

Practical Microservices in gRPC Go feat. GraphQL, Kafka

카카오 T 대리 “서포터즈 기사” 개발 사례

강태훈 travis.kang, 김민영 zest.ly, 문주성 ether.moon, 양진선 david.y, 이하건 brian.lee
카카오모빌리티

01 서포터즈 기사

if(kakao)dev 2019

승차거부 없는 대리운전 나온다...카카오 '시급제' 기사 모집

기사입력 2019/05/29 19:34 송고

(서울=연합뉴스) 채새롬 기자 = 카카오모빌리티가 대리운전 서비스에 승차거부 없는 자동 배정 시스템을 도입한다.

카카오모빌리티는 28일부터 카카오 대리기사 기사용 애플리케이션을 통해 '서포터즈 기사'를 모집하고 있다고 29일 밝혔다.

서포터즈 기사는 오후 9시부터 익일 오전 1시까지 강제배차를 받고 시간당 1만4천원의 고정 시급을 받는다.

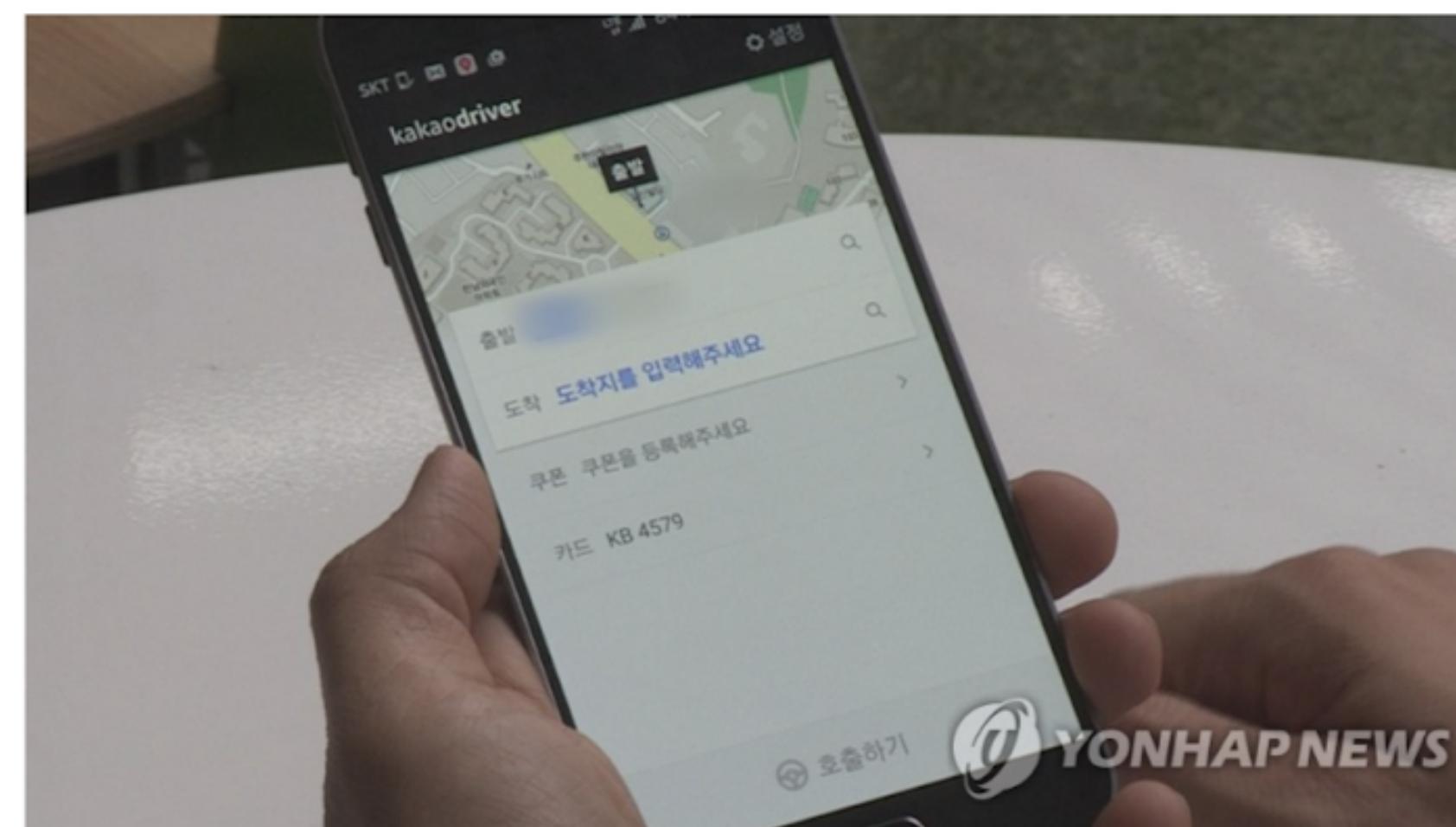
목적지 안내 없이 콜을 받게 되며, 특별한 사정이 있을 때 이외에는 배정을 '거절'할 수 없다. '활동하기'를 선택하지 않으면 평상시처럼 수수료 모드로 운행할 수 있다.

카카오모빌리티는 최근 4주간 운행 이력, 배정 후 취소율, 사고와 과태료 이력, 고객 평점 등을 고려해 서포터즈 기사를 선발할 계획이다.

다음 달 중순 수도권, 안드로이드 단말기를 통해 시범 운영을 시작한다.

카카오모빌리티 관계자는 "대리기사는 안정적인 수익을 확보할 수 있고 이용자는 강제배차를 통해 서비스 편익이 올라갈 수 있다"며 "시범 운영 기간에 부작용을 파악하고 개선해 나갈 것"이라고 말했다.

다만 이 서비스는 택시에는 도입되기 어려울 전망이다. 이 관계자는 "플랫폼 사업자가 일반 택시에 자동 배정을 강제하고 시급제 등을 운영할 권한이 없다"고 말했다.



카카오드라이버
[연합뉴스TV 제공]

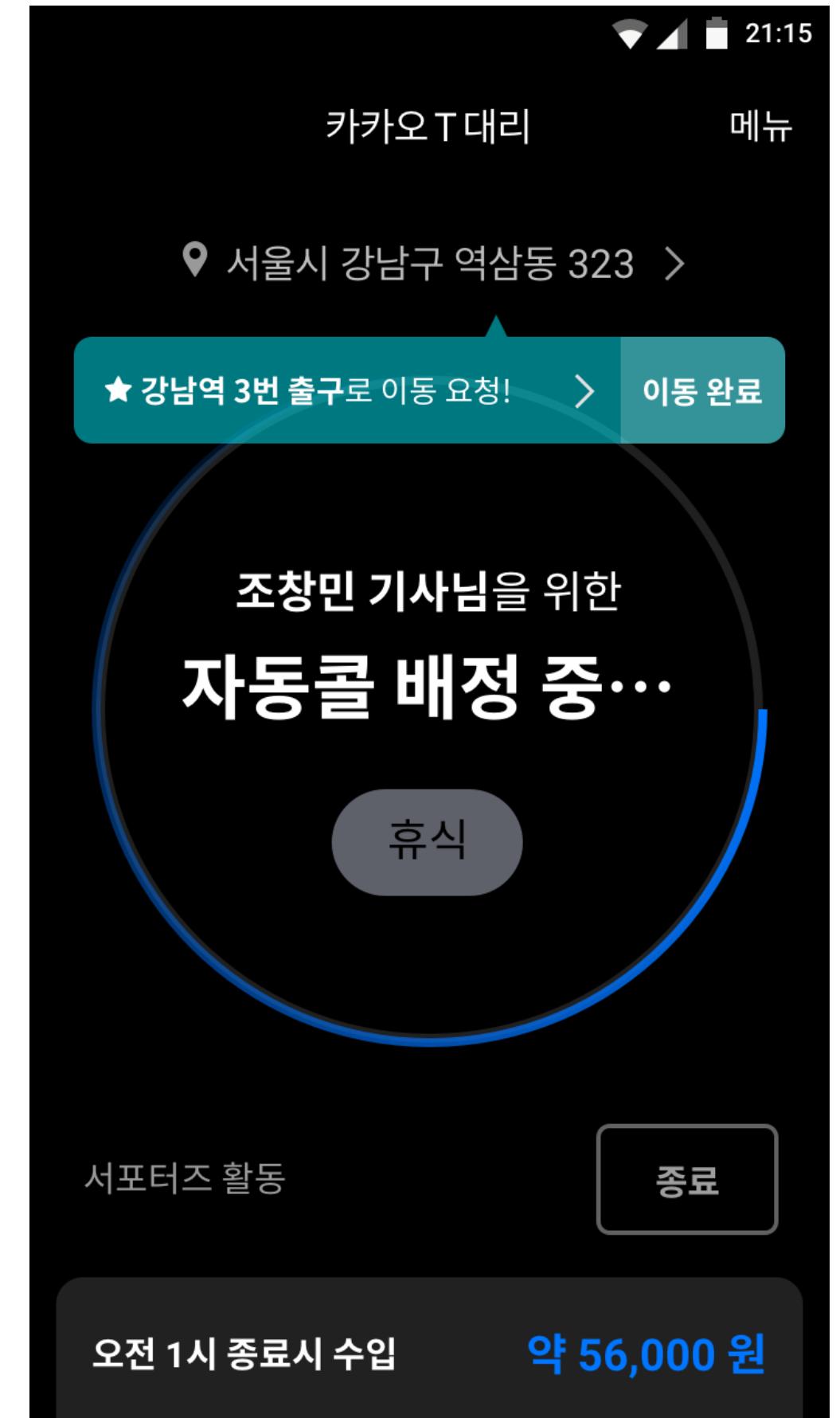
요구 사항 (일부)

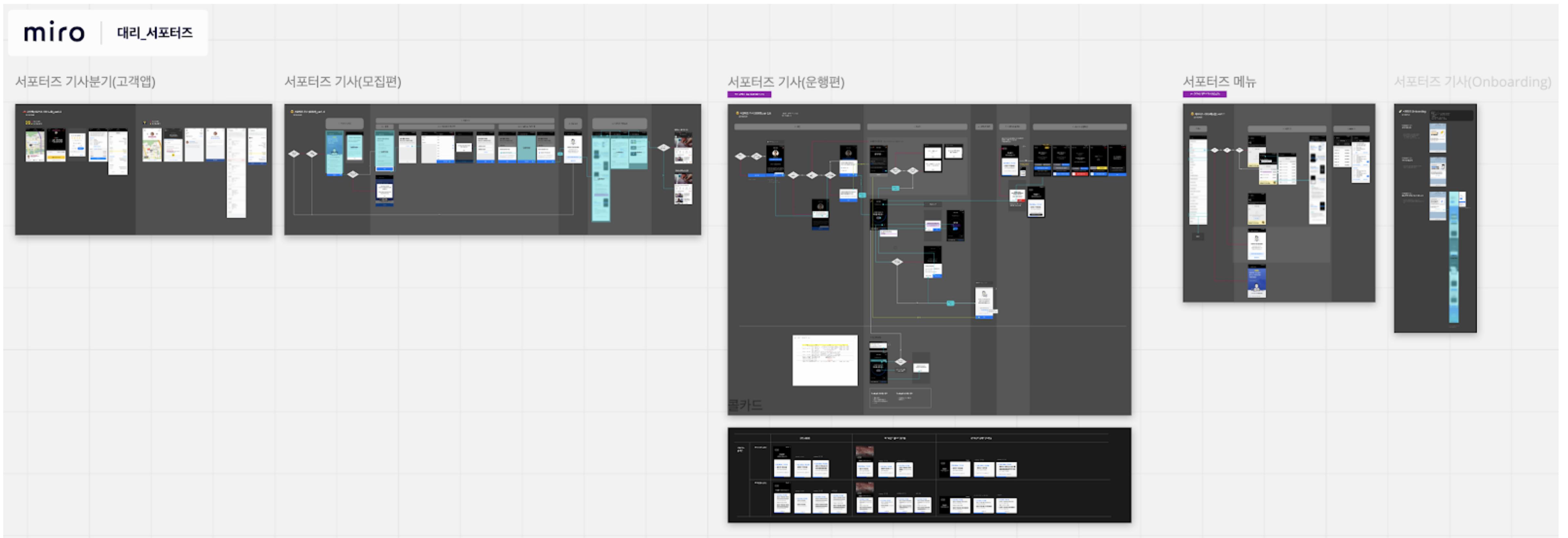
if(kakao) dev 2019

- 일정한 기준을 충족하는 기사군을 대상으로 가입을 받고 지원할 수 있어야 한다.
- 서포터즈 기사군은 기계학습에 기반한 더욱 지능적인 배정의 대상이 되어야 한다.
- 목적지가 표시 되지 않고 N초내에 거부하지 않으면 자동 배정되어야 한다.
- 특정시간대(21시 ~ 01시)에 일 1회 참여만 가능해야 한다.
- 활동 지역은 알고리즘에 의해 일정 수준의 수요가 특정 밀도내에 존재하는 지역으로 한정한다.
- 활동시간 중 자유롭게 휴식 및 재개가 가능해야 한다.
- “장소 이동”이라는 새로운 형태의 추천기능이 추가된다.
- 기사수익이 건당 지급에서 시간제로 변경된다. 지급 분단위 올림 계산.

⋮

약 90 개의 사용자 스토리





Players

if(kakao)dev 2019

카카오 T 대리개발파트 인력 구성



파트내 최고참



Tech Lead



Rails 전문



E-commerce 출신



Java 전문

카카오 T 대리개발파트 상황

if(kakao)dev 2019

외부 영입

성숙해지는 사업에
따른 인재배치

주요 기술 Stack

Rails, Vert.x, Java,
Groovy, JRuby 등

운영 대응

급성장세덕에 운영에
많은 시간을 쓰는 중

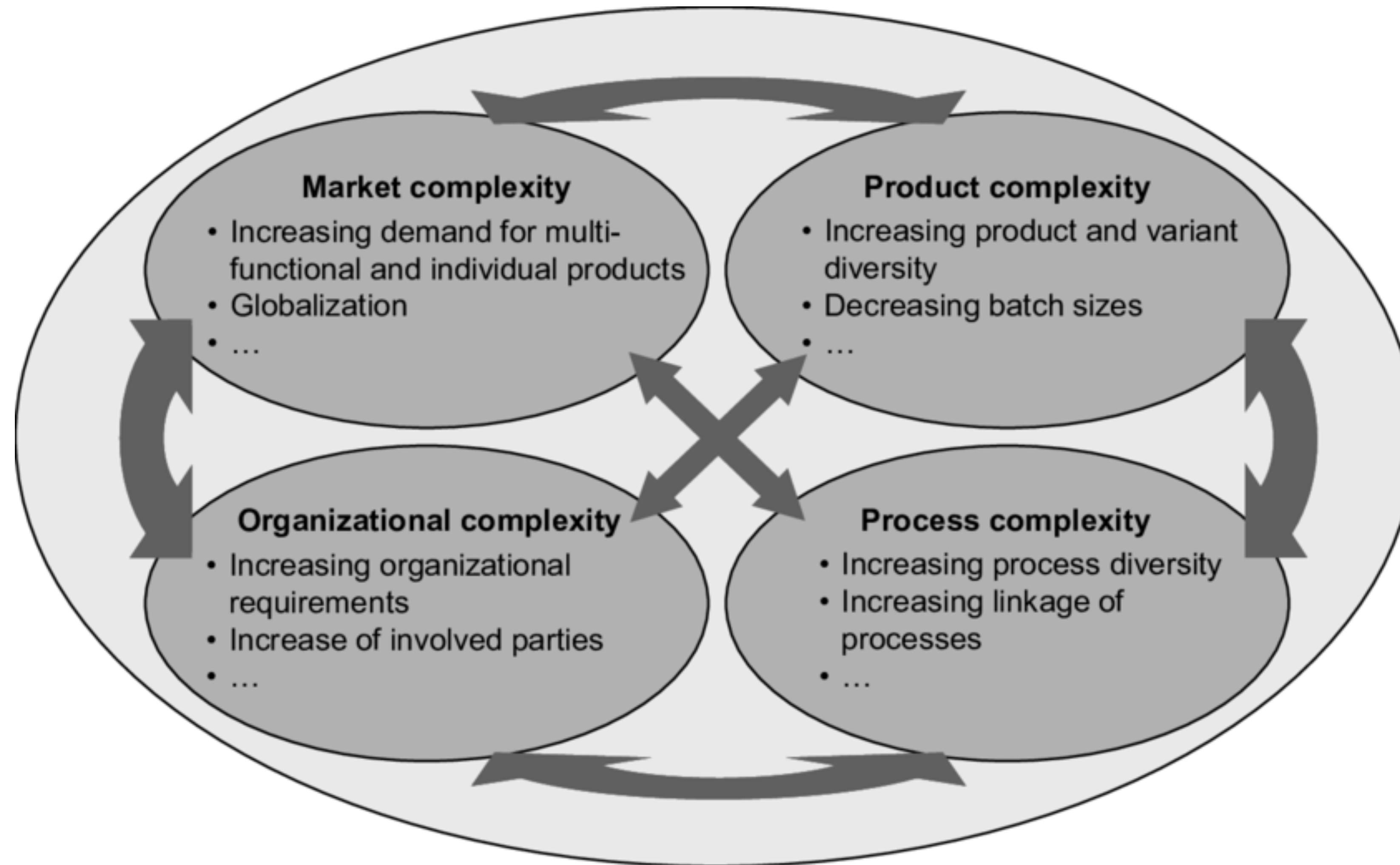
학습 비용 증가

급격히 복잡해지는 시스
템 이해에 많은 노력



개발 복잡도의 상관 관계

if(kakao)dev 2019

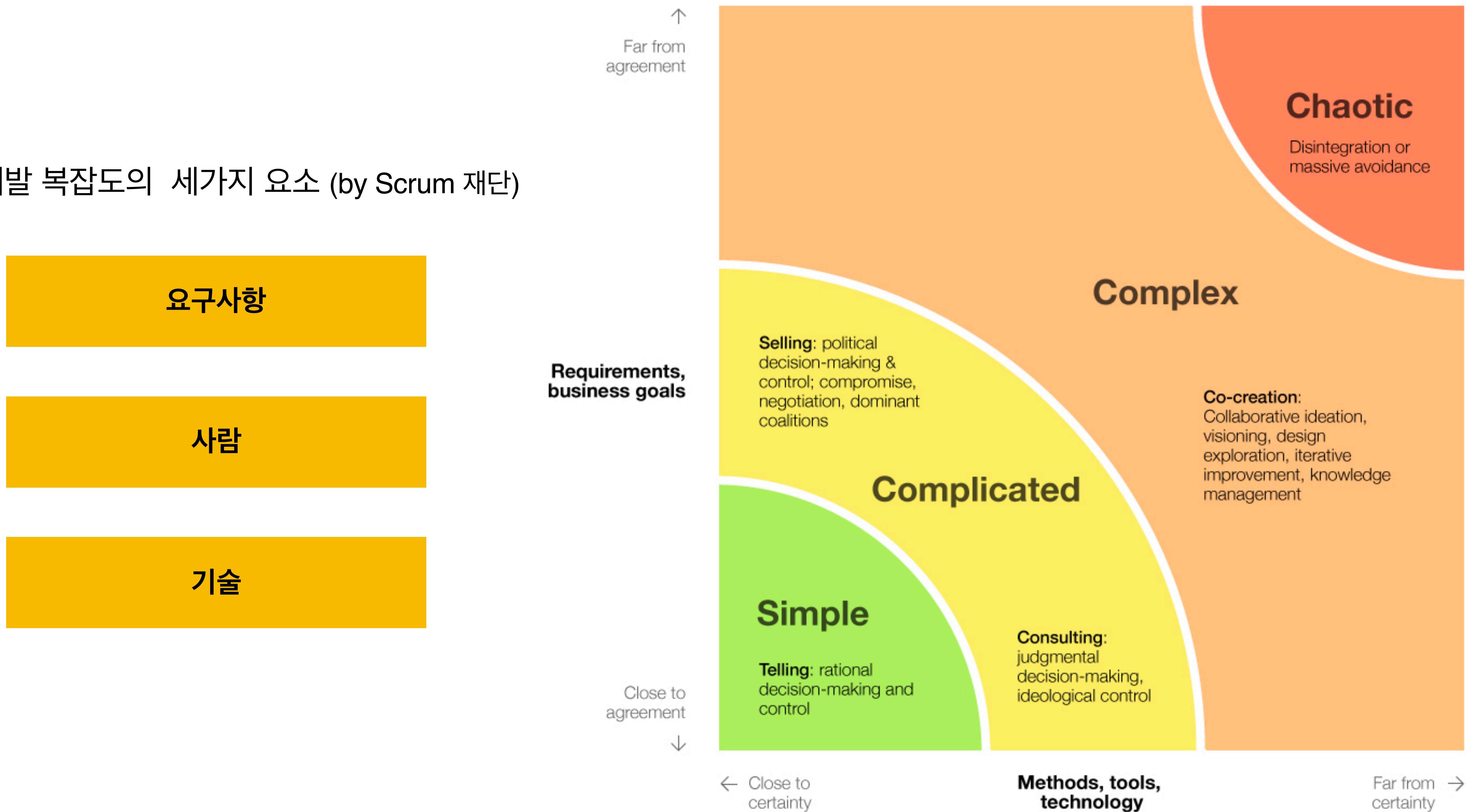


* 인용: Maik Maurer 의 Structural Awareness in Complex Product Design

복잡, 암튼 복잡

if(kakao)dev 2019

제품 개발 복잡도의 세가지 요소 (by Scrum 재단)



02 생각의 정리부터

if(kakao)dev 2019

어떻게 설계 / 구상해야 할까?

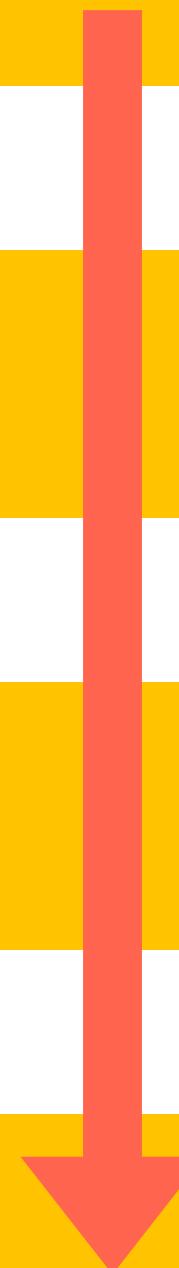
if(kakao)dev 2019

새로운 것을 개발하기 앞서 생각을 정리하여 계획한다.

해당 계획을 짧은 문서에 담는다.

일을 시작하기 전에 소수의 선택된 사람들에게 이 계획을 승인 받는다.

사내 모든 개발자에게 이 계획 문서를 공유한다.



- 작성자 목록
- 승인자 목록: 문서를 검토해 줄 수 있는 조력자
- 요약: 무엇을 하기 위해 제안 하는가?
- 배경: 서비스가 출현하게 된 동기
- 구조: Diagram 으로 설명
- 고려했던 대안: 고민했던 구조의 장단점 기술
- 구현: API 구조, 개체 모델링 등 상세 구현 사항
- 서비스 SLA: 정확도, 수용량, 지연율, 가용율 등
- 서비스 의존성: Upstream, Downstream 의존성
- 부하 및 성능 테스트: SLA 의 수용량, 지연율 검증
- 보안 고려 사항: 민감한 정보를 어떻게 다룰 지
- 테스트 및 출시 계획
- 지표 및 감시 사항
- 고객 지원 고려 사항

서포터즈 기사 Routing 서비스 RFC

작성자 목록

brian.lee@kakaomobility.com
travis.kang@kakaomobility.com
zestly@kakaomobility.com

승인자 목록

anderson.jo@kakaomobility.com
currying@kakaomobility.com
ether.moon@kakaomobility.com
david.y@kakaomobility.com
sean.you@kakaomobility.com

요약

서포터즈 기사의 운행을 담당하는 서비스

배경

서포터즈 기사는 새로운 기사 단체의 dispatch를 데이터 기준 dispatch 서비스에 추가하지 않고 새로운 서비스로 제작

참고

- 서포터즈 ML API

구조

이해 diagram 들에서 초록색은 Data platform 을 나타냄.

요소

- Call: 고객 앱에서 발생하는 대리 운전 서비스 요청
- Routing Service: 모든 Call 을 입력받아 서포터즈 명령을 발송하거나 기존 Dispatch service 에 전달
- ML Gateway (기정): 미리 정의된 모델에 따라 입력되는 서포터즈 기사를 추천

* 기사그룹 서비스(기정)에선 특정 기사군의 활동모드 ON/Off 를 mongo 에 활동모드 정보를 동기화

Call 수행 흐름

Call 수행은 Call 이 생성되는 시점에 요청을 받아 작동된다.

```

sequenceDiagram
    participant Client
    participant RS [Routing Service]
    participant TS [Tagging Service]
    participant MQ [Message Queue]
    participant MLG [ML Gateway]
    Client->>RS: Call
    activate RS
    RS->>TS: Forward Call to Tagging Service
    activate TS
    TS->>MQ: Tagging Service to Message Queue
    activate MQ
    MQ->>MLG: Message Queue to ML Gateway
    activate MLG
    MLG->>Client: Call Result
    deactivate Client
    deactivate MLG
    deactivate MQ
    deactivate TS
    deactivate RS
    
```

1. 모든 Call은 서포터즈 기사 Routing Service 로 입장
2. 모든 Call 을 전달
3. 서포터즈 기사에 적합한 Call 인지 판별하고 해당하는 기사를 추천한다.
4. 모든 Call
5-1. 서포터즈 기사가 Call의 경로를 알면 바로 기존 Dispatch 직통
5-2. 주차된 서포터즈 기사가 없으면 보내기 기사에게 Call 수행 명령 발송
6. 기사에게 Call 수행 후
Commander Service

Call 수락 흐름

Call 수락은 기사가 콜을 수락하는 시점에 기준과 동일하게 wheel-api 의 reject api(PUT {wheel-api}/d/v1/calls/reject) 를 호출하고, 여기서 kafka 를 action_name: "CALL_REFUSED" 인 activity 메시지를 producing 한다. Routing service 는 이 메시지를 consume 하여 wheel-dispatch 에 일반 배정 요청을 한다.

```

sequenceDiagram
    participant Client
    participant WA [Wheel-api]
    participant AMQ [Activity Message Queue]
    participant RS [Routing Service]
    Client->>WA: Call 수락
    activate WA
    WA->>AMQ: CALL_ACCEPTED 메시지 producing
    activate AMQ
    AMQ->>RS: Activity Message Queue consuming
    activate RS
    RS->>Client: CALL_ACCEPTED
    deactivate Client
    deactivate RS
    deactivate AMQ
    
```

1. Call 수락시 기준과 동일하게 wheel-api 로 입장
2. CALL_ACCEPTED 메시지 producing
3. CALL_ACCEPTED 메시지 consuming
4. 배치원으로 풀백 POST /api/v1/calls/dispatch_results.json

Call 거절 흐름

Call 거절은 기사가 콜을 수락하는 시점에 기준과 동일하게 wheel-api 의 reject api(PUT {wheel-api}/d/v1/calls/reject) 를 호출하고, 여기서 kafka 를 action_name: "CALL_REFUSED" 인 activity 메시지를 producing 한다. Routing service 는 이 메시지를 consume 하여 wheel-dispatch 에 일반 배정 요청을 한다.

```

sequenceDiagram
    participant Client
    participant WA [Wheel-api]
    participant AMQ [Activity Message Queue]
    participant RS [Routing Service]
    Client->>WA: Call 거절
    activate WA
    WA->>AMQ: CALL_REFUSED 메시지 producing
    activate AMQ
    AMQ->>RS: Activity Message Queue consuming
    activate RS
    RS->>Client: CALL_REFUSED
    deactivate Client
    deactivate RS
    deactivate AMQ
    
```

1. Call 거절 시 기준과 동일하게 wheel-api 로 입장
2. CALL_REFUSED 메시지 producing
3. CALL_REFUSED 메시지 consuming
4. 일반 배정 요청

이동 흐름

이동은 ML pipeline 이 맥락에 따라 특정 기사를 이동해야 될 신호를 감지할 경우 이를 message queue 에 publish 함으로 trigger.

김태훈 travis.kang
오전 9:18 4월 24일

+brian.lee@kakaomobility.com 일반 배정 요청 하는 것이 맞는지 모르겠네요. 일전에 서포터즈 기사 배정 거부시 배정실패로 처리하기로 한 것 같은데..

이하간 brian.lee
오후 4:47 4월 24일

오늘 댓글 달아주셨었군요...ㅎㅎ 좀 전에 이야기 한 것처럼 일반배정으로 보내는건 open spec 에서 제외이고, 어려워 되면 가능한 빨리 넣는 것으로 생각하고 있을게요~

03 프로그래밍 언어

if(kakao)dev 2019

Java 의 장단점

if(kakao)dev 2019

장 점

- 익숙하다.
- Spring Framework Almighty
- 개발자 구인이 용이하다.
- OS 독립: WORE

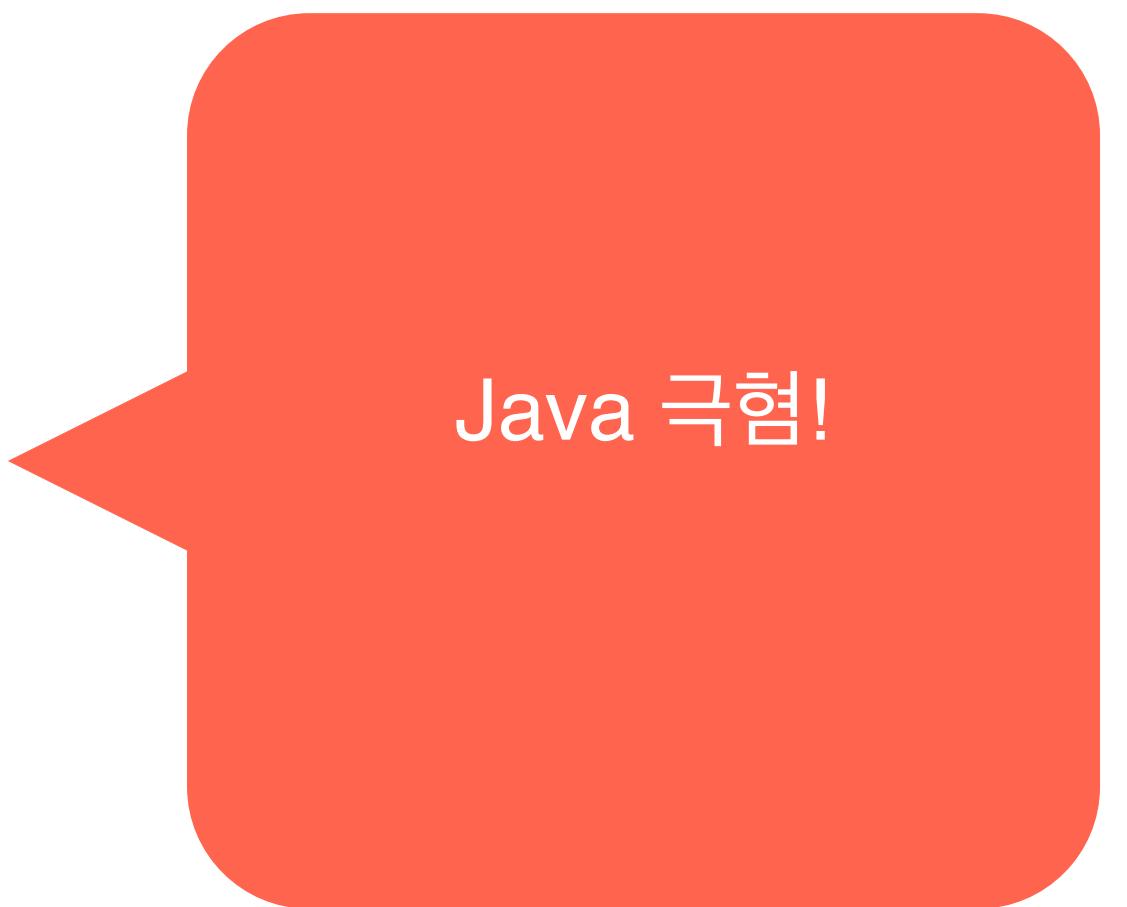


단 점

- Spring Framework 내부를 들여다 보기 어렵다.
- 개발자의 편차가 크다. 특히 우리 파트.
- JVM 튜닝은 골치 아프다.
- Oracle risk. 비용 지출 염려.
- OS 를 변경할 일이 거의 없고,
변경시 다시 compile 하는 것은 비용이 들지 않는다.
배포시 항상 자동 빌드 과정을 거침.

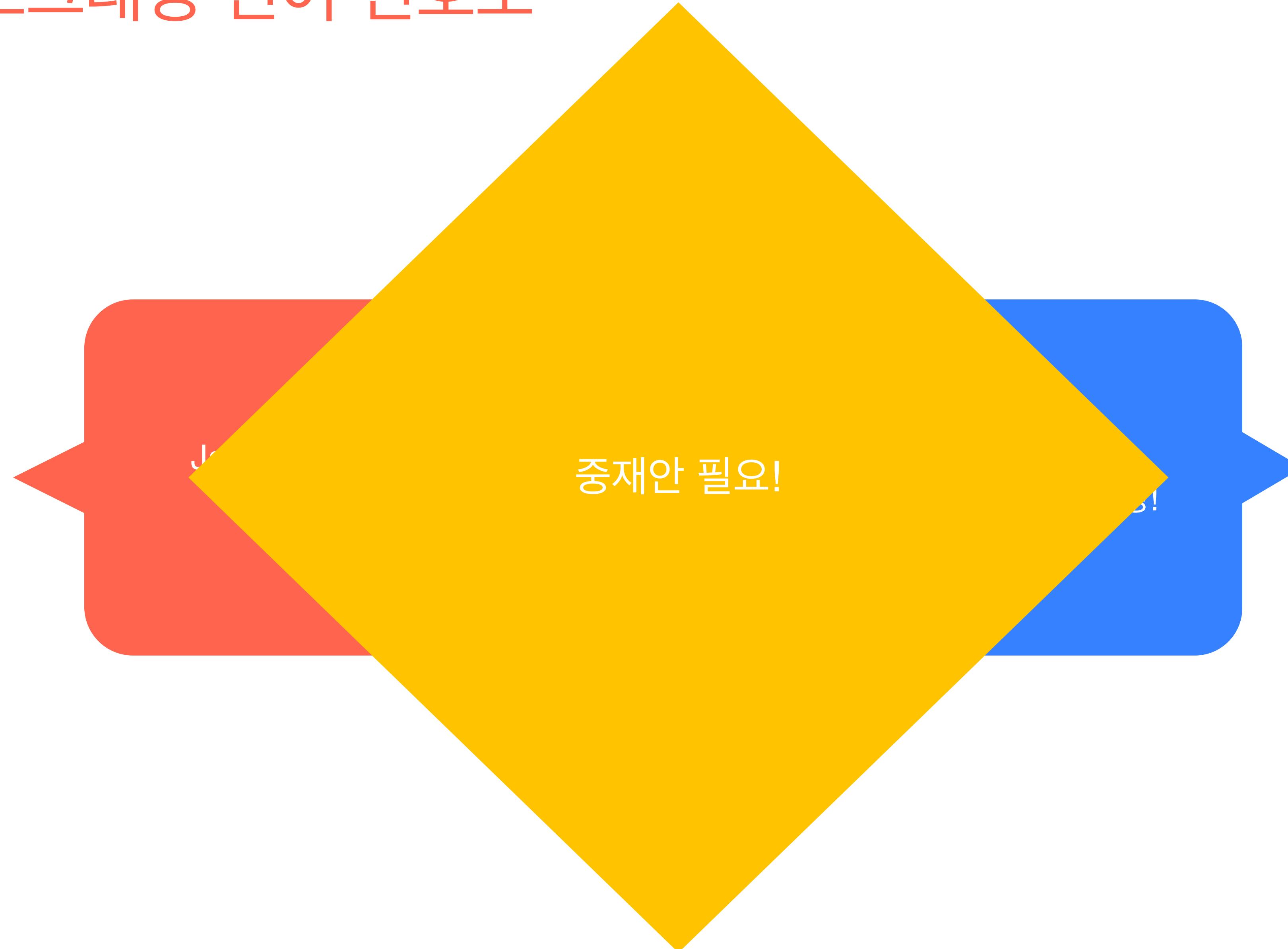
파트원 프로그래밍 언어 선호도

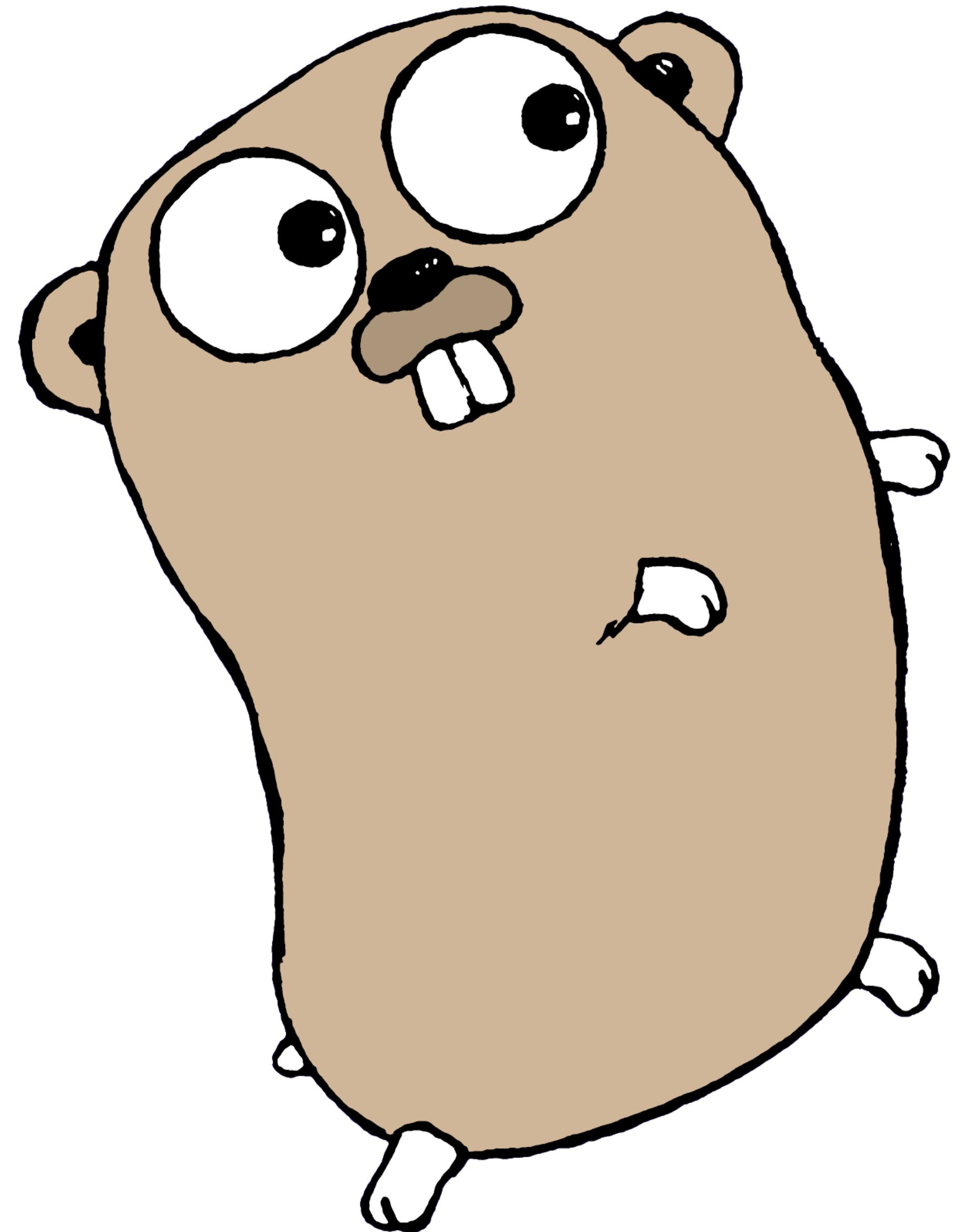
if(kakao)dev 2019



파트원 프로그래밍 언어 선호도

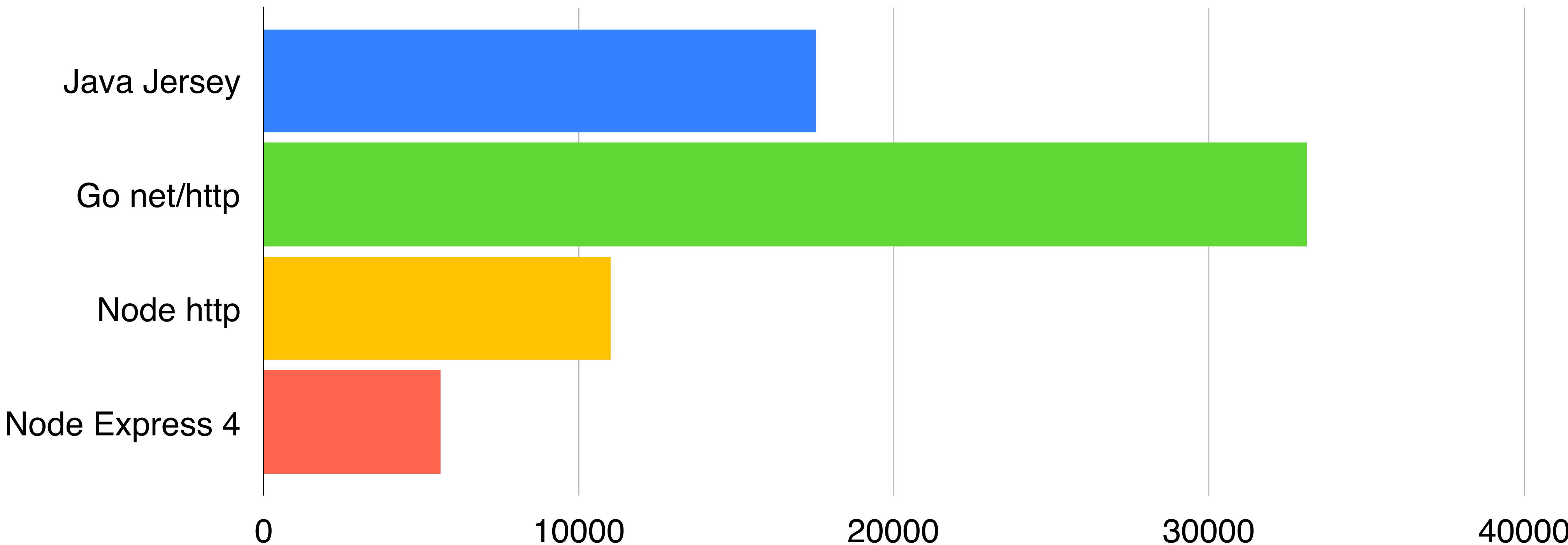
if(kakao)dev 2019





언어별 HTTP 서버 성능

if(kakao)dev 2019



* 출처: <https://muetsch.io/http-performance-java-jersey-vs-go-vs-nodejs.html>

25 vs 56

GO, Go!

if(kakao)dev 2019

- 배우기 쉽다
- 자유도가 낮은 Coding Style. 개발자간 편차 없이 거의 동일한 결과물.
- 적은 메모리 사용, 상대적으로 빠른 성능.
- Java VM 같이 무거운 VM이 없음. GC (간단한 정적 Mark and Sweep, 메모리를 옮기는 Compacting 없음)
- 온전한 Compile 언어. 경량의 정적 Compile 결과물
- No magic. 모두 들여다 볼 수 있음.
- 500배 경량의 Thread 와 Queue = Goroutine 과 Channel. 실수의 여지가 적고, 고성능.
- 직관적인 오류 조작. “예외”, “던지기” 없음.
- Container 시대의 언어. 컴파일 결과물로 단 하나의 실행 파일이 나옴.



Go 의존성 주입, FX

if(kakao)dev2019

Spring framework 의 가장 큰 장점은 “의존성 주입”

- 의존 관계 설정이 실행시에 이루어지므로 모듈간의 결합도 낮춤
- 코드의 재사용성을 높여서 작성된 모듈을 여러 곳에서 소스코드 수정 없이 사용
- 모의 객체 등을 이용한 단위 테스트의 편이성 높여줌



FX로 모두 달성 가능, go.uber.org/fx

- No XML, No Annotations
- No Magic

```
func main() {
    app := fx.New(
        fx.Provide(
            activityService.New,
            config.New,
            commandService.New,
            driverKafka.NewSyncProducer,
            health.NewServer,
            kafkaclient.New,
            kafka.NewSchemaRegistry,
            logger.New,
            metrics.NewReporter,
            metrics.NewScope,
            mongo.New,
            moveHandler.New,
            moveRepo.New,
            moveService.New,
            newrelic.New,
            policyService.New,
            tracer.New,
            zoneService.New,
            zoneService.NewClient,
        ),
        fx.Invoke(
            driverFx.RegisterGRPC,
            driverFx.RegisterMetrics,
        ),
    )
    app.Run()
}
```

04 서비스 구조

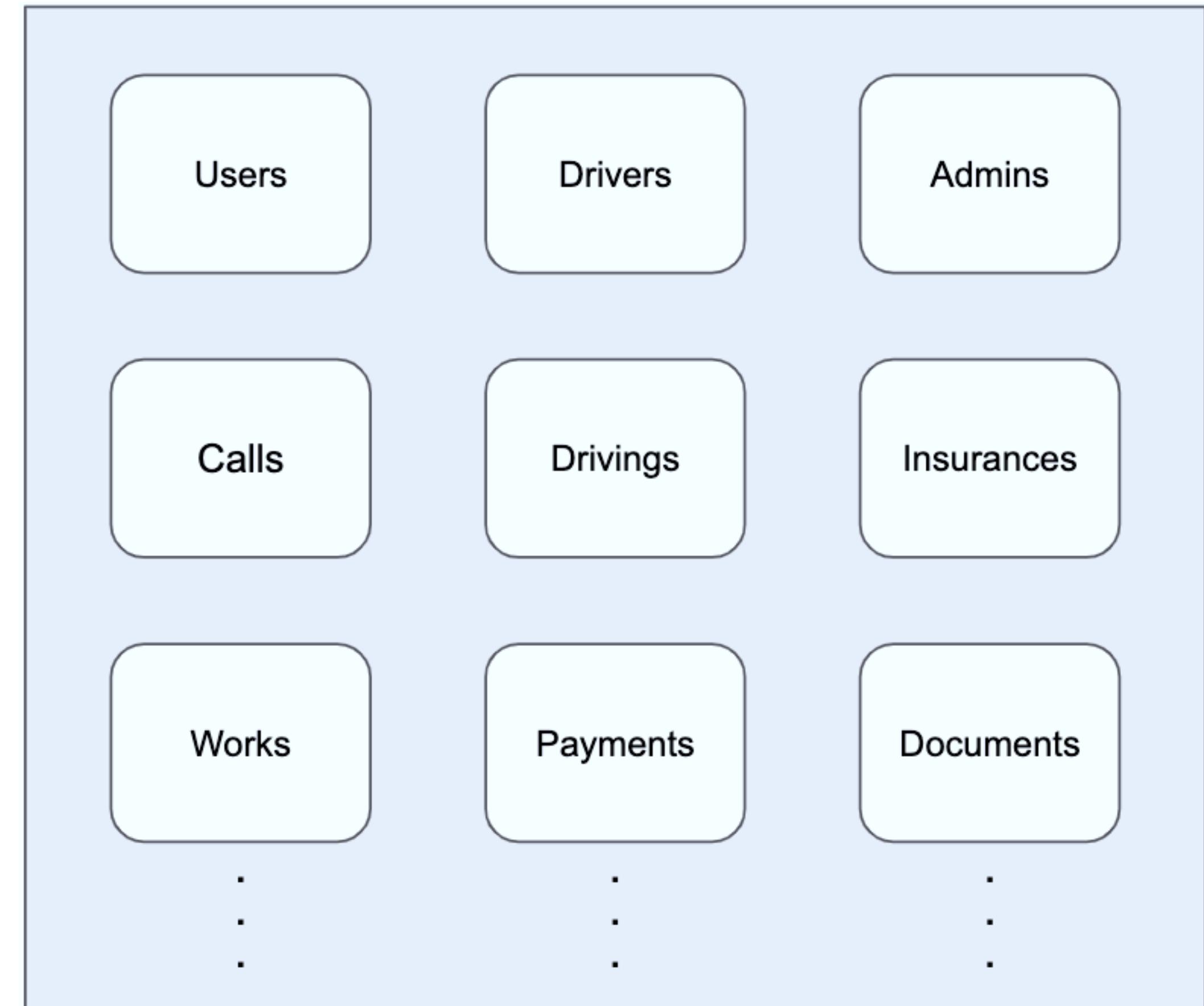
if(kakao)dev 2019

카카오 T 대리 monolith: wheel-webapp

if(kakao)dev 2019

- Ruby on Rails REST API
- 사용자, 기사, 운영 관련

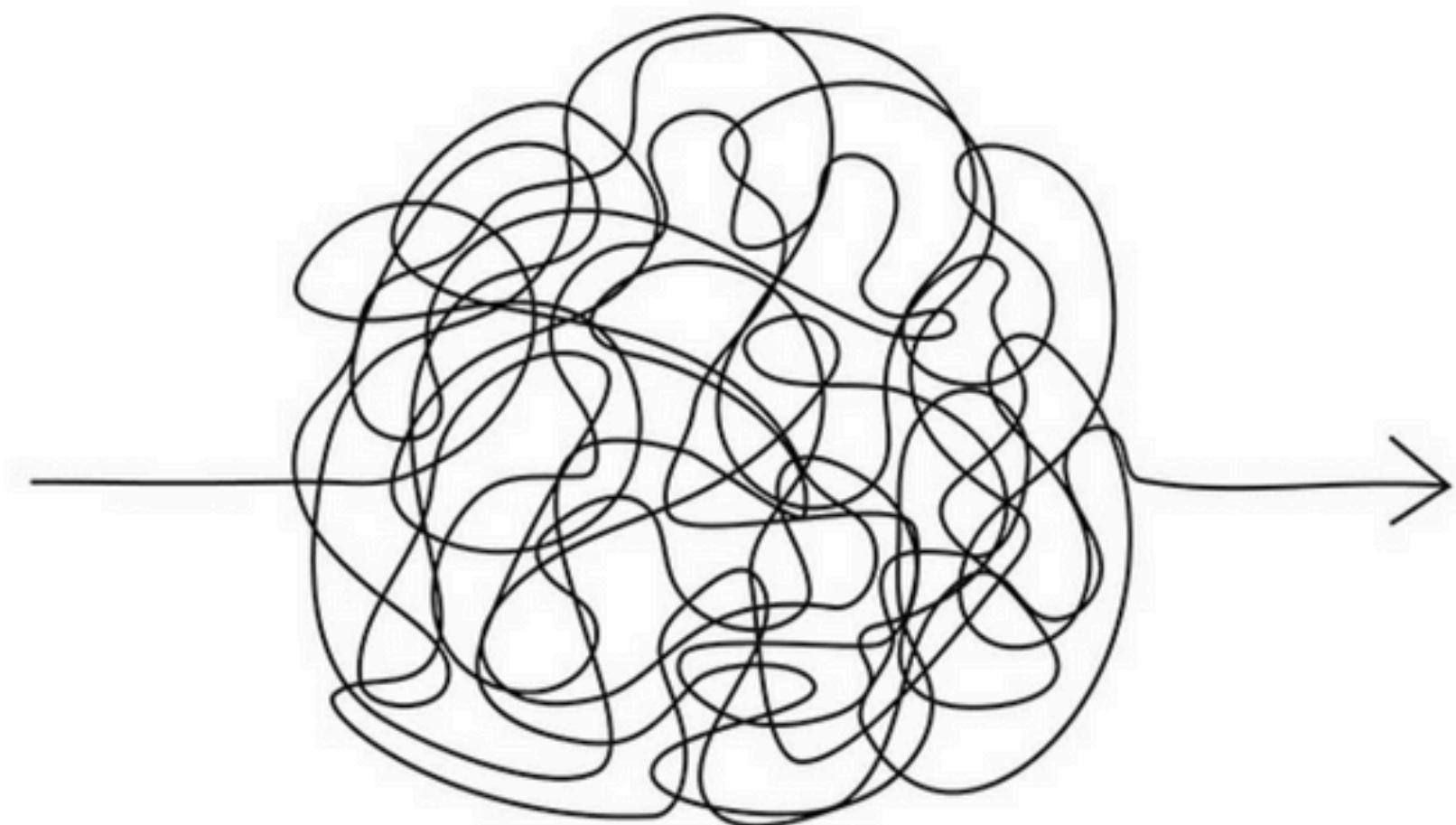
대리 제품의 주요 API와 비즈니스 로직, 웹사이트를
담고 있는 Monolithic Service



카카오 T 대리 monolith: wheel-webapp

if(kakao)dev 2019

- Number of lines of code: **269,878** (10,796 페이지 짜리 책 분량, 대법전은 5,780 페이지)
 - RoR 은 함축적이라 Line 수가 상대적으로 훨씬 적은편.
 - 그럼에도 절대적 양이 많아서 전체를 파악하는 것은 사실상 불가능.
 - 대부분의 구성요소가 유기적으로 얹혀있어 전체를 파악해야 일을 할 수 있음.
 - 고도의 집중력으로 어느 정도 파악하여 일을 진행하지만, 두뇌의 한계로 금방 잊게 됨.
- **3년 6개월간 16,369** 개의 변화
 - 과거에 했던 무언가를 찾아내려면 저 많은 변화중에 찾아내어야 함.
- **35.02%** 의 Test Coverage
 - 60% 이상의 코드는 대체 어떻게 돌아가는지 알 수 없다.
- 배포 스트레스: 상기의 이유로 배포시 큰 용기를 필요로 하여 망설이게 된다.



왜 Microservice Architecture?

if(kakao)dev 2019

- 복잡도가 높은 우리 제품을 모두 이해하는 것은 앞으로도 어려울 것이므로, 잘게 나누어 가급적 작은 부분만 책임지자.
 - 고립된 기능
 - 고립된 데이터 / 상태
- 병렬적으로 간섭없이 개발하고 싶고, 독립적으로 배포하고 싶다.
- 서비스별로 확장을 유연하게 하고 싶다. 필요한 부분만 확장.
- 자주 작은 배포를 하고 싶다.
- 기술적 개성이 강한 우리, 장점을 잘 활용하자. 다른 프로그래밍 언어, 데이터 스토리지 기술 사용. 미래에도 유리.
- 전사 차원의 DKOS라는 Cloud 지원. 이를 잘 활용하려면 Microservice Architecture 가 적절.
- 제품의 각 부분이 단순화 되어 의사소통이 원활해진다.



우리는 Microservice 로...

if(kakao)dev 2019

- Netflix, Twitter, Uber 등 많은 회사들이 Monolith 를 다수의 Microservice 로 검증
- 언어적 가변성 허용: Kotlin, Go, Node.js, Java ...
- 업계경력이 길지 않은 사람을 포함, 개발자들에게 서비스를 소유할 기회를 제공
- 하나의 온전한 코드저장소를 소유하는 것은 보상이 주어지고 가치있는 경험
- 서비스 소유는 책임감이 크지만, 카카오의 개방적, 지식 공유 문화는 새로운 기술의 집합을 선택하도록
- 신규 채용등 조직 확장 중이므로 다수의 서비스로 나누어 지는 것이 자연스러움



Microservice 단점

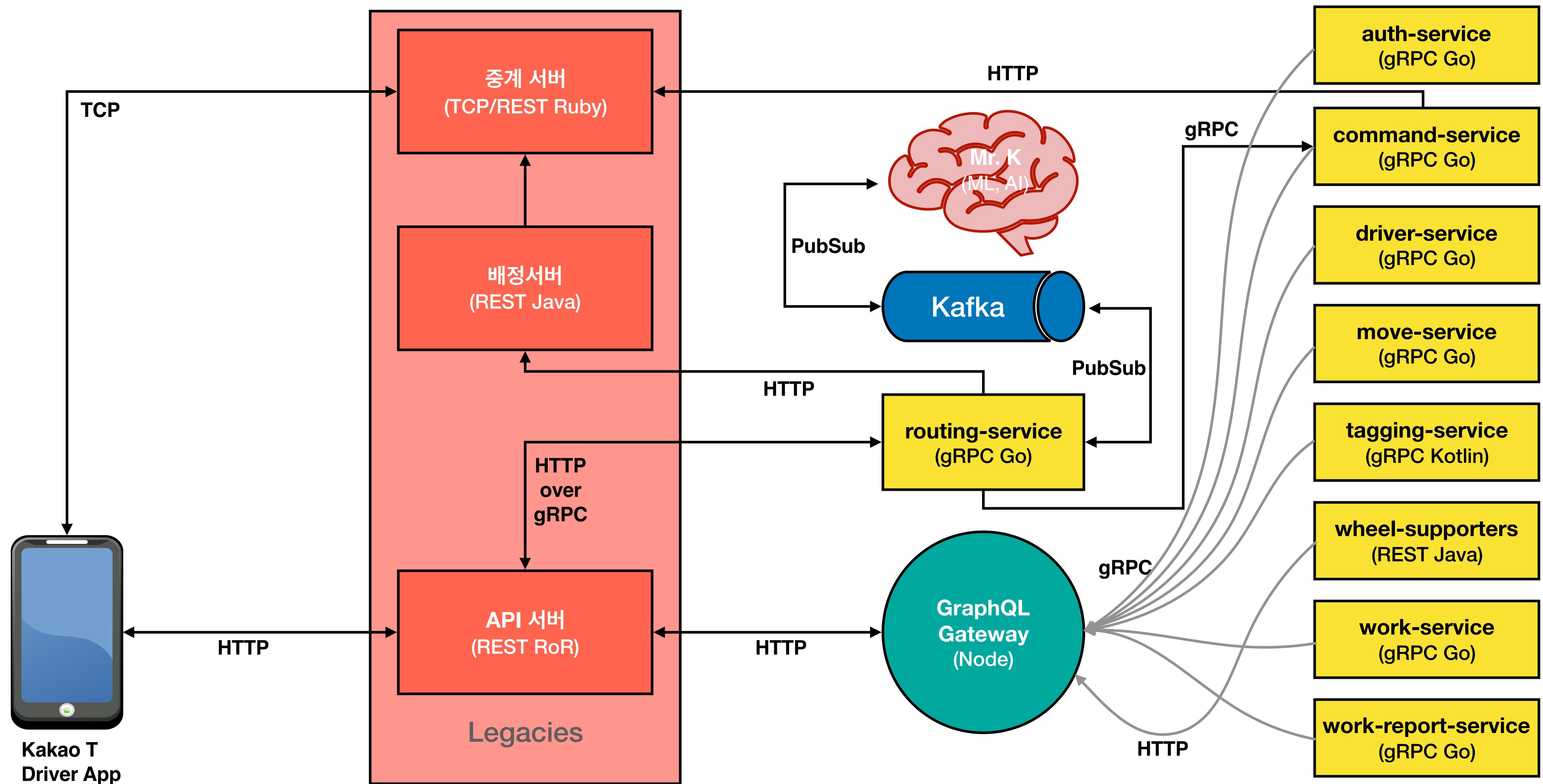
if(kakao)dev 2019

- 기존 하나의 API호출이 Microservice 에서는 다수로 늘어날 수 있음
→ API Gateway
- Microservice 간 네트워크 통신 비용
→ 대부분 DC 내부 비용. 병렬 처리를 염두에 두고 개발
- 서비스 발견이 복잡함
→ 현재는 DNS 를 사용하는데 etcd 나 consul 등으로 효율적 개선 필요
- 공통 기능에 대한 중복 발생
→ 중복기능을 library 화
- 테스팅 및 운영 복잡
→ 경험 쌓는 중. Martin Fowler 의 Testing Strategies in a Microservice Architecture 참고로...



서포터즈 서비스 구조

if(kakao)dev 2019

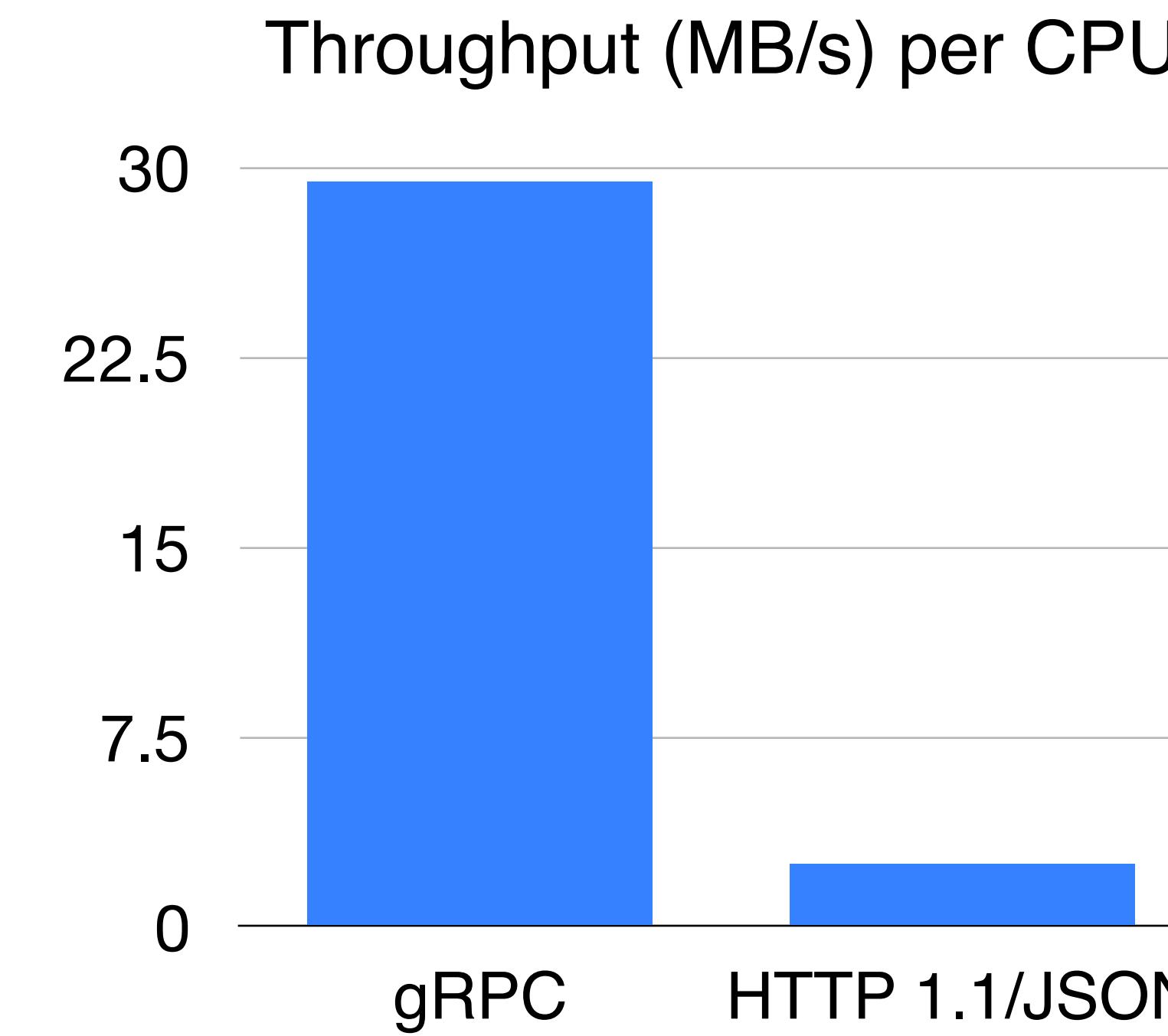
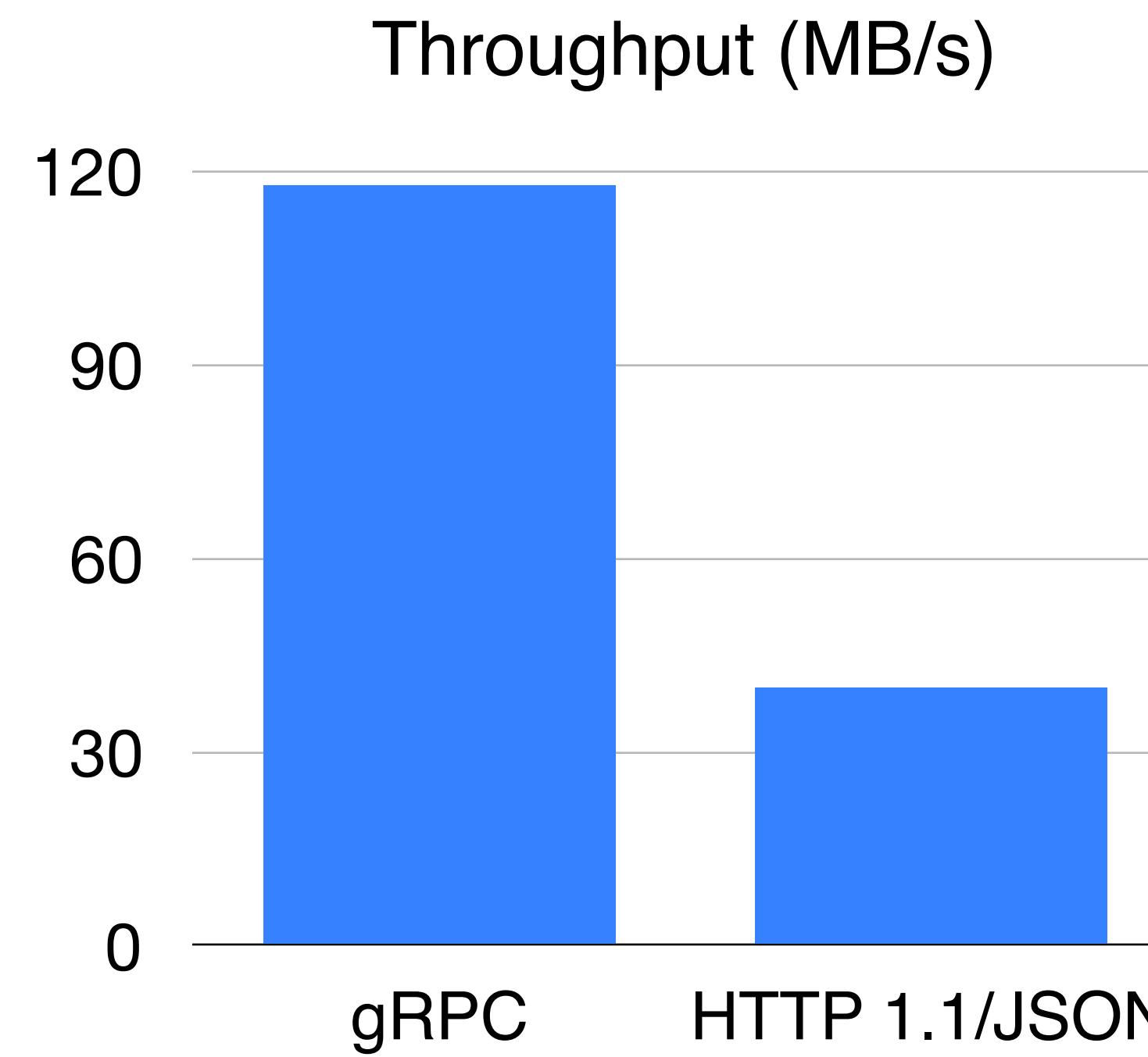


05 서비스 계층

if(kakao)dev 2019

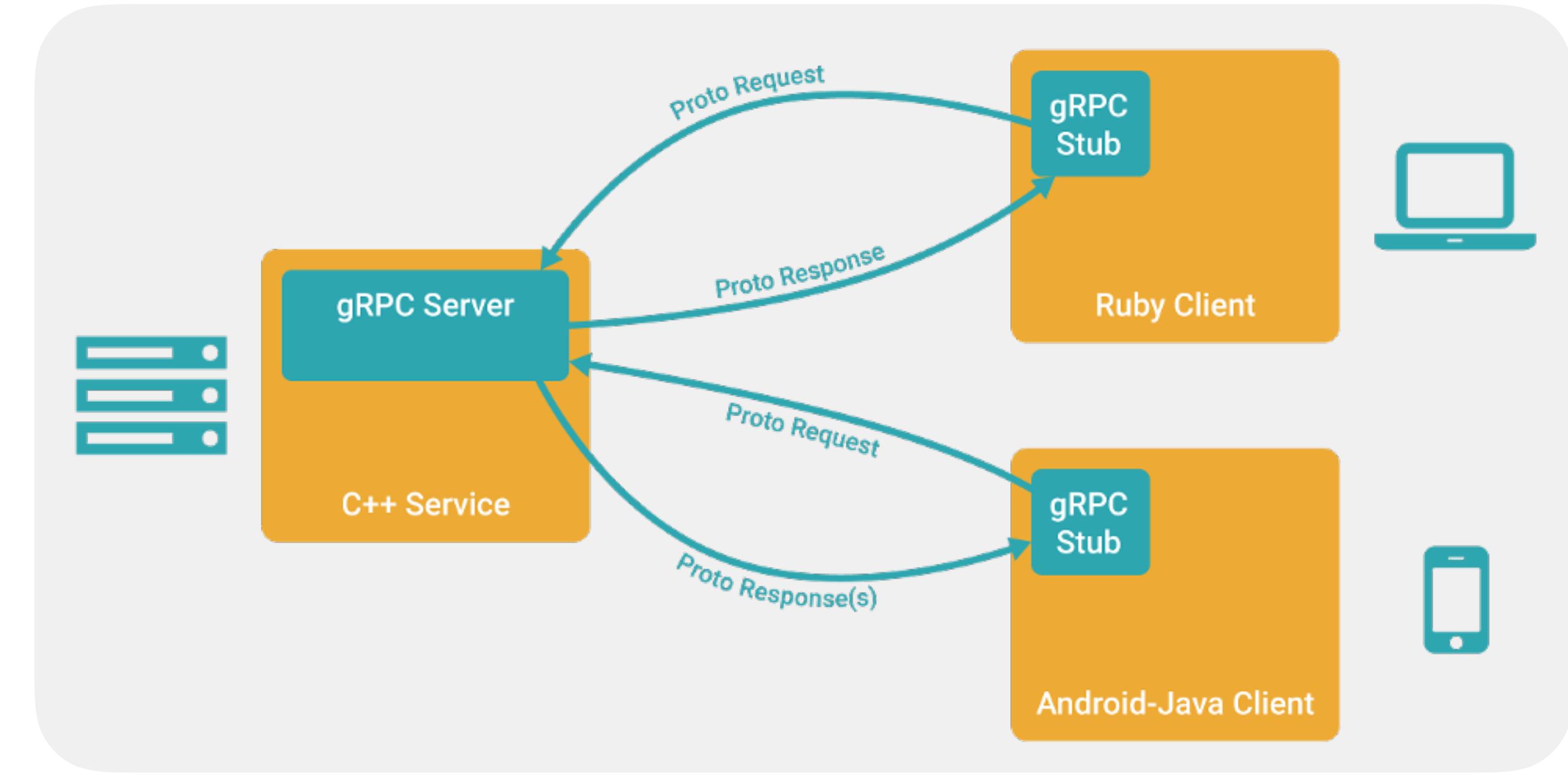
RESTful vs gRPC

if(kakao)dev 2019



gRPC

- 구글의 Stubby 의 오픈 소스 구현체
 - 1주일에 20억개의 컨테이너가 실행
 - 1초에 100억건의 RPC를 처리
- 경량의 이진 프로토콜 (Protobuf3)
 - 네트워크 효율성 증가
 - 효율적인 CPU 사용
- 서비스 호출 건수가 빠르게 증가함에 따라 매헤출마다 잘 정의된 인터페이스를 유지 하는 것이 필요
- 인터페이스를 유지하기 위해 IDL이 필요하다는 것을 깨닫고 gRPC 를 사용하기로 결정
- gRPC 는 서비스 소유자가 제약된 인터페이스 정의를 발행하는 걸 강제. 이는 서비스 통합시 필수 사항
- 요청이 인터페이스와 상이할 경우 gRPC 수준에서 거부
- 하위호환을 준수함으로서 필요시 새로운 버전에 즉시 기능 추가



gRPC UI

- <https://github.com/fullstorydev/grpcui>
- CLI 나 Client 구현 없이 웹페이지에서 gRPC를 테스트 해볼 수 있음
- Postman 과 유사하나 REST가 아닌 gRPC API 용
- Docker 로 bundle 하여 모든 gRPC 서비스에 쉽게 통합

gRPC Web UI
Connected to *localhost:50051*

Service name: work.Work ▾
Method name: Begin ▾

Request Form Raw Request (JSON) Response

Request Metadata

Name	Value
X	
+ Add item	

Request Data

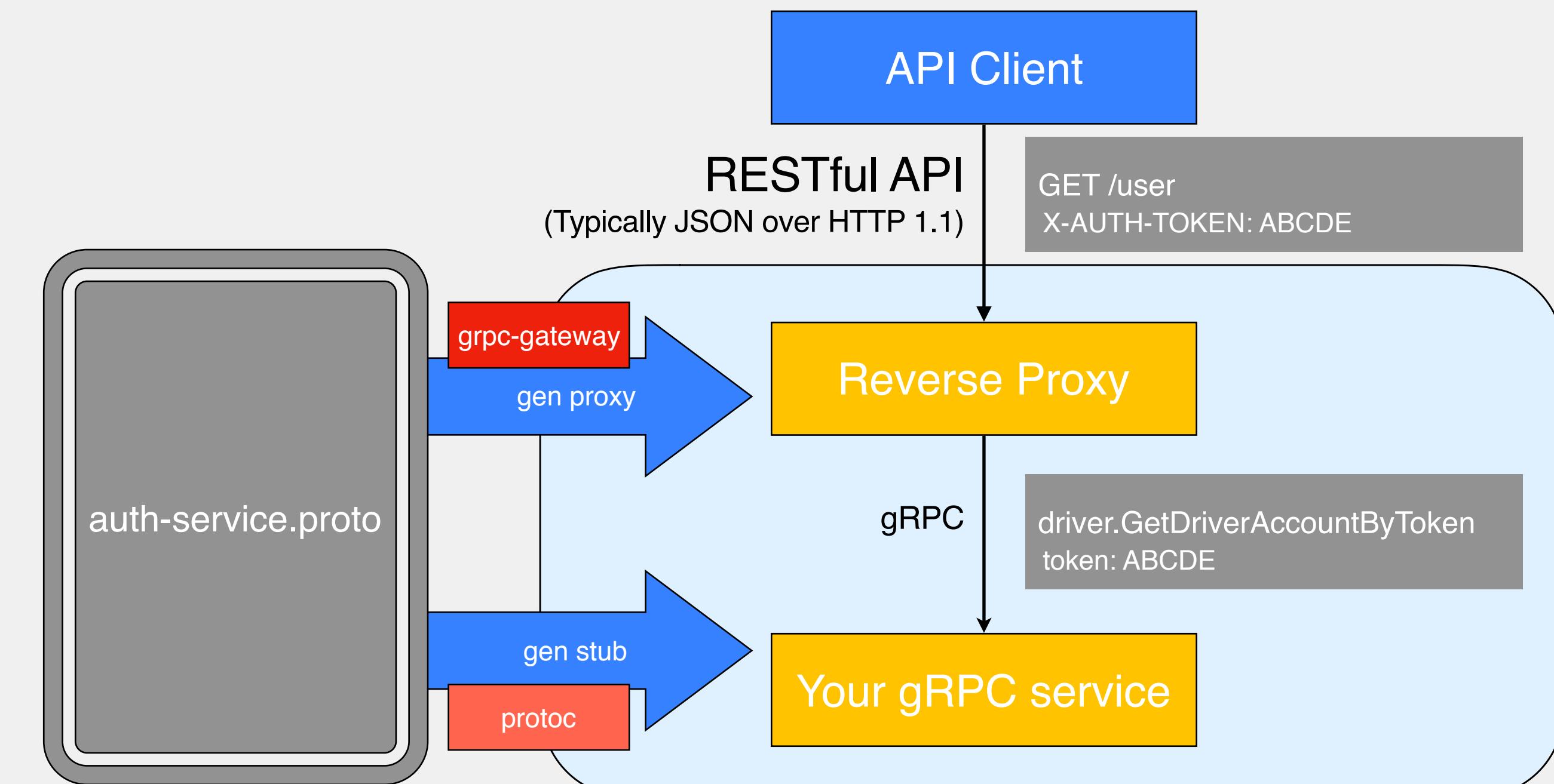
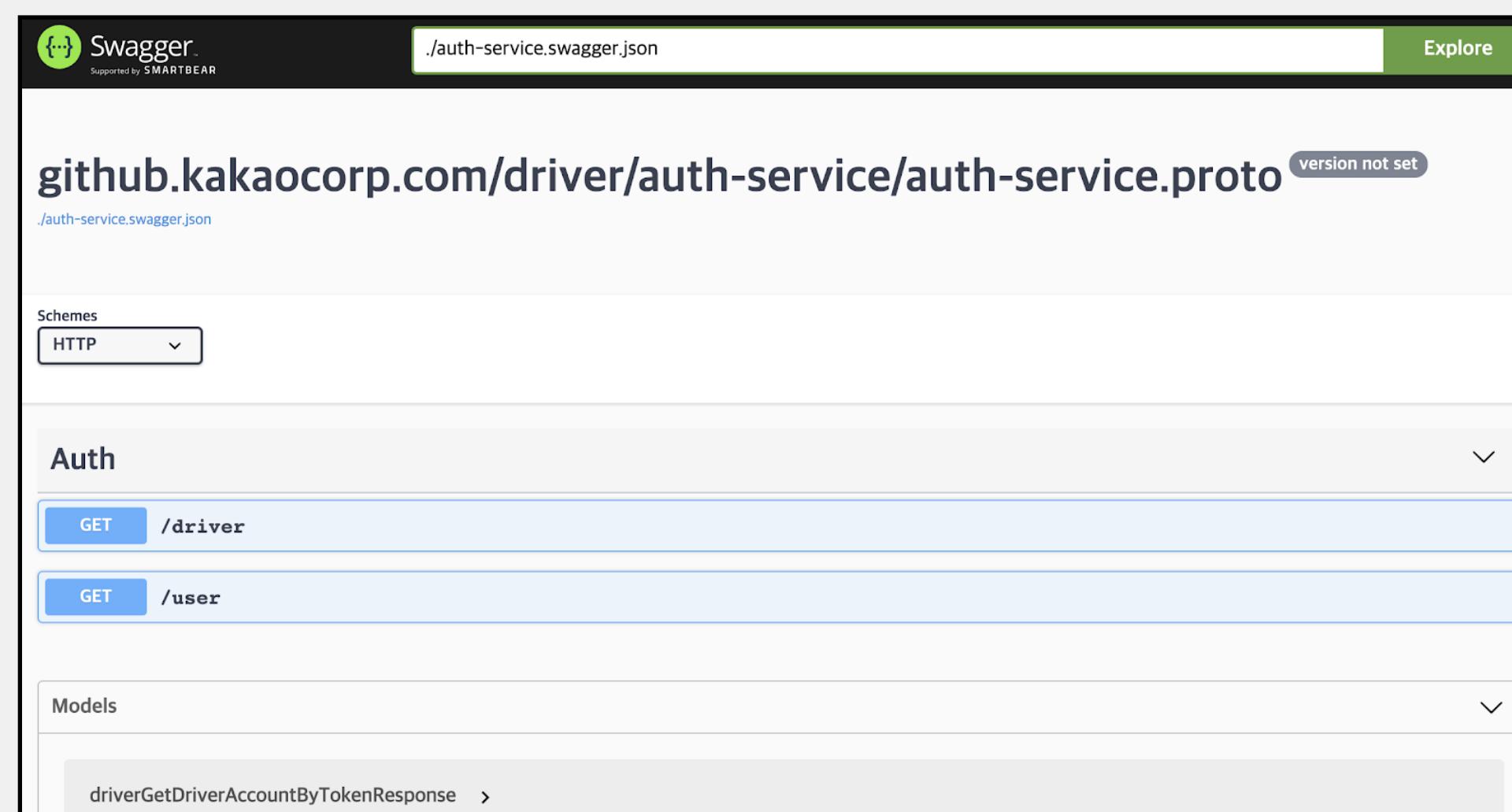
work.BeginRequest

product types.Product	<input type="radio"/> UNKNOWN_PRODUCT ▾
work_type work.WorkType	<input type="radio"/> UNKNOWN_WORK ▾
driver_id string	<input type="radio"/> <input type="text"/>
location types.Coord	<input type="radio"/> unset

gRPC Gateway

if(kakao)dev 2019

- <https://github.com/grpc-ecosystem/grpc-gateway>
- REST 를 선호하는 Client 를 위해 간단한 설정만으로 REST Proxy 제공 가능
- 현재는 Go 언어 구현체만 지원함



IDL Registry

- 서비스에 대한 정의가 서비스와 공존하기 위해 gRPC proto 파일들을 중앙화
- 각 서비스는 정적인 proto 정의에 대한 사본을 가져야 함
실행 시간에 변경된다면, 인터페이스 불일치로 프로세스가 죽을 수 있음
- 이 세상에 딱 하나의 버전 체계만 존재해야 함
gRPC IDL 파일은 현 버전의 모든 서비스가 궁극적으로 하나의 버전화된 뮤음으로 취급되어야 함
이는 모든 서비스 정의의 레지스트리
- 기본적으로 github repository
- IDL 명령행: npm install idl

06 테스트는 어떻게

if(kakao)dev 2019

부하 테스트

if(kakao) dev 2019

ghz 로 gRPC Load Test

Summary:

Count: 200
Total: 181.57 ms
Slowest: 69.60 ms
Fastest: 26.09 ms
Average: 32.01 ms
Requests/sec: 1101.53

Response time histogram:

26.093 [1]	
30.444 [52]	
34.794 [78]	
39.145 [40]	
43.495 [1]	
47.846 [0]	
52.196 [2]	
56.547 [5]	
60.897 [3]	
65.248 [2]	
69.598 [2]	

Latency distribution:

10% in 28.48 ms
25% in 30.08 ms
50% in 33.23 ms
75% in 35.43 ms
90% in 38.89 ms
95% in 55.45 ms
99% in 69.60 ms

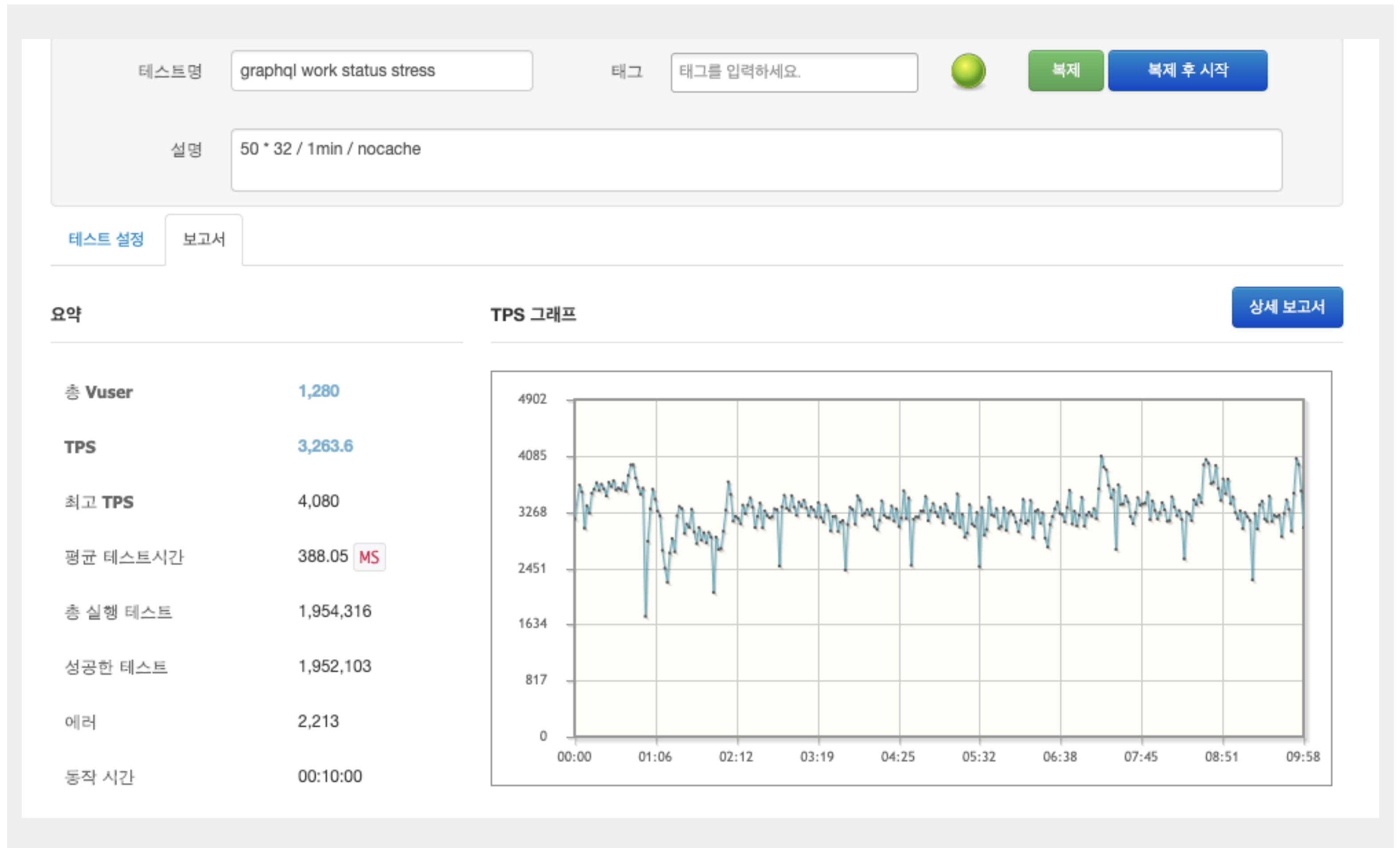
Status code distribution:

[Unavailable]	3 responses
[PermissionDenied]	3 responses
[OK]	186 responses
[Internal]	8 responses

Error distribution:

[8] rpc error: code = Internal desc = Internal error.
[3] rpc error: code = PermissionDenied desc = Permission denied.
[3] rpc error: code = Unavailable desc = Service unavialable.

nGrinder로 HTTP Load Test

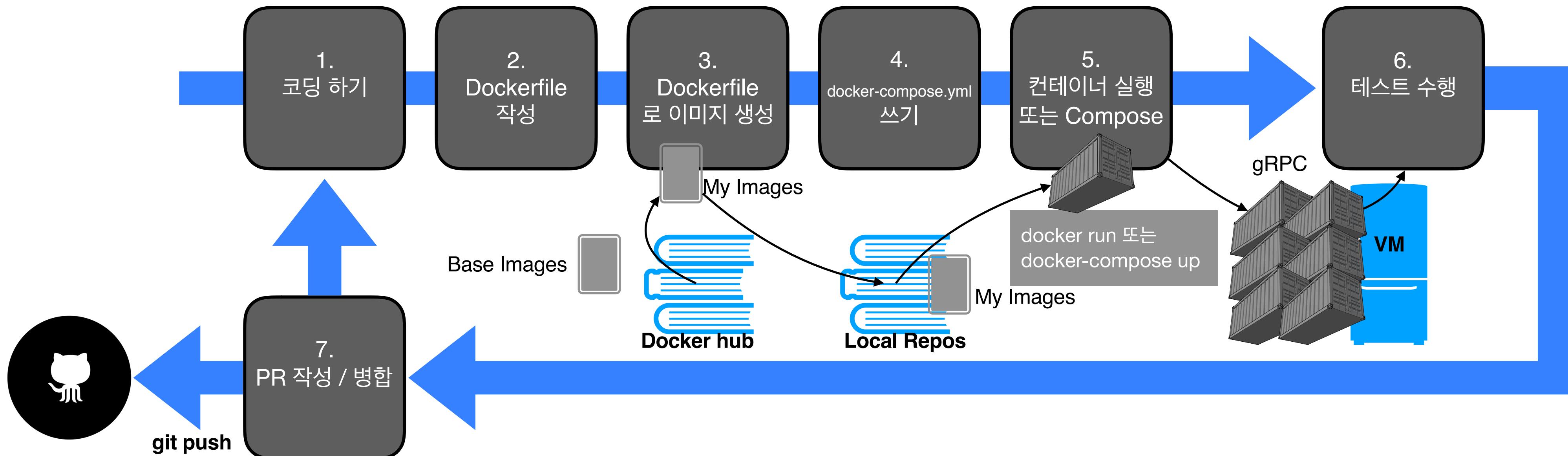


통합 테스트

if(kakao)dev 2019

Docker compose

- 의존성 있는 시스템/서비스들을 docker-compose.yaml 로 packaging
- Dockerize 되지 않은 서비스들은 테스트 서버와 통신 (연내에 모두 Dockerize 계획)



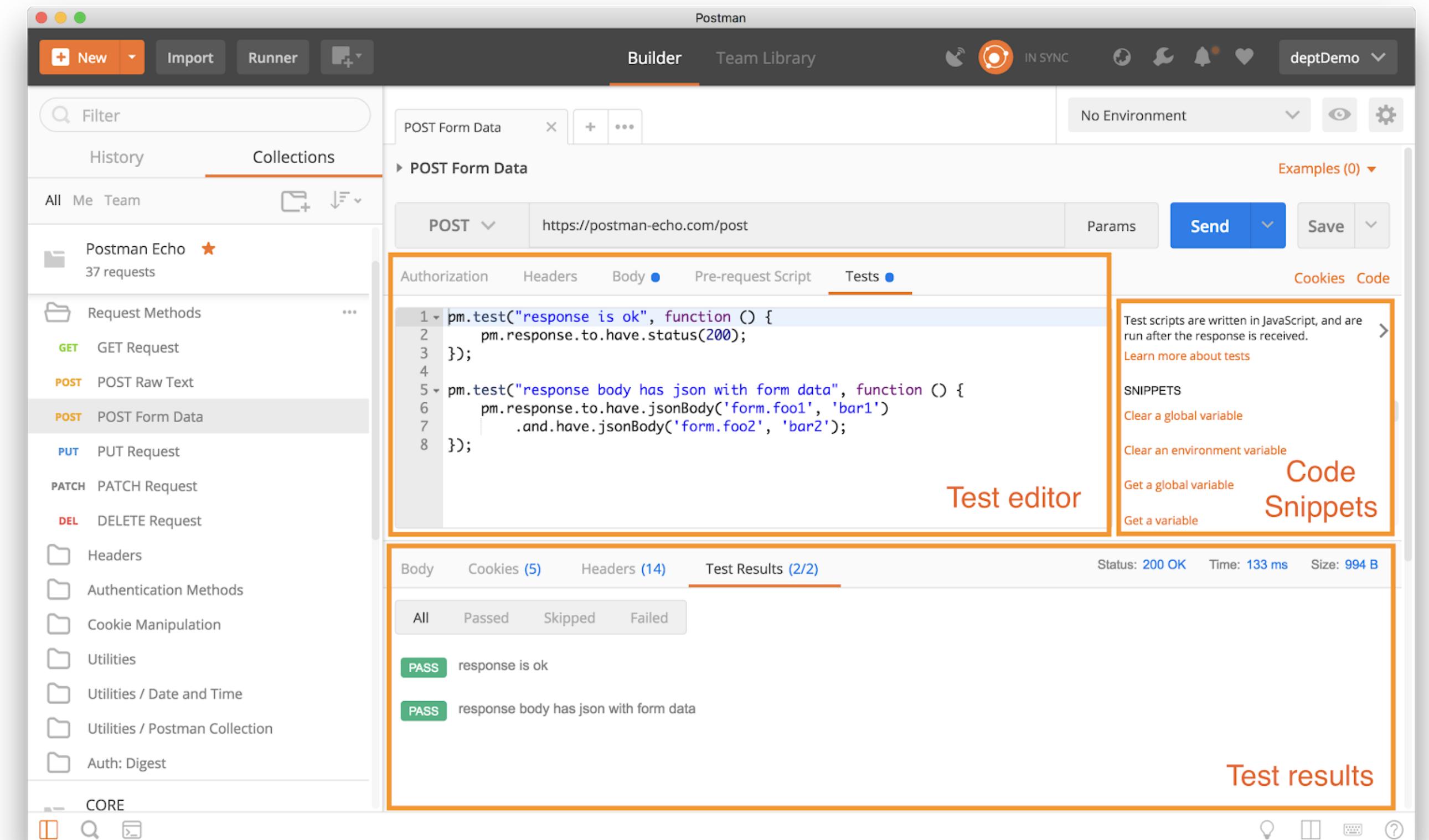
* 출처: <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/docker-application-development-process/docker-app-development-workflow>

통합 테스트

if(kakao)dev 2019

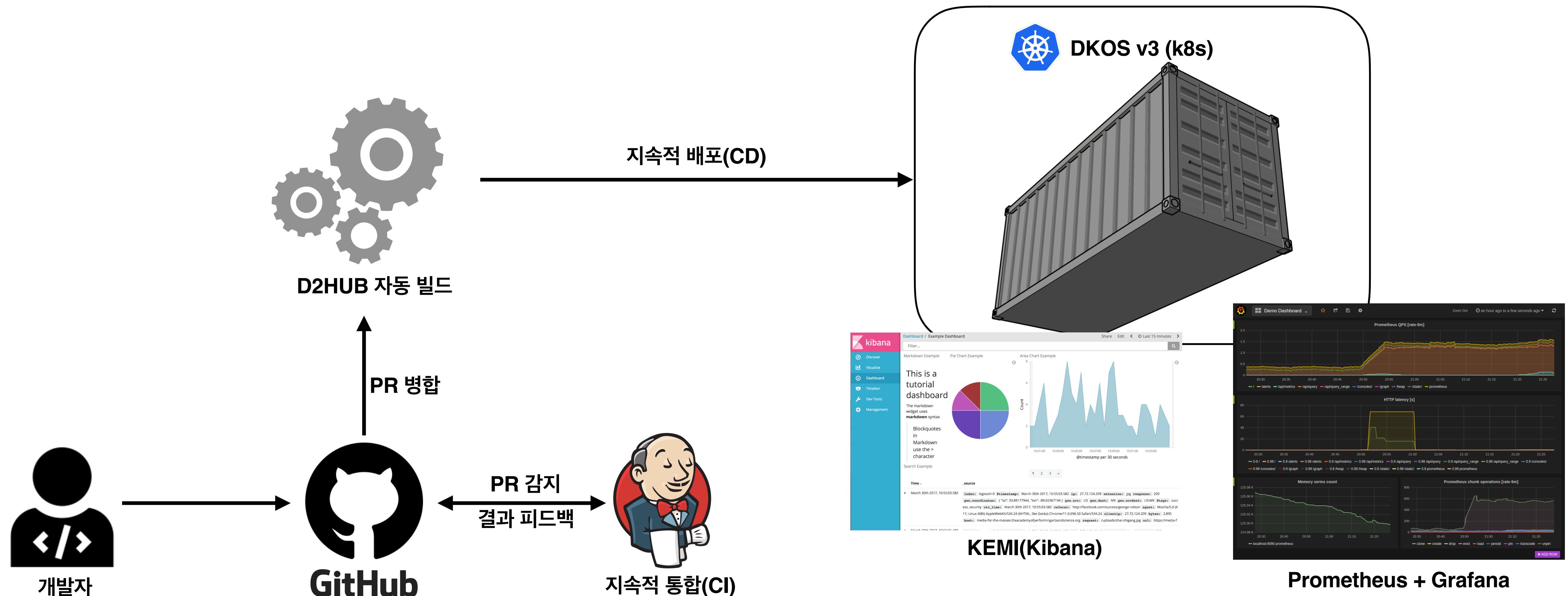
Postman

- Docker Compose로 실행된 서비스를 Postman을 통하여 통합 테스트 실행
- 통합테스트는 API로 trigger되므로 HTTP protocol을 지원하는 것으로 OK
- 이미 다수의 사내 개발자가 Postman client를 사용 중
- 사내 공통 CI와 쉬운 통합
- mock server 제공으로 Client 개발자와 쉬운 협업
- OS 중립(Mac, Win, Linux, Chrome 확장 제공)
- collection testing은 newman으로 향후 내재화 가능
- RESTful API 외에 다른 시스템(데이터베이스 등)과 통합테스트가 어려운 것은 단점



배포, 운영 환경

if(kakao)dev 2019



도전들

if(kakao)dev 2019

- 소수의 팀이라 지능적 배정 개발을 위해 AI파트와 협업을 하기로 결정. 서로간의 이해가 부족한 상황이라 어떻게 협업해야 할지가 고민
- 많은 수의 Microservice를 어떻게 클라이언트 개발쪽에 효율적으로 열어야 할까? 인증, 로깅 등의 공통기능들은 어떻게 해야 하나
- 개발일정이 촉박해서 일단 오픈했으나 장애로 서비스를 중단하였다. 다시 오픈하기 까지 부족했던 요소?
- 앞서 보았듯이 기존 시스템으로부터 입/출력이 존재하고 데이터도 교환해야 한다. 어떻게 효율적으로 할까



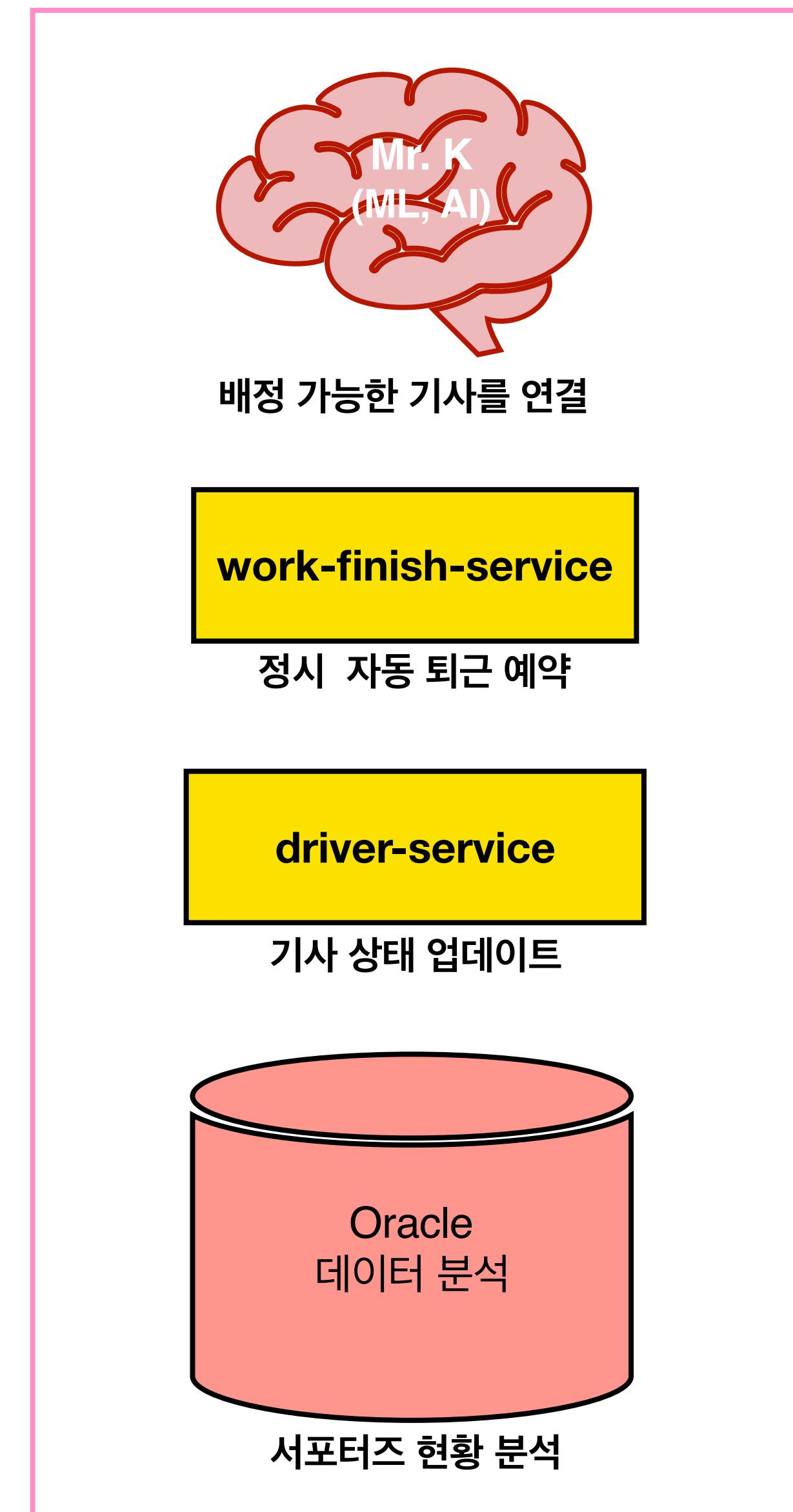
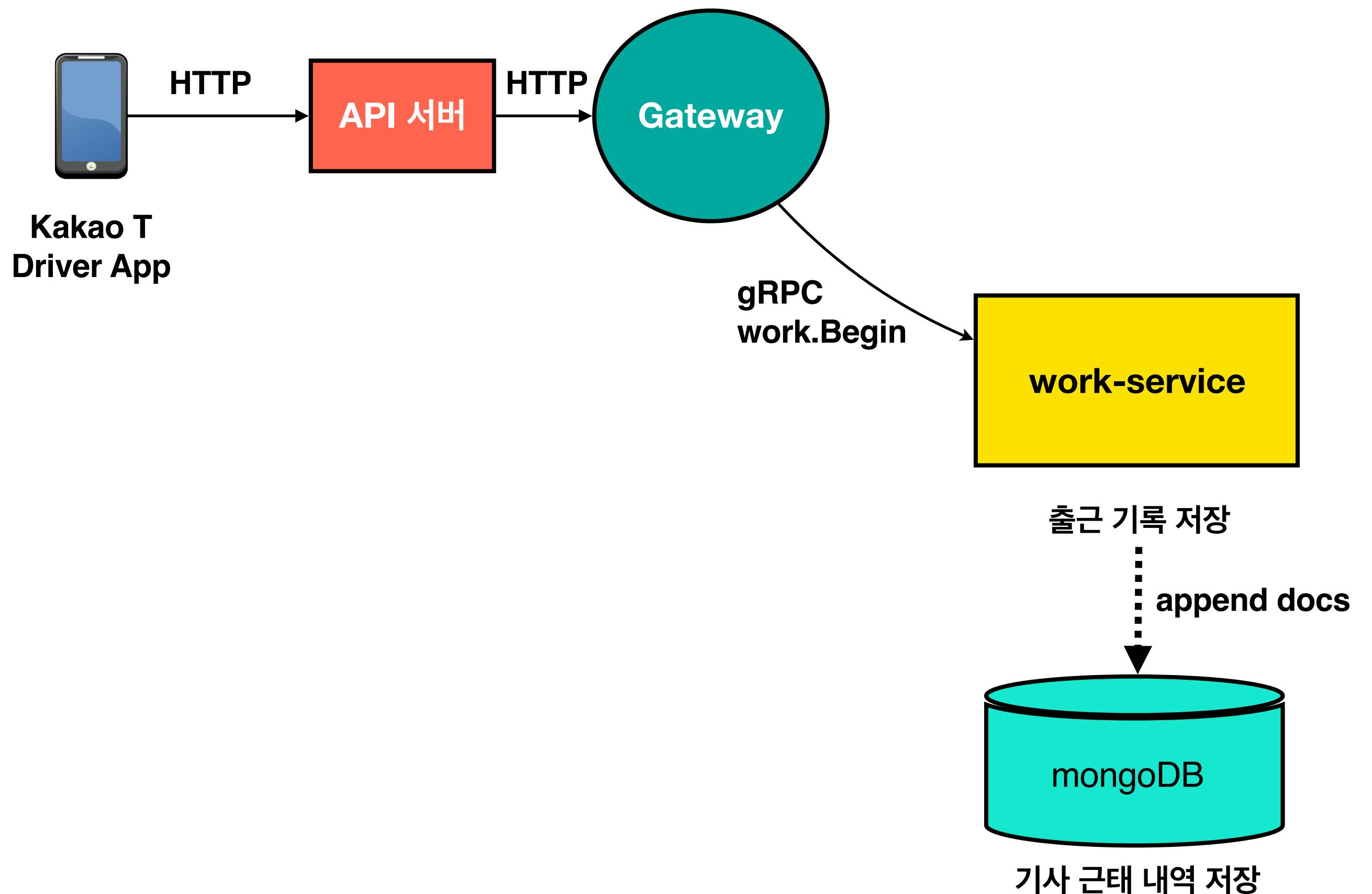
07 if kafka

if(kakao)dev 2019

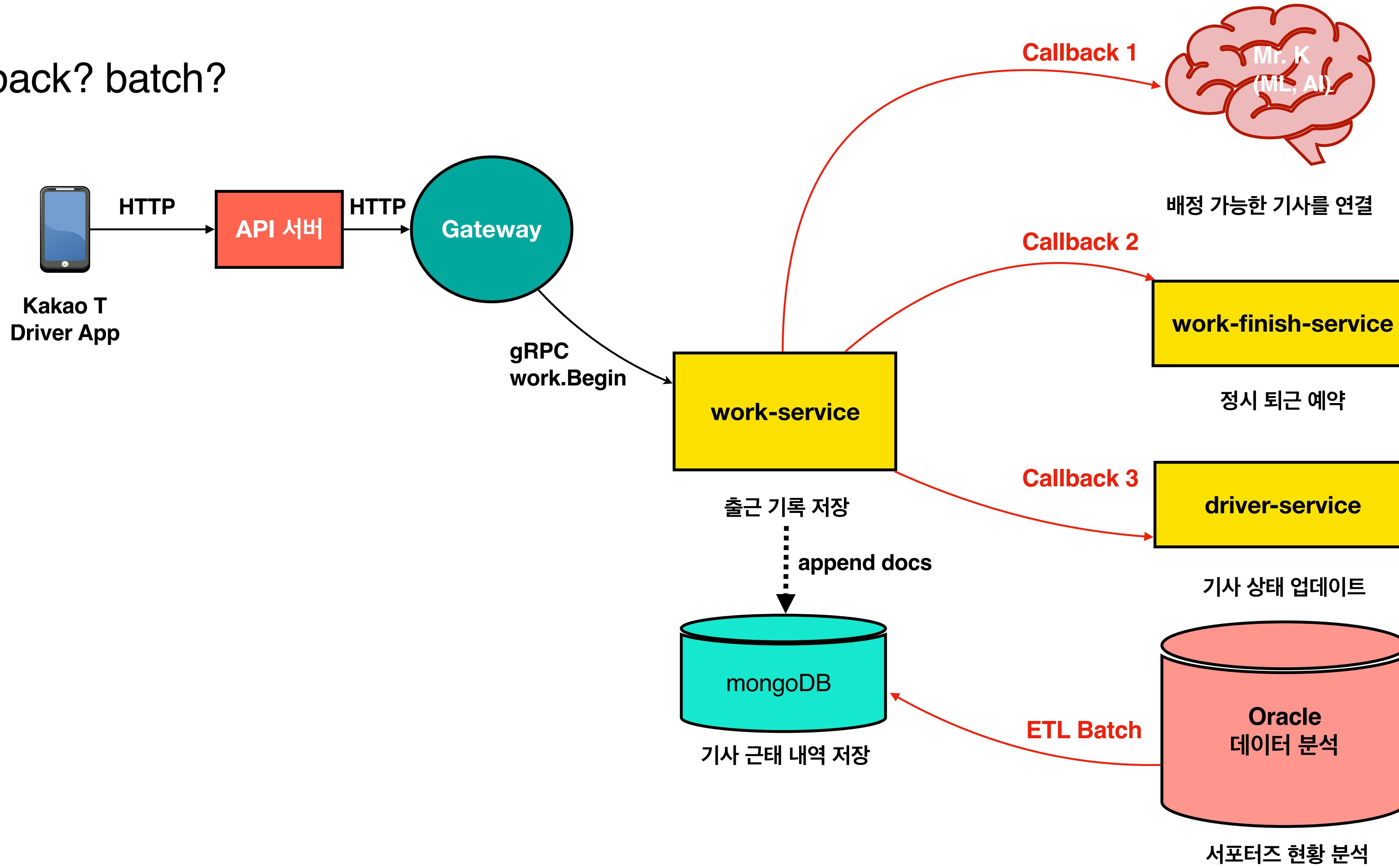
많아진, 그리고 복잡해진 서비스들

if(kakao)dev 2019

서포터즈 기사가 출근할 때 상태를 알아야 하는 서비스들...



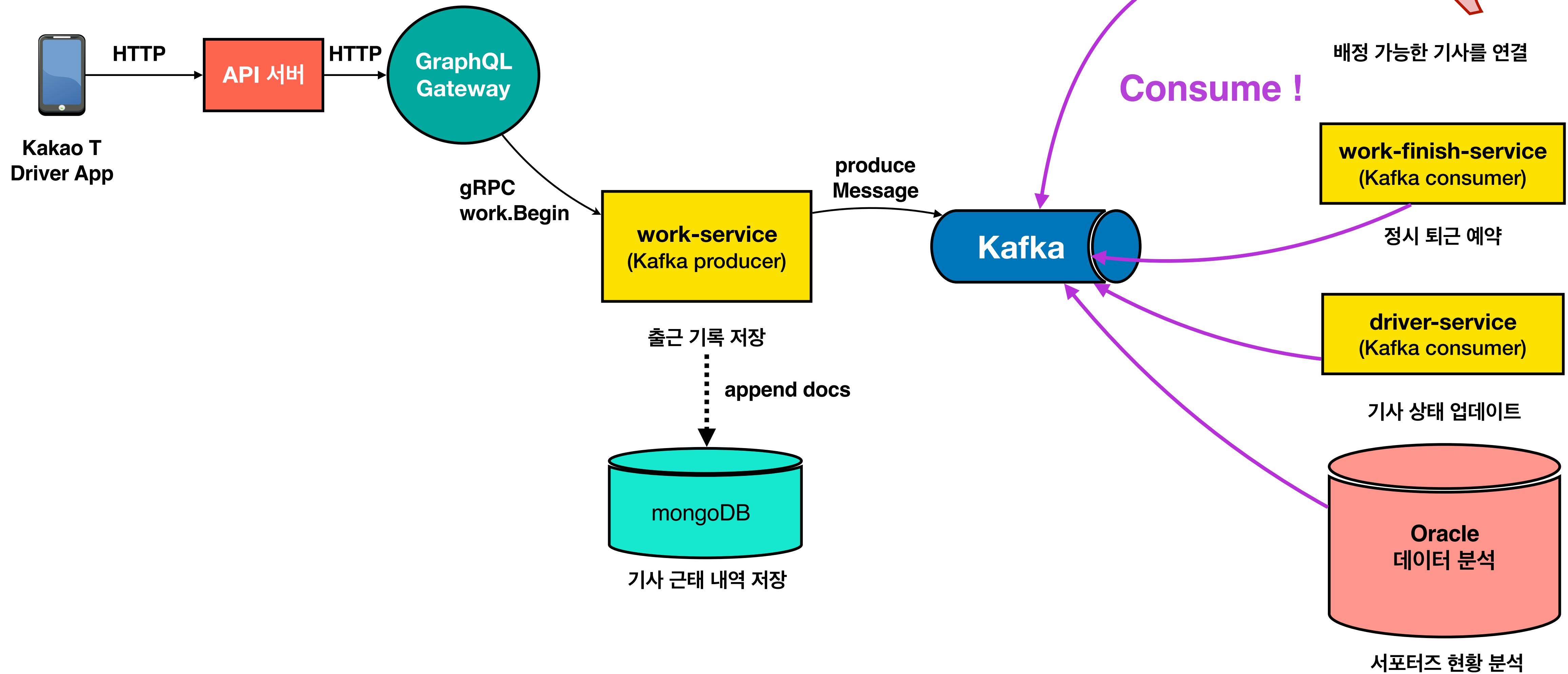
callback? batch?



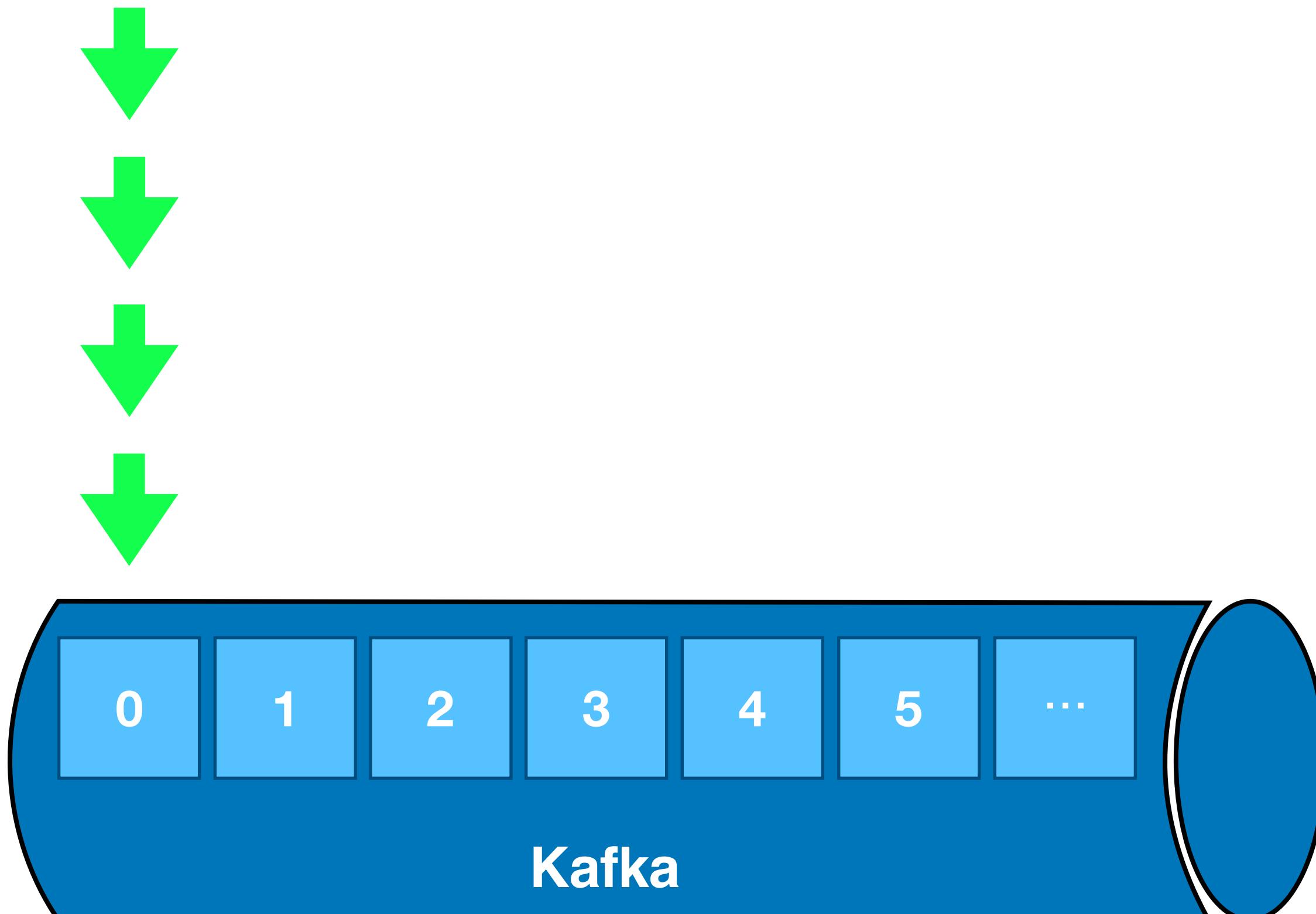
if kafka

if(kakao)dev 2019

Consume !

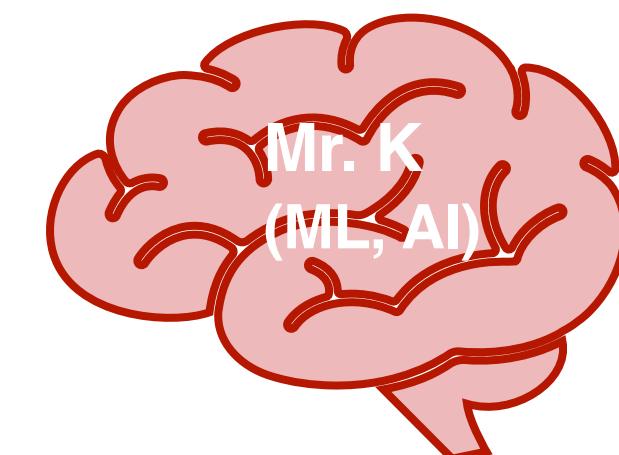


Consumer 의 메시지 처리 상태에 따른 Offset



기사 출근 메시지 Topic

Consumer Group 1



배정 가능한 기사를 연결

Consumer Group 2



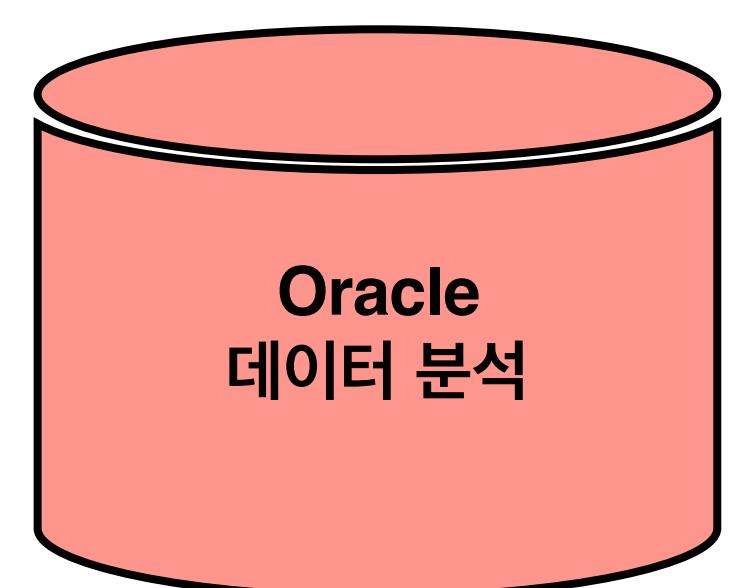
정시 퇴근 예약

Consumer Group 3



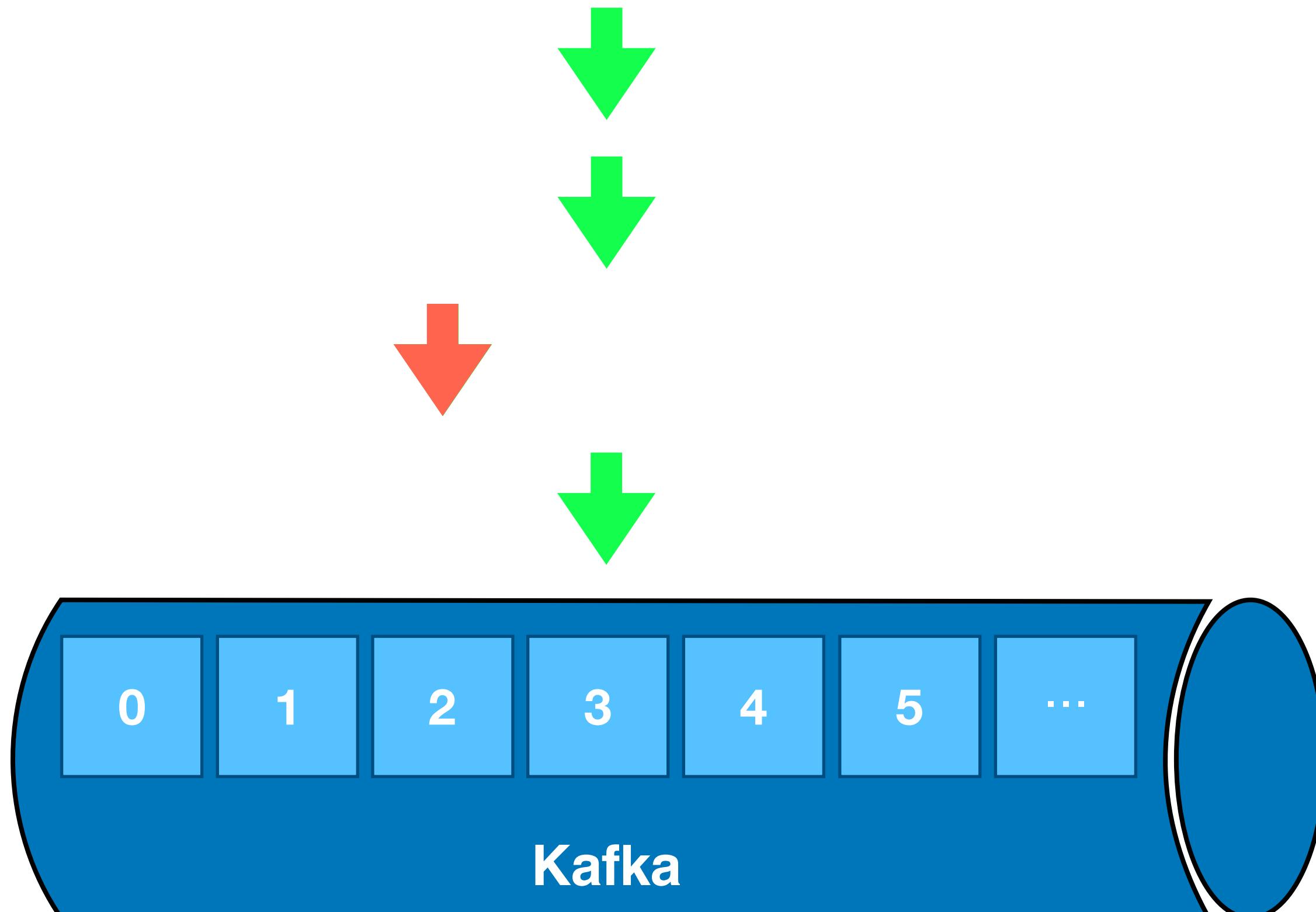
기사 상태 업데이트

Consumer Group 4



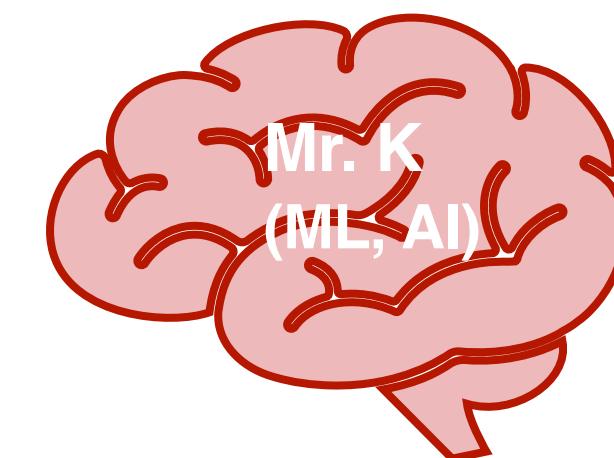
서포터즈 현황 분석

Consumer 의 메시지 처리 상태에 따른 Offset



기사 출근 메시지 Topic

Consumer Group 1



배정 가능한 기사를 연결

Consumer Group 2



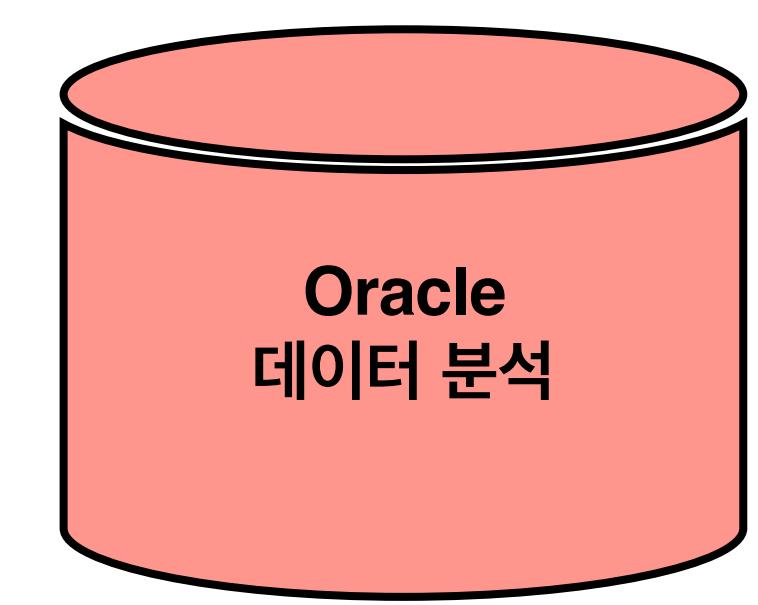
정시 퇴근 예약

Consumer Group 3



기사 상태 업데이트

Consumer Group 4



서포터즈 현황 분석

(MSA에서) Kafka 사용으로 얻은 이점

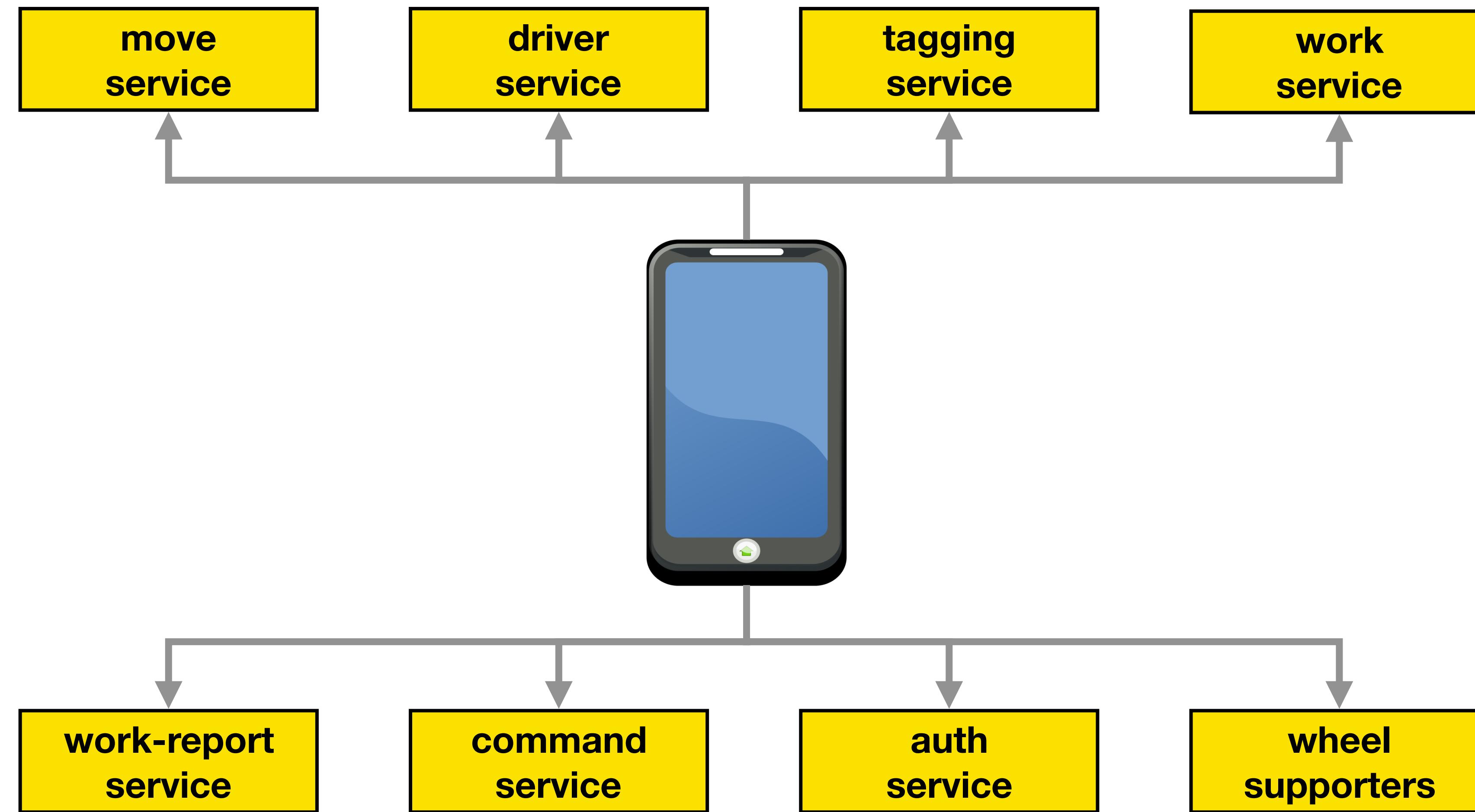
- 서비스 간 메시지 흐름 단순화
- 메시지의 영속성을 보장, 유실 방지
- 패킷 오버헤드 감소
- 자유로운 메시지 처리
- 하나의 이벤트 메시지를 동시에 여러 서비스에서 소비
- 메시지 버저닝

08 API Gateway

if(kakao)dev 2019

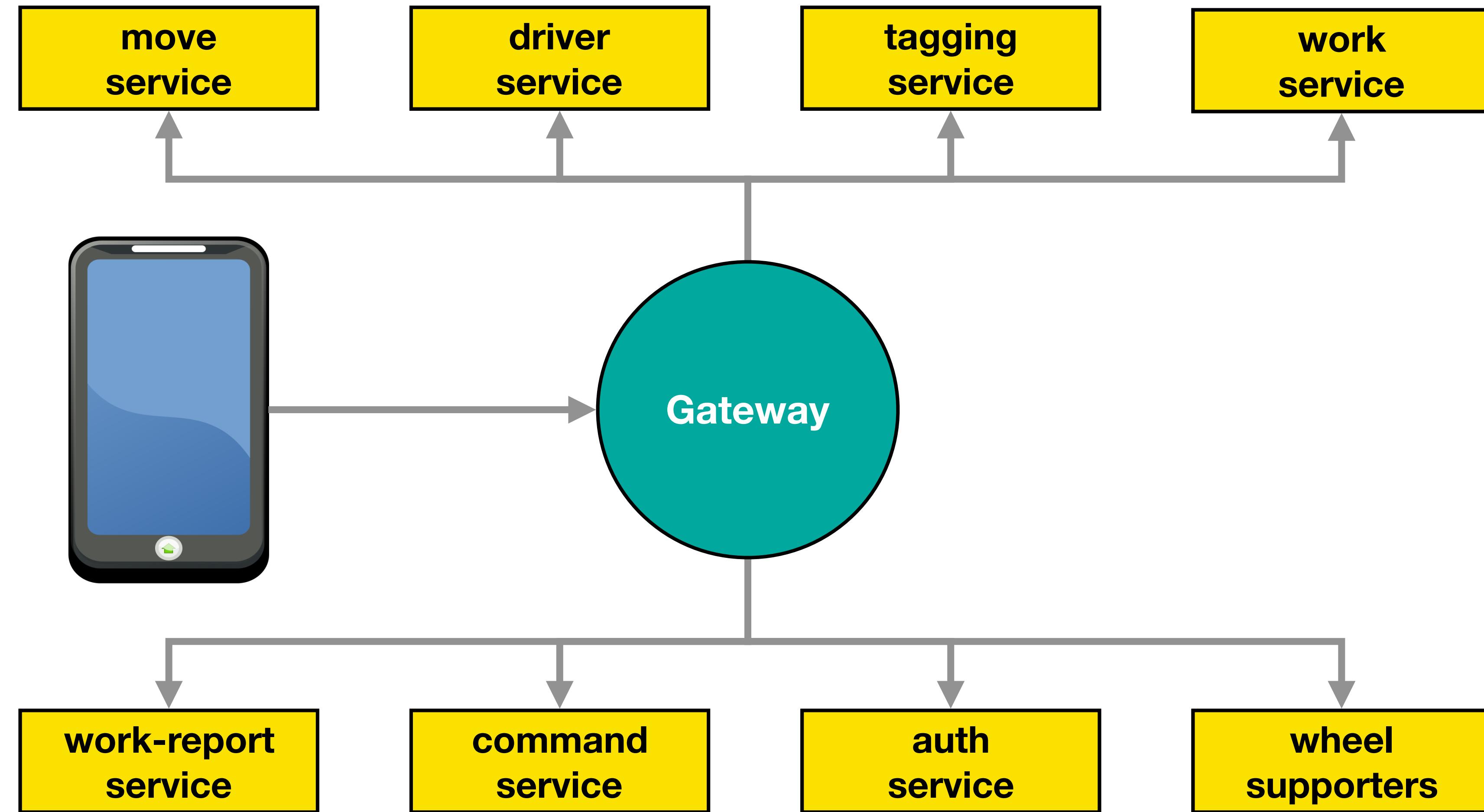
How to Deliver

if(kakao)dev 2019



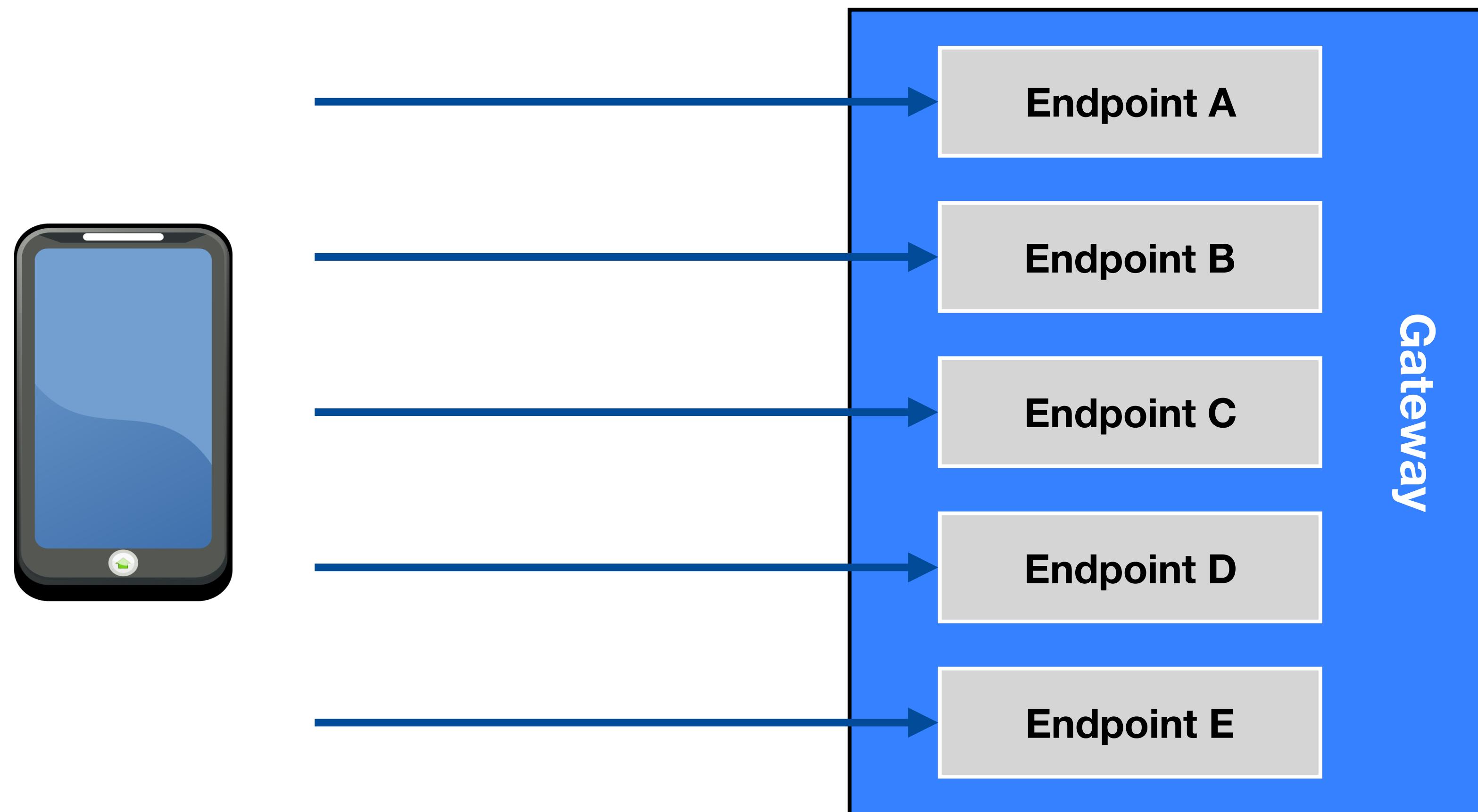
How to Deliver

if(kakao)dev 2019



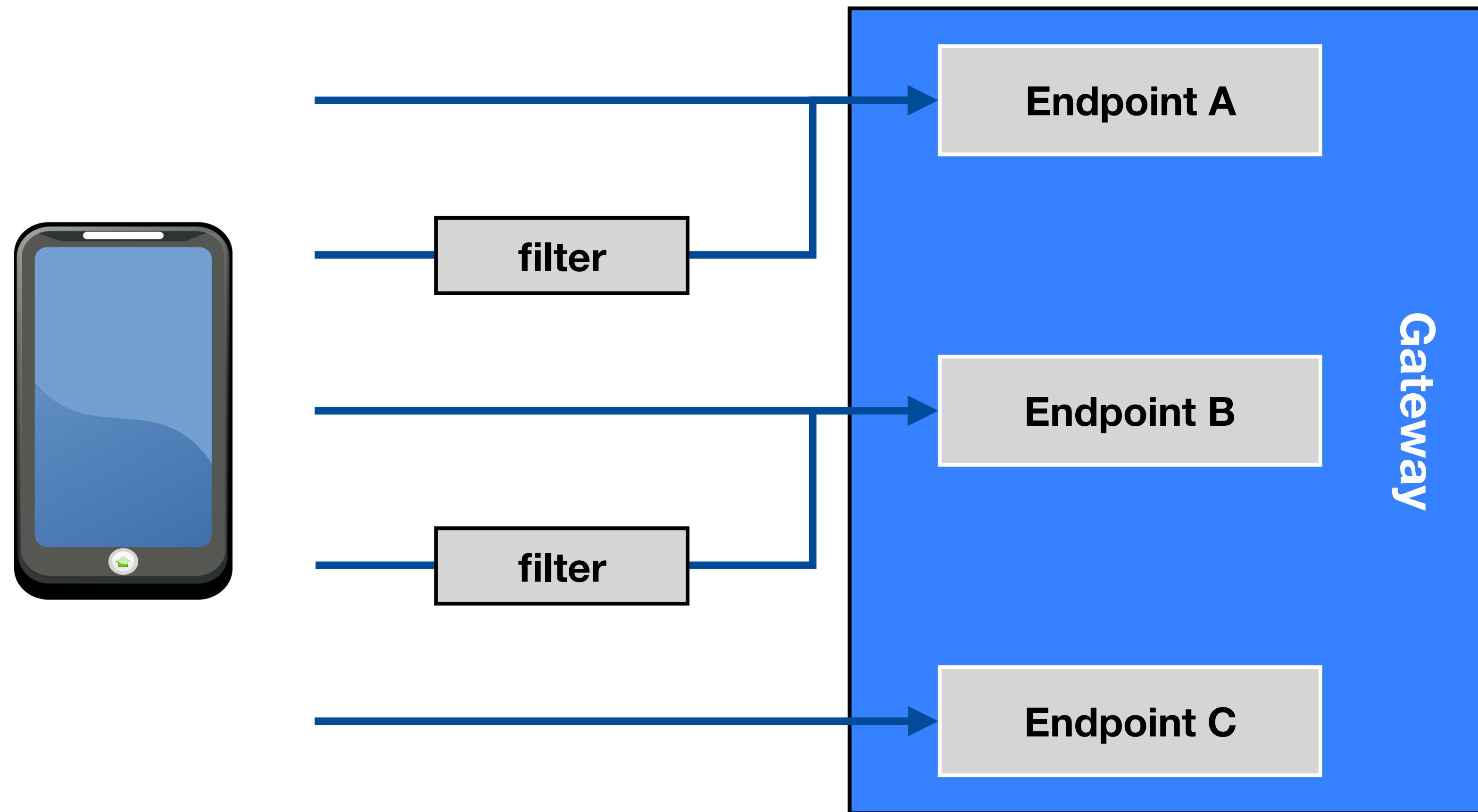
How to Deliver

if(kakao)dev 2019



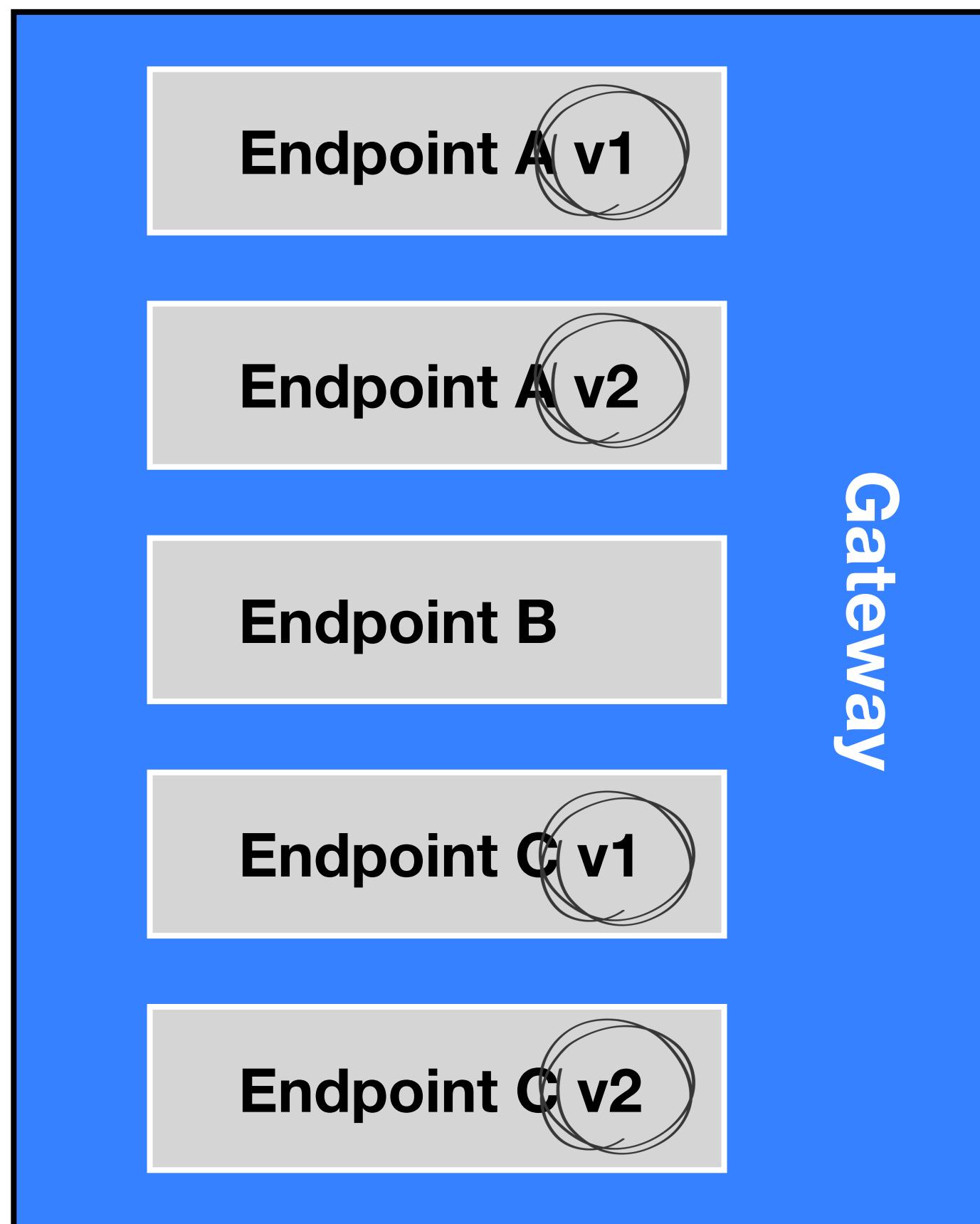
How to Deliver

if(kakao)dev 2019

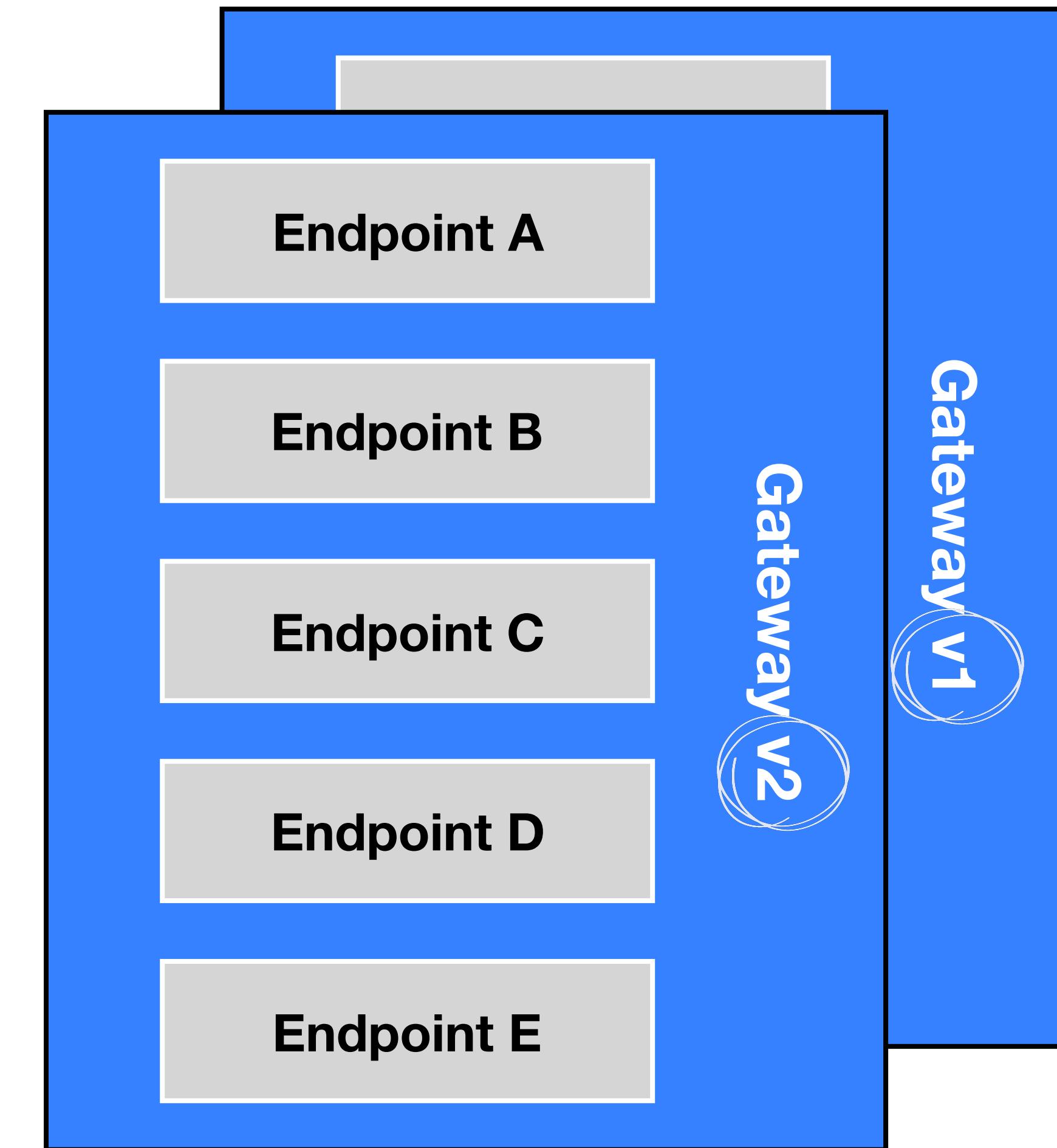


How to Deliver

if(kakao)dev 2019



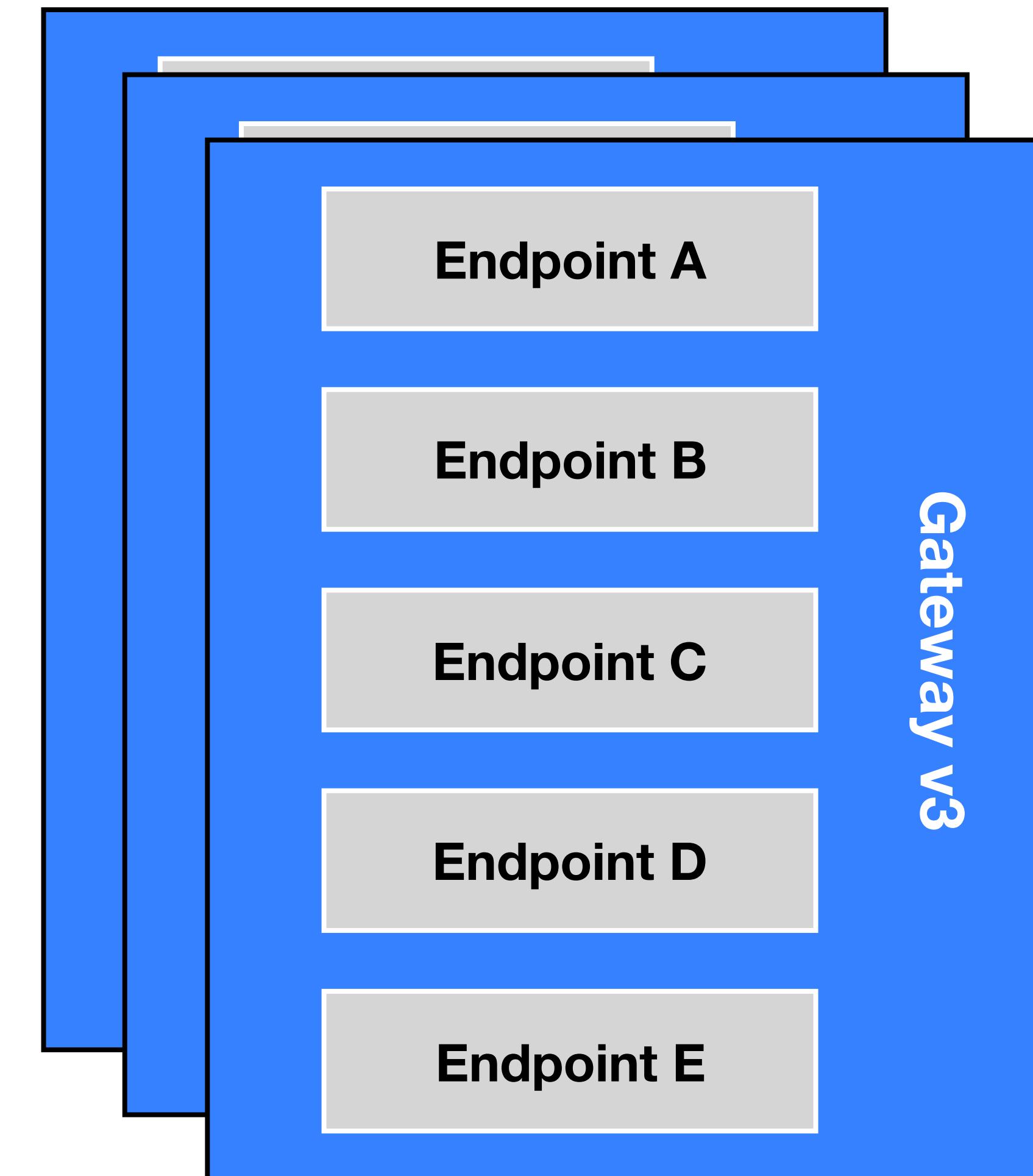
VS



How to Deliver

if(kakao)dev 2019

API doc



GraphQL saves us

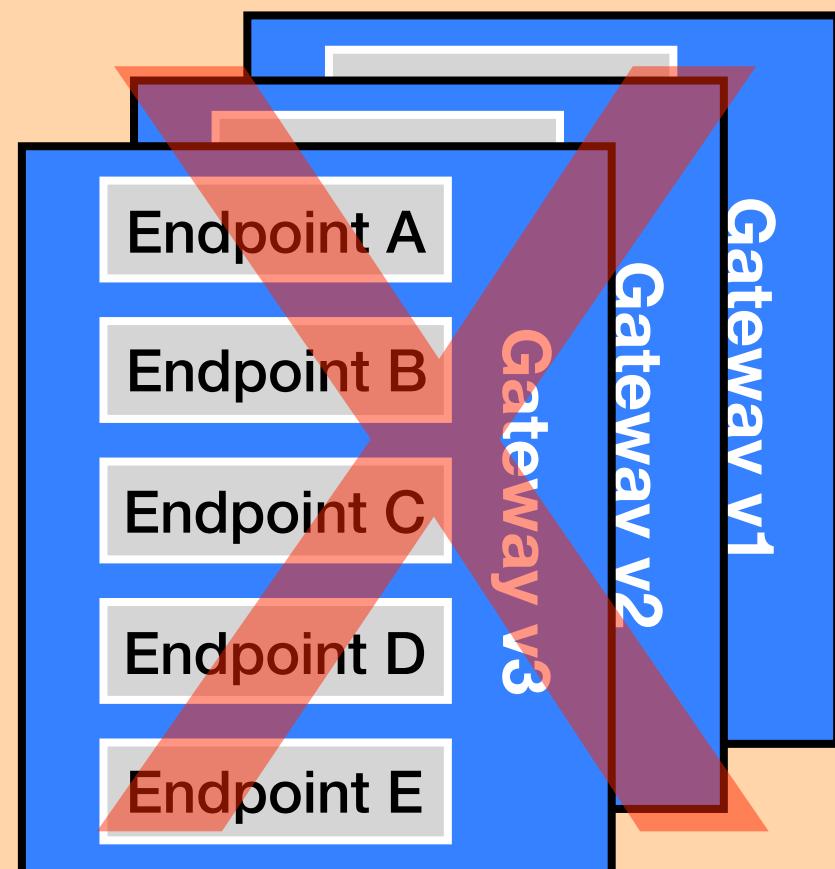
if(kakao)dev 2019



POST /graphql



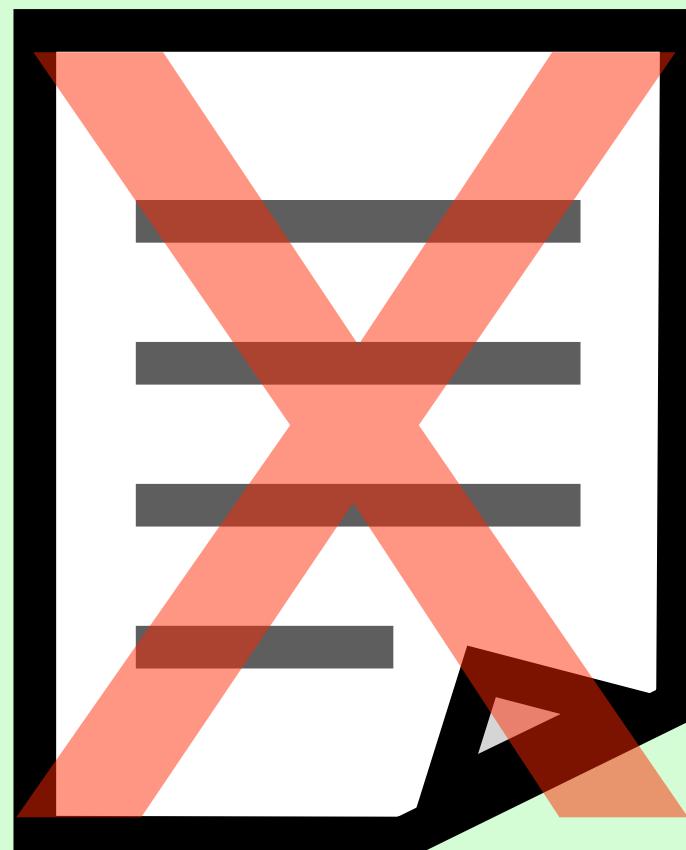
Only One Endpoint



No Versioning

```
query {  
  call(id: 251) {  
    status,  
    fare  
  }  
}
```

Query What I Want



No Document



SHOWTIME





```
1 # Welcome to GraphQL
2 #
3 # GraphQL is an in-browser tool for writing, val
4 # testing GraphQL queries.
5 #
6 # Type queries into this side of the screen, and
7 # typeaheads aware of the current GraphQL type so
8 # validation errors highlighted within the text.
9 #
10 # GraphQL queries typically start with a "{" char
11 # with a # are ignored.  I
12 #
13 # An example GraphQL query might look like:
14 #
15 #   {
16 #     field(arg: "value") {
17 #       subField
18 #     }
19 #   }
20 #
21 # Keyboard shortcuts:
22 #
23 # Prettify Query: Shift-Ctrl-P (or press the pr
24 #
25 # Run Query: Ctrl-Enter (or press the play
26 #
27 # Auto Complete: Ctrl-Space (or just start typ
28 #
29 #
30 #
```

Q Search Query...

No Description

FIELDS

call(id: ID!): Call

driver(driverId: ID, product: Prod

driverDispatchStatus(driverId: ID, Product!): String

listCall(
 product: Product!
 userAgent: String
 location: LocationInput!

driverId: ID
): ListCallResult

activeMove(product: Product!, di
Move

entities(tagName: String!): [Entity

tagByUuid(uuid: ID!): Tag

tag(name: String!): Tag

tags(entityId: String!): [Tag]

entityHasTags(entityId: String!, ta
[String]!): Boolean

workStatus(
 product: Product!



```
1 # Welcome to GraphQL
2 #
3 # GraphQL is an in-browser tool for writing, val
4 # testing GraphQL queries.
5 #
6 # Type queries into this side of the screen, and
7 # typeaheads aware of the current GraphQL type so
8 # validation errors highlighted within the text.
9 #
10 # GraphQL queries typically start with a "{" char
11 # with a # are ignored.
12 #
13 # An example GraphQL query might look like:
14 #
15 #   {
16 #     field(arg: "value") {
17 #       subField
18 #     }
19 #   }
20 #
21 # Keyboard shortcuts:
22 #
23 # Prettify Query: Shift-Ctrl-P (or press the pr
24 #
25 # Run Query: Ctrl-Enter (or press the play
26 #
27 # Auto Complete: Ctrl-Space (or just start typ
28 #
29
30
```

Q Search Call...

No Description

FIELDS

callId: ID

status: String

driverName: String

userName: String

origin: CallLocation

destination: CallLocation

createdAt: DateTime

dispatchedAt: DateTime

endAt: DateTime

fare: Int

distance: Int

stopover: String

appLocation: CallLocation

tags: [Tag]



```
1 query {  
2   call(id:254137) {  
3     |  
4     }  
5   }  
| callId  
status  
driverName  
userName  
origin  
destination  
createdAt  
dispatchedAt
```

QUERY VARIABLES

Search Query...

No Description

FIELDS

call(id: ID!): Call
driver(driverId: ID, product: Product!): Driver
driverDispatchStatus(driverId: ID, product!: Product!): String
listCall(
 product: Product!
 userAgent: String
 location: LocationInput!
 driverId: ID
): ListCallResult
activeMove(product: Product!, direction: Move): Move
entities(tagName: String!): [Entity]
tagByUuid(uuid: ID!): Tag
tag(name: String!): Tag
tags(entityId: String!): [Tag]
entityHasTags(entityId: String!, tags: [String]): Boolean
workStatus(
 product: Product!

 Search Query...

No Description

FIELDS

call(id: ID!): Call

driver(driverId: ID, product: Product!): Driver

driverDispatchStatus(driverId: ID, product!: Product!): String

listCall(
 product: Product!
 userAgent: String
 location: LocationInput!
 driverId: ID
): ListCallResult

activeMove(product: Product!, driverId: ID): ActiveMove

entities(tagName: String!): [Entity]

tagByUuid(uuid: ID!): Tag

tag(name: String!): Tag

tags(entityId: String!): [Tag]

entityHasTags(entityId: String!, tags: [String]!): Boolean

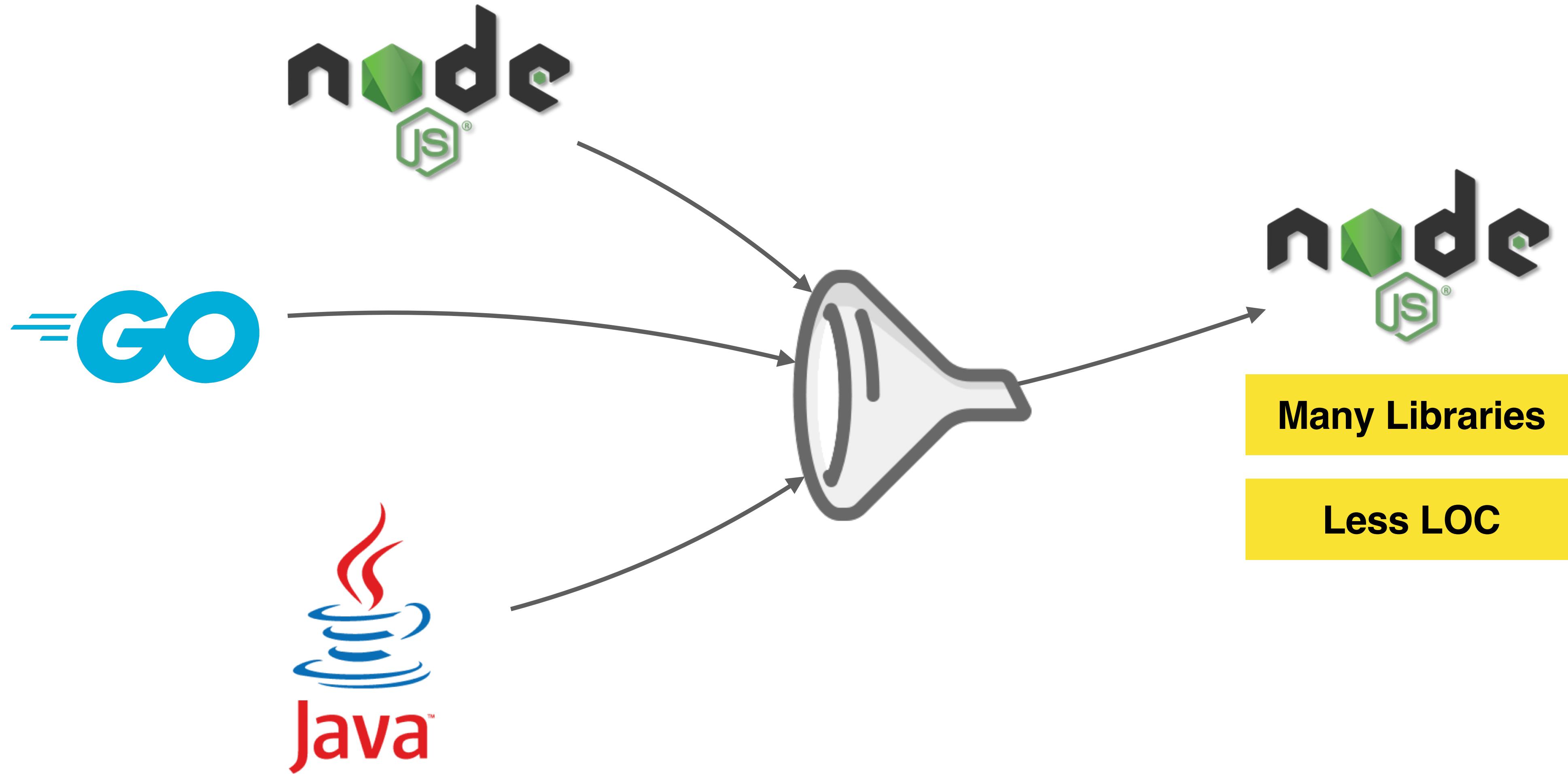
workStatus(
 product: Product!

```
1 query {  
2   call(id:254137) {  
3     origin {  
4       lat  
5       lng  
6     }  
7     fare  
8   }  
9   activeMove(product:DRIVER, driverId:1004550) {  
10    destinationInfo {  
11      poi  
12    }  
13  }  
14}
```

```
{  
  "data": {  
    "call": {  
      "origin": {  
        "lat": "37.39433886412452",  
        "lng": "127.1105739023173"  
      },  
      "fare": 18000  
    },  
    "activeMove": {  
      "destinationInfo": {  
        "poi": "판교역 신분당선 3번출구"  
      }  
    }  
  }  
}
```

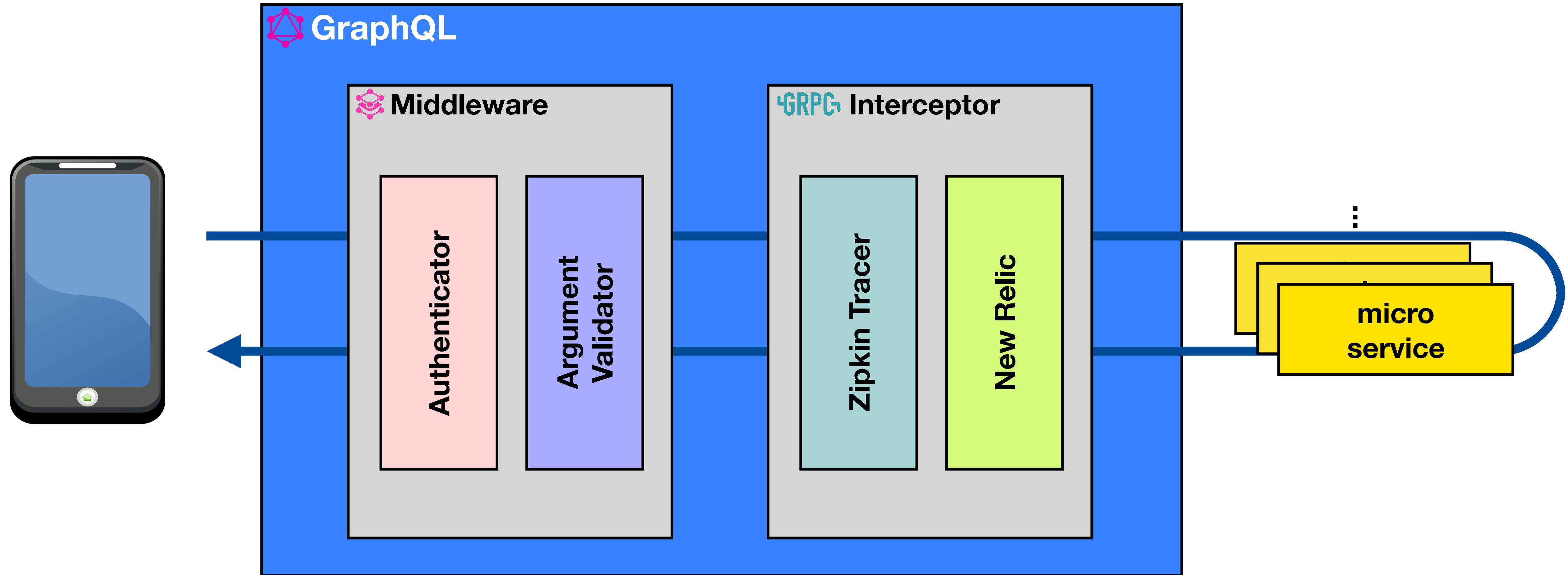
GraphQL 구현체

if(kakao)dev 2019



Middleware 亂이

if(kakao)dev 2019



09 서비스 모니터링

if(kakao)dev 2019

2019. 6. 21 서포터즈 기사 시범 서비스 출시

결과는?



서비스 지연 발생

if(kakao)dev 2019

Peak Time에 서포터즈 기사 기능 추가로 인한 서비스 지연 발생

- 서비스 내 가장 요청이 잦은 API에 Timeout 발생
- Timeout이 발생할 동안 Database의 connection release가 되지 않음
- 해당 DB를 사용하는 모든 요청에 병목으로 작용하여 Timeout 확산

전체 서비스에 지연 현상 전파 → 즉시 롤백 후 원인 파악 및 대처 결정

502 Bad Gateway

nginx

모니터링 시스템 부재의 원인과 대책

if(kakao)dev 2019

사업적 이유로 출시일이 개발 완료 이전에 확정된 상황

- 촉박한 출시 일정
- 출시 이후에 모니터링 시스템 구축 결정
- 빠른 적용이 관건인 상황

Prometheus & Grafana 도입

이벤트 모니터링과 알림에 사용되는 오픈소스 메트릭 수집 도구(Apache 2.0 License)

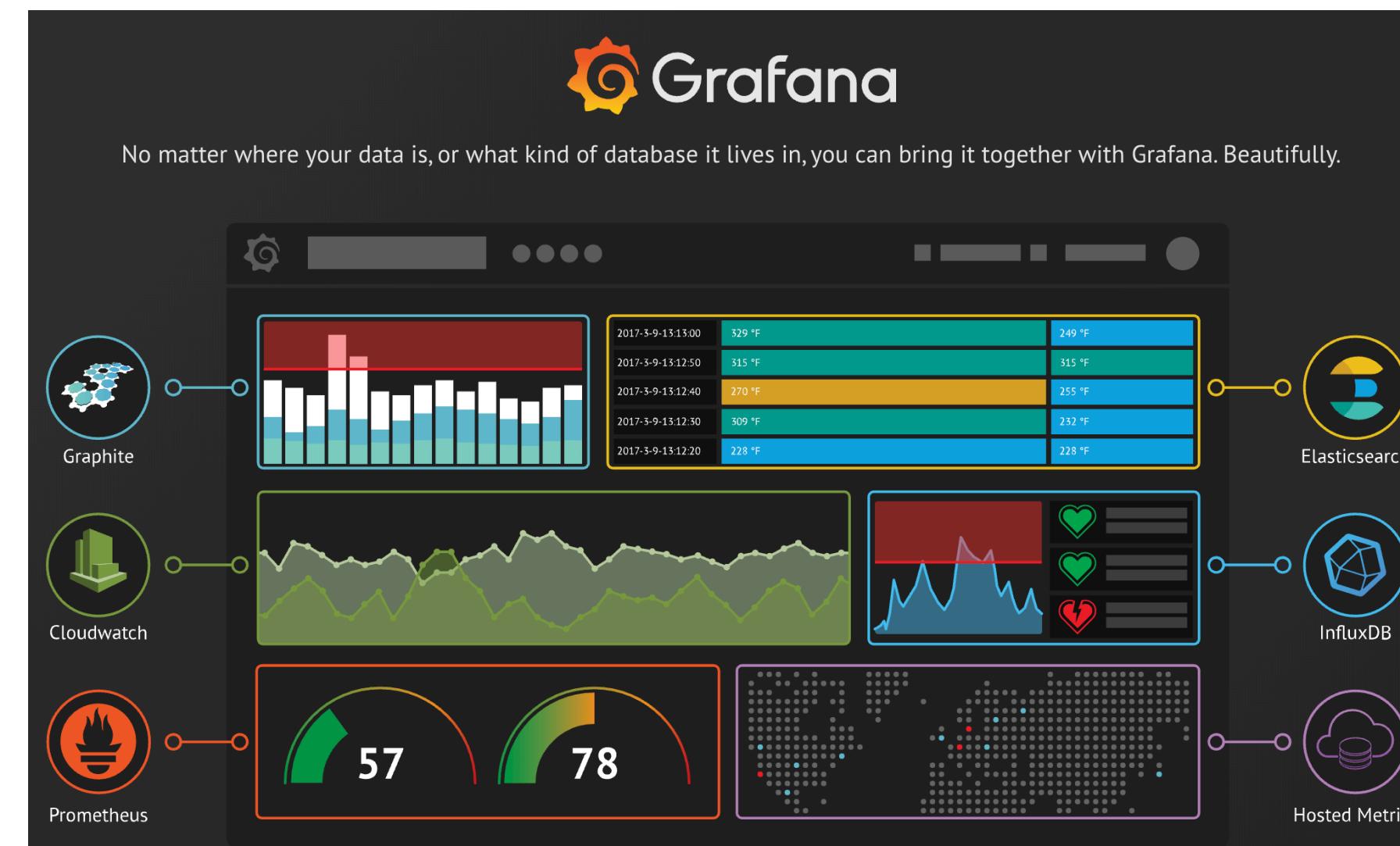
- **Pull** 방식의 메트릭 수집
 - 대부분의 모니터링 도구는 리얼타임으로 클라이언트가 서버로 Push
 - Prometheus 클라이언트에서 Push하지 않고, Prometheus 서버가 Pull
- 모든 데이터를 수집하는 것이 아닌 **일정 주기**(default 15s)로 발생하는 메트릭 수집
 - Application Performance에 유리
 - '추이'를 보는데는 좋지만 APM과 같이 발생한 모든 로그를 추적하는 목적에는 부적합

Prometheus는 Docker와 K8S로 구성된 **컨테이너 환경**에 **최적화**된 모니터링 툴

Grafana

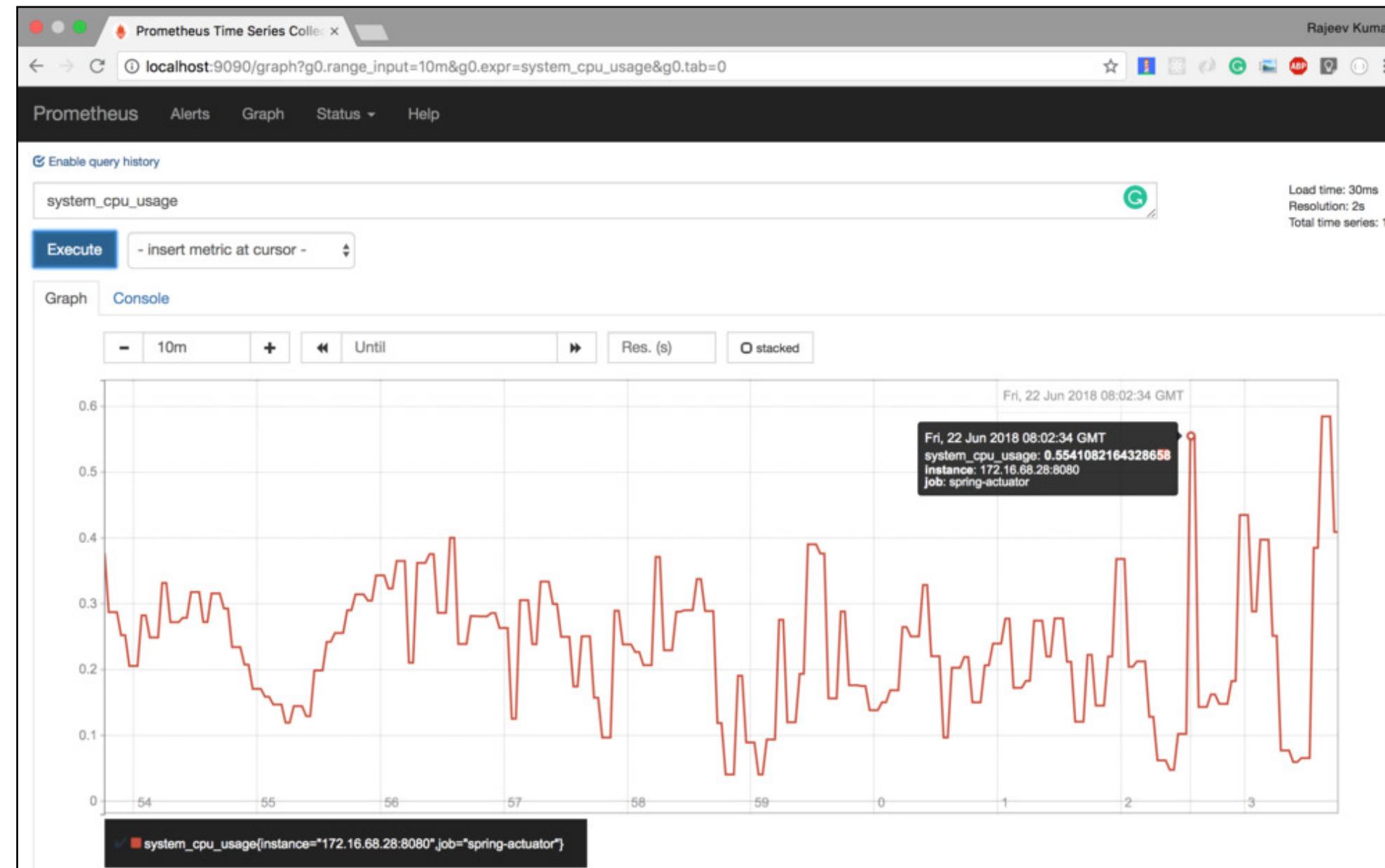
유려하게 시각화 가능한 오픈소스 모니터링 플랫폼(Apache 2.0 License)

- Prometheus Web UI에서 간단한 그래프 확인 가능 → 상용 서비스의 모니터링 환경으로는 부족
- **Prometheus가 권장하는 모니터링 시각화 도구**
<https://prometheus.io/docs/visualization/grafana>
- 데이터 플랫폼 **고유의 Query Language**를 이용하여 데이터 소스로부터 정보를 조회하여 시각화
 - Prometheus, Graphite, influxDB, Elasticsearch, AWS Cloudwatch, RDBMS, etc.

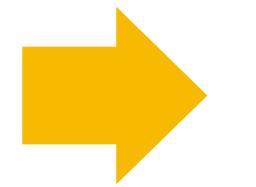


Grafana

if(kakao)dev2019



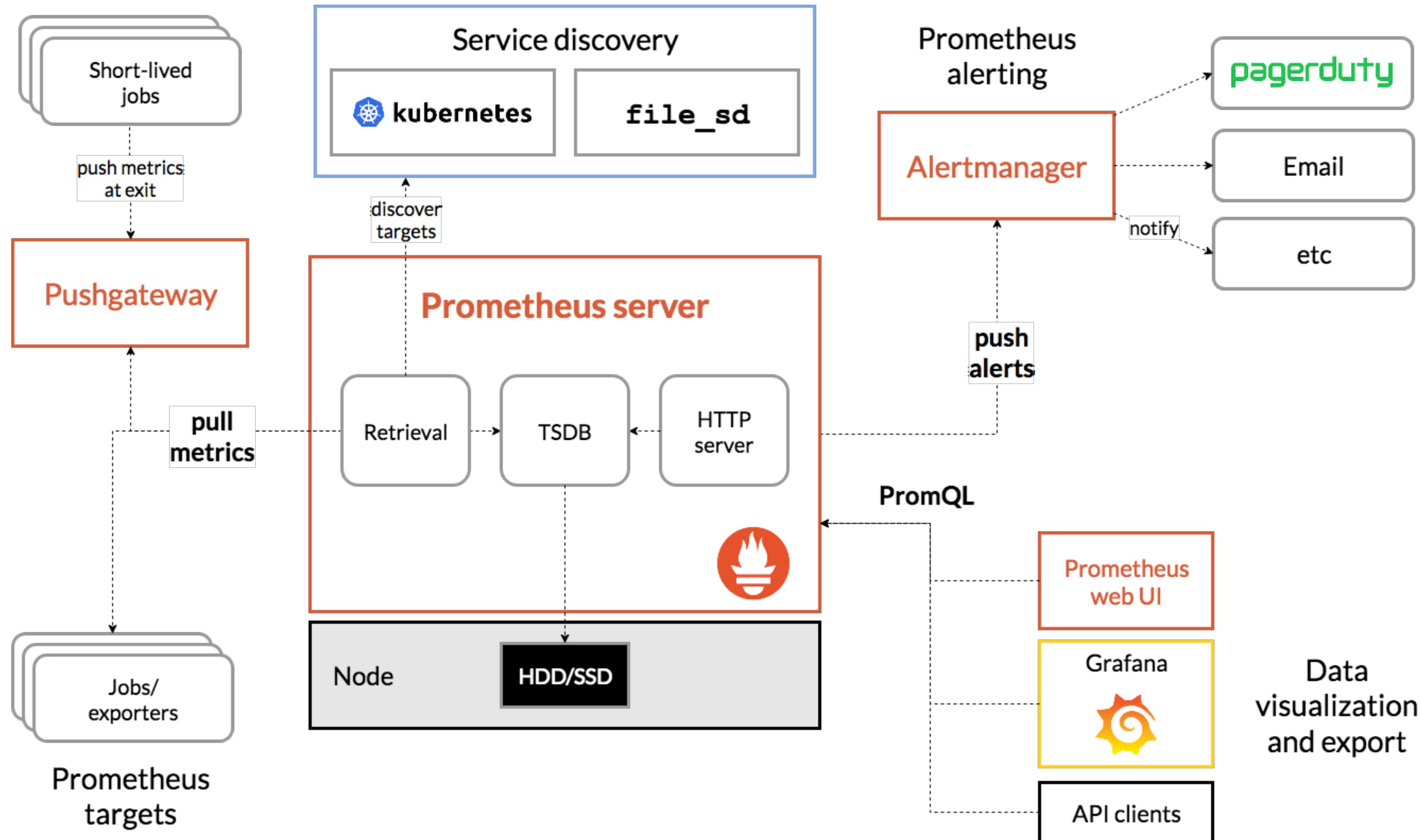
Prometheus Web UI



Grafana

모니터링 시스템 구조

if(kakao)dev 2019

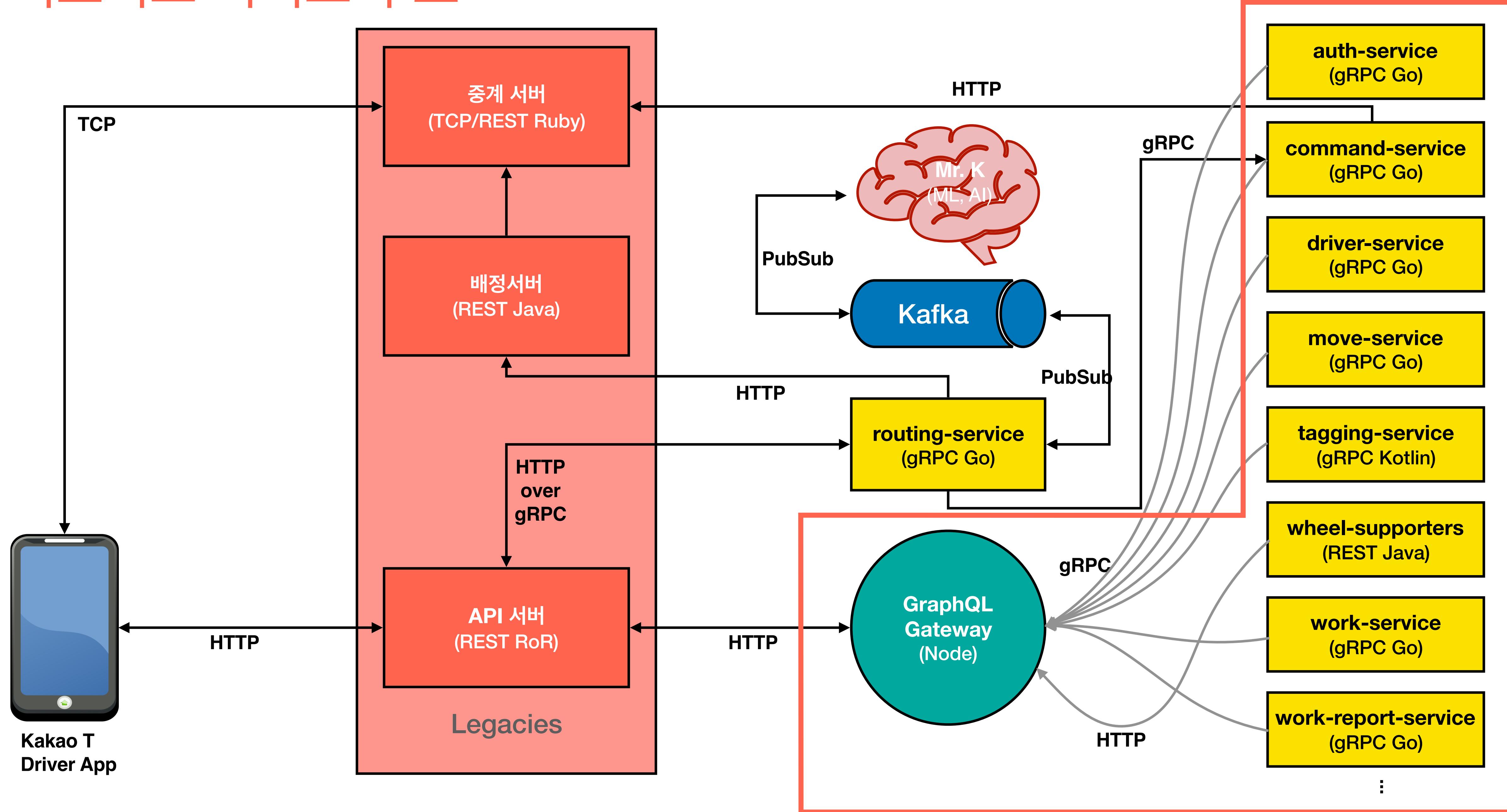


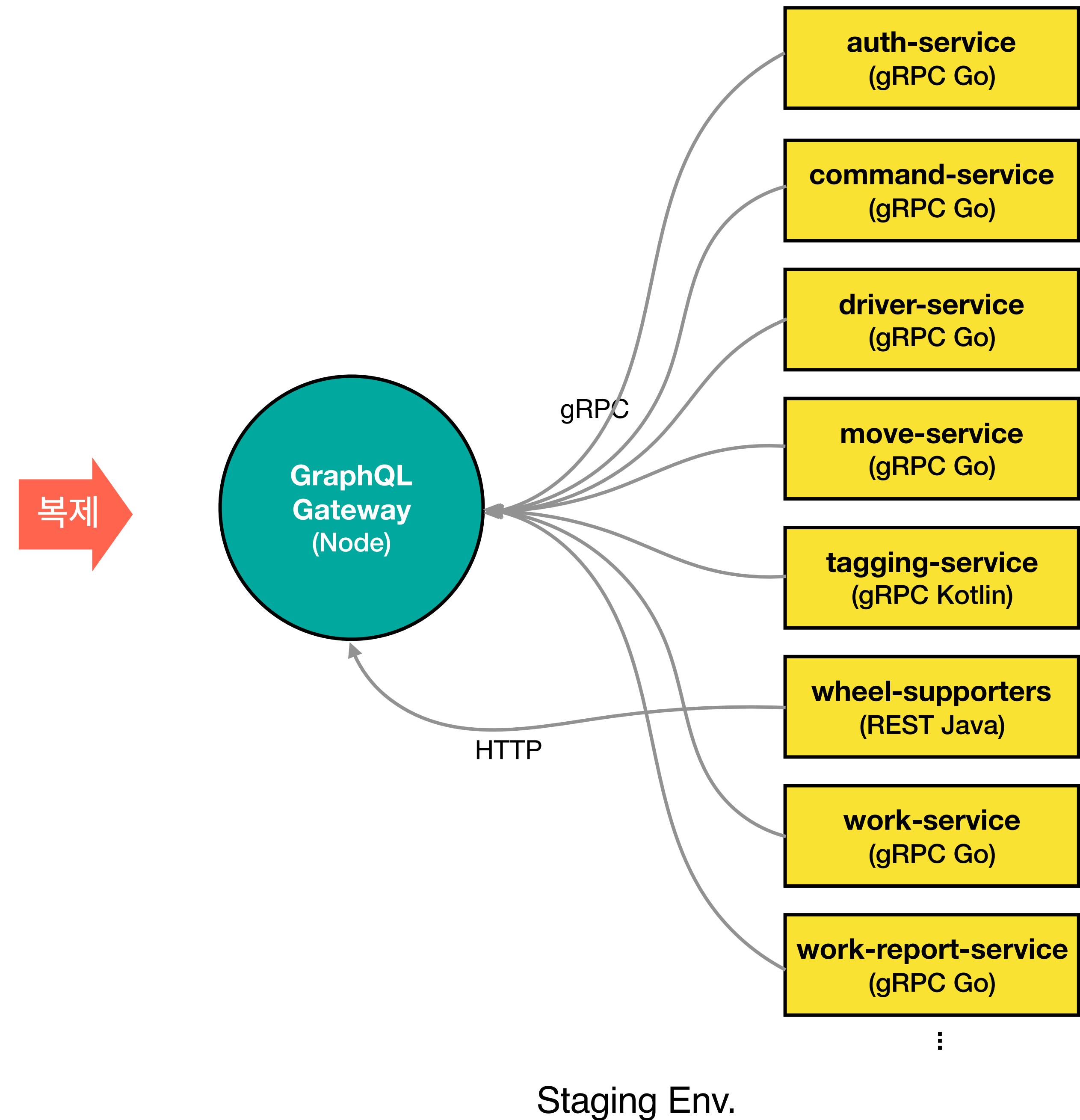
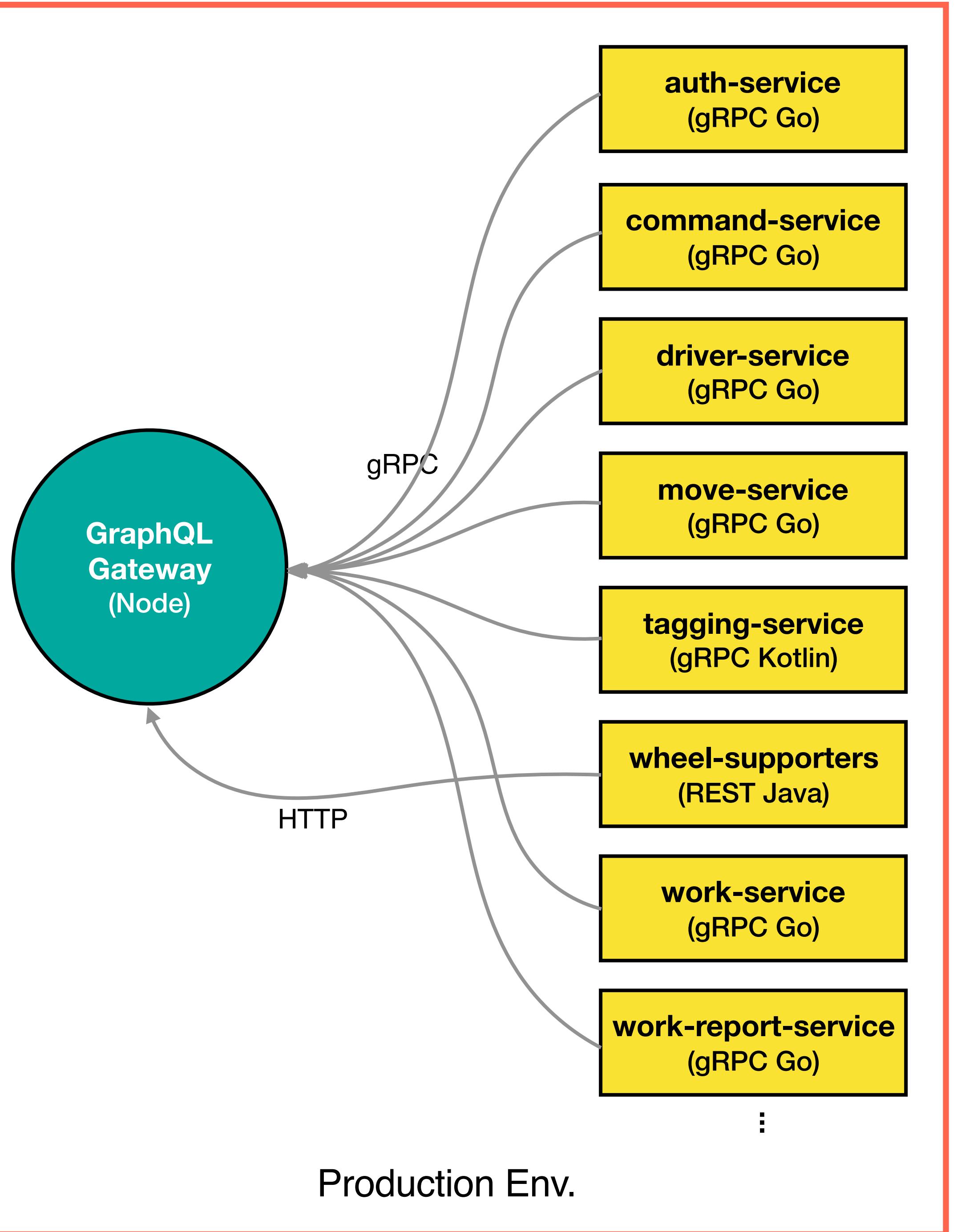
장애의 원인은 무엇이었을까?

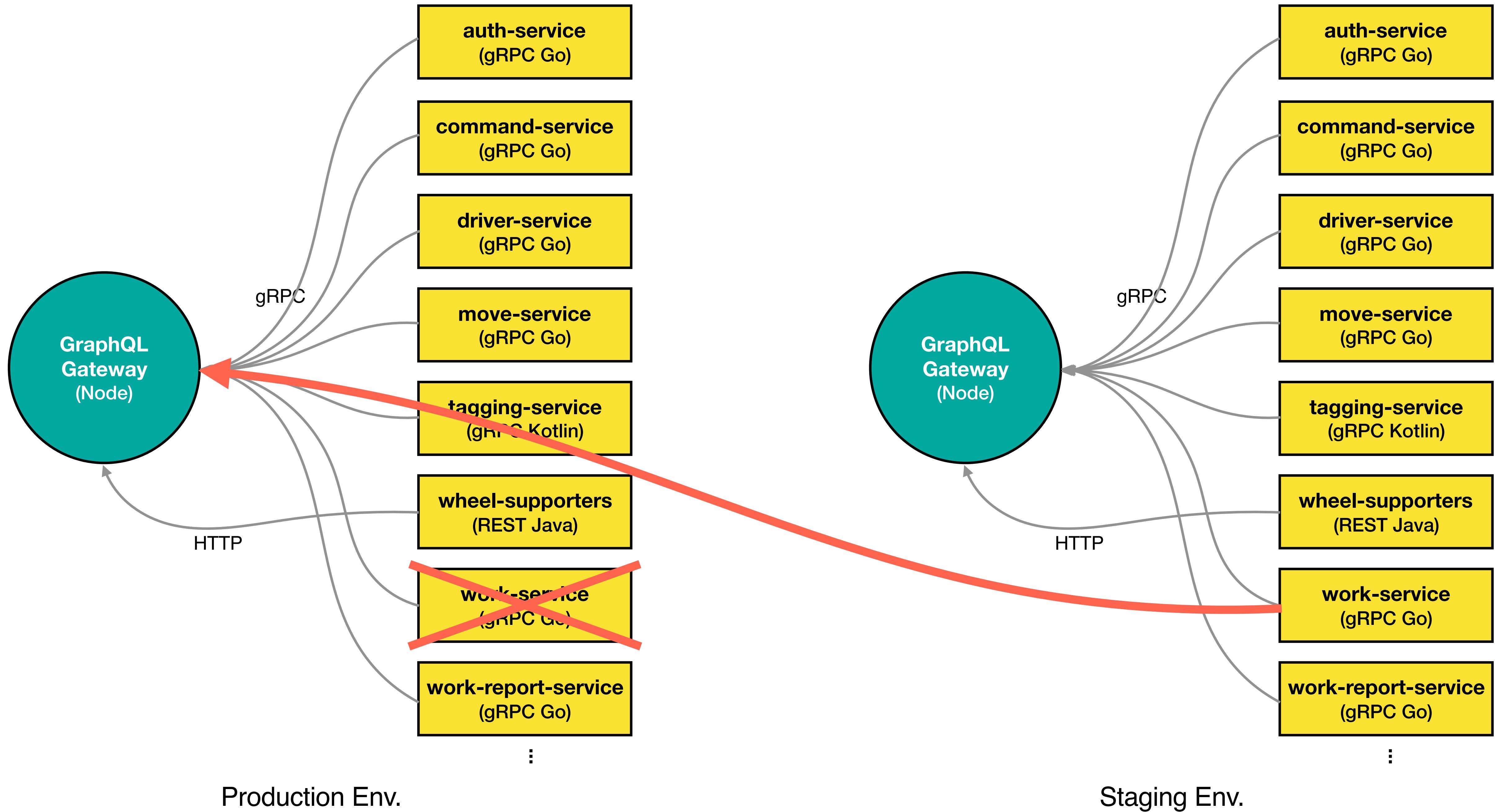
if(kakao)dev 2019

서포터즈 서비스 구조

if(kakao) dev 2019







장애의 원인은 무엇이었을까?

if(kakao) dev 2019

작은 서비스(micro service)의 사소한 실수가 전체 제품(whole product)에 영향

NewRelic, AppDynamics, Jennifer 등 기존에 널리 사용되는 APM으로 요청의 전체 흐름을 분석하기 어려움

- 문제가 되는 구간을 특정할 수 없음
- 트랜잭션이 분산된 여러 서비스를 거쳐 조합되어 처리되기 때문에 분산 로그 추적 시스템이 필요

Zipkin

지연 시간 문제를 해결하는 데 필요한 타이밍 데이터를 수집하기 위한 오픈소스 분산 추적 시스템(Apache 2.0 License)

- C#, Go, Java, Javascript, Ruby, Scala, PHP 등 대부분의 언어를 지원
- In-memory, MySQL, Cassandra, Elasticsearch 등에 데이터를 저장
- HTTP, Kafka, Scribe 등의 전송 프로토콜을 사용 가능

10 Legacies

Legacies

우리가 원하는 것

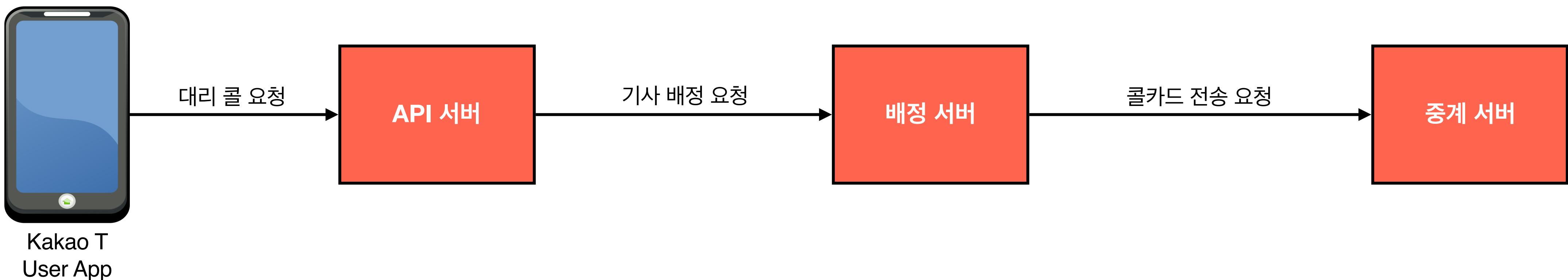
- 너무나 잘 동작하고 있는 **레거시의 변화를 최소화**하고 싶다.
- 새로운 feature는 도메인에 맞는 **마이크로서비스로 분리**하고 싶다.
- 기존의 feature도 도메인에 맞는 **마이크로서비스로 분리**하고 싶다.
- 레거시가 없어질 때 까지 **레거시와 공존**하고 싶다?

Legacies

if(kakao)dev 2019

새로운 서포터즈 배정 flow 를 기존 배정 서버에서 분리하고 싶다.

단, 기존의 배정 시스템에 영향이 없어야 한다.

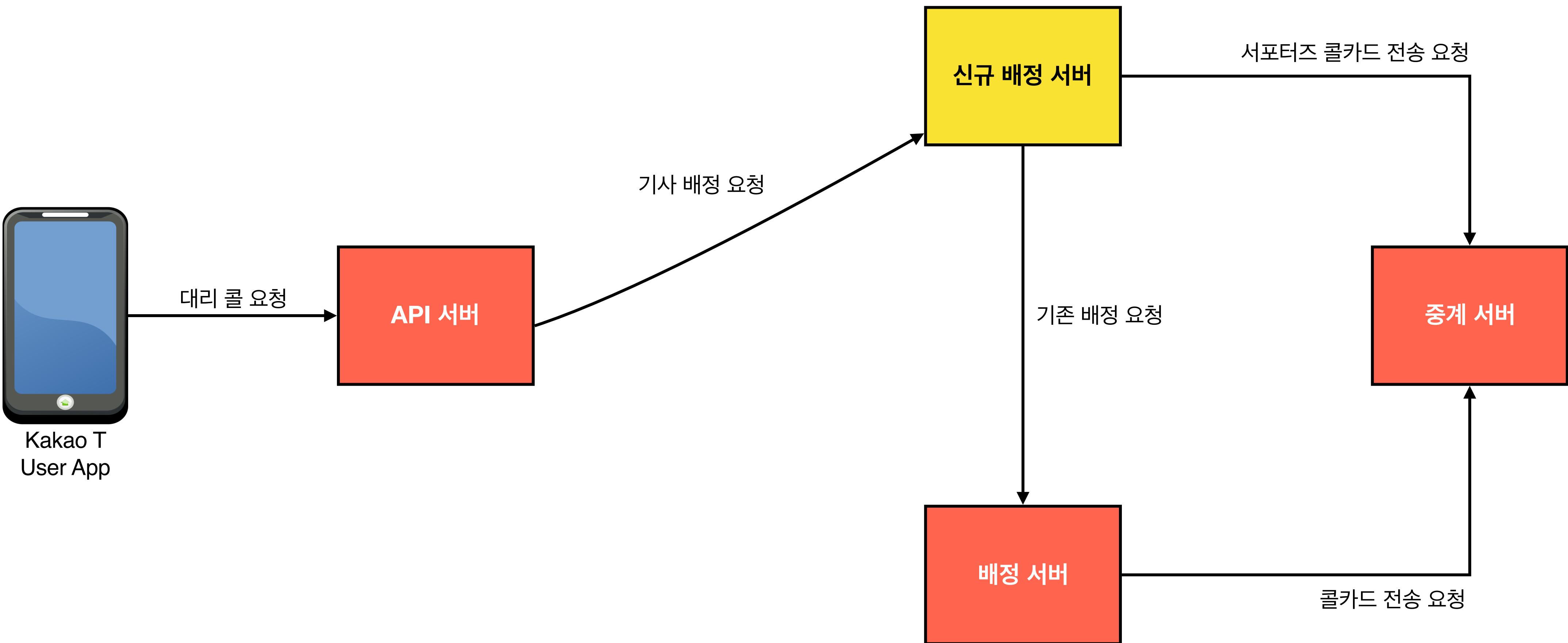


Legacies

if(kakao)dev 2019

기존의 기사 배정 요청을 신규 배정서버로 위임

새로운 배정 플로우가 아닌 경우 기존 배정서버로 전가



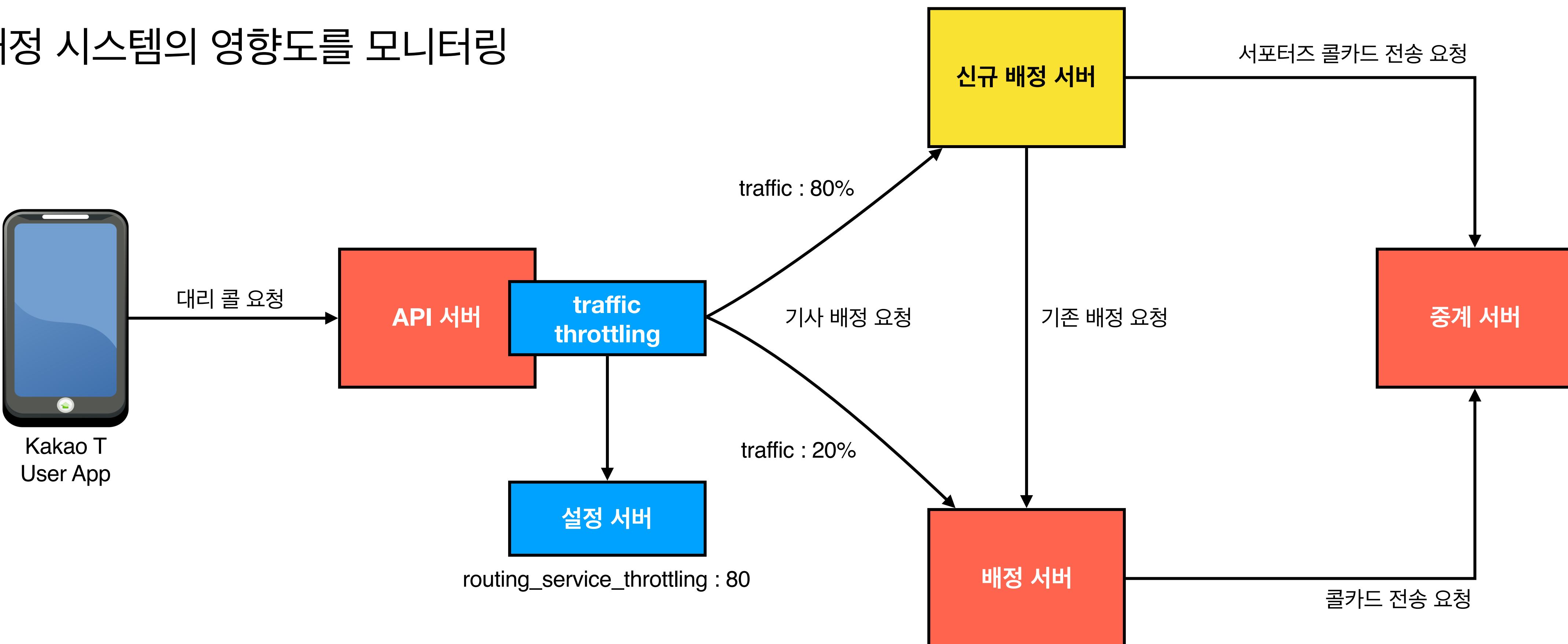
Legacies

if(kakao)dev 2019

feature flag 의 일환으로 api 서버에 **traffic throttling** 기능을 추가

설정서버를 통해 트래픽을 조절

기존 배정 시스템의 영향도를 모니터링

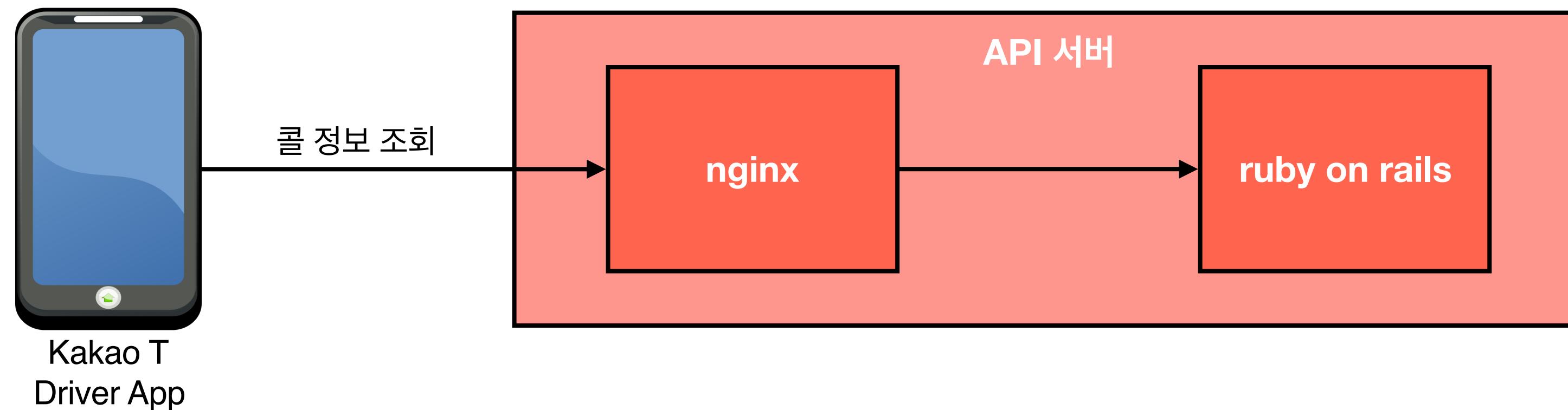


Legacies

if(kakao)dev 2019

콜 관련 api 들을 레거시에서 마이크로서비스로 분리하고 싶다.

단, 기존 api 호출에 영향이 없어야 한다.

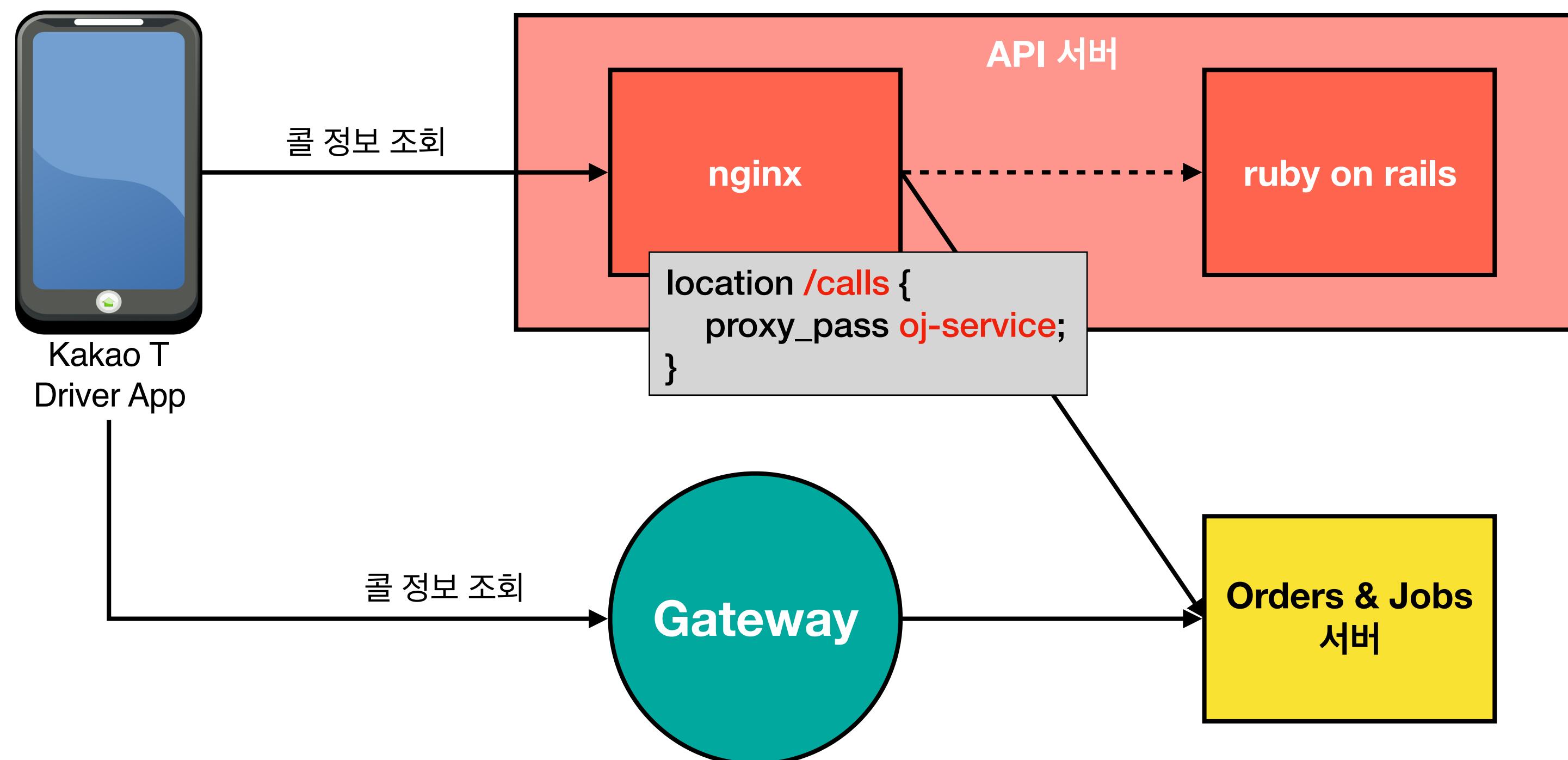


Legacies

if(kakao)dev 2019

nginx reverse proxy 를 활용해서 콜 관련 api 들만 마이크로서비스로 분리

기존 api 호출을 대응하면서 점진적으로 새로운 마이크로서비스 호출로 이관

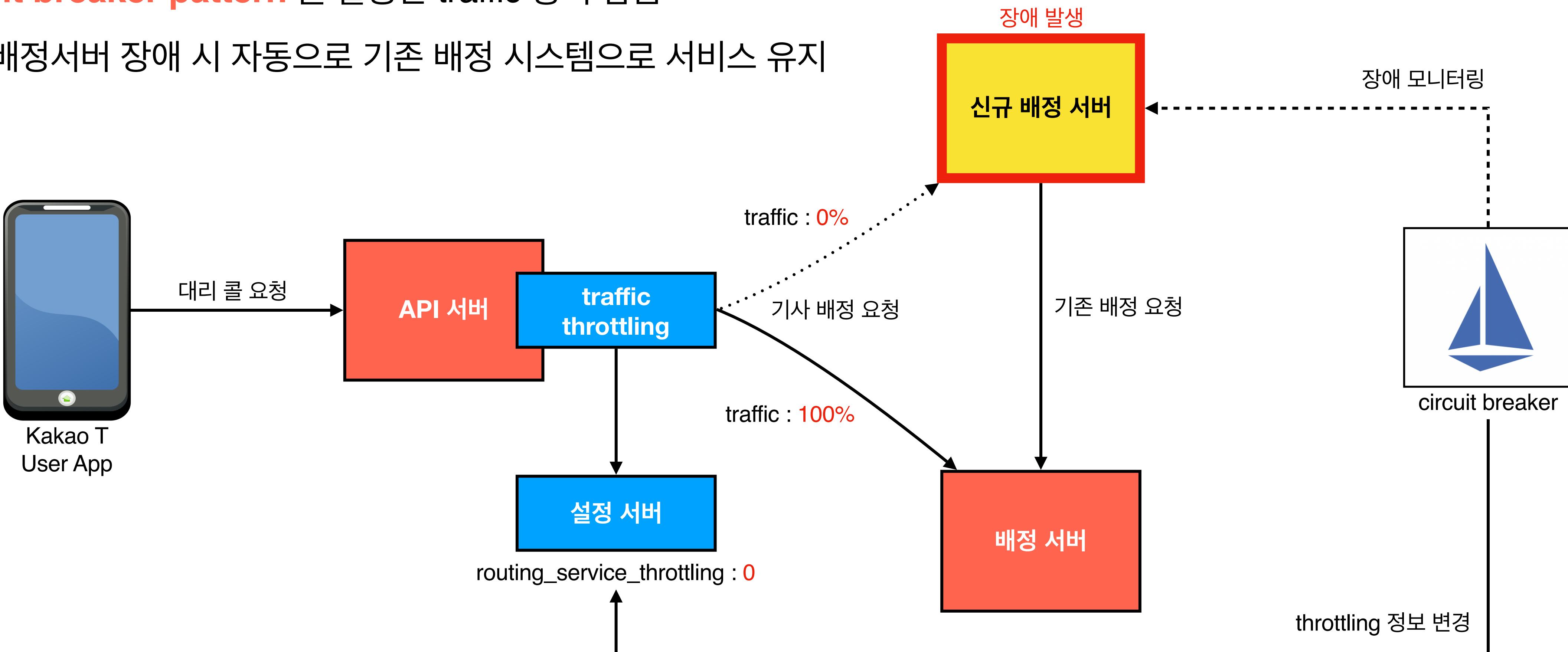


Legacies

if(kakao)dev 2019

우리가 해야 할 것

- **circuit breaker pattern** 을 활용한 traffic 동적 잠금
- 신규 배정서버 장애 시 자동으로 기존 배정 시스템으로 서비스 유지

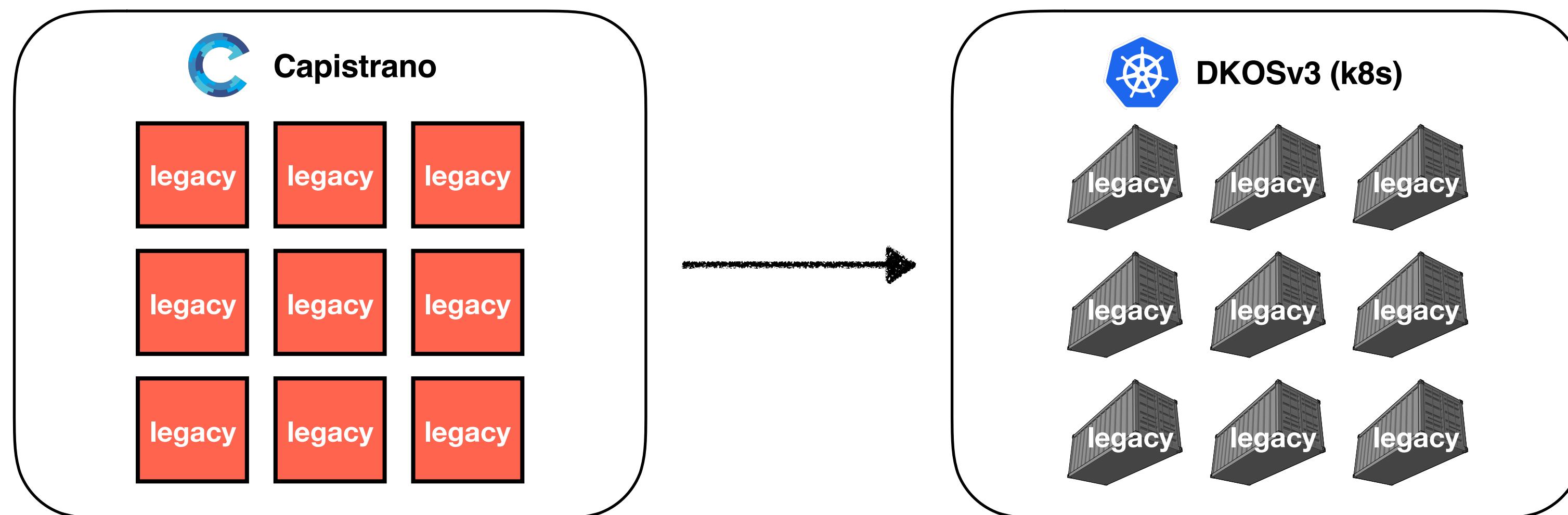


Legacies

if(kakao)dev 2019

우리가 해야 할 것

- 레거시 시스템의 **로컬 환경 구축** 절실
- 레거시 시스템의 신규 VM 증설 및 배포 용이성 필요
- 레거시의 **dockerizing** 및 kubernetes 클러스터로 이관 중



11 성과

if(kakao)dev 2019

사업적 성과

if(kakao) dev 2019

기사 출근율

+5.7%p

운행 완료 비율

+37.6%p

배정 후 취소율

-5.6%p

기술적 성과

if(kakao) dev 2019

기존 제품 영향
최소화로
서비스 개시

14개의
Microservice,
138개의
Instance

17개의
Go 언어 저장소

평균 서비스
응답시간
 $< 10\text{ms}$

99.99% SLA

무중단
배포 및 운영

Zero
Hardware
Provisioning



감사합니다!

if(kakao)dev 2019

