

Prometheus

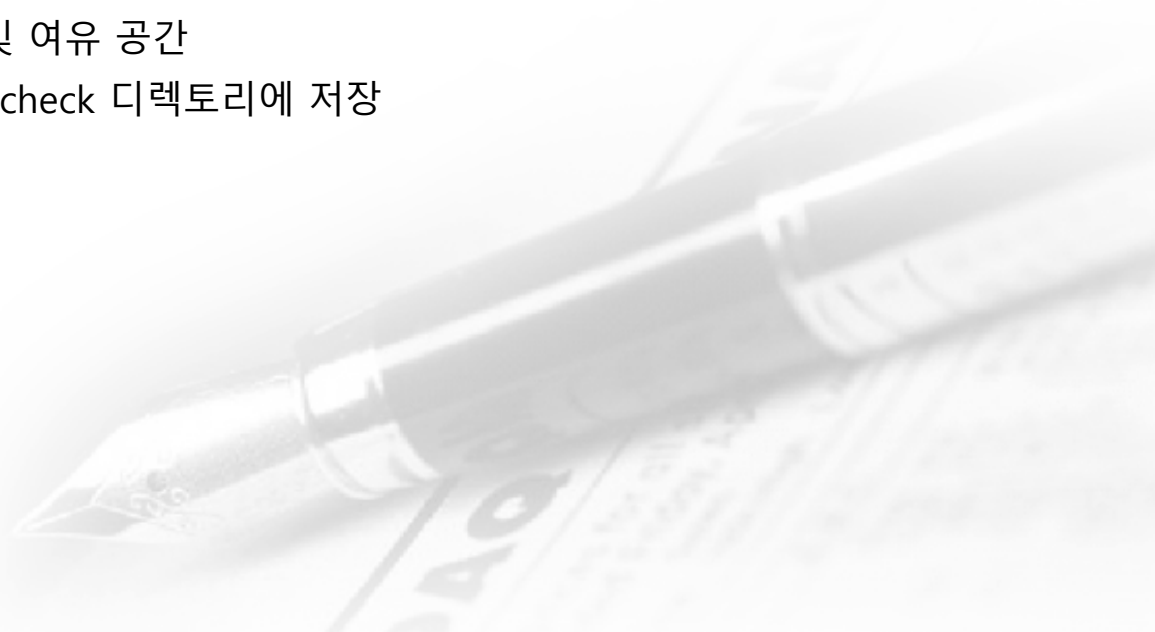
Ansible_Monitoring

❖ 팩트를 이용한 시스템 모니터링

✓ 사용을 할 facts

- 호스트 이름
- 커널 버전
- 네트워크 인터페이스 이름
- 네트워크 인터페이스 IP 주소
- 운영체제 버전
- CPU 개수
- 사용 가능한 메모리
- 스토리지 장치의 크기 및 여유 공간

✓ 추출한 내용은 /var/log/daily_check 디렉토리에 저장



Ansible_Monitoring

❖ 팩트를 이용한 시스템 모니터링

- ✓ 플레이북: monitoring_facts.yml

- hosts: all

vars:

log_directory: /var/log/daily_check

tasks:

- name: Print system info

ansible.builtin.debug:

msg:

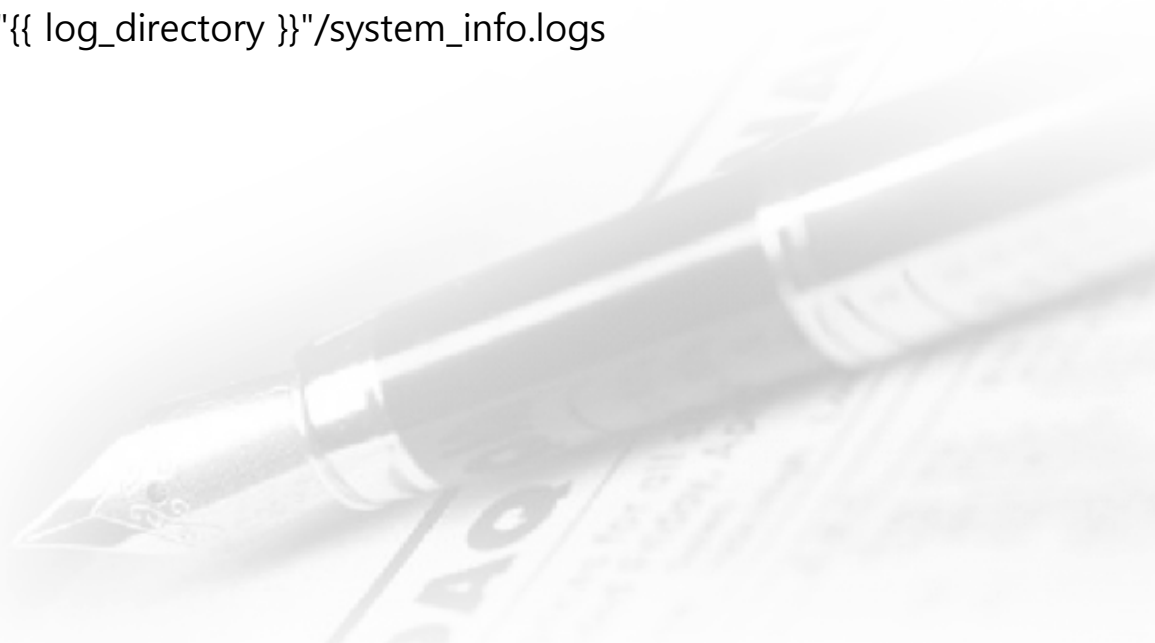
- "##### Start #####"
- "Date: {{ ansible_facts.date_time.date }} {{ ansible_facts.date_time.time }}"
- "HostName: {{ ansible_facts.hostname }}"
- "OS: {{ ansible_facts.distribution }}"
- "OS Version: {{ ansible_facts.distribution_version }}"
- "OS Kernel: {{ ansible_facts.kernel }}"
- "CPU Cores: {{ ansible_facts.processor_vcpus }}"
- "Memory: {{ ansible_facts.memory_mb.real }}"
- "Interfaces: {{ ansible_facts.interfaces }}"
- "IPv4: {{ ansible_facts.all_ipv4_addresses }}"
- "Devices: {{ ansible_facts.mounts }}"
- "##### End #####"

Ansible_Monitoring

❖ 팩트를 이용한 시스템 모니터링

✓ 플레이북: monitoring_facts.yml

```
register: result
become: yes
- name: Create log directory
  ansible.builtin.file:
    path: "{{ log_directory }}"
    state: directory
  become: yes
- name: Print logs to log file
  ansible.builtin.shell: |
    echo "{{ item }}" >> "{{ log_directory }}/system_info.logs
  loop: "{{ result.msg }}"
  become: yes
```



Ansible_Monitoring

❖ 팩트를 이용한 시스템 모니터링

- ✓ 플레이북 문법 체크: `ansible-playbook --syntax-check monitoring_facts.yml -K` register: result

playbook: monitoring_facts.yml

- ✓ 플레이북 실행 체크: `ansible-playbook --check monitoring_facts.yml -K`

PLAY RECAP

master : ok=4 changed=1 unreachable=0 failed=0 skipped=0
rescued=0 ignored=0

worker2 : ok=4 changed=1 unreachable=0 failed=0 skipped=0
rescued=0 ignored=0



Ansible_Monitoring

❖ 팩트를 이용한 시스템 모니터링

- ✓ 플레이북 실행: `ansible-playbook monitoring_facts.yml -K`

PLAY RECAP

```
*****  
*****
```

```
master           : ok=4   changed=1   unreachable=0   failed=0   skipped=0  
rescued=0   ignored=0
```

```
worker2          : ok=4   changed=1   unreachable=0   failed=0   skipped=0  
rescued=0   ignored=0
```

- ✓ 호스트 컴퓨터에서 확인: `sudo cat /var/log/daily_check/system_info.logs`

```
##### Start #####
```

```
Date: 2026-04-06 23:51:07
```

```
HostName: worker2
```

```
OS: Ubuntu
```

```
OS Version: 24.04
```

```
OS Kernel: 6.8.0-107-generic
```

```
CPU Cores: 1
```

```
Memory: {'total': 1953, 'used': 1769, 'free': 184}
```

```
Interfaces: ['tunl0', 'enp0s8', 'lo']
```

```
IPv4: ['10.244.189.64', '10.0.2.202']
```

Ansible_Monitoring

❖ CPU, Memory, Disk 사용율 모니터링

✓ 작성

- 변수를 위한 vars_packages.yml

log_directory: /home/adam/logs

packages:

- dstat
- sysstat



Ansible_Monitoring

❖ CPU, Memory, Disk 사용율 모니터링

✓ 작성

- 설치를 위한 monitoring_system.yml

```
- hosts: all
  vars_files: vars_packages.yml
  become: yes
```

tasks:

1. 로그 디렉토리 생성 태스크를 가장 먼저 추가하세요

```
- name: Create log directory
  ansible.builtin.file:
    path: "{{ log_directory }}"
    state: directory
    owner: adam # 디렉토리 소유자를 adam으로 설정
    group: adam
    mode: '0755'
```

Ansible_Monitoring

❖ CPU, Memory, Disk 사용율 모니터링

✓ 작성

● 설치를 위한 monitoring_system.yml

- name: Install packages on RedHat
ansible.builtin.dnf:
 name: "{{ item }}"
 state: present
 loop: "{{ packages }}"
 when: ansible_facts.os_family == "RedHat"
- name: Install packages on Ubuntu
ansible.builtin.apt:
 name: "{{ item }}"
 state: present
 loop: "{{ packages }}"
 when: ansible_facts.os_family == "Debian"



Ansible_Monitoring

❖ CPU, Memory, Disk 사용율 모니터링

✓ 작성

● 설치를 위한 monitoring_system.yml

- name: Monitoring dstat
ansible.builtin.shell: |
{{ item }} >> {{ log_directory }}/dstat.log
loop:
 - dstat 2 10
 - dstat -cmdlt -D vda 2 10

- name: Monitoring iostat
ansible.builtin.shell: |
{{ item }} >> {{ log_directory }}/iostat.log
loop:
 - iostat
 - echo "=====
 - iostat -t -c -dp vda
 - echo "=====

Ansible_Monitoring

❖ CPU, Memory, Disk 사용율 모니터링

✓ 작성

● 설치를 위한 monitoring_system.yml

- name: Monitoring vmstat
ansible.builtin.shell: |
{{ item }} >> {{ log_directory }}/vmstat.log
loop:
 - vmstat
 - echo "=====
 - vmstat -dt
 - echo "=====
 - vmstat -D
 - echo "=====
- name: Monitoring df
ansible.builtin.shell: |
df -h >> {{ log_directory }}/df.log

Ansible_Monitoring

❖ CPU, Memory, Disk 사용율 모니터링

✓ 작성

- 문법 체크: `ansible-playbook --syntax-check monitoring_system.yml`
`playbook: monitoring_system.yml`
- 실행 체크: `ansible-playbook --check monitoring_system.yml`

PLAY RECAP

```
*****
*****

10.0.2.200      : ok=3    changed=0    unreachable=0    failed=0
                 skipped=5    rescued=0    ignored=0

10.0.2.202      : ok=3    changed=0    unreachable=0    failed=0
                 skipped=5    rescued=0    ignored=0

master         : ok=3    changed=0    unreachable=0    failed=0
                 skipped=5    rescued=0    ignored=0

worker2        : ok=3    changed=0    unreachable=0    failed=0
                 skipped=5    rescued=0    ignored=0
```

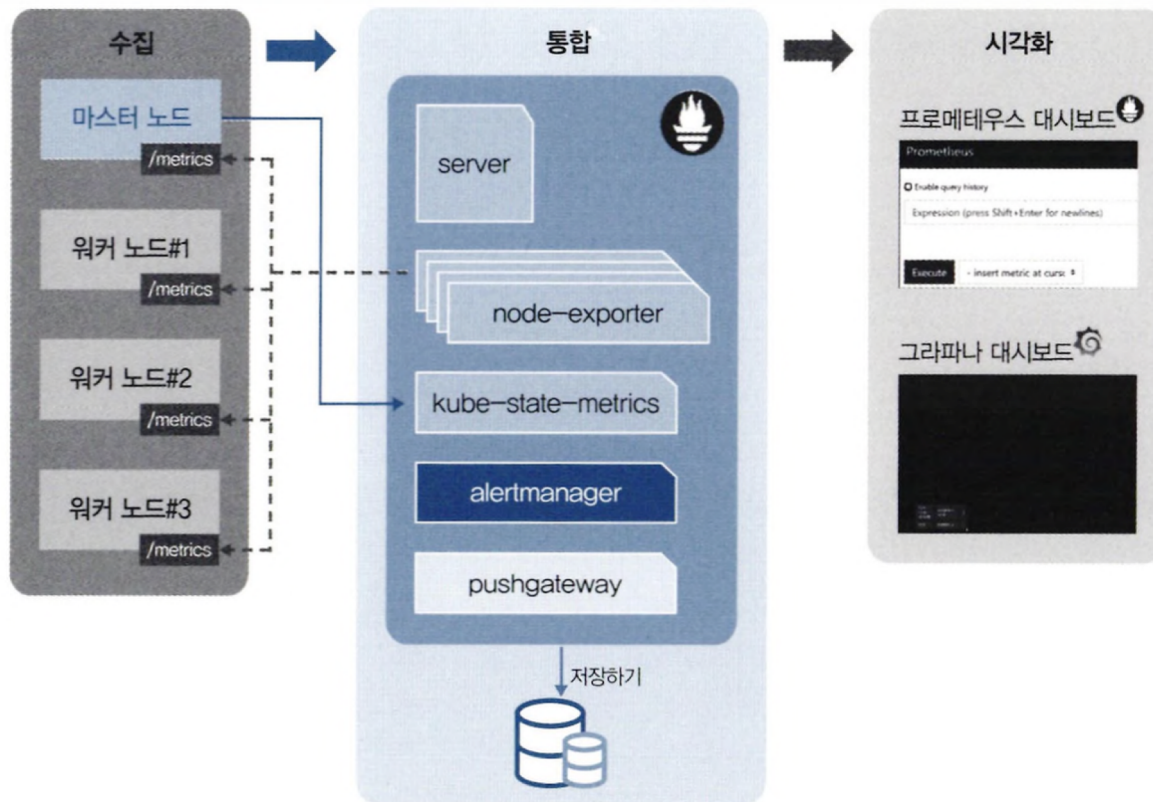

Ansible_Monitoring

- ❖ CPU, Memory, Disk 사용율 모니터링
 - ✓ 작성
 - 실행: `ansible-playbook monitoring_system.yml -K`
`playbook: monitoring_system.yml`



Container Infra Monitoring

- ❖ 컨테이너 인프라 환경의 모니터링
 - ✓ 모니터링을 위한 3단계 구조

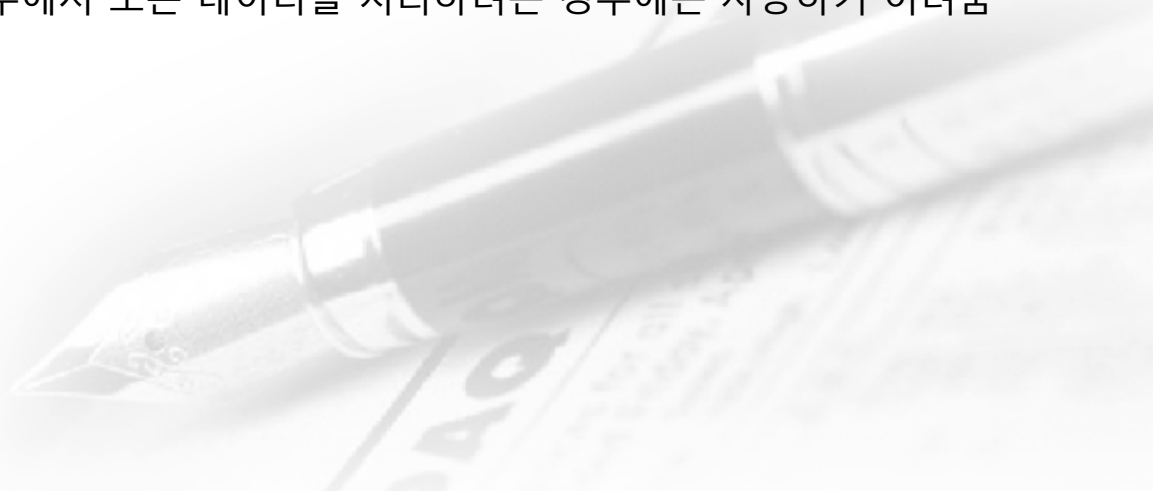


Container Infra Monitoring

❖ 컨테이너 인프라 환경의 모니터링

✓ 모니터링 도구 선택

- 프로메테우스와 그라파나는 오픈 소스 도구
- 오픈 소스는 가능한 한 단일 도구에서 단일 기능만을 구현하는 것을 선호
- 데이터 수집과 통합, 시각화는 서로 다른 영역이므로 이를 함께 구현하지 않으려는 경향이 있는데 서비스형 모니터링 도구는 사용자 편의를 위해 이러한 기능을 모아서 한꺼번에 제공
- 프로메테우스와 그라파나의 혼합 구조를 사용하는 이유
 - ◆ 모니터링 대상마다 라이선스(License) 관련 비용이 발생하고 클라우드 같은 과금제 서비스를 이용하면 네트워크와 저장 공간에 따른 추가 비용이 발생하는데 비용이 민감한 조직은 통합 도구를 사용하는 것이 어려움
 - ◆ 서비스형 모니터링 도구들은 대부분 외부에 데이터를 저장해 모니터링하는데 보안에 민감해서 내부에서 모든 데이터를 처리하려는 경우에는 사용하기 어려움



Container Infra Monitoring

- ❖ 컨테이너 인프라 환경의 모니터링
 - ✓ 데이터 수집 과 통합 도구



Container Infra Monitoring

❖ 컨테이너 인프라 환경의 모니터링

✓ 데이터 수집 과 통합 도구

● Prometheus

- ◆ SoundCloud에서 자사 서비스의 모니터링을 위해 개발한 도구
- ◆ 현재는 오픈 소스로 전환돼 CNCF에서 관리하며 2018년 8월에 졸업(Graduated) 프로젝트
- ◆ 시계열 데이터베이스를 내장하고 있고 자체적인 질의 페이지 외에 시각화 기능은 없으나 그라파나와 연계하면 현업에서 사용할 수 있는 시각화 기능을 제공할 수 있음
- ◆ 중앙 서버에서 에이전트의 데이터를 수집하는 풀(Pull)방식을 사용하므로 쿠버네티스 환경에서 설치된 에이전트를 통해 노드와 컨테이너 상태를 모두 수집해 모니터링할 수 있음
- ◆ 에이전트를 통해 내부 메트릭을 외부로 노출하기 때문에 사용자가 수집 대상에 접속할 수만 있다면 개인 컴퓨터에서도 메트릭을 가져올 수 있기 때문에 모니터링과 관련된 개발을 하기에 편리
- ◆ 일회성으로 모니터링 대상에 대한 세부적인 메트릭도 간단하게 웹 브라우저로 확인할 수 있음
- ◆ 프로메테우스는 완전한 오픈 소스 모델을 선택해 사용자 층이 넓고 관련 자료가 많으며 각종 대시보드 도구나 메신저 등이 프로메테우스와의 연계를 지원하므로 직접 모니터링 시스템을 구축할 때 편리

Container Infra Monitoring

❖ 컨테이너 인프라 환경의 모니터링

✓ 데이터 수집 과 통합 도구

● Datadog

- ◆ 모니터링 데이터를 업체에서 관리하는 경우를 서비스형이라고 하고 데이터를 사용자가 직접 관리하는 경우를 설치형이라고 하는데 데이터독은 서비스형 소프트웨어
- ◆ 웨어(SaaS) 형태로 제공
- ◆ 웹사이트에서 모니터링 대상을 관리할 수 있고 쿠버네티스를 비롯해 여러 클라우드 서비스나 애플리케이션과 연결이 쉬우므로 관리 부담이 적다는 장점이 있음
- ◆ 모니터링 대상 마다 요금을 부과하기 때문에 모니터링 대상이 늘면 비용이 커지는 단점이 있음

● New Relic

- ◆ 데이터독과 같은 서비스형 소프트웨어
- ◆ 데이터독과 비교했을 때 애플리케이션 성능 모니터링(APM, Application Performance Monitoring)에 더 특화되어 있음
- ◆ 데이터독과 마찬가지로 모니터링 대상이 많을수록 비용이 커짐

Container Infra Monitoring

❖ 컨테이너 인프라 환경의 모니터링

✓ 데이터 수집 과 통합 도구

● InfluxDB

- ◆ 2013년 인플럭스데이터(influxckita)에서 개발한 시계열(Time series) 데이터베이스
- ◆ 오픈 소스로 공개된 무료 버전과 클라우드에서 바로 사용할 수 있는 서비스형 소프트웨어, 라이선스를 구매해 설치할 수 있는 엔터프라이즈 버전 유형이 있음
- ◆ 무료 버전은 프로메테우스와 유사하지만 인플럭스DB가 쓰기 성능이 더 뛰어나 대량의 이벤트를 모니터링하는 데는 좀 더 적합
- ◆ 엔터프라이즈 버전은 프로메테우스에서 부족한 부분인고가용성을 위한 분산 저장장을 좀 더 쉽게 구성할 수 있는 기능을 제공
- ◆ 인플럭스DB는 프로메테우스와 더불어 오픈 소스로 모니터링 플랫폼을 구축하기 위한 최선의 도구이고 유료 서비스라는 선택지까지 있어 선택의 폭이 조금 더 넓음
- ◆ 간단한 구성으로 데이터를 받아오는 프로메테우스보다 상대적으로 구성이 어렵다는 단점이 있음

Container Infra Monitoring

- ❖ 컨테이너 인프라 환경의 모니터링
 - ✓ 데이터 수집 과 통합 도구

구분	프로메테우스	데이터독	뉴 릴릭	인플렉스DB
가격	무료	유료	유료	유/무료
형태	설치형	서비스형	서비스형	서비스형/설치형
참고 자료	매우 많음	많음	많음	많음
기능 확장성	매우 좋음	좋음	좋음	좋음



Container Infra Monitoring

- ❖ 컨테이너 인프라 환경의 모니터링
 - ✓ 데이터 시각화 도구



- Grafana
 - ◆ Grafana Labs에서 개발했으며 특정 소프트웨어에 종속되지 않은 독립적인 시각화 도구
 - ◆ 30가지 이상의 다양한 수집 도구 및 데이터베이스들과 연계를 지원
 - ◆ 시계열 데이터의 시각화에 많이 쓰지만 관계형 데이터베이스처럼 데이터를 표 형태로 시각화해 사용할 수도 있음
 - ◆ 기능을 확장하는 플러그인과 개별 사용자들이 만들어 둔 대시보드의 공유가 매우 활발해 필요에 따라 적절히 선택해 사용하면 편리
 - ◆ 오픈 소스라서 사용자의 요구 사항에 맞게 수정이 가능하고 필요에 따라 설치형과 서비스형을 모두 선택할 수 있음

Container Infra Monitoring

❖ 컨테이너 인프라 환경의 모니터링

✓ 데이터 시각화 도구

● 키바나(Kibana)

- ◆ Elasticsearch를 개발한 엘라스틱에서 만든 시각화 도구
- ◆ 엘라스틱서치에 적재된 데이터를 시각화하거나 검색하는 데 사용하고 이러한 데이터를 분석할 때도 사용
- ◆ 엘라스틱서치의 데이터만을 시각화할 수 있기 때문에 프로메테우스의 시계열 데이터를 메트릭비트(Metricbeat)라는 도구로 엘라스틱서치에 전달해야 하는 불편함이 있음
- ◆ 시각화 기능이 매우 강력해서 시각화를 중점적으로 다루는 경우에 사용하면 좋음

● Chronograf

- ◆ 인플렉스DB를 개발한 인플렉스데이터에서 만든 시각화 도구로 오픈 소스로 제공돼 사용자의 편의에 맞게 수정할 수 있음
- ◆ 설치형과 서비스형을 모두 제공해 용도에 따라 선택
- ◆ 키바나와 마찬가지로 자사 제품인 인플렉스DB만 시각화할 수 있으므로 다양한 대상을 시각화할 수 없음

Container Infra Monitoring

- ❖ 컨테이너 인프라 환경의 모니터링
 - ✓ 데이터 시각화 도구

구분	그라파나	키바나	크로노그래프
가격	유/무료	유/무료	유/무료
형태	서비스형/설치형	서비스형/설치형	서비스형/설치형
시각화 대상	다양한 대상 시각화 가능	엘라스틱서치	인플렉스DB
정보량	많음	많음	적음
기능 확장성	좋음	좋음	적음



Container Infra Monitoring

❖ 쿠버네티스 환경에 적합한 모니터링 데이터 수집 방법

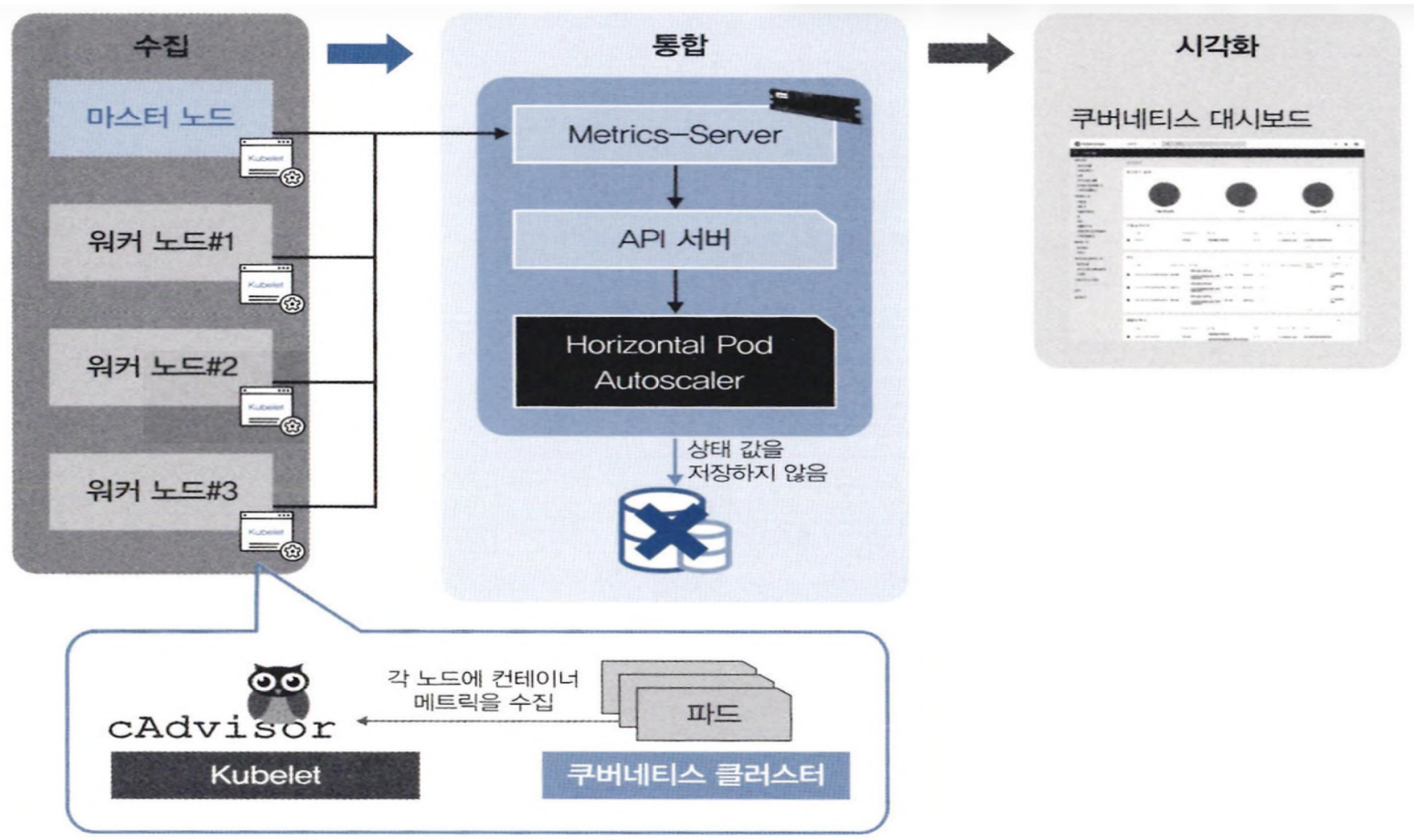
✓ 리소스 메트릭 파이프라인 형태인 메트릭 서버

- 쿠버네티스 노드는 kubelet을 통해 파드를 관리하며 파드의 CPU나 메모리 같은 메트릭 정보를 수집하기 위해 kubelet에 내장된 cAdvisor를 사용
- cAdvisor는 구글이 만든 컨테이너 메트릭 수집 도구로 쿠버네티스 클러스터 위에 배포된 여러 컨테이너가 사용하는 메트릭 정보를 수집한 후 이를 가공해서 kubelet에 전달하는 역할을 수행
- cAdvisor로 수집되고 kubelet으로 공개되는 데이터가 있어도 외부에서 이를 모아서 표현해 주는 도구가 없다면 의미가 없으므로 메트릭 데이터를 수집하는 목적으로 메트릭 서버를 설치해 HPA와 같은 기능을 구현하고 쿠버네티스 대시보드를 설치해 현재 상태를 확인할 수 있음
- 메트릭 서버에서 수집한 데이터로 여러 기능을 수행하도록 구성한 것이 리소스 메트릭 파이프라인(Resource Metric Pipeline)



Container Infra Monitoring

- ❖ 쿠버네티스 환경에 적합한 모니터링 데이터 수집 방법
 - ✓ 리소스 메트릭 파이프라인 형태인 메트릭 서버



Container Infra Monitoring

❖ 쿠버네티스 환경에 적합한 모니터링 데이터 수집 방법

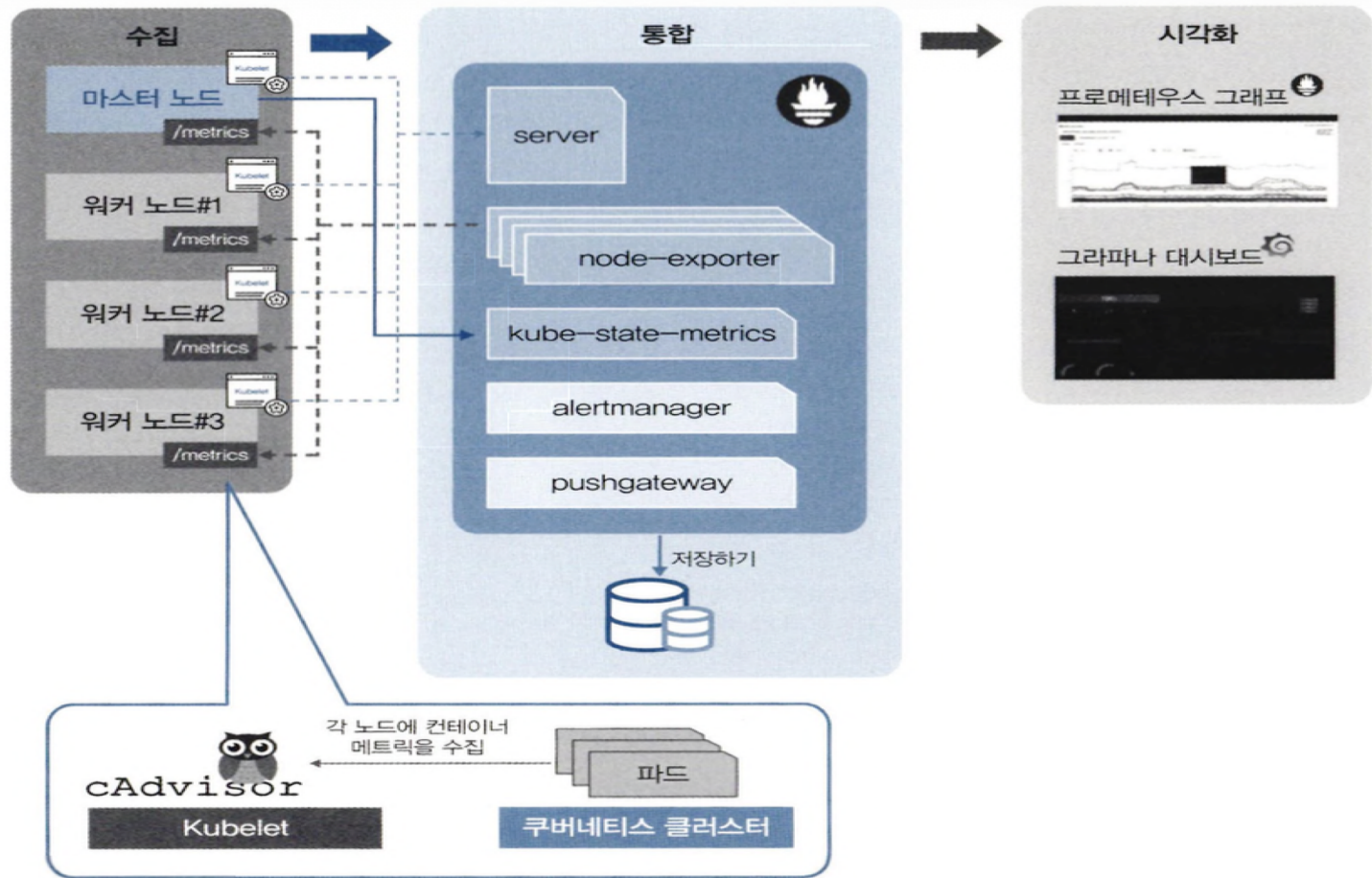
✓ 완전한 모니터링 파이프라인

- 메트릭 서버는 집계한 데이터를 메모리에만 저장하므로 데이터를 영구적으로 보존하기 어렵고 현재 시점의 데이터만 출력
- 메트릭 데이터를 저장 공간에 따로 저장하는 완전한 모니터링 파이프라인(Full Monitoring Pipeline)으로 구축하기를 권장
- 이 설계 방식을 반영한 도구가 프로메테우스
- 완전한 모니터링 파이프라인으로 구성된 프로메테우스는 여러 수집 대상이 공개하는 메트릭 데이터를 모아 시계열 데이터베이스에 저장
- 저장된 데이터는 시간이 지나도 확인할 수 있는 영속적인 데이터
- 누적된 메트릭 데이터로는 쿠버네티스 인프라의 상태 변화를 파악할 수 있고 적절한 위험 감지 및 조치를 취할 수 있음



Container Infra Monitoring

- ❖ 쿠버네티스 환경에 적합한 모니터링 데이터 수집 방법
 - ✓ 완전한 모니터링 파이프라인



Container Infra Monitoring

- ❖ 프로메테우스로 모니터링 데이터 수집 과 통합
 - ✓ 프로메테우스의 오브젝트
 - prometheus-server
 - ◆ 프로메테우스의 주요 기능을 수행하는 요소
 - ◆ 역할
 - 노드 익스포터 외 여러 대상에서 메트릭을 수집해오는 수집기
 - 수집한 시계열 메트릭 데이터를 저장하는 시계열 데이터베이스
 - 저장된 데이터를 질의하거나 수집 대상의 상태를 확인할 수 있는 웹 UI
 - ◆ 프로메테우스의 수집기는 매우 독특한 방법으로 수집 대상을 찾는데 서비스 디스커버리(Service Discovery)라는 방법
 - node-exporter
 - ◆ 노드의 시스템 메트릭 정보를 HTTP로 공개하는 역할
 - ◆ 설치된 노드에서 특정 파일들을 읽고 이를 프로메테우스 서버가 수집할 수 있는 메트릭 데이터로 변환한 후에 노드 익스포터에서 HTTP 서버로 공개
 - kube-state-metrics
 - ◆ API 서버로 쿠버네티스 클러스터의 여러 메트릭 데이터를 수집한 후 이를 프로메테우스 서버가 수집할 수 있는 메트릭 데이터로 변환해 공개하는 역할
 - ◆ 프로메테우스가 쿠버네티스 클러스터의 여러 정보를 손쉽게 획득할 수 있는 것은 모두 쿠커 스테이트 메트릭 덕분

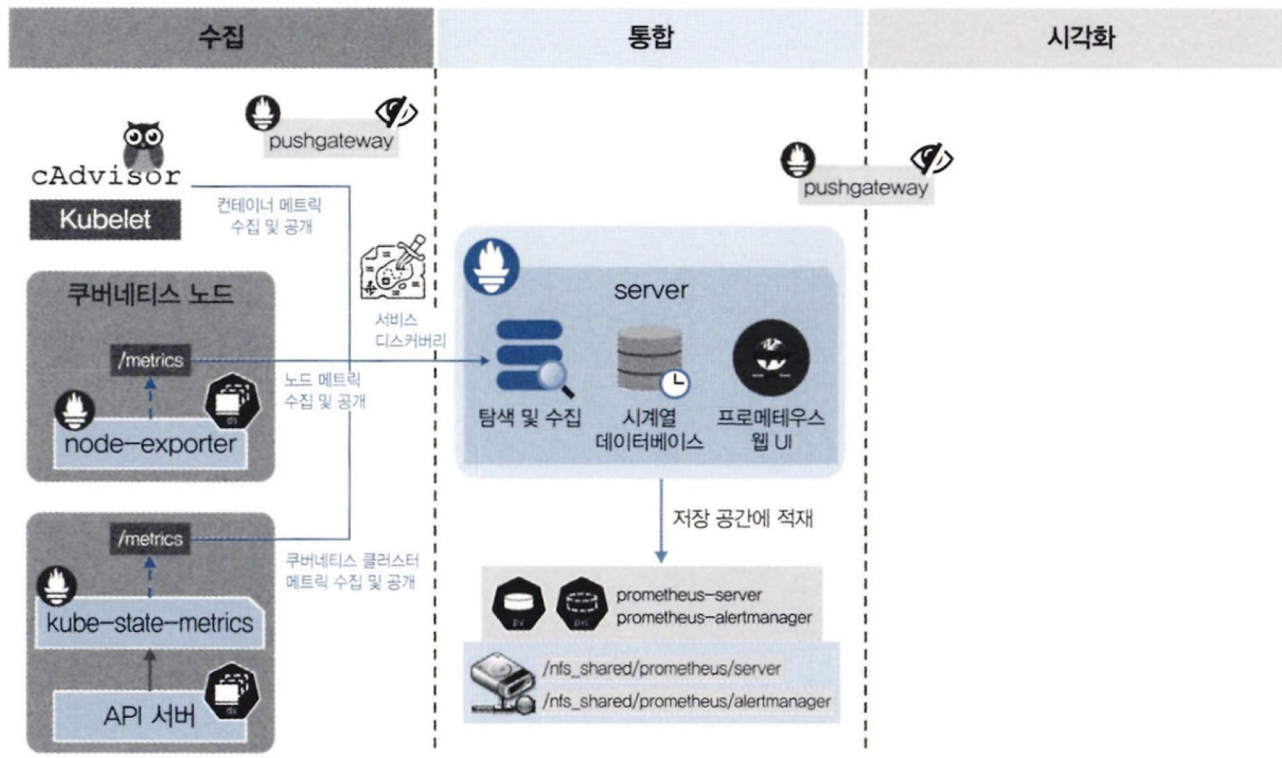
Container Infra Monitoring

- ❖ 프로메테우스로 모니터링 데이터 수집 과 통합
 - ✓ 프로메테우스의 오브젝트
 - alertmanager
 - ◆ 얼럿매니저는 프로메테우스에 경보(alert) 규칙을 설정하고 경보 이벤트가 발생하면 설정된 경보 메시지를 대상에게 전달하는 기능을 제공
 - ◆ 프로메테우스에 설치하면 프로메테우스 서버에서 주기적으로 경보를 보낼 대상을 감시해 시스템을 안정적으로 운영할 수 있게 함
 - pushgateway
 - ◆ 배치와 스케줄 작업 시 수행되는 일회성 작업들의 상태를 저장하고 모아서 프로메테우스가 주기적으로 가져갈 수 있도록 공개
 - ◆ 짧은 시간 동안 실행되고 종료되는 배치성 프로그램의 메트릭을 저장하거나 외부망에서 접근할 수 없는 내부 시스템의 메트릭을 프록시 형태로 제공하는 용도로 사용



Container Infra Monitoring

- ❖ 프로메테우스로 모니터링 데이터 수집 과 통합
 - ✓ 프로메테우스의 모니터링 데이터 흐름



Container Infra Monitoring

❖ Prometheus & Grafana 설치

✓ Helm 설치

- 설치: `curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash`
- 설치 확인: `helm version`

✓ 프로메테우스 스택 설치

- 레포지토리 추가 및 업데이트
 - ◆ `helm repo add prometheus-community https://prometheus-community.github.io/helm-charts`
 - ◆ `helm repo update`
- 네임스페이스 생성: `kubectl create namespace monitoring`
- 차트 설치 (kube-stack-Prometheus: prometheus-stack이라는 이름으로 설치): `helm install prometheus-stack prometheus-community/kube-prometheus-stack -n monitoring`



Container Infra Monitoring

❖ Prometheus & Grafana 설치

✓ 설치 확인: `kubectl get pods -n monitoring`

NAME	READY	STATUS	RESTARTS	AGE
alertmanager-prometheus-stack-kube-prom-alertmanager-0 58m	2/2	Running	0	
prometheus-prometheus-stack-kube-prom-prometheus-0 58m	2/2	Running	0	
prometheus-stack-grafana-54c5448d4-d5lsb 58m	3/3	Running	0	
prometheus-stack-kube-prom-operator-66999dc5df-kph4d 58m	1/1	Running	0	
prometheus-stack-kube-state-metrics-75c97578f5-mgt9k 58m	1/1	Running	0	
prometheus-stack-prometheus-node-exporter-nc7v4 58m	1/1	Running	0	
prometheus-stack-prometheus-node-exporter-qmhmr 58m	1/1	Running	0	
prometheus-stack-prometheus-node-exporter-xg9tn 58m	1/1	Running	0	

Container Infra Monitoring

❖ Prometheus & Grafana 설치

✓ 대시 보드 접속

- 포트 포워딩 이용: # 그라파나 접속 (기본 포트 3000): `kubectl port-forward deployment/prometheus-stack-grafana 3000:3000 -n monitoring --address 0.0.0.0`
- 노드 포트 이용
 - ◆ 서비스 변경: `kubectl patch svc prometheus-stack-grafana -n monitoring -p '{"spec": {"type": "NodePort"}}'`
 - ◆ 할당된 포트 확인: `kubectl get svc -n monitoring`

NAME	TYPE	CLUSTER-IP	EXTERNAL-
IP	PORT(S)	AGE	
prometheus-stack-grafana	NodePort	10.104.210.227	
<none>	80:31383/TCP	61m	
prometheus-stack-kube-prom-alertmanager	ClusterIP	10.106.55.103	
<none>	9093/TCP,8080/TCP	61m	
prometheus-stack-kube-prom-operator	ClusterIP	10.97.0.174	
<none>	443/TCP	61m	
prometheus-stack-kube-prom-prometheus	ClusterIP	10.111.128.237	
<none>	9090/TCP,8080/TCP	61m	
prometheus-stack-kube-state-metrics	ClusterIP	10.104.117.120	
<none>	8080/TCP	61m	
prometheus-stack-prometheus-node-exporter	ClusterIP	10.101.206.250	
<none>	9100/TCP	61m	

Container Infra Monitoring

❖ Prometheus & Grafana 설치

✓ 그라파나 로그인 정보

- ID: admin
- 초기 비밀번호 확인: `kubectl get secret -n monitoring prometheus-stack-grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo`

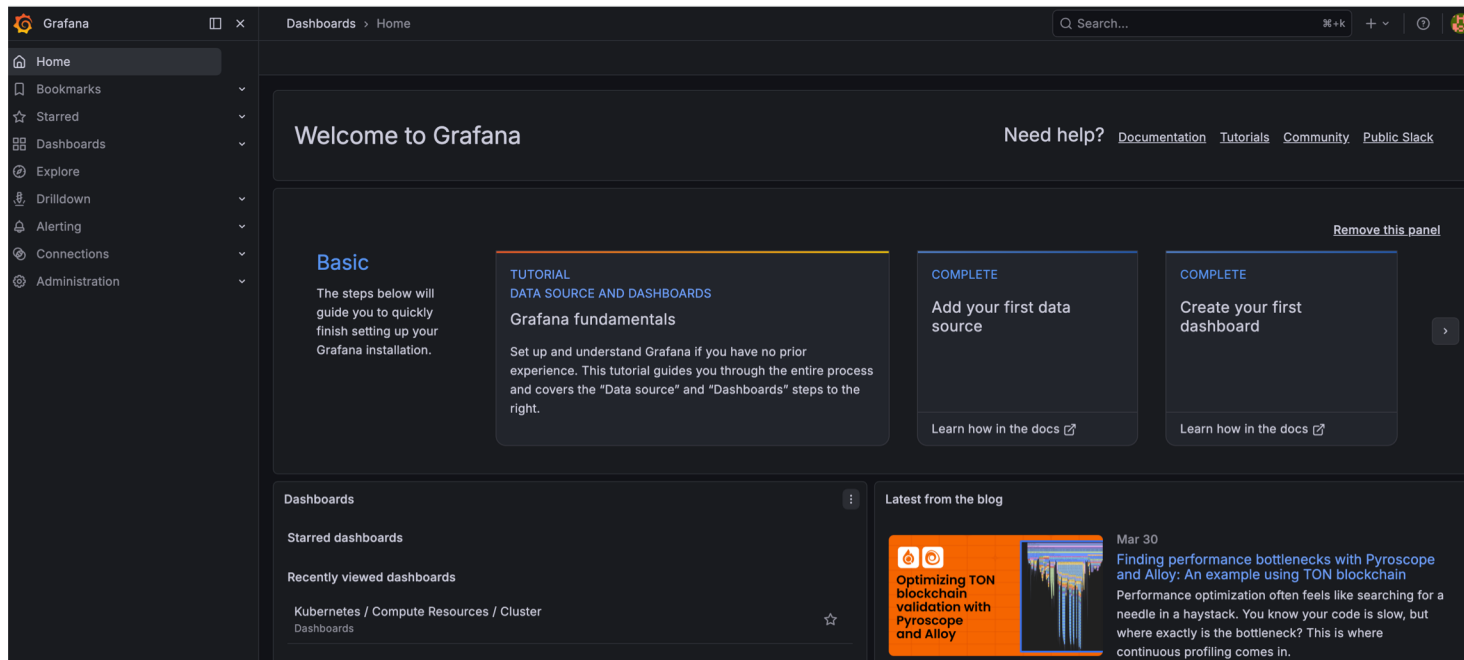


Container Infra Monitoring

❖ Prometheus & Grafana 설치

✓ 그라파나 대시 보드 접속

- 대시보드 확인: <http://13.125.167.205:31383/>

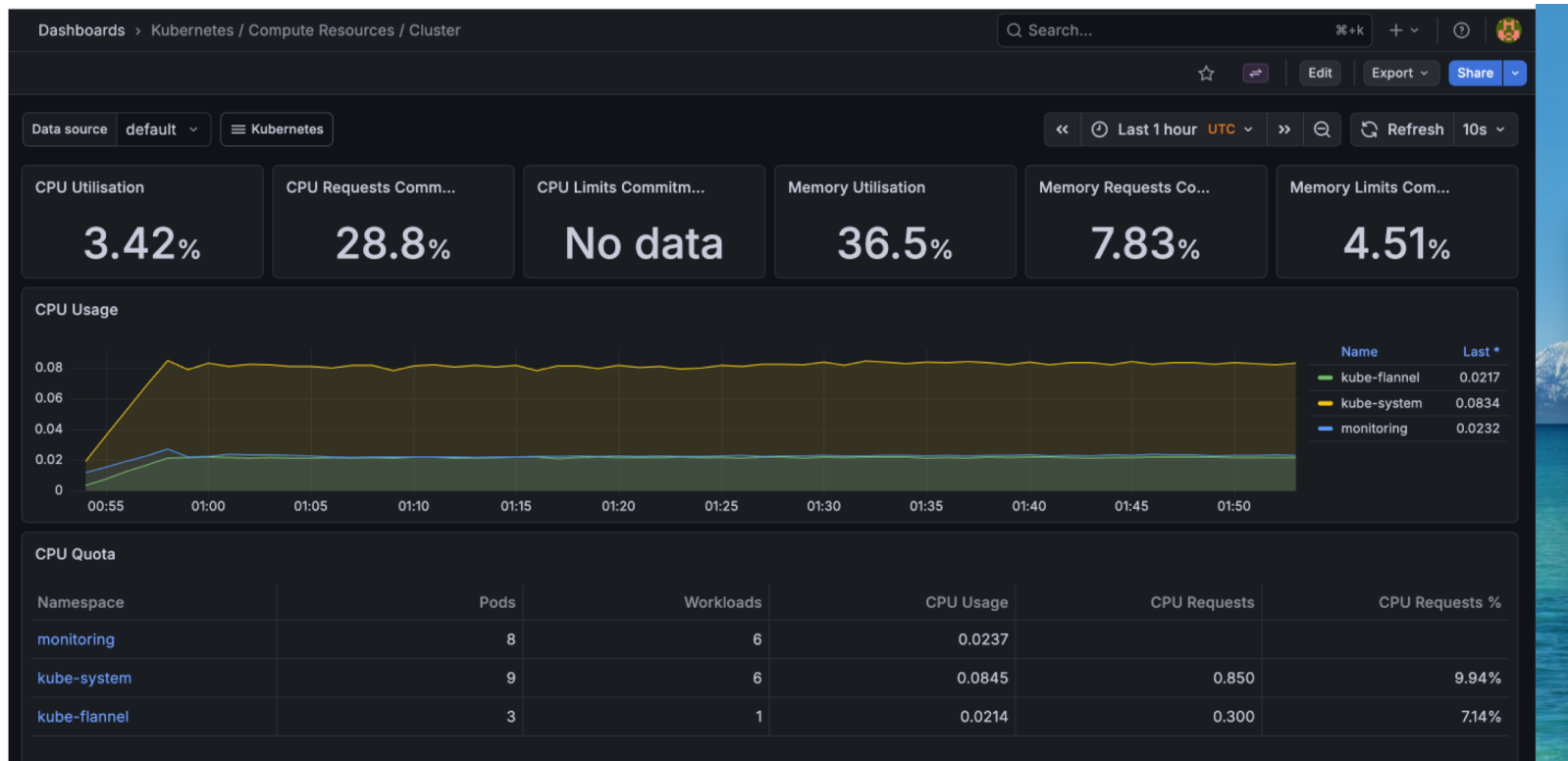


Container Infra Monitoring

❖ Prometheus & Grafana 설치

✓ 그라파나와 쿠버네티스 와 연동 확인

- 그라파나 접속 후 좌측 메뉴의 Dashboards 클릭.
- Browse 선택 후 Kubernetes / Compute Resources / Cluster 등의 기본 대시보드를 클릭하면 클러스터 상태를 바로 볼 수 있음



Container Infra Monitoring

❖ Prometheus & Grafana 설치

✓ Prometheus Web UI 접속

- 서비스 타입을 ClusterIP에서 NodePort로 변경: `kubectl patch svc prometheus-stack-kube-prom-prometheus -n monitoring -p '{"spec": {"type": "NodePort"}}'`
- 서비스 확인: `kubectl get service -n monitoring`

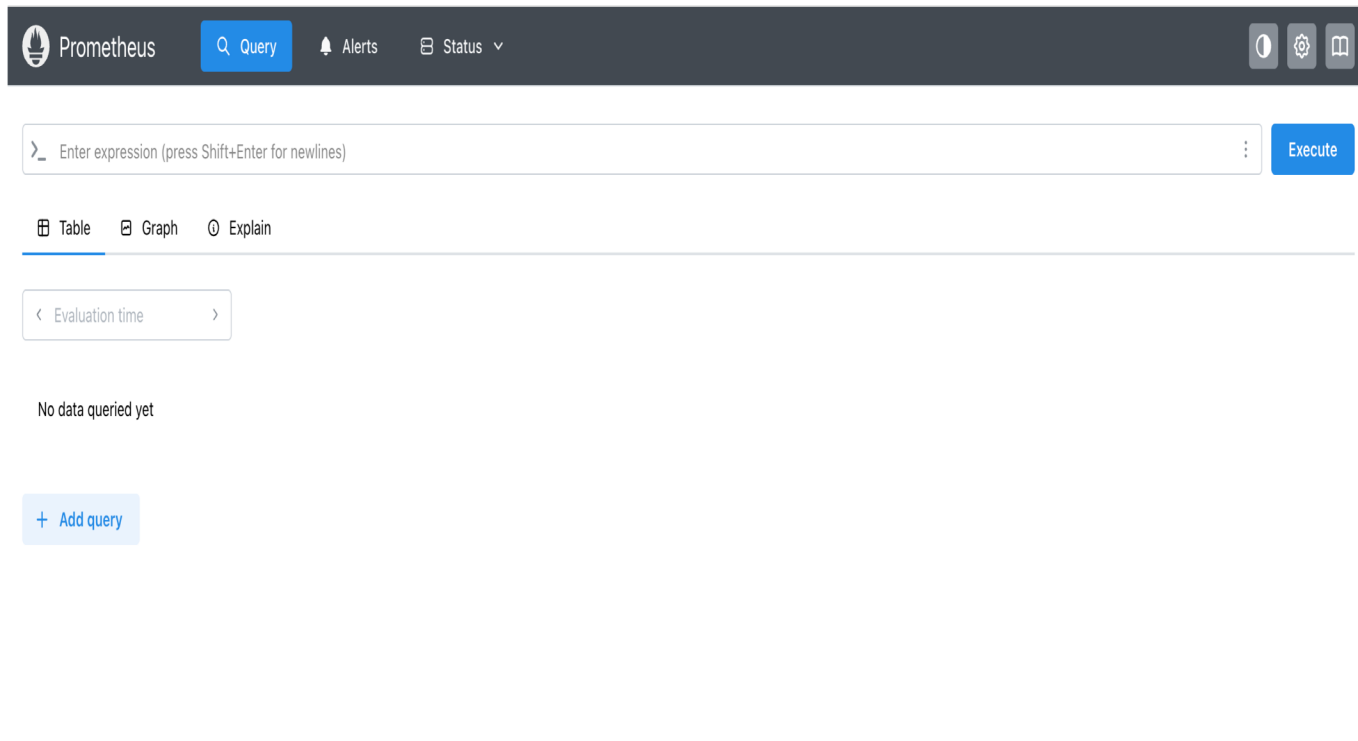
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
PORT(S)	AGE		
alertmanager-operated	ClusterIP	None	<none>
9093/TCP,9094/TCP,9094/UDP	148m		
prometheus-operated	ClusterIP	None	<none>
9090/TCP	148m		
prometheus-stack-grafana	NodePort	10.102.155.166	<none>
80:30320/TCP	149m		
prometheus-stack-kube-prom-alertmanager	ClusterIP	10.104.181.2	
<none> 9093/TCP,8080/TCP	149m		
prometheus-stack-kube-prom-operator	ClusterIP	10.107.66.253	<none>
443/TCP	149m		
prometheus-stack-kube-prom-prometheus	NodePort	10.99.65.79	
<none> 9090:31570/TCP,8080:31242/TCP	149m		
prometheus-stack-kube-state-metrics	ClusterIP	10.98.165.66	<none>
8080/TCP	149m		
prometheus-stack-prometheus-node-exporter	ClusterIP	10.109.107.213	
<none> 9100/TCP	149m		

Container Infra Monitoring

❖ Prometheus & Grafana 설치

✓ Prometheus Web UI 접속

- 접속: <http://13.125.22.252:31570/query>



Container Infra Monitoring

❖ 설치 시 주의 사항

- ✓ 소스 확인: 프로메테우스는 메모리를 상당히 많이 사용하므로 노드의 RAM이 최소 4GB 이상인지 확인
- ✓ StorageClass: 퍼시스턴트 볼륨(PV) 설정이 되어 있지 않으면 Pod 재시작 시 데이터가 사라지므로 운영 환경이라면 nfs나 local-path-provisioner 등을 통해 StorageClass를 구성해야 함
- ✓ 대시보드에 데이터가 출력되지 않는다면 kube-state-metrics나 node-exporter Pod의 상태를 먼저 확인

