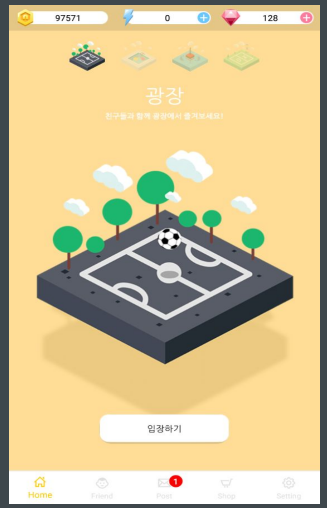
Noriter



개요

모바일 기반의 대규모 광장형 커뮤니티 서비스

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

메시지 처리 중심의 비동기 서버.

개발 중심

- 아이챗 앱의 단점 보완
- 대규모 처리 서비스 개발
- 유저 상호작용 이벤트 개발
- SNG 추구

개발환경

형상관리	Git
개발 언어	Java 8
개발 도구	Spring STS
Database	MySQL, MariaDB(AWS)
이슈 관리 도구	Redmine
협업 도구	Slack
빌드 시스템	Gradle

역할 및 이용기술

총 참여인원 5명
Pair Programming 방식을 취했으며, 설계부터 개발 구현 및 코드 검증까지 모든 과정에 두 개발자가 붙어서 개발함. 즉 전체 개발 과정 참여.
별도 서비스 관리 서버들도 같이 개발.
클라이언트는 외부에서 개발하였을때 slack을 통해 협업 진행, 이후 인원 영입.

기술상세

Git Flow
- 소스 이력을 더 효율적으로 관리하고자 사용. NHN 컨퍼런스 때 알게 된 기능

Spring Boot
- DI와 같은 핵심 기능을 사용하고자 사용

Netty
- 비동기 서버 통신을 위해 채택. 이전 프로젝트에서 사용한 적도 있어 빠른 개발 가능. 일부 소켓을 확장해서 개발

Protobuf
- 클라이언트와의 통신을 위해 사용. 실시간 통신이 이뤄지는 특성 상 JSON은 너무 데이터가 많아 각하하고 byte 단위의 경량 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 구성요소도 같이 관리하기 위해 사용.

React Core
- 비동기 이벤트 주도 방식으로 로직을 처리하기 위해 사용. 상태를 가지지 않으면서도 빠르게 처리하고자 선택.

기술상세

Kafka
- 서버간 메시지 중계기, 로그 데이터의 적재 및 추적에 채택하여 사용. ZeroMQ, RabbitMQ 시도 후, 전자는 문서 및 요구기능 비제공, 후자는 처리량 문제로 박탈.

AWS
- 서버 인스턴스 관리 및 로그 관리로 해당 클라우드 서비스 사용

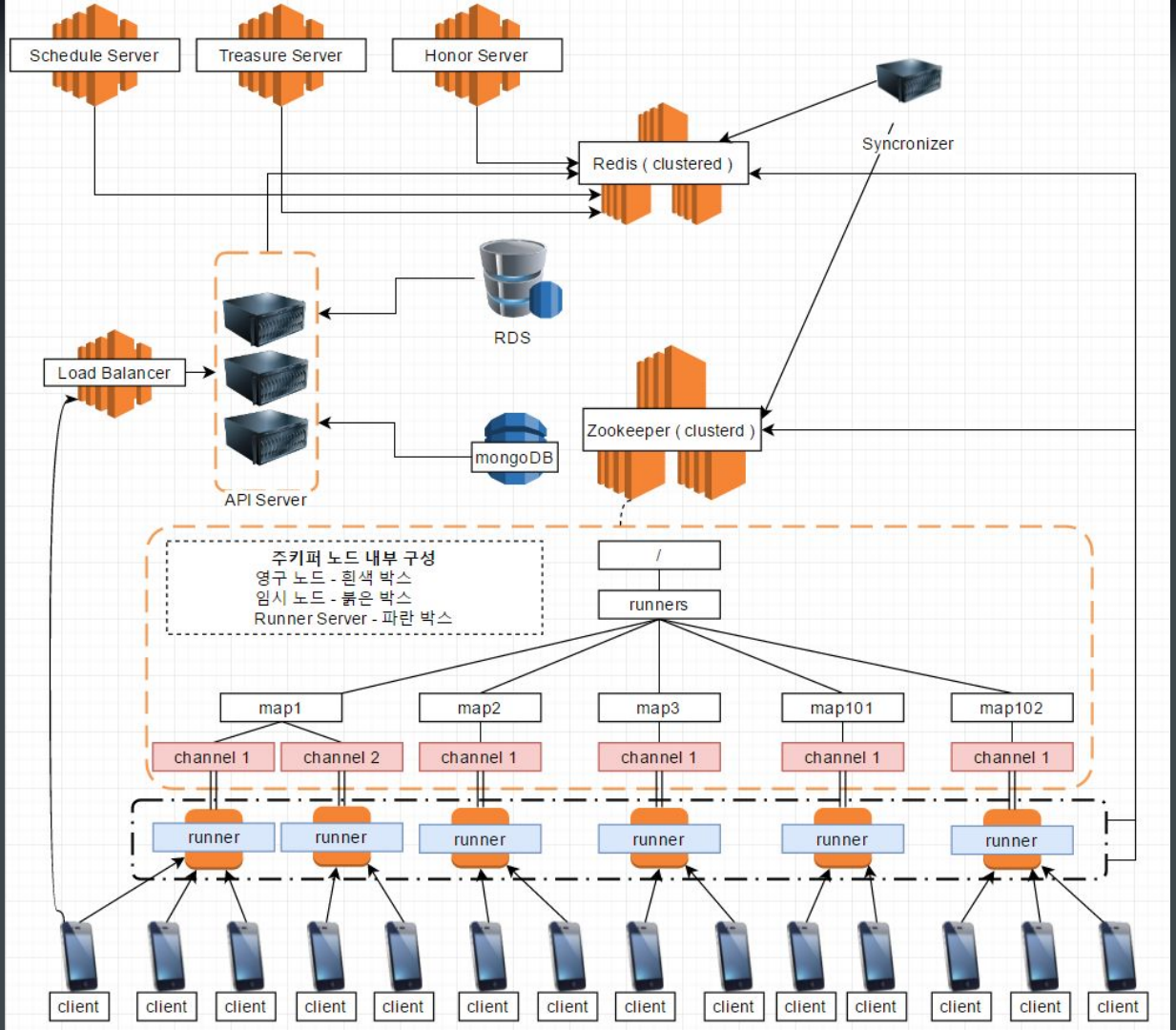
Akka
- 신뢰성 있는 비동기 처리 서버 개발을 위해 적용, 이것을 통해 외부 컨텐츠인 생명체 서버가 설계되었고 이후 서비스 설계에 항시 도입 여부 결정

MongoDB
- 사용자 로그를 저장하기 위해 사용

Admin은 Web으로 개발/Mybatis

아쉬운 점
배포 툴
개발 의견 불일치
서비스 진행
안정화
개발 격차

설계


Schedule Server
Treasure Server
Honor Server
Redis (clustered)
Syncronizer
RDS
Load Balancer
API Server
mongoDB
Zookeeper (clusterd)
주키퍼 노드 내부 구성
영구 노드 - 흰색 박스
임시 노드 - 붉은 박스
Runner Server - 파란 박스
/
runners
map1
map2
map3
map101
map102
channel 1
channel 2
channel 1
channel 1
channel 1
channel 1
runner
runner
runner
runner
runner
runner
client
client
client
client
client
client
client
client
client
client
client
client
client
client
client

설계

Scale In/Out을 고려하여 Stateless하게 개발하는 것이 주된 목표.

각각의 하나의 서버는 위 그림에서 Channel 한 개에 대응됨.
주키퍼를 통해 정보를 제공받고 이에 맞춰 개발 진행함. 서버들이 나눠져 있는데 후회중 -> 수정안 발의 상태

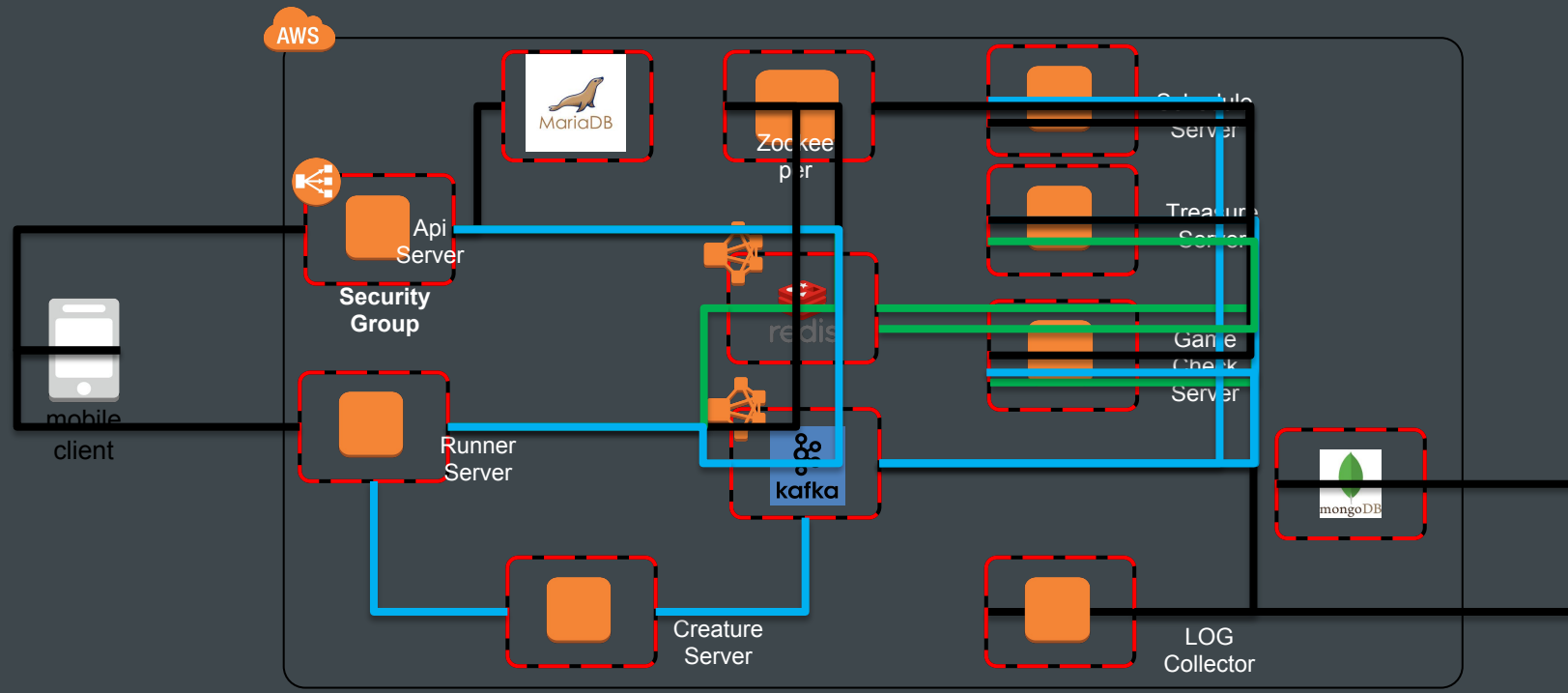
인지 사항

- 비 효율적인 EC2 인스턴스의 사용 -> 비용 증가(실 구동시간 << 대기시간)
- 공통 처리부분에 대한 코드 중복 -> 개발 지연
- 실제 사용자 데이터 저장소 접근 제한에 따른 서버 기능 확장의 비 용이성

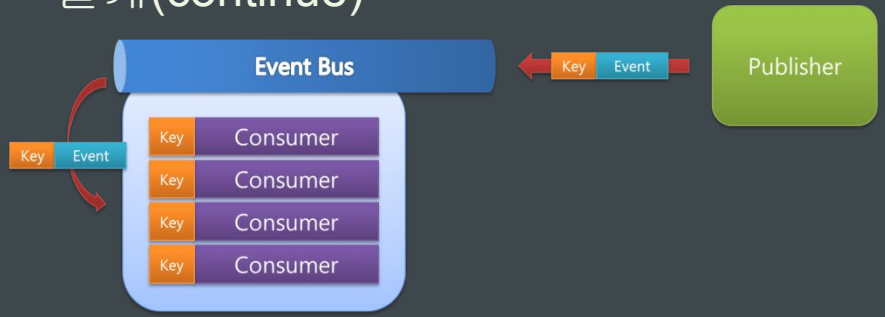
개진 의견

- 불필요한 서버 분할 억제, 컨텐츠 처리 중심의 서버 인스턴스 개발(기존 서버 통합)
- 공통 프로젝트 대상 확장(기존 타 개발자가 코드를 묶어 포함 프로젝트로 함으로써 코드의 중복 제거함)
- 사용자 데이터 저장 위치 변경(마이그레이션) -> 전부터 같이 언급되어 왔으나 상세 방안은 진행되지 못함, 골자는 RDS 내의 사용자 데이터를 Nosql 쪽으로 이전하거나 캐싱을 두어 타 서버들로 하여금 접근을 용이하게 하자는 취지.

설계


AWS
MariaDB
Zookeeper
Api Server
Security Group
redis
mobile client
Runner Server
kafka
Creature Server
LOG Collector
mongoDB

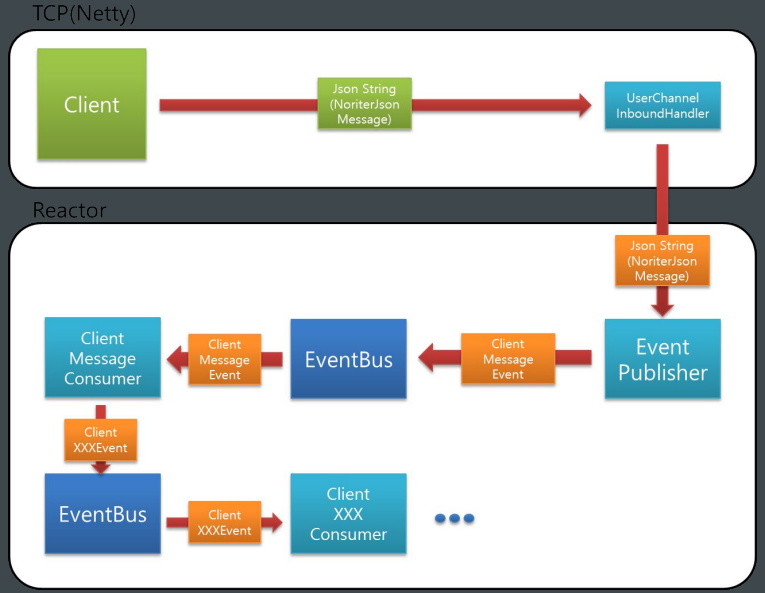
설계(continue)



Reactor를 기반으로 하여 모든 처리를 상태없이 처리하는 방식으로 개발

Event Bus를 생성하여 Consumer들을 등록

Key를 기반으로 등록하며 Publisher에서 송출한 메시지의 Key에 따라 처리 Consumer 지정

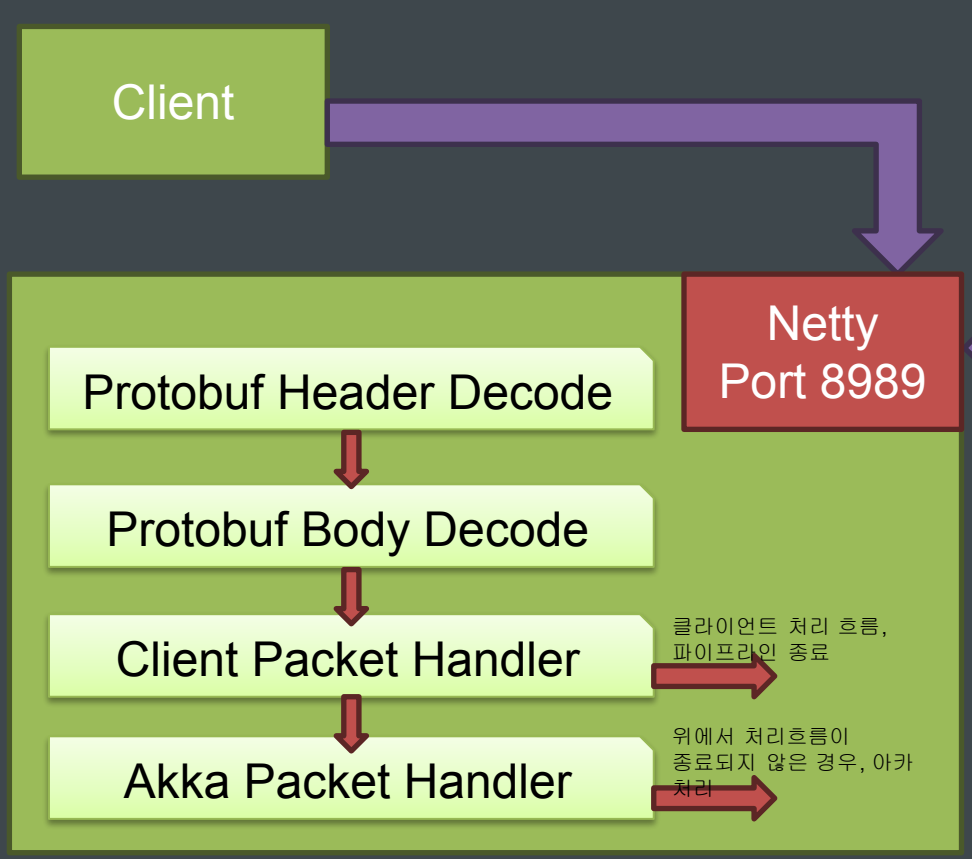


TCP 레벨에서 클라이언트는 JSON으로 통신

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

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

설계(continue)

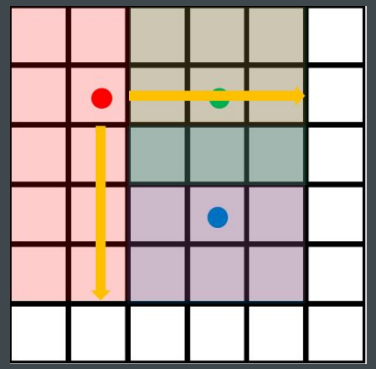


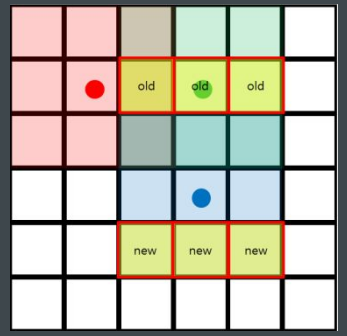
Netty는 같은 포트로 2개의 서버에서 다른 포맷의 메시지를 수신함.

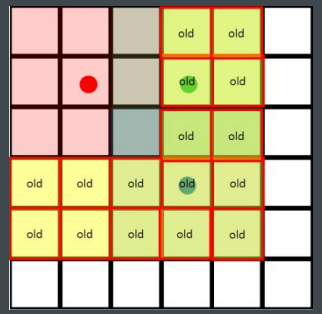
네티 핸들러를 상속하여 기능 확장.

파이프라인 처리 흐름을 분기하는 로직을 확장함으로써 수신부 추가 없이도 기능 확장과 처리의 용이성 확보.

통신(continue)







수백명이 실시간으로 통신을 하며 서로의 정보를 갱신하는 시스템에서 통신량을 최대한 줄이면서도 지연이 되지 않도록 고려

타일 기반으로 사용자가 배치되는 점에 착안.

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

이 청크에 대해서만 통신을 주고받아 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.

설계(continue)

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

Redis를 사용하며 효율적으로 데이터를 저장하고 가져오는 방식 고려,

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

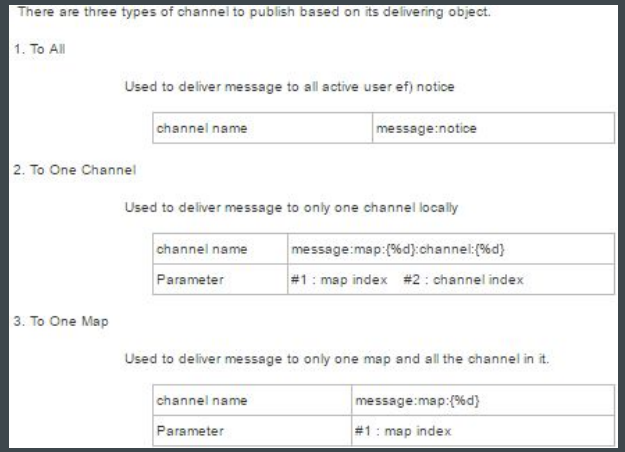
등록	Runner 서버
삭제	Runner 의 퀴즈 결과 처리가 끝난 경우, by GameCheckServer
값 업데이트	Runner의 퀴즈 개시 상태에 따라 변경, by Runner
사용처	Runner, Quiz

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
```

설계(continue)



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

1. To All

Used to deliver message to all active user ef) notice

channel name	message:notice

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

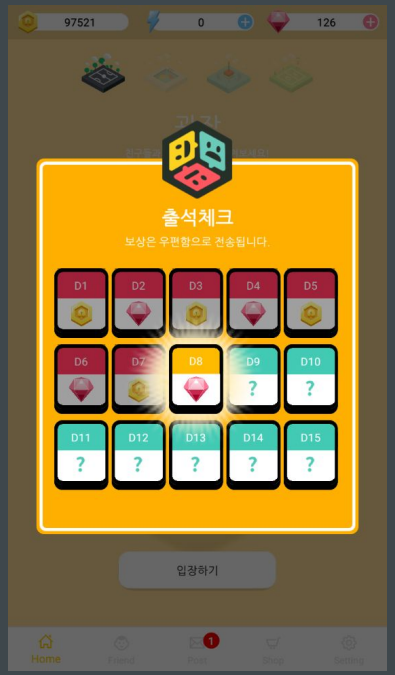
통신 채널의 경우 Runner 사이에서 통신하는 메시지에서만 Redis Pub/sub 사용

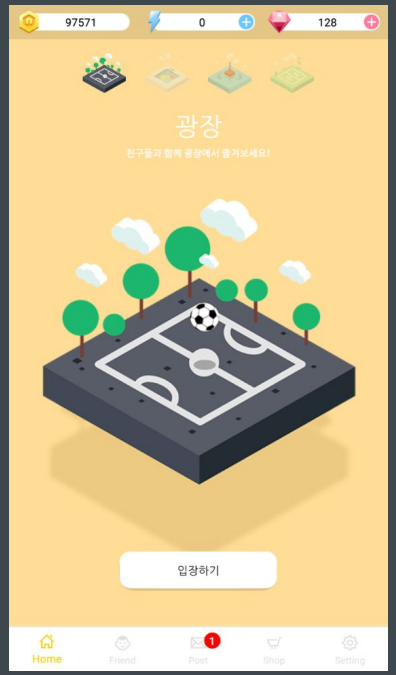
이외는 Kafka의 Pub/Sub을 응용하였음

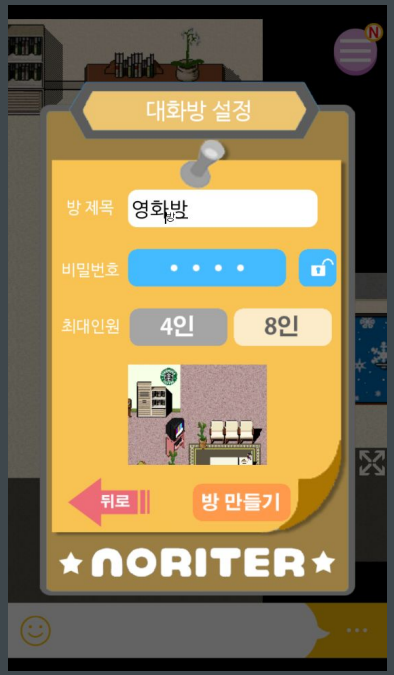
Redis는 String 기반의 데이터를 저장하므로 바이너리 데이터를 전송할 때 별도의 작업이 부가적으로 필요함.

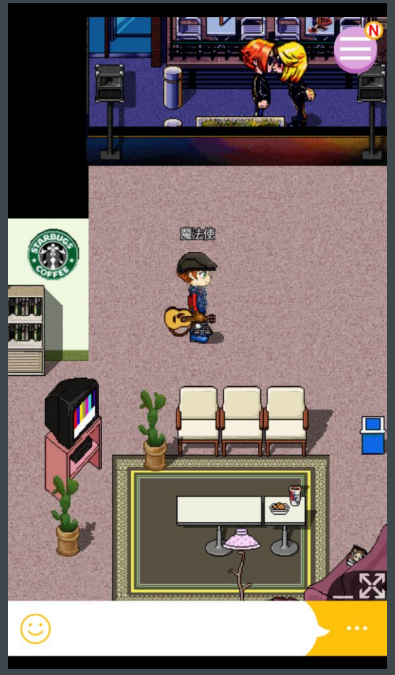
별도로 외부에 Zookeeper 노드를 감시하는 주체를 두고 노드에 문제가 감지되면 이를 파악하여 Redis의 공용 공간에 결함이 없도록 처리함.

스크린샷



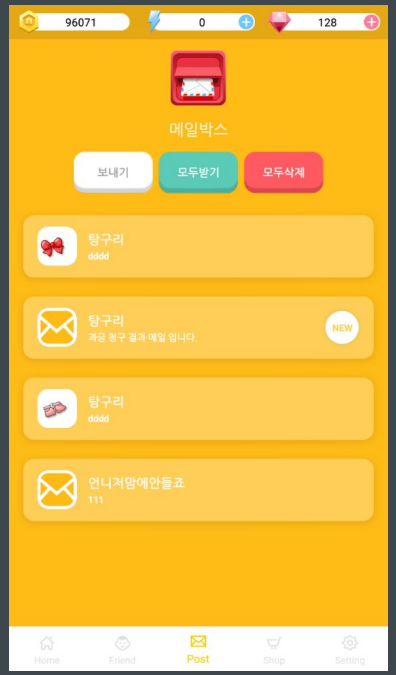


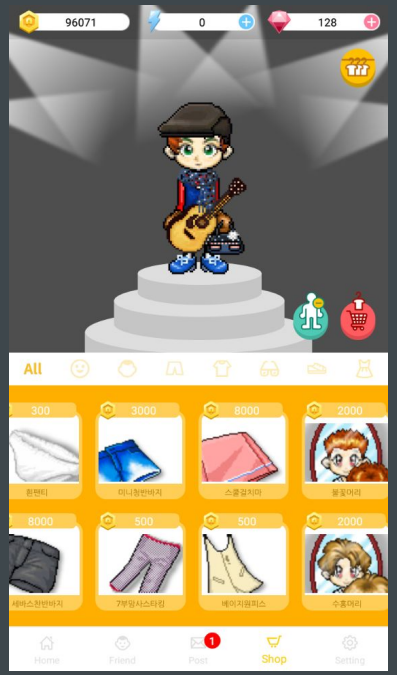


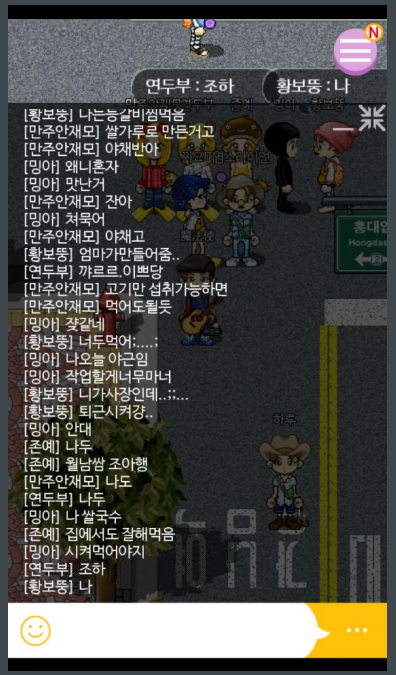


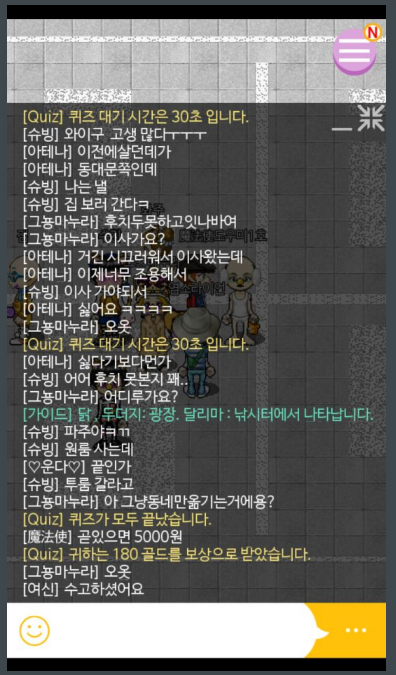
주요 플레이 화면(클라이언트 측)

스크린샷(계속)









주요 플레이 화면(클라이언트 측)

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 이용한 Log 수집(MongoDB)
- DynamoDB를 통한 Mail 저장소 구축
- 관리자 사이트를 통한 서비스 관리 기능
- 기타 Features
- 개발(리팩터링) 현재 진행형(2018. 03. 28 기준)

고찰

Pair 프로그래밍을 통해 개발 부분 부분 전반에 대해 파악할 수 있었으나 개발 지연으로 조절 노력 진행.

모바일 환경에 대비하여 네트워크가 끊기는 경우를 고려, 부자연스럽지 않도록 예외 처리 진행에 노력.

모바일의 네트워크 연결 끊김이 인지되지 않아 TCP keep-alive 와 같은 기본 설정도 변경하며 테스트.

비동기 방식으로 stateless 하게 서버를 구성하는데 미숙하여 시간이 추가로 소모.

클라이언트 개발자와의 통신은 비교적 무난했으나 개발의 지연은 서버 개발에 제동이 걸려 확고한 일정 조율의 필요성 확인.

Redis 및 Zookeeper와 같은 것을 통해 서버 자체만이 아닌 전체 구조가 고려되는 개발을 통해 설계를 보는 시야 확장.

설레발 개발 위험성 인지

오버 아키텍처 신경써야 함 인지

로그 중요성 인지

개발자간 소통 중요성 인지

사용자 소통의 신중함은 진리