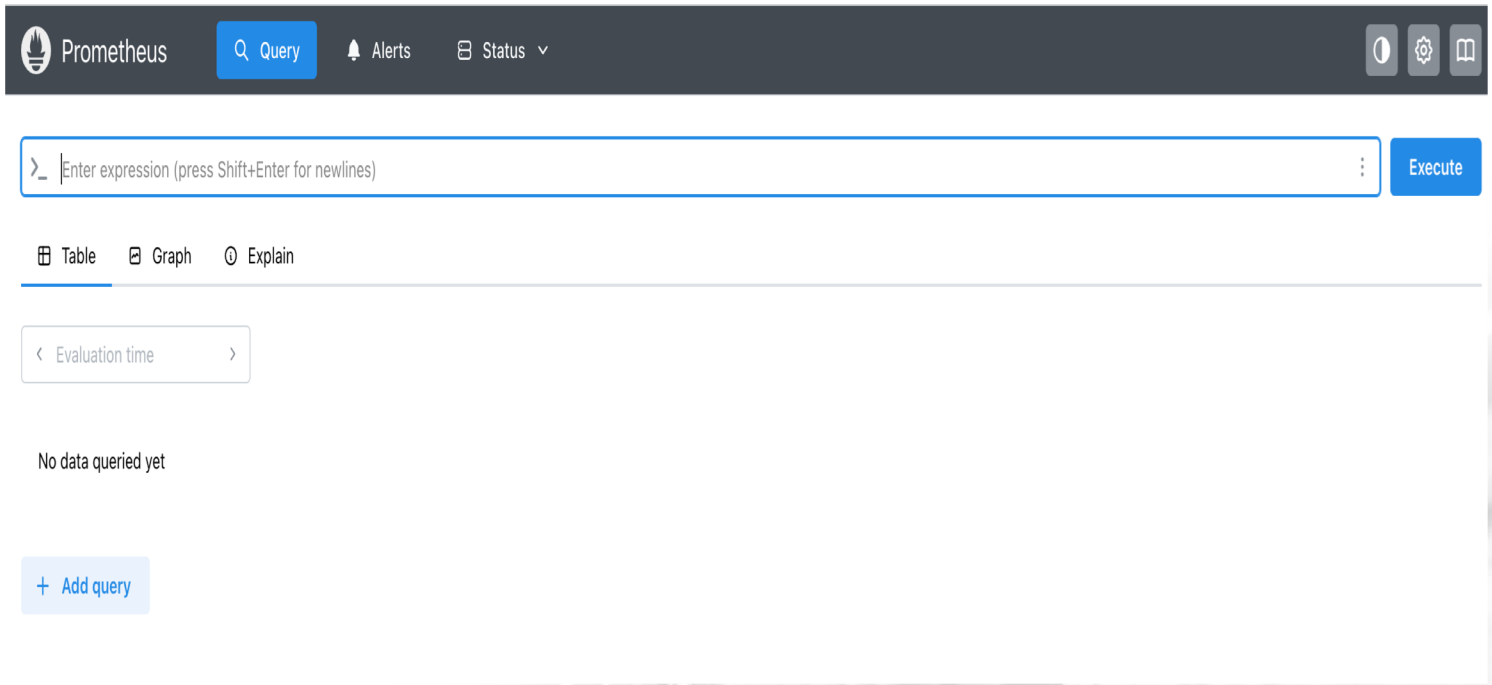
Prometheus Server

Prometheus_Server

- Prometheus Web UI
  - ✓ Query는 프로메테우스의 웹 UI에서 제공하는 가장 중요한 내용을 처리하는 페이지

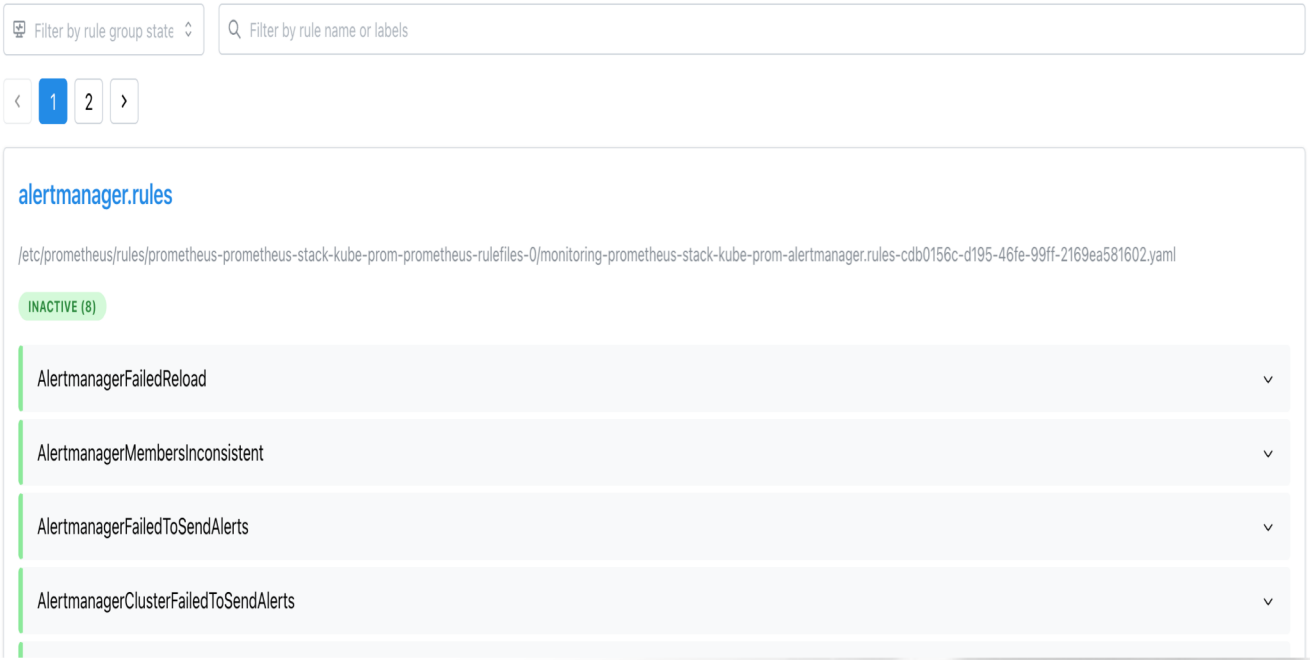


Prometheus_Server

- Prometheus Web UI
  - ✓ Query 메뉴
    - 쿼리 입력기
      - 프로메테우스가 적재한 메트릭 데이터를 조회할 수 있는 표현식(Expression)을 입력하는 곳
      - 사용하는 표현식은 PromQL(Prometheus Query Language)이라는 프로메테우스에서 제공하는 쿼리 언어
      - 프로메테우스는 시계열 데이터베이스를 사용하므로 다른 RDBMS처럼 쿼리문을 작성해 효과적으로 필요한 메트릭을 추출
      - PromQL 문법에 맞는 표현식을 쿼리 입력기에 입력한 후에 Enter를 누르거나 Execute 버튼을 눌러 필요한 메트릭 값을 가지고 옴
      - PromQL은 프로메테우스가 수집한 메트릭을 표현하는 가장 중요한 문법
    - Execute: 쿼리 입력기에 입력한 PromQL을 실행하는 버튼으로 PromQL 표현식에 맞는 메트릭 데이터를 화면에 출력
    - Graph: PromQL로 프로메테우스가 적재한 메트릭 데이터를 확인할 때 시각적으로 표현해주는 옵션으로 Graph 탭을 누르면 데이터를 시각화한 영역형 또는 막대형 차트를 출력

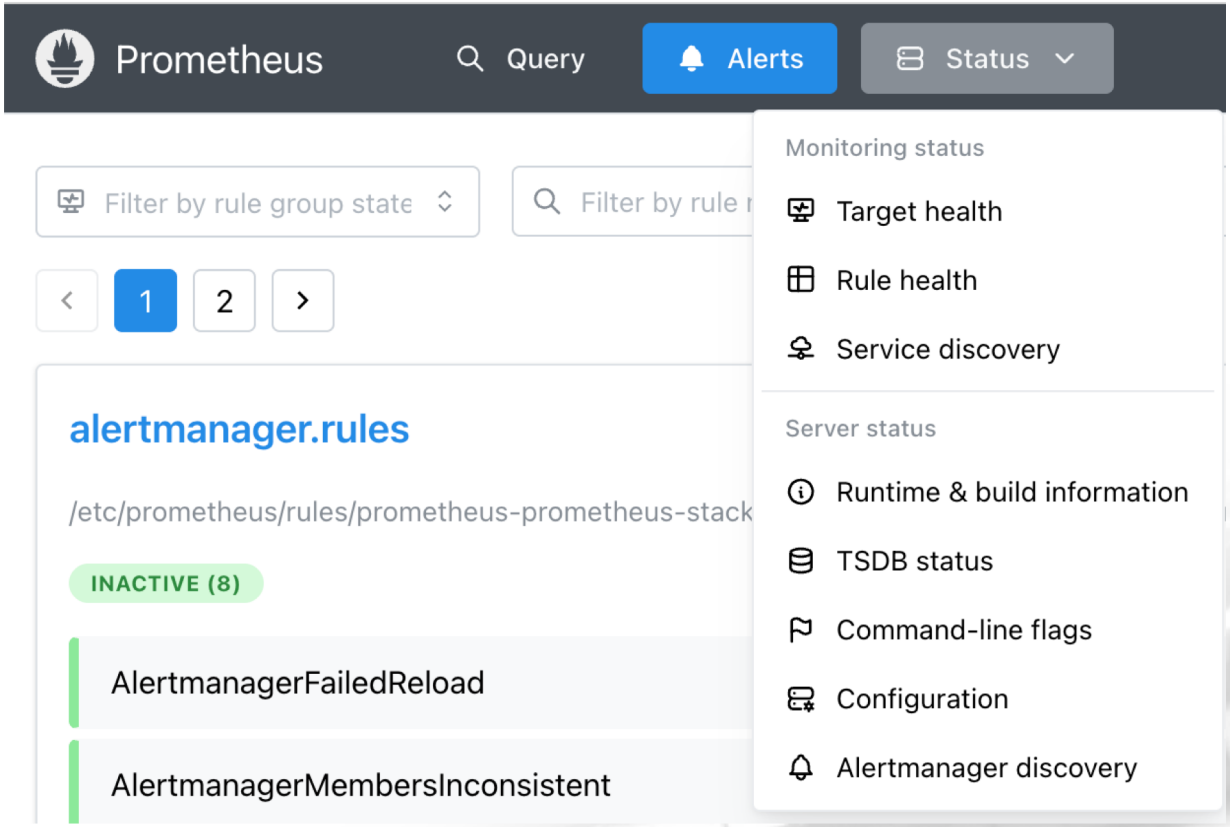
Prometheus_Server

- Prometheus Web UI
  - ✓ Alert(경보)은 상단 왼쪽에 있는데 경보 화면에서는 현재 프로메테우스 서버에 등록된 경보 규칙과 경보 발생 여부를 확인할 수 있음



Prometheus_Server

- Prometheus Web UI

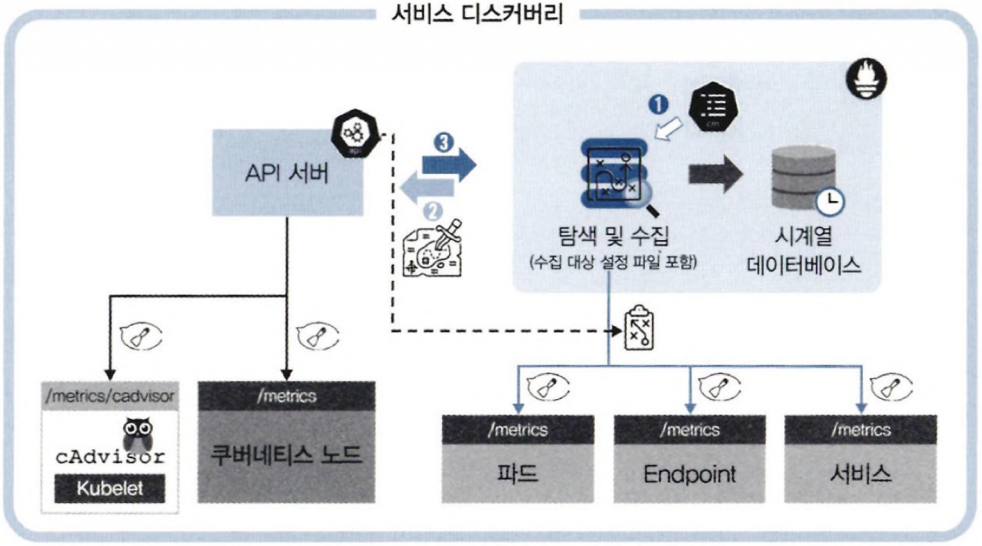


프로메테우스 서버 활용

- Prometheus Web UI
  - ✓ Status 하위 메뉴
    - Runtime & Build Information: 프로메테우스 서버의 업타임 같은 런타임 관련 정보, 버전을 타내는 빌드 정보 등 여러 정보를 확인할 수 있음
    - Command-Line Flags: 프로메테우스 서버가 실행될 때 인자로 입력받았던 값을 보여줌
    - Configuration: 프로메테우스 서버가 구동될 때 설정된 값을 표시
    - Targets health: 프로메테우스 서버가 수집해오는 대상의 상태를 확인할 수 있음
    - Service Discovery: 프로메테우스 서버가 Service Discovery 방식으로 수집한 대상들에 대한 정보를 요약해 보여 줌

Prometheus_Server

- ❖ Service Discovery
  - ✓ 개요
    - 프로메테우스는 수집 대상을 자동으로 인식하고 필요한 정보를 수집
    - 정보를 수집하려면 일반적으로 에이전트를 설정해야 하지만 쿠버네티스는 사용자가 에이전트에 추가로 입력할 필요없이 자동으로 메트릭을 수집할 수 있음
    - 프로메테우스 서버가 수집 대상을 가져오는 방법인 Service Discovery 덕분
  - ✓ Service Discovery 동작 순서
    - 프로메테우스 서버는 컨피그맵에 기록된 내용을 바탕으로 대상을 읽어옴
    - 읽어온 대상에 대한 메트릭을 가져오기 위해 API 서버에 정보를 요청
    - 요청을 통해 알아온 경로로 메트릭 데이터를 수집



Prometheus_Server

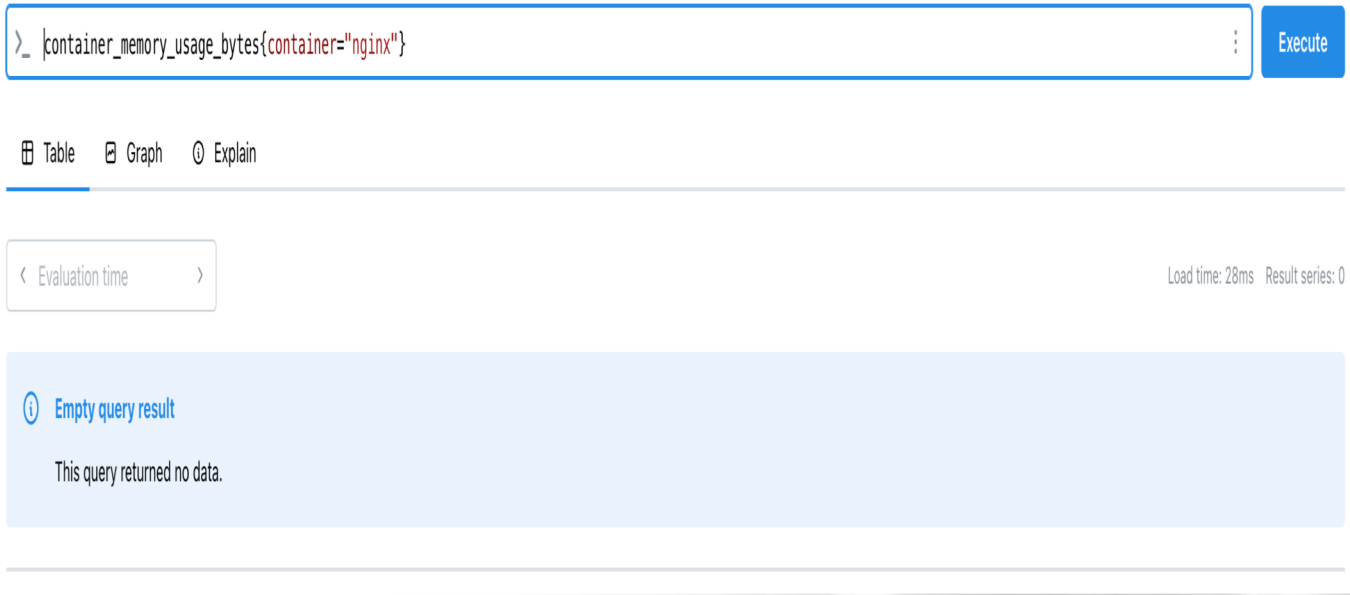
- Service Discovery
  - ✓ 이와 같은 순서로 프로메테우스 서버와 API 서버가 주기적으로 데이터를 주고받아 수집 대상을 업데이트하고 수집 대상에서 공개되는 메트릭을 자동으로 수집
  - ✓ Service Discovery 방법은 대상에 따라 크게 2가지 경로로 나누는데 쿠버네티스 API 서버에 직접 연결돼 메트릭을 수집하는 cAdvisor과 API 서버가 경로를 알려 주어 메트릭을 수집할 수 있는 에이전트

Prometheus_Server

- Service Discovery
  - ✓ cAdvisor로 메트릭 수집하고 확인
    - 프로메테우스 웹 UI로 가서 Advisor로 수집된 메트릭은 container라는 이름으로 시작하므로 쿼리 입력기에 container_memory_usage_bytes를 입력하고 Execute 버튼을 누름
    - 배포: kubectl create deployment nginx --image=nginx
    - 쿼리 검색기를 수정: container_memory_usage_bytes{container="nginx"}

Prometheus_Server

- ❖ Service Discovery
  - ✓ cAdvisor로 메트릭 수집하고 확인
    - 삭제: kubectl delete deployment nginx
    - 확인



Prometheus_Server

- Service Discovery
  - ✓ cAdvisor로 메트릭 수집하고 확인
    - 삭제: kubectl delete deployment nginx
    - 확인

Prometheus_Server

- Service Discovery
  - ✓ Exporter로 수집
    - 사전 준비 작업
      - API 서버에서 등록돼 경로를 알 수 있게 해야 함
      - 익스포터가 데이터를 프로메테우스 타입으로 노출해야 함
    - 애너테이션으로 메트릭 수집하기
      - 애너테이션은 일반적으로 소스 코드에 주석으로 추가되는 메타데이터로 여러 애플리케이션이나 다른 도구에 정보를 전달하는 역할을 수행
      - 프로메테우스는 애너테이션을 이용해 수집 대상을 판별하고 경로를 재조합하는데 애너테이션으로 경로를 노출하고 메트릭을 수집하려면 다음과 같이 설정해야 함
        - 프로메테우스에 메트릭을 공개하려는 애플리케이션은 애너테이션이 매니페스트에 추가돼 있어야 하며 애너테이션에 추가되는 구문은 prometheus.io/로 시작
        - 작성된 매니메스트를 쿠버네티스 클러스터에 배포해 애너테이션을 포함한 정보를 API 서버에 등록
        - 프로메테우스 서버가 prometheus.io/로 시작하는 애너테이션 정보를 기준으로 대상의 주소를 생성
        - 프로메테우스 서버가 애너테이션을 기준으로 만든 대상의 주소로 요청을 보내 메트릭 데이터를 수집

Prometheus_Server

- ❖ Service Discovery
  - ✓ Exporter로 수집
    - ● 애플리케이션 매니페스트 생성: nginx-status-annot.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: monitoring
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        prometheus.io/port: "9113"
        prometheus.io/scrape: "true"
```

Prometheus_Server

- ❖ Service Discovery
  - ✓ Exporter로 수집
    - ● 애플리케이션 매니페스트 생성: nginx-status-annot.yaml

```
spec:
  containers:
  - name: nginx
    image: sysnet4admin/nginx-status
    ports:
    - containerPort: 80
  - name: nginx-exporter
    image: nginx/nginx-prometheus-exporter:0.8.0
    ports:
    - containerPort: 9113
      name: metrics
    name: metrics
    env:
    - name: SCRAPE_URI
      value: http://localhost:80/stub_status
```

Prometheus_Server

- ❖ Service Discovery
  - ✓ Exporter로 수집
    - 배포: kubectl apply -f nginx-status-annot.yaml
    - 확인: kubectl get pods

```
lertmanager-prometheus-stack-kube-prom-alertmanager-0   2/2     Running   0
14h
nginx-69cf857c8b-srv85                                  2/2     Running   0          31m
prometheus-prometheus-stack-kube-prom-prometheus-0      2/2     Running   0
14h
prometheus-stack-grafana-dcb957796-86m79                3/3     Running   0
14h
prometheus-stack-kube-prom-operator-85cb656d59-ht79z    1/1     Running   0
14h
prometheus-stack-kube-state-metrics-75c97578f5-qglnk    1/1     Running   0
14h
prometheus-stack-prometheus-node-exporter-dwzhr         1/1     Running   0
14h
prometheus-stack-prometheus-node-exporter-gr28q         1/1     Running   0
14h
prometheus-stack-prometheus-node-exporter-m24vr         1/1     Running   0
14h
```

Prometheus_Server

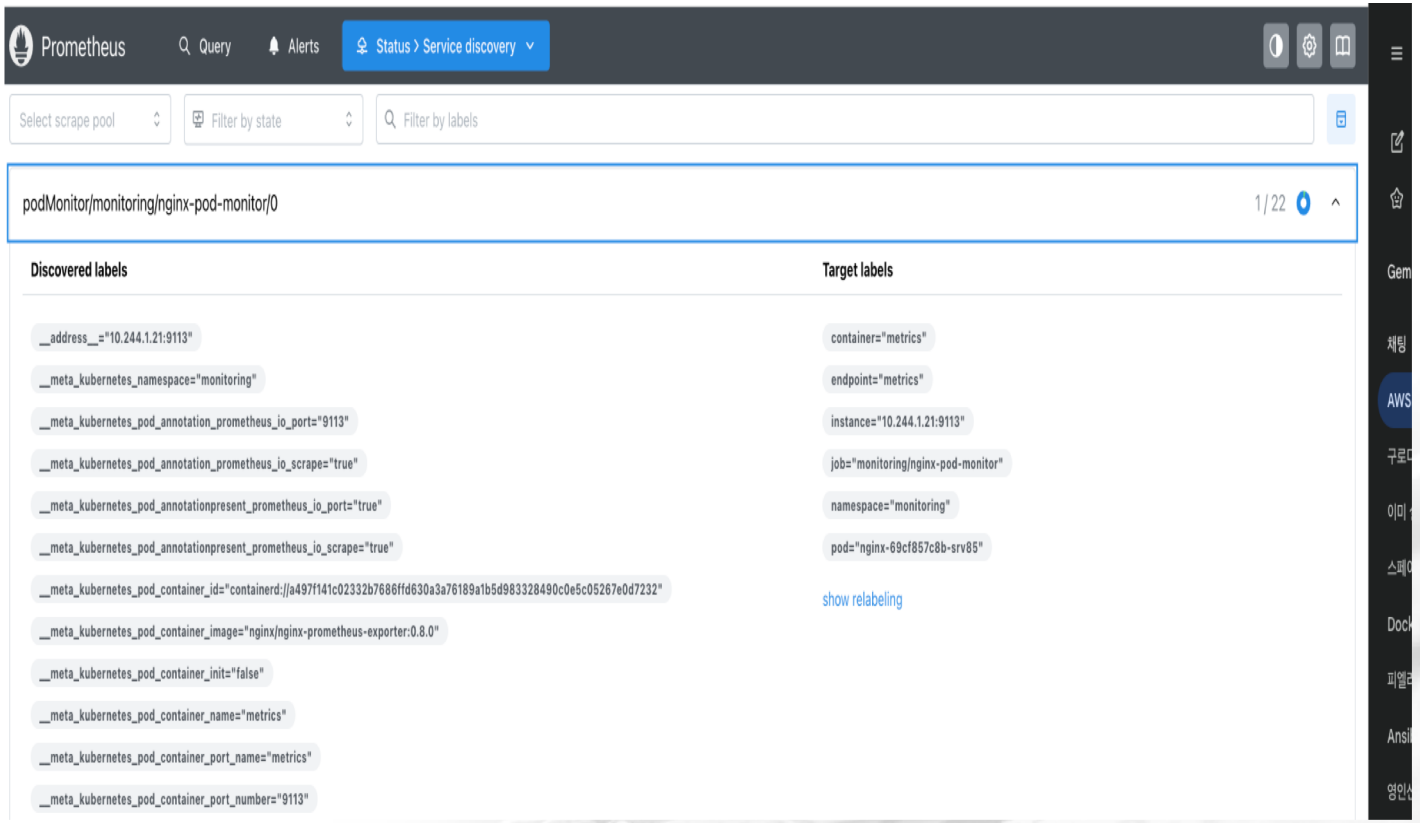
- ❖ Service Discovery
  - ✓ Exporter로 수집
    - ● 파드를 모니터링할 PodMonitor 매니페스트: podmonitor.yaml

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: nginx-pod-monitor
  namespace: monitoring
  labels:
    release: prometheus-stack # 확인하신 리리스 이름 입력
spec:
  selector:
    matchLabels:
      app: nginx # Deployment의 spec.template.metadata.labels와 일치해야 함
  podMetricsEndpoints:
  - port: metrics # Deployment에서 설정한 포트 이름
    interval: 30s
```

    - ● 배포: kubectl apply -f podmonitor.yaml

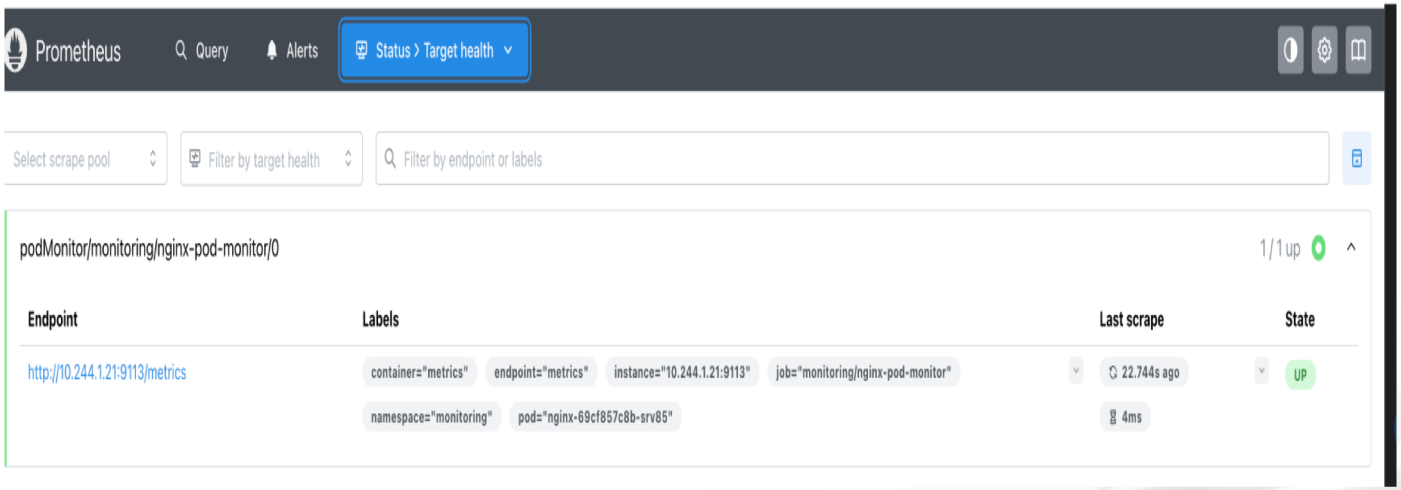
Prometheus_Server

- ❖ Service Discovery
  - ✓ Exporter로 수집
    - ● 서비스 디스커버러리 확인



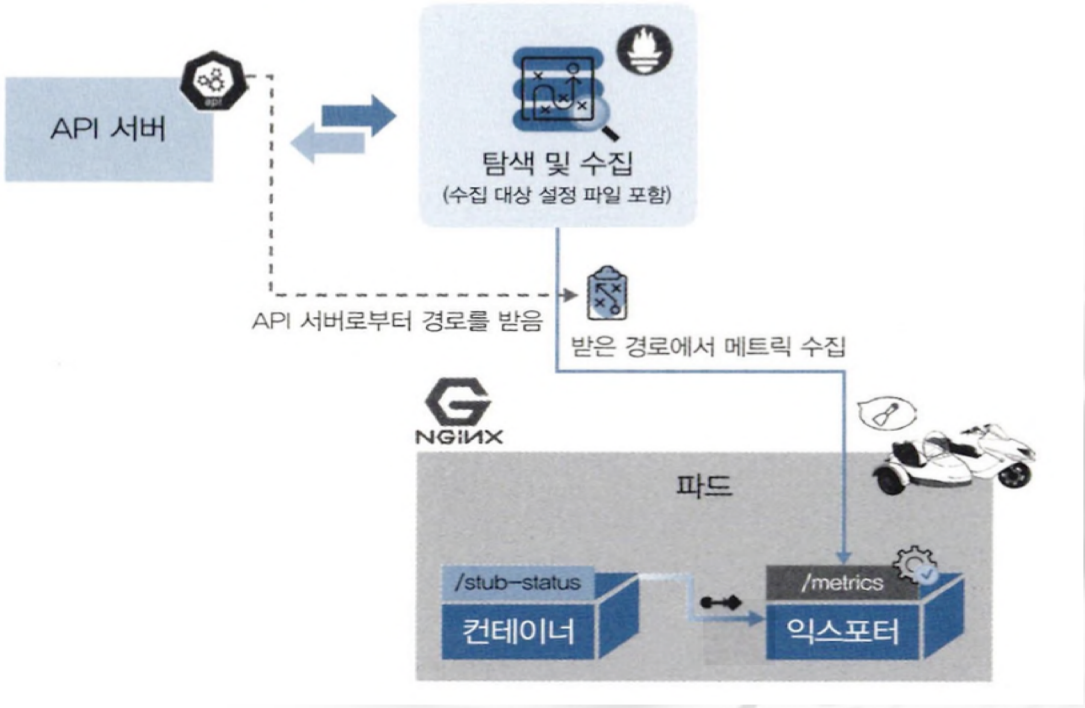
Prometheus_Server

- ❖ Service Discovery
  - ✓ Exporter로 수집
    - targethealth



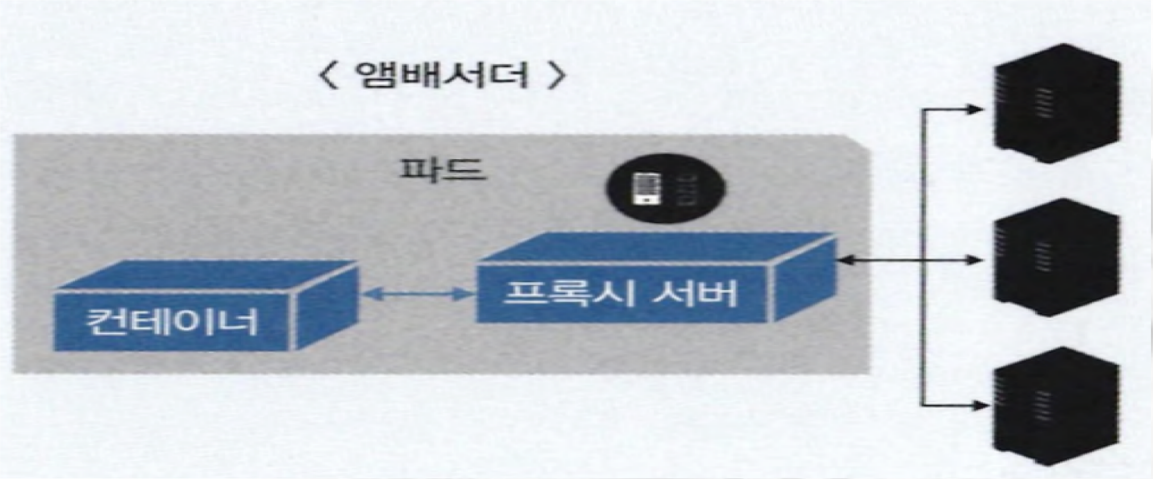
Prometheus_Server

- Service Discovery
  - ✓ Exporter로 수집
    - 사이드카를 추가해 메트릭 수집: nginx-prometheus-exporter는 멀티 컨테이너 패턴 중에 하나인 사이드카 패턴으로 작성되어 있는데 사이드카 외에 여러 가지의 패턴으로 파드의 멀티 컨테이너를 구성할 수 있는데 사이드카 패턴을 적용하면 구조가 변경됨



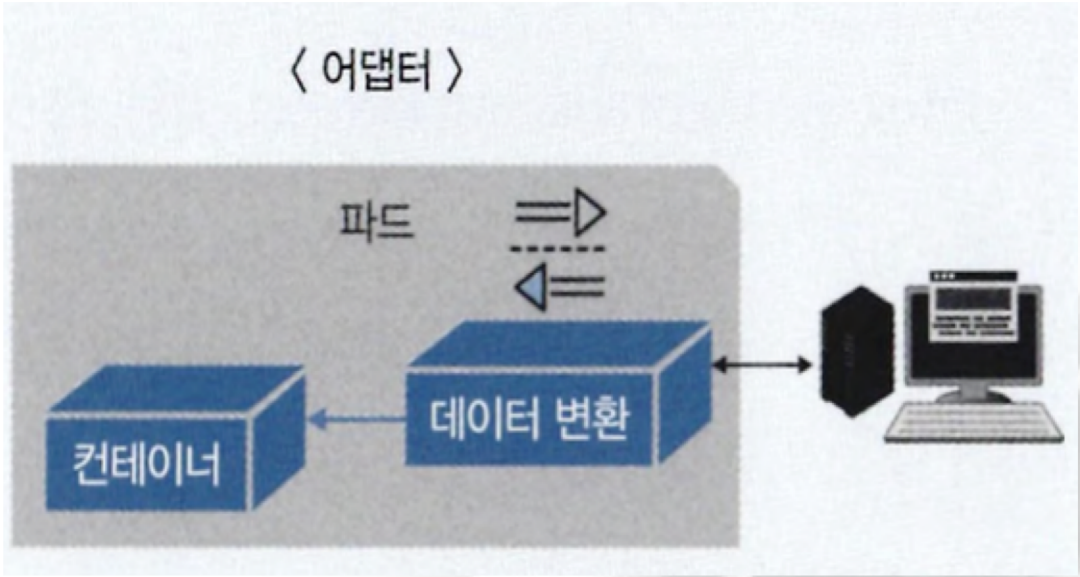
Prometheus_Server

- ❖ Service Discovery
  - ✓ Exporter로 수집
    - ● 멀티 컨테이너 패턴
      - ◆ 사이드카(Sidecar): 메인 컨테이너의 기능을 확장하거나 기능을 향상하고자 할 때 추가하는 패턴
      - ◆ 앰배서더(Ambassador): 사용자가 외부에서 접근할 때 앰배서더 컨테이너를 통해 통신이 이루어지는 형태로 세부적인 외부 접근 대상이나 방법은 앰배서더 컨테이너에 위임하는 방식인데 주로 프록시 컨테이너를 구성해 외부의 접근을 제어하고 내부 자원을 보호하는 구성이 앰배서더 패턴에 속함



Prometheus_Server

- ❖ Service Discovery
  - ✓ Exporter로 수집
    - ● 멀티 컨테이너 패턴
      - ◆ 어댑터(Adapter): 메인 컨테이너의 정보를 외부에서 사용할 수 있는 형식으로 변환하는 컨테이너가 추가된 형태인데 데이터 형태를 변환하므로 NGINX의 익스포터는 어댑터 패턴이라고도 할 수 있음



Prometheus_Server

- Service Discovery
  - ✓ Exporter로 수집
    - ● 리소스 정리
      - ◆ kubectl delete -f podmonitor.yaml
      - ◆ kubectl delete -f nginx-status-annot.yaml

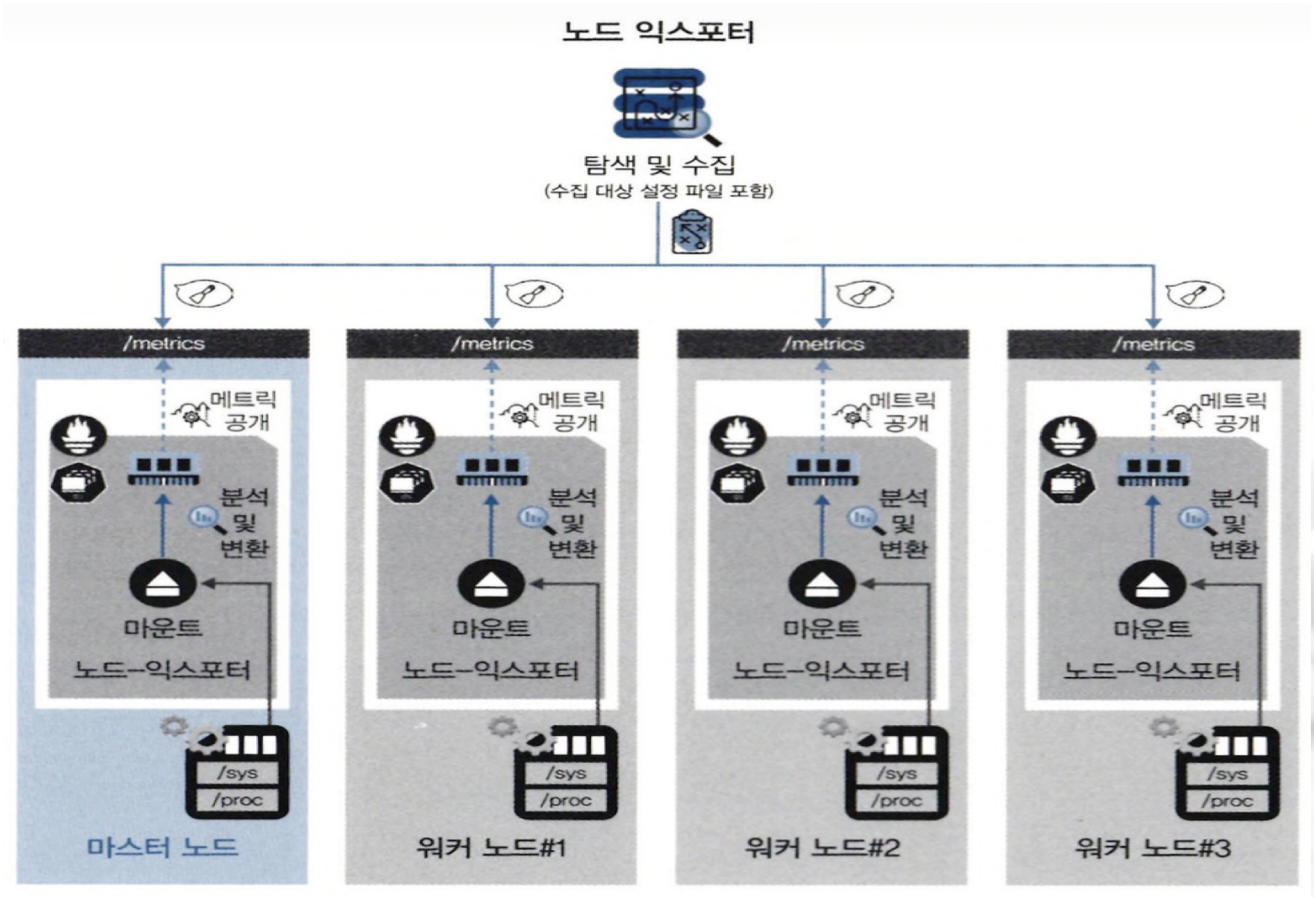
Prometheus_Server

- ❖ Service Discovery
  - ✓ Exporter로 수집
    - ● 다양한 종류의 프로메테우스 익스포터

분야	데이터베이스	메시징 시스템	스토리지	외부 모니터링 시스템
주요 관심사	안정성 지표 내부 구조	처리량 전달 성공율	사용량	외부 시스템 통합
대표적 수집 대상	ElasticSearch MongoDB MySQL Oracle PostgreSQL	Kafka NATS MQTT RabbitMQ	Ceph Gluster GPFS NetApp E-Series	AWS CloudWatch Google Stackdriver (Operation Suite) Azure Monitor JMX

Prometheus_Server

- Node Exporter를 이용한 Node Metric 수집
  - ✓ Node Exporter 동작 원리



Prometheus_Server

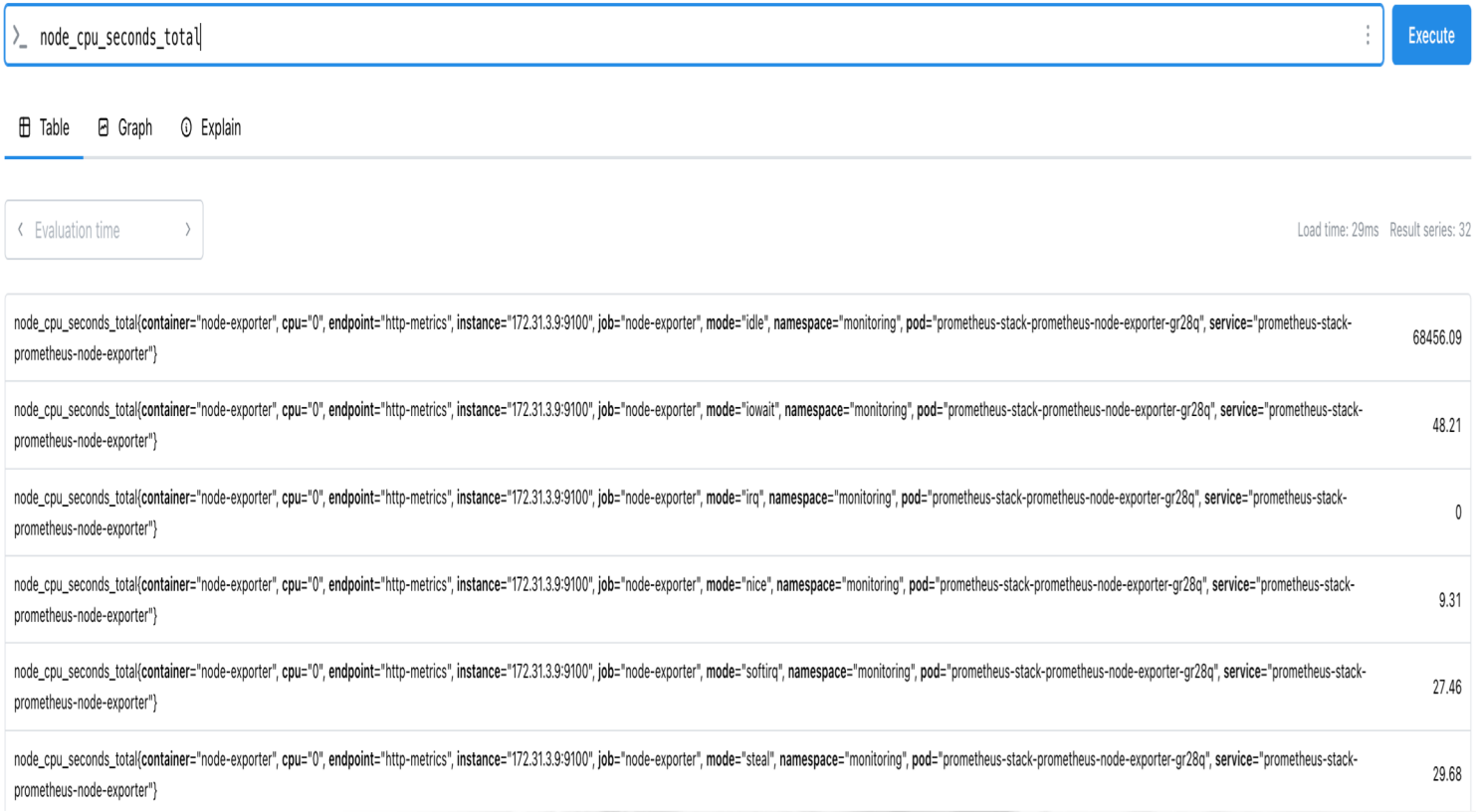
- ❖ Node Exporter를 이용한 Node Metric 수집
  - ✓ Node Exporter 동작 원리
    - 노드 익스포터는 노드의 /proc과 /sys에 있는 값을 메트릭으로 노출하도록 구현되어 있음
    - 노드 익스포터를 실행하기 위해서 쿠버네티스 노드마다 1개씩 배포할 수 있는 데몬셋으로 배포
    - 노드 익스포터가 배포하는 핵심 메트릭인 /proc과 /sys에는 다음 정보들이 저장
      - ◆ /proc
        - 리눅스 운영 체제에서 구동 중인 프로세스들의 정보를 파일 시스템 형태로 연결한 디렉터리로 현재 구동 중인 프로세스들의 프로세스 ID가 디렉터리 형태로 나타나며 각 디렉터리 내부에 해당 프로세스에 대한 상태나 실행 환경에 대한 정보가 들어 있음
        - 모니터링 도구는 이 디렉터리를 통해 시스템에서 구동 중인 프로세스의 정보나 상태, 자원 사용량 등을 알 수 있음
      - ◆ /sys
        - 저장 장치, 네트워크 장치, 입출력 장치 같은 각종 장치를 운영 체제에서 사용할 수 있도록 파일 시스템 형태로 연결한 디렉터리로 기존에 /proc에 연결돼 있던 커널 장치 드라이버 등이 /proc에서 /sys로 분리됨
        - 디렉터리 내부에 있는 파일들을 분석하면 전체 장치의 상태를 알 수 있기 때문에 모니터링 도구는 이 디렉터리를 시스템 장치의 상태 정보를 수집하는 데 사용

Prometheus_Server

- ❖ Node Exporter를 이용한 Node Metric 수집
  - ✓ Node Exporter 동작 원리
    - 노드 익스포터는 노드의 /proc과 /sys에 있는 값을 메트릭으로 노출하도록 구현되어 있음
    - 노드 익스포터를 실행하기 위해서 쿠버네티스 노드마다 1개씩 배포할 수 있는 데몬셋으로 배포
    - 노드 익스포터가 배포하는 핵심 메트릭인 /proc과 /sys에는 다음 정보들이 저장
      - ◆ /proc
        - 리눅스 운영 체제에서 구동 중인 프로세스들의 정보를 파일 시스템 형태로 연결한 디렉터리로 현재 구동 중인 프로세스들의 프로세스 ID가 디렉터리 형태로 나타나며 각 디렉터리 내부에 해당 프로세스에 대한 상태나 실행 환경에 대한 정보가 들어 있음
        - 모니터링 도구는 이 디렉터리를 통해 시스템에서 구동 중인 프로세스의 정보나 상태, 자원 사용량 등을 알 수 있음
      - ◆ /sys
        - 저장 장치, 네트워크 장치, 입출력 장치 같은 각종 장치를 운영 체제에서 사용할 수 있도록 파일 시스템 형태로 연결한 디렉터리로 기존에 /proc에 연결돼 있던 커널 장치 드라이버 등이 /proc에서 /sys로 분리됨
        - 디렉터리 내부에 있는 파일들을 분석하면 전체 장치의 상태를 알 수 있기 때문에 모니터링 도구는 이 디렉터리를 시스템 장치의 상태 정보를 수집하는 데 사용

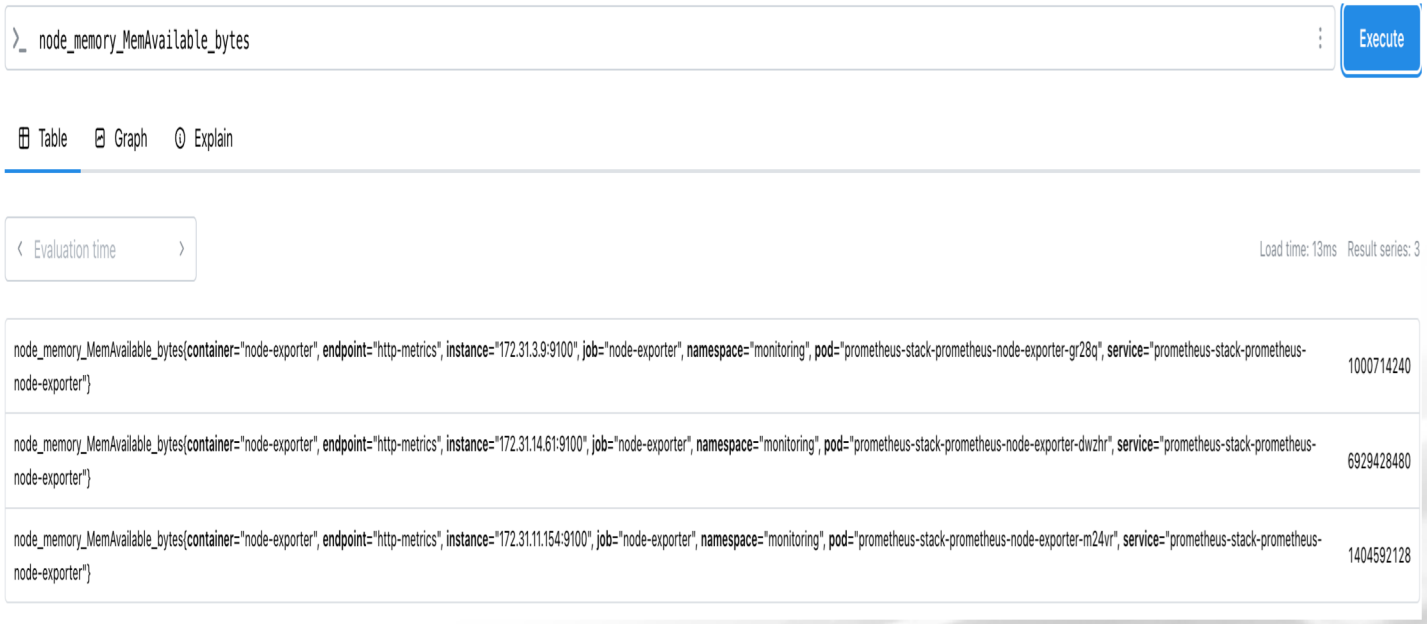
Prometheus_Server

- ❖ Node Exporter를 이용한 Node Metric 수집
  - ✓ Node Exporter로 수집된 메트릭 확인
    - CPU 상태 별로 사용된 시간을 조회: node_cpu_seconds_total



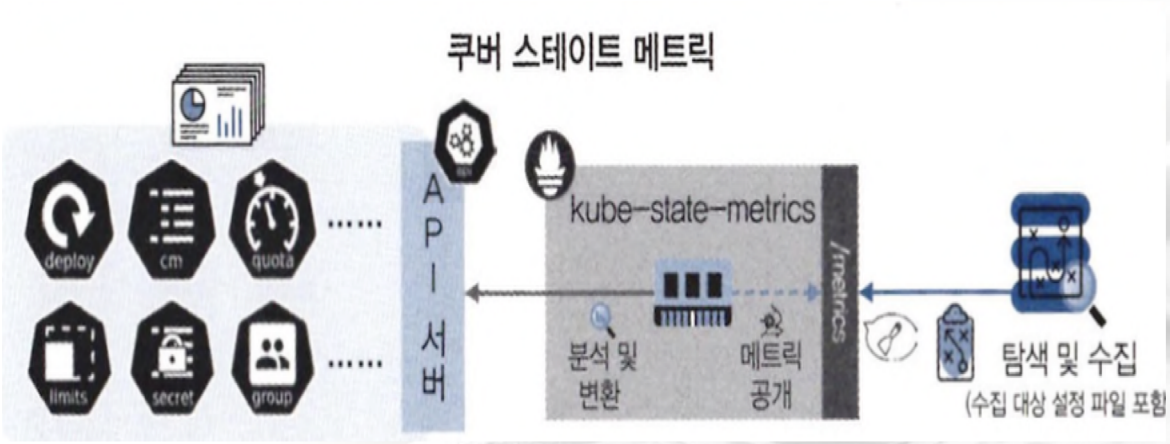
Prometheus_Server

- ❖ Node Exporter를 이용한 Node Metric 수집
  - ✓ Node Exporter로 수집된 메트릭 확인
    - ● 현재 사용 가능한 메모리의 용량 조회: node_memory_MemAvailable_bytes



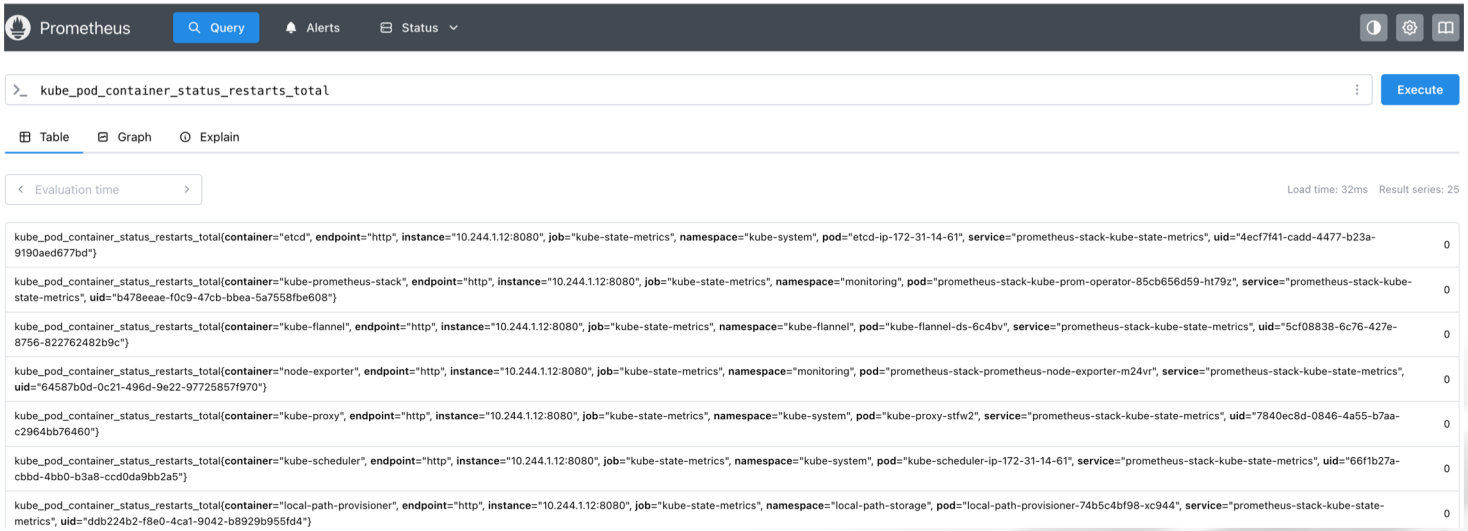
Prometheus_Server

- kube-state-metrics을 이용한 Cluster Metric 수집
  - ✓ 개요
    - kube-state-metrics은 이름에서 알 수 있듯이 쿠버네티스 상태를 보여주는 메트릭
    - 쿠버네티스 상태에 관한 정보를 가지고 와서 프로메테우스가 받아들일 수 있는 메트릭 형태로 변환하고 변환한 메트릭을 HTTP 서버를 통해 /metrics 주소로 공개하는 부분은 노드 익스포터와 동일
    - 각 노드의 특정 디렉터리를 마운트해서 사용하지 않고 이미 API 서버에 모인 값들을 수집한다는 점이 다름
    - kube-state-metrics 작동 방식



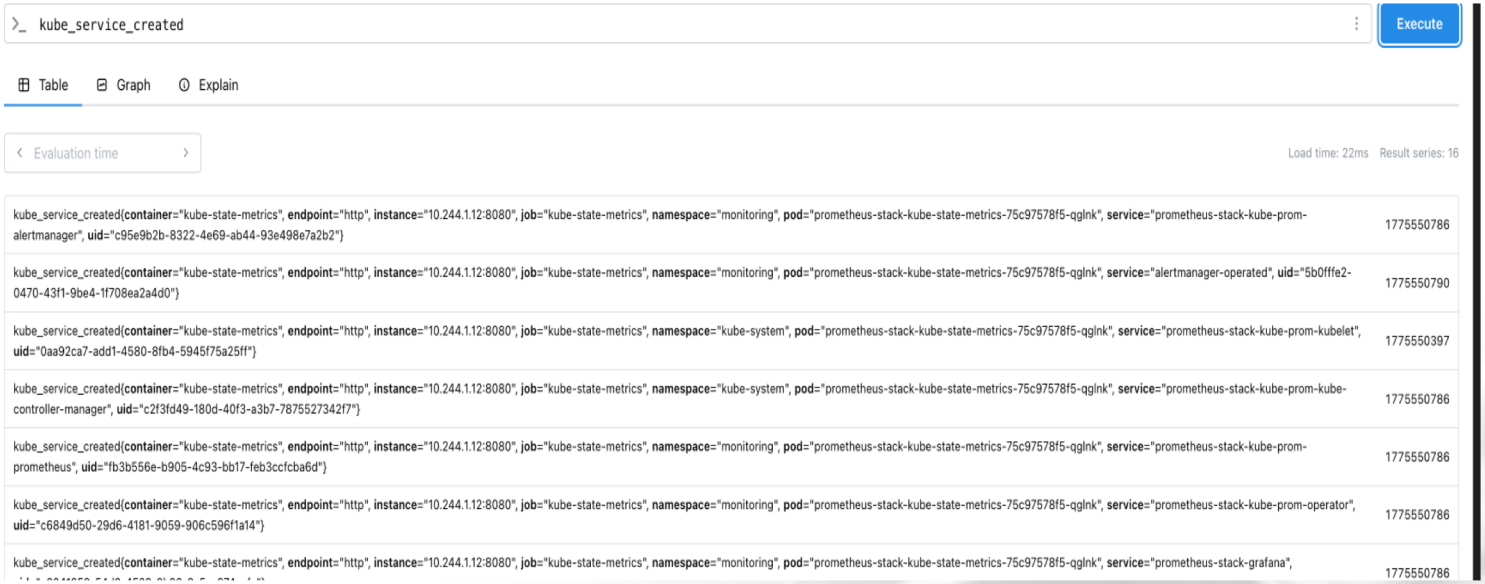
Prometheus_Server

- ❖ kube-state-metrics을 이용한 Cluster Metric 수집
  - ✓ kube-state-metrics으로 수집된 데이터 확인
    - 쿠버네티스에 배포된 파드가 다시 시작하는 경우 이를 누적해 기록한 메트릭 데이터: kube_pod_container_status_restarts_total



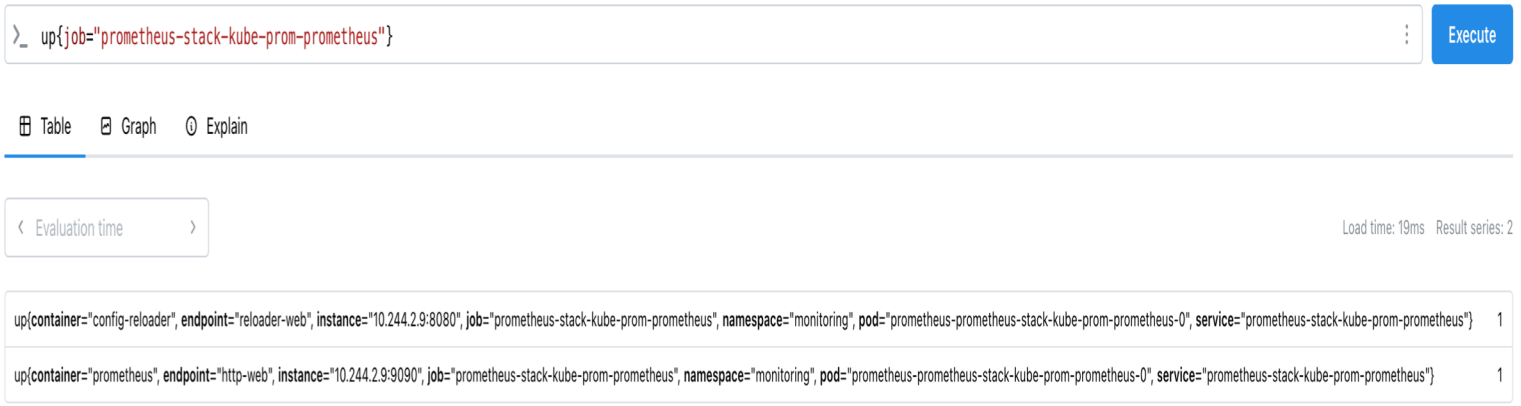
Prometheus_Server

- ❖ kube-state-metrics을 이용한 Cluster Metric 수집
  - ✓ kube-state-metrics으로 수집된 데이터 확인
    - 쿠버네티스에 존재하는 서비스들이 생성된 시간 확인(Unix Time): kube_service_created



Prometheus_Server

- ❖ PromQL로 메트릭 데이터 추출
  - ✓ 메트릭 데이터 구조
    - Web UI에서 쿼리 입력기에 up{job="prometheus-stack-kube-prom-prometheus"} 를 입력: 결과로 나온 메트릭 데이터를 보면 up이라는 메트릭 이름을 가지는 대상을 검색하고 그 중에 job="prometheus-stack-kube-prom-prometheus"라는 레이블 이름을 검색 조건에 추가하는데 이를 일반적으로 필터링이라고 하고 검색 조건에 맞는 메트릭 데이터가 존재하고 추출에 성공하면 1이라는 값으로 표현



Prometheus_Server

- ❖ PromQL로 메트릭 데이터 추출
  - ✓ 메트릭 데이터 구조
    - ● 메트릭 값의 종류
      - ◆ node_cpu_seconds_total 과 kube_pod_container_status_restarts_total은 누적되는 메트릭 데이터 값으로 카운터(Counter)타입이라고 하며 node_memory_MemAvailable_bytes 와 kube_service_created는 특정 시점의 메트릭 데이터 값으로 게이지(Gauge) 타입이라고 하는데 이 두 가지 타입 외에도 히스토그램과 서머리 타입이 있음
      - ◆ Counter
        - ▪ 누적된 값을 표현하는 데 사용하는 메트릭 타입
        - ▪ 카운터에 누적된 값으로 구간별로 변화율을 파악해 해당 값이 어느 정도로 추세로 증가하는지 알 수 있음
        - ▪ 이벤트나 오류 등이 급증하는 구간을 파악하는 데 적합
        - ▪ 값이 누적되기 때문에 특정 순간의 데이터를 표현하는 데는 적합하지 않음
        - ▪ 카운터는 값을 중점적으로 보기보다 값이 얼마만큼 변했는지 변화율을 주로 확인

Prometheus_Server

- ❖ PromQL로 메트릭 데이터 추출
  - ✓ 메트릭 데이터 구조
    - ● 메트릭 값의 종류
      - ◆ Gauge
        - ▪ 특정 시점의 값을 표현하는 데 사용하는 메트릭 타입
        - ▪ 카운터가 누적된 값을 표현해 증가만을 고려하는 것과 달리 게이지는 시점별로 증가나 감소를 모두 표현 가능
        - ▪ CPU 온도나 메모리 사용량 등은 누적된 값이 필요하지 않고 조회하는 순간의 값이 중요하므로 게이지 타입을 사용
      - ◆ Histogram
        - ▪ 사전에 미리 정의한 구간 안에 있는 메트릭 값의 빈도를 측정
        - ▪ 익스포터를 구현하는 단계에서 정의한 구간을 버킷(bucket)이라고 함
        - ▪ 예를 들어 히스토그램을 사용해 클라이언트가 서버로 HTTP 요청을 한 경우에 응답 시간과 맞는 버킷에 값을 추가하고 이벤트 횟수를 저장해 표시할 수 있음

Prometheus_Server

- ❖ PromQL로 메트릭 데이터 추출
  - ✓ 메트릭 데이터 구조
    - ● 메트릭 값의 종류
      - ◆ Summary
        - ▪ 히스토그램과 비슷하게 구간 내에 있는 메트릭 값의 빈도를 측정 클라이언트 요청에 따른 응답 시간을 관측하고 저장할 때 사용할 수 있지만 히스토그램과 달리 구간이 지정되는 것이 아니라 프로메테우스 자체으로 0~1 사이로 구간을 미리 정해 놓음
      - ◆ 일반적으로 메트릭 이름이 어떤 단어로 끝나느냐에 따라 메트릭 값의 타입을 추정할 수 있는데 total로 끝나면 누적한 값이므로 카운터 타입이고 bytes 또는 created라는 단어로 끝나면 해당 시점의 용량 또는 생성됨을 의미하므로 게이지 타입

Prometheus_Server

- PromQL로 메트릭 데이터 추출
  - ✓ 메트릭 데이터 구조
    - 메트릭 레이블
      - 모든 메트릭 데이터는 하나 이상의 레이블을 가짐
      - 프로메테우스의 레이블은 일반적인 주석이 아니라 메트릭 데이터의 다양한 내용을 표현하는 유일한 방법
      - 단순히 1~2개의 레이블이 아니라 제공하고 싶은 다수의 내용을 key-value 형태로 넣음
      - 이렇게 제공되는 다수의 레이블로 관리자는 원하는 레이블을 검색하고 선택적으로 추출할 수 있음
      - up{job="prometheus-stack-kube-prom-prometheus"} 메트릭 데이터에는 6개의 레이블이 있지만 up{job="kubelet"} 메트릭 데이터에는 7개의 레이블이 존재

Prometheus_Server

- ❖ PromQL로 메트릭 데이터 추출
  - ✓ 메트릭 데이터 구조
    - ● 메트릭 레이블 매처
      - ◆ 종류
        - =: 조건에 맞는 레이블 값이 같은 메트릭을 조회
        - !=: 조건에 넣은 값과 레이블 값이 다른 메트릭을 조회
        - =~: 조건에 넣은 정규 표현식에 해당하는 메트릭을 조회
        - !~: 조건에 넣은 정규 표현식에 해당하지 않은 메트릭을 조회

Prometheus_Server

- PromQL로 메트릭 데이터 추출
  - ✓ PromQL 연산자
    - 연산자 종류
      - 비교 연산자: 레이블 매처와 유사한 형태이나 비교 대상이 숫자이므로 크기를 구분하는 조건들이 있는데 사용되는 연산자는 ==, !=, >, <, <=, >=
      - 논리 연산자: 수집된 메트릭에서 보고 싶은 범위를 지정하는 and, or, unless
      - 산술 연산자: +, -, *, /, %, ^
      - 집계 연산자: 평균(avg), 합계(sum), 계수(count)와 같이 수집된 메트릭을 종합하고 분석하는 연산자로 메트릭의 값을 좀 더 의미 있는 데이터로 정리
    - 다시 시작한 적이 없는 파드만 검색: kube_pod_container_status_restarts_total > 0
    - sum(up{job="kubernetes-nodes"}) == 4 and avg(up{job-'kubernetes-nodes"}) == 1
    - node_memory_MemAvailable_bytes / 1024 / 1024 / 1024
    - avg(node_cpu_seconds_total{mode="idle"})