콘서트 예약 서비스에서 제공하는 API 는 다음과 같습니다.

◆ 토큰 발급
◆ 포인트 충전
◆ 잔액 조회
◆ 콘서트 목록 조회
◆ 콘서트 날짜 조회
◆ 콘서트 좌석 조회
◆ 예약
◆ 결제

부하테스트는 적절한 운영스펙을 유추하기 위해 진행을 하게 되는데,
부하테스트가 필요한 기능의 조건은 아래처럼 생각을 했습니다.

◆ 성능이 중요한 기능
◆ 실제 많은 트래픽이 예상되는 기능

따라서, 조건에 따라 부하테스트가 필요한 기능은
“ 토큰 발급, 콘서트 좌석 조회 ”
3가지로 판단했습니다.

테스트 시나리오

- 토큰 발급

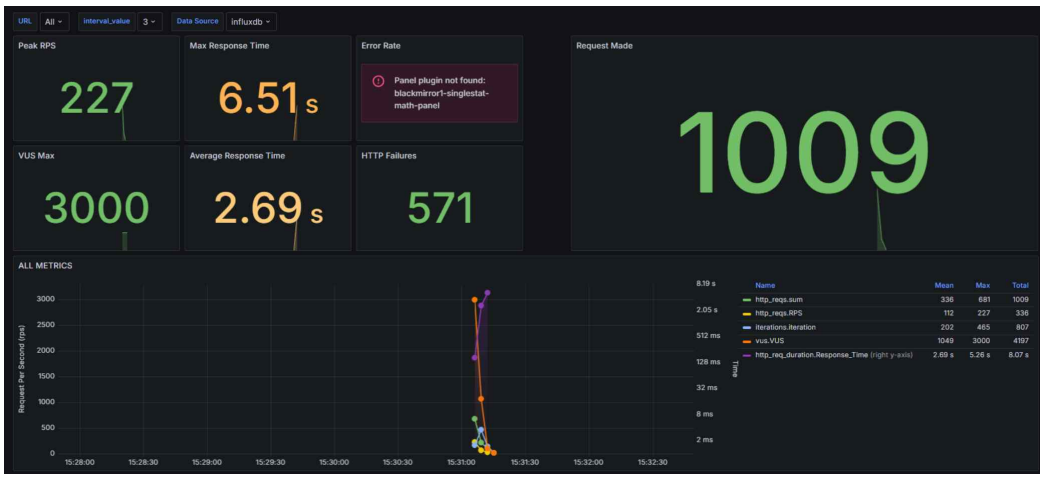
: 10초동안 3000명이 토큰발급을 한번씩 (총 3000번) 시도 했을 때 정상발급하는지 여부

[테스트 한 스크립트]

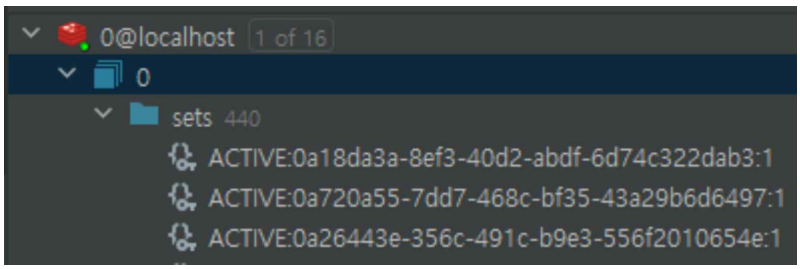
```
export let options = {
    vus: 3000,  // 가상 사용자 설정
    iterations: 3000,  // 총 3000번의 요청 실행 (각 사용자당 1회씩)
    duration: '10s',  // 10초 동안 실행
};

export default function () {
    issue()
    sleep(1);
}

function issue() {
    const body = JSON.stringify( value: {
        userId: 1
    });

    let response = http.post(
        `http://localhost:8080/api/token/issued`,
        body,
        {
            headers: {
                'Content-Type': 'application/json'
            }
        }
    )
    // 요청이 성공했는지 확인
    check(response, {
        'is status 200': (r) => r.status === 200
    });
}
```

userId 는 같아도 발급은 계속 되기에 1로 진행하였습니다.

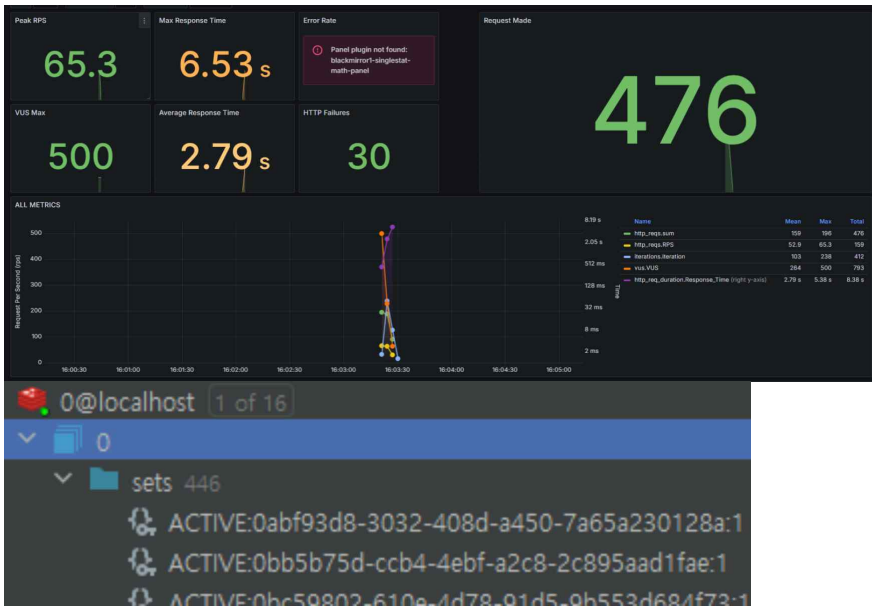
[k6 run 한 후 Grafana 화면]



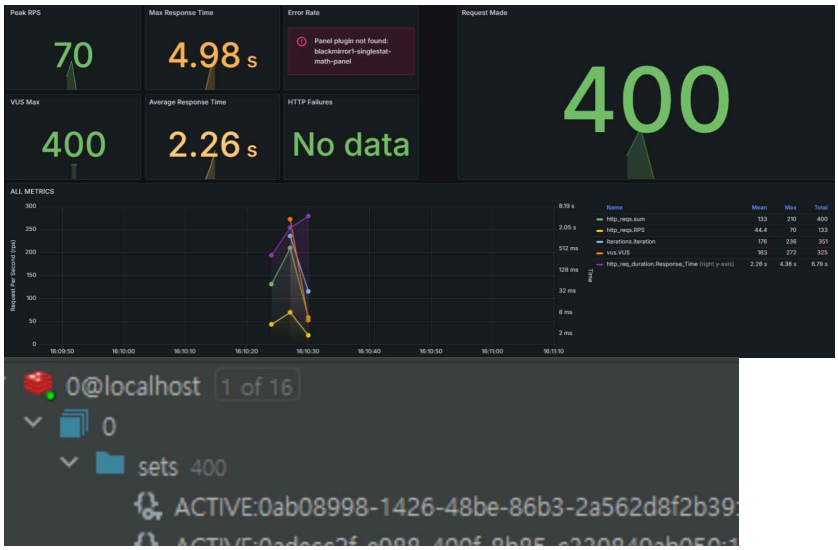
3천명이 몰렸지만 Reqeust 자체는 1009개 밖에 만들지 못했고,
실제 Token이 저장되는 Redis 에는 아래 사진처럼 440개만 들어온 걸 확인할 수 있었습니다.



[10초에 500명이 한번씩 발급 시도 (총 500번)]



[10초에 400명이 한번씩 발급 시도 (총 400번)]



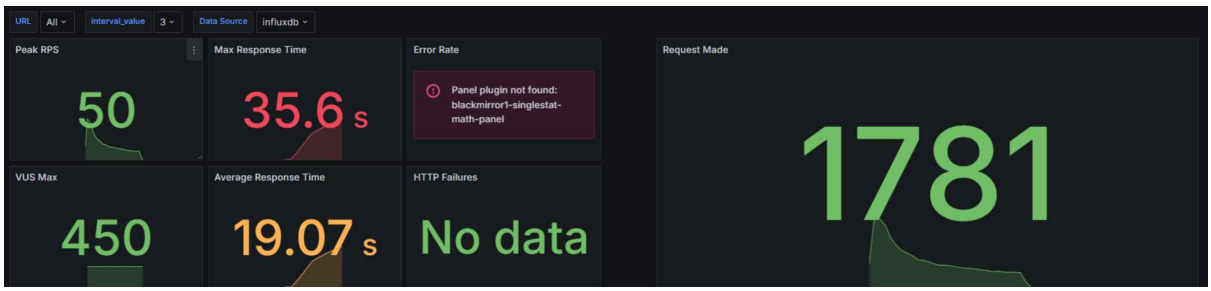
400번은 현재 환경에서 유실없이 정상적으로 발급되는걸 확인할 수 있었습니다.

: 평상시 트래픽 (50명) 과 몰렸을 때 (400명) 의 시나리오를 병렬로 실행

[시나리오 설정 부분]

```
export let options = {
    /*vus: 400,  // 가상 사용자 설정
    iterations: 400,  // 총 3000번의 요청 실행 (각 사용자당 1회씩)
    duration: '10s',  // 10초 동안 실행*/
    scenarios: {
        normal_traffic: {
            executor: 'constant-vus',  // 평상시 트래픽을 유지하는 시나리오
            vus: 50,                   // 50명의 가상 사용자를 유지
            duration: '1m',            // 1분 동안 유지
        },
        spike_traffic: {
            executor: 'ramping-vus',   // 몰렸을 때
            startVUs: 0,
            stages: [
                { duration: '10s', target: 400 },  // 10초 동안 400명으로 증가
                { duration: '30s', target: 400 },  // 30초 동안 400명 유지
                { duration: '10s', target: 0 },     // 10초 동안 0명으로 감소
            ],
            startTime: '10s',          // 10초 후에 시작하여 평상시 트래픽과 겹치게 실행
        },
    },
};
```

[결과]



유실 없이 요청이 모두 진행되는걸 확인할 수 있었습니다.

- 콘서트 좌석 조회

: A 콘서트에는 사람이 몰리지 않고, B콘서트에 1000명이 동시에 몰렸을 때의 시나리오

[테스트 스크립트의 옵션]

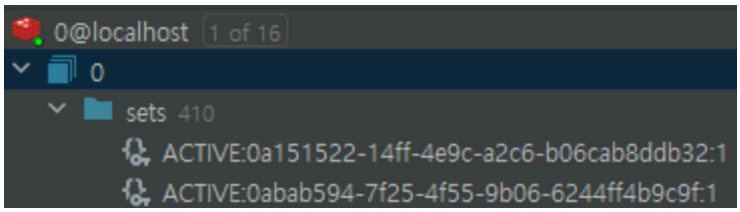
```
export let options = {
    scenarios: {
        B_concert: {
            executor: 'per-vu-iterations',
            vus: 10,                    // 10명의 가상 사용자를 유지
            iterations: 1,
            exec: 'getBConcertSeat',    // B 콘서트 접근을 시뮬레이션하는 함수 실행
        },
        A_concert: {
            executor: 'per-vu-iterations',
            vus: 400,                       // 400명의 가상 사용자를 설정
            iterations: 1,
            exec: 'getAConcertSeat',        // A 콘서트 접근을 시뮬레이션하는 함수 실행
        },
    }
};
```

[A Concert 좌석조회시 사용한 테스트 스크립트]

(B도 파라미터만 다르게 되어있습니다)

```
export function getAConcertSeat() {

    const token = issueToken();
    const concertDateTime = encodeURIComponent( uriComponent: "2024-08-15 10:00");

    let response = http.get(
        `http://localhost:8080/api/concert/seats?concertId=1&concertDateTime=${concertDateTime}&userId=1`,
        JSON.stringify( value: {}),
        {
            headers: {
                'Content-Type': 'application/json',
                'Queue-Token': token
            }
        }
    )
    // 요청이 성공했는지 확인
    let chesResp = check(response, {
        'is status 200': (r) => r.status === 200
    });
    if (!chesResp) {
        console.log("A 실패")
    }
    sleep(1);   //결과 본 후 최소 1초는 가만히 있는다고 가정
}
```

[결과]



토큰은 410개 모두 발급되었지만

각 response 실패 시 찍는 console.log() 를 확인해보면 B콘서트 조회는 하나도 실패하지 않지만 A콘서트를 대부분 실패하는걸 확인했습니다.

해당 부분은 Step20인 장애보고서에서 다루겠습니다.