13장. 레디스 운영하기

> 레디스를 장애없이 운영하기 위해 실시간으로 모니터링하고 레디스 대시보드를 작성해보자 !

레디스 모니터링 구축

1. 프로메테우스와 그라파나 이용하기

익스포터

시스템의 상태를 실시간으로 스크랩해서 메트릭을 수집하는 프로그램

- **레디스 익스포터**: 레디스 인스턴스의 실시간 정보를 수집

```
# 레디스 패스워드 지정 or 실행 포트 변경 등 변경이 있으면 익스포터 실행
$ cd redis_exporter
$ nohup ./redis_exporter --redis.addr=redis://127.0.0.1:73
password=passsword > redis_exporter.log &
```

- **노드 익스포터**: 레디스가 실행되는 서버의 하드웨어와 OS 관련 메트릭을 수집

프로메테우스

메트릭 기반의 오픈소스 모니터링 시스템

- 시계열 형태로 간단하고 빠른 데이터 수집
- 타깃으로 직접 접근해 데이터를 pull 방식으로 수집

```
# 모니터링할 타깃 prometheus.yml에 지정
$ cd prometheus
$ vi prometheus.yml
```

- 알람 규칙을 설정해서 수집한 메트릭을 그라파나로 보고 메트릭별 임계치를 지정할 수 있다.

그라파나

오픈소스 메트릭 데이터 시각화 도구

- 여러 데이터 소스를 연동할 수 있음
- 프로메테우스를 데이터 소스로 추가하면 각 레디스 서버의 메트릭을 수집한 프로메테우스의 정보를 시각화

얼럿 매니저

모니터링하는 대상이 임계치에 도달했을 때 사용자에게 통지해주는 프로그램

- 서비스의 상태를 전달받을 엔드포인트를 SMS, 이메일, 슬랙 등으로 지정 가능

```
# 특정 임계치에 도달했을 때 디스코드를 이용해 알람을 받도록 얼럿 매니저 설정
$ cd alertmanager
$ vi alertmanager.yml

# alertmanager.yml
route:
    group_by: ['alertname', 'job']
    group_wait: 0s
    group_interval: 5s
    repeat_interval: 1m
    receiver: discord

receivers:
- name: discord
    discord_configs:
        - webhook_url: https://discord.com/api/webhooks/aaa/b
```

2. 레디스 플러그인과 그라파나 이용하기

1. 그라파나에서 레디스 플러그인 설치
2. RedisGrafana에서 제공하는 대시보드 이용
3. 실시간으로 레디스 상태 확인 !

레디스 익스포터를 설치할 필요가 없어 온프레미스/클라우드 레디스에도 실시간 대시보드 확인 가능

특징

- 슬로우 로그, 어떤 클라이언트가 어떤 커맨드를 수행했는지 하나의 창에서 확인 가능
- redis-cli 패널을 제공해 그라파나에서 바로 레디스에 커맨드를 수행 가능

레디스 버전 업그레이드

우리 레디스는요 . . .

- 릴리스 주기가 다른 DB에 비해 굉장히 빠른편
- EOL도 짧은 편

버그와 보안 취약점 방지를 위해 최소한 EOL되지 않은 버전을 사용하도록 주기적으로 업그레이드하자.

업그레이드 방법

1. 업그레이드할 버전의 레디스 인스턴스를 새로운 서버에 설치한 뒤 기존 버전의 레디스의 데이터를 복제한다.
    - 운영 중인 애플리케이션에서 레디스 접속 정보 변경 필요
    - 다운타임이 존재하지 않음
2. 실행 중인 레디스 인스턴스를 중지한 다음 신규 버전의 레디스 소스 파일로 재실행한다.
    - 레디스의 접속 정보를 변경하지 않아도 됨
    - 싱글 구성의 레디스라면 다운타임이 존재

센티널 구성의 레디스 버전 업그레이드

1. 신규 버전의 레디스 바이너리 파일 다운로드
2. 센티널 인스턴스 모두 중단
3. 신규 버전 폴더에 기존의 `sentinel.conf` 복사
4. 신규 바이너리 파일을 이용해 3대의 센티널 인스턴스 시작
5. 복제본 인스턴스 중단

    - 현재 실행중인 레디스 인스턴스의 정보가 설정 파일에 반영되도록 `config rewrite` 수행

6. 신규 버전 폴더에 기존의 `redis.conf` 복사
7. 신규 바이너리 파일을 이용해 복제본 인스턴스 시작
8. 센티널에서 수동 페일오버 수행
9. 기존 마스터 인스턴스 중단
10. 신규 버전 폴더에 기존의 `redis.conf` 복사
11. 신규 바이너리 파일을 이용해 기존 마스터 인스턴스 시작
12. 센티널에서 수동 페일오버 수행

클러스터 구성의 레디스 버전 업그레이드

A,B,C 노드가 마스터, D,E,F 노드가 복제본이라고 가정

1. 복제본 노드(D, E, F) 각각 버전 업그레이드
2. D 노드에서 페일오버
3. A 노드 업그레이드
4. E 노드에서 페일오버
5. B 노드 업그레이드
6. F 노드에서 페일오버
7. C 노드에서 업그레이드

레디스 운영 가이드

장애와 성능 저하 요소는 무엇이 있을까

🔍 레디스 설정 항목

maxmemory-policy

메모리 한계에 도달했을 때 어떤 키를 제거할 것인가 (기본값: `noeviction` - 에러 반환)

- 로직에 따라 애플리케이션의 장애로 이어질 수 있음 에러 반환하니까

- 임의로 데이터가 삭제되더라도 계속 새로운 입력을 받게하려면 `allkeys-lru` 권장

stop-writes-on-bgsave-error

RDB 스냅샷이 정상 저장되지 않으면 모든 쓰기 작업 중지 → 더 큰 장애 방지 (기본값: `yes`)

- 모니터링을 따로 두고 이 설정을 비활성화해서 백업을 못해도 어플리케이션이 돌게 해야 함

자동 백업 옵션

- 백업이 포그라운드로 수행된다면 . . .
  - 실행되는 동안 다른 작업들 대기 → 성능에 영향
- 백업이 백그라운드로 수행된다면 . . .
  - COW 방식으로 작동하기 때문에 메모리 사용량이 최대 maxmemory의 2배까지 증가 가능

    → 최악의 경우 OOM 발생

→ 백업은 의도한 시간에 의도한 레디스 인스턴스에서 실행되도록 설정하자

save 옵션

```
# save 기본값: 1시간 동안 1번, 5분 동안 100번, 1분 동안 10000번 변경
> CONFIG GET save
1) "save"
2) "3600 1 300 100 60 10000"

# save 기본 설정값 비활성화
> CONFIG SET save ""
OK
```

appendonly 옵션

```
# appendonly 기본값: AOF 파일 크기가 기존 파일보다 100% 증가하면 자동
> CONFIG GET auto-aof-*
1) "auto-aof-rewrite-percentage"
2) "100"
3) "auto-aof-resrite-min-size"
4) "67108864"
```

```
# appendonly 기본 설정값 비활성화
> CONFIG SET auto-aof-rewrite-percentage 0
OK
```

🔍 레디스 성능 최적화

오래 걸리는 커맨드 사용

(복습) 레디스 특징

- 레디스는 싱글 스레드로 동작
- 모든 요청은 이벤트 루프를 이용한 순차적 실행
- 한 번에 하나의 커맨드만 처리 가능, 나머지는 대기

→ O(N) 이상의 시간 복잡도를 갖는 커맨드는 사용을 지양하자

🤔 뭐가 오래 걸리나?

- 한 번에 여러 키에 접근하는 경우 (당연함)
- 하나의 자료 구조 안에 여러 개의 아이템을 갖고 있는 경우
- 아이템 개수에 비례해 실행 시간이 증가하는 방식으로 동작하는 경우

레디스에서의 트랜잭션 사용

- 원자적으로 수행돼야 하는 커맨드의 일관성을 보장할 수 있음
  - `MULTI`: 트랜잭션 시작
  - `EXEC`: 입력했던 커맨드 원자적 실행 후 성공 시 결과 반환

루아 스크립트

가볍고 빠르며 임베디드가 가능한 스크립트 언어

- 레디스는 루아 스크립트 실행 기능을 내장하여 데이터 조작/계산 작업을 루아 스크립트로 처리할 수 있다.
- 레디스 내에서 원자적으로 실행되고 다른 클라이언트 요청을 수용하지 않아 데이터 일관성을 유지한다.
- 트랜잭션과 비슷하지만 롤백이 발생하지 않는다.

- 데이터를 매번 네트워크로 전송하지 않고 서버에서 계산을 수행해서 성능을 향상시킬 수 있다.
- `SCRIPT LOAD`: 루아 스크립트를 레디스에 로드 후 hash 값 반환
  - 한 번 로드한 이후에는 해시값을 사용해 스크립트 실행 가능

주의 사항

- 트랜잭션의 길이가 길어지지 않도록 주의
- 블로킹 커맨드를 사용할 수 없음
  - 트랜잭션 내부에서 블로킹될 경우 레디스가 무한 대기에 빠짐

has-get / has-del

데이터 존재 여부를 확인한 뒤 처리하는 패턴

- 네트워크 부하를 늘림
- 불필요한 작업을 수행해 성능을 저하시킴 (라운드 트립 2번 발생)
- 원자성 문제 발생 가능

→ 결론: 쓰지 말자

클라이언트 출력 버퍼 사이즈

- 동시에 많은 클라이언트 요청을 처리해야 하는 환경에서는 출력 버퍼 크기를 늘릴 것
- 복제를 사용하는 경우 필수 ! 큰 데이터를 복제할 때 기본값으로는 부족할 수 있다.

```
# redis.conf
client-output-buffer-limit normal 0 0 0
client-output-buffer-limit replica 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60
```

레디스 키스페이스 알림

레디스 내부 키에 대한 변경 사항 모니터링 및 pub/sub 채널을 이용한 변경 사항 구독

ex) 키의 만료를 감시하다가 만료 이벤트 발생 시 애플리케이션에서 추가 작업 수행

```
# notify-keyspace-events 설정으로 어떤 알림을 수신할 지 지정
# 만료된 키 이벤트 수신
> CONFIG SET notify-keyspace-events Ex
OK

# 키가 만료될 때 메시지 수신 (pub/sub 채널 구독)
> SUBSCRIBE __keyevent@0__:expired
1) "message"
2) "__keyevent@0__:expired"
3) "my_key"
```

주의 사항

- 많은 키가 동시에 만료되는 경우
  - pub/sub 버퍼 크기를 초과하면 새 메시지가 유실될 수 있다.
  - pub/sub은 fire and forget 방식으로 한 번 발행된 이벤트는 재확인할 수 없다.

  → 많은 키가 동시에 만료될 가능성이 있는 서비스는 pub/sub 버퍼 크기를 늘리자

  → 아니면 레디스 인스턴스를 여러 개 사용해 pub/sub 이벤트를 분산 처리하자.

🔍 레디스 모니터링

슬로우 로그

실행 속도가 느린 커맨드를 기록하는 로그

```
> SLOWLOG GET
1) 1) (integer) 1923
      2) (integer) 16966344048
      3) (integer) 35233
      4) 1) "SCAN"
                2) "10179327"
                3) "COUNT"
                4) "50000"
```

- **실행 시간**: 명령이 실행된 시간 정보
- **실행 시간**: 명령이 실행되는 데 소요된 시간
- **명령**: 느리게 수행됐던 커맨드
- **인자**: 느린 명령에 대한 인자 정보

설정 변경

- `slowlog-log-slower-than` 설정값으로 기준 변경 `(기본 10,000ms = 10초)`
- `slowlog-max-len` 설정값으로 레코드 개수 제한 `(기본 128개)`

그래프 지표

컴퓨팅 자원(CPU, 메모리, 네트워크, …)을 얼마나 쓰고 있는가

- **과다 사용**: 높은 지연 시간 및 성능 저하 초래
- **과소 사용**: 요구사항에 맞지 않는 리소스 사용중

CPU

- 시간 복잡도가 높은 커맨드를 자주 사용하면 부하가 늘어남
- 집합 자료 구조의 카디널리티가 높을수록 성능 저하의 원인이 됨
- 단일 hash에 저장되는 항목이 너무 많으면 CPU에 영향을 줌

메모리

- `used_memory`: 레디스가 현재 할당한 메모리
- `DatabaseMemoryUsagePercentage`가 100%에 도달하면 `maxmemory` 정책 작동

`데이터셋 크기, TTL을 고려해 메모리 사용량을 관리하고 사용량이 급증하는 경우를 미리 감지하자`

네트워크

- 병목 현상 발생 시 레디스 성능에 영향을 줌
- 네트워크 지표가 특정 지점에서 더 이상 상승하지 않는다면 레디스 실행 환경에서 네트워크 임계치에 걸려 병목이 발생하는지 확인 필요
- 네트워크 사용량이 증가하는 경우
    - **읽기 작업이 원인**: 복제본을 주로 활용하는지 확인, 복제본을 쓰고 있다면 추가 복제본 구성

    - **쓰기 작업이 원인**: 서버 사양을 업그레이드 or 클러스터 모드로 변경해 부하 분산

커넥션

- 레디스의 활성 연결 수와 신규 연결 수를 주시
    - 활성 연결 수 급증: 애플리케이션에서 문제 발생 or 유휴 연결 문제(tcp-keepalive 설정 활용)
- 새로운 연결 설정은 비용이 많이 드니까 `커넥션 풀`을 두고 기존 연결을 재사용할 것

복제

- 복제 지연이 발생하는지 확인
    - 복제 지연의 급증 == 마스터 노드의 속도를 복제본이 따라가지 못함
    - 복제본이 전체 동기화 요청 시 마스터가 스냅샷을 생성하여 성능 저하 발생 가능