



# 서버 성능 테스트 Tips

## - 사례를 통해 얻은 교훈들

---

신호용

Arther.Moon

NGLE - 카카오게임즈

# 발표자 소개

if (kakao) dev 2019

신호용 (NGLE 소속)



NGLE CORPORATION

[www.ngle.co.kr](http://www.ngle.co.kr)

PLATFORM

인증, 빌링, 소셜

SERVICE

WEB, APP

GAME

PC, mobile, VR

소프트웨어 테스트

품질 컨설팅

마켓 관리 지원

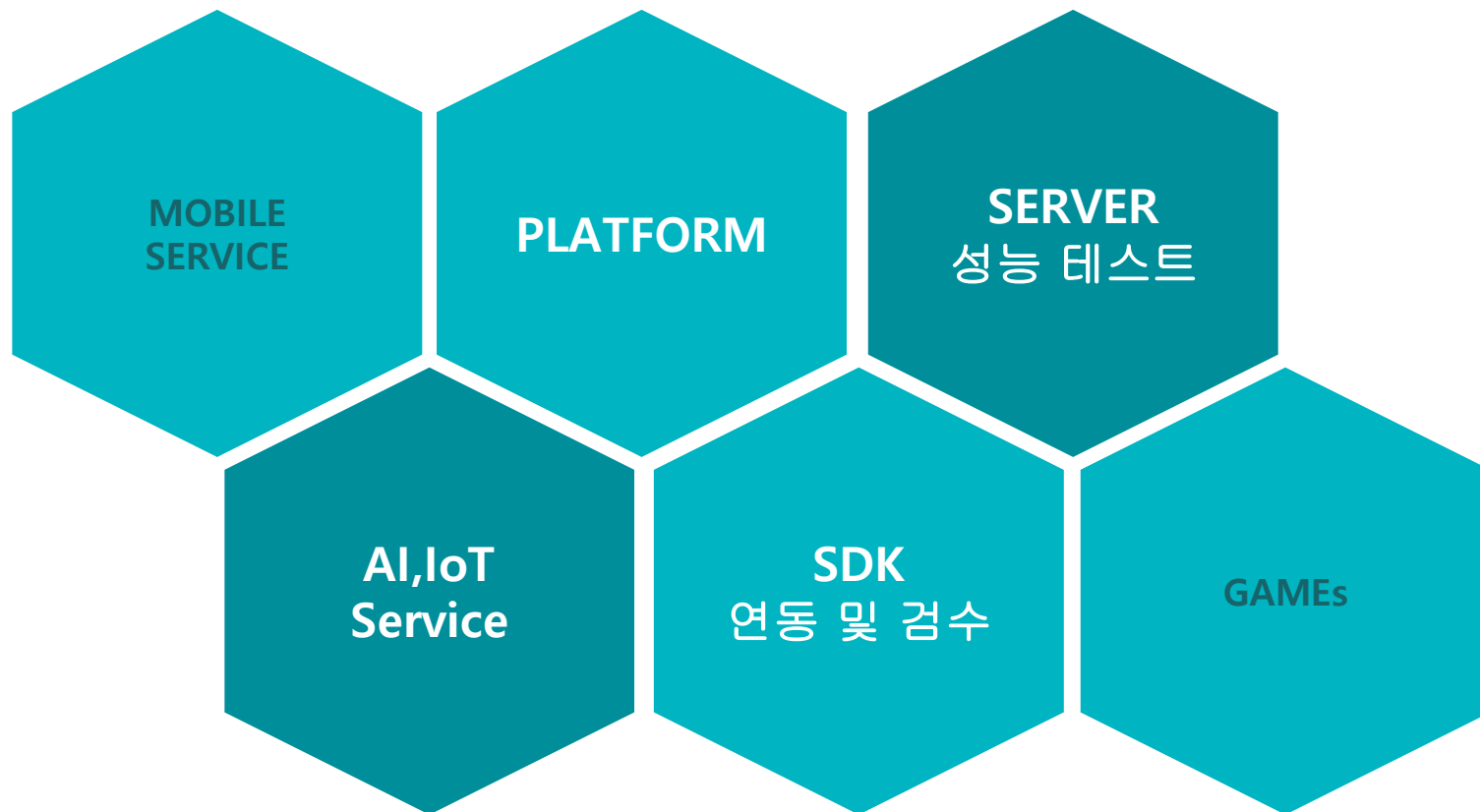
# 발표자 소개

if (kakao) dev 2019

신호용 (NGLE 소속)



NGLE CORPORATION  
[www.ngle.co.kr](http://www.ngle.co.kr)



# 발표자 소개

if (kakao) dev 2019

신호용 (NGLE 소속)

하고 있는 일

## 카카오게임즈에서 런칭한 대부분 서비스의 서버 성능 테스트



도움:

Arther.moon – 카카오게임즈 QA팀

## 01 서버 성능 테스트 개요

1.1 성능 테스트의 필요성

1.2 성능 테스트의 목적

## 02 성능 테스트 진행 사례 및 교훈

2.1 초급 : 이용자의 다양한 액션 누락

2.2 초급 : 최종 시스템의 성능 테스트 누락

2.3 초급 : 예외 상황에 대한 테스트 누락

2.4 중급 : Connection을 유지하는 100만 이용자

2.5 중급 : API검증 실패

2.6 고급 : Server Push로 액션이 시작되는 서비스

2.7 고급 : 120개의 봇을 수동으로 실행한다?

## 03 더 좋은 성능 테스트를 위해 준비하고 있는 것

## 04 오늘의 정리

# 01 서버 성능 테스트 개요

if (kakao) dev 2019

# 성능 테스트 필요성

if (kakao) dev 2019

## 문제 상황



## 해결 방법

◆ 교통 및 신호 체계 개선

◆ 도로 확장

◆ 버스 전용 차선 도입

◆ 우회 도로 개설

◆ 차량 2부제 도입

◆ 대중교통의 활성화



교통 혼잡에 대응 방안을 정하기 전에...

현재 도로의 최대 용량을 파악 후 개선할 부분이 있는지 확인이 필요

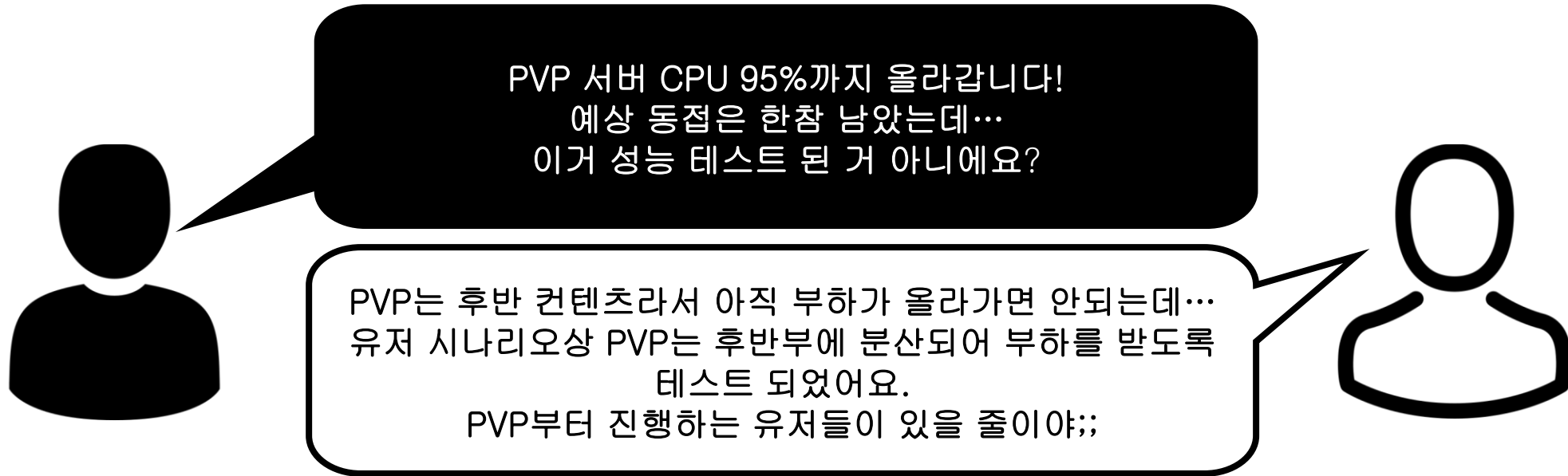
- 시스템의 성능 기준을 정의
- 실제 사용자의 액션과 유사한 시나리오를 작성하여 실제 유저의 부하와 유사한 부하 유입
- 시스템의 병목 구간을 찾아서 튜닝 완료
- 임계값을 기반으로 이용자 증가/감소에 대응할 수 있는 데이터 제공
- 시스템의 무결성을 검증하여 예상하지 못한 장애를 예방 및 대응
- 수집된 데이터들을 바탕으로 시스템 구성 별 성능 예측
- 현재의 성능 테스트를 개선하여 더 나은 성능 테스트 체계 구축



## 02 성능 테스트 진행 사례 및 교훈

if (kakao) dev 2019

## 2.1 초급 : 이용자의 다양한 액션 누락










이용자들은 다양한 흐름으로 콘텐츠를 소비한다!

하나의 시나리오로는 다양한 이용자들의 흐름을 테스트 할 수 없다.

## 2.1 초급 : 이용자의 다양한 액션 누락

### 개선 : 다양한 이용자의 흐름 세분화 및 API별 시나리오 명세 작성

#### 이용자의 유형별 시나리오 세분화

	이용자시나리오_고급
	이용자시나리오_과금
	이용자시나리오_초급
	이용자시나리오_초급_PVP
	이용자시나리오_초급_레벨업
	이용자시나리오_초급_소셜
	이용자시나리오_초급_인던

#### 액션 별 호출 API와 플레이 타임 명시

action	details	play time(ms)	function
초기 설정	초기화	1.5	rank_version
	하트 충전	1.5	etc_make_heart
	월드맵 이동	1.5	rank_map
스테이지(1)	튜토리얼	4	etc_tutorial
	스테이지 랭킹	1.5	rank_stage_ranking
	게임 시작	21	rank_game_start
	튜토리얼	6	etc_tutorial
	게임 종료	2	rank_game_end
	튜토리얼	3	etc_tutorial
	상점	2	buy_shop_event_info
	스테이지 랭킹	1.5	rank_stage_ranking
스테이지(2)	게임 시작	26	rank_game_start
	튜토리얼	1.5	etc_tutorial
	게임 종료	2	rank_game_end

#### API별 호출 수, 플레이 타임 명시

총 API 수	총 시나리오 길이	
105개	665초	약 12분
빌드		
FriendsGem_real_23_1.0.69_180204_03h04m.apk 빌드 기준		
API Call Count		
항목	호출수	비고
CMD_VERSION	1	
CMD_MAP	3	
CMD_STAGE_RANKING	12	
CMD_GAME_START	15	
CMD_GAME_END	15	
CMD_MISSION_LIST	2	

## 2.2 초급 : 최종 시스템의 성능 테스트 누락

### 신규 서비스에서 장애 발생

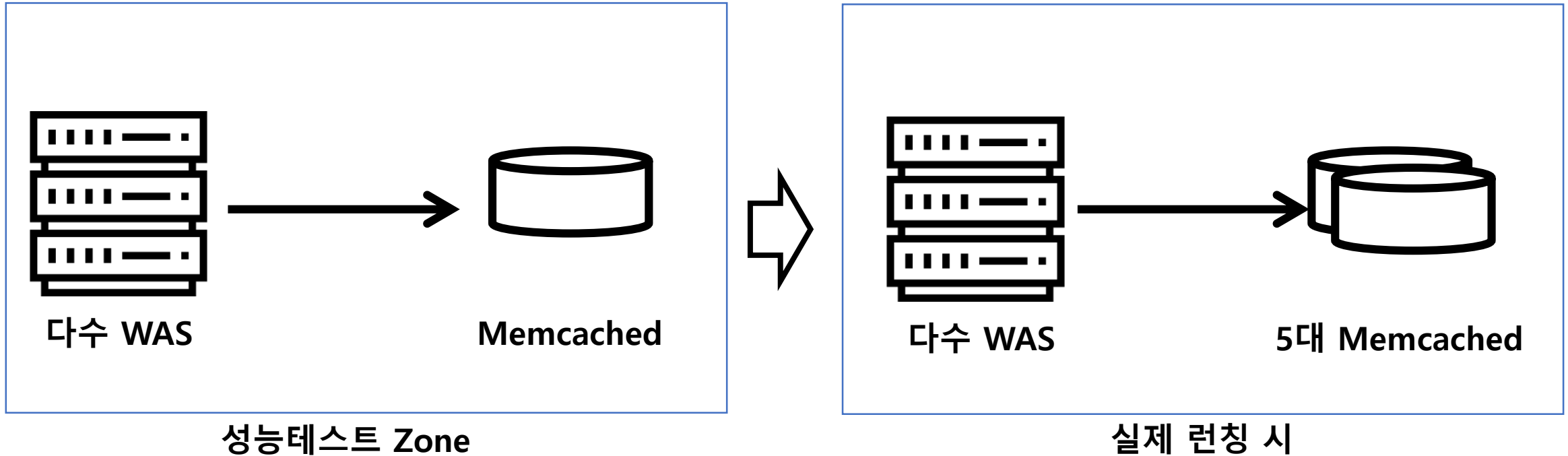
- 서비스 런칭 후 동시 접속자 3만에서 장애
- Memcached 응답속도가 느려지는 현상 발생
- php-fpm의 프로세스에서 지연 현상 발견
- 이후 응답 없음 현상이 발생함
- 분산 Memcached 구조에서의 친구 리스트 호출 회수 증가가 원인

php-fpm이 설치된 장비의 memory full이 원인으로 확인



## 2.2 초급 : 최종 시스템의 성능 테스트 누락

성능 테스트 때는 왜 발견되지 않았나?



분산 memcached 구조에서의 비정상적인 호출 횟수가 원인

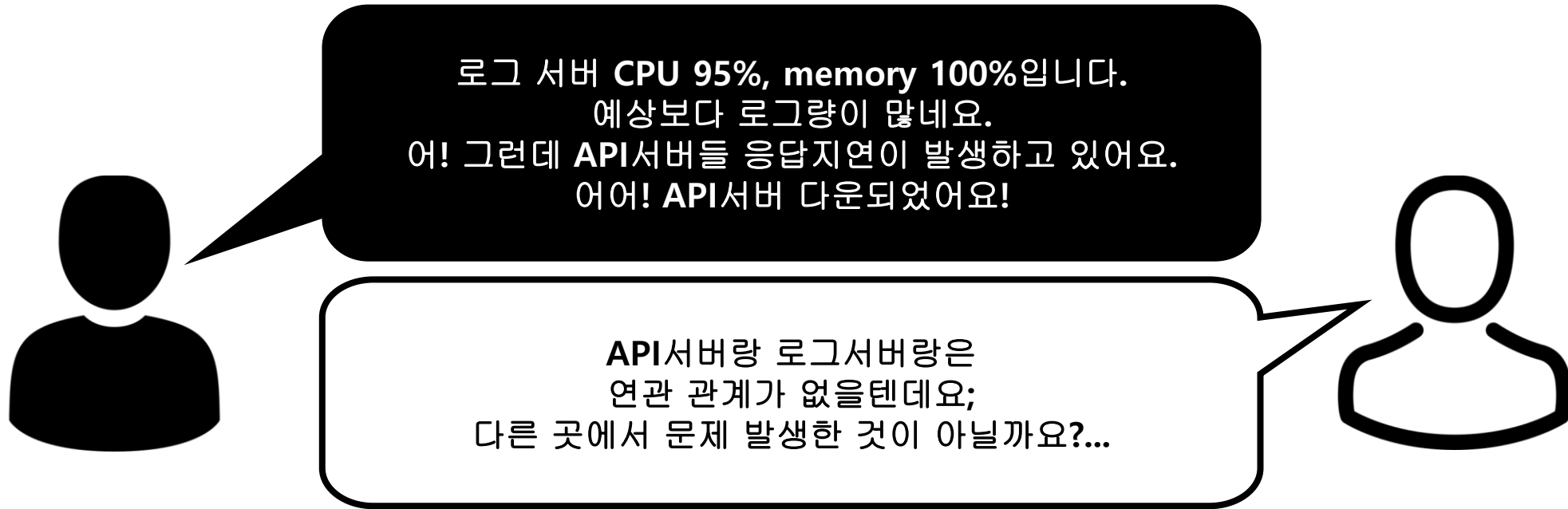
성능 테스트에서는 단일 memcached 구조였으나 런칭 직전 분산 구조로 변경!

## 2.2 초급 : 최종 시스템의 성능 테스트 누락

### 개선 : 성능 테스트 프로세스 개선

- ◆ 시스템에 대한 확장성 및 변경 가능성에 대해 리뷰 강화
- ◆ 성능 테스트 진행 전 테스트 환경에 대해 담당 개발자 검증 단계 도입
- ◆ 최종 런칭 스펙과 동일한 구성으로 성능 테스트 진행하도록 프로세스 강화
- ◆ 성능 테스트 종료 후 시스템의 구조 변경은 서비스 담당자들이 합의하도록 변경

## 2.3 초급 : 예외 상황에 대한 테스트 누락



각 서버 별 연관 관계도 확인이 필요하다.

## 2.3 초급 : 예외 상황에 대한 테스트 누락

### 개선 : 모든 구성 서버들에 대해 예외 상황 테스트 진행

◆ 각 서버들에 대한 예외적인 상황에서의 테스트 방법 및 영향 범위를 명시하여 테스트 진행

action	details	impact scope
게임(API) 서버 장애 (scale in/out 확인)	1. 게임 서버 1대 장애 발생(2대로 시작후 1대 장애 발생)	게임 플레이 영향 및 정상화 확인
	2. 게임 서버 나머지 한대 장애 발생(게임 서버 모두 장애)	게임 플레이 진행 불가 ( pvp 진행 유저 상태 파악)
	3. 게임 서버 1대 정상화 (1대 정상화)	게임 플레이 정상화
	4. 게임 서버 1대 추가 (scale-out)	게임 서버 추가 확인 / 기존 유저 게임 정상
	5. 게임 서버 1대 제거 (scale-in 상황에 따라 테스트 제외 가능)	게임 서버 제거 확인 / ELB에서 서버 제거 /기존 유저 게임 정상
pvp 서버 장애	1. PVP 서버 1대 장애 발생(2대로 시작후 1대 장애 발생)	pvp 플레이 영향 및 정상화 확인
	2. PVP 서버 나머지 한대 장애 발생(게임 서버 모두 장애)	pvp 플레이 진행 불가 (로비 진입 가능 여부 파악)
	3. PVP 서버 1대 정상화 (1대 정상화)	pvp 플레이 정상화
	4. PVP 서버 1대 추가 (scale-out)	서버 추가 확인 / 기존 유저 게임 정상
DB 장애 (common)	1. DB connection 실패	게임 플레이 영향 및 정상화 확인 / 부하 1천 정도 유입
	2. DB 정상화 (fail over 확인)	
DB 장애 (game)	1. DB connection 실패	게임 플레이 영향 및 정상화 확인 / 부하 1천 정도 유입
	2. DB 정상화 (fail over 확인)	
DB 장애 (log)	1. DB connection 실패	게임 플레이 영향 및 정상화 확인 / 부하 1천 정도 유입
	2. DB 정상화 (fail over 확인)	

◆ Auto scaling 및 DB Failover에 대해서도 사전에 검증을 진행



## 2.4 중급 : Connection을 유지하는 100만 이용자

### 특이 사항

- 이용자 100만명이 30분 동안 모두 인입되며 각각 Connection을 유지

대규모 트래픽의 분산, Connection 모두 유지 - Load balancing 중요



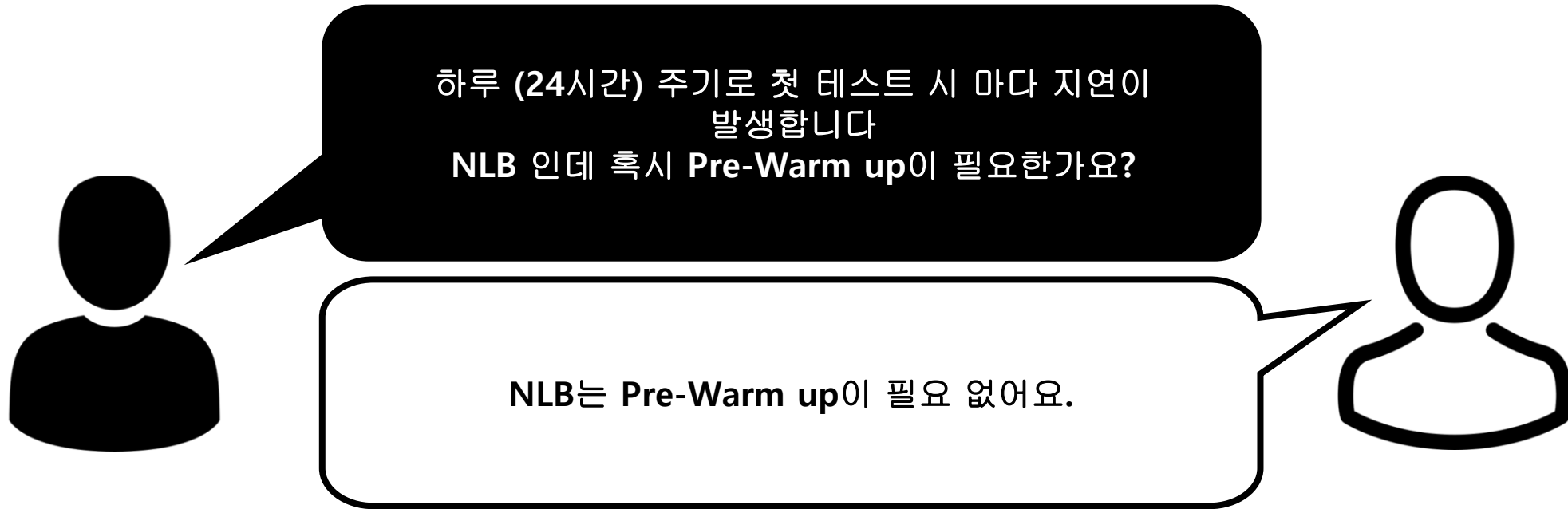
AWS  
Network Load Balancer



초당 수백만 개의 요청을 로드 밸런싱 할 수 있고  
변동이 심한 트래픽 패턴도 처리할 수 있다고 안내되어 있음

약 70만 vuser 인입 시 응답 지연 현상 및 Error 발생 발견

## 2.4 중급 : Connection을 유지하는 100만 이용자



NLB에서는 Pre-warm up이 없다고 하지만...

이후 2주 더 테스트 진행했지만, 지속적인 장애 발생. 원인은?

## 2.4 중급 : Connection을 유지하는 100만 이용자

### 문제 해결

- NLB에서도 Pre-Warm up이 필요하다는 답변
- NLB의 Pre-Warm Up을 요청하여 현상 해결

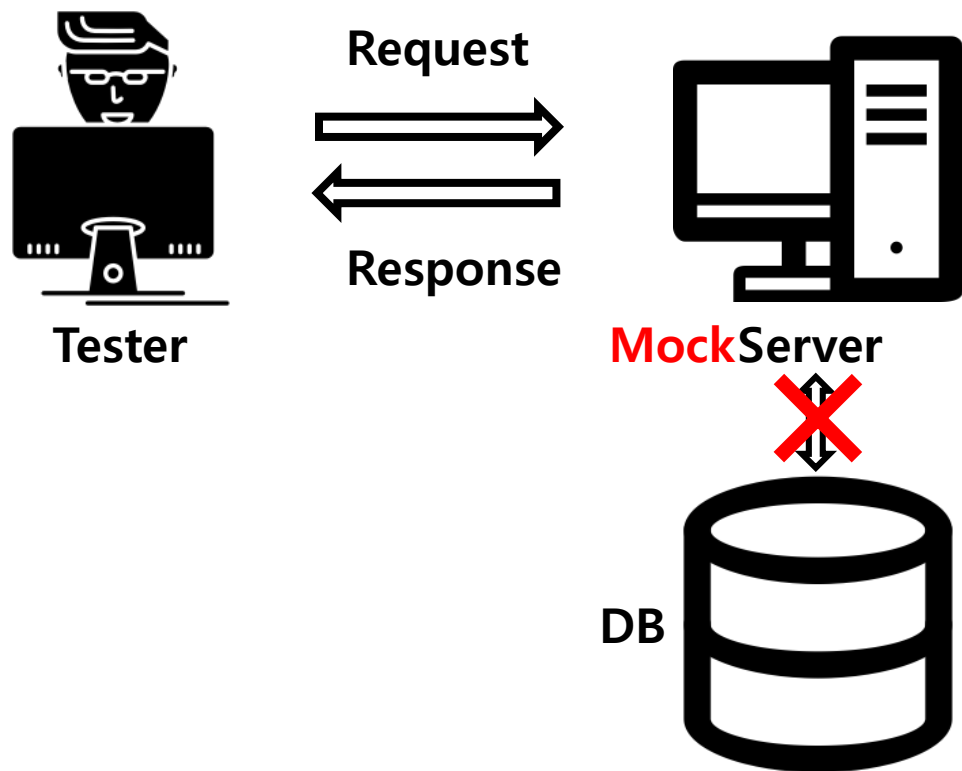
### 교훈

- 가이드를 참고 하되, 목표치 까지 직접 테스트하는 것이 확실한 결과를 측정할 수 있음

## 2.5 중급 : API 검증 실패

### API 검증이란?

- 성능 테스트 진행 전 테스트 대상 API들이 정상적으로 동작하는 것을 확인하는 작업
- Request parameter 및 return value를 확인



### 실패 상황

- 테스트 대상 API Server에서 정상 응답 확인 완료
- 성능 테스트 진행 :  
단위 서버당 성능 측정 수치가 비정상적으로 높은 것이 관측
- API 및 처리 로직, DB, cache 전체 구성에 대해 검토
- DB에 데이터가 기록되지 않는 현상 발견

Server가 **MOCK**으로 확인됨

daum사전

mock

1. (한정적) 가짜의 2. 모조의 3. 거짓의 4. ...을 조롱하다 5. 흉내내며 놀리다

## 2.5 중급 : API 검증 실패

### 왜 이런 상황이 발생했을까?

- 해외 개발사의 서비스로서 사전에 서버 구성이나 성능테스트의 방법에 대한 충분한 커뮤니케이션 부족
- API호출 시 Return Value는 Cache에서 정상적인 값을 가져왔기 때문에 예상하지 못함
- Data Write는 200 OK가 Return되어 정상적으로 기록된 것으로 판단하였음

### 개선 :

- ◆ 성능 테스트 전 사전에 개발자 – QA간에 성능테스트에 대한 논의 강화
- ◆ API검증 강화 : Write기능의 API는 검증 단계에서 어드민 툴 등으로 실제 데이터 확인

## 2.6 고급 : Server Push로 액션이 시작되는 서비스

### 특이 사항

- client와 server는 web socket으로 통신하며 server push로 액션이 시작



#### 고민

- Test Tool 로 사용 중인 nGrinder는 Vuser 단일 Thread로 실행됨
- 하나의 Thread로 Listening 상태 유지하면서, 별도의 Thread로 통신이 필요
- 성능툴의 부하량이 증가하고, 원활하게 시나리오를 동작시키기 어려움

봇클라이언트로 테스트 진행 했으나, 개발자 리소스 증가

## 2.6 고급 : Server Push로 액션이 시작되는 서비스

### 💡 문제 해결

Network Programming을 편하게 할수 있는 FrameWork

- 입출력(IO)만 담당하는 공용 Thread를 Netty로 개발
- Server와의 request와 response는 I/O Thread가 수행
- nGrinder의 vuser는 I/O Thread의 send queue에 데이터를 넣고, receive queue에서 데이터를 가져가는 역할



웹 서버 성능 테스트의 한계 극복

성능 테스트 진행 시, 개발자 리소스 절감

## 2.7 고급 : 120개의 봇을 수동으로 실행한다?

### 특이 사항

- 외부 개발사에서 제작한 봇 클라이언트로 테스트를 진행
- 봇 클라이언트 1개당 500EA Vuser 생성(목표 동접 6만)
- 총 120개의 클라이언트를 **동시에(!) 실행**시켜서 부하를 발생해야 함
- 이용자의 닉네임은 중복되지 않도록 생성되어야 함



#### 고민

- 120개의 봇 클라이언트를 어떻게 동시에 실행/종료 해야하는가?
- 새로운 닉네임을 생성하려면 120개의 봇 설정 파일을 매번 수정해야하나?
- 유저 수 증가, 봇 클라이언트 추가 시 매번 수동으로 작업해야 하나?



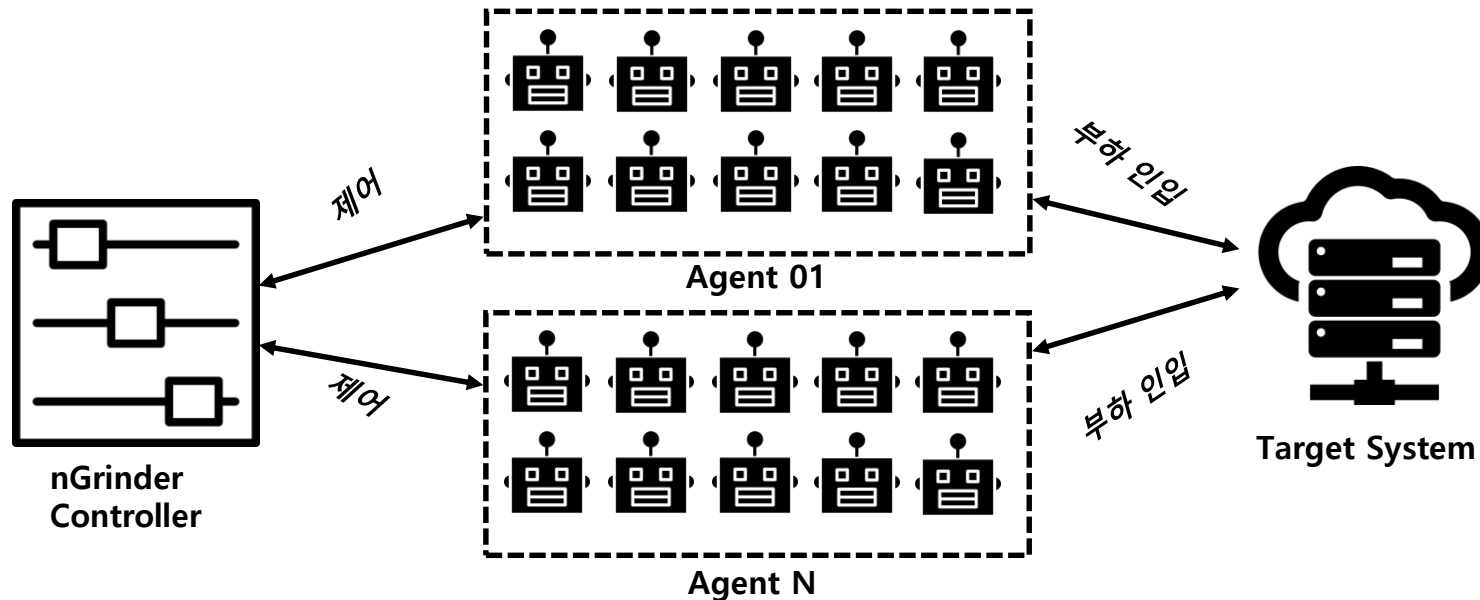
# 2.7 고급 : 120개의 봇을 수동으로 실행한다?



## 2.7 고급 : 120개의 봇을 수동으로 실행한다?

### 💡 문제 해결

- 사용하던 성능 테스트 툴의 Controller가 봇 클라이언트를 제어하도록 변경
- 한 개의 Controller가 N개의 봇을 실행하는 BAT File 구동 및 제어



# 03 더 좋은 성능테스트를 위해 준비하고 있는 것

if (kakao) dev 2019

## 시스템의 기본 성능에 대한 기준을 만들 수 있지 않을까?

◆ AWS의 EC2 instance 종류 별 적정 TPS, 혹은 수용 가능한 동시 접속자 수.

*예시) C4.xlarge는 1000TPS, 혹은 C4.xlarge 한대 기준 동시접속자 3000 명*

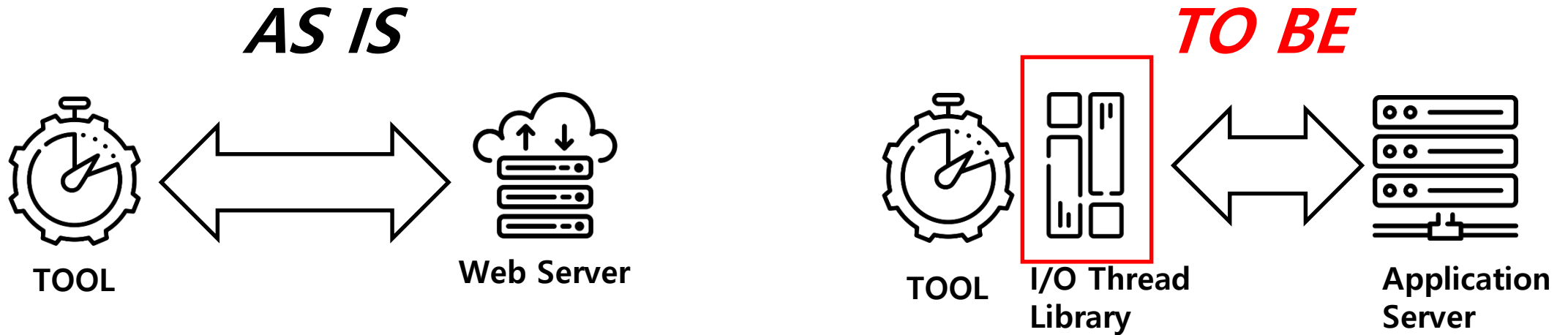
◆ 시스템 구성에 대한 적정 TPS

*예시) Front와 API로 구성되어있고, PHP-FPM과 NginX이니까 2000TPS*

*ing*  
결과 데이터 분류 & 저장

# 테스트 가능 환경의 확장

if (kakao) dev 2019



- Web Server를 주로 테스트
- http, ws로 한정

- Web Server + **Application**
- http, ws + **TCP**

*ing*

## Library 개선 및 사례 수집

## 04 오늘의 정리

- 시스템의 성능 기준을 정의
  - 실제 사용자의 액션과 유사한 시나리오를 작성하여 실제 부하와 가까운 부하 예측
  - 시스템의 병목 구간을 찾아서 튜닝 완료
  - 정확한 임계값을 기반으로 이용자 증가/감소에 대응할 수 있는 데이터 제공
  - 시스템의 무결성을 검증하여 예상하지 못한 장애를 예방
  - 수집된 데이터들을 바탕으로 시스템 구성 별 성능 예측
  - 현재의 성능 테스트를 개선하여 더 나은 성능 테스트 체계 구축
- 2.1 이용자의 다양한 액션 누락
- 2.2 최종 시스템의 성능 테스트 누락
- 2.4 Connection을 유지하는 100만 이용자
- 2.3 예외 상황에 대한 테스트 누락
3. 더 좋은 성능 테스트를 위해 준비하고 있는 것

# 질문 & 응답