

Going Fully Serverless In Real World



이상현

- AWS Serverless Hero | (2019 ~)
- Mirror | CEO (2022.11 ~)
- 캐치패션 | CTO (2019.05 ~ 2022.06)
- 빙글 (2013.02 ~ 2019.05)

목차

1. 증거 - Serverless로 다 만들수 있다
2. 한계 - Serverless로 이건 못만든다
3. 고찰 - 도대체 왜 Serverless로 안만드는걸까?
4. QnA

실무에서,
지금 당장,
(거의 모든 경우에)
완전히 Serverless로
만들수 있습니다.

1. 증거:

제가 지난 6년동안
Serverless로만 했습니다.
(지금도 하고 있습니다.)

빙글

- <https://www.vingle.net/>
- 관심사 기반 커뮤니티 서비스
- 2016년 정점엔 MAU 8백만명. 앱 사용자만 2백만명
- Ruby on rails 단일앱
 - (1.5년동안 재작성)
 - typescript / NodeJS serverless microservices
- EC2 → Lambda로 옮기면서 AWS 비용 90% 이상 절감함
- 가장 많을때, Backend팀 5명
- 최종적으로 RDS / Redshift / Redis 등 일부 데이터 스토리지 제외하면 완전 Serverless

캐치패션

- <https://www.catchfashion.com/men>
- 해외 명품 직구 Aggregator 서비스
- 상품이 4백만개, SKU가 1천만개
- 2020년 매출 500억
- 처음부터 완전히 Serverless로 구축.
- Aurora RDS, Cache용 Redis 제외하면 Fully Serverless
- 블랙프라이데이 당일에도 서버 모니터링 / 스케일링 아무도 안함...
- Fronted / Backend / iOS 앱 / Android 앱 전부,
개발자 8명이함
- 한달 AWS 비용이 백만원 미만 (그것도 절반이상 DB / Redis...)

- **TypeScript HTTP API Framework with OpenAPI Support on Lambda + ApiGateway**
(API Gateway + Lambda)
- **Server side rendering for ReactJS**
(CloudFront + Lambda@Edge + API Gateway + Lambda + S3)
- **Background Jobs (static cron, dynamic schedule, retries...)**
(SQS + Lambda / KinesisStream + Lambda / Cloudwatch + Lambda)
- **Multi region cache(redis) synchronization**
(KinesisStream + Lambda + S3 + Redis)
- **User behavior data collecting / processing / recommendation**
(KinesisFirehose + S3 + Lambda + Athena)
- **Image / Video (GIF => MP4) encoding**
(S3 + DynamoDB + Lambda)
- **Realtime chat**
(DynamoDB + API Gateway + Lambda)
- **Crawling 30k+ web pages per day on-demand**
(SQS + Lambda + DynamoDB + S3)
- **ML image inference on-demand**
(API Gateway + Lambda + S3)
- **E-Commerce transaction management**
(API Gateway + Lambda + S3)
- **Dynamic Sitemap Generation (~4M pages)**
(CloudFront + Lambda + S3 + Athena) + (Lambda + S3)

그럼 왜
“(거의 모든 경우에)” 냐?

2. 한계

2023년 현재 Serverless의 한계

1. 하드웨어

1. CPU / 메모리가 매우 많이 필요한 작업은 불가능하거나 비효율적.
2. network로 < 5~10ms 이내의 응답이 보장되어야 하는 작업은 어려움.
TCP Socket 직접 통신 안됨.

2. 인프라

1. RDS →
Aurora Serverless 등이 있긴 하지만,
아직도 진정한 의미의 “Serverless”라고 하긴 힘듦.
2. Cache / Search →
Redis / ElasticSearch에 대한 완벽한 Serverless 대체제가 없음.

3. 언어

1. 가상 머신이 필요한 JVM / C# 계열은 여전히 일부 페널티를 받음.
(하지만 이젠 매우 소소한 수준이긴 함)
2. JS / Python / Rust / Go등 런타임이 가벼운 언어들이 이득

그래서 2023년 기준 “현실적인” Serverless 목표는:

1. EC2를 한대도 직접 쓰지 않는다.
 - 모든 Application Code는 Lambda로
2. 외부 인프라를 전부 Serverless 대체제로 만든다.
 - 단순 Key-Value Store Redis다 => DynamoDB
 - Hadoop 으로 뭐 하는거다 => S3 / Athena
 - DynamoDB, S3, Athena, Kinesis ...
3. 컨테이너 / EC2 기반 서비스는 정말로 정말로 불가피할때만 쓴다.
 - 특히 어플리케이션 특성상 퍼포먼스나 기능이 도저히 구현이 불가능한 경우
 - ex) ML training / TCP Socket 기반 실시간 게임...

제약이 뭐 이렇게 많나..
싶으시겠지만

다시 말해,
당신이 매일 만들고 있는 그

DB + Rest API + Background Job

당연히 Serverless로 만들수 있습니다.

ZDC302



How CATCH FASHION built a serverless ML inference service with AWS Lambda

Kurt Lee
CTO
CATCH FASHION

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

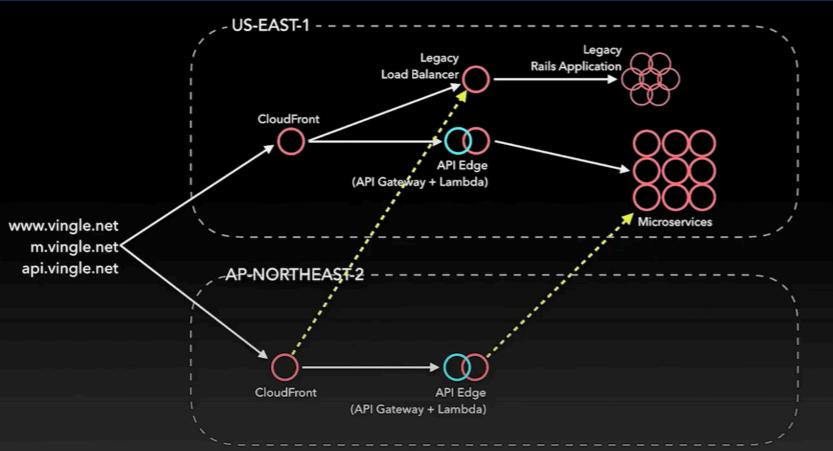


Kurt Lee
Tech Lead
Vingle

Kevin Kim
SA Manager
AWS

Today, we'll be joined by Kurt Lee of Vingle

SUMMIT SEOUL



Lambda@Edge를 통한
점진적 서비스 이전 및
멀티 리전 트래픽 길들이기

이상현
Tech Lead, 빙글

© 2018, Amazon Web Services, Inc. or Its Affiliates. All rights reserved.

36:45

AWS SUMMIT Seoul

Vingle 서비스 및 코드 개발 현황

20,101 commits
5 branches
169 releases
26 contributors

Web server / Background Worker / REST API / ORM / Business logic	Feed Search Card Writing Admin Spam	Unit test - 30분 Deployment - 30분
--	---	-------------------------------------

이상현 Technical Leader
빙글

17:34

3. “그럼 도대체 왜 다들 Fully Serverless로 안만들지?” 에 대한 고찰

결론:
기술이 아니라 인간과 조직의 문제다.

내가 있는 조직 / 회사에서 Serverless를
도입하고 싶다면 해결해야할 문제들:

1. 관성
2. 합리화
3. Marketing-Driven 기술결정

3.1. 관성

- 새로운 기술을 배우고 적용한다 Y/N
= (새로운-기술의-이득) >
(새로운-기술을-배우는-노력) + 강-원래-하던거-하려는-관성)
- 심지어 영어로 된 문서 찾아봐야 한다?
- 심지어 한국 대기업에서는 이걸 한 사례를 못찾겠다?
- 심지어 결과적인 이득이
“개발자들이 운영 업무를 안해도 되게 되는” 거다?

3.2. 합리화

- 일반론으로, “Cloud Native로 한다”
= “Serverless / Managed Service를 적극 활용한다”
IDC에서 돌리던걸 그대로 그냥 EC2에 올리는건 Cloud Native가 아님..
- 또, “Serverless로 뭔가를 만든다”
= “뭔가를 Fully Stateless, Distributed System으로 만든다”
- “Lambda로 했더니 이게 안돼요” (**X**)
“Cloud Native하게 이걸 해결하는 방법을 모르겠어요” (**O**)
“이걸 Stateless, Distributed System으로 만드는 방법을 모르겠어요” (**O**)

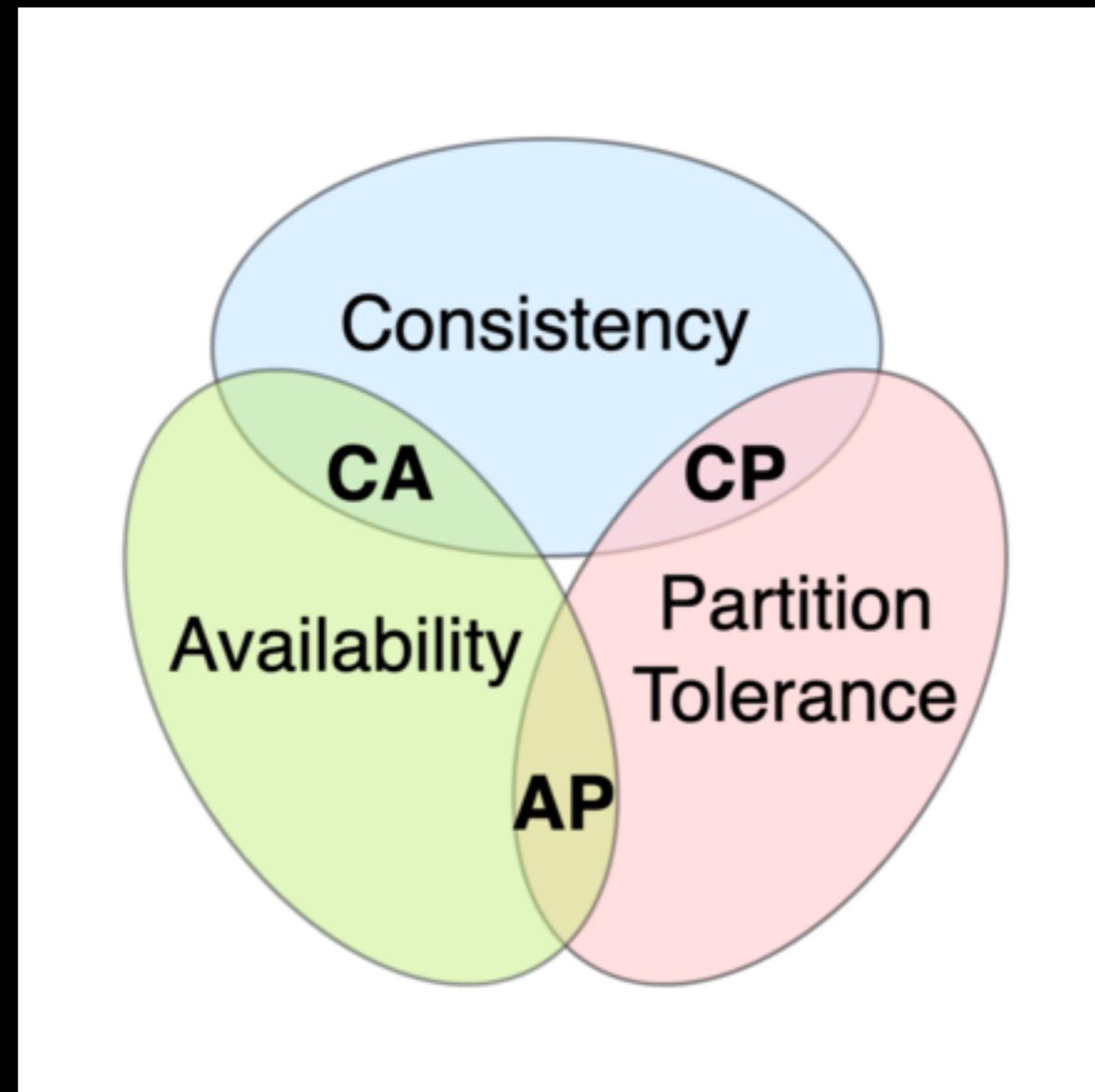
예시 1) “Lambda에 RDS 물렸더니 Max Connection 에러나요”

- Lambda가 아니라 그 어떤 플랫폼이라도,
“사용자 요청에 따라 필요할때만 Application을 올려서 실행하는 런타임”
이라면, 사용자 요청이 DB Max Connection 보다 많이 생기면 터짐
- 애초에 “트래픽이 늘어난다”라고 했을땐, 주어진 선택지는 두개뿐임.
 1. 트래픽이 몰리면 그 앞단 (Reverse Proxy나 / CDN등)에서
트래픽을 차단하고, 유저한테 “트래픽 몰려서 못쓴다”라고 하던지
 2. 트래픽이 몰리면 그 트래픽을 커버 할수있도록
Backend Application을 스케일링 하던지
- “고객이 몰렸는데, Application은 문제없이 스케일링 됬으나
DB가 스케일링 안되서 실패”

VS

“고객이 몰렸는데, Application이 스케일링 안되서
DB까지 가기도 전에 실패”
- “고객이 몰려도 문제없이 고객들에게 서비스를 한다”
 - 라는 목표에는 뭐가 더 가깝나...?

예시 2) “Lambda에서는 local memory 에 저장하면 안되나요?
Session Storage는 못쓰나요?”



예시 2) “Lambda에서는 local memory 에 저장하면 안되나요? Session Storage는 못쓰나요?”

- Local Memory 쓴다
 - = 그 고객은 반드시 그 container나 instance에 붙어야 한다
 - = Stateful한 Application이다
 - = Scaling 이 불가능하다.
- 예를들어, 컨테이너 하나가 100 Request per sec 을 지원하는데
 - 유저 10명이 각각 10 RPS로, 하나의 컨테이너를 사용하고 있었음
 - 그런데 중간에 갑자기 유저 1명이 20 RPS로 사용량 증가
 - ?? 그럼 그 유저 1명을 빼서 다른 컨테이너로...?
그 컨테이너 local memory에 state가 있으면...?
- “Stateless가 강제되서 local memory를 못쓰는 대신, scalability를 보장”
vs
“Stateful 하게 local memory를 아무때나 쓸수 있는 대신,
local memory 사용량 관리해야하고 scalability를 보장못함

3.3. Marketing-Driven 기술결정

- 지금 당신이 쓰는 그 아키텍처, 그 언어, 그 프레임워크
왜 그걸 선택하셨나요?
- 우리 팀의 기술 숙련도, 서버비 효율성, 인건비 효율성,
새로운 기능 개발과 배포 속도,
이런게 종합적으로 고려됬나요?
- 지금 그 서비스 데이터 블락체인에 올리는거,
정말로 “기술적인” 기술결정인가요?
- 해커뉴스나 트위터에서 힙하다고 해서 쓰는게 아닌거 맞나요?

3~4년 전엔 확실히 “기술문제”:

- 실제로 Serverless에 대한 best practice나 가이드 찾기가 쉽지 않았음. 해외에서도 마찬가지
- Language, Performance 등에서 제약이 많았음
- Serverless로 뭔가를 만드는것의 Risk는 명확한 반면, Serverless로 뭔가를 만들었을때의 이득은 경험해본 사람이 적음.
- 심지어 “Public Cloud” 조차 경험 안해본 사람도 부지기수...

이제는 기술 문제가 아니라 인간/조직/기술결정의 문제

- AWS 뿐만 아니라 모든 Public Cloud가 Serverless / Fully Managed / Cloud Native 방향으로 움직이고 있음.
- 당신이 평범한 (DB + Rest API + Background Job) 요구사항의 서비스를 새로 만들어야한다면, 이제는 “Serverless로 만들 이유” 는 명확하고 길지만, “Serverless로 만들지 않을 이유” 는 사실 몇가지 안됨..

결론:
이러면 겁먹지 말고
Serverless로 해보세요.

- 요구사항이 비교적한 평범한 Modern backend application
그러니까 DB + Rest API + Background Job이다
- 적은 인원으로, 운영 (모니터링 / 스케일링) 등에 시간을 전혀 쓰지 않고 빠르게
정말 “기능 개발” 만 집중하고 싶다
- 팀원들이 새로운 기술을 배우거나 적용하는것을 즐기고, 거기서 동기부여가 된다.
- EC2에 쓰는 돈을 아끼고 싶다.

예를들어,

- 새로운 서비스를 새로 만드려고 하는 스타트업
- 이미 존재하는 Legacy Application이 있지만,
완전히 독립적으로 분리할수 있는 Microservice를 만들고 싶고,
기존 개발 환경이 너무 너무 싫을때.

Q&A



mirror

일상을 정리하는
마법같은 브라우저

www.mirror.work