

개발내역 약술서

개발 약술서

개발내역

root@/# 김일범

개발 프로젝트 상술서

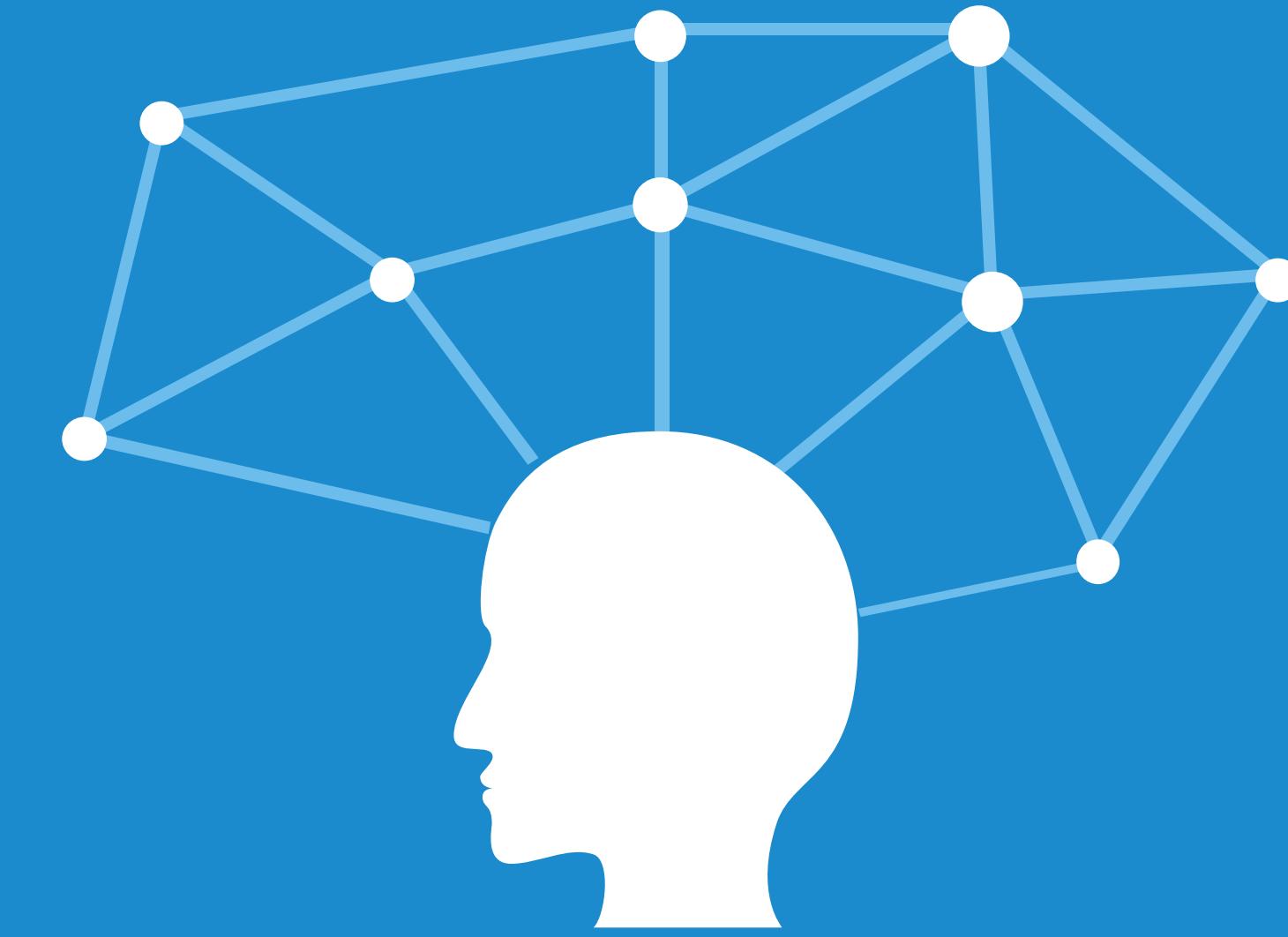
kimilb412@gmail.com





개발 : 2014년 06월
웹 골프 게임

HTML5, CANVAS 및 Box2d을 통한 2차원 물리 골프 게임
Animation Render, Resource preloading 등 기법 적용



Minigolf

개발 환경

개발 언어

- Javascript, HTML

개발 도구

- Eclipse, Aptana, notepad++

Database

- Mysql

역할 및 이용 기술

역할

- 총원 3명 중 리드
- 클라이언트 일괄 개발
- 서버 통신, 엔진 적용, 이미지 렌더링, 이벤트 처리, 사운드 처리 등

기술

- preload.js, box2d.js, canvas api, ajax, Microsoft Azure
- SVN
- HTML 5 API

Minigolf

기술 채택 사유

- Box2d
 - Physics.js, p2.js, verlet.js, matter.js 등 다양한 기술 중 box2d.js 채택.
 - 당시 공부하던 물리엔진 기술 중 이해도가 가장 높았고 이에 따라 구현에 필요한 기능 적재 적용 가능
- Canvas
 - DOM 구조를 이용한 애니메이션 렌더링에는 큰 제한이 존재
 - 자유로운 렌더링을 위해 선택
- Preload.js
 - 자원 로딩 지연을 줄이기 위해 선택
- Pixi.js
 - Oxygen과 비슷한 일부 이벤트 처리 메커니즘을 사용하기 위해 선택
- Microsoft Azure
 - Node server 배포를 위해 사용(당시 계정이 있는 사람이 존재하여 클라우드 서버 사용 목적으로 선택)
- SVN
 - 소스 관리를 위해 당시 Naver에서 지원하던 SVN 사용

Minigolf

기억하고 싶은 내용

- 물리 충돌 알고리즘
 - 수 많은 오브젝트들이 화면에 렌더링되며 프레임에 맞춰 수 많은 충돌 연산 구현이 필요
 - 다른 물리 엔진(퀘이크 등)의 내부 로직을 분석하여 아이디어를 획득
 - 쿼드 트리 충돌 알고리즘을 통한 연산 대상 축소
- 렌더링 자연 축소
 - 이미지 애니메이션 등 화면에 그려줄 데이터 크기가 상당하여 매 단계 로딩 시 시간이 오래 걸림
 - Pre-loading 등 여러 기법들을 학습하여 적용
- 사용된 알고리즘 및 기법들
 - AABB Collision, Collision Resolving, Physical Vector applying, Canvas rendering
 - Window sliding, Loading Reserving(pre loading), Quad-tree detection, viewport resolving
 - XHR request, prototype polymorphism, 2D physics algorithm(box2d) 등
- 참고 링크
 - <https://gamedevelopment.tutsplus.com/tutorials/quick-tip-use-quadtrees-to-detect-likely-collisions-in-2d-space--gamedev-374>
 - <http://kimilb412-2.blogspot.com/search/label/Box2D>

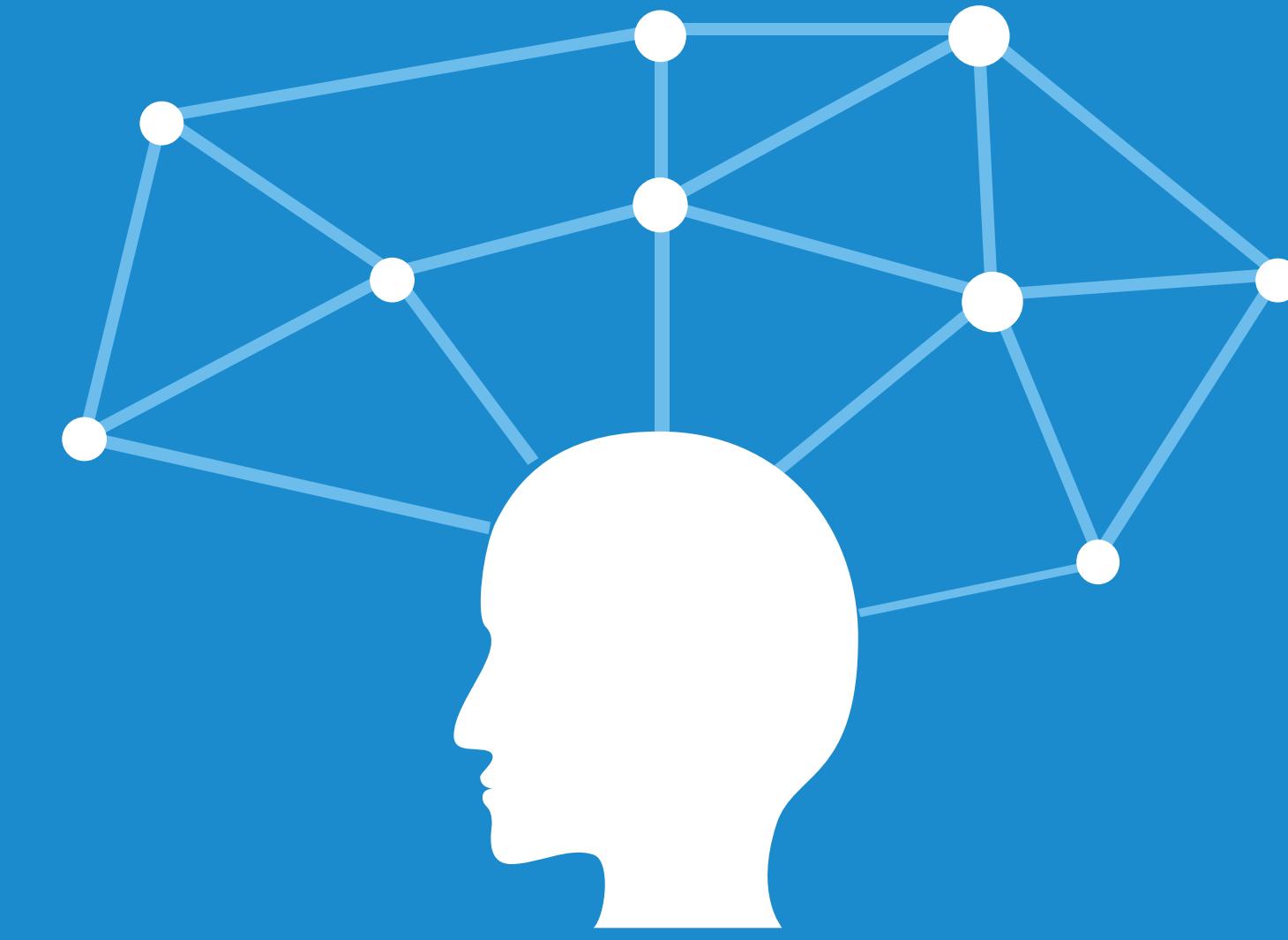
Owlet

개발 : 2014년 07월

소상공인 챌린지 정부 프로젝트

웹앱 기반 정보제공 앱을 목적하는 소상공인을 위한 앱
저작 서비스

에디터를 통해 화면을 제작하면 모바일/웹 앱이 도출



Owlet

개발 환경

개발

- Javascript, Java, HTML

개발 도구

- Eclipse Kepler, Aptana, notepad++

Database

- Mysql

이슈 관리 도구

- Redmine

형상 관리 도구

- Git

역할 및 이용 기술

역할

- 총원 5명

클라이언트 일괄 개발

- 서버 통신, 대시보드 화면, 업로드, 통계, 지도 화면

사용 기술, 라이브러리

- Spring 3.0

- Full calendar API

- Google map API

- CKEditor

- HTML & JSP

- Google Chart, Raphael

- Dslick, drag/drop.js, Modernizer, bootstrap

Owlet

기술 채택 사유

- Full calendar API
 - 달력 뷰 및 일정관리 기능을 제공하며 사용자정의가 용이한 오픈 소스 API로 채택
 - 쿠폰, 일정 관련 기능에 활용
- Google Map API
 - 핀 끌기 및 위치 찾기 등의 개발에 사용 이력이 있어 채택
- CKEditor
 - 웹 오픈 소스로 편집기를 손쉽게 제작할 수 있어 채택
 - 고급 기능 또한 지원되며 플러그인을 통해 사용자 정의 기능 확장해 사용
- HTML&JSP
 - Client View로 사용. HTML5/CSS3/JS로 Semantic 구조 작성
 - 서버 담당자와 Spring 3.0을 공부하던 시기에 채택
- Spring Framework 3 with MVC
 - 웹개발 서버 프레임워크로 사용. IOC, AOP 개념을 통한 웹 서버 개발
- Google Chart, Raphael
 - 사용자 통계를 보여주기 위한 차트 API로 채택. D3.js는 제외함
- Dsslick, drag/drop.js, Modernizer, bootstrap
 - 높은 사용자 경험을 제공하기 위해 채택, 드래그, 플랫 디자인 제작에 적합

Owlet

기억하고 싶은 내용

- 클라이언트 요구사항 불확실성
 - 처음으로 실제 외부 클라이언트가 존재하는 상태에서 개발
 - 요구사항이 모호하고 변경되는 바를 이해함
 - 개념적으로 배운 프로젝트 설계 방법론, OOP 등 여러 이론의 토대에 대해 이해하기 시작
- 이슈 관리 도구
 - Jira, trello, gitlab 등 여러 가지 이슈 관리 도구가 존재하나 redmine을 이용한 경험
 - 클라이언트에 프로젝트 산출과 일정을 제공하는 ‘구체적인’ 자료 제시 가능
 - 각 팀원간 업무 현황 파악 용이, 계획적 분배 가능
- Agile 개발 방법론
 - 문서화 없음
 - 불필요하게 들어가는 노고를 제거한 경험
 - 그러나 처음 시도하는 만큼 모호한 기준에 대해 되려 시간을 버린 경험
 - 모든 방법론이 최적은 아니므로 취사선택하는 방법을 고찰

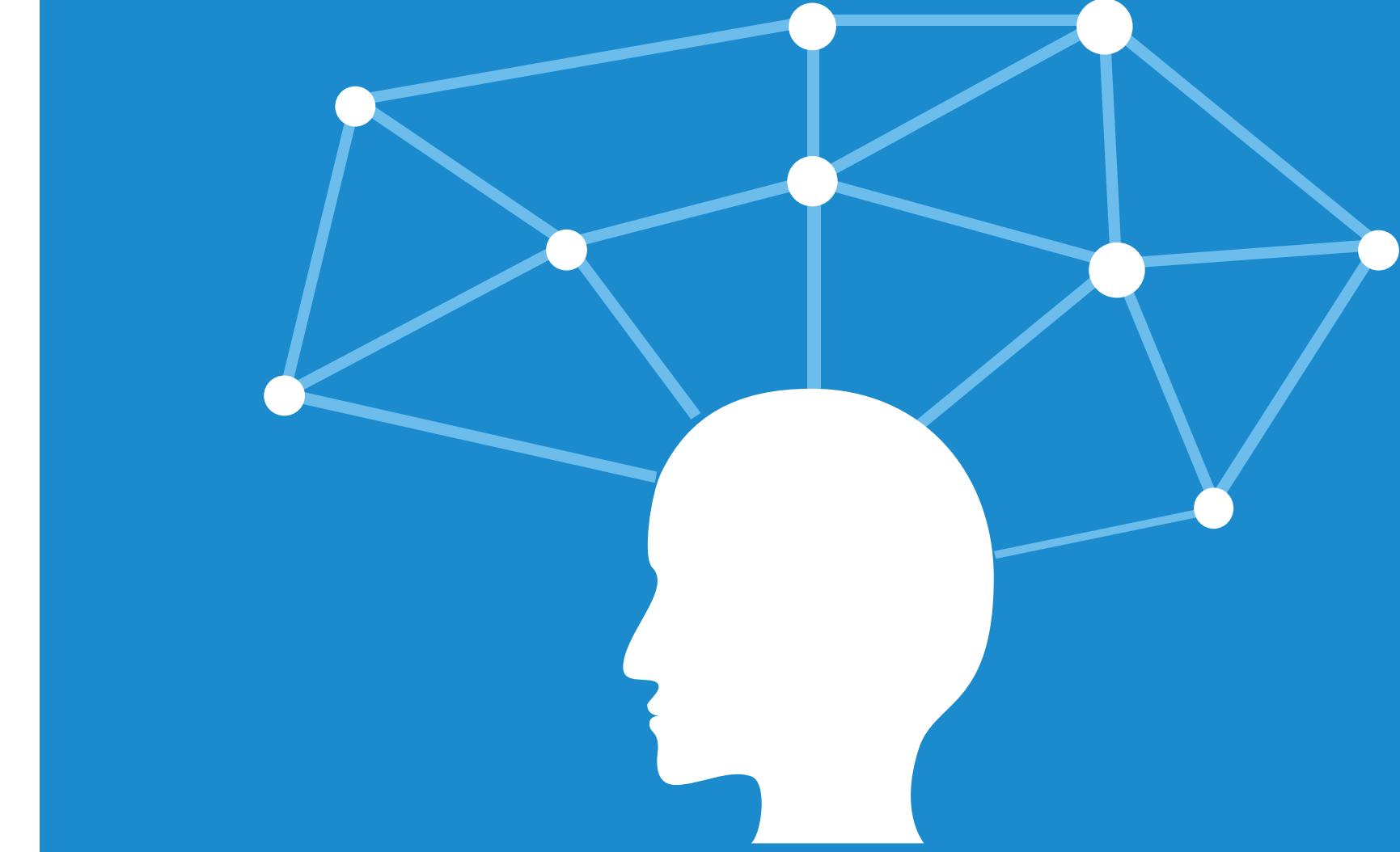
Ichatt

개발 : 2015년 10월
캐릭터 기반의 소통형 메신저



일종의 네트워크 소셜 게임(SNG)을 표방한 앱으로, 아바타
관련 꾸미기 컨텐츠, 카페 관리 기능 등 탑재.

입사 후 미출시 상태의 앱 보완하여 출시



lchatt

역할 및 이용 기술

개발 환경

개발

- JDK 8

개발 도구

- Eclipse Mars, Android Studio

Database

- Mysql, MariaDB(AWS)

이슈 관리 도구

- Redmine

형상 관리 도구

- Git

플랫폼

- Android, Linux Server(Unbuntu AWS)

협업 도구

- Slack

역할

- 총원 4명

- 외주 앱 개발 결과물인 안드로이드 앱 고도화

- 클라이언트 버그 수정

- 인앱 결재, 통신 모듈, DB 통신 수정 등

사용 기술, 라이브러리

- Spring 4.0, Spring JPA, Mockito

- Mybatis에서 Hibernate로 전환

- Android SDK

- Gradle

- AWS(EC2, RDS, Lambda, S3, Route etc)

- MQTT(RabbitMQ & ActiveMQ)

- Google Protobuf, Logstash, TDD

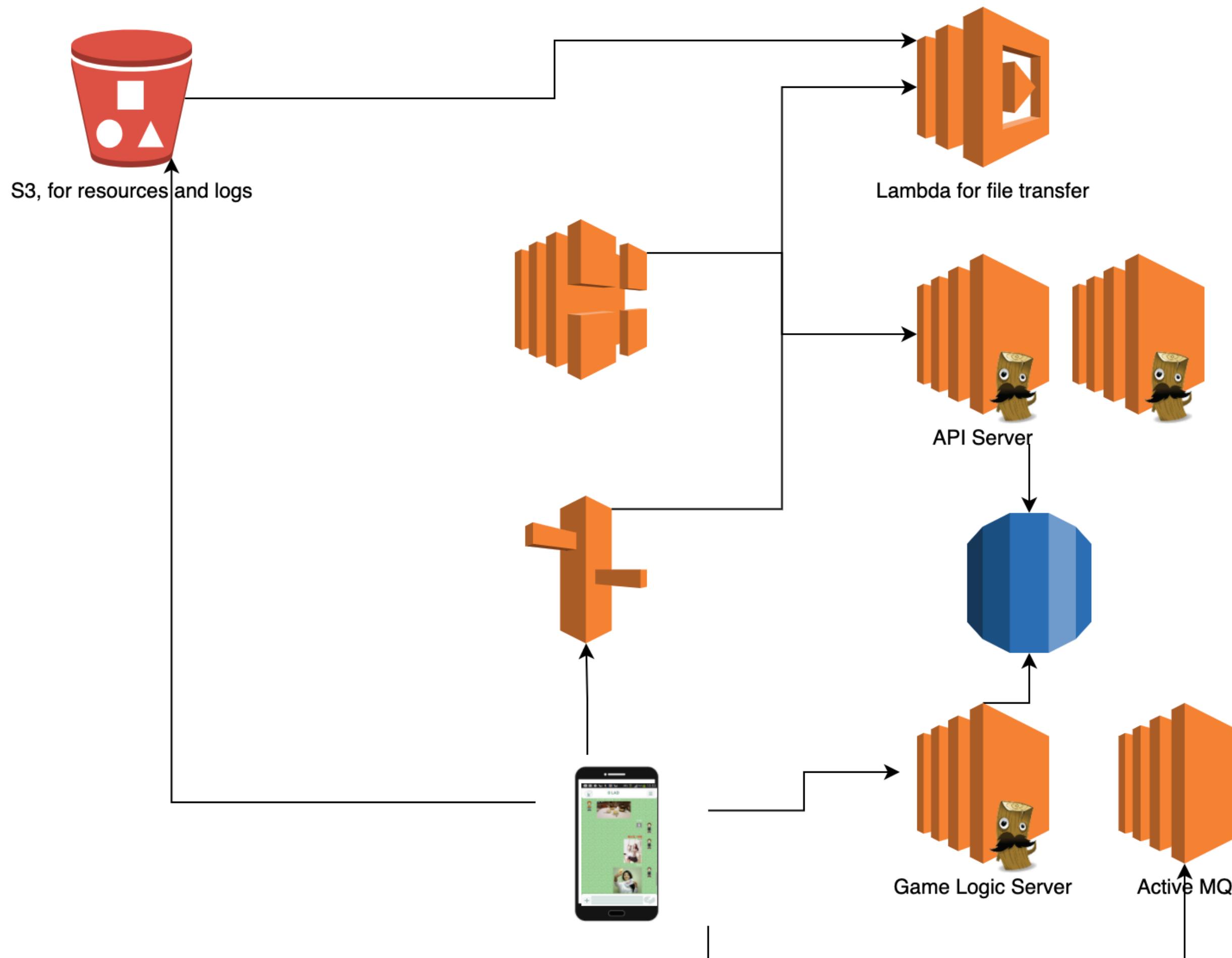
- Google Developer API

lchatt

기술 채택 사유

- Spring Framework 4
 - 웹서버 프레임워크로 채택
- Spring JPA
 - Mybatis 를 Hibernate로 교체하는데 사용, QueryDSL 등을 이용해 query-free 한 환경을 위해 채택
- Spring Mockito
 - 테스트 프레임워크로 사용
- Gradle
 - 라이브러리 의존 테스트 및 자동 태스크 수행
- AWS
 - Lambda 등 서버 운용을 위해 사용
- Message queue service
 - SQS 대신 Mqtt ActiveMQ, RabbitMQ를 이용하여 알림에 사용
 - SQS는 불안정했던 이력
- Google protobuf
 - Client-Server 통신에 이용, 통신 부하를 줄이기 위해 채택
- Logstash
 - 서버 로그를 S3에 적재하기 위해 사용
- Google Developer API
 - 앱 출시, 인앱 아이템 등 어플리케이션 관리를 위해 사용

lchatt



대략적인 초기 구성도

- 본 구성은 최종이 아니며 초기 다소 복잡하게 구성되어 변경 작업(구조 개선) 진행

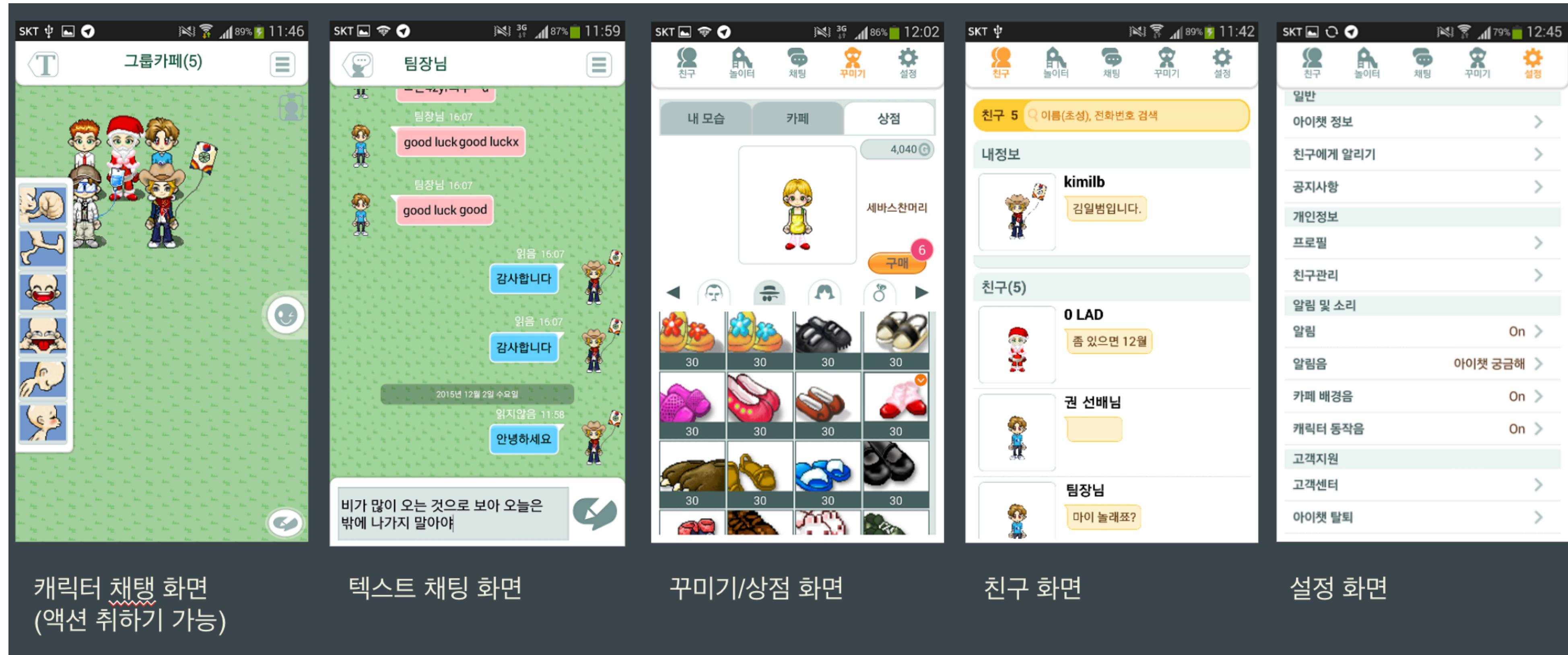
lchatt

작업 상세

- DB 접근 포인트를 하나로 축소. Game Server, Api Server 다수에서 이뤄지던 것을 단일 서버에서만 접근하도록 수정
- JMS 포맷으로 AWS SNS가 연동된 불필요한 메시징 제거
- API 서버가 복수의 ORM framework를 사용하고 있어 Spring JPA, QueryDSL 사용하는 것으로 통합
- Protobuf 통신으로 데이터 통신 오버헤드 간소화
- 통신 프로토콜 변경하여 불필요한 데이터 통신 제거
- Logstash 로 로그 S3 적재
- Google InApp Purchase 기능 추가
- 게시판 기능 추가
- AWS 리전 Tokyo에서 Seoul로 이전
- Seoul Region에서 Lambda가 지원되지 않아서 Lambda 기능 제거 후 대체
- SNS 서비스 사용 일절 제거
- 앱 자체 UX 이슈 수정
- ‘더보기’ 관련 앱 내 기능 확장
- Oauth 2.0을 통해 Google 로그인 기능 추가
- DB JPA 전환에 따른 테이블 구조 변경
- SQL 스크립트 관리 시작
- Push 구조 개선
- 리팩터링
- 쿠폰 시스템
- Version 1 런칭

lchatt

스크린샷



lchatt

고찰

- 코드의 가독성과 잘 정의된 명명 규칙 및 성실한 준수가 큰 도움이 됨을 확인
- 개인의 잘못된 코드로 인해 다른 사람들이 고생하는 것을 경험하여 개인의 책임 중요성 크게 확립
- 구조적 문제로 인해 수정 시 사이드 이펙트 영향 확인, 코드 이력 철저함 필요성 인지
- Rabbit MQ와 같은 메시지 큐를 통해 Push 기능 구현 시 QOS 지원 수준에 따른 이슈 발생
 - 통신 구조 등 대폭 수정했으나 재개발 수준의 규모로, 초기 설계 중요성 확인
- 중복된 코드, Dead code 존재
 - 없어야 할 코드가 무엇인지 인지하여 Clean Code에 대해 고찰하게 됨
- Redmine, Slack, Git 통한 협업 효율성 및 코드 관리 효과 경험
- 코드 명명법 및 작성 전반에 대한 규칙 제정으로 가독성 통일성 향상
 - 타인 코드 리뷰 및 분석에 드는 시간 감소
- 처음 접하는 기술에 대해 주 1회 1~2시간 스터디 및 고찰회의를 통해 도입 유무를 결정
- 학습 커버리지와 일정간 조율 필요성 확립

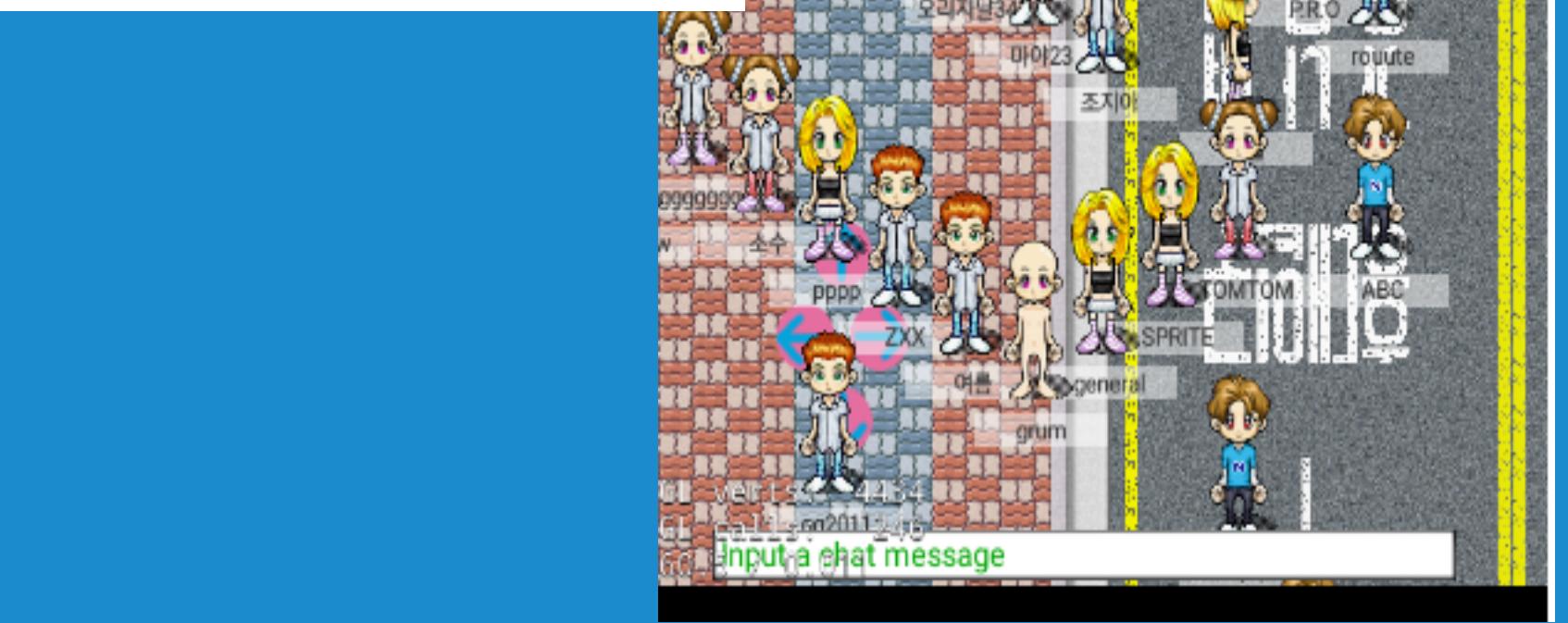
Noriter

개발 : 2017년 01월

모바일 기반의 대규모 광장형 커뮤니티 서비스

‘놀이터’라 명명한 공간 내에서 익명의 다수가 모여 만남을 갖고 대화를 나누며 이벤트를 진행하고 상호작용하는 생활형 게임 서비스

메시지 처리 중심의 비동기 서버. SNG.



Noriter

역할 및 이용 기술

개발 환경

개발

- JDK 8

개발 도구

- Spring STS

Database

- Mysql, MariaDB(AWS)

이슈 관리 도구

- Redmine

형상 관리 도구

- Git

빌드 시스템

- Gradle

협업 도구

- Slack

역할

- 총원 5명
- Pair programming 을 통해 개발
- = 설계부터 구현 및 코드 검증 테스트까지 모든 과정에
붙어서 참여함. 즉 전체 개발 과정 참여

MSA 를 통해 서비스 관리 서버들을 같이 개발

클라이언트 외부 개발자와 Slack 을 통해 협업함

Noriter

사용 기술

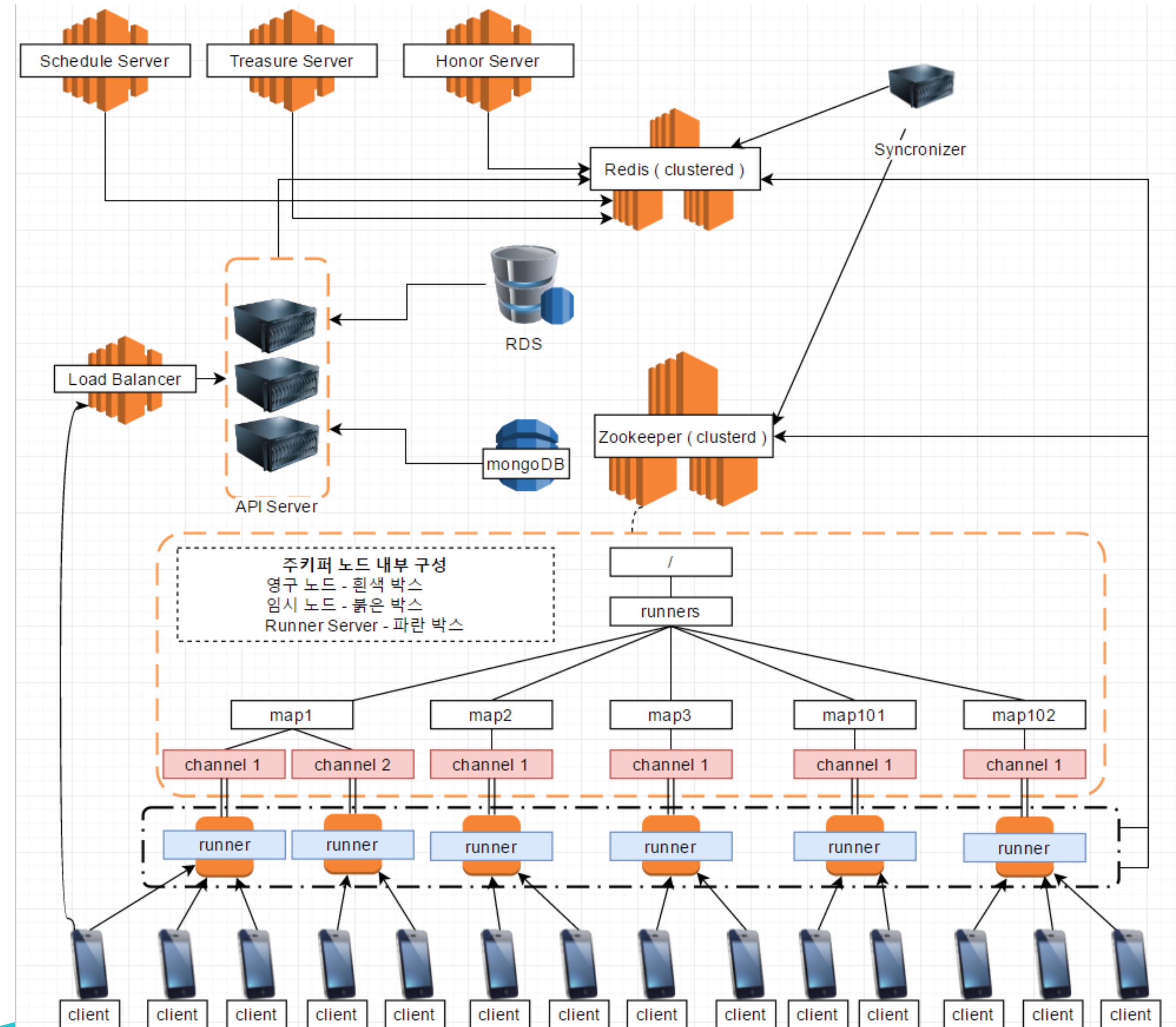
- Git Flow
 - 소스 이력을 더 효율적으로 관리하고자 하용. NHN 컨퍼런스 때 알게되어 시도하게 됨
- Spring Boot
- Netty
 - 비동기 서버 통신을 위해 채택. 이전 프로젝트에서도 사용한 경험 있어 빠른 개발 가능
 - 기능 확장에 열려 있어 사용
- Protobuf
 - 클라이언트와 경량이 통신을 위해 사용. 실시간 통신 이뤄져야 하여 Json은 너무 커서 각하하고 Byte 단위 통신을 위해 채택
- Redis
 - 분산 처리를 위해 Stateless한 구조로 설계한 서버에서 데이터를 관리하기 위해 사용. 공용 메모리로써 사용하는 것이 주된 목적. 전체 사용자 관리를 위함
- Zookeeper
 - 분산 배치된 서버의 내구성을 위해 선택. 이 외에 외부 API 서버와 같은 다른 구성서버들을 관리하기 위해 사용
- React Core
 - 비동기 이벤트 주도 방식으로 로직을 처리하기 위해 사용. 상태를 가지지 않으면서도 빠른 처리를 위해 채택

Noriter

사용 기술

- Kafka
 - 서버간 메시지 중계기로써 사용. 로그 데이터의 적재 및 추적에 채택하여 사용. ZeroMQ, RabbitMQ 후보군 중 전자는 문서 및 요구하는 기능 미제공, 후자는 처리량 문제로 제외.
- AWS
 - 서버 인스턴스 관리 및 로그 관리를 위한 클라우드 서비스 사용
- Akka
 - 신뢰성 있는 비동기 처리 서버 개발을 위해 적용, 이 기술로 외부 컨텐츠 서버가 구축되었고 이후 설계에 도입 여부를 위한 지식 마련
- MongoDB
 - 사용자 로그를 저장하기 위해 사용
- Web
 - 관리자 페이지 사용을 위한 기술 일체.

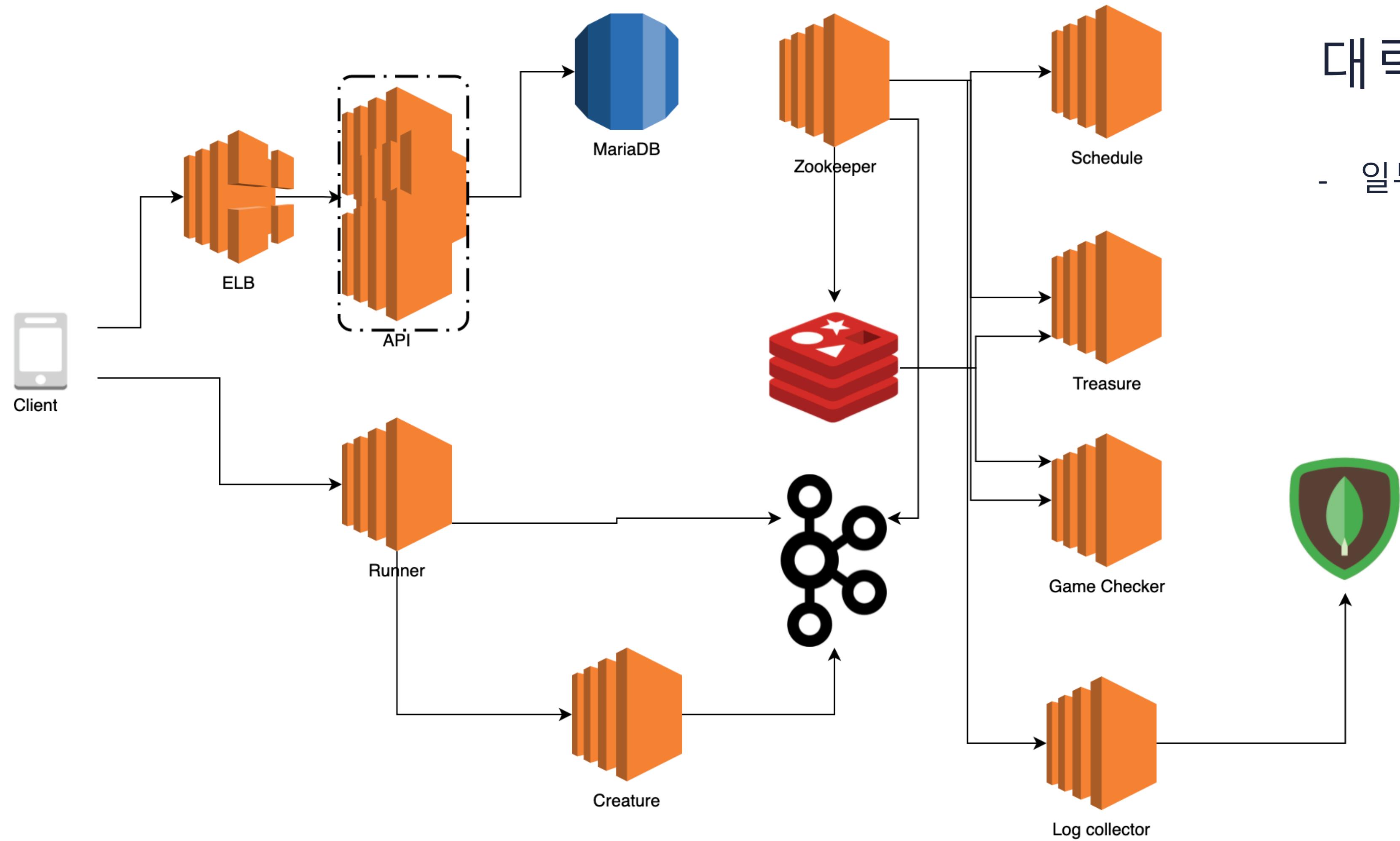
Noriter



대략적인 구성도

- 일부 생략된 부분이 존재합니다.

Noriter



대략적인 구성도

- 일부 생략된 부분이 존재합니다.

Noriter

설계

Scale In/Out이 자유로운 서버를 개발하는 것이 주된 목표

각각의 서버는 Channel 하나에 대응되며 주키퍼를 통해 정보를 제공 받는 식으로 개발

고려 사항

비효율적인 EC2 인스턴스 사용 경험 방지 -> 비용 증가(실 구동 시간 << 대기 시간)

공통 처리부분에 대한 코드 중복 -> 개발 지연

실제 사용자 데이터 저장소 접근 제한에 따른 서버 기능 확장의 비 용이성

개진 의견

불필요한 서버 분할 억제. 분리만이 최상은 아님 -> 기존 컨텐츠 서버는 통합

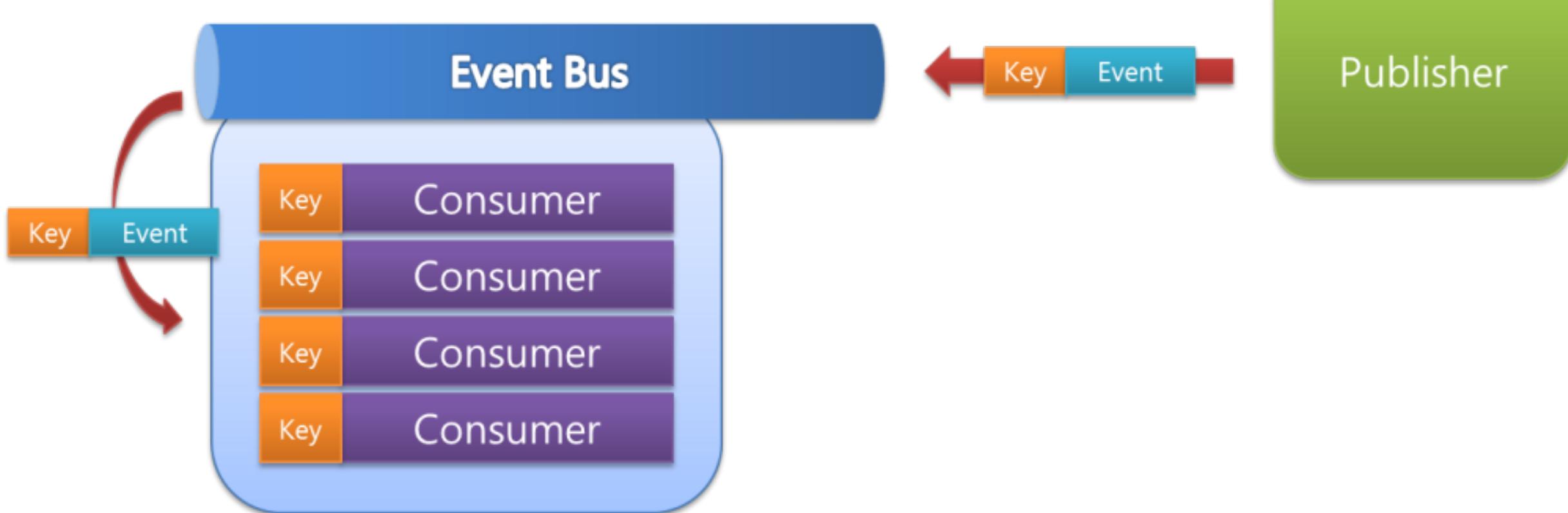
공통 프로젝트 대상 확장(기존 타 개발자가 코드를 묶어 포함 프로젝트로 둠으로써 코드 중복 제거)

사용자 데이터 저장 위치 변경(マイ그레이션) -> RDS 접근이 제한된 구조를 변경하지 못하는 경우

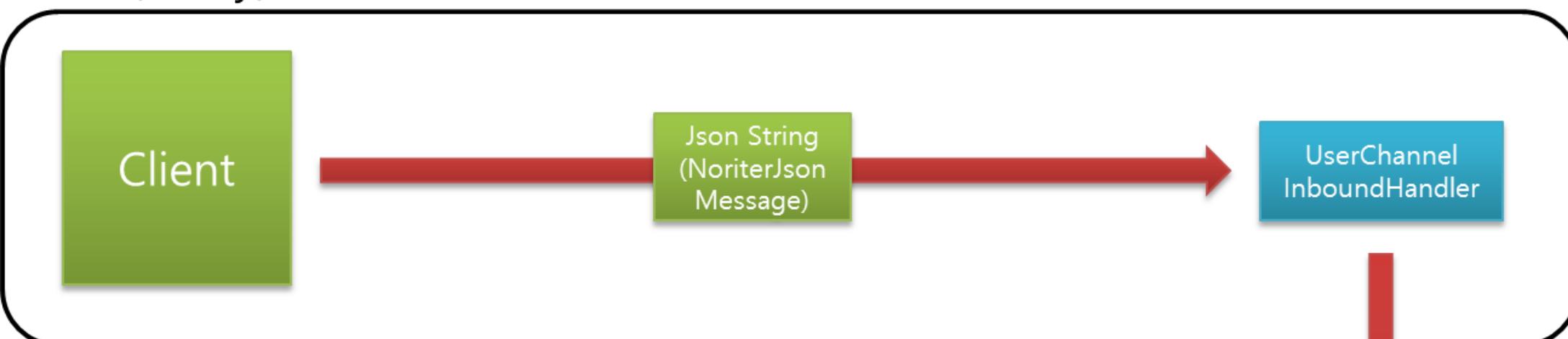
NoSQL 또는 Cache를 두는 개선안 제언

Noriter

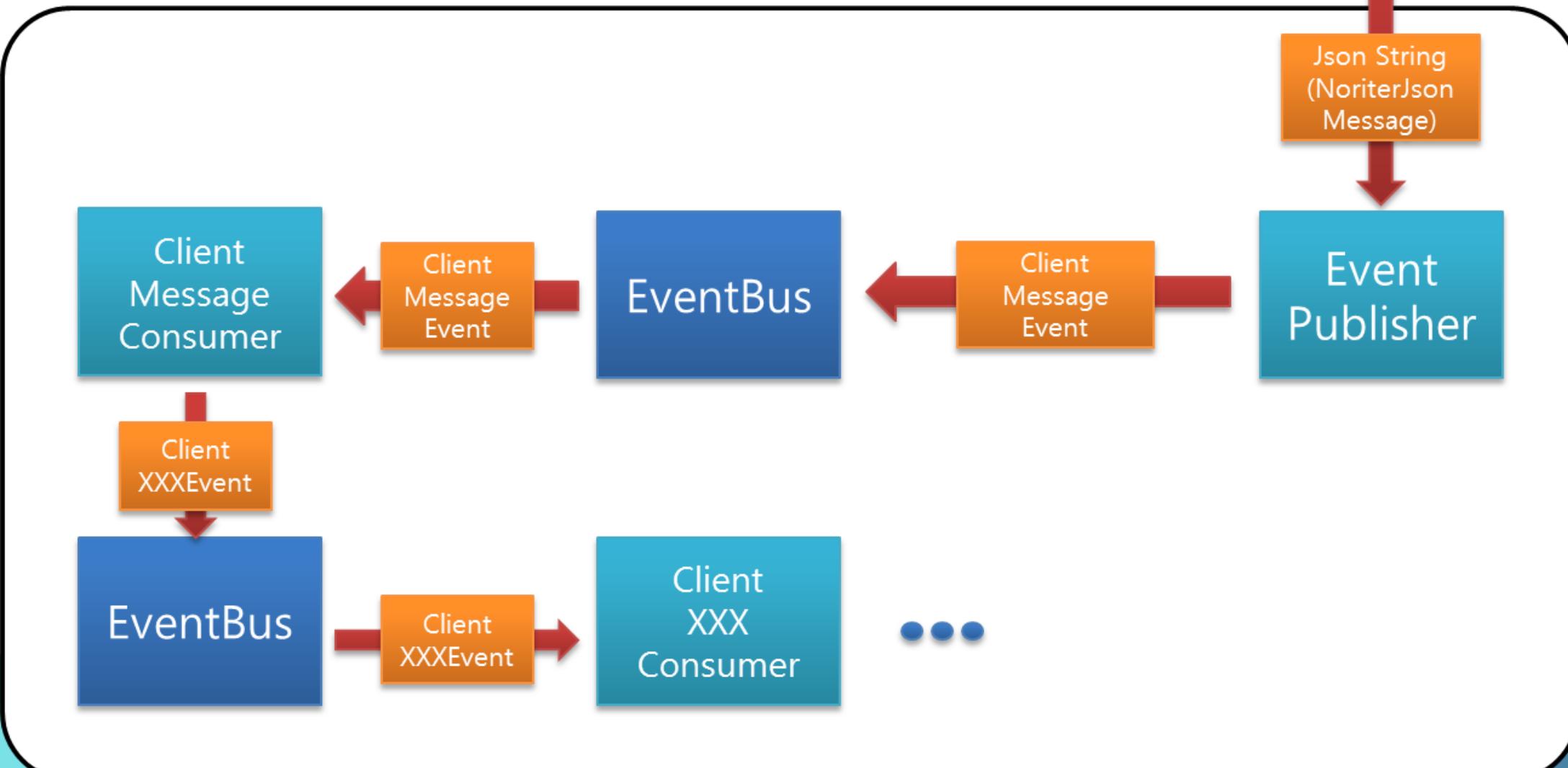
Event Bus



TCP(Netty)



Reactor



React Handler

TCP 레벨에서 클라이언트는 JSON으로 통신

수신 이벤트를 분석해야 하므로 초기에 ClientMessageConsumer가 처리하고 여기서 분별된 이벤트는 각각 알맞은 처리기로 전송돼 처리됨

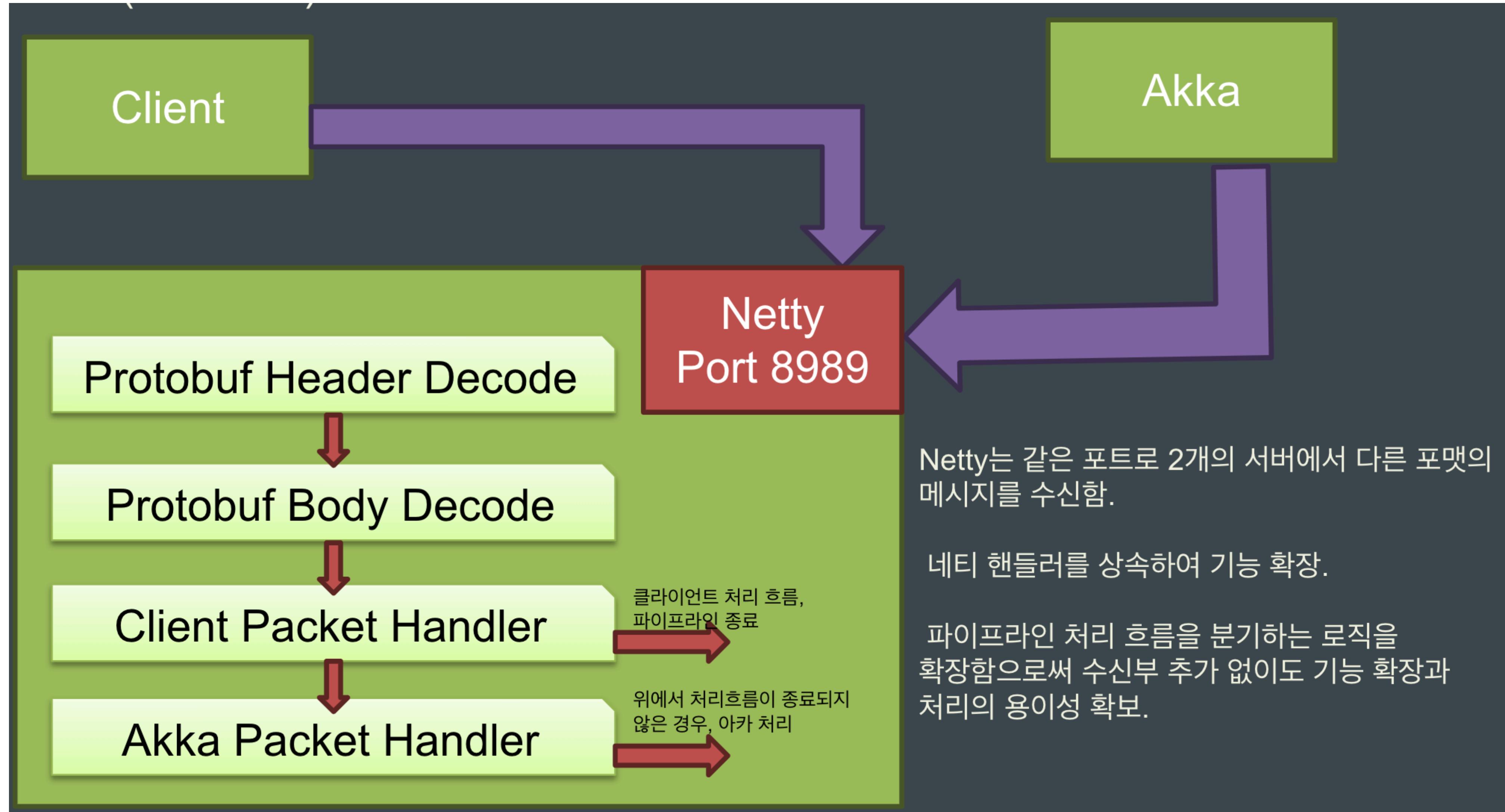
각 처리기에 비동기적으로 동작할 필요가 있는 경우,
새로운 이벤트를 생성하여 Event Bus에 던져
처리하도록 구성

Reactor를 기반으로 하여 모든 처리를 상태 없이 처리
하는 방식으로 개발

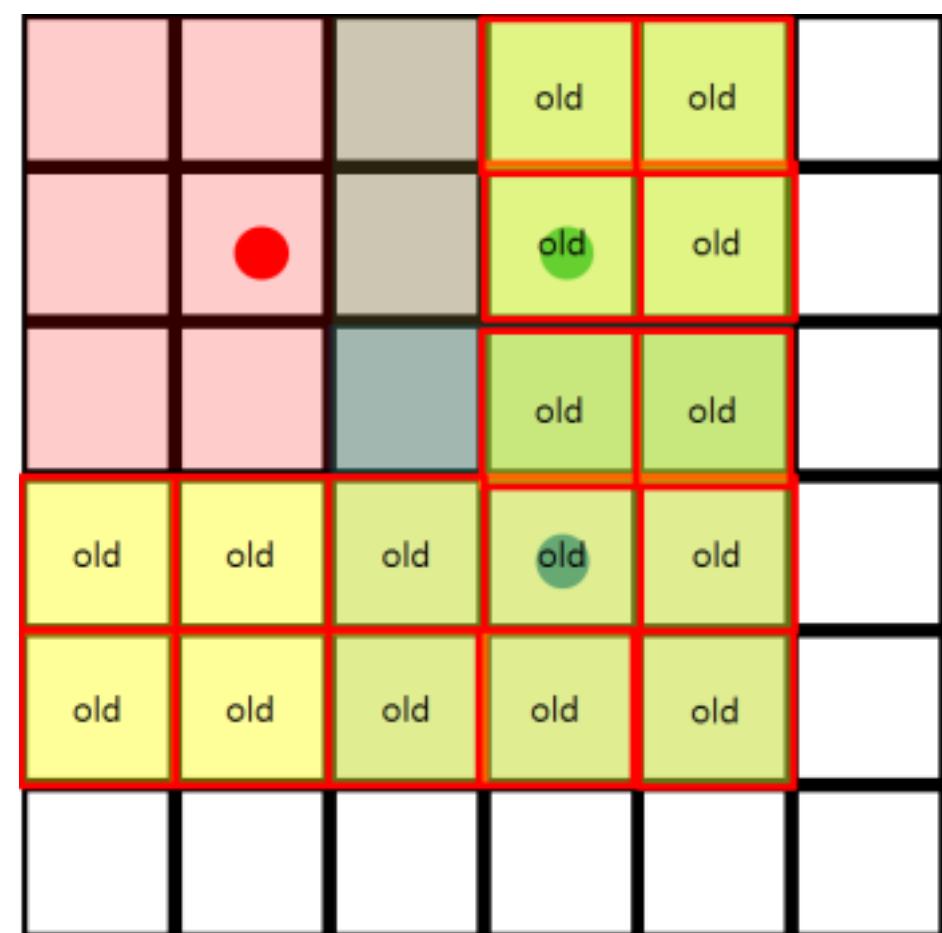
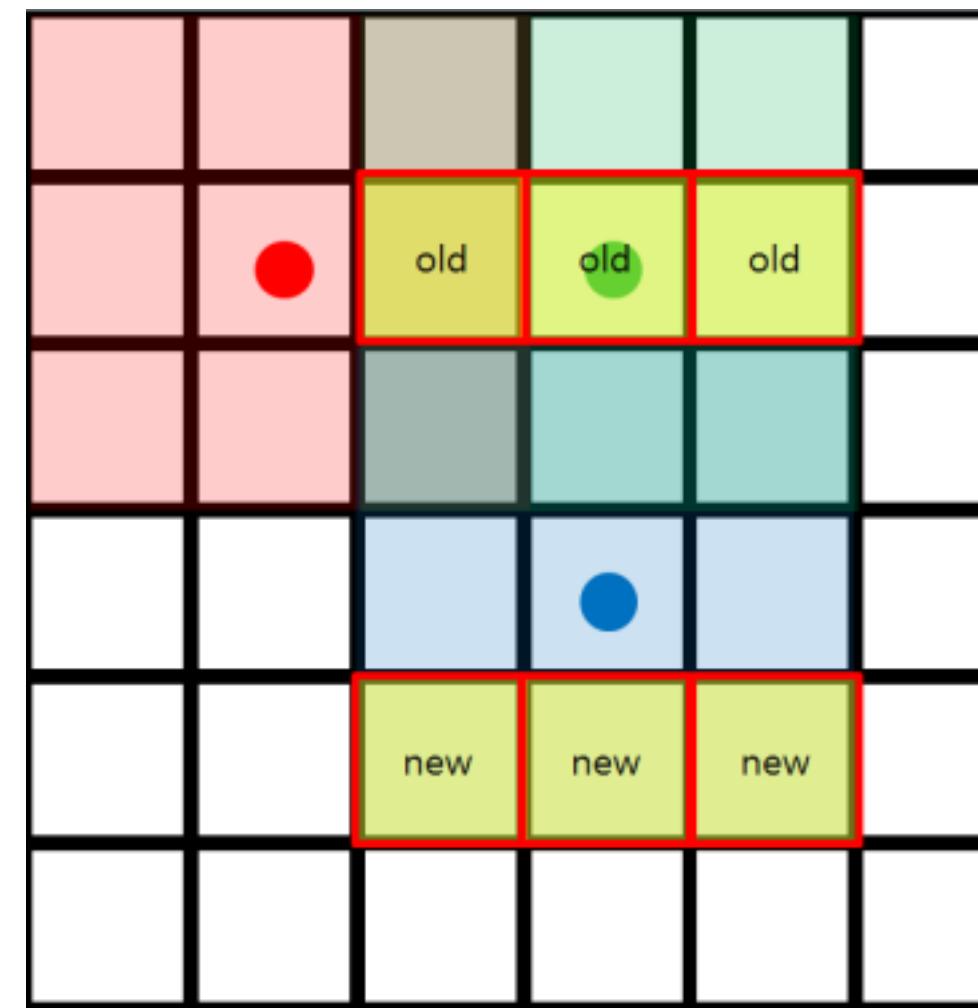
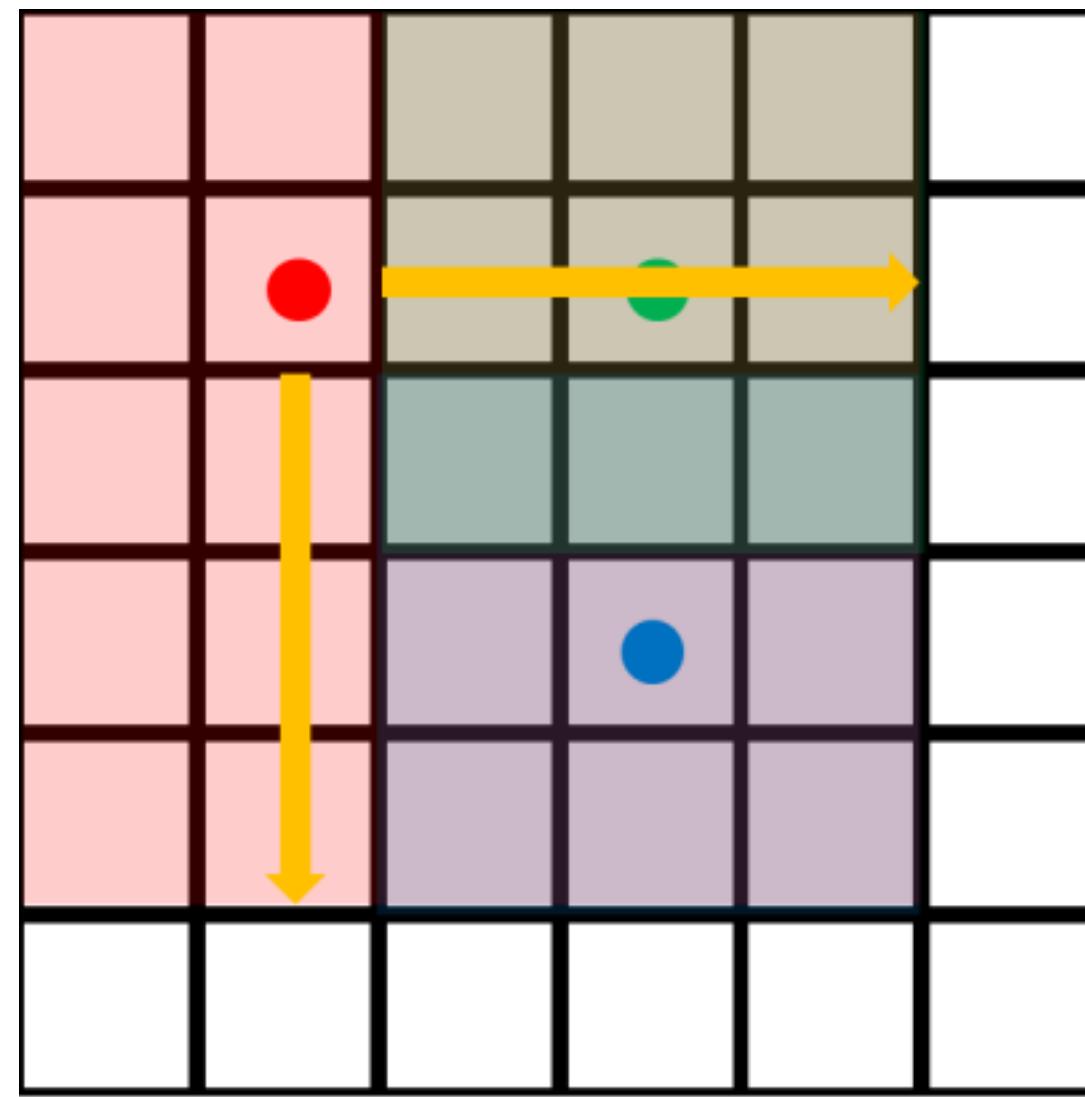
Event Bus를 생성하여 Consumer를 등록

Key 를 기반으로 등록하며 Publisher에서 송출한 메시
지의 Key에 따라 처리 Consumer 지정

Noriter



Noriter



통신 채널 관리

수백명이 실시간으로 통신을 하며 서로의 정보를 간
신하는 시스템에서 통신 부하를 최대한 효율적으로
하면서도 자연이 없도록 고려해 설계

타일 기반으로 사용자가 배치되는 점에 착안

한 셀의 데이터를 청크(chunk)로 정의하고 한 개인의
사용자가 소유한 청크의 양과 변동에 따라 통신할
대상을 필터링하도록 함

이 청크에 대해서만 통신을 주고 받고 관심 영역 밖의
데이터는 수신하지 않음

Chunk 관리 요점

- 사용자는 로그인 시 자신의 관할 청크 범위를 서버에 제출하며, 서버는 이 범위를 통해 사용자가 확정할 가시내의 사용자 정보를 제공해야 한다.
- 사용자는 위치의 변경이 있을 때마다 서버에 이를 통보하며, 서버는 매순간 사용자의 관할 청크가 변경되는지를 감시한다.
- 청크에서 관리될 정보는 관할자 목록과 내부자 목록 두 가지가 있다.
- 관할자 목록이란 해당 청크를 가시범위로 두고 있는 사용자의 목록을 뜻하며, 내부자 목록이란 해당 청크상에 위치하고 있는 사용자의 목록을 말한다.
- 서버는 매순간 청크에서 관리되는 두 목록 정보를 감시하며 변동이 있게되면 이를 사용자에게 통지할 의무를 지닌다.

Noriter

Redis를 사용하여 효율적으로 데이터를 저장하고 가져오는 방식 고려,

Redis의 키는 아래의

- 맵 목록 관리
- 채널 목록 관리
- Runner 식별
- Runner 상태
- 통지

- 전체
- 단일 맵
- 단일 채널
- 퀴즈 서버
- 보물 서버

- 퀴즈 저장

퀴즈 다 퍼져

키를 설계하고 Pub/sub 기능을 통해 실제로 필요한 경우에만 그러나 Zookeeper의 동작에

방해되지 않는 히든 채널로써 사용

Runner 상태

한 개의 Runner 서버를 특정하는 키로, 한 개 서버 즉, 한개 채널에 대응한다.

Runner 가 퀴즈를 발의하는 경우 이 값을 사용하며, 3가지 상태 전이(ready, process, done)를 진행한다. 이 키의 존재

등록	Runner 서버
삭제	Runner 의 퀴즈 결과 처리가 끝난 경우, by GameCheckServer
값 업데이트	Runner의 퀴즈 개시 상태에 따라 변경, by Runner
사용처	Runner, Quiz

키값	타입	설명
quiz:all:map:{%d;mapIndex}:channel:{%d:channelIndex}:state	String	퀴즈 개시 상태값을 저장한다. ready, proc

예) String 타입으로 get 명령을 통해 조회 가능.

```
172.30 172.30:7003> get quiz:all:map:101:channel:1:state  
-> Redirected to slot [936] located at 172.30 172.30:7001
```

Noriter

통신 채널의 경우 Runner 사이에서 통신하는 메시지에서만 Redis Pub/sub 사용

There are three types of channel to publish based on its delivering object.

1. To All

Used to deliver message to all active user e.g) notice

channel name	message:notice
--------------	----------------

이외는 Kafka의 Pub/sub을 응용하였음

Redis는 String 기반의 데이터를 저장하므로 바이너리 데이터를 전송할 때 별도의 작업이 부가적으로 필요함

2. To One Channel

Used to deliver message to only one channel locally

channel name	message:map:{%d}:channel:{%d}
Parameter	#1 : map index #2 : channel index

3. To One Map

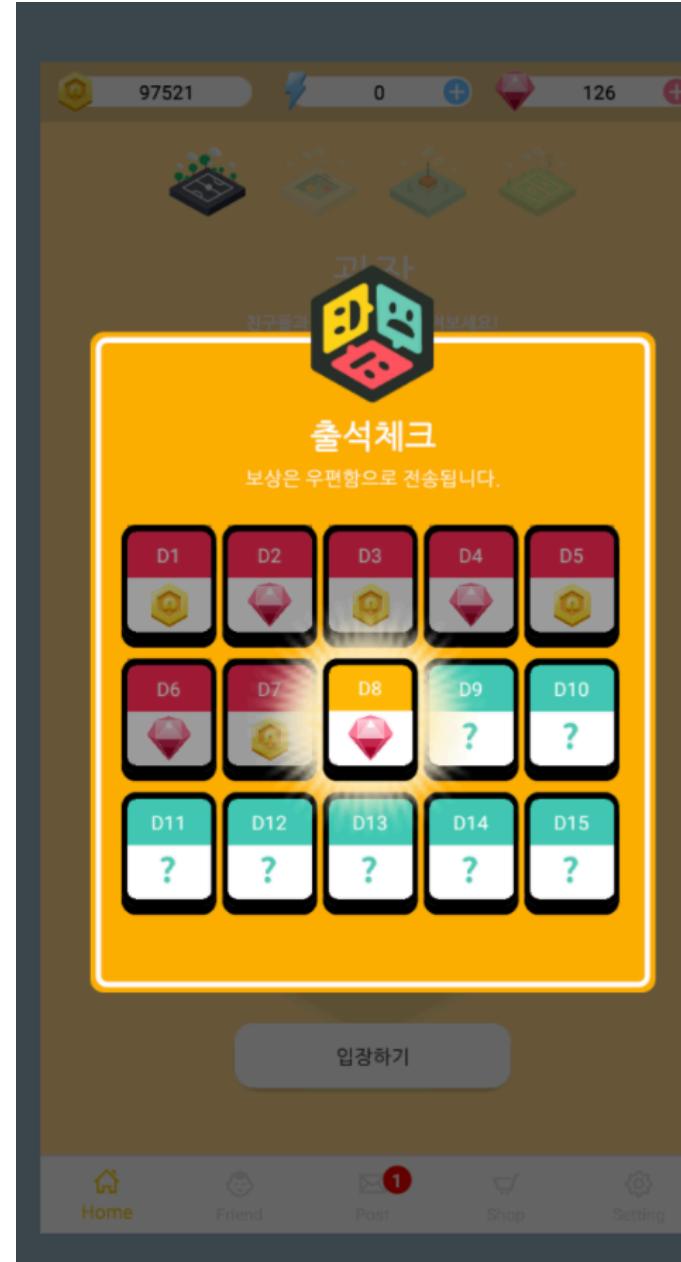
Used to deliver message to only one map and all the channel in it.

channel name	message:map:{%d}
Parameter	#1 : map index

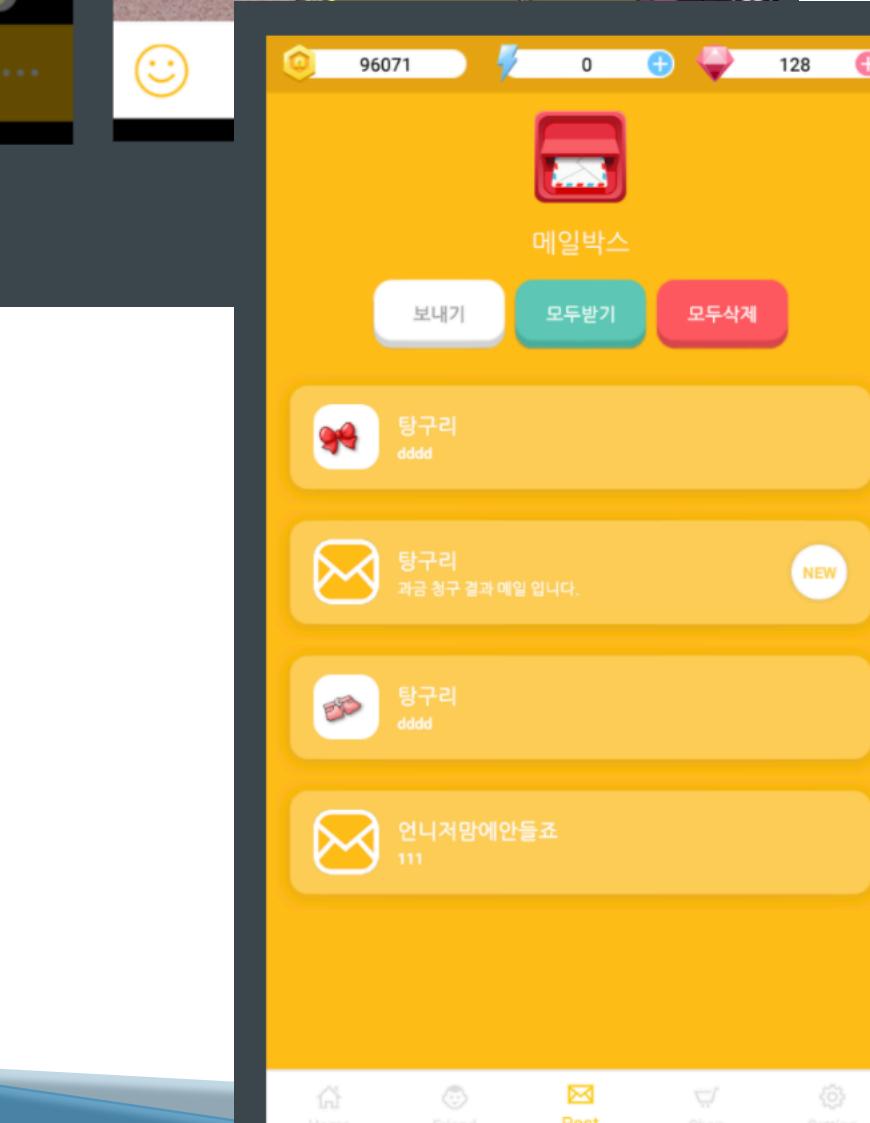
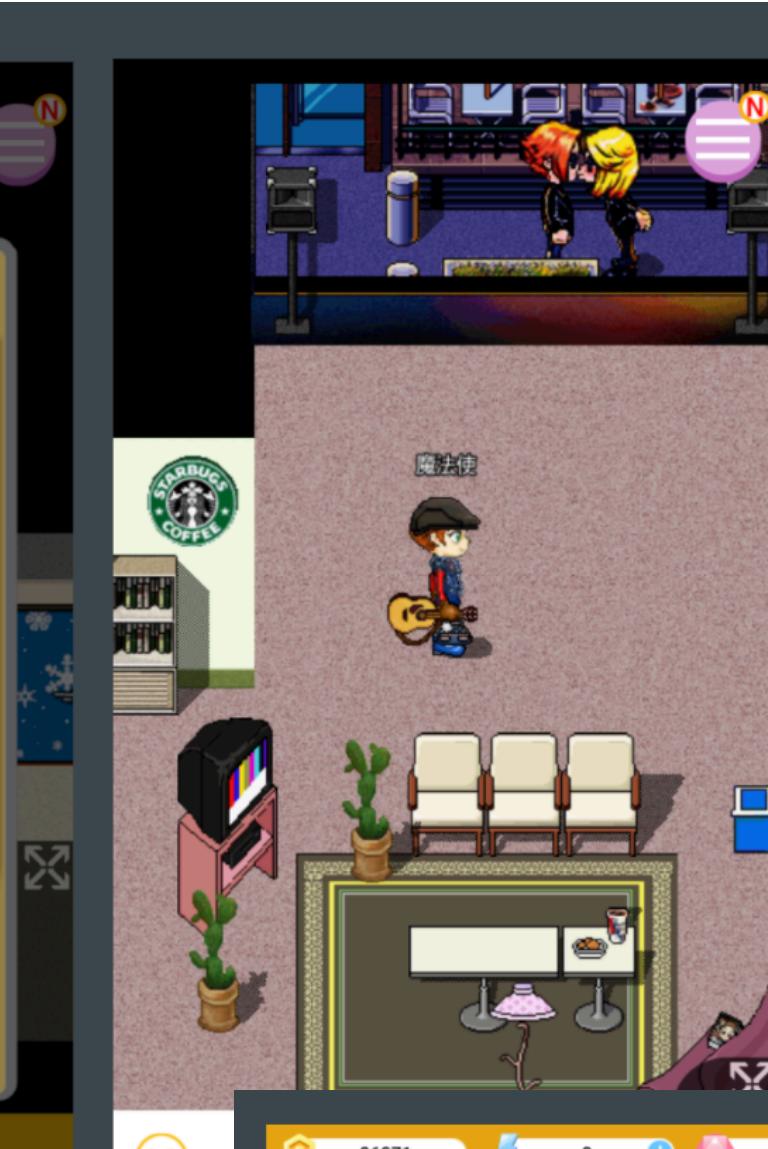
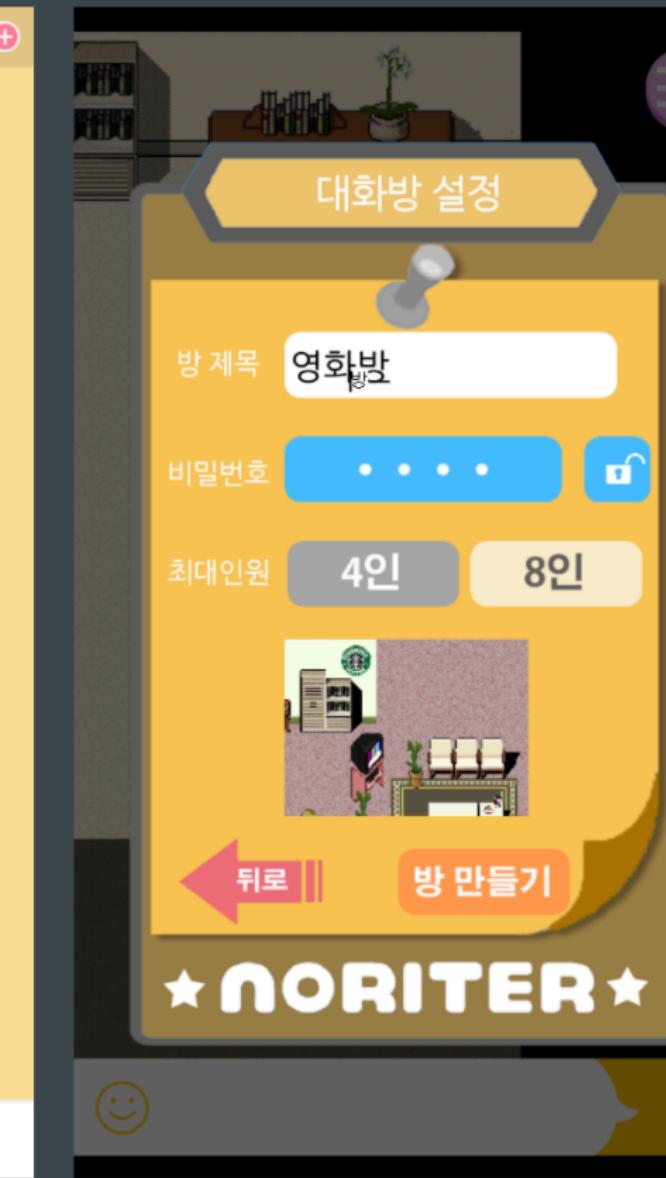
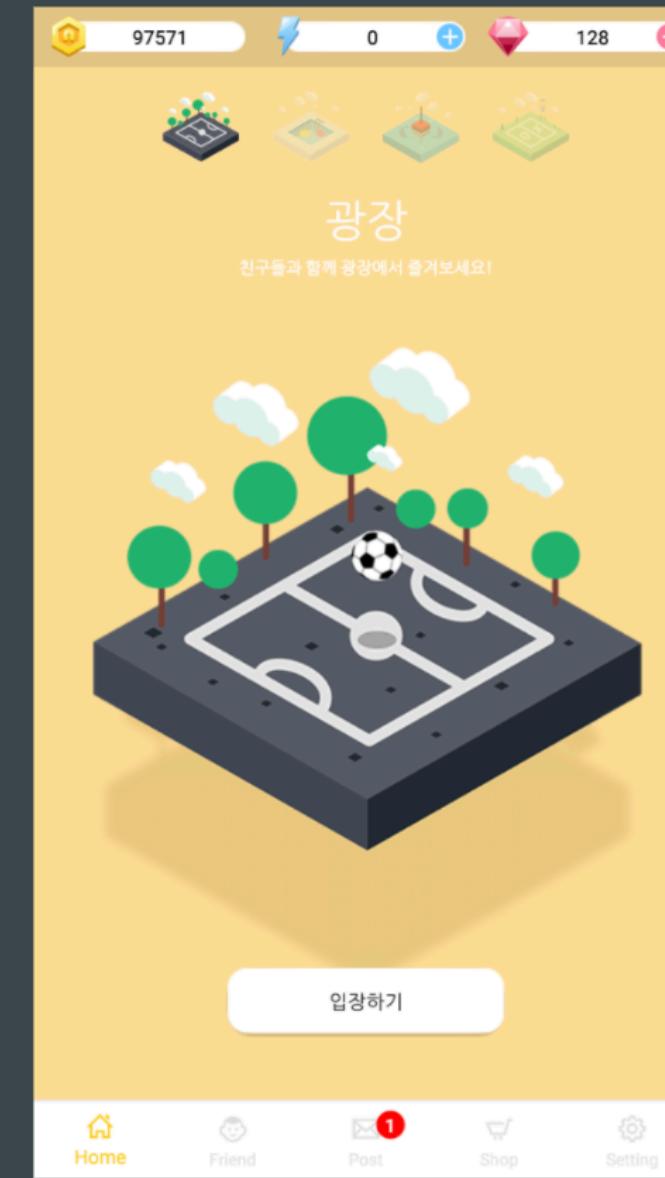
별도로 외부에 Zookeeper 노드를 감시하는 주체를 주도 노드에 문제가 감지되면 이를 파악하여 Redis 공용 공간에 결함이 없도록 처리함

Noriter

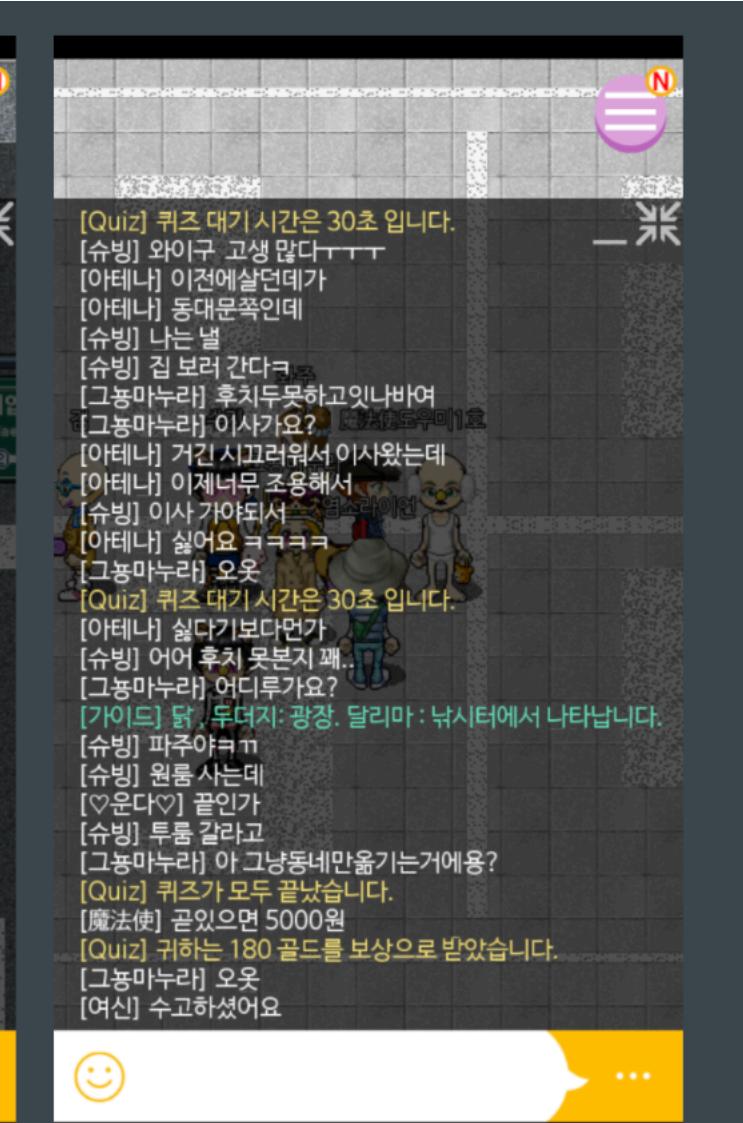
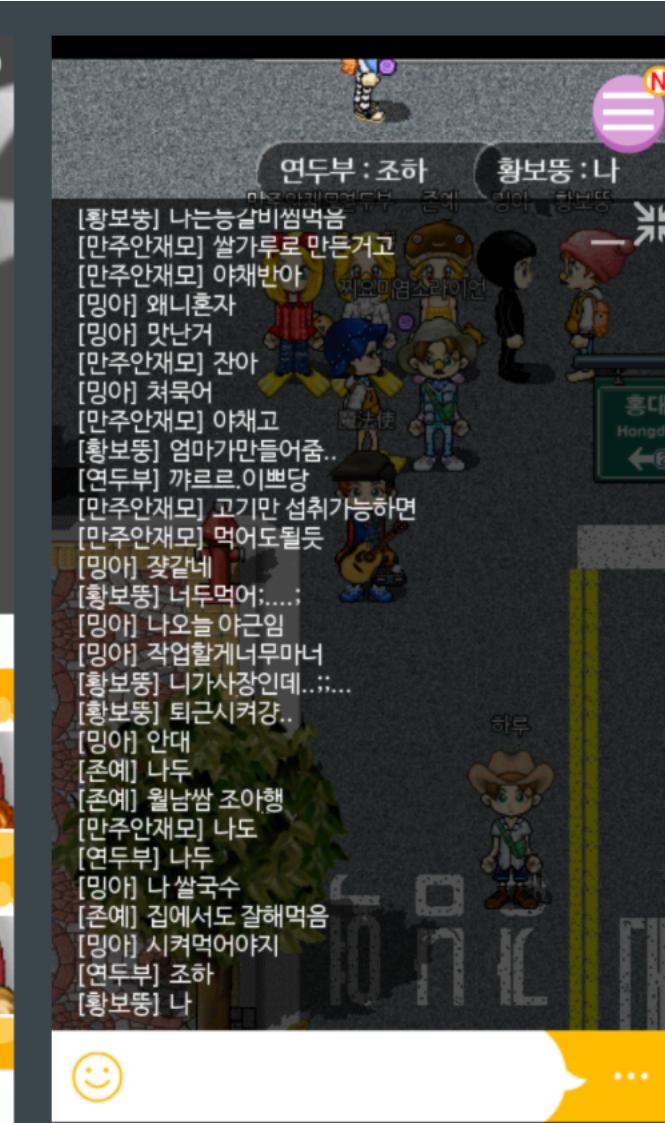
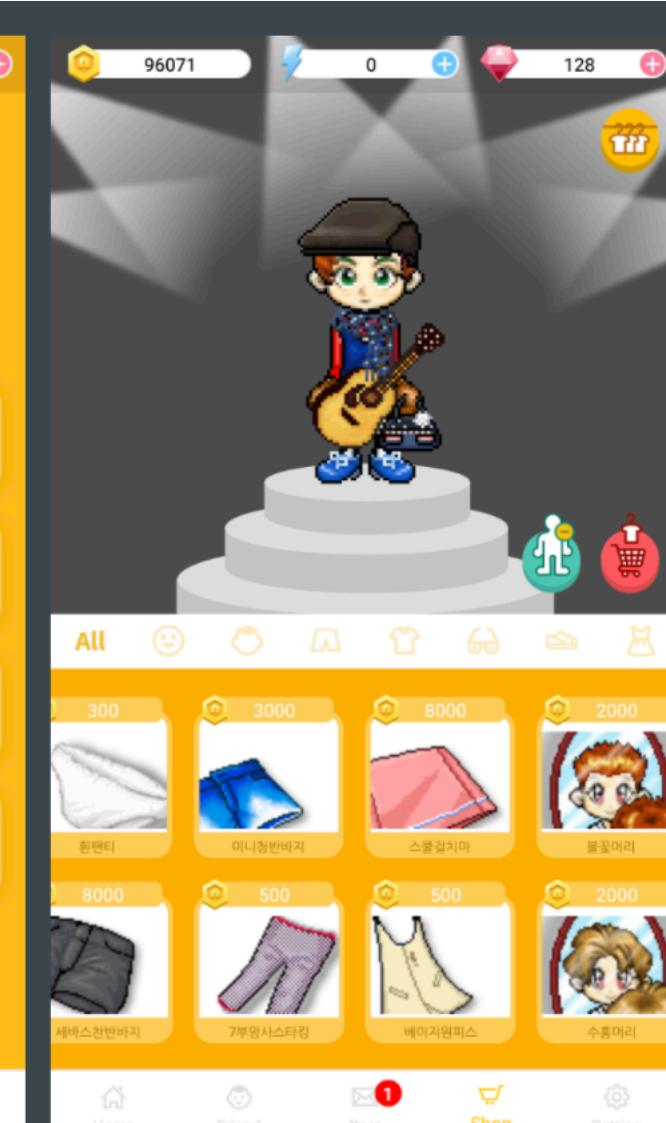
스크린샷



주요 플레이 화면(클라이언트 측)



주요 플레이 화면(클라이언트 측)



Noriter

작업 상세

- 필요 데이터 통신 프로토콜 설계
- Test Program 제작
- 기본 메시지 처리 기능 개발
- Zookeeper를 통한 노드 관리 기능 추가
- Redis를 통한 사용자 및 공용 데이터 정보 저장/조회 기능 추가
- 서버가 죽었을 경우, Redis의 정보를 정리해주는(동기화) Synchronizer 개발
- Kafka pub/sub을 통한 서버간 메시지 송수신
- Kafka Topic을 통한 Log 수집(MongoDB)
- DynamoDB를 통한 Mail 저장소 구축
- 관리자 사이트를 통한 서비스 관리 기능
- 기타 Features
- 개발 리팩터링(2018. 03. 28 ~ 퇴사 전)

Noriter

고찰

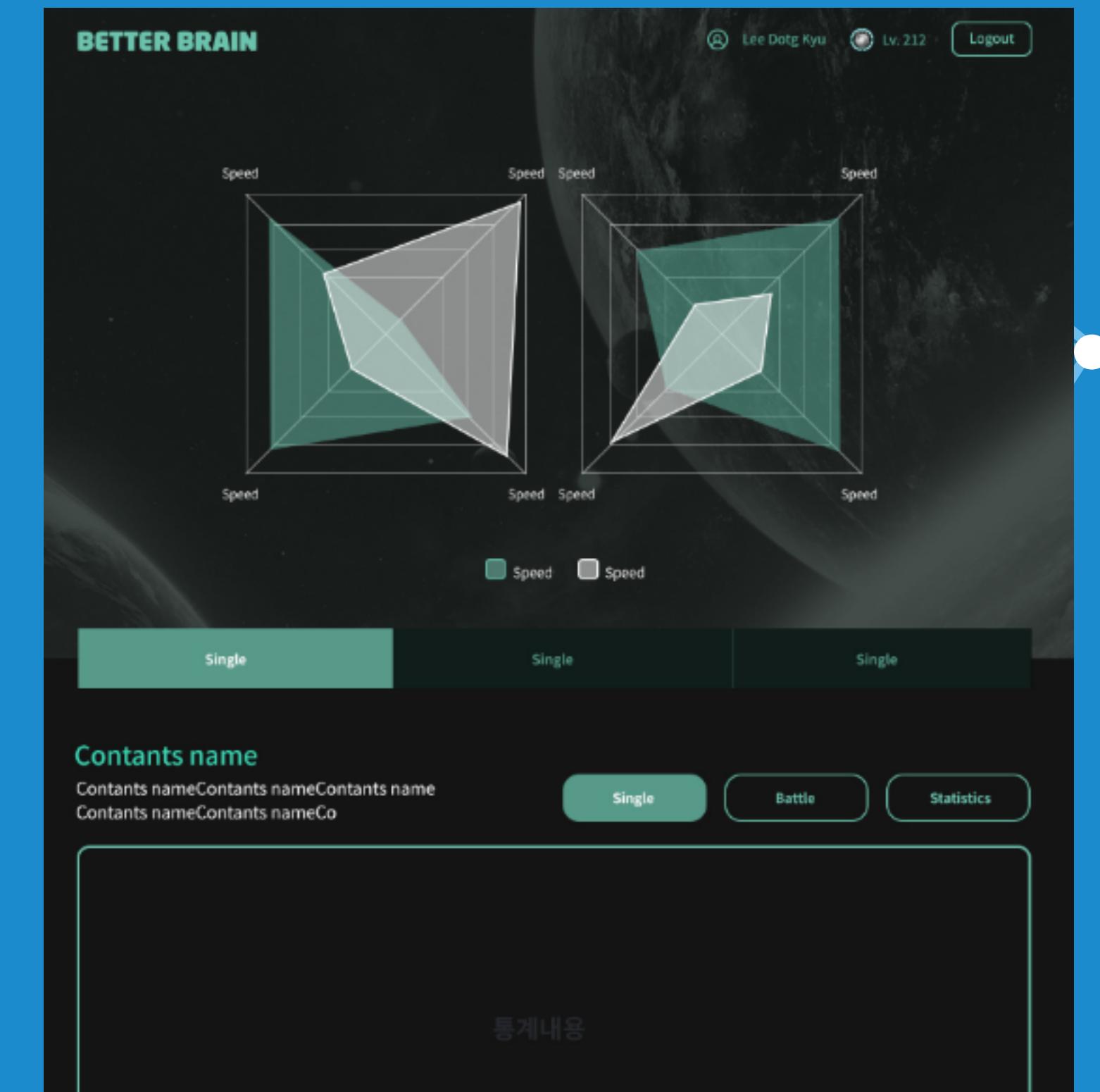
- 페어 프로그래밍을 통해 개발 부분 전반에 파악 가능했으나 개발 시간은 증가함
- 모바일 환경에서는 기지국 변경으로 네트워크가 끊기는 경우가 많음 자연스러운 연결 처리 필요
- 모바일 네트워크 끊김이 인지되지 않는 경우가 있음 서버의 TCP keep-alive 설정 조율도 필요할 수 있음
- 비동기 방식의 서버 구축 미숙한 시기 개발 시간 증가
- 클라이언트 개발 지연이 발생하는 경우 서버 제동이 걸려 확고한 일정 조율의 필요성 확인
- Redis, Zookeeper 같은 기술을 통해 서버 자체뿐이 아닌 전체 구조가 고려되는 설계를 통해 시야 확장
- 설레발 개발 위험성 인지
- 오버 아키텍처 지양, 주의 필요성
- 로그 중요성 확인
- 개발자간 소통 중요성 확인
- 사용자 소통의 신중함은 진리
- 상태 없는 코딩에서 확장되어 함수형 패러다임에 대한 관심 확보

Better Brain

개발 : 2016년 09월

교육용 컨텐츠를 중심으로 사용자간 실시간으로 경쟁하는
반응형 웹 서비스

모바일 혹은 웹에서 국어, 영어, 수학, 과학 등 교육용 관련 컨텐츠를 학습함과 동시에 자신의 실력을 실시간으로 동시에 접속한 타 사용자와 경쟁하며 문제를 풀기 위한 서비스 목적



Better Brain

역할 및 이용 기술

개발 환경

개발

- Javascript, HTML

개발 도구

- WebStorm

Database

- MySQL, MongoDB

이슈 관리 도구

- Redmine

형상 관리 도구

- Git

협업 도구

- Slack

역할

- 총원 4명
- 클라이언트 개발
- 서버 경쟁 서비스 개발

사용 기술

- NodeJs

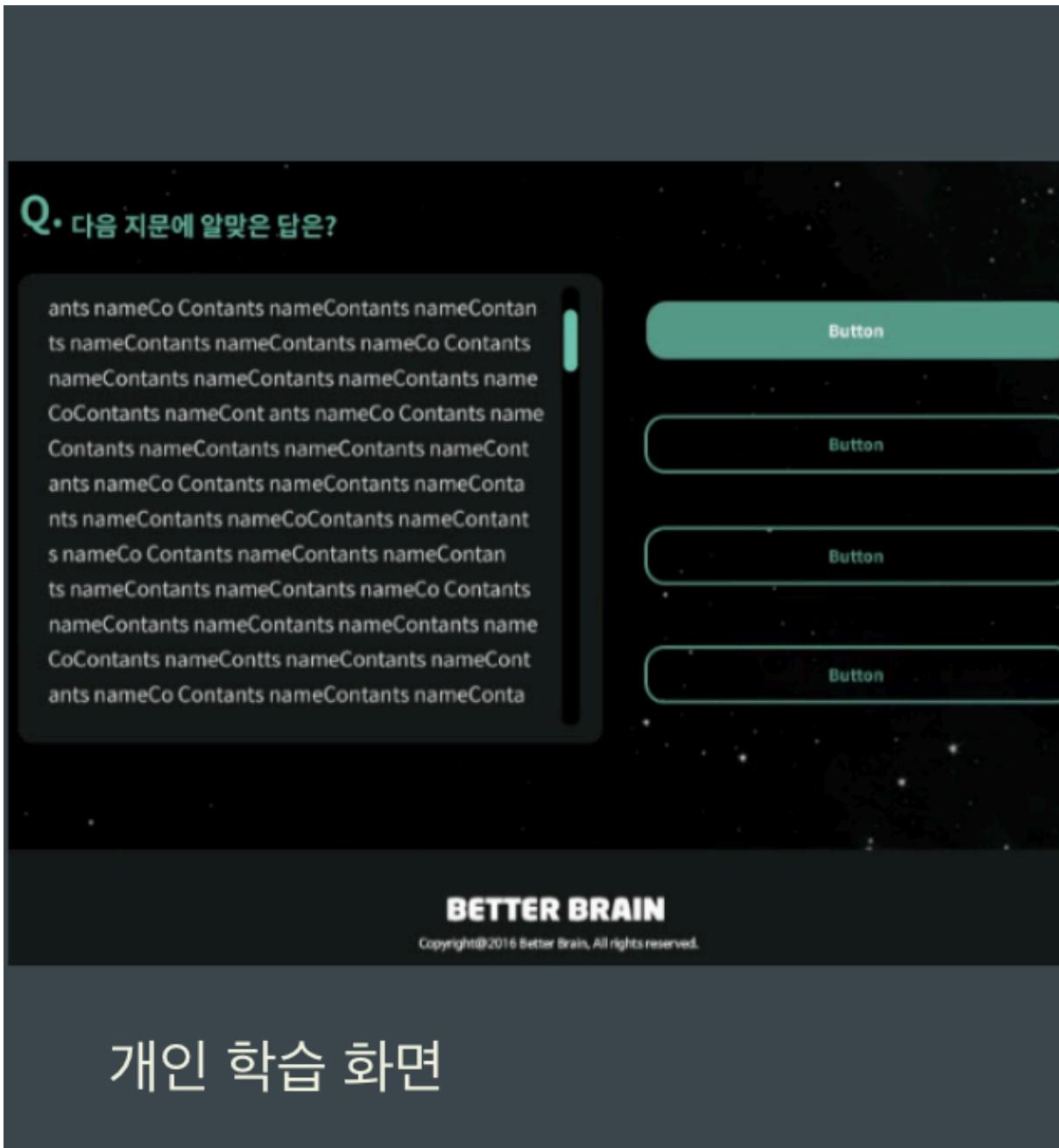
Better Brain

사용 기술

- Git Flow
 - 소스 이력을 더 효율적으로 관리하고자 하용. NHN 컨퍼런스 때 알게되어 시도하게 됨
- PIXI.js
 - 클라이언트 동적 렌더링을 위해 채택. 본 프로젝트 내 배틀 전광판 출력에 사용할 Phaser도 고려했으나 간단하여 채택
- Jquery
 - 당시 서버 개발자들이 unserscore나 lodash보다 Jquery에 더 익숙하여 채택
- SocketIO
 - 실시간 통신을 위해 웹소켓 채택, Battle 모드 전용 처리 서버와 통신
- NodeJS/Express
 - API 서버 개발, Express 프레임워크 이용
- Jasmine.js
 - BDD 기반의 테스트 코드 작성을 위해 이용
- Howler.js
 - 사운드 출력을 위해 채택
- HTML5/CSS3
 - 표준 시멘틱 웹 페이지 구현에 사용
- MongoDB
 - 사용자의 데이터의 용이한 처리를 위해 사용

Better Brain

스크린샷



개인 학습 화면



인증 화면



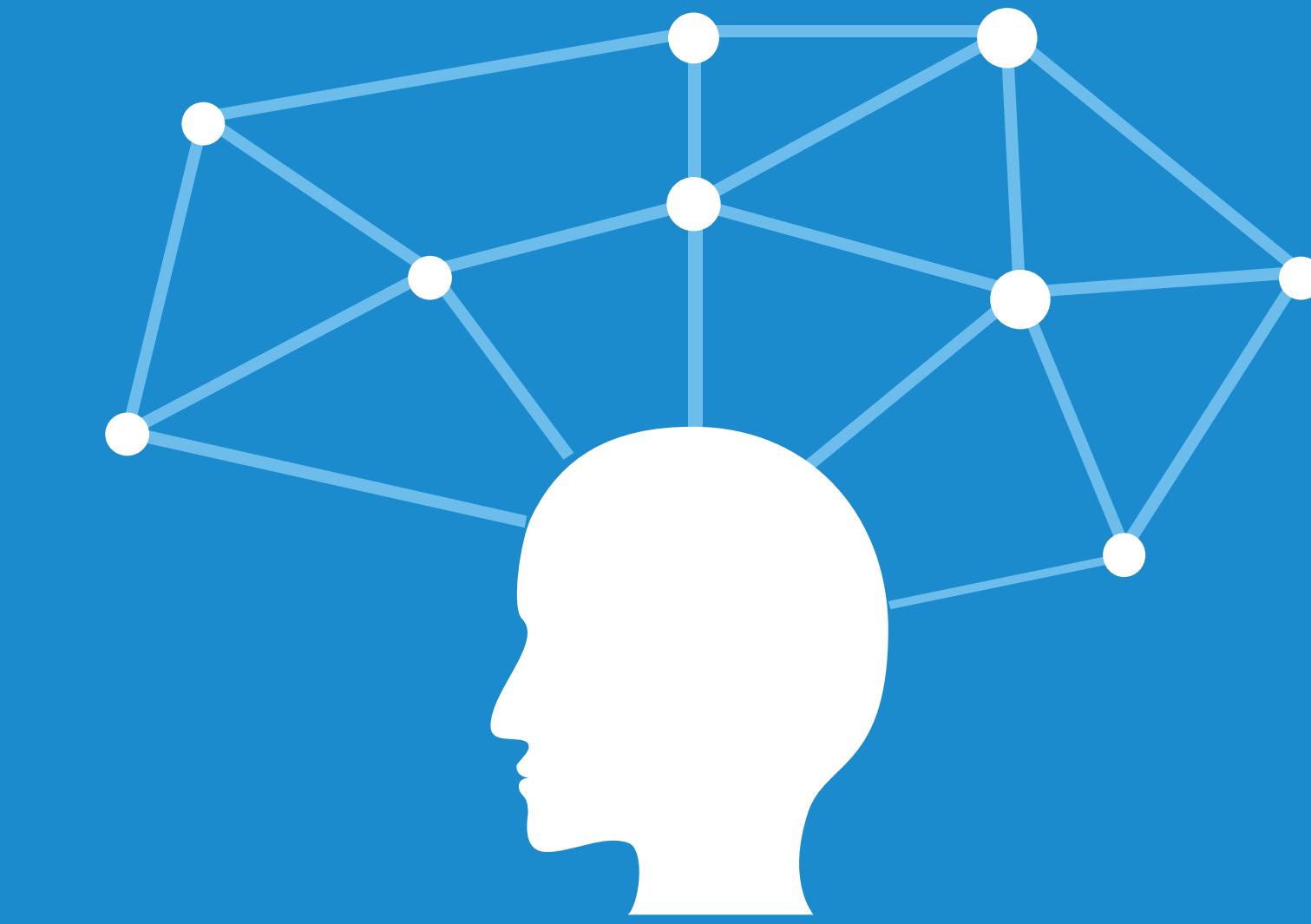
경쟁 풀이 화면

Better Brain

고찰

- 개발 시간이 상당히 짧게 측정되어 기능 우선 개발 방식을 취함
- 외부 디자인 팀 지연으로 일정 차질이 생겨 적합한 대처를 위해 고심
- 비동기 방식의 자바스크립트 개발 방식 적응
- Jasmine.js 테스트 도구로 자바스크립트 테스트를 마련했으나 스크립트 언어 특성상 관리가 어려움
 - 정말 필요한 부분에만 적용하여 개발 시간을 우선 확보
- 개발 팀간 사고하는 결과 프로그램이 달라 이를 맞추기 위해 부단히 노력
 - 기획 내용이 모호한 경우 상당한 혼선을 초래함
- BDD / TDD 미묘한 차이 인지

EP 수집 서비스



EP 수집 서비스

개요

- 제휴 맺은 업체들의 EP 데이터를 수집한 뒤 가공하여 전용 저장소에 적재하는 시스템
- 검색에 노출하기 위한 기본 상품 데이터들을 수집하여 목적에 맞게 분류, 가공하고 걸러서 내부 표준의 데이터로 만든 후 상품 정보를 저장하는 서버
- 각 수집 처리는 비동기 방식을 지향하며, 비즈니스 요구에 맞는 기능의 확장에 열려 있는 시스템
- project-reactor를 적용하여 추상화된 스레드 관리 기능을 이용
- 네트워크 자원 사용량의 효율을 위해 적절한 흐름제어 기능 탑재 등 수집 특화적인 서버
- 지정된 스케줄에 따라 전체 혹은 요약 EP라 칭해지는 파일을 수집

개발 중심 목표

- 특정 시간에 수집 수량에 무관하게 무조건 수집 가능해야 함
- 수집은 최대 1시간 이내로 완료되어야 함
- 데이터 적재 부하를 줄여야 하며 잘못 수집된 경우 정정할 수 있어야 함

EP 수집 서비스

개발 환경

개발

- JDK 8

개발 도구

- IntelliJ

Database

- MySQL(AWS)

이슈 관리 도구

- GitLab

형상 관리 도구

- Git

빌드 시스템

- Gradle

협업 도구

- Telegram

역할 및 이용 기술

역할

- 총원 2명
- Pair programming 을 통해 개발
- 개발 초기부터 최종까지 전 기능 개발

기술 스택

- Spring Boot
- Redis
- Kafka
- React(RX), Flux
- Swagger
- AWS
- pinpoint
- Scouter
- sonar-qube, nexus, code-pipeline

EP 수집 서비스

Git flow

- 소스 브랜치 전략의 기준으로 삼기 위해 사용

Spring Boot

- 스프링이 제공하는 유용한 기능을 사용하기 위해 적용

Redis

- 데이터 캐시 및 운영관리 목적으로 사용

Kafka

- 데이터 부하 분산을 위해 사용(LZ4 압축)

React

- Event-bus 방식의 이벤트 핸들링 처리(비동기)를 위해 사용

Flux

- 스레딩 기법의 추상화 기능을 적용하기 위해 사용

Swagger

- 인페이스 문서화 적용

AWS

- 인프라 정책으로 클라우드 서비스를 이용

Pinpoint

- 서버 모니터링을 위해 사용

Scouter

- 초기 핀포인트에 비동기 이벤트가 잡히지 않아 추가적인 모니터링을 위해 사용

Code-pipeline

- 배포 관리를 위해 사용

Sonar-qube

- 소스 코드의 품질을 검증하기 위해 사용

Nexus

- 배포 담당자의 한계로 모듈 코드 의존성을 위해 사용

EP 수집 서비스

특징

지식이 없는 상황에서 상당히 많은 데이터를 빠르게 적재할 필요성

- 최대 21GB의 데이터를 1시간 이내에 수집하여 적재를 끝내야 하는 제약
- 네트워크 부하를 줄이기 위해 Redis를 적용
- 입력 부하를 분산시키기 위해 Kafka를 적용
- 끊기지 않는 데이터 다운로드를 위한 Http 처리
- 기타 유관 부서가 필요로 하는 데이터를 제공
- 데이터 검증, 수집 필터링 및 조건부 수집 등 부수적 요구사항 적용
- 일관적이지 않은 수집 방식을 추상화하여 통일(format, ftp etc)
- 스케줄, 전체, 요약, 수신기 등으로 서버를 나눠 구성
- 과거 ES 적재 기능 지원
- 외부 DB 접근을 위해 ETL 사용

과제

- 데이터 수집 방식 통일
- 스레드 경합
- 고속 수집에 의한 Redis-Kafka 메모리 오버플로우, 흐름제어
- 레디스 병목 적은 대량 캐시 지원
- 좀비 상품, 삭제
- DB 적재 부하, 네트워크 한계
- 수집 시점 동기화 등

해결

- 데이터 수집 방식 통일(인터페이스 치환; 어댑터 확장)
- 스레드 경합(reactive)
- 고속 수집에 의한 Redis-Kafka 메모리 오버플로우, 흐름제어(GC, Dump 분석, Flow-Buffer)
- 레디스 병목 적은 대량 캐시 지원(lettuce -> partial batch -> reactive)
- 좀비 상품, 삭제(Redis)
- DB 적재 부하, 네트워크 한계(kafka, consumer, scale up)
- 수집 시점 동기화 등(redis synchronization)

아쉬운 점

- 유관 부서의 협조 부족
- 서버 자동 회복 구조 회복
- 큰 회사들의 지원 미비
- 코드 구조가 모듈-서브 모듈이었는데 배포 담당자의 한계로 분리하게 됨

Image Collect Server

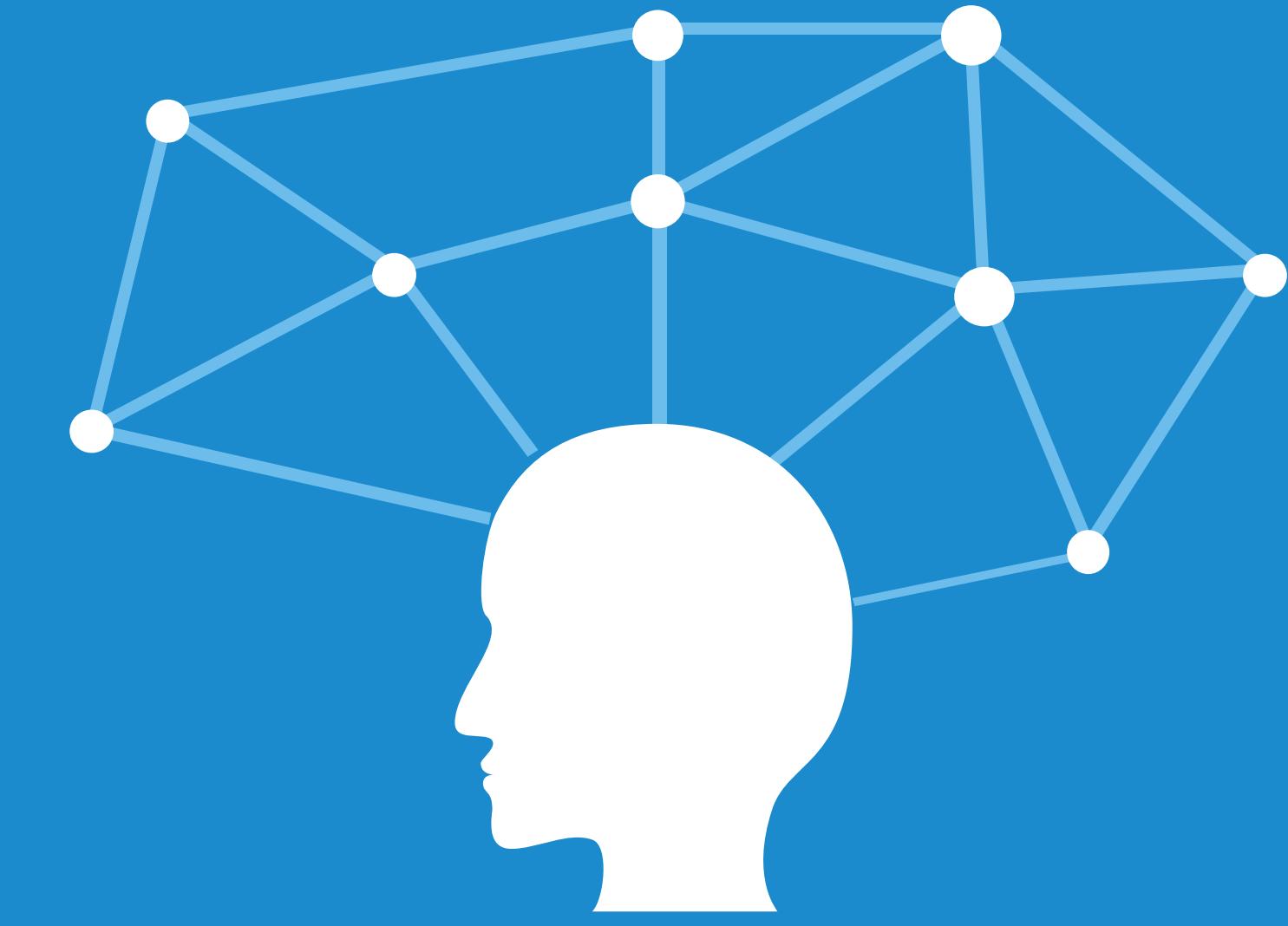


Image Collect Server

개요

- 수집한 상품들의 이미지를 수집하여 가공하여 내부 저장소에 저장하는 서버
- 실시간으로 이미지 데이터를 신속하게 수집하여 가공 및 저장소 업로드를 수행하는 시스템
- 하루 약 3억 건의 이미지 데이터를 수집하여 8억 건의 산출물을 제공해야 함
- 정상적인 이미지 외에도 수집에 실패한 이미지를 수집하며 검증, 갱신 기능이 포함된 시스템

개발 중심 목표

- 실시간으로 최대한 빠르게 이미지를 수집해야 함
- 수집은 한 개 파일에 대해 4가지의 가공물을 제공
- 데이터 수집에 이미지와 상태에 일관성이 깨져서는 안됨
- 가능한 한 필수 타입의 이미지 포맷을 수집함

Image Collect Server

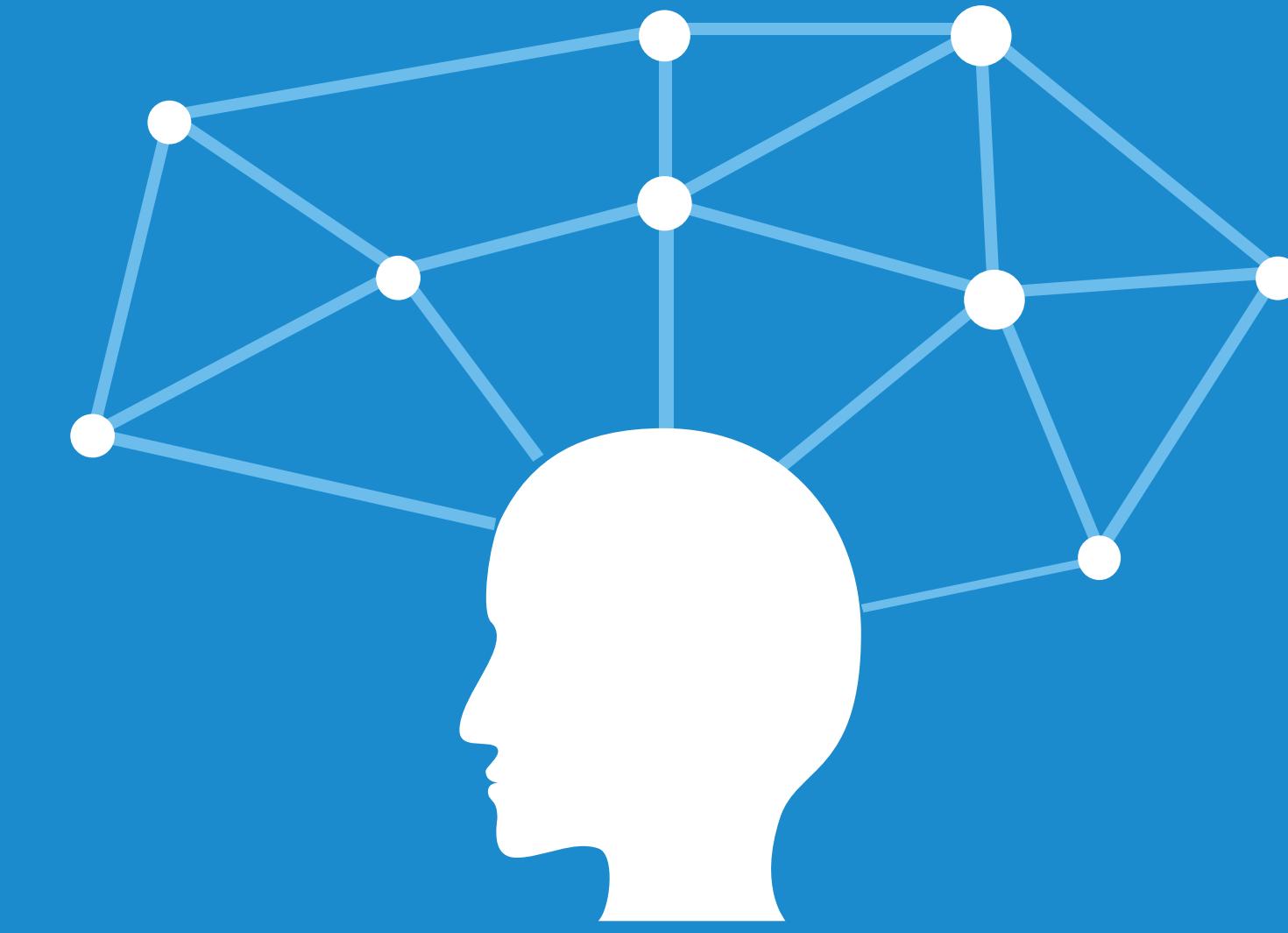
특징

- 동일한 이미지 수집을 지양
- 이미지 캐시 정책으로 이미지 갱신이 이루어지지 않는 경우
- 저장 경로 문제
- 표준을 따르지 않는 Http 응답 처리 - 틀린 정보, 정보 제공 없음, octet-stream
- CMYK 색상 처리 문제
- Animated Gif 처리 문제
- Diff Animated GIF 처리 문제
- Transparent Background PNG 처리 중 배경 알파 제거 현상
- OpenJDK 의 GIF 미지원 문제
- 이미지 포맷 표준이 미지원 되는 문제
- 이미지 수집 대상 선정 문제
- 30X 응답 코드
- 대용량 이미지
- 잘못된 이미지 수집 문제, 쿼리 스트링 문제
- 수집기 수집 범위 분배 문제
- 전체 재수집 이슈

해결

- 동일한 이미지 수집을 지양(mysql update on duplicated -> 검증 서버 생성)
- 이미지 캐시 정책으로 이미지 갱신이 이루어지지 않는 경우(Cdn re-upload -> 삭제 서버 생성)
- 저장 경로 문제(S3 삭제 이슈 -> 새 경로로 이미지 저장 분산)
- 표준을 따르지 않는 Http 응답 처리 - 틀린 정보, 정보 제공 없음 octet-stream(File header 분석, Tika)
- CMYK 색상 처리 문제(CMYK 처리기 개발)
- Animated Gif 처리 문제(GIF frame 절취 코드 개발)
- Transparent Background PNG 처리 중 배경 알파 제거 현상(배경 삭제, 흰색)
- Diff Animated GIF 처리 문제(ICafe Gif reader 적용)
- OpenJDK 의 GIF 미지원 문제(github Gif reader 적용)
- 이미지 포맷 표준이 미지원 되는 문제(github Gif reader 적용)
- 이미지 수집 대상 선정 문제(256 그룹으로 분배)
- 30X 응답 코드(계속 추적)
- 대용량 이미지(타임아웃 처리)
- 잘못된 이미지 수집 문제, 쿼리 스트링 문제(이미지 캐시, 변환으로 수집 중단)
- 수집기 수집 범위 분배 문제(책임 연쇄 패턴, Redis, file, property, default)
- 전체 재수집 이슈(서버 scale out)

Crawling Server



Crawling Server

개요

- EP를 제공하지 못하는 업체들의 상품 페이지를 '크롤', '스크랩'하는 서버
- 실시간으로 이미지 데이터를 신속하게 수집하여 가공 및 저장소 업로드를 수행하는 시스템

개발 중심 목표

- 지정된 솔루션사는 모두 크롤링할 수 있는 기능 지원
- 최대 6시간 이내 크롤 완료
- 크롤링 필터는 자체 검증 가능하며, 코드 수정 없이 변경 가능해야 함
- 크롤 제한 사항에 대한 매너를 지켜야 함

Crawling Server

특징

- 솔루션 사를 사용하는 업체들이 주 크롤링 대상
- 수집 한계는 시간, 상품 수로 결정지어짐
- 독립몰을 크롤링 해야 할 수 있음
- 제휴 업체 서버의 부하가 있어서는 아니 됨
- Crawler4J를 이용해 개발
- 수집 필터는 서버의 재기동 없이 적용 가능해야 함
- 필터는 매일 검증이 가능해야 함

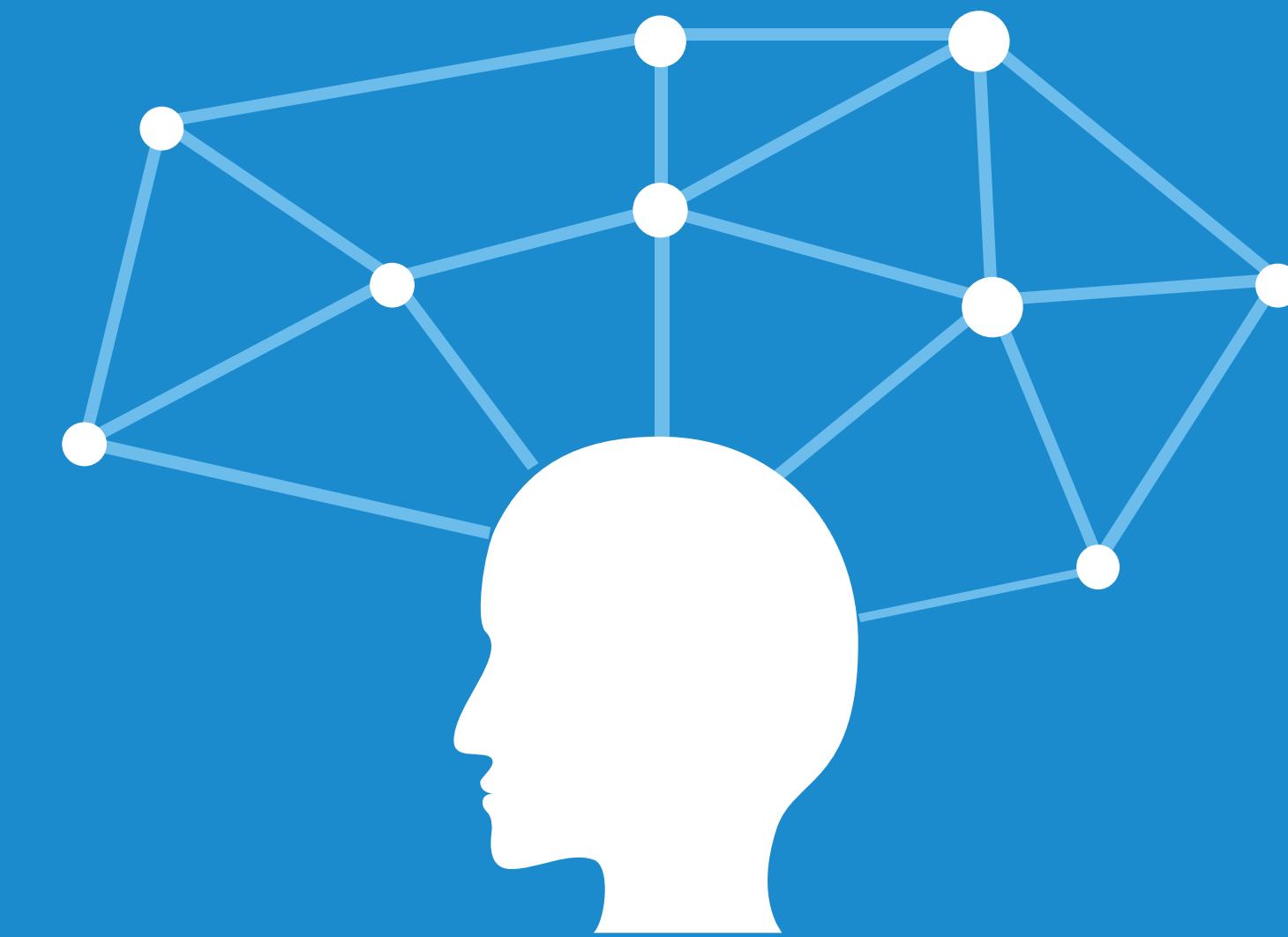
과제

- 크롤링이 동시에 2회 이상 진행되지 않도록 제약
- 크롤 대상 업체는 약 202개
- 크롤 대상이 한계를 넘기는 업체를 크롤해야 하는 일이 발생
- 분산 스크래핑 처리 필요
- 스레드 처리별 대기 시간 효율화 적용 필요
- 분산 수집된 데이터를 중앙 집중적으로 모아야 함

해결

- 크롤링이 동시에 2회 이상 진행되지 않도록 제약(Redis)
- 크롤 대상 업체는 약 202개(결국 노가다)
- 별도 서버가 시한을 제거한 상태에서 크롤
- 분산 스크래핑 처리 필요(Kafka partition)
- 스레드 처리별 대기 시간 효율화 적용 필요(업체 최적화 스크랩 핸들러 적용)
- 분산 수집된 데이터를 중앙 집중적으로 모아야 함(Filebeat, Logstash 적용)

Vertica Project



Vertica Project

개요

- 기존 시스템 MySQL 의 지연 이슈 및 색인 한계 이슈로 RDMBS 교체를 위한 프로젝트
- OLTP 에서 OLAP 기반의 컬럼형 데이터베이스(Vertica)를 이용해 교체
- 기존 시스템과 연동되는 모든 프로젝트의 중단 없는 교체

개발 중심 목표

- 색인, 지연 이슈 해결
- 기존 시스템 및 연동되는 모든 프로젝트의 무중단 교체
- 속도는 동일하거나 개선되어야만 함

Vertica Project

특징

- OLTP 형식에 알맞은 쿼리를 OLAP 에 맞게 모두 수정
- DB 관리부서가 없어 자체적으로 도맡아 처리
- 타부서 쿼리 튜닝 필요
- 3000만건 업데이트에 20분 이내 소요

과제

- 단건 Insert들을 모두 묶어서 batch 처리로 수정
- 타부서 쿼리들을 튜닝하여 이슈가 없어야 함
- Vertica driver가 없는 부서의 경우 API 제공
- 기존 이용하던 쿼리 방식을 변경하기에 로직 또한 알맞게 수정해야 함
- 예상치 못한 수 많은 DB 성능 이슈 및 에러가 발생
- 다양한 사용 사례가 적어 부족한 지식을 보완해야 함
- 색인 성능, 조회 성능이 예상과 달리 매우 나오지 않음

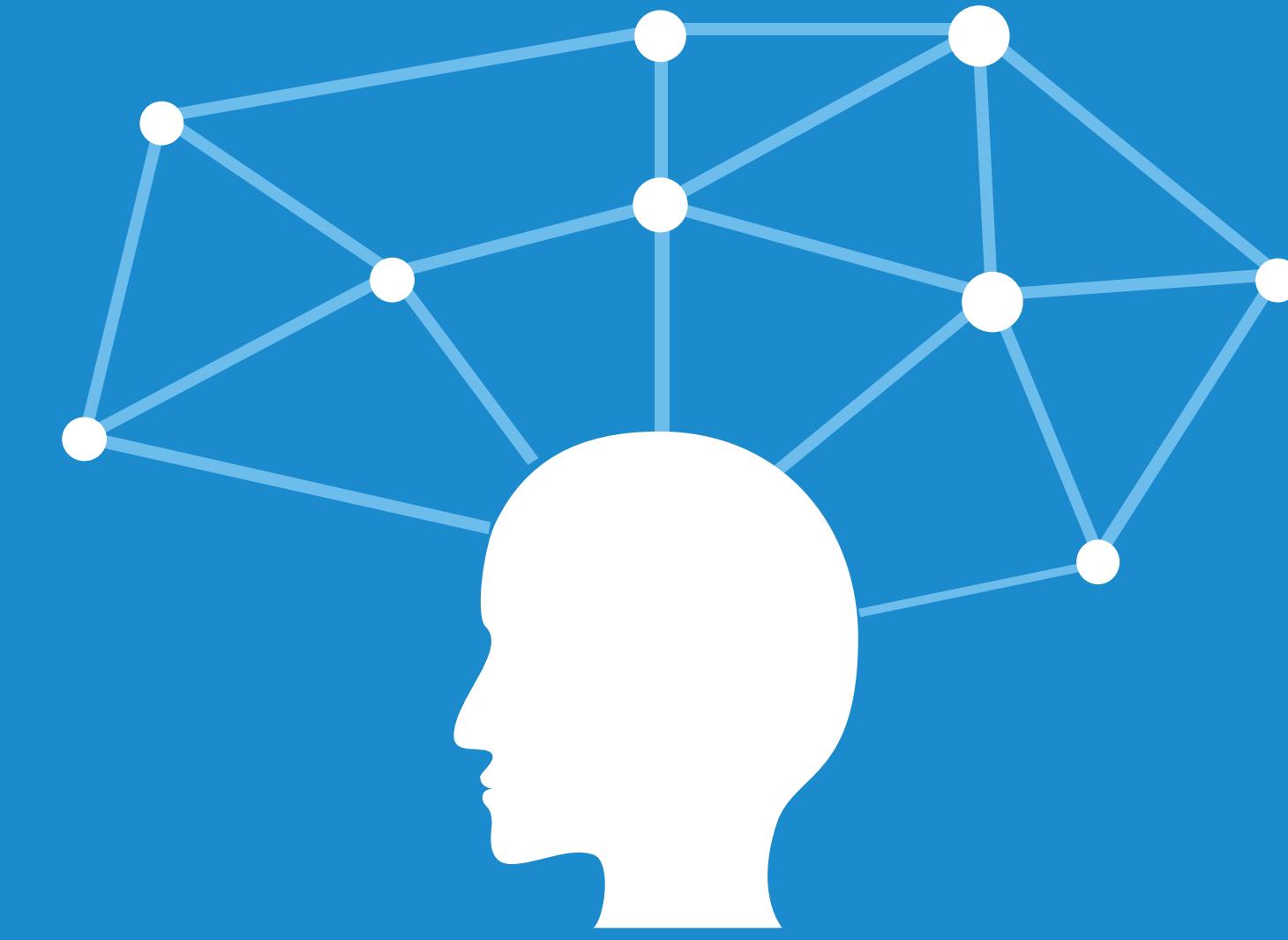
해결

- 쿼리 튜닝 및 개선작업을 통해 기존 속도의 약 90%까지 허용
- Resource pool 및 자원 할당을 계정 별로 나눠 DB의 자원 분할 사용
- Vertica forum 활동
- 버티카 수석 담당자와 지속적인 문의/상담을 통한 개발
- 역정규화를 통한 조회 속도 상향 시도

이슈

- 대량의 적재, 갱신은 문제 없으나 조회에 상당한 부하
- 다른 프로젝트 중 DB에 크게 의존적인 프로젝트의 경우 접근 1회 보류

Vertica Advanced Proj.



Vertica Advanced Project

개요

- 성공적으로 교체한 Vertica Project에서 추가적으로 비용 교체를 위한 서버 아키텍처 수정

개발 중심 목표

- 서버 인프라 비용 최소화(Redis, Kafka 등 제거)
- 교체된 DBMS에 맞는 구조화를 통한 성능 최대화

Vertica Advanced Project

특징

- 모든 테이블 역정규화
- Redis, Kafka 같은 부가적인 통신 흡을 제거
- Vertica의 대량 적재 성능 최대한 활용
- 부하에 따른 서버 증감 개선으로 비용 감소

과제

- Redis 제거
- Kafka 제거
- 역정규화에 따른 쿼리 변경
- SQS 연동
- Autoscale 통한 서버 조율을 진행
- 성능 상향

결과

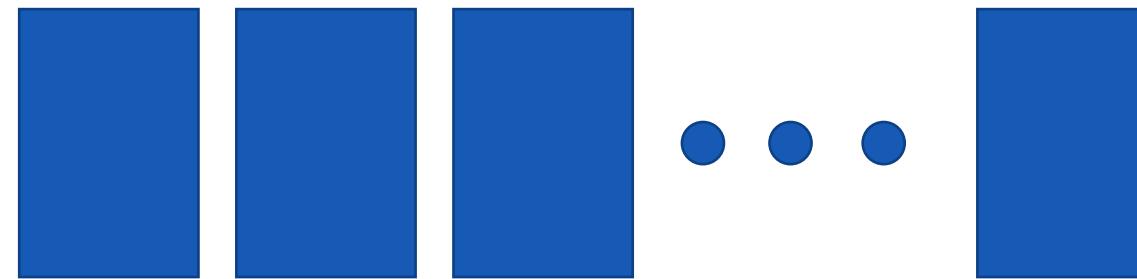
- 이미지 수집 서버 성능 최대 450% 향상
- 서버 비용 축소
 - Redis 제거
 - Kafka 제거
- Reader 테이블 생성을 통한 조회 성능 확보
- 기존 서버 대비 수집 시간 90~110% 유지

이슈

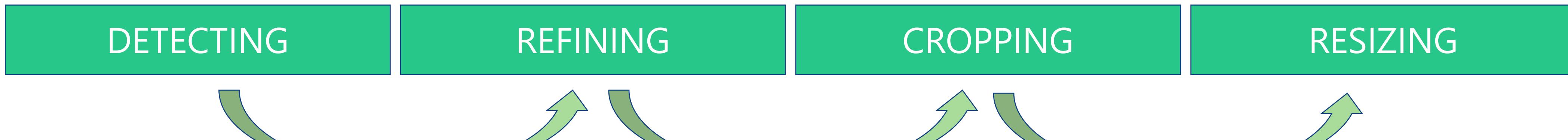
- Vertica는 최적의 해가 아닌 것으로 확인(색인, 조회 성능이 낮음)
- 역정규화 또한 Vertica에서 상당한 DV가 생성되어 DV purge에 많은 시간 소요
- (진행중)

Vertica Advanced Project

이미지 처리 프로세스 변환 중 개선 예



기존은 Stream 기반의 처리



처리 흐름에 1개 스레드, 최대 코어 수 만큼 처리 흐름이 생성되며 Image 인덱스(슬롯)간 경합 방지로 인한 락 제어 코드 존재 등

이후 처리 다중화(스레드 추상)

