200215

### code review

\* typeof (ABC) (x) → typeof ABC

\* 작성 file 끝에 new line 넣어주기 <https://minz.dev/19> → cat 관련 문제

\* docker image에 version 명시하는 것이 좋음.

- docker image는 docker hub로부터 온다.

\* 변수 이름 Id vs ID → Id가 많이 쓰이고 가독성도 좋다

- e.g. socketIdToUser와 같이 Camelcase 중간에 오는 경우.

200216

//TODO: hypermeeting-devops 테스트 (hello world돌아가는지 정도?)

\* pub/sub(redis)에서 subscribe한 객체는 broadcasting을 받기 위해 무한히 blocking된다. → sub을 또 다른 목적으로 사용할 수 없는 이유

\* 내가 무언가(모듈 등)을 도입했을 때 package.json 등에 생소한 모듈이 있다 → npm에서 내가 도입한 무언가와의 의존성을 찾아보자.

<Kubernetes>

pod(파드): 여러 컨테이너가 모인 서비스 or 하나의 컨테이너로 구성되어있음(즉 하나 이상)

cluster(클러스터): 운영하는(배포하는) 하나의 형태

helm(헬름):

1. 클러스터 각 환경에 따라 달라지는 값을 정해두고 이에 따라 배포하는 매커니즘.

2. 쿠버네티스 차트를 관리하기 위한 도구

차트: 매니페스트 템플릿 구성하고 패키지로 관리, 매니페스트 파일 생성

매니페스트: 매니페스트 파일에 기초해 쿠버네티스 리소스 관리

\* 실무에서는 로컬 및 운영 클러스터를 막론하고 여러 환경에 배포해야하기 때문에 애플리케이션은 모두 차트로 패키징해 kubectl 대신 helm으로 배포 및 업데이트.

200217

**배경**

* 언제사용?
* 물리 머신이 가진 메모리보다 더 많은 데이터를 저장해야 할 경우
* Failover를 통해 HA(high availability)를 보장해야 할 경우
* Redis master-slave(using sentinel) vs Redis cluster

**1. Master Slave**



* Master : slave = 1 : n 가능
* Master는 Data 변경시 변경 내용을 backlog에 기록하고 slave는 backlog 를 바탕으로 replication.
* Master가 죽으면 slave가 master에게 주기적으로 connection 요청
* 복구 가능 - Master 살아나고 slave는 replication 수행해 Master와 동기화
* 복구 불가 - Slave중 하나를 master로 승격, 기존의 master를 slave로 강등(**Sentinel** 방식)

**Sentinel**

* 위의 복구 가능 경우 Master의 downtime은 redis cluster의 가용성을 저하(그 동안 write 동작 수행 불가하기 때문), sentinel방식은 이를 해결해줌.
* Redis 관리자 간섭 없이 자동으로 failover → HA
* redis와 별도로 여러 Sentinel process.(fail over를 위해 보통 최소 3개, 홀수 개)
* 홀수 개수로 split brain(additional master)를 방지
* 2개 이상으로 SPOF 방지

**HAProxy**

* master RW(Read/Write), slave RO(Read only) 이기 때문에 client는 각각의 IP, Port를 알고 적절히 붙어서 동작을 해야한다.
* 따라서 master 교체상황에서 client의 redis 설정 또한 변경. → 일일히 계속 바꾸는것은 쉬운 일이 아니기 때문에 **HAProxy** 이용.(HAProxy가 tcp-check로 주기적으로 master, slave 동작 파악)
* HAproxy는 client에게 redis의 master, slave에 일정하게 접근 할 수 있는 end-point를 제공.

**2. Redis Cluster**



* Redis에서 제공하는 replication, 샤딩 기법.
* Sentinel과는 별도의 솔루션.
* 각 redis는 다른 모든 redis들과 직접 연결하여 **gossip protocol**을 통해 통신. → Multi-master, multi-slave.
* client 또한 모든 redis와 직접 연결해 data 주고받음.
* gossip Protocol 기본 port는 16379 → (Redis보다 10000 높은 번호를 사용)
* 각 master는 **Hash Slot**이라는 data 저장구역을 다른 master와 나누어 소유.(위 그림은 hash slot을 3개로 균등 분할해 구성한 모습)
* CRC16을 이용해 16384개의 슬롯 균등 분배됨
* 운영 중단 없이 Hash slot 다른 노드로 이동 가능
* 각 Master에 할당된 hash slot은 redis 관리자에 의해 동적으로 변경 가능
* Master와 Slave 추가삭제 또한 동적으로 가능
* 따라서 위 그림은 1:1이지만, Slave 추가를 통해 Master : slave = 1 : n 가능
* Client는 cluster에 포함된 아무 redis에게 요청을 보냄.
* 처리 가능할 경우 - redis에선 처리 가능한 요청은 처리
* 처리 불가할 경우 - 처리가능한 redis의 정보를 client에게 전달.
* e.g. slave에게 write를 보내면 해당 slave는 처리가능한 master redis 정보를 client에게 전달.
* master 죽을 시 slave는 gossip Protocol을 통해 master의 죽음을 파악한 뒤 스스로 master로 승격. → 이 때 replication은 async이기 때문에 죽음으로 **data** **정합성**이 깨질 수 있다.
* 깨진 정합성으로 인해 data 충돌이 발생할 경우 **무조건** 나중에 master가 된 data 기준으로 정합성을 맞춤.

**References**

<https://redis.io/topics/cluster-tutorial>

<https://www.letmecompile.com/redis-cluster-sentinel-overview/>

<https://goodgid.github.io/Redis-Master-Slave-and-Cluster/>

[https://medium.com/garimoo/redis-documentation-2- HYPERLINK "https://medium.com/garimoo/redis-documentation-2-레디스-클러스터-튜토리얼-911ba145e63"레디스 HYPERLINK "https://medium.com/garimoo/redis-documentation-2-레디스-클러스터-튜토리얼-911ba145e63"- HYPERLINK "https://medium.com/garimoo/redis-documentation-2-레디스-클러스터-튜토리얼-911ba145e63"클러스터 HYPERLINK "https://medium.com/garimoo/redis-documentation-2-레디스-클러스터-튜토리얼-911ba145e63"- HYPERLINK "https://medium.com/garimoo/redis-documentation-2-레디스-클러스터-튜토리얼-911ba145e63"튜토리얼 HYPERLINK "https://medium.com/garimoo/redis-documentation-2-레디스-클러스터-튜토리얼-911ba145e63"-911ba145e63](https://medium.com/garimoo/redis-documentation-2-레디스-클러스터-튜토리얼-911ba145e63)

200218

TODO: proposed rebase done.

~~, kotlin쪽 살펴보기~~

TODO:

1. PO 위에서 간단한거 사용해서 잘 동작하는지 확인

2. WAPL service에서 받은 요청 **RPC**처럼 PO로 띄워 HM server에 전달하기