HW1

2021320314 고건영

1. 대시보드 생성과 각 메트릭 별 screenshot

2. 각 메트릭 별 쿼리와 설명

- **cpu usage (왼쪽 위 메트릭)**

query: `100 * (sum by (instance)(rate(node_cpu_seconds_total{mode!="idle"}[1m])) / (sum by (instance)(rate(node_cpu_seconds_total[1m])))`

이 쿼리는 CPU가 idle 상태가 아닌 시간의 비율을 계산하여 실제 CPU 사용률을 퍼센트로 표시한다. `rate()` 함수는 최근 1분간 CPU 사용 시간의 변화량을 계산하고, `mode!="idle"`을 통해 비유휴 CPU 시간을 추출한다. 마지막으로 전체 CPU 사용 시간 대비 사용 중인 시간의 비율을 곱해서 CPU 사용률로 나타낸다. 100을 곱해서 백분율로 표현한다. 여기서는 cpu 사용량이 운영체제(OS)와 기본 프로그램 때문에 약 2프로정도를 유지하는 모습을 보인다.

- **Memory usage (오른쪽 위 메트릭)**

query: `(1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)) * 100`

이 쿼리는 사용 가능한 메모리 대비 총 메모리의 비율을 계산하여 메모리 사용률을 퍼센트로 나타낸다. `MemAvailable`은 실제로 즉시 사용할 수 있는 메모리를 의미하며, 이를 `MemTotal`로 나누어 남은 메모리 비율을 계산한다. 1에서 이 값을 빼고 100을 곱하면 현재 사용 중인 메모리의 백분율을 얻을 수 있다. 여기서는 메모리 사용량이 평소에 28%~30% 정도를 유지하는 모습을 보인다.

- **Nginx accepted connections (왼쪽 아래 메트릭)**

query: `nginx_connections_accepted`

이 쿼리는 Nginx가 수락한 누적 연결 수를 나타낸다. 시간에 따라 값이 계속 증가하는 카운터이다. `rate(nginx_connections_accepted[1m])` 형태로 초당 증가율을 확인할 수 있으나, 직관적으로 연결수를 확인하기 위해 단순 메트릭만 사용하였다. 그래프 상에서는 평소에 평평한 모습이고, 별다른 연결이 없음을 확인할 수 있다.

- **Nginx HTTP Requests(오른쪽 아래 메트릭)**

query: `nginx_http_requests_total`

이 메트릭은 Nginx가 처리한 HTTP 요청의 총 누적 개수를 나타낸다. 시간에 따라 값이 계속 증가하는 카운터이다. 위와 마찬가지로 `rate(nginx_http_requests_total[1m])` 형태로 초당 증가율을 확인할 수 있으나, 직관적으로 연결수를 확인하기 위해 단순 메트릭만 사용하였다. 그래프 상에서는 지속적으로 약간식 우상향하는 모양이고 지속적인 HTTP 요청을 처리하고 있음을 보인다. 이는 메트릭 수집에 의한 요청으로 보인다.

3. ApacheBench loads 시뮬레이션

- **Small load: ab -n 2000 -c 100**

17:53:00 쯤 apachebench로 요청을 보냈고 그 결과, cpu 사용량이 약 2.6프로까지 상승했다가 요청을 다 처리한 후 다시 2프로대로 하강하는 모습을 보임을 확인할 수 있었고, 아주 작은 수준의 부하가 가해졌음을 확인하였다.

추가로 nginx http requests 와 nginx accpeted connections도 보낸 요청 수 2000 만큼 지표가 상승하였음을 확인할 수 있었다.

- **Medium load: ab -n 10000 -c 500**

17:59:30 쯤 apachebench로 요청을 보냈고 그 결과, cpu 사용량이 약 5프로까지 상승했다가 요청을 다 처리한 후 다시 감소하는 모습을 보임을 확인할 수 있었다. 요청을 2000개 보냈을 때보다는 더 높은 수준의 부하가 가해졌음을 확인하였다.

추가로 nginx http requests 와 nginx accpeted connections도 보낸 요청 수 10000 만큼 지표가 상승하였음을 확인할 수 있었다.

- **Large load: ab -n 100000 -c 1000**

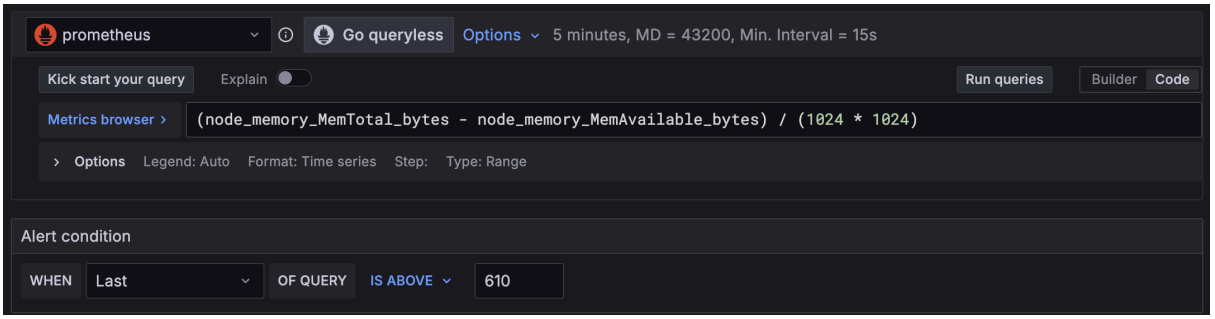
17:47:30 쯤 apachebench로 요청을 보냈고 그 결과, cpu 사용량이 약 25프로정도까지 상승했다가 요청을 다 처리한 후 다시 2%대로 복귀한 모습을 보임을 확인할 수 있었고, 이전의 실험들보다 훨씬 더 높은 수준의 부하가 가해졌음을 확인하였다.

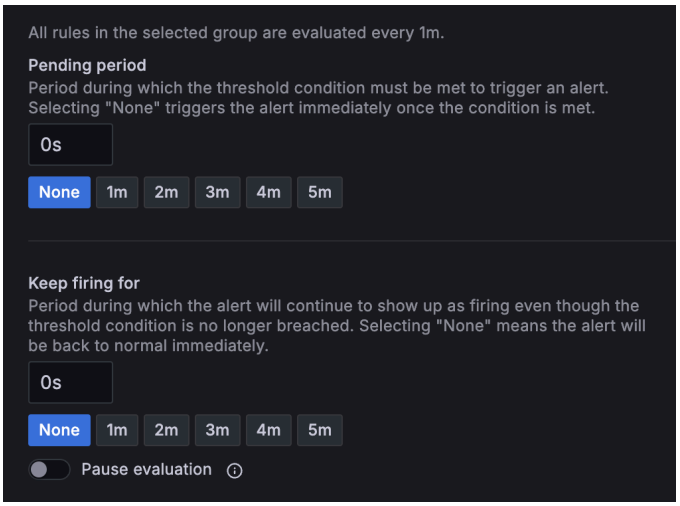
추가로 nginx http requests 와 nginx accpeted connections도 보낸 요청 수 100000 만큼 지표가 상승하였음을 확인할 수 있었다.

3가지 실험 모두에서 메모리 사용량의 경우 29%~32% 구간을 유지하고, 요청과 별 관련이 없어 보인다. 이를 통해 Nginx 요청 처리에서는 CPU 사용률이 주요 자원 소비 요소이며, 메모리 사용량은 상대적으로 안정적으로 유지됨을 확인할 수 있었다.

4.Alerting 실험

- **query and alert condition**





query: `(node_memory_MemTotal_bytes - node_memory_MemAvailable_bytes) / (1024 * 1024)`

이 쿼리는 현재 사용 중인 메모리 양을 MB 단위로 계산한다. `MemTotal`에서 `MemAvailable`을 빼면 실제로 사용 중인 메모리 바이트 값을 얻을 수 있고, 이를 `1024 * 1024`로 나눠 MB로 변환한다.

alert condition: `WHEN` **`Last`** `OF QUERY` **`IS ABOVE`** **`610`**

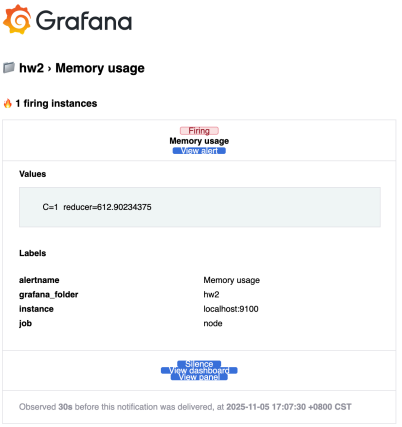
이 설정은 가장 최근(Last) 측정된 메모리 사용량 값이 610MB를 초과할 경우 알람을 발생시키는 조건이다.즉, 실시간으로 메모리 사용량이 기준치를 넘는 순간, 해당 패널은 Alerting 상태로 전환된다.

pending period는 Alert 조건이 충족되었을 때 즉시 알람을 발생시키는 것이 아니라, 조건이 일정 시간 동안 지속되는지 검증하는 지연 시간을 의미하고 메모리 사용량이 임계값(610MB)을 넘는 즉시 Alert가 발생할 수 있도록 `0s`로 설정하였다.

keep firing for는 Alert가 이미 활성화된 상태에서, 임계값을 다시 정상 범위로 내려가더라도 일정 시간 동안 Alert 상태를 유지하는 시간이다. 메모리 사용량이 다시 기준 이하로 떨어지면 Alert 상태는 즉시 해제될 수 있도록 `0s`로 설정하였다.

이러한 설정을 일시적인 스파크를 검출하기에 적합한 설정이다.

- **실험 결과**



왼쪽 사진은 기존 메모리 사용량 메트릭 그래프이다. 그래프 중앙의 빨간색 점선 시점에서 메모리 사용량이 설정한 임계값(610MB)을 초과하면서 Alert가 발생하였다. 해당 시점 이후 메모리 사용량은 일정 구간 동안 높은 수준을 유지하며 패널 상태가 Firing 상태로 전환된다. 이후 그래프에서 초록색 점선 시점은 메모리 점유가 감소하여 임계값 아래로 다시 내려간 순간을 의미한다. 이 시점을 기준으로 Alert 조건이 더 이상 충족되지 않기 때문에 알람 상태가 정상(Normal)으로 자동 복구되었다.

오른쪽 사진은 Grafana에서 Memory usage Alert가 실제로 발동(Firing) 된 상태를 사전에 지정해둔 이메일로 전송 받은 것이다. 알림 메시지에는 현재 측정된 메모리 사용량 값 (`reducer=612.90`)이 함께 표시된다.