



Pivotal.

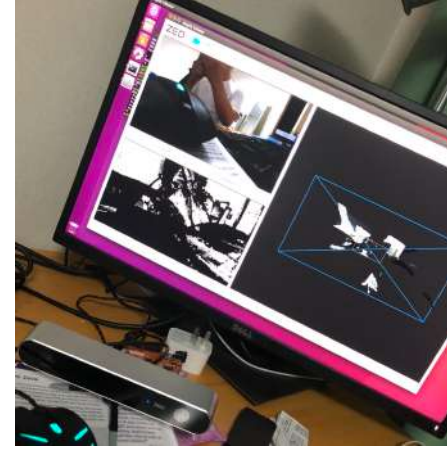
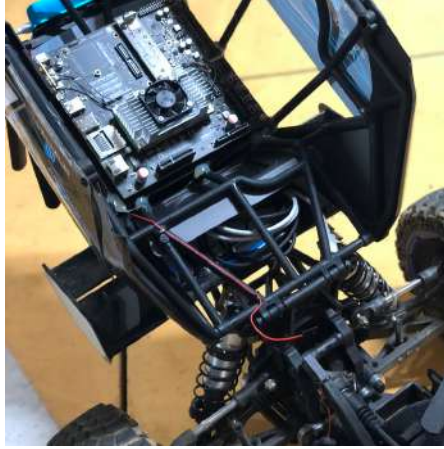
# Netflix MSA Platform

---

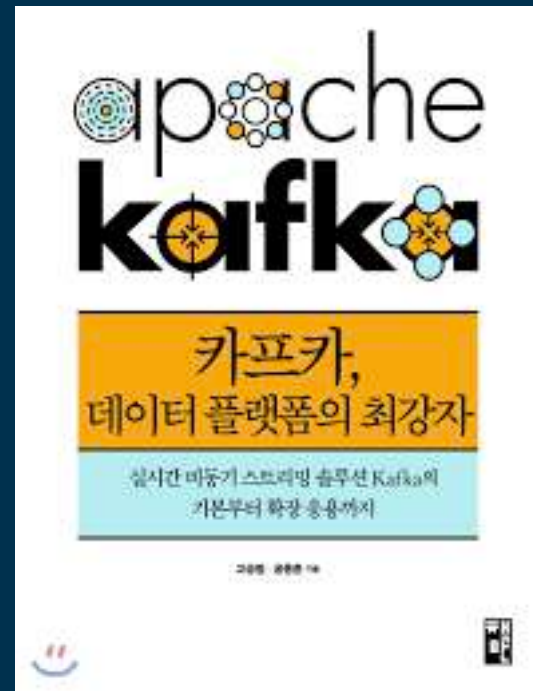
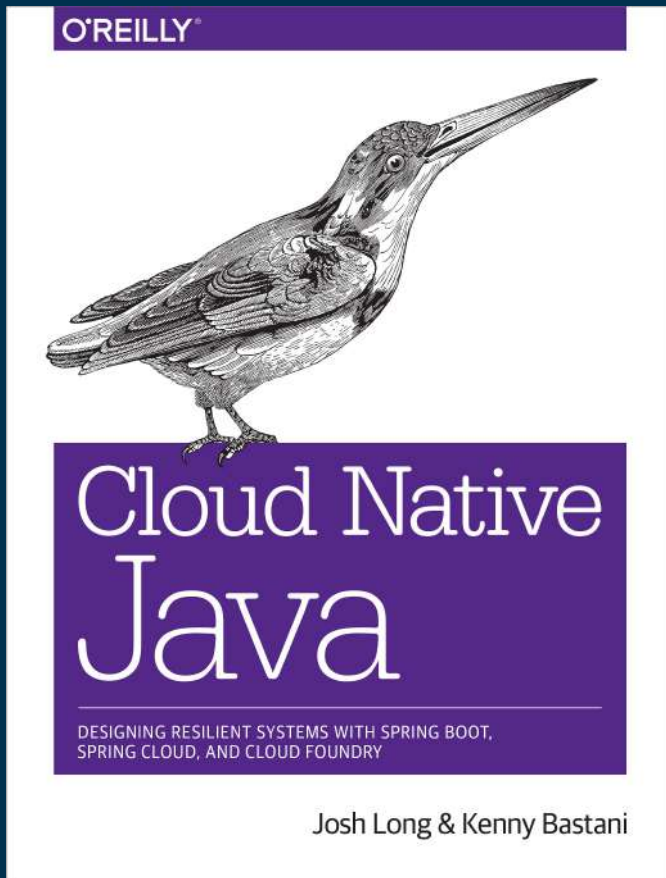
Younjin Jeong  
[yjeong@pivotal.io](mailto:yjeong@pivotal.io)

April, 2018

# AutoRally Platform Instructions



[https://github.com/AutoRally/autorally\\_platform\\_instructions](https://github.com/AutoRally/autorally_platform_instructions)





# Agenda

- Netflix 소개
- MSA 고려사항
- 조직 변화
- MSA 플랫폼 기술
- 도입





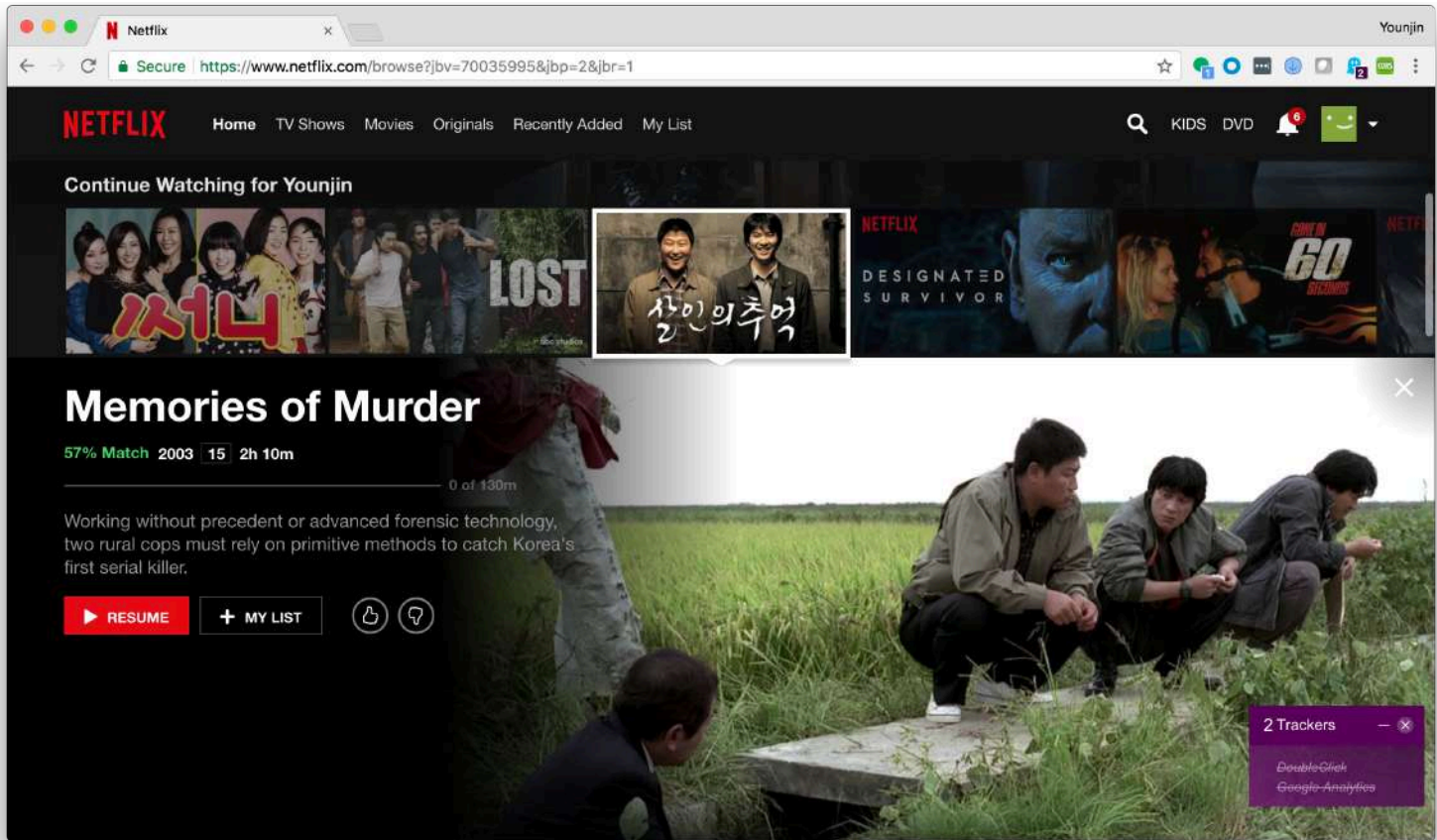
Overview

---

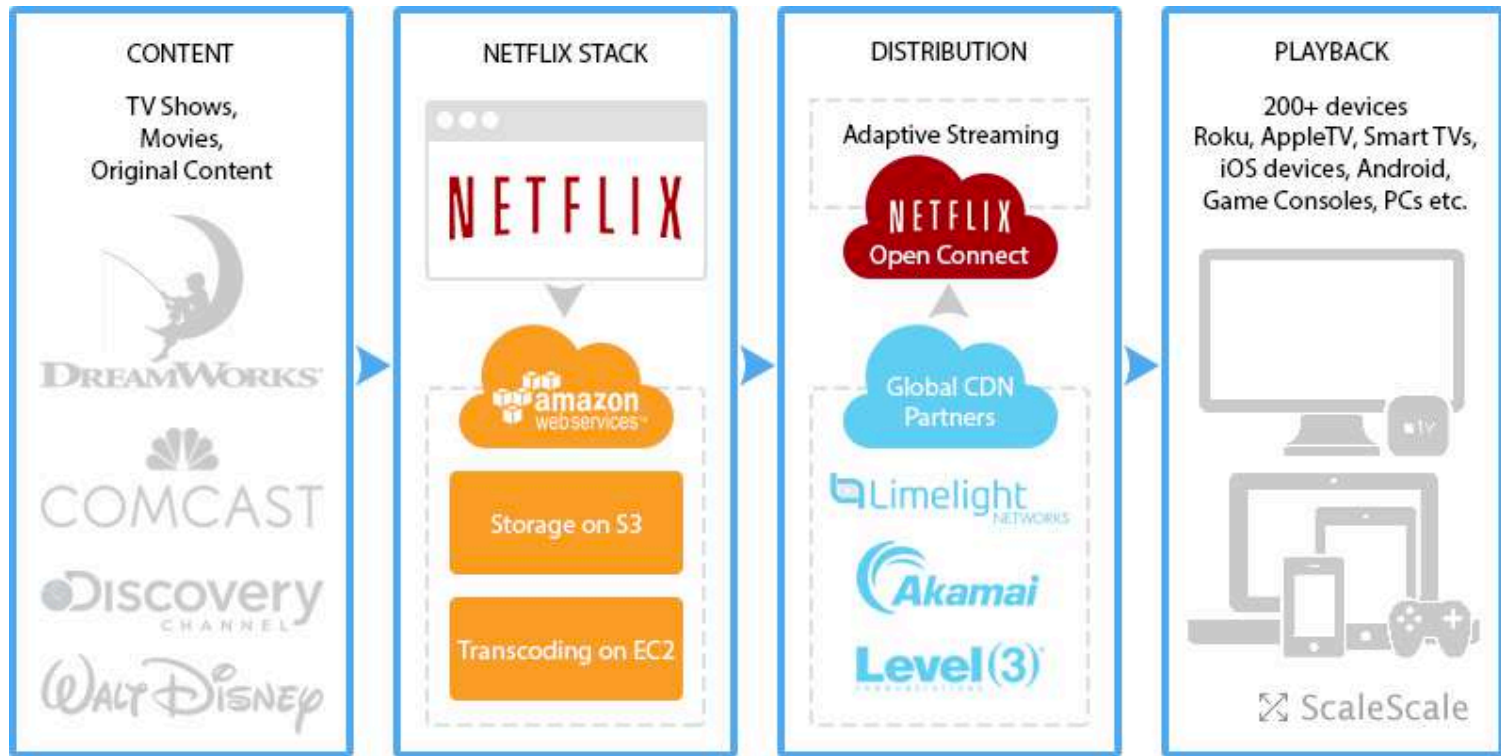
# Introducing Netflix



2018년 현재 넷플릭스는 1억 1천 8백만명의 가입자에 일 1억시간 이상의 비디오 스트리밍을 제공하는 회사입니다.



2016년 초에 한국에서도 서비스를 시작했으며, 넷플릭스 오리지널을 비롯해 수많은 콘텐츠를 서비스 하고 있습니다.



비디오 콘텐츠를 다루는 서비스의 특성상 콘텐츠의 계약에 따른 물량의 확보, 비교적 큰 파일의 저장, 높은 재생 품질, 다양한 단말 기기의 지원 등을 다양한 국가에 다운타임 없이 서비스 하는 것이 넷플릭스 엔지니어링의 목표입니다.

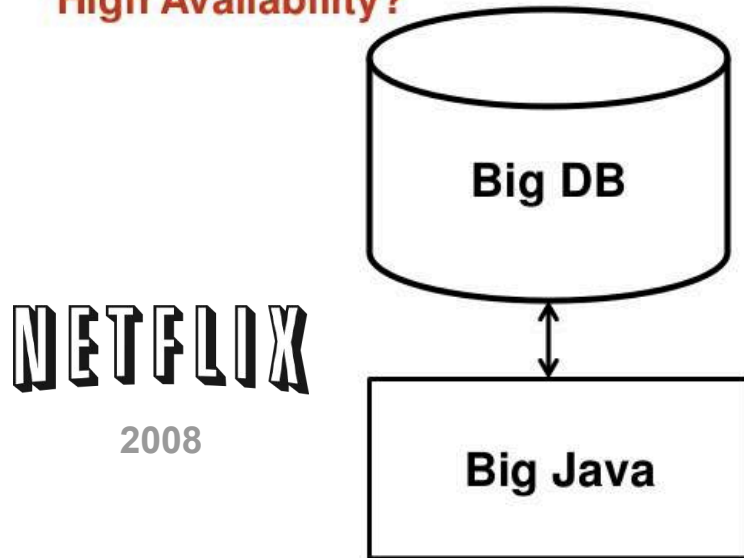


# 넷플릭스 2008 – Big Java Big Oracle

## August 2008 – DB Corruption

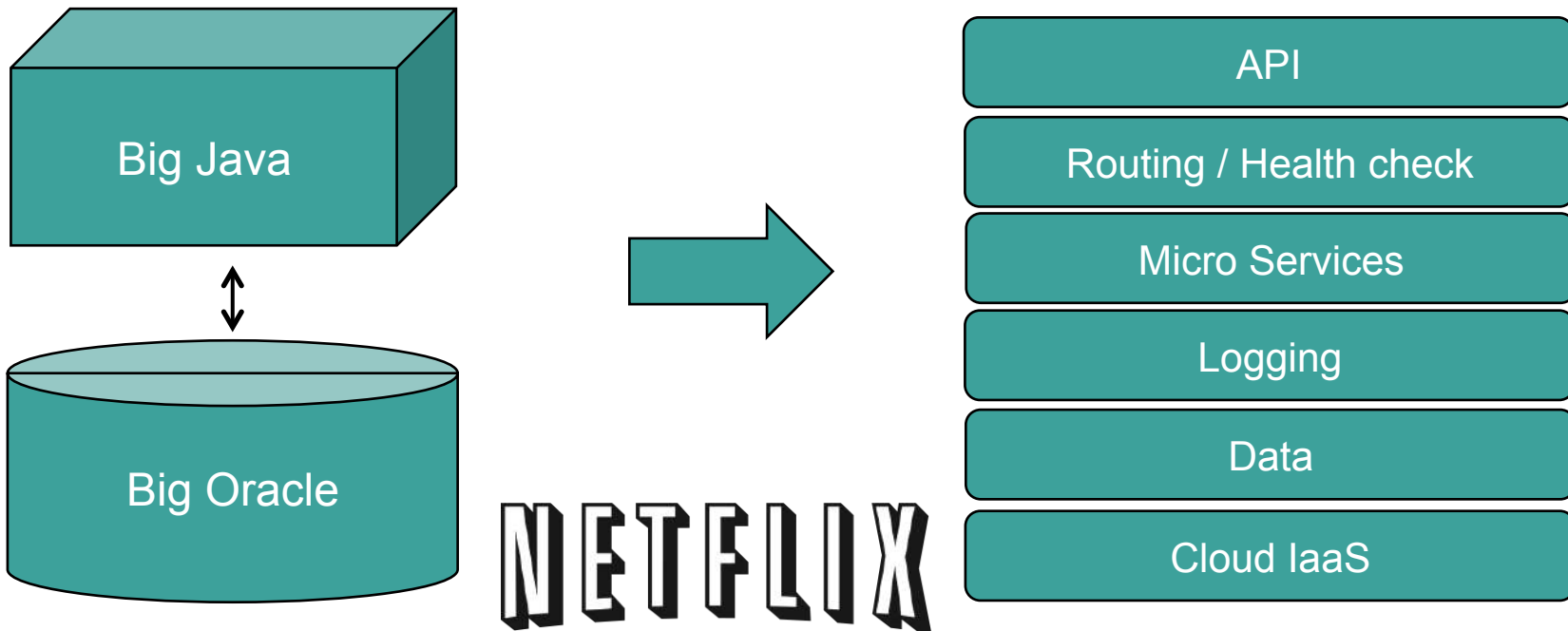


## High Availability?

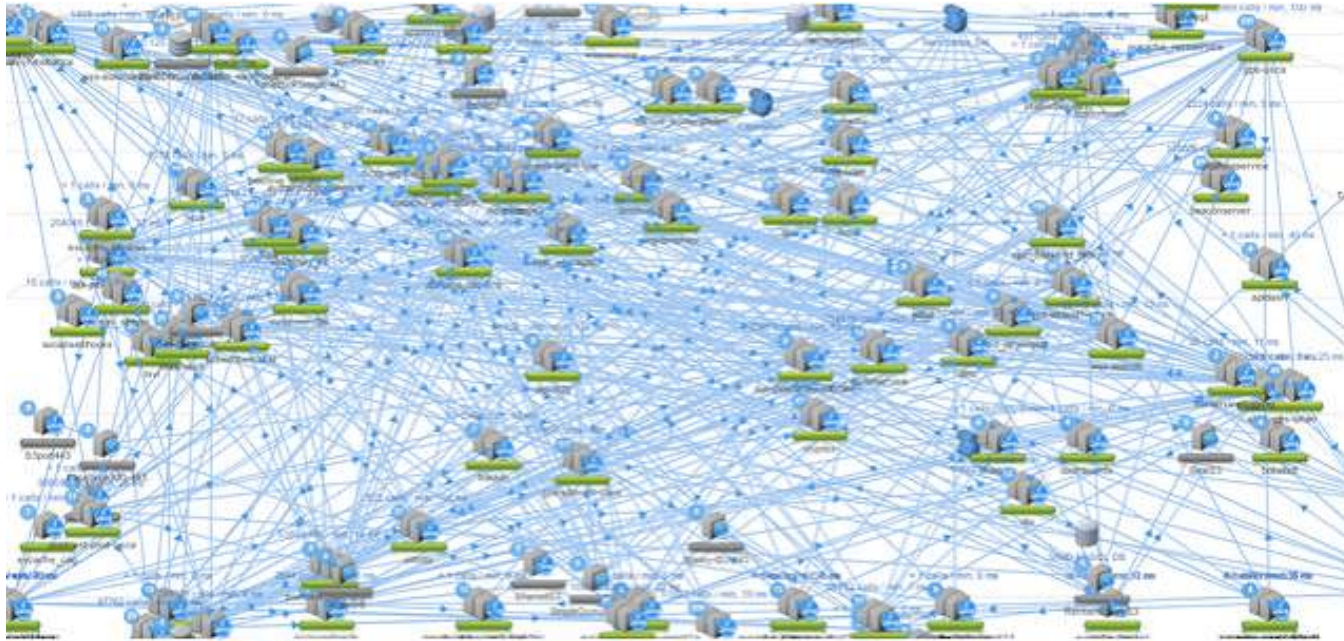


<http://www.slideshare.net/KevinMcEntee/netflix-incloudsmarch8-2011forwiki>

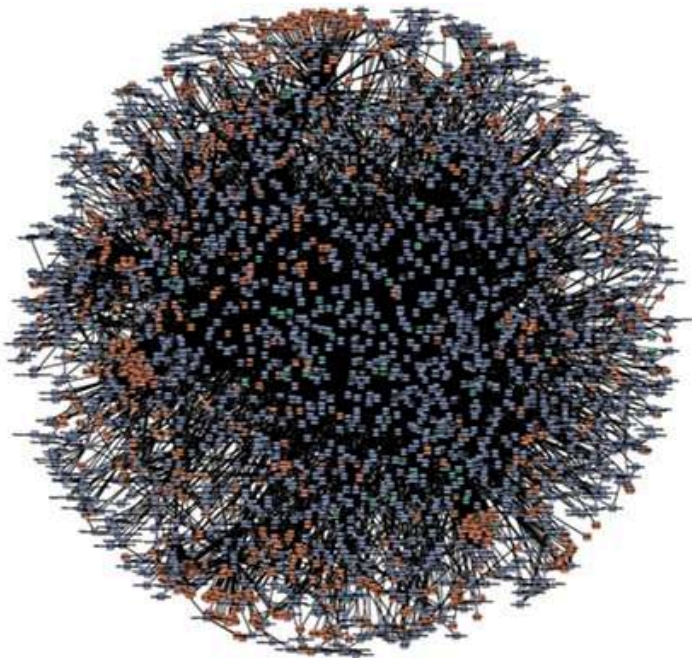
# Cloud Native – 7 years (2008 – 2015)



# 현재 - Micro Services Architecture



<http://www.slideshare.net/stonse/microservices-at-netflix>



amazon.com<sup>®</sup>



NETFLIX

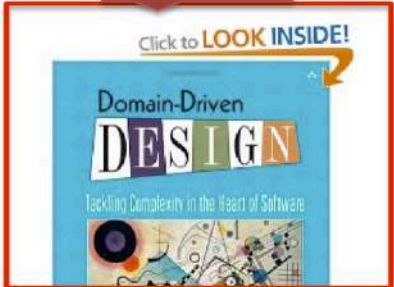
이 회사는 아마존닷컴과 더불어 다수의 작은 서비스로 거대한 하나의 애플리케이션을 구성하는 방식을 취하고 있습니다. 보통, “마이크로 서비스”라고 불리는 방식입니다.



Page Context:  
{ type: Book, id: ISBN-10 0-321-83457-7 }

ImageService

Domain-Driven Design and BookService



Domain-driven Design: Tackling Complexity in the Heart of Software [Hardcover]  
Eric Evans (Author)

★★★★☆ (12 customer reviews) Like (15) ReviewService

RRP: £46.99  
Price: £30.09 & this PriceService in the UK with Super Saver Delivery. See details and  
You Save: £16.90 (36%)

In stock. InventoryService  
Dispatched from and sold by Amazon.co.uk. Gift-wrap available.

Want guaranteed delivery by Friday, March 23? Order it in the next 21 hours and 9 minutes, and choose Express de

Customers Who Bought This Item Also Bought

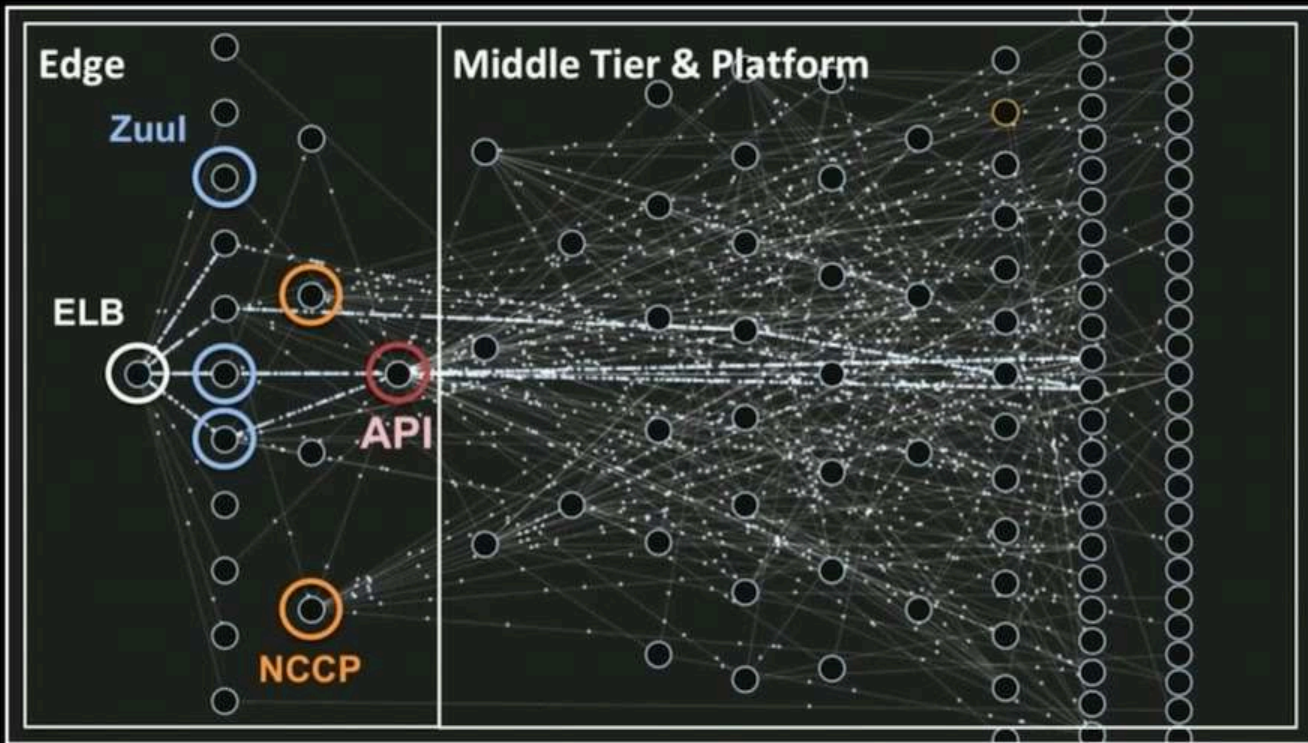
OthersAlsoBoughtService

<p>ImageService</p>	<p>BookService</p>				
<p>Patterns of Enterprise Application Architecture... by Martin Fowler ★★★★☆ (14) £33.59</p>	<p>Growing Object-Oriented Software, Guided by Test... by S... ★★★★☆ (14) £18.35</p>	<p>Clean Code: A Handbook of Agile Software Craf... by Robert C. Martin ★★★★☆ (27) £17.00</p>	<p>Enterprise Integration Patterns: Designing, Build... by Gregor Hohpe ★★★★☆ (16) £29.61</p>	<p>Continuous Delivery: Reliable Software Releases Thr... by Jez Humble ★★★★☆ (7) £19.97</p>	<p>Refactoring: Improving the Design of Existing Co... by Martin Fowler ★★★★☆ (24) £29.61</p>

Product details  
Hardcover: 560 pages  
Publisher: Addison Wesley; 1 edition (20 Aug 2003)

BookService





Filmed at  
**QCon** San Francisco 2016

Brought to you by  
**InfoQ**  
urcu

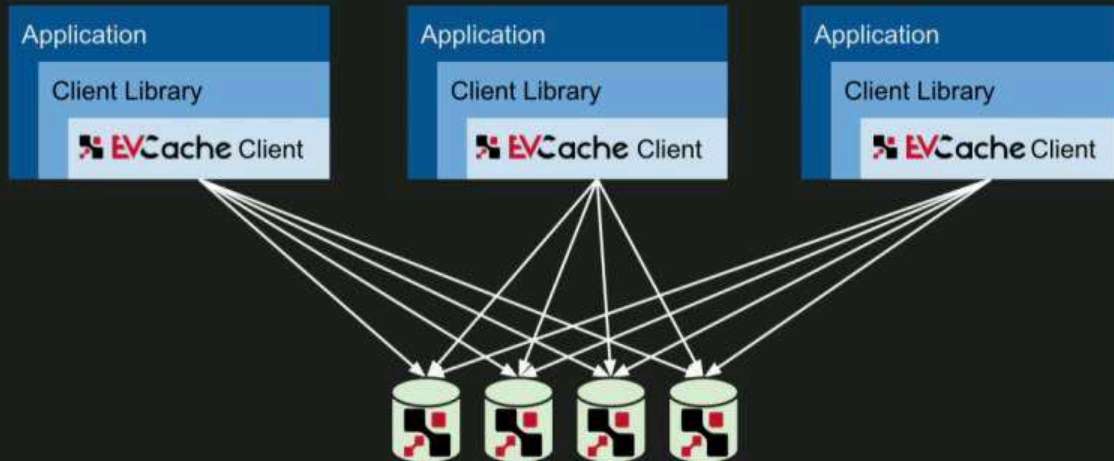
그리고 이 경험을 다양한 영상과 테크 블로그등으로 공유하고 있습니다.  
이들 중 유튜브에서 “혼돈의 제왕”으로 검색하시면 조쉬 에반스님의 강연에서 많은 아이디어를 얻을 수 있을 것입니다.



Sept 15-17, 2016  
thestrangeloop.com

## Use Case: Transient Data Store

Time →



넷플릭스의 구조를 설명하는 강연 중 특히 인상적인 것 중 하나는 스캇 맨스필드님이 강연한 “숨겨진 마이크로 서비스”입니다. EVCache로 알려진 넷플릭스의 캐시와 플랫폼 사용의 장점을 살펴볼 수 있습니다.

# NETFLIX | OSS

또한 넷플릭스가 공개하고 있는 다양한 오픈소스들은 직접 가져다 사용하기엔 여러가지 어려움이 있지만 왜 만들었는지, 어떻게 만들었는지 두가지 측면에서 살펴본다면 MSA 구성에 도움이 될 것입니다.

The screenshot shows a web browser displaying the Netflix Tech Blog. The page features the Medium logo and the Netflix Tech Blog header. The main content is an article titled "Automated Canary Analysis at Netflix with Kayenta" by Michael Graff and Chris Sanden, published on April 10. The article includes a detailed system metrics dashboard for a canary deployment. The dashboard compares a baseline configuration (myapp-baseline) with a canary configuration (myapp-canary) across various metrics.

Metric Name	Result
CPU	Pass
Error Requests	Pass
Incoming TCP Connections	Pass
Latency 50th	High
Latency 90th	High
Latency 99th	Pass
Load Average	Pass
Outgoing TCP Connections	Pass
Successful Requests	High
Threads	Pass
Unhealthy	Pass

The chart shows system metrics over time, comparing the baseline (orange) and canary (blue) configurations. The metrics include CPU, Error Requests, Incoming TCP Connections, Latency (50th, 90th, 99th), Load Average, Outgoing TCP Connections, Successful Requests, and Threads. The canary configuration shows higher latency and successful requests compared to the baseline.

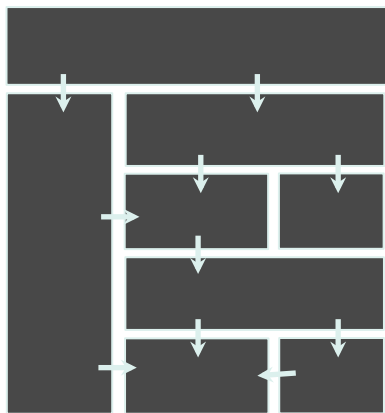
현재는 미디어로 이사한 넷플릭스의 테크 블로그 역시 다양한 기술의 배움에 도움이 될 것입니다. 슬라이드를 만들고 있는 이 시점에도 Kayenta 라는 새로운 도구를 설명하고 있네요.



Dive Deep

# 넷플릭스 MSA 전환 효과 및 고려사항



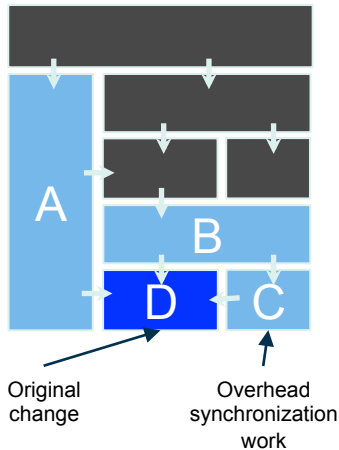
# 모놀리틱 구조의 한계



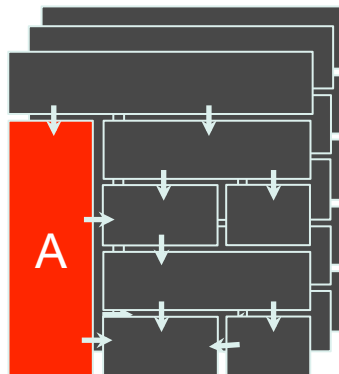
 모듈  
 의존성 관계



D 모듈의 변경



A의 Heat 가 높을 때..



## 모놀리틱 구조의 한계

D모듈 변경 시 A,B,C의 추가 테스트 필요

D모듈의 작은 변화도 전체 App의 배포를 요구

D모듈만 테스트하고는 릴리즈 할 수 없음

A,B,C,D모듈이 동시에 변경할 경우 엄청난 부가적인 작업이 필요

결론적으로 매우 느린 릴리즈 사이클이 형성

A 모듈이 집중적으로 많이 사용되는 경우에도 어쩔 수 없이 전체 App을 Scale 해야 함 (메모리, Disk 등의 비효율성)

A 모듈이 문제가 생기면 장애가 나머지 모듈에도 영향을 줄 수 밖에 없음

## 모놀리틱 구조의 한계

D모듈 변경 시 A,B,C의 추가 테스트 필요

D모듈의 작은 변화도 전체 App의 배포를 요구

D모듈만 테스트하고는 릴리즈 할 수 없음

A,B,C,D모듈이 동시에 변경할 경우 엄청난 부가적인 작업이 필요

결론적으로 매우 느린 릴리즈 사이클이 형성

A 모듈이 집중적으로 많이 사용되는 경우에도 어쩔 수 없이 전체 App을 Scale 해야 함 (메모리, Disk 등의 비효율성)

A 모듈이 문제가 생기면 장애가 나머지 모듈에도 영향을 줄 수 밖에 없음

## MSA 도입 시 개선 사항

더 신속한 업데이트

여러 팀이 독립적으로 릴리즈 사이클을 결정하여,  
요구사항을 더 빠르게 반영  
(API 기반의 *Contract*)

독립적인 확장성

부하가 발생하는 모듈만 독립적으로 Scale-Out/In 할 수 있는 구조

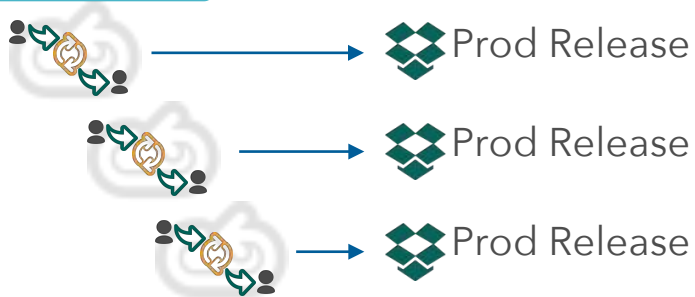
장애의 전파없는 가용성

장애를 하나의 모듈에만 고립시켜, 전체적인 서비스에는 영향을 끼치지 않음  
(더 빠른 *Root Cause* 분석)

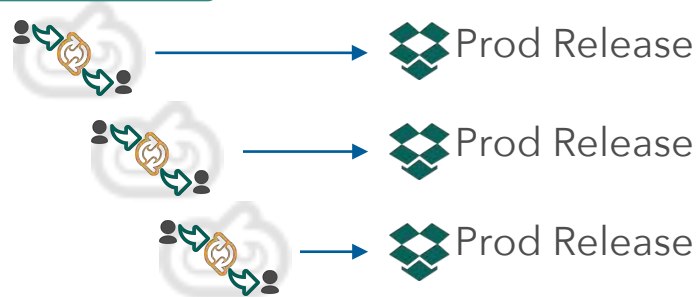
성과  
유저 경험

애플리케이션의 사이즈가 축소됨에 따라 Start 시간과 Scale 시간이 단축

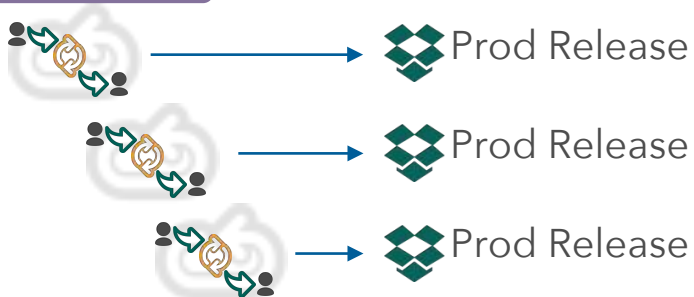
### CATALOG



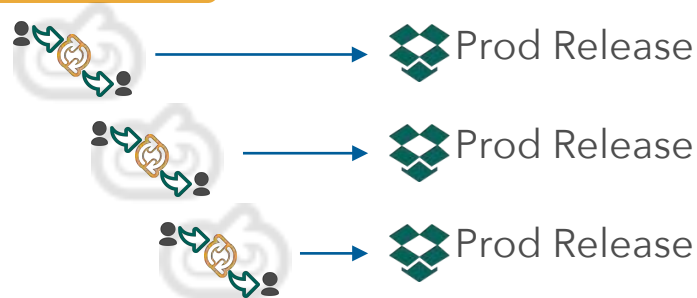
### INVENTORY



### REVIEWS



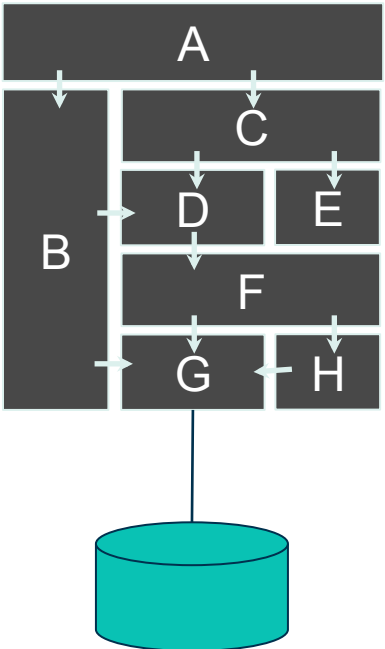
### SHIPPING



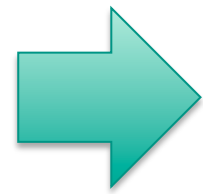
독립적인 확장성

장애의 전파없는  
가용성

# Interface



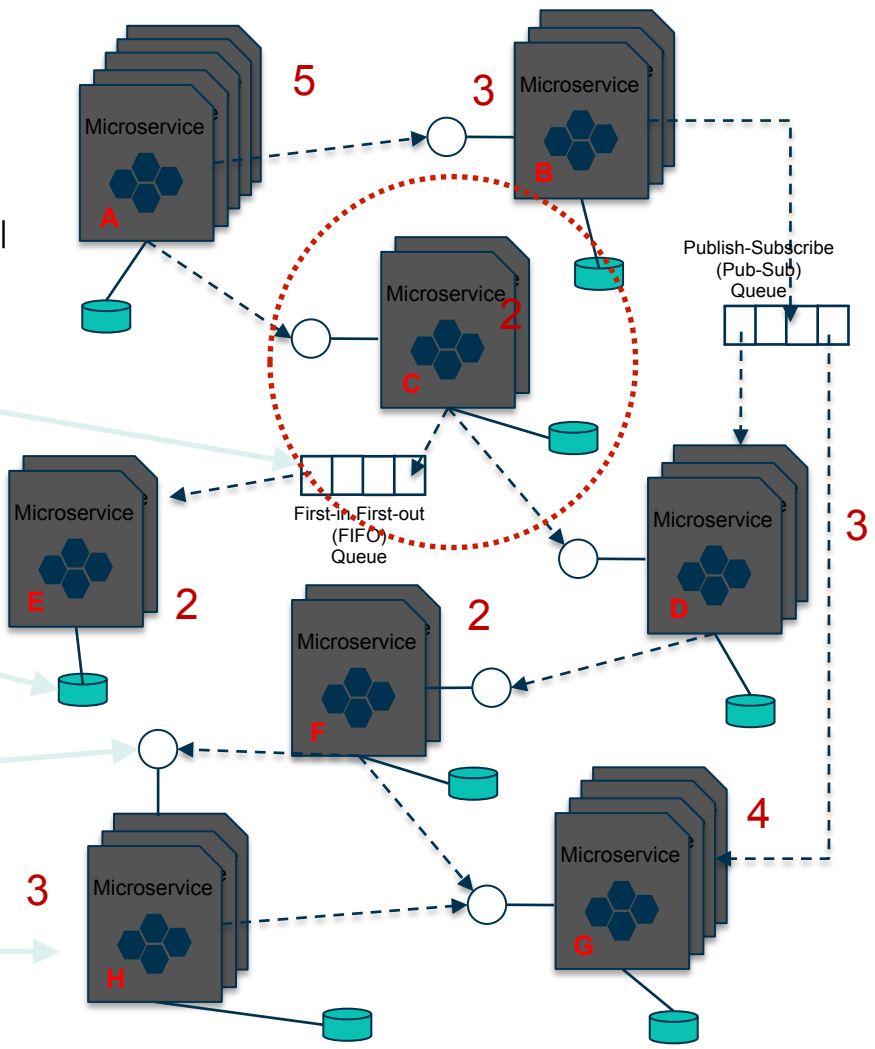
Queues, Pub-Sub 등의 비동기성 큐를 통해 독립적으로 Auto-Scale을 할 수 있는 환경



각각의 State 존재

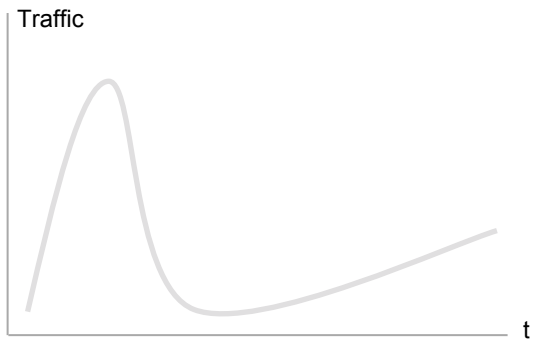
약속된 API로 통신

각각의 마이크로 서비스는 독립적으로 Scale-Out/In

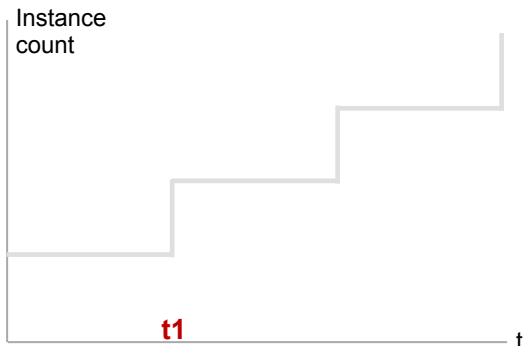


# 성과와 유저경험

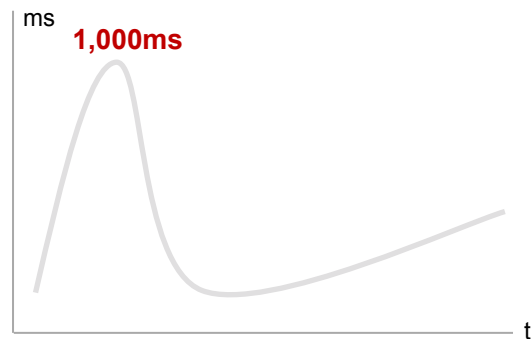
### Traffic Pattern



### Compute Resources

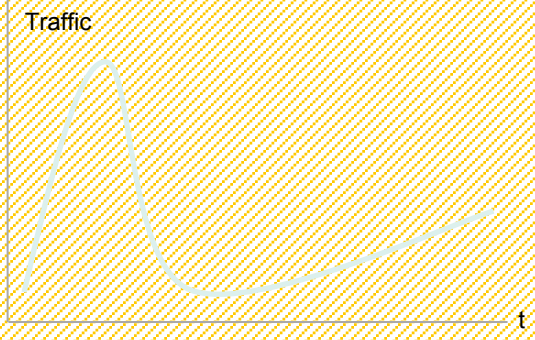


### Response Time

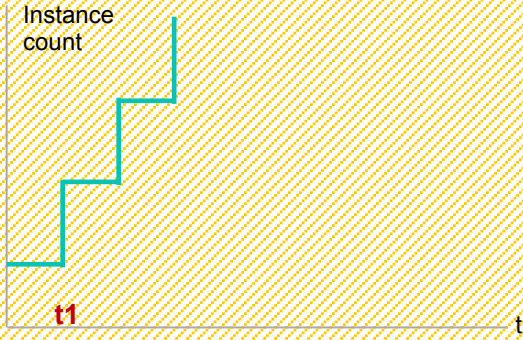


Long Start-up Time

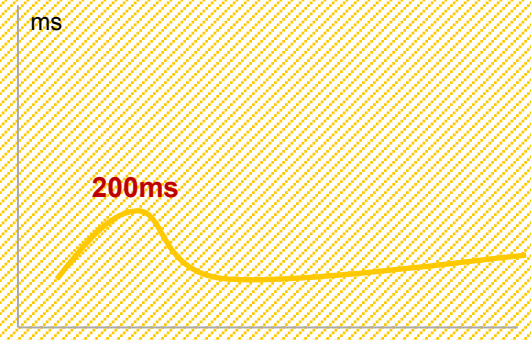
### Traffic Pattern



### Compute Resources



### Response Time



Short Start-up Time



## 넷플릭스가 고려했던 포인트...

마이크로 서비스에 적합한 **문화**

마이크로 서비스에 적합한 **조직 구성**

마이크로 서비스 동작상태에 대한 **모니터링** 필요

**장애 고립**을 제어할 수 있는 역량 필요

**API 기반 서비스 연동**이 필요

클라우드 기반 **서비스 생명주기 관리** 필요

**팬아웃(Fan-Out) 현상**에 대한 제어가 필요

**다양한 데이터 서비스**를 지원하는 도구 제공이 필요

수많은 마이크로 서비스의 **설정 관리** 필요

서비스 문제 발생시 **자동 복구**가 필요

서비스에서 발생하는 **로그의 취합 저장 분석** 필요

**장애를 사전에 테스트**할 수 있는 환경이 필요

## 넷플릭스의 해결책들...

조직적 변화

셀프 서비스로의 패러다임 변화

자유와 책임 중심의 개발 문화

Configuration Server

Service Discovery

Circuit Breaker

API Gateway

Distributed Tracing

Zero Downtime Delivery

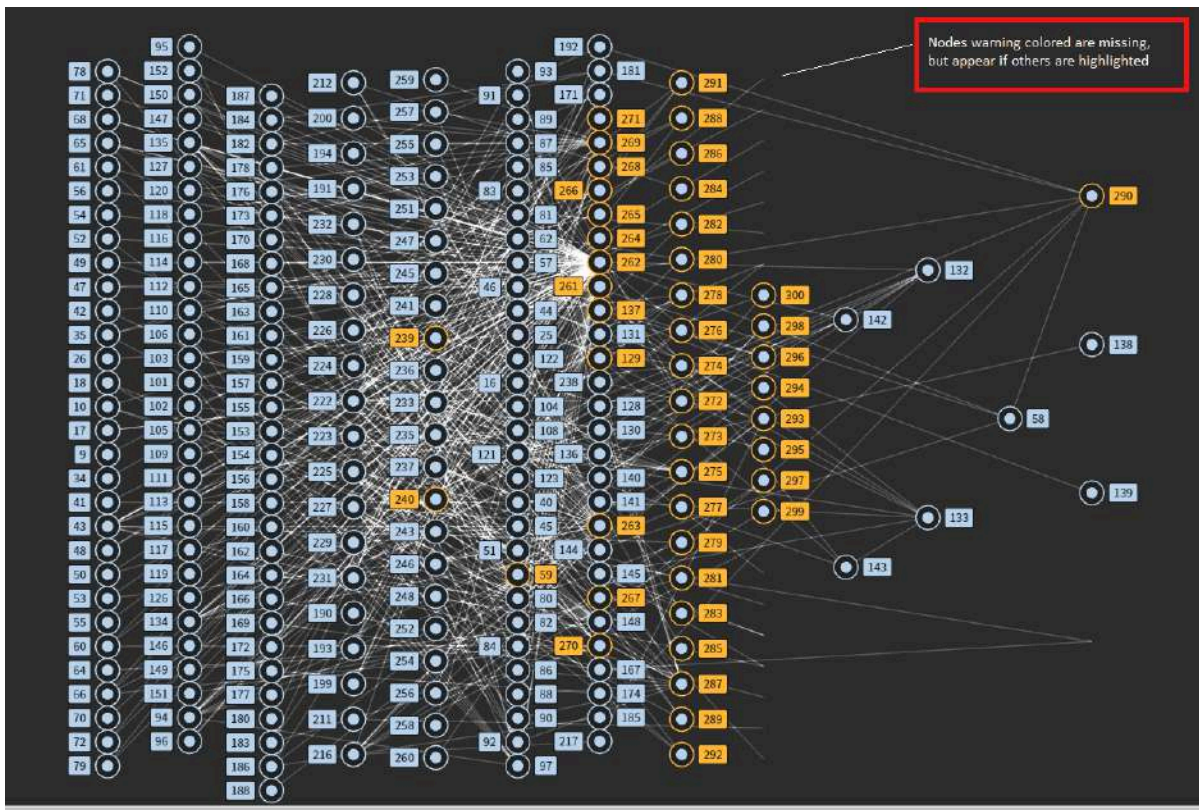
Fault Injection Test

Chaos Engineering

Persistence Cache Layer

Sidecar / Library

기술적 변화



As you've seen from some examples, Netflix micro system looks very complicated, but since you understand what their platform does, it's a combination of a Micro service and Platform tools.

### Persistence Systems



### Platform Libraries



### Infrastructure Services



Netflix, is keep developing various tools to solve “their problems” based on this eco system.  
However, it’s not easy to build up this kind of eco system.

Then How?

# 넷플릭스의 조직 변화

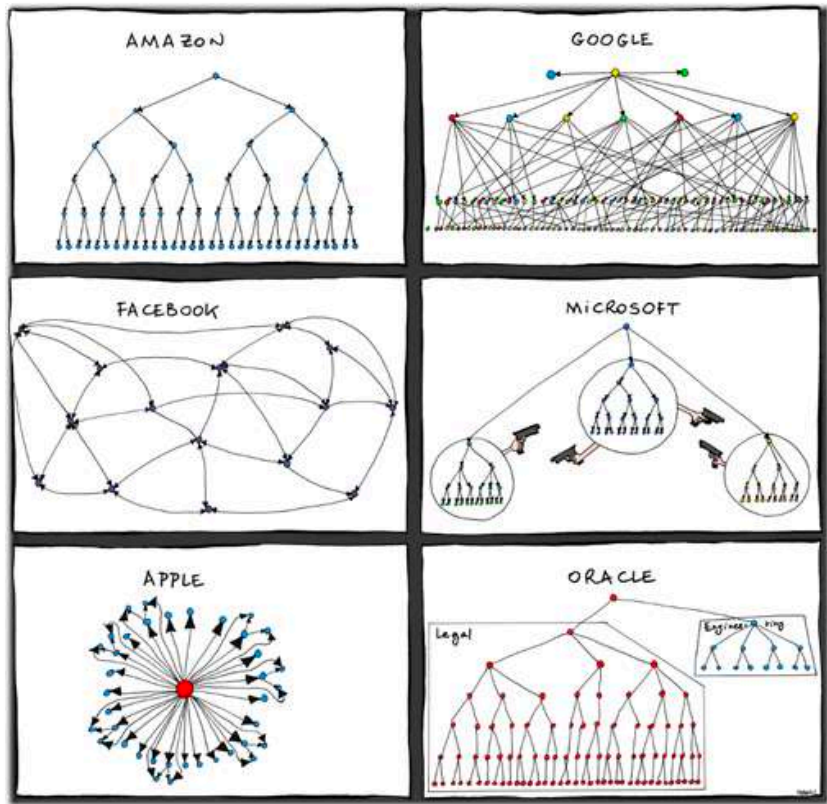
## 넷플릭스가 고려했던 포인트...

마이크로 서비스에 적합한 문화
마이크로 서비스에 적합한 조직 구성
마이크로 서비스 동작상태에 대한 모니터링 필요
장애 고립을 제어할 수 있는 역량 필요
API 기반 서비스 연동이 필요
클라우드 기반 서비스 생명주기 관리 필요
팬아웃(Fan-Out) 현상에 대한 제어가 필요
다양한 데이터 서비스를 지원하는 도구 제공이 필요
수많은 마이크로 서비스의 설정 관리 필요
서비스 문제 발생시 자동 복구가 필요
서비스에서 발생하는 로그의 취합 저장 분석 필요
장애를 사전에 테스트할 수 있는 환경이 필요

## 넷플릭스의 해결책들..

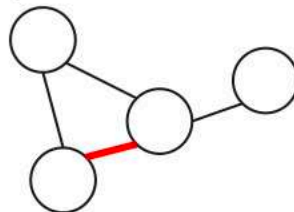
조직적 변화	자유와 책임 중심의 개발 문화
	셀프 서비스로의 패러다임 변화
기술적 변화	Configuration Server - 설정
	Service Discovery - 서비스 등록/탐색
	Circuit Breaker - 장애 고립
	API Gateway - API 기반 지원
	Distributed Tracing - 모니터링
	Zero Downtime Delivery - 배포
	Fault Injection - 인위적 장애 주입
	Chaos Engineering
	Persistence Cache Layer
	Sidecar / Library



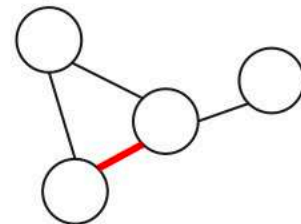


## conway's law

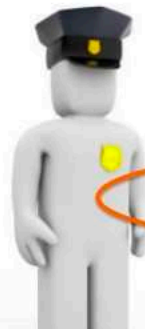
system:



organization:



## The Laws which Rule over Us



**Moore's Law** Computing power doubles every 18-24 months

**Metcalf's Law** Network becomes more useful the more devices are connected to it

**Conway's Law** Organizations design systems which copy the organization

**Brook's Law** Adding more people to a late project makes it later



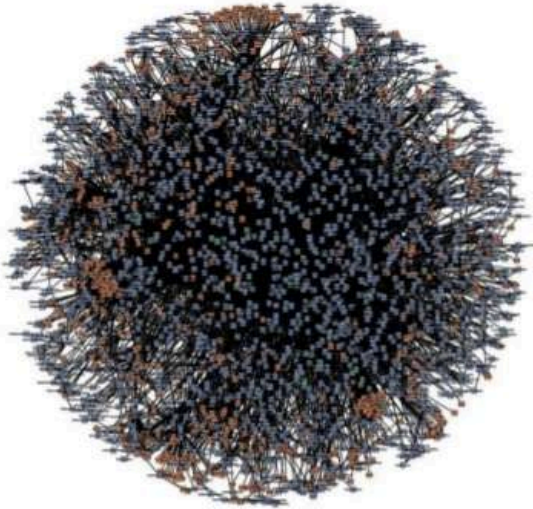
NETFLIX PRINCIPAL

---

# You Build It, You Run It You Support It

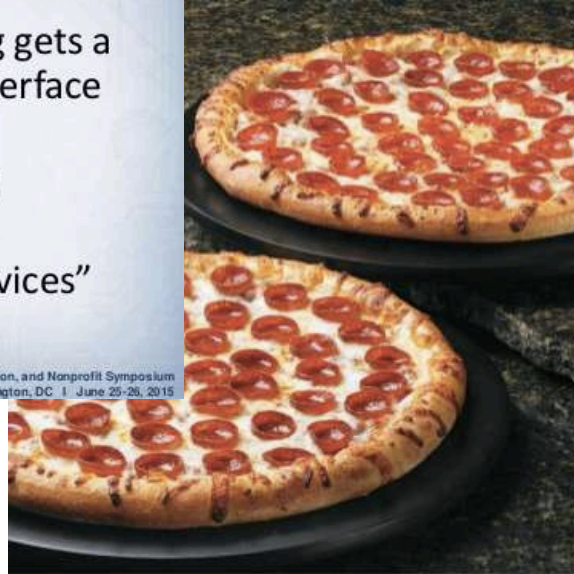
Netflix culture made all this possible.

## 2 pizza “DevOps” team



- Service-Oriented Architecture (SOA)
- Everything gets a service interface
- Primitives
- “Microservices”

AWS Government, Education, and Nonprofit Symposium  
Washington, DC | June 25-26, 2015



- Decentralized
- Two-pizza teams
- Agility, autonomy, accountability, and ownership
- “DevOps”

AWS Government, Education, and Nonprofit Symposium  
Washington, DC | June 25-26, 2015

In 2012..

## Implications for IT Operations

- Cloud is run by developer organization
  - Product group's "IT department" is the AWS API and PaaS
  - CorpIT handles billing and some security functions

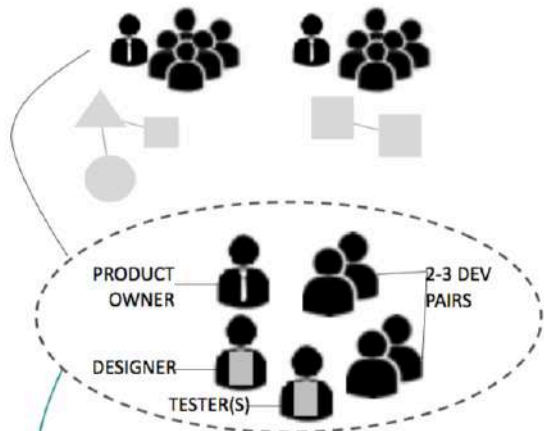
Cloud capacity is 10x bigger than Datacenter

- Datacenter oriented IT didn't scale up as we grew
- We moved a few people out of IT to do DevOps for our PaaS

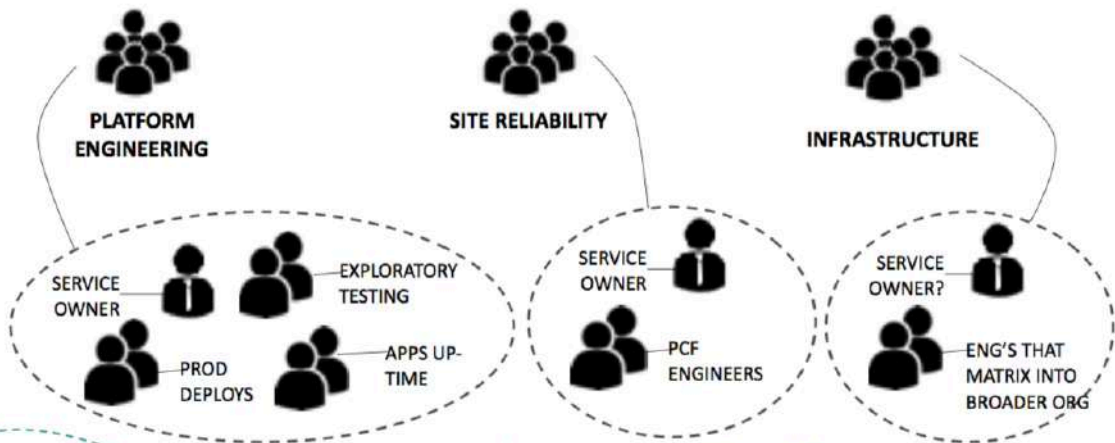
- Traditional IT Roles and Silos are going away
  - We don't have SA, DBA, Storage, Network admins for cloud
  - Developers deploy and "run what they wrote" in production

The Netflix logo, consisting of the word "NETFLIX" in white, uppercase letters on a red rectangular background.

## BUSINESS CAPABILITY TEAMS



## AGILE OPERATIONS SHARED SERVICES



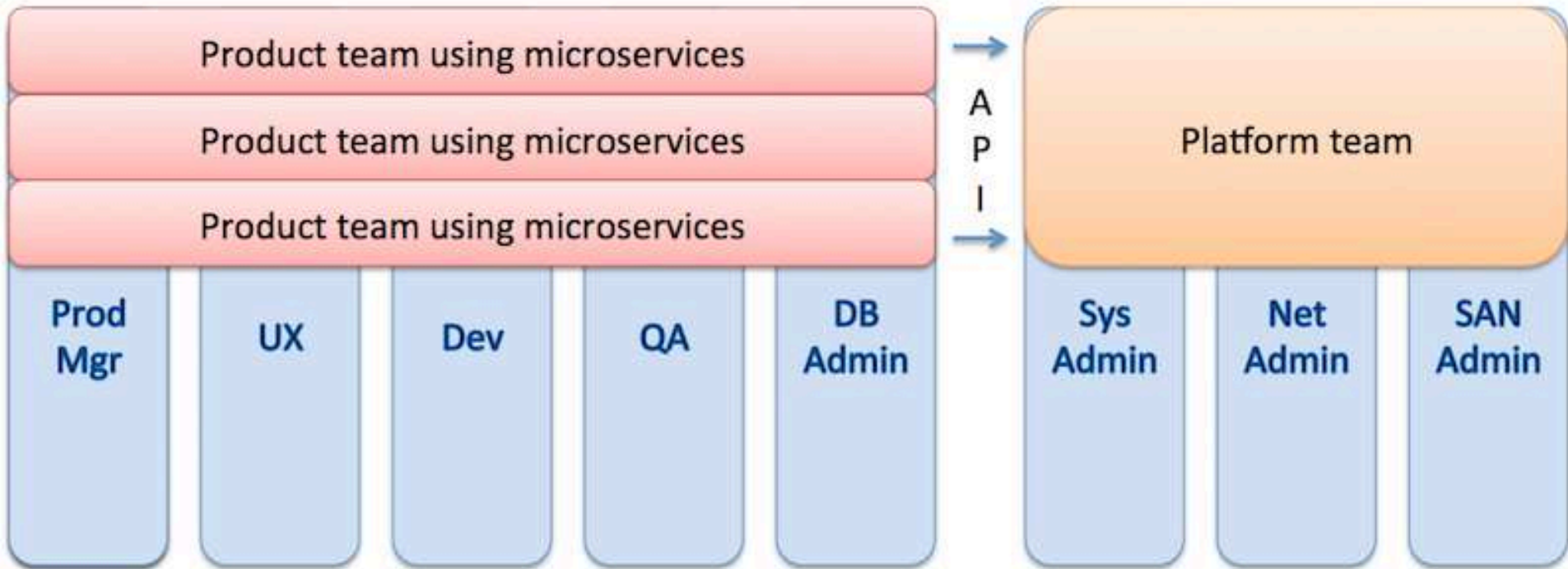
**CONTINUOUS DELIVERY PIPELINE (TESTING, BUILD AND RELEASE AUTOMATION)**

NON-PROD PAAS AND RELATED STACKS / SERVICES

PRODUCTION PAAS AND UNDERLYING TECHNOLOGY STACKS / SERVICES

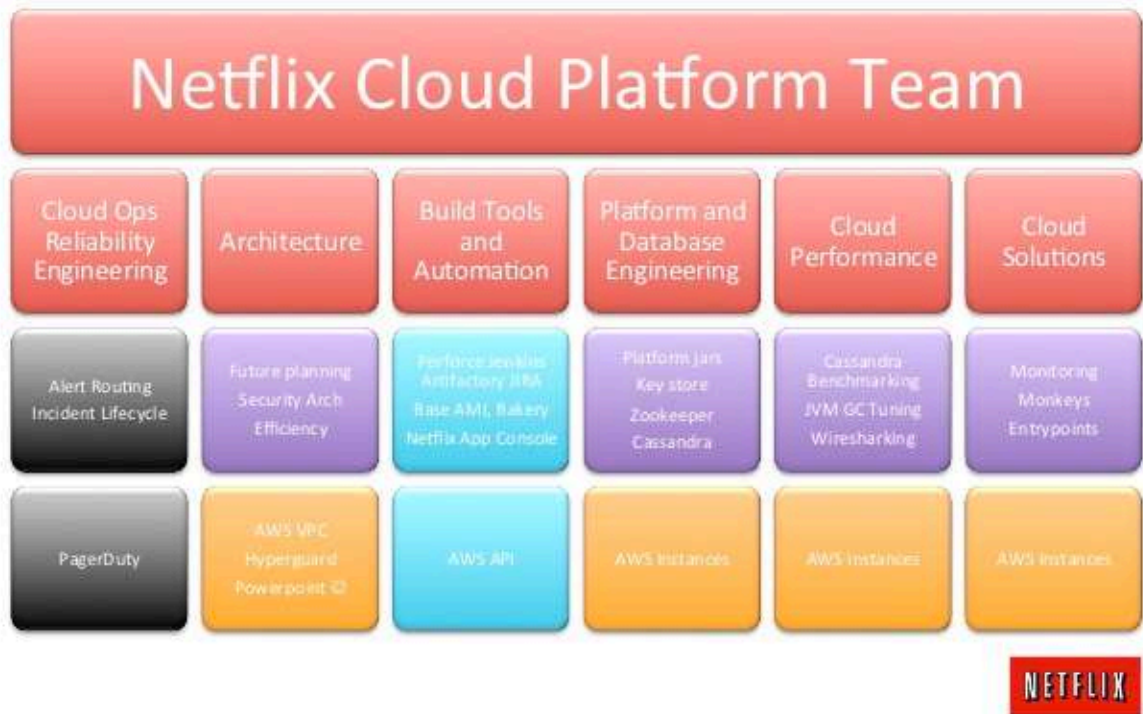






# Netflix PaaS Organization

Developer Org Reporting into Product Development, not ITops





```
build.gradle
1 apply plugin: 'nebula'
2 apply plugin: 'war'
3 apply plugin: 'netflix.ospackage-tomcat'
4 apply plugin: 'nebula.dependency-lock'
5
6 dependencies {
7     compile 'netflix:base-server:latest.release'
8     compile 'javax.ws.rs:jsr311-api:1.1.1'
9     provided 'javax.servlet:javax.servlet-api:3.1.0'
10
11     testCompile 'junit:junit-dep:4.10'
12     testCompile 'org.mockito:mockito-all:1.9.5'
13 }
14
15 ospackage {
16     requires('apache-tomcat8')
17 }
```

넷플릭스는 Nebula 라는 Gradle 플러그인 기반 빌드 도구를 만들어서 사용하고 있습니다. 이는 애플리케이션에 필요한 패키지 뿐만 아니라 플랫폼과 연동할때 필요한 도구를 손쉽게 개발자가 추가할 수 있도록 합니다.

<https://medium.com/netflix-techblog/how-we-build-code-at-netflix-c5d9bd727f15>

Then what's the most important thing?

---

The key part is “Platform”

That provides common functionality to  
Every single micro-services in “easy way”.

Then How?

**넷플릭스 MSA 기술**

## 넷플릭스가 고려했던 포인트...

마이크로 서비스에 적합한 문화
마이크로 서비스에 적합한 조직 구성
마이크로 서비스 동작상태에 대한 모니터링 필요
장애 고립을 제어할 수 있는 역량 필요
API 기반 서비스 연동이 필요
클라우드 기반 서비스 생명주기 관리 필요
팬아웃(Fan-Out) 현상에 대한 제어가 필요
다양한 데이터 서비스를 지원하는 도구 제공이 필요
수많은 마이크로 서비스의 설정 관리 필요
서비스 문제 발생시 자동 복구가 필요
서비스에서 발생하는 로그의 취합 저장 분석 필요
장애를 사전에 테스트할 수 있는 환경이 필요

## 넷플릭스의 해결책들..

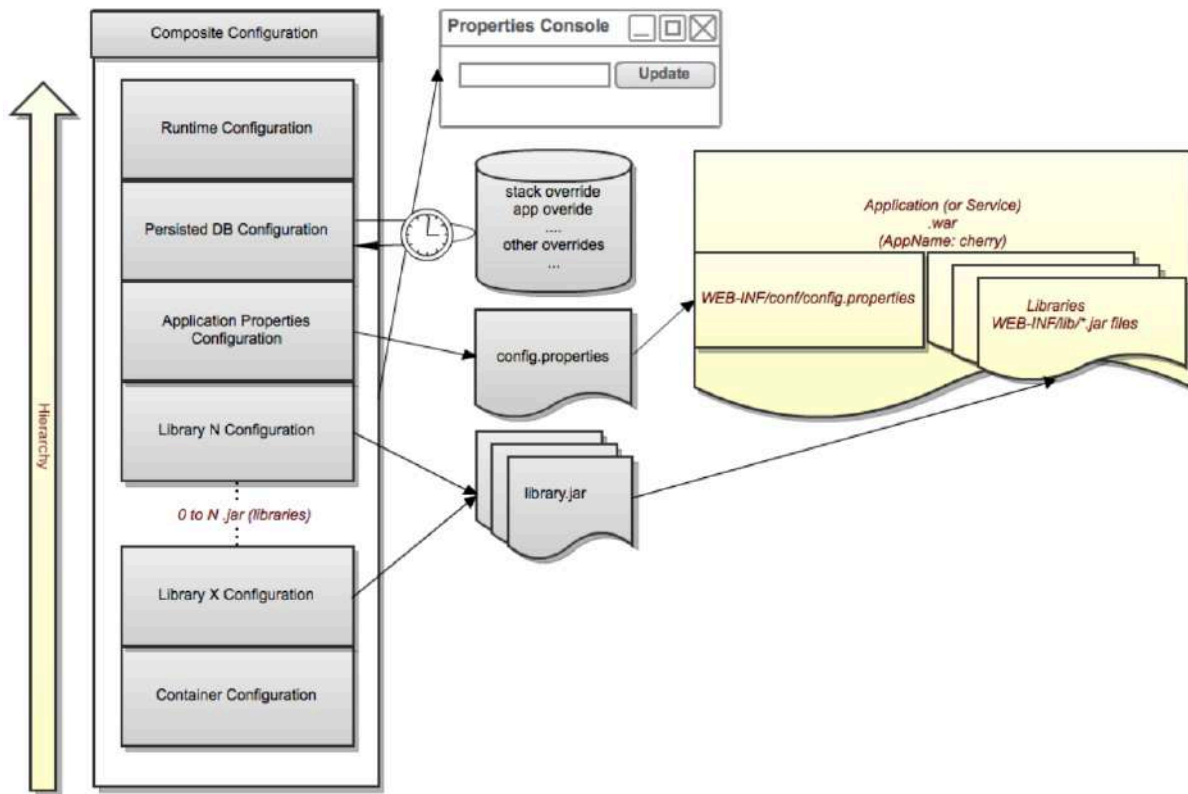
조직적 변화	셀프 서비스로의 패러다임 변화
	자유와 책임 중심의 개발 문화
기술적 변화	1 Configuration Server - 설정
	2 Service Discovery - 서비스 등록/탐색
	3 Circuit Breaker - 장애 고립
	4 API Gateway - API 기반 지원
	5 Distributed Tracing - 모니터링
	6 Zero Downtime Delivery - 배포
	7 Fault Injection - 인위적 장애 주입
	8 Chaos Engineering
	Persistence Cache Layer
	Sidecar / Library

# NETFLIX

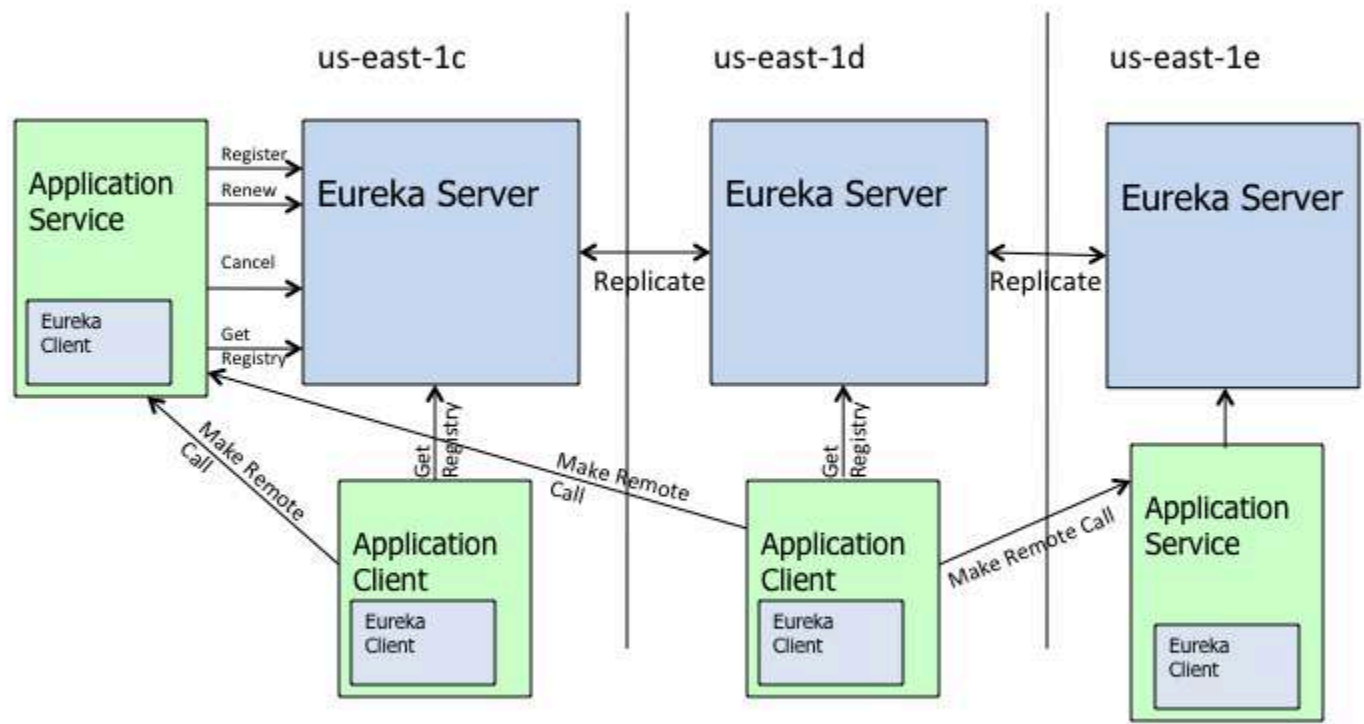
## OSS

- Externalized Config Server
- Service Register & Discovery
- Circuit Breaker
- API Gateway
- Load Balancing with IPC
- Realtime Monitoring
- Zero Downtime Delivery
- Fault Injection Test
- Chaos Engineering
- Etc...

# Netflix Externalize Configuration – Archaius



# Netflix Service Discovery - Eureka



<http://techblog.netflix.com/2012/09/eureka.html>

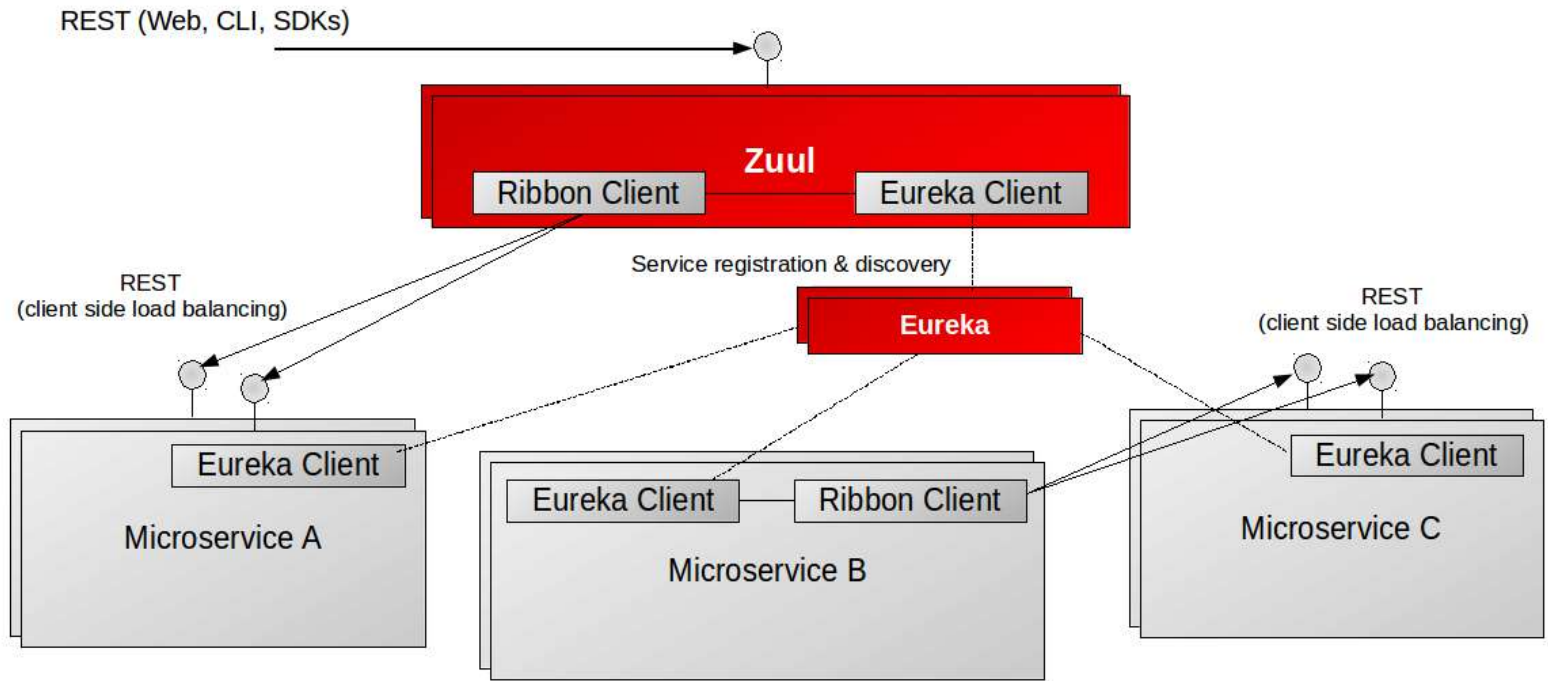


# Netflix Circuit Breaker - Hystrix



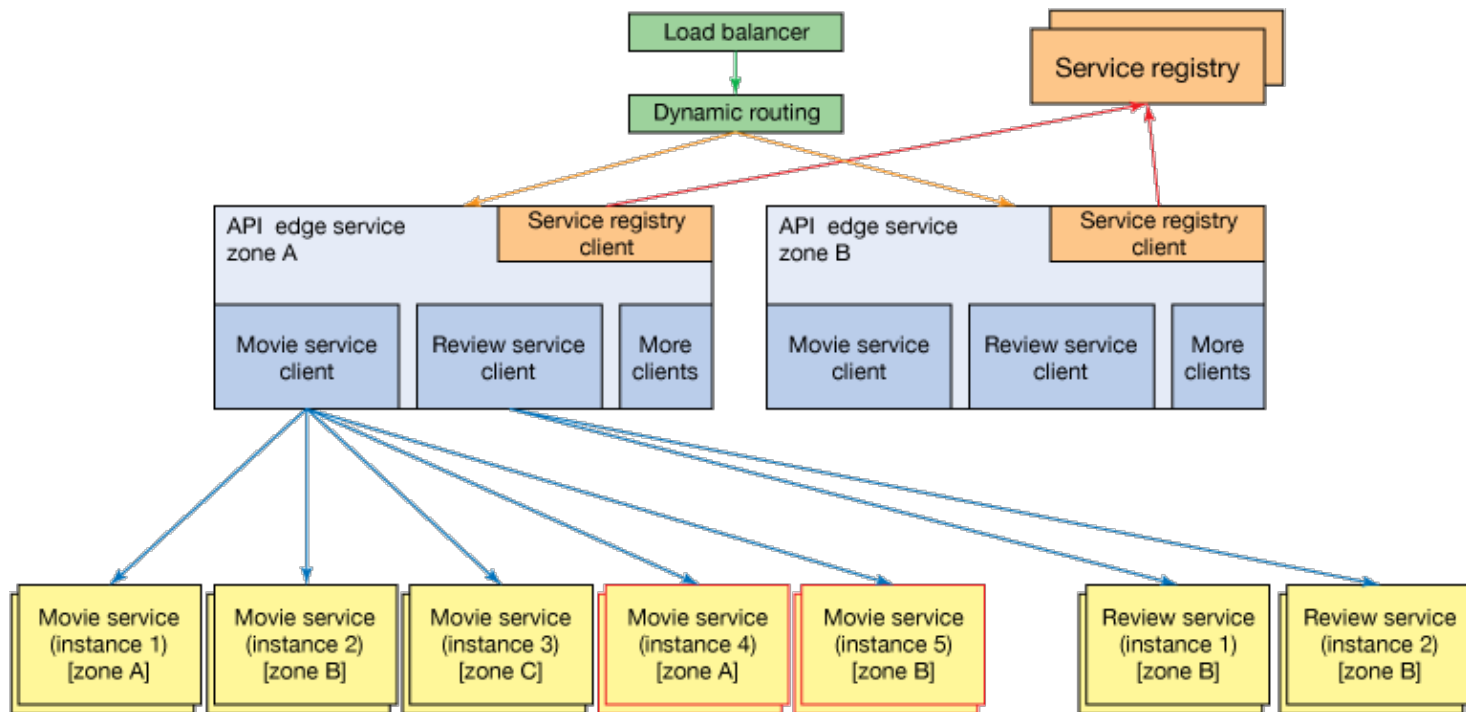
<https://medium.com/netflix-techblog/introducing-hystrix-for-resilience-engineering-13531c1ab362>

# Netflix API Gateway – Zuul



<https://medium.com/netflix-techblog/announcing-zuul-edge-service-in-the-cloud-ab3af5be08ee>

## Netflix Load Balancing with IPC – Ribbon

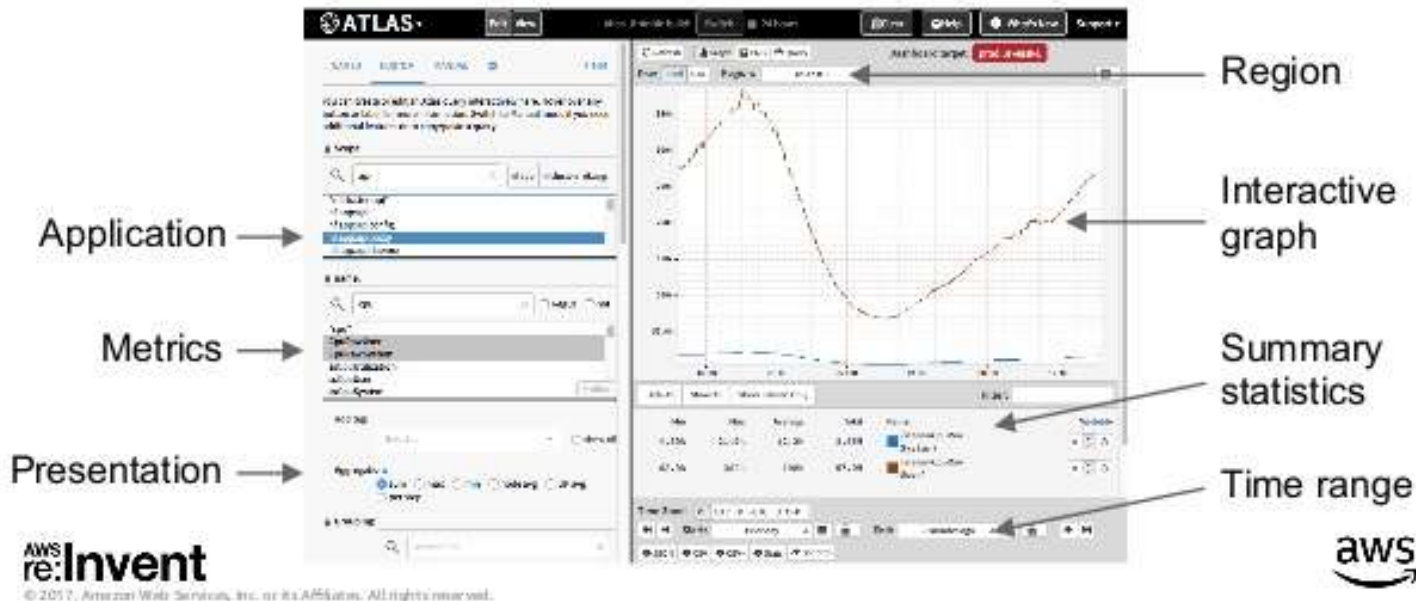


<https://medium.com/netflix-techblog/announcing-ribbon-tying-the-netflix-mid-tier-services-together-a89346910a62>

## Netflix Realtime Monitoring – Atlas

## Netflix Atlas

- Cloud-wide and instance monitoring:



<https://www.slideshare.net/brendangregg/how-netflix-tunes-ec2-instances-for-performance>

# Netflix CI/CD – Spinnaker

The screenshot shows the Spinnaker Deck interface for a pipeline named 'demo'. The pipeline is currently in a 'Manual Judgment: Go/No Go' state. The execution flow is as follows:

- Bake
- Verify Bake
- Integration Tests
- Unit Tests
- Deploy Canary
- Manual Judgment: Go/No Go (Current step)
- Deploy To Prod
- Wait 4 hours
- Cleanup Old Server Groups

The 'Manual Judgment: Go/No Go' step is currently in progress, with a duration of 00:25. The overall pipeline duration is 01:19.

**MANUAL JUDGMENT: GO/NO GO DETAILS**

Step	Started	Duration	Status
Manual Judgment: Go/No Go	2015-11-09 21:42:10 PST	00:25	<span style="color: blue; font-weight: bold;">RUNNING</span>

**MANUAL JUDGMENT: GO/NO GO**

**Manual Judgment** | **Task Status**

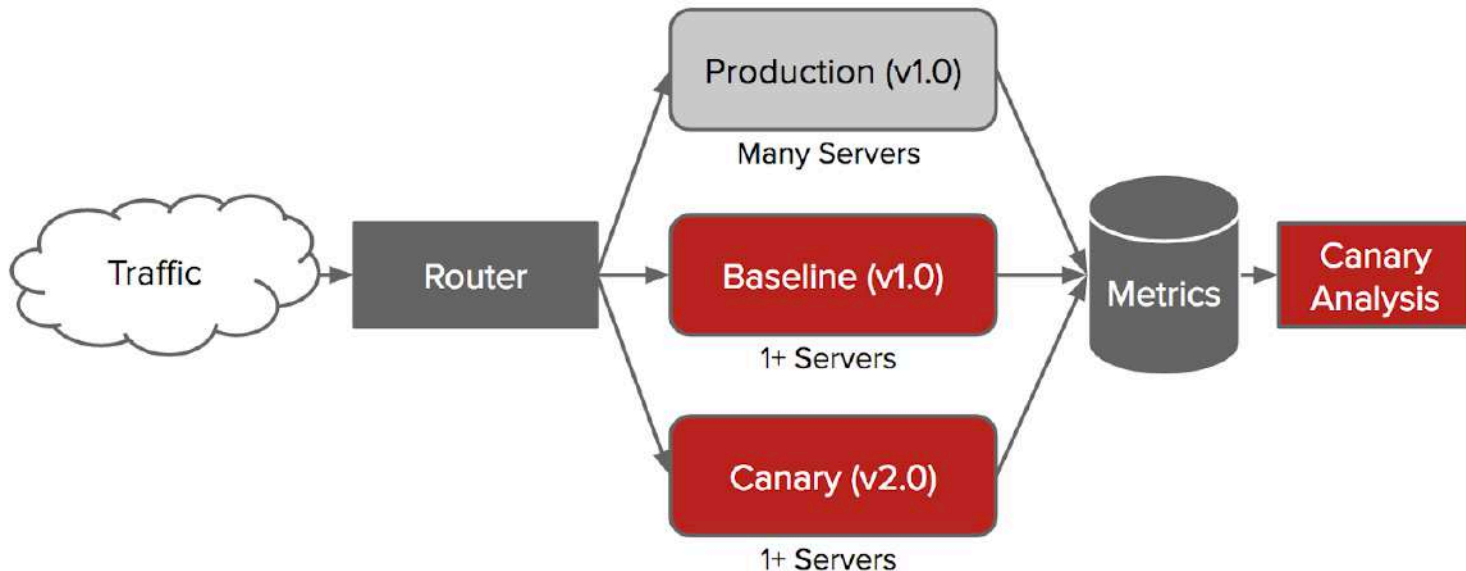
**Instructions** Verify the canary scores are correct and you haven't received any emails or something.

[Continue](#) [Stop](#)

[Source](#) | [Permalink](#)

<https://medium.com/netflix-techblog/global-continuous-delivery-with-spinnaker-2a6896c23ba7>

## Netflix Zero Downtime Delivery – Kayenta



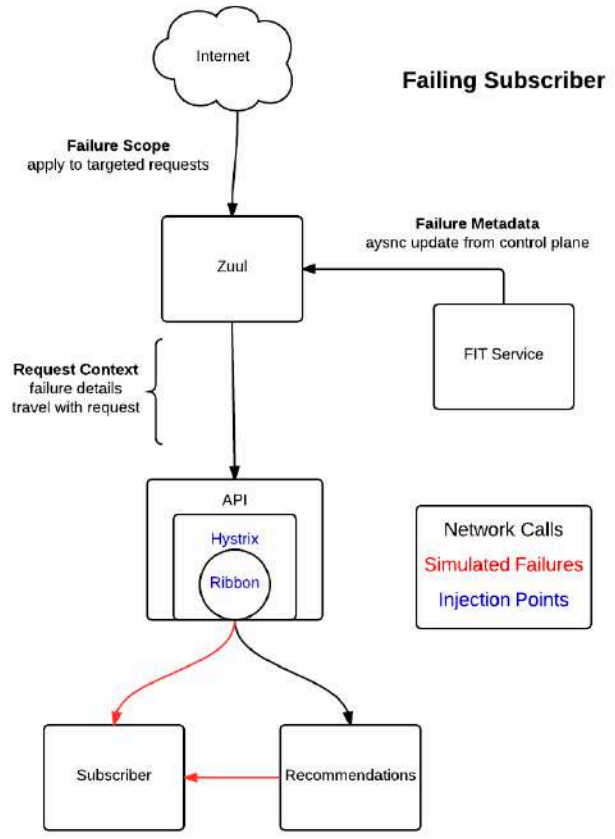
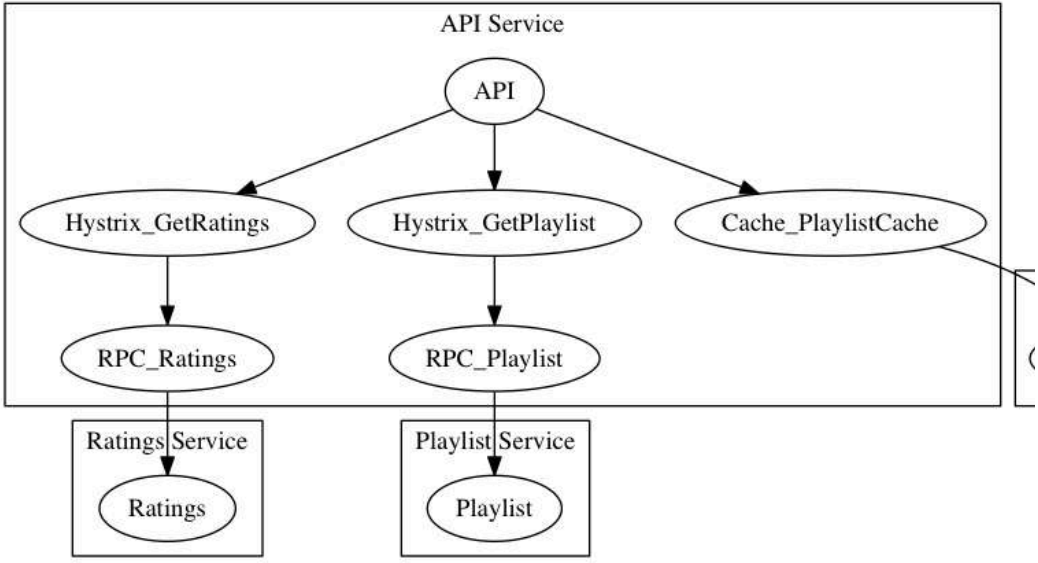
<https://medium.com/netflix-techblog/automated-canary-analysis-at-netflix-with-kayenta-3260bc7acc69>



# Netflix Zero Downtime Delivery – Kayenta



# Netflix Fault Injection Test – FIT And Automate it



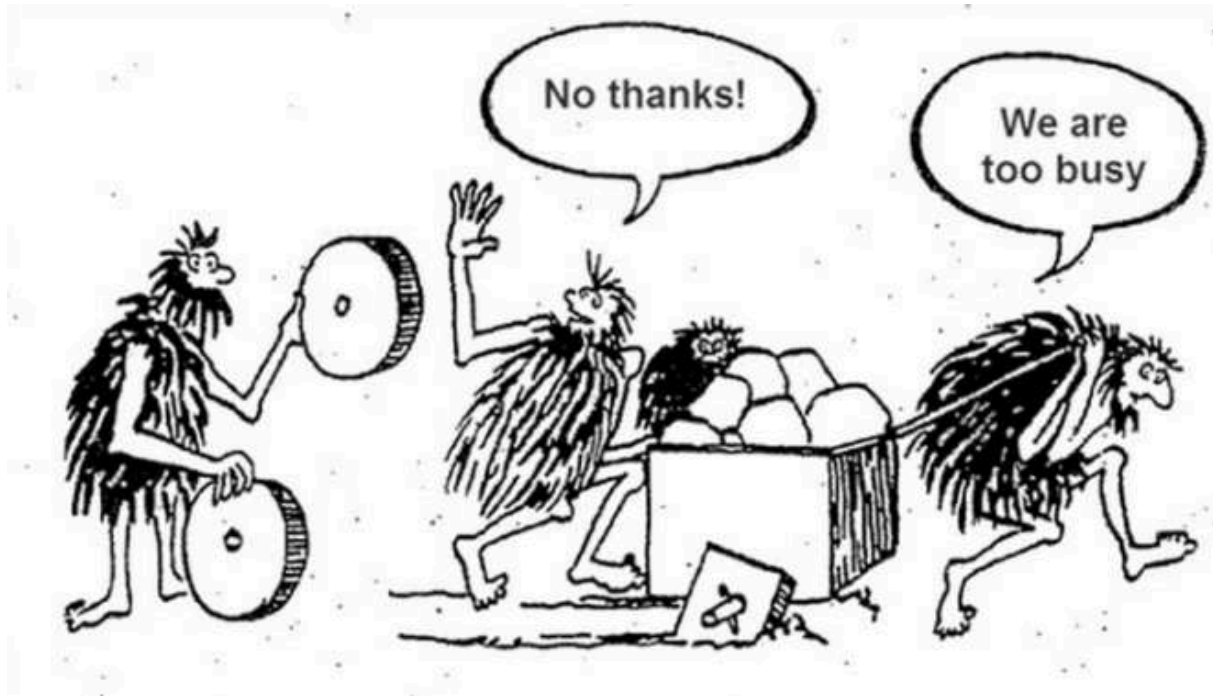
# Netflix Chaos Engineering – Simian Army



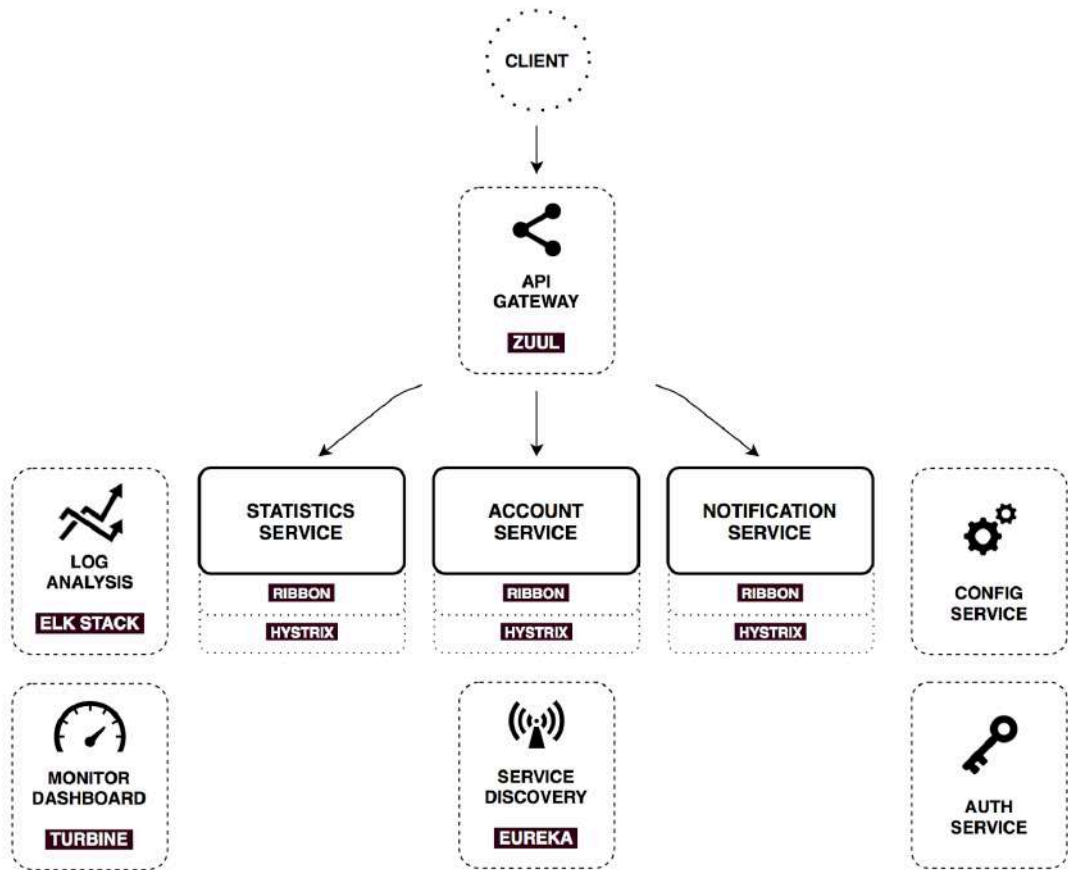
Then How?

---

# Adopting Netflix

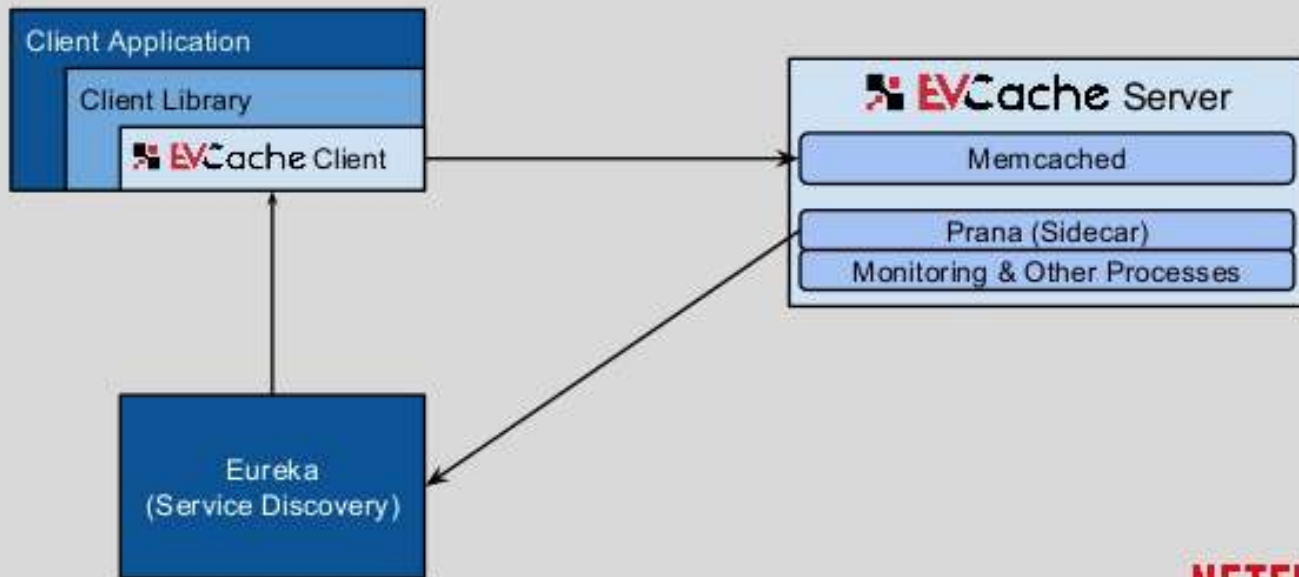


MSA를 구성하는 핵심 중 하나는 '재사용성' 입니다.





# Architecture



NETFLIX

Filters Clear All

Unpin

+ - Group by Pipeline Show 5 per group

Configure

Start Manual Execution

## SEARCH

## PIPELINES

- templation
- wait forever on ci builds
- cbt
- migratable
- multiregion deploy
- sandbox
- major parallel
- minor parallel
- long wait

 demo

Reorder Pipelines

## STATUS

- Running
- Failed
- Completed
- Not Started
- Canceled
- Stopped

Filtered by: PIPELINE: demo

## demo

Configure

Start Manual Execution

## MANUAL START

chrisb@netfix.com  
2015-11-09 21:41:16 PST

Status: RUNNING

Duration: 01:19



Hide Details

## MANUAL JUDGMENT: GO/NO GO DETAILS

Step	Started	Duration	Status
Manuel Judgment: Go/No Go	2015-11-09 21:42:10 PST	00:25	RUNNING

## MANUAL JUDGMENT: GO/NO GO

Manual Judgment Task Status

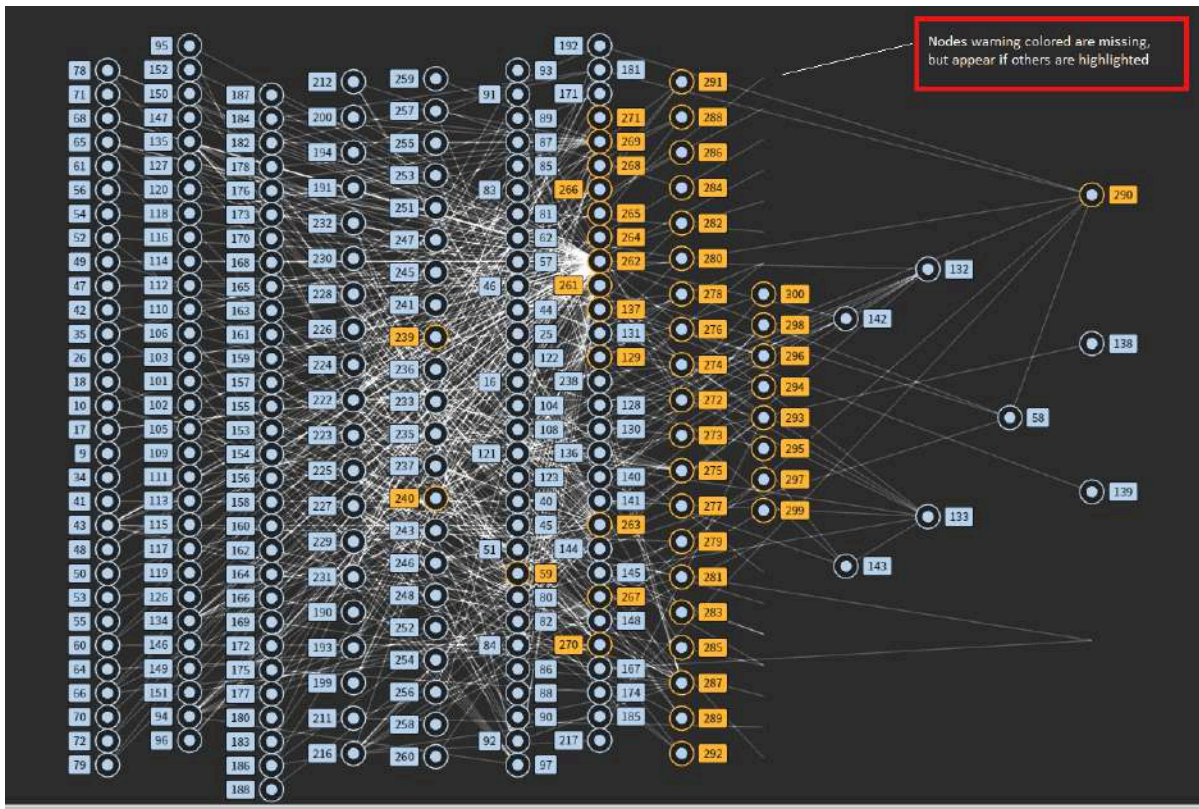
**Instructions** Verify the canary scores are correct and you haven't received any emails or something.

Continue

Stop

Source | Permalink

스피내커(Spinnaker)는 넷플릭스의 CI/CD 도구로 역시 다양한 넷플릭스 플랫폼 도구와 연동합니다. 그리고 지금 보는 화면은 자동화된 '카나리' 업데이트 과정에서 스코어에 따라 수동 조작을 요구하고 있습니다.



따라서 일견 엄청나게 복잡해 보이는 이 넷플릭스의 체계 구성은 앞서 설명한 바와 같이 다양한 애플리케이션의 동작과 이 동작을 지원하기 위한 저장소, 그리고 플랫폼을 통한 연동이 핵심이라고 할 수 있습니다.



애플리케이션의 오케스트레이션, 각 마이크로 서비스가 필요로 하는 서비스의 생성과 바인딩, 그리고 서비스 디스커버리와 같은 기능을 제공하는 것이 거대한 에코시스템이 가져야할 기본이라고 할 수 있습니다.

# Minimum Viable “Cloud” Platform

## 애플리케이션의 배포

- 수많은 팀에서 각자의 애플리케이션에 대한 ‘일배포’를 위해서는 배포의 자동화가 필요
- 블루 / 그린 과 같은 배포의 기법을 사용하여 ‘다운타임 없이’ 배포를 수행할 수 있어야 함
- 배포된 애플리케이션의 안정적 동작을 위해 다수의 AZ를 사용한 오케스트레이션이 필요
- 다양한 언어와 프레임워크를 지원해야 함

## 서비스의 생성과 바인딩

- 각 애플리케이션을 운용하는 팀은 필요에 따라 제한된 쿼타 안에서 서비스를 생성할 수 있어야 함
- 생성된 서비스 정보는 사람이 관리하는 것이 아닌, 시스템이 접근 정보와 패스워드 등을 관리
- 바인딩을 통해 생성된 서비스 정보는 애플리케이션이 동작하는 환경에 ‘항상 동일하게’ 제공되어야 함

## 로그의 수집과 분석

- 신축성이 핵심인 클라우드 환경에서는 원격 로깅이 필수
- 마이크로 서비스간 요청과 응답에 대한 정보의 수집, 각 서비스의 로그 정보 수집등이 가능해야 함
- 수집된 로그를 표현할 수 있어야 하며, 필요에 따라 목적에 맞는 저장소로 전달 할 수 있어야 함
- 실시간에 가까운 지표를 수집하고 확인할 수 있는 공통된 체계가 필요

## 컨테이너의 지원

- 자원 사용의 효율성을 위해 가급적이면 컨테이너로 워크로드를 구동
- 클라우드 서비스 공급자와 관계 없이 멀티 리전, 또는 동일 지역내에 멀티 클라우드 전략을 통해 높은 가용성을 확보할 수 있음
- 개발자 친화적 환경과 운영 팀 친화적 환경 모두를 지원할 필요가 있음.

넷플릭스 플랫폼 서비스들은 ‘누가 어디에 있는지는 시스템이 알고 있으며’, ‘나는 필요할때 어딘가에 만들어 사용하면 되고’, ‘그렇게 사용하면서 발생하는 모든 정보는 쉽게 참조가 가능’ 한 원칙을 가지고 있습니다.



Spring eco system provides a “easy way” to build Micro Services and Cloud Native Applications

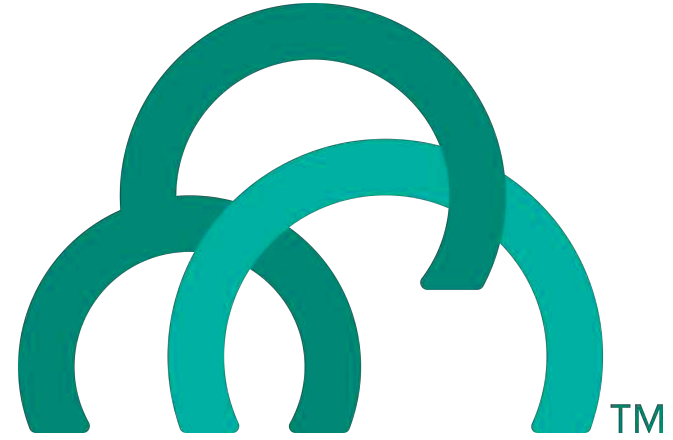


Pivotal Cloud Foundry provides the “Platform” functionality to every single micro services.





**12 Factor Application**



**12 Factor Operation**



Thank You!

---

**감사합니다**

# Some of Our Offices



San Francisco



Atlanta



Berlin



Boston



Boulder



Chicago



Dallas



Denver



Dublin



London



Los Angeles



New York



Paris



Palo Alto



Seattle



Singapore



Sydney



Tokyo



THANK YOU!

**Any Questions?**