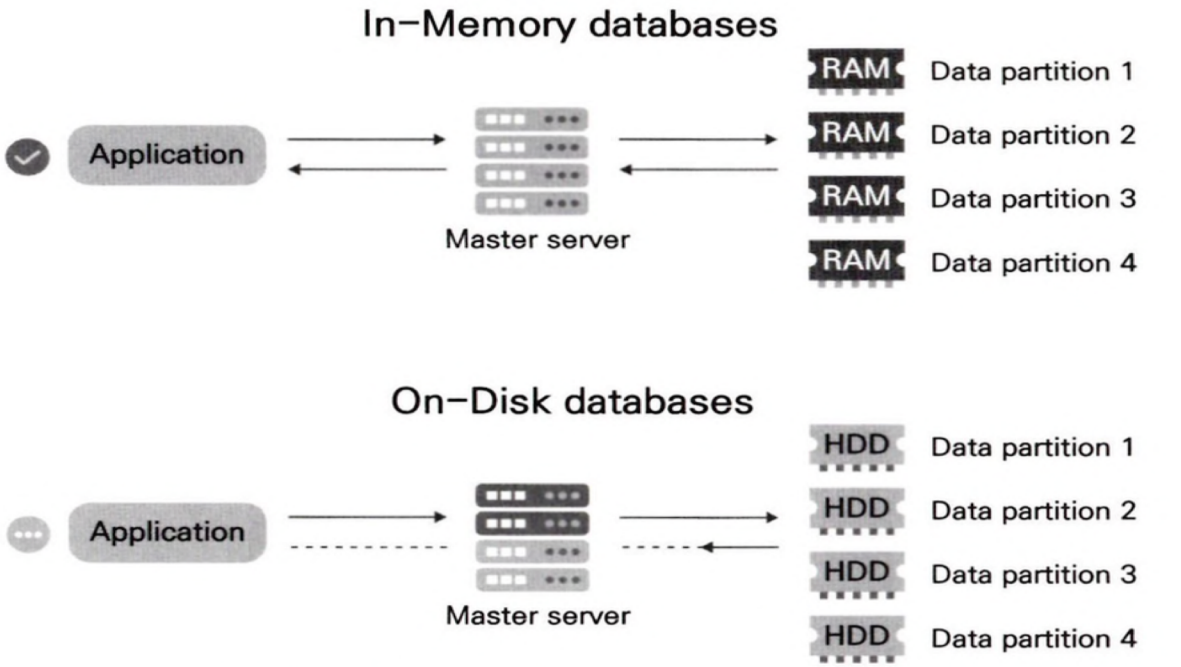
Redis

Redis

- Redis(REmote DIctionary Server)
  - ✓ 특징
    - Redis는 메모리 키 값 데이터 구조 스토어
    - 싱글 스레드로 동작
    - 속도가 매우 빠름



Redis

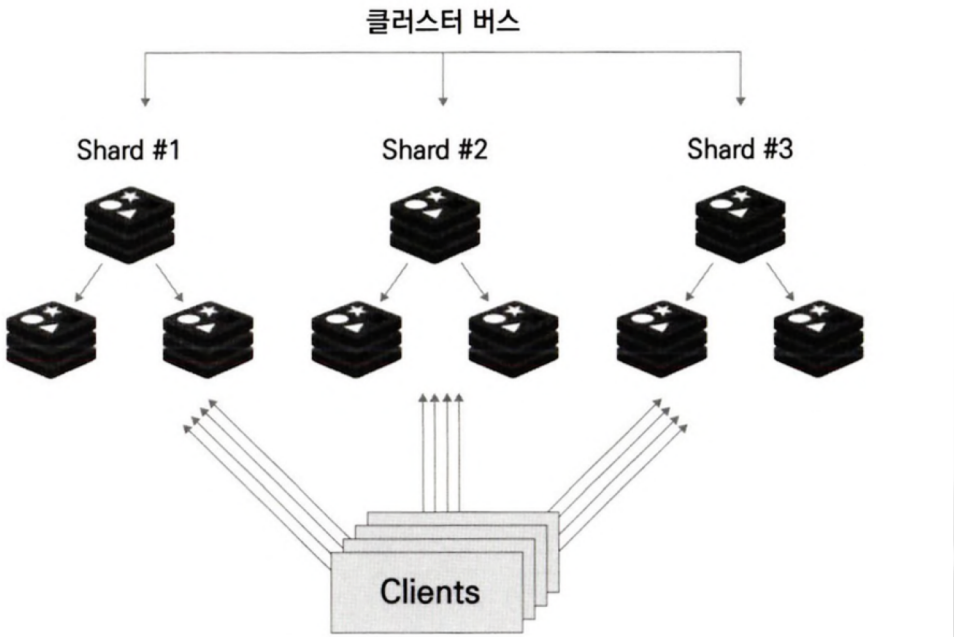
- ❖ Redis(REmote DIctionary Server)
  - ✓ 특징
    - ● 고가용성
      - ◆ 레디스는 자체적으로 HA(High Availability) 기능을 제공
      - ◆ 복제를 통해 데이터를 여러 서버에 분산시킬 수 있으며 sentinel은 장애 상황을 탐지해 자동으로 fail-over(시스템에 장애가 발생했을 때 예비 시스템으로 업무를 즉시 전환하여 서비스 중단 없이 운영을 유지하는 기술)를 수행
      - ◆ 애플리케이션이 센티널을 이용해 레디스에 연결하는 구조에서는 마스터에 장애가 발생하더라도 레디스로의 엔드포인트를 변경할 필요 없이 페일오버가 완료돼 정상화된 마스터 노드를 사용할 수 있음

Redis

- Redis(REmote DIctionary Server)
  - ✓ 특징
    - 확장성
      - 레디스에서 클러스터 모드를 사용한다면 손쉬운 수평적 확장이 가능
      - 데이터는 레디스 클러스터 내에서 자동으로 샤딩된 후 저장되며 여러 개의 복제본이 생성될 수 있음
      - 데이터의 분리는 데이터베이스 레이어에서 처리되며 애플리케이션에서는 대상 데이터가 어떤 샤드에 있는지 신경 쓰지 않아도 되므로 일반 레디스를 사용할 때와 동일하게 데이터를 가져오고 저장할 수 있음
      - 클러스터 구조에서 모든 레디스 인스턴스는 클러스터 버스라는 프로토콜을 이용해 서로 감시하고 있으며 이를 이용해 클러스터의 마스터 노드에 문제가 발생하면 자동으로 페일오버를 시켜 고가용성을 유지

Redis

- ❖ Redis(REmote DIctionary Server)
  - ✓ 특징
    - ● 확장성



Redis

- ❖ Redis(REmote DIctionary Server)
  - ✓ 특징
    - ● 주요 클라우드 벤더들이 상품화해서 서비스 제공
      - ◆ AWS는 Amazon ElastiCache for Redis를 Google Cloud는 Cloud Memory store for Redis를 Microsoft Azure는 Azure Cache for Redis를 제공하며 국내에서는 NHN CLOUD가 EasyCache를 NAVER CLOUD가 Cloud DB for Redis를 제공
      - ◆ 위의 제품을 통해 대규모 서비스를 구축하면 높은 가용성 그리고 강력한 성능을 달성할 수 있으며 이를 통해 사용자에게 더욱 빠르고 안정적인 서비스를 제공할 수 있음

Redis

- ❖ Redis(REmote DIctionary Server)
  - ✓ Micro Service Architecture 와 Redis
    - ● 데이터 저장소로서의 레디스
      - ◆ 레디스는 마이크로서비스 아키텍처에서 각 서비스별 개별 저장소로 사용하기에 적합
      - ◆ 설치가 간편하고 최소한의 리소스로 막대한 처리량을 낼 수 있으며 다양한 자료 구조를 제공하면서도 사용이 간단하기 때문에 마이크로서비스의 요구 사항에 맞는 데이터를 저장하기에도 편리
      - ◆ 고가용성을 위해 로드 밸런서나 프록시 등 추가적인 서비스를 설치할 필요가 없어 하나의 튼튼한 저장소로서의 역할을 수행하기에 적합
      - ◆ 메모리에 있는 데이터가 영구 저장되지 않기 때문에 데이터 저장소로 사용하기 위해 레디스의 도입을 고민할 때 데이터의 영속성을 고민할 수 있으나 레디스의 데이터는 AQF(Append Only File) 와 RDB(Redis DataBase) 형식으로 디스크에 주기적으로 저장할 수 있으며 레디스에 장애가 발생해 데이터가 유실되더라도 백업 파일을 이용하면 다시 복구할 수 있음

Redis

- ❖ Redis(REmote DIctionary Server)
  - ✓ Micro Service Architecture 와 Redis
    - ● 메시지 브로커로서의 레디스
      - ◆ 마이크로서비스 아키텍처에서 각 서비스는 완전히 분리돼 있는 구조로 동작하기 때문에 서로 다른 서비스 간에 지속적인 통신이 필요
      - ◆ 메시징 큐 혹은 stream과 같은 메시지 브로커를 이용해 서비스들 간에 비동기적으로 데이터를 전달할 수 있는 통신 채널을 구현하는 것이 좋음
      - ◆ 레디스는 NoSQL 데이터 저장소로 알려져 있기 때문에 단순하게 데이터를 저장할 수 있는 저장소로만 생각하기 쉽지만 사실 레디스는 서비스 간 메시지를 전달할 때 매우 유용하게 사용할 수 있음
      - ◆ 레디스의 pub/sub 기능은 가장 간단한 메시징 기능으로 굉장히 빠르게 동작하며 간단하게 사용할 수 있음
      - ◆ 1개의 채널에 데이터를 던지면 이 채널을 듣고 있는 모든 소비자는 데이터를 빠르게 가져갈 수 있음
      - ◆ pub/sub에서 모든 데이터는 전달된 뒤 삭제되는 일회성으로 모든 메시징 상황에 적합하진 않지만 fire-and-forget 패턴이 필요한 간단한 알림 서비스에서는 유용하게 사용할 수 있음

Redis

- ❖ Redis(REmote DIctionary Server)
  - ✓ Micro Service Architecture 와 Redis
    - ● 메시지 브로커로서의 레디스
      - ◆ 레디스의 list 자료 구조는 메시징 큐로 사용하기 알맞은데 list에서 데이터는 빠르게 push/pop을 할 수 있으며 애플리케이션은 list에 데이터가 있는지 매번 확인할 필요없이 대기하다가 list에 새로운 데이터가 들어오면 읽어 갈 수 있는 블로킹 기능울 사용할 수도 있음
      - ◆ 레디스의 stream 자료 구조를 이용하면 레디스를 완벽한 스트림 플랫폼으로 사용할 수 있음
      - ◆ stream은 아파치 카프카 시스템에서 영감을 받아 만들어진 자료 구조로 데이터는 계속해서 추가되는 방식으로 저장(append-only)
      - ◆ 카프카처럼 저장되는 데이터를 읽을 수 있는 소비자와 소비자 그룹이 존재해 데이터의 분산 처리도 가능하며 저장된 데이터를 시간대별로 검색하는 것도 가능

Redis

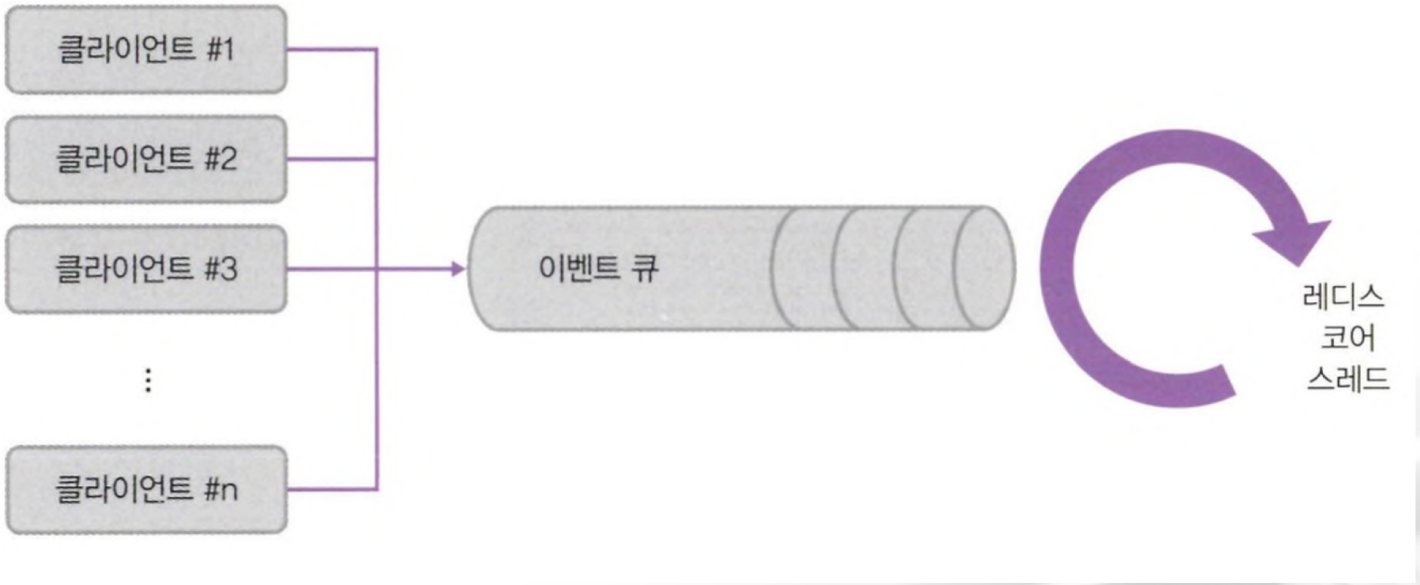
- Redis(REmote DIctionary Server)
  - ✓ 반 영구적 저장
    - RDB(Redis DataBase)
      - 메모리에 있는 데이터 전체에서 스냅샷을 작성하고 이를 디스크로 저장하는 방식을 사용하는데 스냅샷을 생성하는 기능이므로 데이터를 백업하거나 복원하기가 매우 간단해서 특정 시간마다 여러 개의 스냅샷을 생성하고 데이터를 복원해야 한다면 특정 시간의 스냅샷 파일 하나만 그대로 로딩하면 되기 때문
      - 스냅샷 이후에 변경된 데이터는 복구할 수 없다는 단점이 있는데 복구할 수 없는 데이터는 그대로 유실 상태가 됨
    - AOF(Append Only File)
      - Redis에 데이터가 변경되는 이벤트가 발생하면 이를 모두 저장하는 방식으로 데이터를 생성, 수정, 삭제하는 이벤트들을 초 단위로 취합하여 로그 파일에 작성하고 특정 데이터를 열 번 수정하면 수정 명령어 열 번이 AOF 파일에 모두 기록되는데 모든 데이터의 변경 기록들을 보관하고 있으므로 최신의 데이터 정보를 백업할 수 있기 때문에 서버를 복구할 때 RDB 방식에 비해 데이터 유실량이 적은 장점이 있으며 최악의 경우 초 단위 데이터들만 유실
      - AOF 방식의 단점은 로딩 속도와 파일 크기로 데이터를 복구할 때 로그에 쌓인 변경 사항들을 다시 실행하므로 하나의 데이터에서 같은 명령어가 여러 번 실행될 수 있는데 스냅샷을 한 번에 로딩하는 RDB 방식보다 느리며 같은 맥락으로 저장 기간에 비례하여 이벤트 로그가 누적되기 때문에 RDB 방식과 비교했을 때 로그 크기는 커짐

Redis

- Redis(REmote DIctionary Server)
  - ✓ 반 영구적 저장
    - Redis는 일반적으로 AOF와 RDB를 동시에 사용하여 데이터를 백업하는데 매일 OO 시마다 RDB를 생성하고 RDB 생성 이후에 변경되는 데이터는 AOF로 백업
    - RDB 데이터는 세 개의 파일을 유지하고 AOF 파일은 매일마다 다른 파일 이름으로 저장하는데 역시 3일치 데이터를 유지하는 형태로 사용할 수 있음

Redis

- Redis(REmote DIctionary Server)
  - ✓ 이벤트 루프



Redis

- Redis(REmote DIctionary Server)
  - ✓ 이벤트 루프
    - Redis는 사용자들이 실행한 명령어들을 이벤트 루프(event loop) 방식으로 처리
    - 이벤트 루프 방식은 클라이언트가 실행한 명령어들을 이벤트 큐에 적재하고 싱글 스레드로 하나씩 처리하는 방식
    - 메모리라는 속도가 매우 빠른 저장 매체를 사용하기 때문에 싱글 스레드로 데이터를 빠르게 처리할 수 있고 멀티 스레드 환경에서 발생할 수 있는 문맥 교환(context switching)이 없으므로 효율적으로 시스템 리소스를 사용할 수 있는 장점이 있음
    - 멀티 스레드 환경에서 공유 자원에 동시에 여러 스레드가 접근한다면 dead lock이 발생할 수 있지만 Redis의 코어 스레드는 싱글 스레드로 구성되어 있어 Dead Lock 같은 현상이 발생하지 않음
    - 싱글 스레드 특성을 이용하여 분산 락으로 Redis를 사용할 때도 있는데 MSA 환경의 여러 컴포넌트는 공유 자원을 사용할 수 있으므로 경쟁 상태(race condition)가 될 수 있는데 이때 Redis를 사용하면 공유 자원의 점유 여부를 저장할 수 있음
    - 싱글 스레드 코어의 단점은 Redis 명령어 중 전체 데이터를 스캔하는 명령어들이 있는데 이런 명령어의 처리 시간은 데이터 크기에 비례하므로 실행 시간이 길어질 수 있는데 싱글 스레드이므로 이 명령어를 처리하는 동안 다른 명령어를 처리할 수 없기 때문에 다른 명령어들이 이벤트 큐 에 저장되어 기다리는 시간이 길어지는 만큼 응답 속도가 떨어지는 문제점이 발생하기도 함

Redis

- Redis(REmote DIctionary Server)
  - ✓ 이벤트 루프
    - Redis 서버를 구축할 때 단독 서버로 아키텍처를 구성하면 장애에 적절히 대응할 수 없어서 Production 환경에서는 기본적으로 한 대의 Master(master)와 하나 이상의 Replica(replica) 서버를 한 세트로 구성
    - 일반적으로 쓰기, 수정. 삭제 같은 작업은 Master에서 실행되고 Master 노드 데이터를 Replica 서버들에 복제해서 데이터를 동기화하는 형태로 구성하는데 이런 작업을 리플리케이션(replication)이라고 하며 한 세트에 포함된 Redis 서버들은 모두 같은 데이터를 저장
    - Master 서버에 장애가 발생하면 Replica 서버 중 한 대가 Master 역할을 대신하기 때문에 서비스 도중 Replica가 Master 역할을 맡아도 데이터 유실없이 서비스할 수 있어서 시스템의 고가용성을 유지할 수 있음
    - 데이터가 중요하여 유실되면 안 되는 상황이라면 두 대 이상 의 Replica 서버를 하나의 세트에 추가할 수 있으며 Replica 서버의 숫자는 데이터 중요도에 따라 결정
    - 캐시 데이터처럼 유실되어도 상관없다면 Replica 서버를 한 대만 유지해도 되는데 유실된 캐시 데이터는 다시 캐시하면 그만이지만 Redis를 주 데이터 저장소로 사용하는 경우에는 두 대 이상 설정하는 것도 고려

Redis

- Redis(REmote DIctionary Server)
  - ✓ 이벤트 루프
    - 클라이언트들은 Master 서버에만 데이터를 생성하고 수정, 삭제하는 작업을 수행하고 Master에 변경이 발생하면 변경된 데이터들을 Replica 서버에 복제
    - Replica 서버에 데이터를 수정해도 Master 서버에는 영향이 없음
    - Redis를 사용하는 클라이언트 입장에서는 여러 서버 중 어떤 서버가 Master인지 알아야 하는데 Redis는 서버 역할을 모니터링하고 상태를 관리하는 솔루션을 제공하는데 Redis Sentinel 과 Redis Cluster가 대표적이며 Master 서버를 항상 모니터링하고 있으며 Master 서버에 장애가 발생하면 다른 Replica 서버를 Master 서버로 선출하는 기능을 제공

Redis

- Redis(REmote DIctionary Server)
  - ✓ 장점
    - 빠른 성능
      - 데이터를 디스크 또는 SSD에 저장하는 대부분의 데이터베이스 관리 시스템과는 달리 모든 Redis 데이터는 서버의 주 메모리에 상주
      - Redis와 같은 인 메모리 데이터베이스는 디스크에 액세스해야 할 필요를 없앰으로써 검색 시간으로 인한 지연을 방지하고 CPU 명령을 적게 사용하는 좀 더 간단한 알고리즘으로 데이터에 액세스할 수 있음
      - 일반적으로 작업을 실행하는 데 1밀리초 미만 소요
    - 인 메모리 데이터 구조
      - Redis를 사용하면 사용자가 다양한 데이터 유형에 매핑되는 키를 저장할 수 있음
      - 기본적인 데이터 유형은 String 으로서 텍스트 또는 이진 데이터가 이에 해당하며 최대 크기는 512MB
      - Redis는 문자열이 추가된 순서대로 유지되는 Lists of Strings, Sets of unordered Strings, 점수에 따라 정렬되는 Sorted Sets, 필드와 값 목록을 저장하는 Hashes, 데이터 세트에서 고유한 항목을 세는 HyperLogLog를 지원
      - 거의 모든 유형의 데이터가 Redis를 사용하여 인 메모리에 저장될 수 있음

Redis

- ❖ Redis(REmote DIctionary Server)
  - ✓ 장점
    - ● 다양성과 사용 편의성
      - ◆ Redis는 개발과 운영을 좀 더 쉽고 좀 더 빠르게 수행할 수 있는 여러 가지 도구를 제공
      - ◆ Pub/Sub는 메시지를 채널에 게시하며 채널에서 구독자에게 전달
      - ◆ 채팅과 메시징 시스템에 매우 적합
      - ◆ TTL 키는 해당 기간 후에는 스스로를 삭제하는 지정된 Time To Live 값을 가질 수 있음
      - ◆ 데이터베이스를 불필요한 데이터로 채우지 않도록 하는 데 유용
      - ◆ 원자성 카운터는 경합 상태가 일관성 없는 결과를 생성하지 않도록 함
    - ● 선호하는 개발 언어 지원
      - ◆ Redis 개발자는 백 개가 넘는 오픈 소스 클라이언트를 사용할 수 있으며 Java, Python, PHP, C, C++, C#, JavaScript, Node.js, Ruby, R, Go를 비롯한 다수의 언어를 지원

Redis

- ❖ Redis(REmote DIctionary Server)
  - ✓ 장점
    - ● 복제 및 지속성
      - ◆ Redis는 Master-Slave 아키텍처를 사용하며 비동기식 복제를 지원하여 데이터가 여러 Slave 서버에 복제될 수 있는데 이렇게 하면 주 서버에 장애가 발생하는 경우 요청이 여러 서버로 분산될 수 있으므로 향상된 읽기 성능과 복구 기능을 모두 제공할 수 있음
      - ◆ Redis는 안정성을 제공하기 위해 특정 시점 스냅샷(Redis 데이터 세트를 디스크로 복사)과 데이터가 변경될 때마다 이를 디스크에 저장하는 Append Only File(AOF) 생성을 모두 지원
      - ◆ 장애 발생 시 Redis 데이터를 신속하게 복원할 수 있음

Redis

- Redis(REmote DIctionary Server)
  - ✓ 사용 사례
    - 주 데이터 저장소: AOF. RDB 백업 기능과 Redis 아키텍처를 사용하여 주 저장소로 데이터를 저장할 수 있지만 메모리 특성상 용량이 큰 데이터 저장소로는 적절하지 않음
    - Caching
      - 다른 데이터베이스 앞에 배치된 Redis는 성능이 뛰어난 인 메모리 캐시를 생성하여 액세스 지연 시간을 줄이고 처리량을 늘리며 관계형 또는 NoSQL 데이터베이스의 부담을 줄여줌
      - 캐시된 데이터를 한곳에 저장되는 중앙 집중형 구조로 구성하면 MSA 환경의 수평 확장되는 모든 애플리케이션이 Redis 한 곳만 바라보므로 데이터 일관성을 유지할 수 있는 장점이 있음
    - 세션 관리
      - Redis는 세션 관리 작업에 매우 적합
      - Redis를 세션 키에 대한 적절한 TTL과 함께 빠른 키 값 스토어로 사용하면 간단하게 세션 정보를 관리할 수 있음
      - 세션 관리는 주로 게임, 전자 상거래 웹 사이트, 소셜 미디어 플랫폼을 비롯한 온라인 애플리케이션에 필요

Redis

- Redis(REmote DIctionary Server)
  - ✓ 사용 사례
    - 실시간 순위표
      - 명령어 중 ZRANGE, ZREVRANGE. ZRANGEBYSCORE, ZREVRANGEBYSCORE는 ZSet(Sorted Set) 자료 구조를 사용하면 요소가 목록에 유지되고 점수에 따라 정렬되므로 손쉽게 동적 순위표를 생성하여 게임에서 앞서있는 사람이 누구인지 보여주거나 좋아요를 가장 많이 받은 메시지를 게시하거나 선두에 있는 사람이 누구인지 보여주려는 다양한 사례에 사용할 수 있음
    - 분산 락(distribured lock): 분산 환경에서 여러 시스템이 동시에 데이터를 처리할 때는 특정 공유 자원의 사용 여부를 검증하여 Dead Lock을 방지할 필요가 있는데 이때 Redis를 분산 락으로 사용할 수 있음

Redis

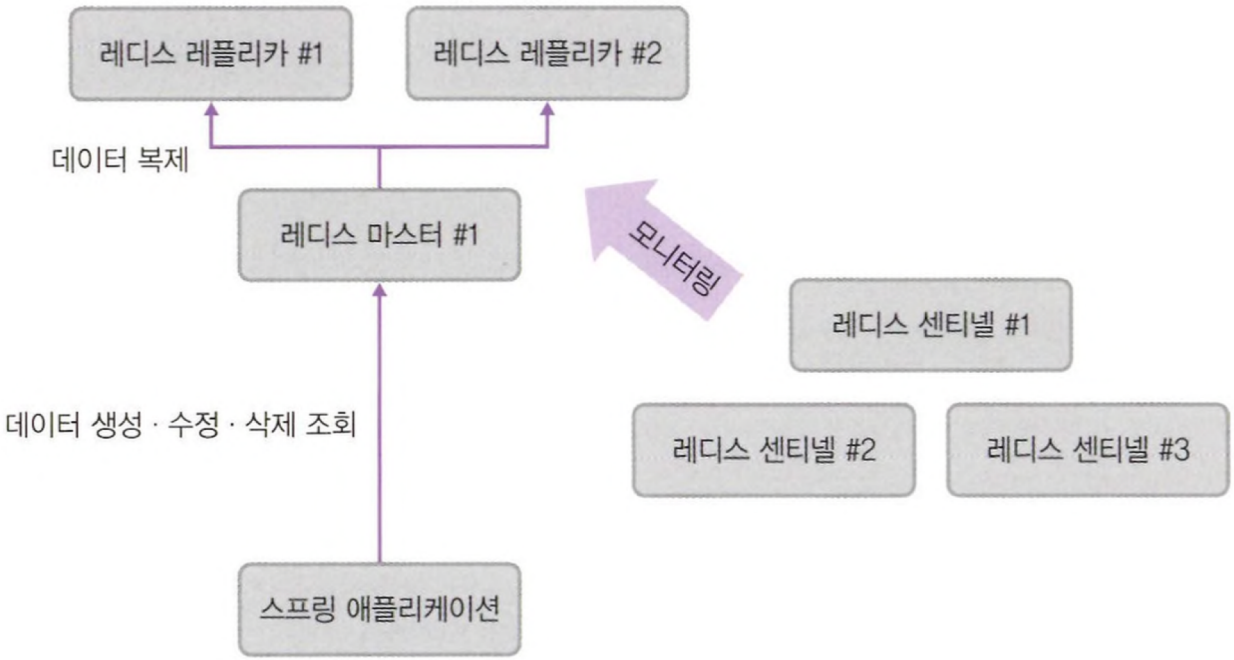
- Redis(REmote DIctionary Server)
  - ✓ 사용 사례
    - 속도 제한
      - Redis는 이벤트 속도를 측정하고 필요한 경우 제한할 수 있음
      - 클라이언트의 API 키에 연결된 Redis 카운터를 사용하여 특정 기간 동안 액세스 요청의 수를 세고 한도가 초과되는 경우 조치를 취할 수 있음
      - 속도 제한기는 포럼의 게시물 수를 제한하고 리소스 사용량을 제한하며 스패머의 영향을 억제하는 데 주로 사용
    - 대기열
      - Redis List 데이터 구조를 사용하면 간단한 영구 대기열을 손쉽게 구현할 수 있음
      - Redis List는 자동 작업 및 차단 기능을 제공하므로 신뢰할 수 있는 메시지 브로커 또는 순환 목록이 필요한 다양한 애플리케이션에 적합
    - 채팅 및 메시징
      - Redis에서는 패턴 매칭과 더불어 PUB/SUB 표준을 지원
      - Redis를 사용하여 고성능 채팅방, 실시간 코멘트 스트림 및 서버 상호 통신을 지원할 수 있음
      - PUB/SUB를 사용하여 게시된 이벤트를 기반으로 작업을 트리거 할 수 있음

Redis

- ❖ Redis(REmote DIctionary Server)
  - ✓ Redis Sentinel Architecture
    - Redis Sentinel 아키텍처는 Redis Sentinel, Redis Master. Redis Replica로 구성
    - Redis Sentinel은 Redis 서버들(Master, Replica)을 관리
    - Sentinel은 Redis 서버들의 상태를 주기적으로 모니터링하고 Master 서버가 서비스할 수 없는 상태가 되면 다른 Replica를 Master 서버로 변경
    - Master 서버가 한 대고 Replica 서버가 두 대 이상으로 구성되어 있으면 새로 승급된 Master 서버부터 생성된 데이터를 복제하도록 나머지 Replica 서버들의 설정을 변경하는데 이렇게 Master 서버에 문제가 생겨 다른 Replica가 Master가 되는 것을 장애 복구(failover)라고 함
    - 클라이언트는 Master 서버에 명령어를 실행
    - 클라이언트는 Master와 Replica 서버 중 에서 Master 서버와 커넥션을 맺어야 하므로 서버 정보를 확인해야 함
    - Redis Sentinel 구조에서 클라이언트는 Redis Sentinel 서버들에 Master 서버를 질의하고 Master 서버에 명령어를 실행
    - Master 서버가 죽어 새로운 Master 서버가 선출되면 Sentinel은 클라이언트에 새로운 Master 서버의 정보를 리포트하므로 애플리케이션에서 Redis 커넥션을 설정할 때 Master 주소가 아니라 Sentinel 서버들의 주소를 설정해야 Sentinel이 지정한 Master Redis와 커넥션을 맺고 명령어를 실행

Redis

- Redis(REmote DIctionary Server)
  - ✓ Redis Sentinel Architecture



Redis

- ❖ Redis(REmote DIctionary Server)
  - ✓ Redis Sentinel Architecture
    - Redis Sentinel 세 대와 Redis Master 한 대, Redis Replica 두 대로 구성된 아키텍처를 표현
    - Redis Sentinel도 단독 서버로 구성한다면 장애에 대응할 수 없음
    - 단독 서버로 구성한 Sentinel에 장애가 발생하면 Redis 서버들을 모니터링할 수 없음
    - Redis Sentinel 서버는 세 대 이상의 홀수로 구성하는 것이 좋은데 그 이유는 Sentinel 서버들은 모니터링 결과를 공유 저장소에 저장하지 않고 각 서버에 각각 저장하는데 Sentinel 서버들과 Master 서버 사이의 네트워크 상태나 Sentinel 호스트 서버의 상태 같은 여러 변수에 따라 일부 Sentinel 서버들은 Master 서버 모니터링에 실패할 수 있는데 Sentinel 서버가 Master 서버 모니터링에 실패하면 Sentinel 서버들은 동의 절차 작업을 실시하는데 일종의 다수결 투표 작업을 하고 다수 표를 얻은 결과를 사용하여 장애 복구 절차를 실행하는데 이런 다수결 투표 작업을 쿼럼(quorum)이라고 하는데 Sentinel이 세 대인 경우 쿼럼 값은 2로 설정되서 두 대 이상의 Sentinel 서버는 Master 서버가 장애라고 판단하면 Master 서버를 서비스에서 제외하고 장애 복구 절차를 실행하는데 네 대의 Sentinel을 설치하고 쿼럼 값을 2로 설정하면 장애 판단을 할 수 없는 경우가 발생하는데 두 대는 장애라고 판단하고 나머지 두 대는 정상이라고 판단할 수 있기 때문에 이런 쿼텀 과정 때문에 Sentinel 서버 개수는 홀수를 유지하는 것을 권장

Redis

- Redis(REmote DIctionary Server)
  - ✓ Redis Cluster Architecture
    - Redis 3.0 버전 이후부터 Redis Cluster 기능이 제공
    - Redis 클러스터는 클러스터에 포함된 노드들이 서로 통신하면서 고가용성을 유지
    - 샤딩 기능까지 기본 기능으로 사용할 수 있음
    - 클러스터 내부에는 Master 노드와 Replica 노드를 설정하여 운영할 수 있음
    - Sentinel 구성처럼 Master 노드와 Replica 노드는 서로 짝을 이루어서 데이터를 복제
    - 클러스터를 구성하려면 세 개의 Master 노드는 반드시 필요한데 설정에 따라 Replica 노드의 개수는 0개 혹은 그 이상으로 설정할 수 있지만 고가용성을 위해 반드시 한 개 이상의 Replica를 설정하는 것을 권장
    - Sentinel 구조와 비교하면 모니터링을 위한 별도의 Sentinel 서버를 구축할 필요가 없는데 클러스터 내부의 모든 노드는 모두 서로 연결되어 있는 메시(mesh) 구조로 되어 있으며 가십 프로토콜(gossip protocol)을 사용하여 서로 모니터링을 하기 때문에 Master 노드에 장애가 발생하면 Replica 노드가 Master 노드로 대체됨

Redis

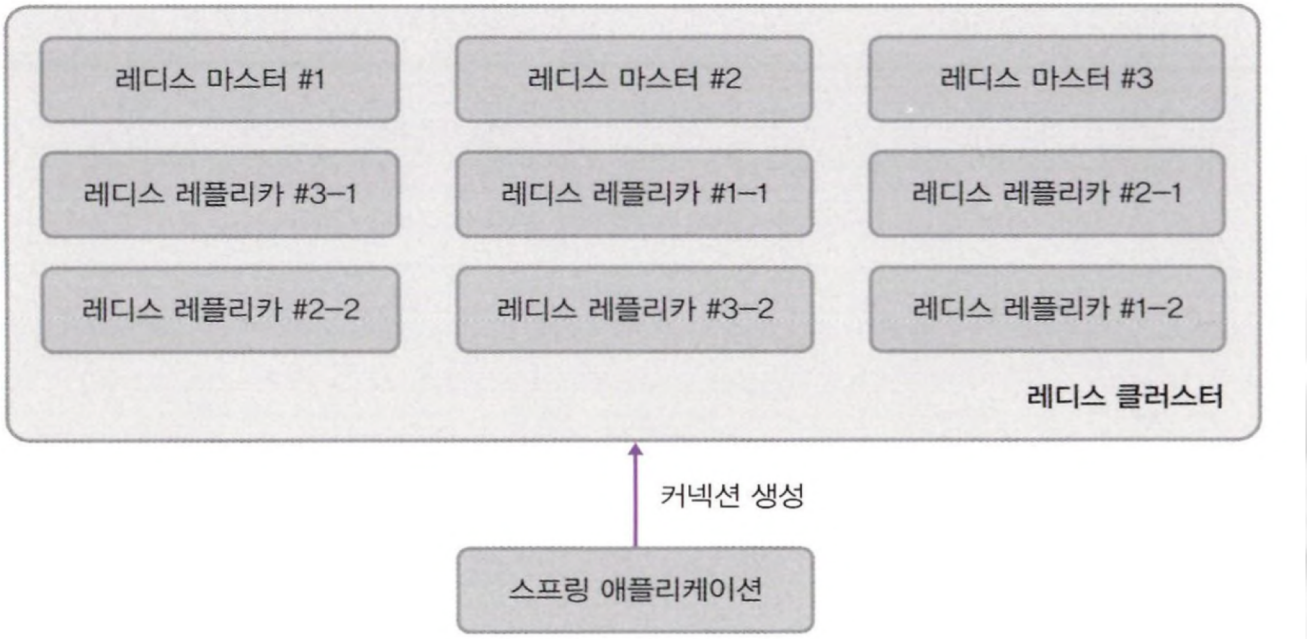
- Redis(REmote DIctionary Server)
  - ✓ Redis Cluster Architecture
    - 클러스터에는 데이터를 Master 노드들에 분배하는 샤딩 기능이 있음
    - Master 노드들은 균등하게 데이터를 분배하여 저장하는데 특정 Master 노드가 다른 Master 노드보다 더 많은 데이터를 저장할 수도 있는데 이런 정보는 클러스터의 모든 노드가 공유하고 있으며 클라이언트에도 공유되기 때문에 클라이언트는 별도의 샤딩 알고리즘을 구현할 필요없이 Redis 클러스터의 알고리즘을 사용하면 됨
    - Redis 클러스터의 샤딩 알고리즘
      - 해시 함수를 사용한 데이터 분배 방식을 이용
      - 해시 함수 값은 항상 0~16383 값을 리턴하는데 클라이언트는 데이터의 키 값을 해시 함수로 실행한 함수 값을 사용하여 어떤 Master 노드에 저장할지 결정
      - 클러스터의 각 Master 노드는 해시 결과 값 범위를 갖고 있어서 세 대의 서버가 데이터를 균등하게 분배하기로 결정했다면 해시 함수 값 0~5460은 1번 Master 노드에 5461~10922는 2번 Master 노드에 10923~16383은 3번 Master 노드에 균등 분배
      - 클라이언트가 실행한 해시 함수의 결과 값이 2321 이라면 0~5460을 저장하는 1번 Master 노드에 저장하는데 마찬가지로 클라이언트가 데이터를 조회할 때도 해시 함수의 결과 값에 따라 데이터가 저장된 노드 위치를 알 수 있음

Redis

- Redis(REmote DIctionary Server)
  - ✓ Redis Cluster Architecture
    - Cluster에 노드를 추가하거나 제거한다면 Redis 클러스터 명령어를 사용하여 해시 함수 값 범위를 조정할 수 있는데 조정된 범위에 포함되는 Redis 데이터들은 자동으로 재분배되어 설정된 위치로 이동하는데 이를 리밸런싱(re-balancing) 혹은 리샤드(re-shard)라고 함
    - 클라이언트는 리밸런싱 과정 중에 이동된 데이터 위치를 실시간으로 추적할 수 없음
    - 해시 결과 값이 5460인 데이터를 클라이언트가 읽으려고 할 때 이 해시 값에 해당하는 데이터들이 1번 노드에서 2번 노드로 재분배 중이라면 재분배 시간 동안 클라이언트는 여전히 1번 노드에 질의할 수 있고 해당 데이터는 리밸런싱되어 2번 노드에 있을 수 있음
    - 클라이언트가 1번 노드에 질의하면 노드는 (error) MOVED 결과와 함께 데이터를 관리하고 있는 Redis 노드의 주소를 응답하고 클라이언트는 이 정보를 사용하여 다시 해당 서버에 질의하는데 이 과정이 (error) Moved Redirection

Redis

- Redis(REmote DIctionary Server)
  - ✓ Redis Cluster Architecture
    - 그림은 한 대의 Master와 두 대의 Replica를 하나의 세트로 구성한 클러스터 아키텍처로 세 대의 Master가 있으므로 총 세 대의 세트에 데이터가 샤딩되는데 그림에서 Master #1의 데이터는 Replica #1-1과 Replica #1-2에 복제



Redis

- Redis(REmote DIctionary Server)
  - ✓ Redis Cluster Architecture
    - 애플리케이션 클라이언트는 Redis 클러스터의 노드 중 하나라도 연결되면 클러스터의 전체 상태 정보를 확인할 수 있으며 클러스터 내부의 노드는 전체 노드 정보를 알 수 있기 때문
    - 클러스터에 한 번 접속한 클라이언트는 장애가 발생한 노드나 확장을 위해 추가한 노드 정보들도 모두 업데이트받기 때문에 Production 환경에서 운영 중 증설을 하더라도 애플리케이션의 설정을 변경할 필요가 없음