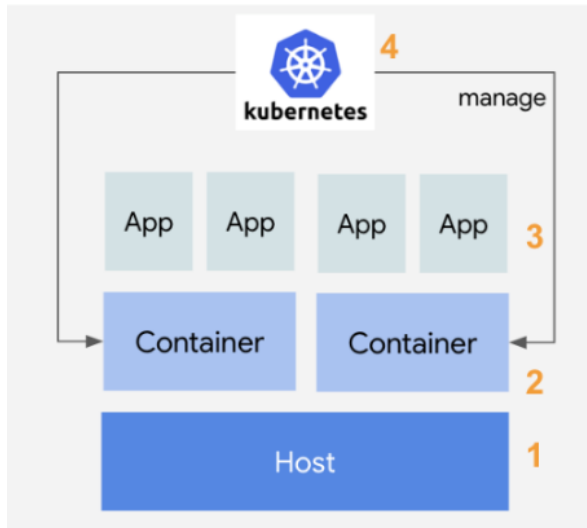


< Monitoring Concept >

K8s 클러스터에서 데이터를 수집할 수 있는 오브젝트는 크게 4가지 계층으로 볼 수 있다.



- 1) Host : 노드의 cpu, 메모리, 디스크, 네트워크, 사용량과 노드 os에 커널에 대한 모니터링
- 2) Container: 노드에 기동되는 각각의 컨테이너에 대한 정보, 컨테이너의 CPU, 메모리, 디스크, 네트워크 사용량을 모두 모니터링
- 3) Application: 컨테이너에서 구동되는 개별 애플리케이션에 대한 모니터링. Ex) 컨테이너에서 기동되는 node.js기반의 애플리케이션의 http 에러 빈도 등
- 4) Kubernetes: 컨테이너를 컨트롤 하는 쿠버네티스 자체에 대한 모니터링(쿠버네티스의 자원인 서비스, Pod, 계정 정보 등)

--> 이를 수집지표, Metric 이라고 부른다.

3가지 : 리눅스 커널 (node-exporter)

서비스, 디플로이먼트..k8s 리소스(kube-state-metric)

컨테이너 (cAdvisor)

**프로메테우스 서버(모니터링 에이전트)가 주기적으로 노드 익스포터에게 요청을 한다.
(pull 방식) 그것을 TSDB에 연동 해서 데이터를 쌓는다.**

프로메테우스는 비관계형 TSDB가 활용된다. 속도가 빠른 장점이 있고, 신뢰도가 낮다.

1. Node-exporter가 어떻게 동작하는가?

- Kube-state-metric, cadvisor 메트릭을 프록시 해주며 sum 등으로 연산을 한다.
- 사용자는 Record (rules alias 쿼리, 문법을 쉽게 치환)) Prometheus 서버에게 물어본다 -> 해당하는 expression(실제 날라가는 쿼리)을 노드 exporter에게 물어본다 -> 원시 쿼리 (kube-state-metric, cadvisor)로부터 받아온 메트릭을 object key, value 로 1차 필터링 후 sum 등으로 연산해서 반환한다.
- 노드 exporter 쿼리는 k8s 버전에 종속적이다.
- 프로메테우스 서버는 주기적으로 노드 exporter에 pull 요청을 한다.
- 리눅스 커널 메트릭을 수집한다.

K8s 의 **DaemonSet** 으로 배포되어있다.

각 노드들에 node-exporter가 설치되어 메트릭을 직접 수집한다.

2. Kube-state-metric 서버

- Kubernetes 정보 (파드, 컨트롤러, 서비스) 메트릭을 수집한다.

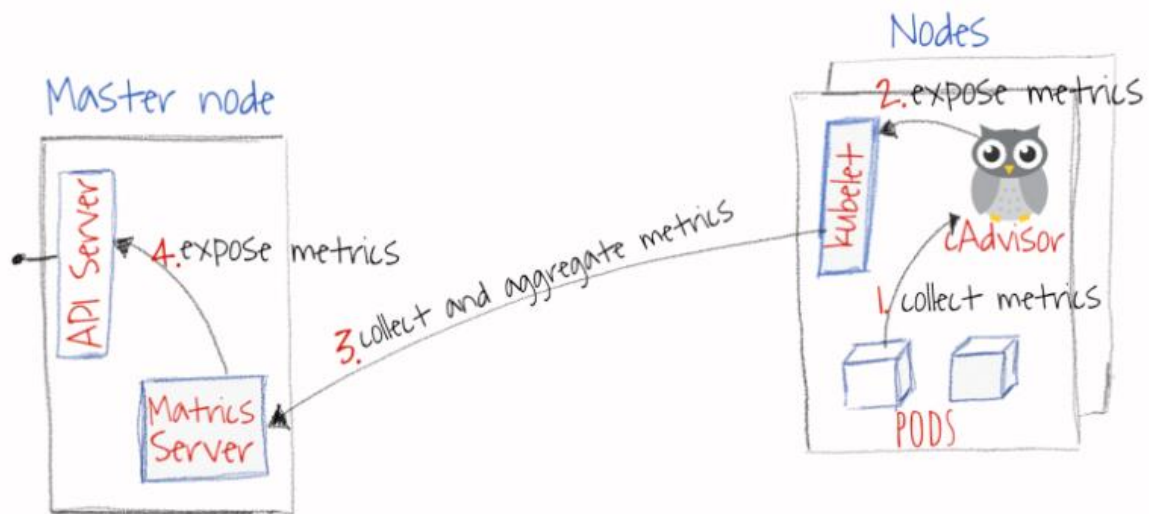
K8s가 이미 갖고 있는 데이터를 kube api 서버로부터 요청하여 가져오기 때문에 cluster 내에 단 하나의 component만 배포한다.

3. Cadvisor

- 컨테이너 감독관

kubelet에 cAdvisor라는 모니터링 에이전트가 탑재되어 있어 이를 통해서 개별 컨테이너(pod)의 메트릭 정보들을 가져 올 수 있다.

(container Advisor) 실행중인 컨테이너의 리소스 사용량 및 성능 데이터를 분석하고 노출한다.



- 1) CAdvisor는 컨테이너 및 노드에 대한 메트릭을 수집한다.
- 2) Kubelet은 kubelet API를 통해 메트릭을 expose 한다.
- 3) Metrics Server는 Kubelet API를 호출하여 컨테이너 및 노드 리소스 사용량을 가져온다.
- 4) Metrics Server는 k8s aggregation API 를 통해 메트릭을 expose 한다.

4. PROMQL

1) Instant Vector

동일 시간대에 샘플의 집합

| | | |
|--|-------|---|
| prometheus_http_requests_total{} | | Execute |
| Table | Graph | Load time: 18ms Resolution: 14s Result series: 13 |
| Evaluation time | | |
| prometheus_http_requests_total{code="200", handler="/-/ready", instance="localhost:9090", job="prometheus"} | 3 | |
| prometheus_http_requests_total{code="200", handler="/api/v1/label/name/values", instance="localhost:9090", job="prometheus"} | 4 | |
| prometheus_http_requests_total{code="200", handler="/api/v1/labels", instance="localhost:9090", job="prometheus"} | 3 | |
| prometheus_http_requests_total{code="200", handler="/api/v1/metadata", instance="localhost:9090", job="prometheus"} | 14 | |
| prometheus_http_requests_total{code="200", handler="/api/v1/query", instance="localhost:9090", job="prometheus"} | 41 | |

*쿼리 : <http://localhost:9090/api/v1/query?query={{query}}&time={{time}}>

Ex) curl '<http://172.31.10.31:31081/api/v1/query?query=up&time=2022-08-07T05:30:00.781Z>'

```
{
  "status" : "success",
  "data" : {
    "resultType" : "vector",
    "result" : [
      {
        "metric" : {
          "__name__" : "up",
          "job" : "prometheus",
          "instance" : "localhost:9090"
        },
        "value": [ 1435781451.781, "1" ]
      },
      {
        "metric" : {
          "__name__" : "up",
          "job" : "node",
          "instance" : "localhost:9100"
        },
        "value" : [ 1435781451.781, "0" ]
      }
    ]
  }
}
```

Selector

```
prometheus_http_requests_total{code=~".*"}

```

Offset

```
prometheus_http_requests_total offset 1m

```

1분 전에 데이터

aggregation operation

<https://prometheus.io/docs/prometheus/latest/querying/operators/>

```
sum(prometheus_http_requests_total{})

```

Sum, Count, min, max, avg, stddev(표준편차), stdvar(표준분산), count_values, bottomk(샘플value 중 가장 작은것), topk, quantile

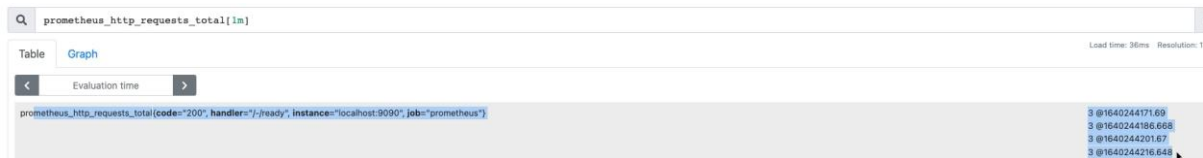
Functions

<https://prometheus.io/docs/prometheus/latest/querying/functions/>

Rate: 시계열의 초당 평균 증가율을 계산 , irate, changes, histogram_quantile, ceil, abs, sort

2) Range Vector

특정 시간 대에 여러 개의 값



* 쿼리 : start 및 end time 사이를 step 만큼 나누어 해당 시간마다 instant vector 쿼리를 날려 얻은 결과치를 뭉쳐서 받는 형태이다.

http://localhost:9090/api/v1/query_range?query={{query}}&start=2015-07-01T20:10:30.781Z&end=2015-07-01T20:11:00.781Z&step=15s

Ex)curl 'http://172.31.10.31:31081/api/v1/query_range?query=up&start=2022-08-08T05:00:00.781Z&end=2022-08-08T05:40:00.781Z&step=15s'

```
{
  "status": "success",
  "data": {
    "resultType": "matrix",
    "result": [
      {
        "metric": {
          "__name__": "up",
          "job": "prometheus",
          "instance": "localhost:9090"
        },
        "values": [
          [ 1435781430.781, "1" ],
          [ 1435781445.781, "1" ],
          [ 1435781460.781, "1" ]
        ]
      },
      {
        "metric": {
          "__name__": "up",
          "job": "node",
          "instance": "localhost:9091"
        },
        "values": [
          [ 1435781430.781, "0" ],
          [ 1435781445.781, "0" ],
          [ 1435781460.781, "1" ]
        ]
      }
    ]
  }
}
```

3) Scalar

시간 정보가 없는 값. 단순한 숫자