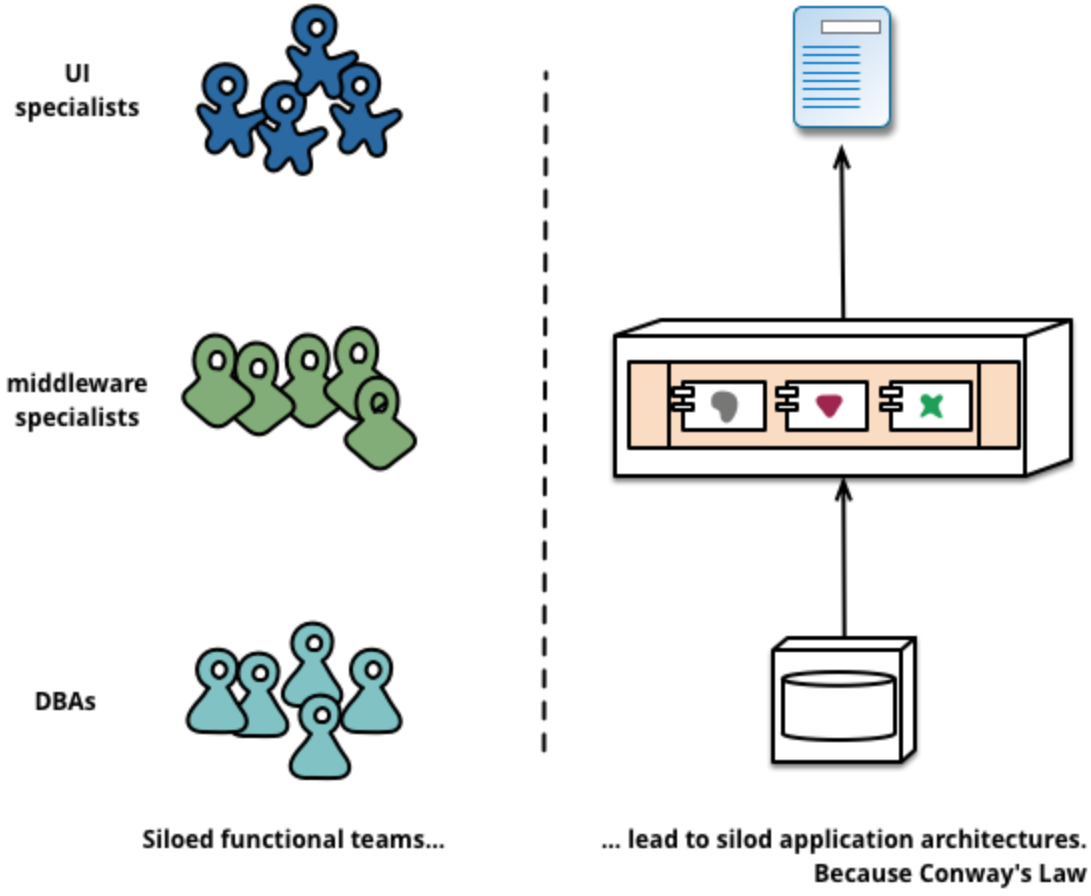
> organizations which design systems ... are constrained to **produce designs** which are **copies of the communication structures of these organizations** - M. Conway

- 아키텍처는 그 팀의 의사소통 방식처럼 나온다.
- 빠른 변화, 혁신을 위해서는 그에 맞는 팀 문화가 필요함
- 과연 우리팀은, 우리 조직은 그런 준비가 되어있는가?

더 자세한 내용은 DevOps를 공부...

Conway's Law의 예시

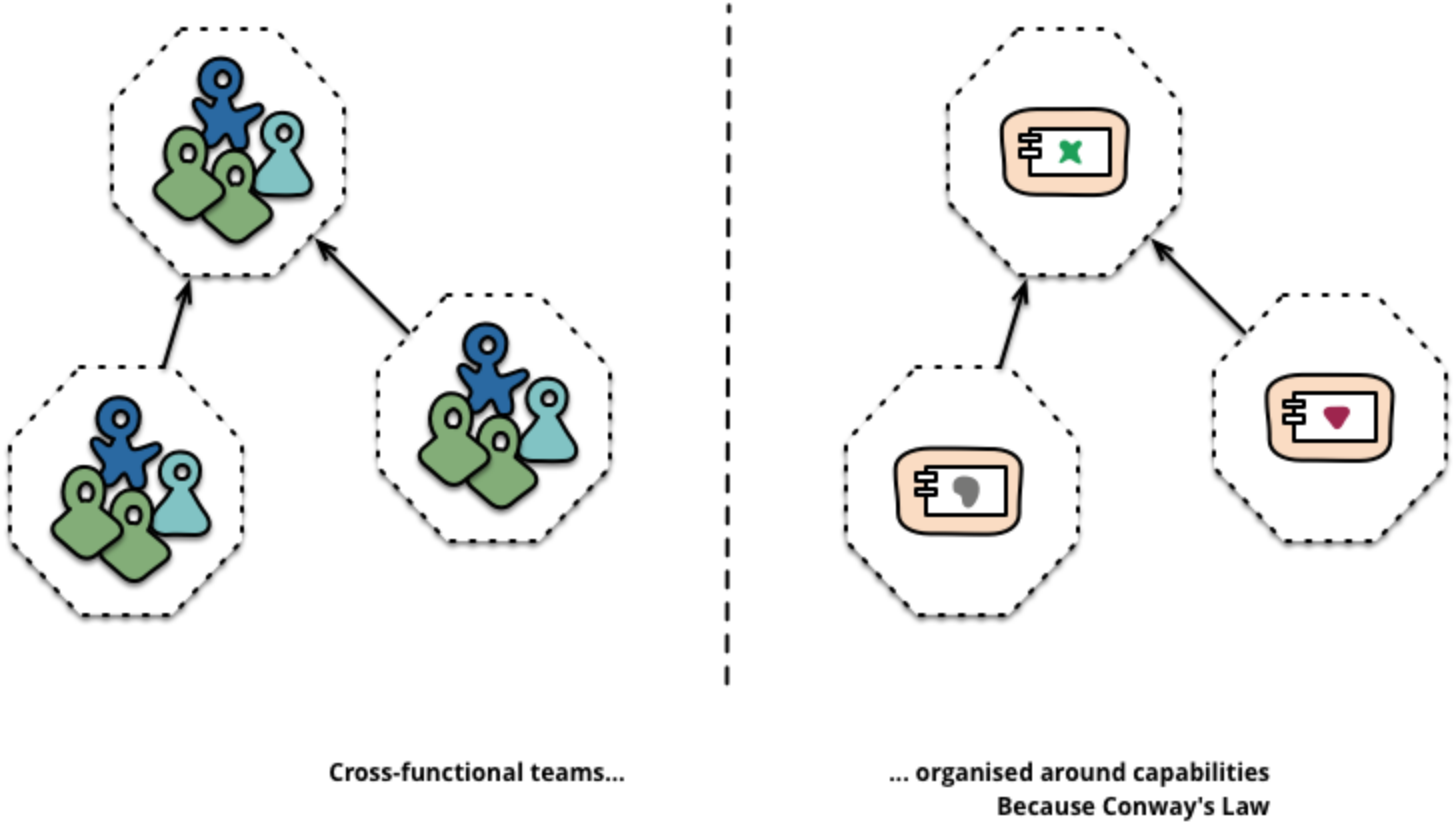
단절된 팀구조는 단절된 아키텍처를



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

Conway's Law의 예시

상호작용하는 팀구조는 상호작용하는 아키텍처를



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

Fallacies of Distributed Computing

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

운영의 복잡성

How to

- provision resources in a scalable and cost-efficient way?
- operate dozens or hundreds of microservice components effectively without multiplying efforts?
- avoid reinventing the wheel across different teams and duplicating tools and processes?
- keep track of hundreds of pipelines of code deployments and their interdependencies?
- monitor overall system health and identify potential hotspots early on?

운영의 복잡성

How to

- track and debug interactions across the whole system?
- analyze high amounts of log data in a distributed application that quickly grows and scales beyond anticipated demand?
- deal with lack of standards and heterogeneous environments including different technologies and people skill sets?
- value diversity without locking into a multiplicity of different technologies that need to be maintained and upgraded over time?

못할 건 없습니다!

AWS가 도와줄 수 있습니다.

Microservices and the Cloud

온디멘드 자원

- 필요한, 다양한 자원을 곧바로 충원

최고의 실험 공간

- 규모의 경제가 주는 낮은 가격
- 논리적으로 완전 분리된 환경이 주는 낮은 리스크
- 작게 시작해서 거대하게 스케일링이 가능

Programmability

- API, CLI, SDK 제공
- 반복작업, 모니터링은 인간이 아닌 컴퓨터가
    - 더 이상 새벽에 깰 필요가 없어요..

Microservices and the Cloud

코드로 관리하는 인프라

```
AWSTemplateFormatVersion: '2010-09-09'
Description: A simple EC2 instance
Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-2f726546
      InstanceType: t2.micro
```

- 이게 가능해지니 DevOps가 생긴게 아닌지..

Microservices and the Cloud

Continuous Delivery

Programmability으로 장비 충원, 배포의 자동화가 가능해짐

- 장비에 새로운 버전의 소프트웨어를 지속적 배포
- 사용자에게 더 빠르게 전달되는 서비스
- 이젠 한달에 한번 배포하지 않아도..!

Microservices and the Cloud

Managed Services

> Don't reinvent the wheel!

- Computing (EC2, EMR, ECS)
- Storage (S3, RDS, DynamoDB)
- Monitoring (Cloudwatch)
- etc...

대부분의 기능이 한 역할만 제대로 수행하는 형식.

- 다양하게 응용이 가능함

Microservices and the Cloud

Service Orientation

AWS 마저도 Microservices로 되어있음

- AWS의 기능들은 MSA의 문제점들을 해결하려는 중요한 도구들
- 마치 레고블럭처럼 조합하여 문제 해결 가능

Microservices and the Cloud

Polyglot

- 다양한 Storage (S3, NoSQL, RDBMS)
- 다양한 DB (Aurora, Mysql, Oracle)
- 다양한 OS (Linux, Windows)
- 상용 소프트웨어도 Market Place에서 구매가능

정리하면

- Monolithic : 모여있음의 불편함
- Microservices : 잘 나뉘어진 작은 서비스들의 상호작용
- Conway's Law : 조직의 변화가 중요하다
- Silver-Bullet이 아니지만, 현재 필요한 기술은 맞다
- MSA의 도전과제를 해결하기 위해 노력해야 한다

AWS는 MSA를 잘 쓰게 도와준다
= AWS를 잘 쓰려면 MSA를 잘 써야한다

복습용 자료

- Martin Folwler's Microservices
  https://martinfowler.com/articles/microservices.html
- AWS 백서 / Microservices on AWS
  https://d0.awsstatic.com/whitepapers/microservices-on-aws.pdf

다음으로 무엇을 공부하면 좋을까요?

- Domain Driven Design
- Continuous Integration & Deployment (CI/CD) / DevOps
- Serverless Architecture
- AWS Well-Architected Framework
- Storage on AWS
- EC2 T2 and Other Instance Families
- Docker Conatiner
- AWS Cloudformation
- etc...