클라우드컴퓨팅 과제1

2022320009 이수현

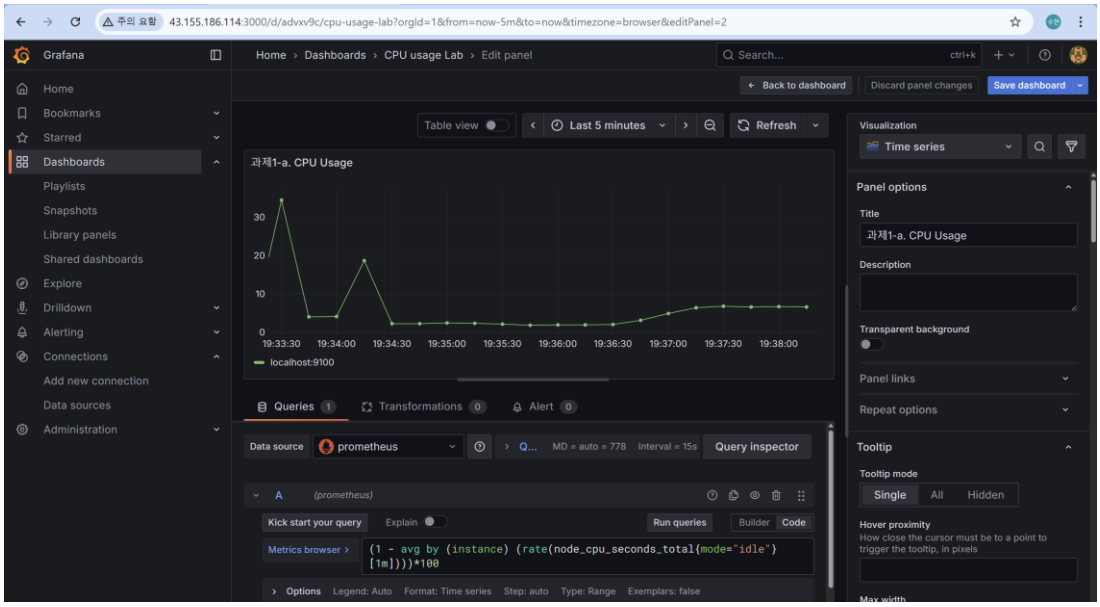
Problem 1.

Create a Grafana dashboard with the following performance metrics and include your query. You can generate the following performance metrics by combining and calculating various metrics. Some can be outputted as a single metric.-provide the dashboard screenshot and the query in text format separately:

a. CPU usage (%, Normalized across all cores; not a per-core sum, rate window=1m)

쿼리) (1 - avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[1m])))*100

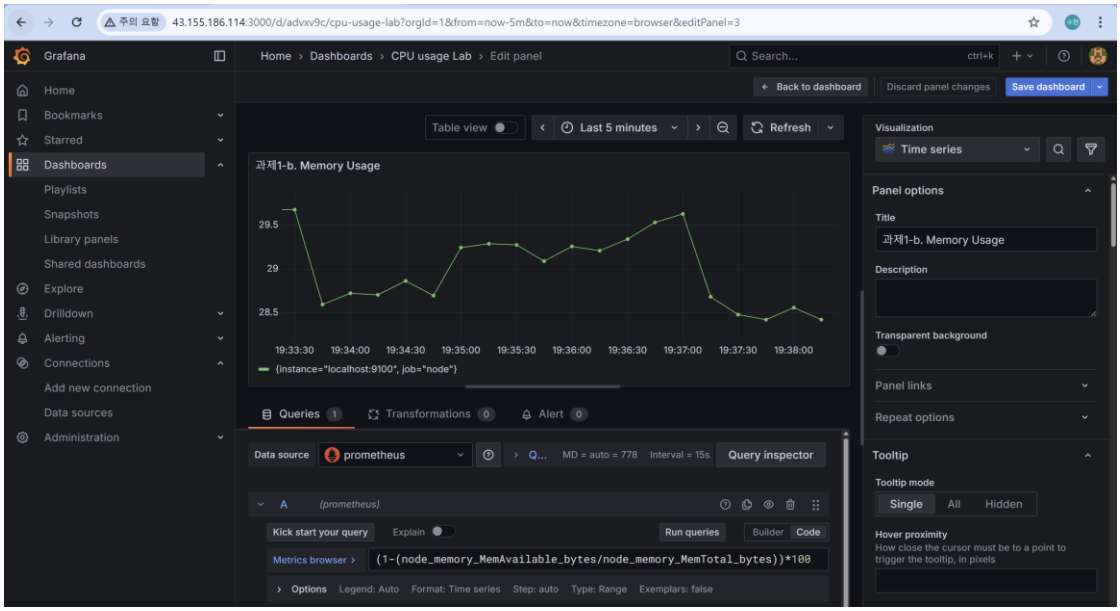
이미지)



b. Memory usage (%)

쿼리) (1-(node_memory_MemAvailable_bytes/node_memory_MemTotal_bytes))*100

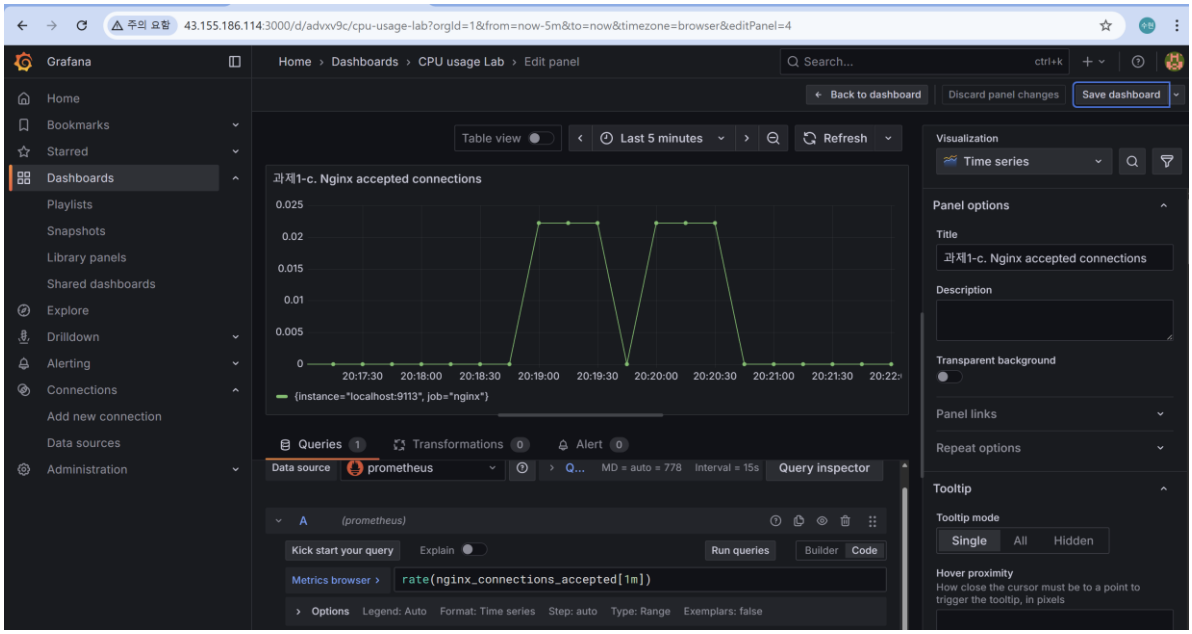
이미지)



c. Nginx accepted connections

쿼리) rate(nginx_connections_accepted[1m])

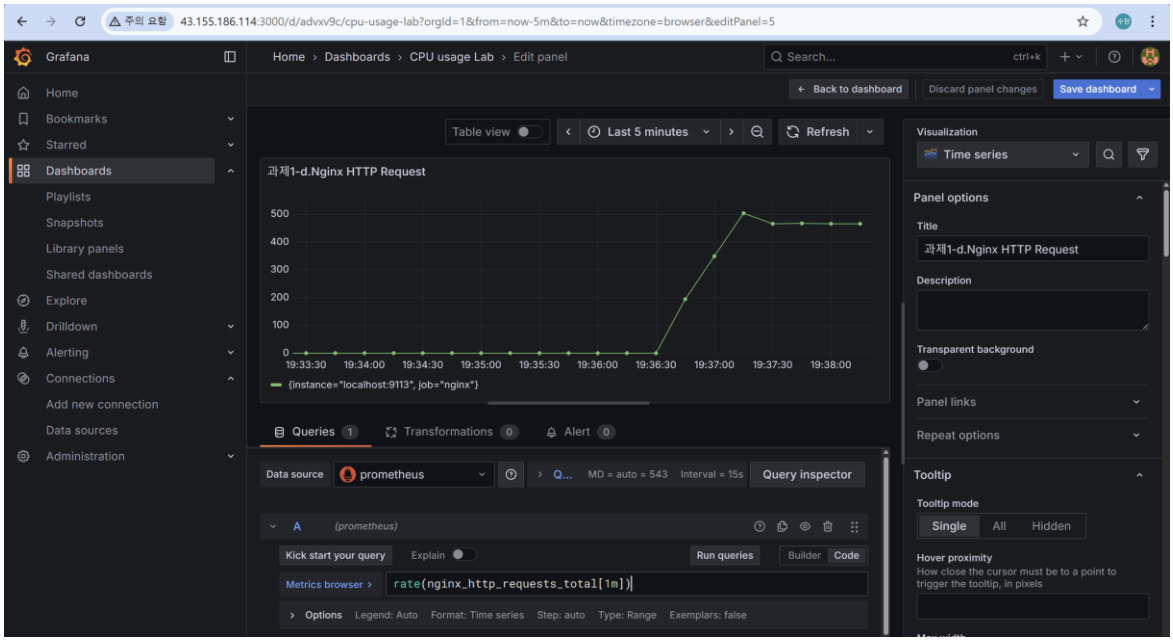
이미지)



d. Nginx HTTP Requests

쿼리) rate(nginx_http_requests_total[1m])

이미지)



Problem 2.

Briefly explain each query and its composition (2 -4 sentences per panel)

Node Exporter, Nginx Prometheus Exporter의 공식 깃허브와 curl localhost:9100/metrics등의 방법에서 존재하는 메트릭들과 그 의미를 찾아볼 수 있었습니다. 1-a의 요구사항에 rate window=1m으로 되어있어 모든 rate()함수는 1m로 하였습니다.

https://github.com/nginx/nginx-prometheus-exporter

a. CPU usage

(1 - avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[1m])))*100

Node_cpu_seconds_total은 CPU가 일하지 않는 idle한 상태의 누적 시간입니다. Rate를 이용해 1m의 초당 평균 idle한 비율을 구하였습니다. Normalized across all cores라고 요구하였으므로 avg by (instance)를 사용하여 모든 코어 값을 서버별로 평균 냅니다. 1-idle한 상태의 비율 = cpu 사용율 이므로 이렇게 1에서 비율을 빼주고 *100을 하여 %로 맞춰줍니다.

(추가) 여기서 avg와 avg by (instance) 두개를 고민했는데, avg by (instance)를 해야 서버별로 하나의 평균 CPU usage를 보여주기 때문에 avg by (instance)를 사용했다. 다만 어차피 지금 서버가 1개이기 때문에 avg만 사용해도 동일한 결과가 나올 것으로 생각된다.

b. Memory usage (%)

(1-(node_memory_MemAvailable_bytes/node_memory_MemTotal_bytes))*100

사용가능한 메모리인 node_memory_MemAvailable_bytes와 시스템의 총 메모리를 나타내는 node_memory_MemTotal_bytes를 이용하여 사용 가능한 메모리의 비율을 구하고 1에서 빼주어 사용 중인 메모리의 비율을 구했습니다. 마찬가지로 100을 곱해주어 %로 만들어주었습니다.

(추가) Node exporter에서 사용중인 메모리의 양을 나타내는 메트릭은 여러가지이다. Node_memory_Buffers_bytes, node_memory_Cached_bytes등이 있지만 buffer와 cache를 직접 더해서 구하는 것은 애매하기 때문에 정확하게 제공되는 memAvailable을 사용하여 전체에서 빼는 것이 더 정확하다.

```
node_memory_Bounce_bytes 0
# HELP node_memory_Buffers_bytes Memory information field Buffers_bytes.
# TYPE node_memory_Buffers_bytes gauge
node_memory_Buffers_bytes 5.9211776e+07
# HELP node_memory_Cached_bytes Memory information field Cached_bytes.
# TYPE node_memory_Cached_bytes gauge
node_memory_Cached_bytes 8.65738752e+08
# HELP node_memory_CommitLimit_bytes Memory information field CommitLimit_bytes.
# TYPE node_memory_CommitLimit_bytes gauge
```

```
node_memory_MemAvailable_bytes 1.490055168e+09
# HELP node_memory_MemFree_bytes Memory information field MemFree_bytes.
# TYPE node_memory_MemFree_bytes gauge
node_memory_MemFree_bytes 6.91326976e+08
# HELP node_memory_MemTotal_bytes Memory information field MemTotal_bytes.
# TYPE node_memory_MemTotal_bytes gauge
node_memory_MemTotal_bytes 2.063347712e+09
# HELP node_memory_Mlocked_bytes Memory information field Mlocked_bytes.
# TYPE node_memory_Mlocked_bytes gauge
```

c. Nginx accepted connections

rate(nginx_connections_accepted[1m])

nginx_connections_accepted은 Nginx의 시작 이후 지금까지의 누적된 총 연결된 수를 나타내는 counter이다. 이 값에 rate를 적용하여 1분을 기준으로 초당 평균 신규 연결 수가 몇 개인지 계산합니다.

Name	Type	Description	Labels
`nginx_connections_accepted`	Counter	Accepted client connections.	[]

d. Nginx HTTP Requests

rate(nginx_http_requests_total[1m])

nginx_http_requests_total은 마찬가지로 Nginx가 시작 이후 지금까지 처리한 총 HTTP 요청 수를 나타냅니다. Rate를 적용하여 1분 윈도우 동안 초당 평균 HTTP 요청 수를 계산합니다.

`nginx_http_requests_total`	Counter	Total http requests.	[]

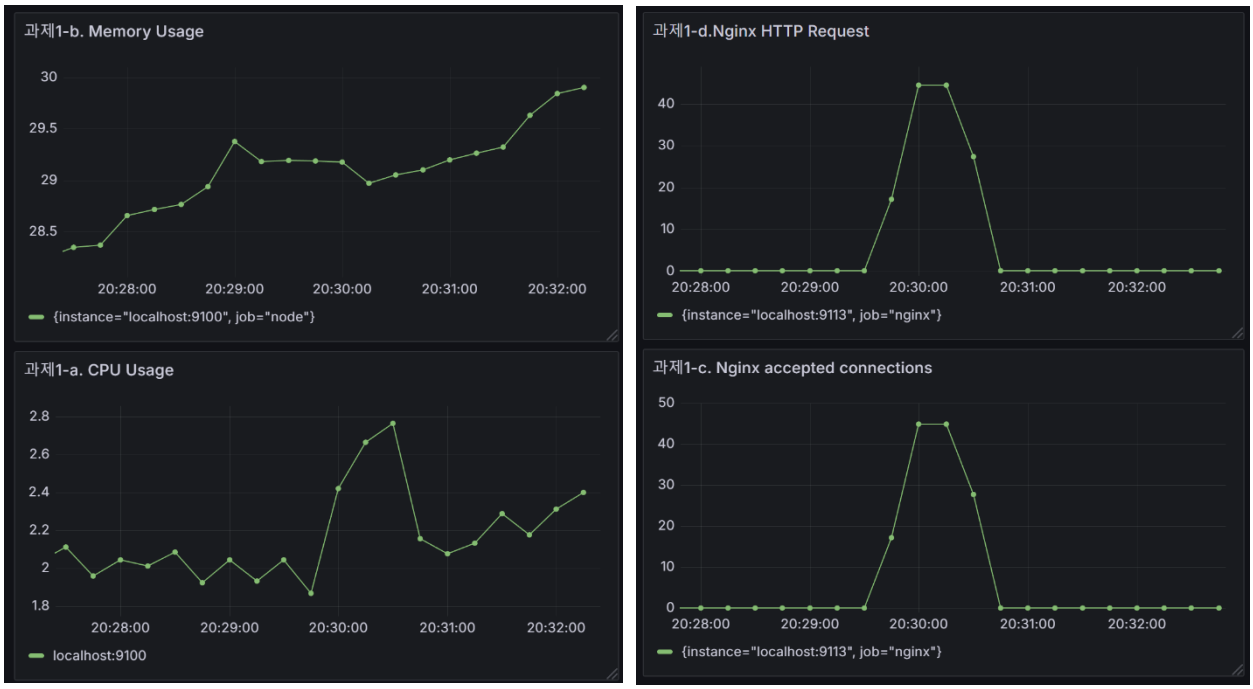
Problem 3.

Simulate the following ApacheBench loads and explain your results -include 'CPU usage' graph screenshots:

스크린캡쳐 순서가 b-Memory Usage | d-Nginx HTTP Request

a-CPU Usage | c-Nginx accepted connections 순서입니다.

a. Small load: ab -n 2000 -c 100



- 작은 부하 전 -> 작은 부하 후

  A의 CPU Usage는 2-2.2 사이 -> 2.8 까지 올랐다가 내려감

  B의 Memory Usage는 그냥 올라가는 추세의 지속

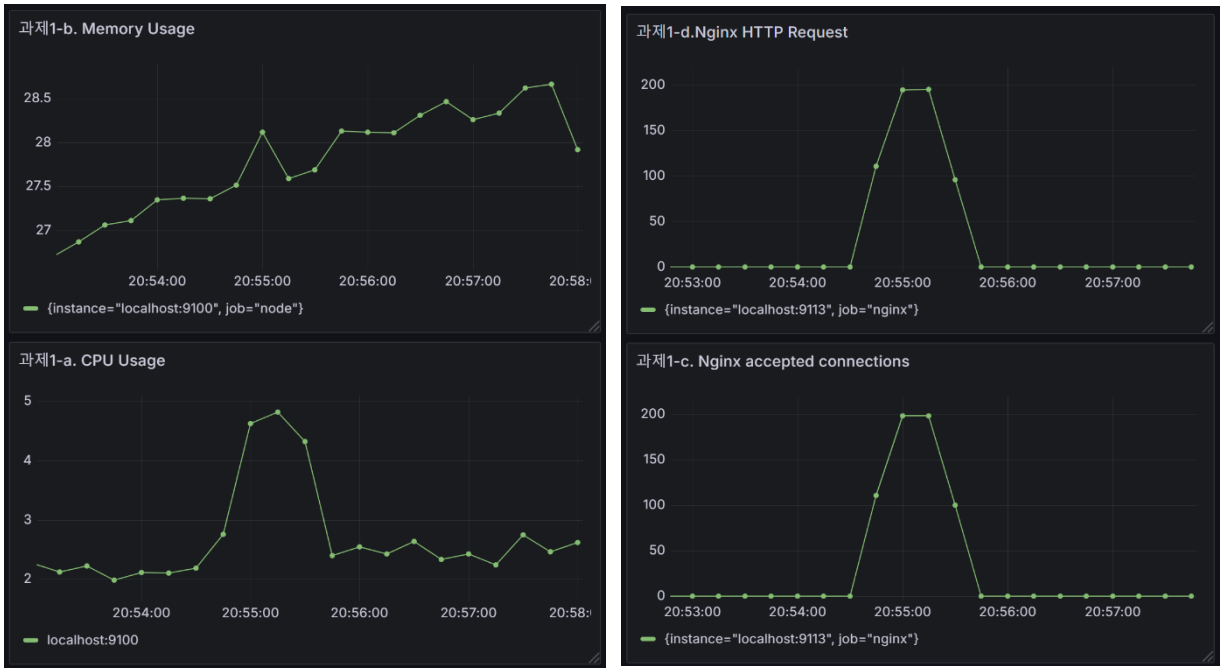
  C의 Nginx accepted connections는 0 -> 40넘는 값까지 올랐다가 내려감

  D의 Nginx HTTP Request는 0 -> 40-50사이까지 올랐다가 내려감

- 해석

  C와 d의 결과를 보면 작은 부하를 잘 감지한 것으로 볼 수 있다. 다만 CPU Usage는 2에서 2.8정도까지 올랐다 내려갔음을 보아 부하가 그렇게 크지 않았으며, memory Usage는 크게 영향 받지 않고 올라가는 추세의 지속인 것으로 보아 small load라서 cpu부하도 0.8정도 증가한 것에서 그쳤고, 특히 메모리 사용량에는 아무런 영향을 주지 못했음을 볼 수 있다. Small load정도의 부하는 서버에 큰 부담이 되지 않음을 숫자와 그래프로 확인할 수 있다.

b. Medium load: ab -n 10000 -c 500



- 중간 부하 전 -> 중간 부하 후

  A의 CPU Usage는 2.2 -> 5까지 올라갔다 내려감

  B의 Memory Usage는 영향 없음

  C의 Nginx accepted connections는 0 -> 200정도까지 올라갔다 내려감

  D의 Nginx HTTP Request는 0 -> 200정도까지 올라갔다 내려감

- 해석

  마찬가지로 c와 d의 결과를 보면 중간 부하를 잘 감지하였음을 알 수 있다. 이전의 작은 부하는 0 -> 4-50 정도의 스파이크를 찍었는데 중간 부하는 0->200의 스파이크를 찍은 것을 확인할 수 있고, CPU Usage도 더 큰 사용률(5정도)을 보였다. 중간 부하가 작은 부하에 비해 5배 정도 강했는데 (실제로 2000,100 -> 10000, 500) 그 효과가 증명되었다. 마찬가지로 Memory Usage에는 영향이 없었다.

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

- 큰 부하 전 -> 큰 부하 후

  A의 CPU Usage는 0 -> 6.5 이상 올라갔다 내려감

  B의 Memory Usage가 영향을 주지 않음 (다른 a,c,d는 21:03:00 – 21:06:30 정도까지 영향이 있었지만 b는 21:04:30 정도에서 갑자기 치솟은 것으로 보아 우연이라고 판단하였다)

  C의 Nginx accepted connections는 0->450~500 까지 올라갔다 내려감

  D의 Nginx HTTP Request는 0 -> 450~500까지 올라갔다 내려감

- 해석

  C와 d에서는 0->450~500정도의 큰 스파이크를, a의 CPU Usage에서는 2->6.5의 큰 스파이크를 볼 수 있다. 이전의 두개보다 더 큰 부하였고, 그래서 더 큰 CPU 사용량과 Nginx connections, requests값이 나왔다. 이번에는 이전 작은 부하, 중간 부하와 다르게 memory Usage에 영향을 끼친 것으로 보인다.

최종 비교

Small load, medium load, large load의 결과를 살펴본 결과, 부하가 커질수록 a의 CPU Usage, c와 d의 Nginx connections, requestes 값들이 정비례하여 커짐을 확인할 수 있다.

추가 : 또한 Memory Usage에는 영향을 주지 않고, CPU 자원만을 사용했음을 볼 수 있다. 명령어인 ab -n <total 개수> -c <한번에 몇개> 는 웹 서버에 부하를 주는 명령어인데 이는 메모리를 쓸 일이 없는 단순한 작업으로, nginx페이지에 연결 수락, 요청 해석, 작은 파일 찾기, 응답 전송의 간단한 작업이다. 또한 그 파일 크기가 크지 않았기 때문에 메모리까지 쓸 필요가 없다. 메모리 사용량이 급증하는 경우는 대용량 파일을 처리하거나, 복잡한 웹페이지를 생성하는 등의 복잡한 작업을 할 때이다.

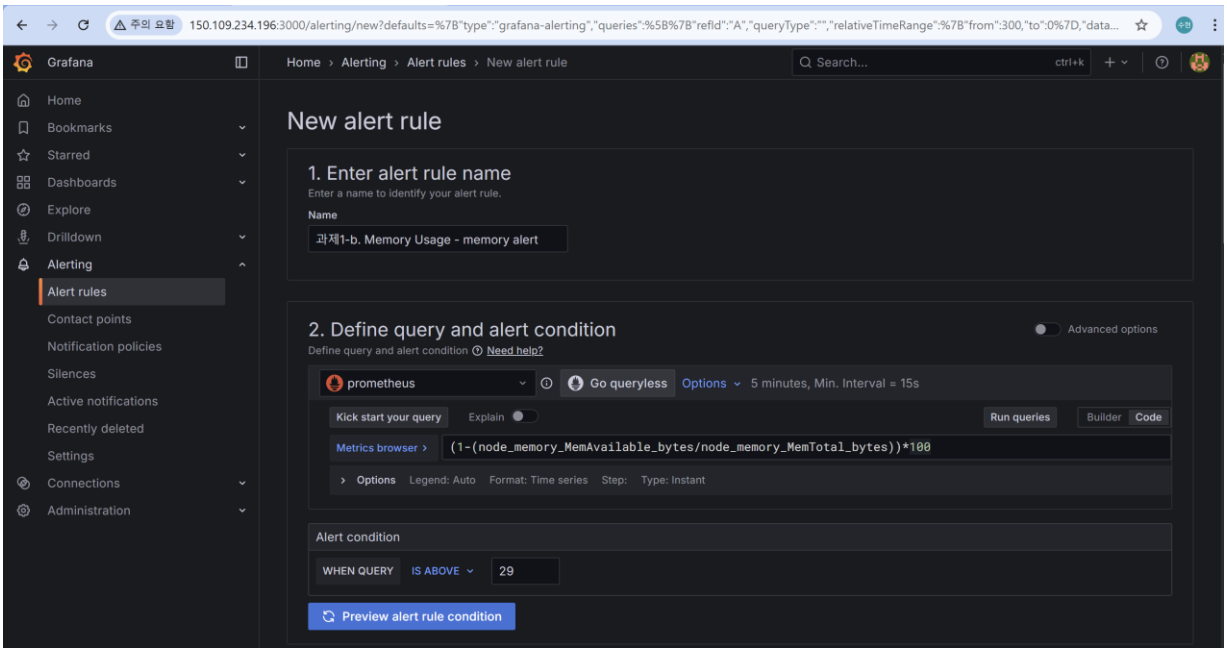
```
Server Software:        nginx/1.24.0
Server Hostname:        150.109.234.196
Server Port:            80

Document Path:          /
Document Length:        615 bytes
```

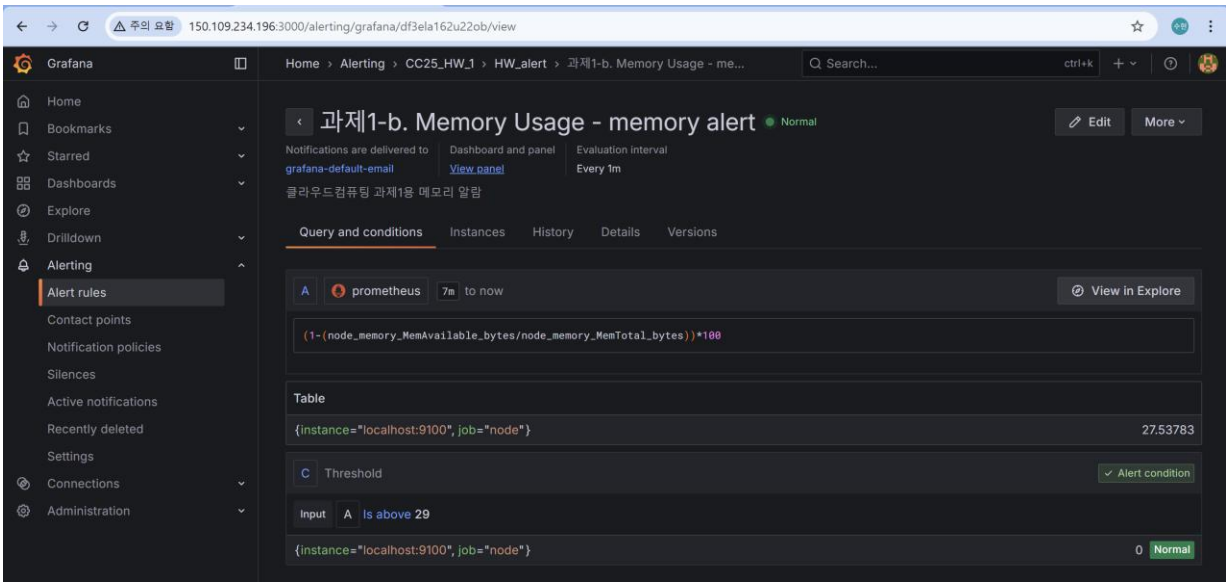
(출력 결과에서 615 bytes를 확인할 수 있다)

Problem 4.

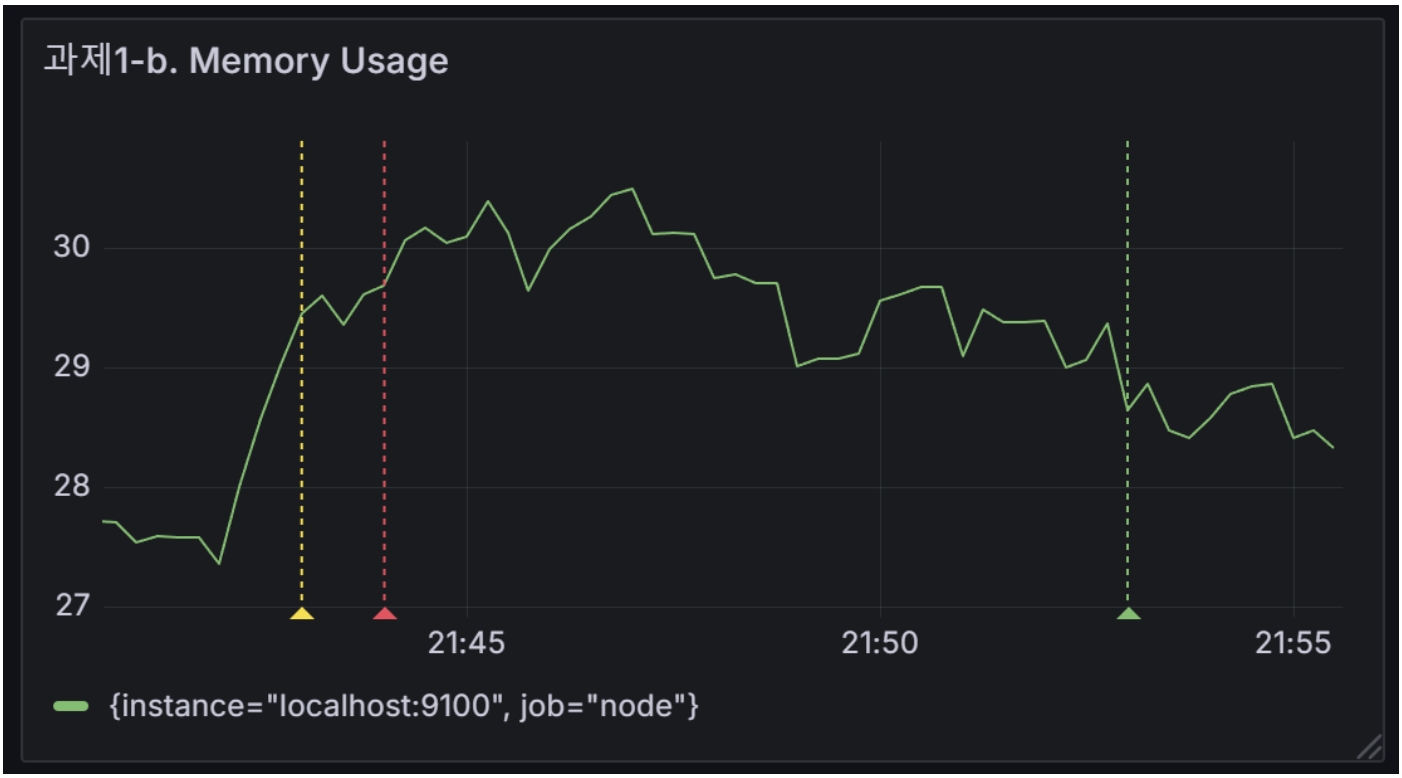
Create an alarm rule for the 'Memory (MB)' query and show it being triggered -include 'define query and alert condition' setting and alert triggered graph screenshots

1) Query와 알람 조건 스크린샷





2) alert triggered graph screenshots



Where query is above 29로 알람 조건을 설정하였기 때문에 29가 넘었을 때 pending상태(노란색)가 되고 (문제에 딱히 설명이 없어 실습 ppt에 있는 1m으로 하였습니다) pending시간인 1분이 지났음에도 메모리 사용량이 29를 넘어있기 때문에 알람이 발생(빨간색)하였다. 그 이후에 메모리 사용량이 29보다 낮아지면 알람이 해제(초록색)된다.