

8th
Week

어!? 마지막 수업이네요?!

▷ 출석 체크도 한 번 해보시면 어떠세요?!

- <https://modulabs.co.kr/>

- 모두연 홈페이지 → 로그인 → 마이페이지 → 참여한 랩·풀잎 → 자세히 보기 → 내 풀잎스쿨 출석 확인하기

▷ Ground Rule

- 가급적 지각/결석 하지 않기
- 가급적 Camera 켜 놓고 수업 참여하기
- 가급적 적극적으로 참여하기
- 3시간이 넘더라도 배고프다고 화내지 않기
- Slack 잊지 않기
- 꼭 끝까지 함께하기

잡담 & 지난 수업 관련 이야기



Agenda

- ▷ [10m] make-friendship
- ▷ [40m] Resource Management
- ▷
- ▷ [10m] Break-Time
- ▷ [50m] Monitoring (Prometheus/Grafana)
- ▷ [20m] Quiz

Kubernetes

Resource Management

Status

remote > **kubectl get nodes -o wide**

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
master	Ready	control-plane,master	46d	v1.22.5	192.168.100.200	<none>	Ubuntu 20.04.4 LTS	5.4.0-100-generic	containerd://1.5.8
worker1	Ready	<none>	46d	v1.22.5	192.168.100.201	<none>	Ubuntu 20.04.4 LTS	5.4.0-100-generic	containerd://1.5.8
worker2	Ready	<none>	46d	v1.22.5	192.168.100.202	<none>	Ubuntu 20.04.4 LTS	5.4.0-100-generic	containerd://1.5.8

remote > **kubectl describe nodes master**

...

Capacity:

cpu:

2

ephemeral-storage:

25155844Ki

hugepages-2Mi:

0

memory:

2030728Ki

pods:

110

Allocatable:

cpu:

1800m

ephemeral-storage:

23183625793

hugepages-2Mi:

0

memory:

1404040Ki

pods:

110

...

Non-terminated Pods:

(13 in total)

Namespace	Name	CPU Requests	CPU Limits	Memory Requests	Memory Limits	Age
-----	----	-----	-----	-----	-----	---
kube-system	calico-kube-controllers-5788f6558-c4dmn	30m (1%)	1 (55%)	64M (4%)	256M (17%)	11d
kube-system	calico-node-jtspn	150m (8%)	300m (16%)	64M (4%)	500M (34%)	46d

...

Allocated resources:

(Total limits may be over 100 percent, i.e., overcommitted.)

Resource	Requests	Limits
-----	-----	-----
cpu	1100m (61%)	1500m (83%)
memory	559001600 (38%)	1578231040 (109%)
ephemeral-storage	0 (0%)	0 (0%)
hugepages-2Mi	0 (0%)	0 (0%)

Events:

<none>

Units

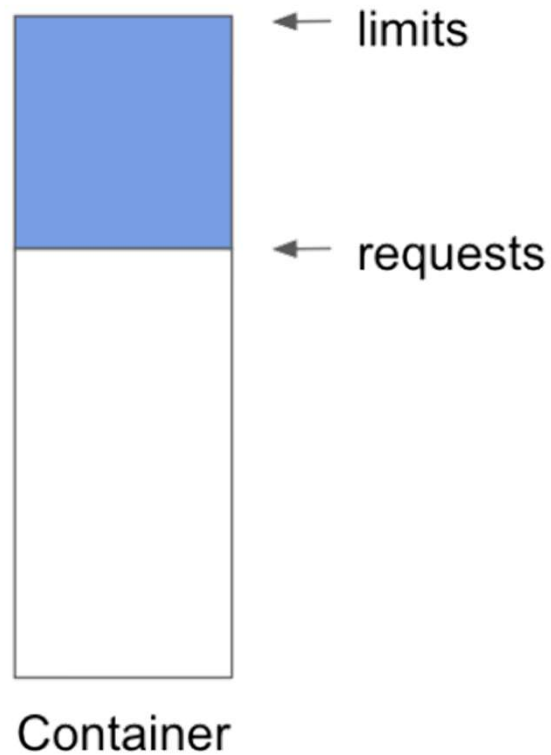
- CPU : **ms** (밀리 세컨드), 1000ms = 1 vCore (가상 CPU 코어)
 - . 1 = 1000ms, 0.5 = 500ms
- Memory : **Mi** (MiB, 메비바이트), 1 MiB = 1024 KiB

바이트 크기 v · d · e · h					
SI 접두어		전통적 용법		이진 접두어	
기호(이름)	값	기호	값	기호(이름)	v값
kB (킬로바이트)	$1000^1 = 10^3$	KB	$1024^1 = 2^{10}$	KiB (키비바이트)	2^{10}
MB (메가바이트)	$1000^2 = 10^6$	MB	$1024^2 = 2^{20}$	MiB (메비바이트)	2^{20}
GB (기가바이트)	$1000^3 = 10^9$	GB	$1024^3 = 2^{30}$	GiB (기비바이트)	2^{30}
TB (테라바이트)	$1000^4 = 10^{12}$	TB	$1024^4 = 2^{40}$	TiB (테비바이트)	2^{40}
PB (페타바이트)	$1000^5 = 10^{15}$	PB	$1024^5 = 2^{50}$	PiB (페비바이트)	2^{50}
EB (엑사바이트)	$1000^6 = 10^{18}$	EB	$1024^6 = 2^{60}$	EiB (엑스비바이트)	2^{60}
ZB (제타바이트)	$1000^7 = 10^{21}$	ZB	$1024^7 = 2^{70}$	ZiB (제비바이트)	2^{70}
YB (요타바이트)	$1000^8 = 10^{24}$	YB	$1024^8 = 2^{80}$	YiB (요비바이트)	2^{80}

※ 참고 : <https://ko.wikipedia.org/wiki/메비바이트>

Requests & Limits

- **requests** : container가 생성될 때 요청하는 리소스
- **limits** : container가 생성되고 CPU/Memory가 더 필요한 경우 추가로 더 사용할 수 있는 리소스





Example - requests

- dd : 파일을 변환하고 복사하는 것이 주 목적인 유닉스 및 유닉스 계열 운영 체제용 명령 줄 유틸리티
- /dev/zero : 읽기를 위해 가능한 많은 널 문자(ASCII NUL, 0x00)를 제공하는 유닉스 계열 운영 체제의 특수 파일
- /dev/null : 기록 대상이 되는 모든 데이터를 버리지만 쓰기 작업은 성공했다고 보고하는 장치 파일

pod-requests.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: requests

spec:
  containers:
    CPU를 100% 사용하는 예제
    - image: busybox
      command: ["dd", "if=/dev/zero", "of=/dev/null"]

      name: dd

      resources:
        requests:
          cpu: 200m
          memory: 10Mi
```

```
remote > git clone https://github.com/whatwant-school/advanced-kubernetes.git
remote > cd advanced-kubernetes
```

```
remote > kubectl create -f 08-week/requests/pod-requests.yaml
```

```
pod/requests created
```

```
remote > kubectl exec -it requests -- top
```

```
Mem: 1166988K used, 89596K free, 2060K shrd, 41788K buff, 692296K cached
CPU: 12.2% usr 39.3% sys 0.0% nic 48.1% idle 0.1% io 0.0% irq 0.1% sirq
Load average: 1.14 1.13 1.09 4/394 17
PID PPID USER STAT VSZ %VSZ CPU %CPU COMMAND
1 0 root R 1312 0.1 1 50.3 dd if /dev/zero of /dev/null
8 0 root R 1320 0.1 0 0.0 top
```

일부러 vCPU=2 환경에서 실행했다.
dd = 최대 CPU 소비, but 1 thread
실행환경 2 cpu, ∴ 50%
→ requests cpu 200m, but use 1000m

※ 참고 : [https://ko.wikipedia.org/wiki/Dd_\(유닉스\)](https://ko.wikipedia.org/wiki/Dd_(유닉스))

※ 참고 : <https://ko.wikipedia.org/wiki//dev/zero>

※ 참고 : https://ko.wikipedia.org/wiki/널_장치

Example - limits

- requests를 설정하지 않으면, limits 값으로 requests 값 설정됨
- 실행환경 2 cpu, limits cpu 200m → ∴ 10%

pod-limits.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: limits

spec:
  containers:
    - image: busybox
      command: ["dd", "if=/dev/zero", "of=/dev/null"]

      name: dd

      resources:
        limits:
          cpu: 200m
          memory: 10Mi
```

```
remote > git clone https://github.com/whatwant-school/advanced-kubernetes.git
remote > cd advanced-kubernetes
```

```
remote > kubectl create -f 08-week/limits/pod-limits.yaml
```

```
pod/limits created
```

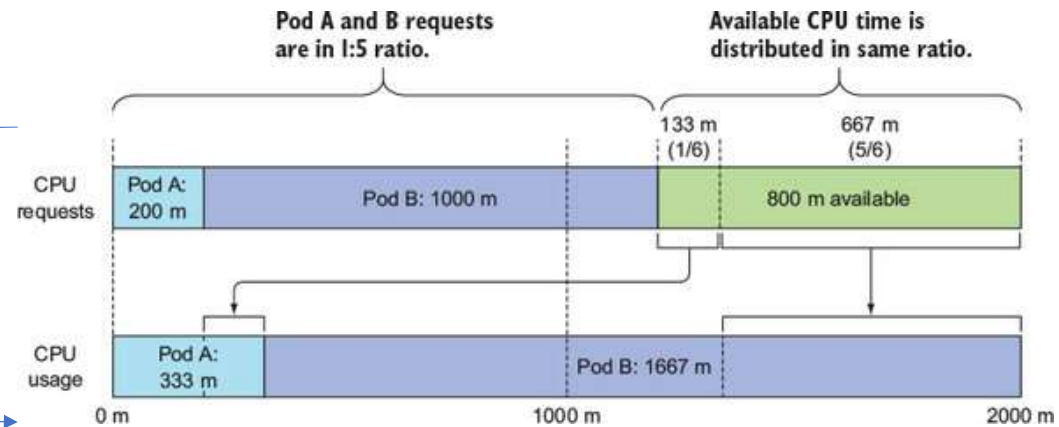
```
remote > kubectl exec -it requests -- top
```

```
Mem: 1172240K used, 84344K free, 2076K shrd, 42848K buff, 693392K cached
CPU:  2.5% usr  8.4% sys  0.0% nic 88.8% idle  0.0% io  0.0% irq  0.0% sirq
Load average: 0.60 1.00 1.07 2/399 14
  PID  PPID  USER      STAT  VSZ %VSZ  CPU %CPU COMMAND
    1     0 root        R    1312  0.1    0  9.9 dd if /dev/zero of /dev/null
    7     0 root        R    1320  0.1    1  0.0 top
```



CPU time sharing

이렇게 요청했지만,
실제로는 전체 CPU를 비율만큼 사용한다.





containers always see the node's memory/cpu, not the container's

- **cpu.cfs_period_us** : CPU 자원을 정기적으로 할당 받을 주기(microsecond 단위)

. 파라미터의 max 1 second, min 1,000 microsecond

- **cpu.cfs_quota_us** : 모든 task들이 한 주기(period) 동안 실행할 수 있는 총 시간(microsecond 단위)

. 예를 들어, 매 1초당 0.2초 동안 단일 CPU에 액세스할 수 있어야 하는 경우 `cpu.cfs_quota_us=200,000`, `cpu.cfs_period_us=1,000,000`

. 예를 들어, 두 개의 CPU를 완전히 활용(2 core) 한다면 `cpu.cfs_quota_us=200,000(200ms)`, `cpu.cfs_period_us=100,000(100ms)`

. `cpu.cfs_quota_us`의 값을 -1로 설정하면, CPU 시간 제한을 설정하지 않는다는 의미이며, 이는 root cgroup을 제외한 모든 cgroup의 기본 값임

```
remote > kubectl exec -it limits -- cat /sys/fs/cgroup/cpu/cpu.cfs_period_us
```

```
100000 100,000 = 100ms
```

```
remote > kubectl exec -it limits -- cat /sys/fs/cgroup/cpu/cpu.cfs_quota_us
```

```
20000 20,000 = 20ms
```

1 CPU 사용률 = $\text{cpu.cfs_quota_us} / \text{cpu.cfs_period_us} * 100$
 $= 20,000 / 100,000 * 100$
 $= 20\%$
→ vCPU 1개의 20% 성능까지 사용

※ 참고 : <https://velog.io/@jrlee/cgroup-subsystem-CPU>



QoS (Quality of Service) – 1/2

- **BestEffort** : 최하위 우선순위

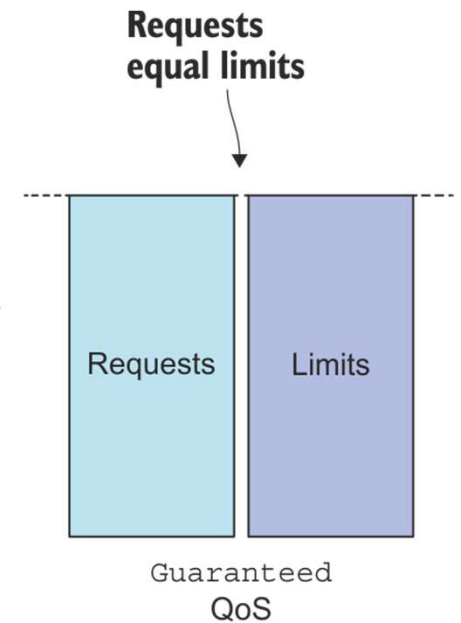
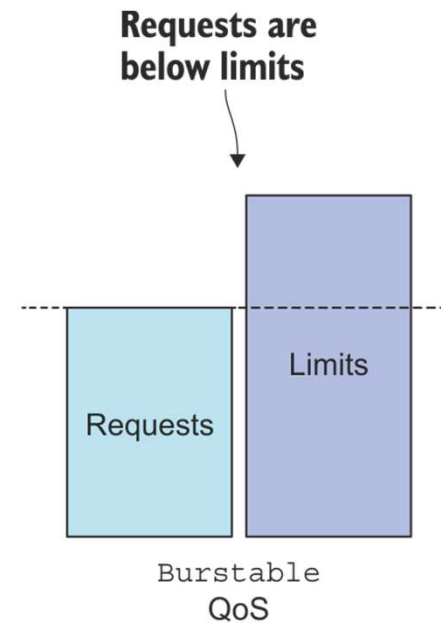
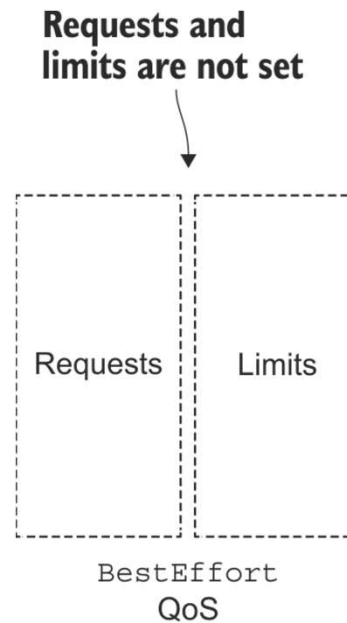
- . requests/limits 지정하지 않은 pod
- . 가장 먼저 종료
- . 메모리가 충분하면 최대 메모리 사용

- **Burstable**

- . BestEffort/Guaranteed에 해당하지 않는 Pod
- . requests ~ limits 범위의 리소스 얻음

- **Guaranteed** : 최상위 우선순위

- . 3가지 조건 충족되어야 함
 - ① requests/limits 모두 설정
 - ② 각 container에 모두 설정
 - ③ requests == limits



QoS (Quality of Service) – 2/2

- 2개 container를 갖고 있는 경우, 각 container QoS에 따른 Pod의 QoS 결과

Table 14.1 The QoS class of a single-container pod based on resource requests and limits

CPU requests vs. limits	Memory requests vs. limits	Container QoS class
None set	None set	BestEffort
None set	Requests < Limits	Burstable
None set	Requests = Limits	Burstable
Requests < Limits	None set	Burstable
Requests < Limits	Requests < Limits	Burstable
Requests < Limits	Requests = Limits	Burstable
Requests = Limits	Requests = Limits	Guaranteed

Table 14.2 A Pod's QoS class derived from the classes of its containers

Container 1 QoS class	Container 2 QoS class	Pod's QoS class
BestEffort	BestEffort	BestEffort
BestEffort	Burstable	Burstable
BestEffort	Guaranteed	Burstable
Burstable	Burstable	Burstable
Burstable	Guaranteed	Burstable
Guaranteed	Guaranteed	Guaranteed

※ 참고 : <https://livebook.manning.com/book/kubernetes-in-action/chapter-14/133>

which process gets killed when memory is low

- QoS 클래스에 따라 해당 프로세스 종료

- 동일하면? → OOM Score (Out of Memory)

. 아래 2가지 기준을 QoS 클래스를 기반으로 한 고정된 OOM Score 조정

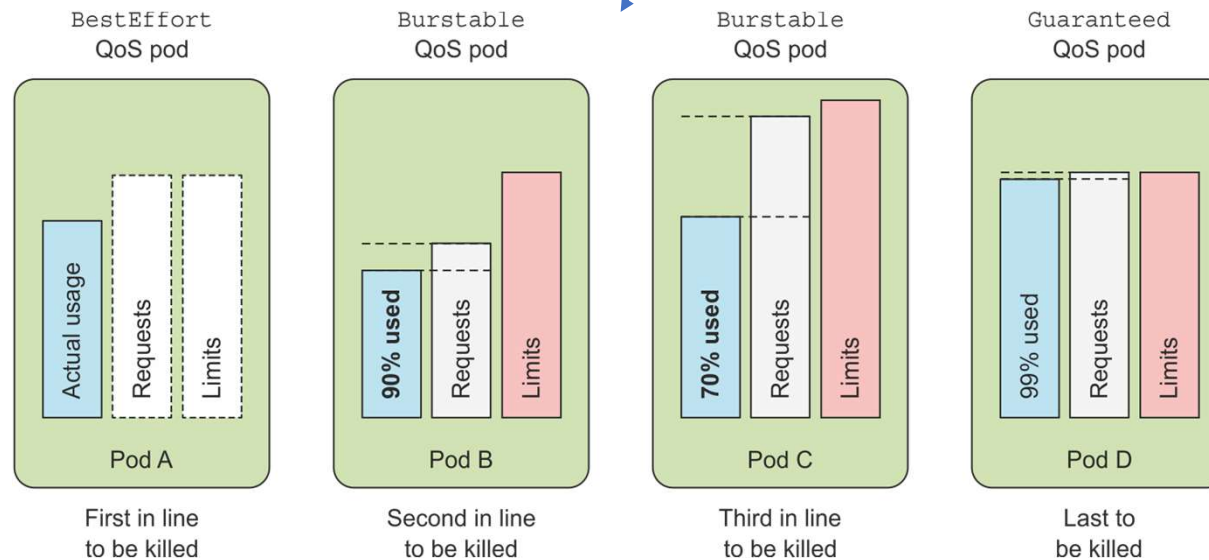
① 프로세스가 소비하는 가용 메모리 비율

② requests Memory

Requests 대비하여

사용하는 비율이 높은 Pod가 먼저 종료

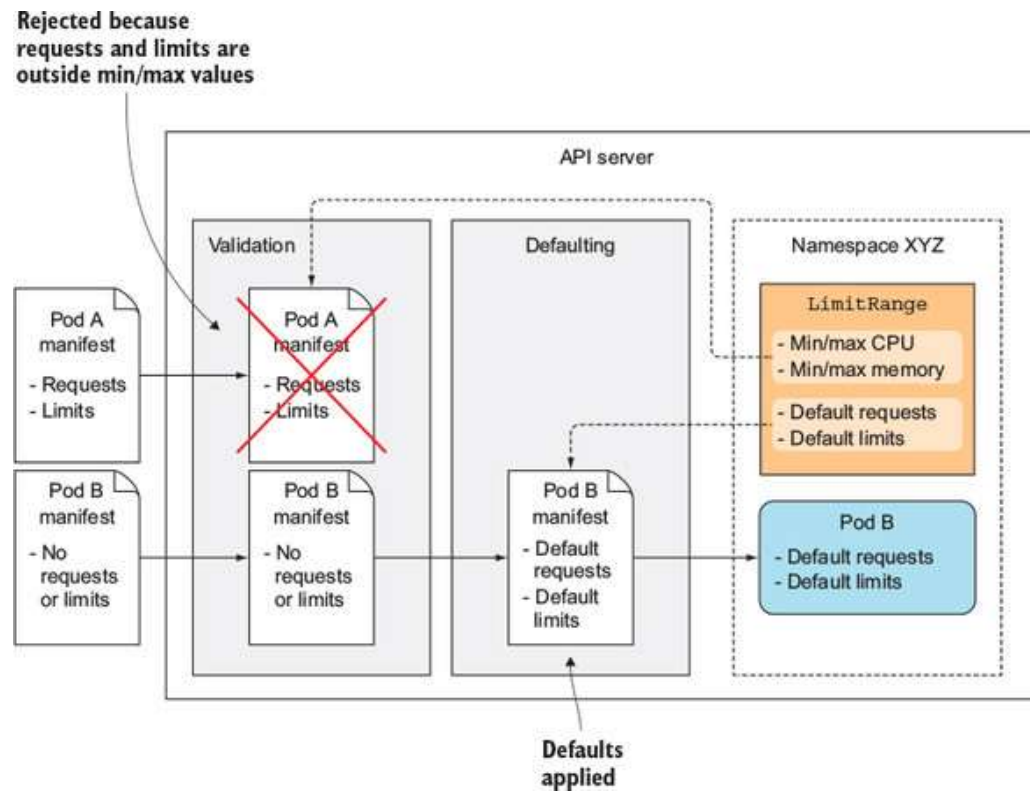
4개의 Pod에 대해서
어떤 것부터
프로세스 종료되는지 설명





LimitRange

- Namespace 단위로 리소스에 대한 min/max/default 값 설정



정해 놓은 *min/max* 값을 벗어난 요청은 거절

따로 명시하지 않아도 *default* 값 기본 적용

Example - LimitRange 1/2

limitrange.yaml

```
apiVersion: v1
kind: LimitRange
metadata:
  name: limitrange
```

```
spec:
  limits:
  - type: Pod
    min:
      cpu: 50m
      memory: 5Mi
    max:
      cpu: 1
      memory: 1Gi
```

```
  - type: Container
    defaultRequest:
      cpu: 100m
      memory: 10Mi
    default:
      cpu: 200m
      memory: 100Mi
    min:
      cpu: 50m
      memory: 5Mi
    max:
      cpu: 1
      memory: 1Gi
    maxLimitRequestRatio:
      cpu: 4
      memory: 10
```

```
- type: PersistentVolumeClaim
  min:
    storage: 1Gi
  max:
    storage: 10Gi
```

```
remote > git clone https://github.com/whatwant-school/advanced-kubernetes.git
remote > cd advanced-kubernetes
```

```
remote > kubectl create namespace limitrange-test
```

```
namespace/limitrange-test created
```

```
remote > kubectl create --namespace limitrange-test -f 08-week/limitrange/limitrange.yaml
```

```
limitrange/limitrange created
```

```
remote > kubectl describe --namespace limitrange-test limitranges limitrange
```

Name:	limitrange					
Namespace:	limitrange-test					
Type	Resource	Min	Max	Default Request	Default Limit	Max Limit/Request Ratio
----	-----	---	---	-----	-----	-----
Pod	cpu	50m	1	-	-	-
Pod	memory	5Mi	1Gi	-	-	-
Container	cpu	50m	1	100m	200m	4
Container	memory	5Mi	1Gi	10Mi	100Mi	10
PersistentVolumeClaim	storage	1Gi	10Gi	-	-	-

Example - LimitRange 2/2

- LimitRange가 설정된 namespace에 Pod를 생성하면 resource가 어떻게 설정될지 알아보자.

pod-nolimit.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nolimit

spec:
  containers:
  - name: main
    image: busybox
    command: ["dd", "if=/dev/zero", "of=/dev/null"]
```

```
remote > git clone https://github.com/whatwant-school/advanced-kubernetes.git
remote > cd advanced-kubernetes
```

```
remote > kubectl create --namespace limitrange-test -f 08-week/limitrange/pod-nolimit.yaml
```

```
pod/nolimit created
```

```
remote > kubectl exec -it --namespace limitrange-test nolimit -- top
```

```
Mem: 1158312K used, 98272K free, 2076K shrd, 47104K buff, 672428K cached
CPU:  2.7% usr  6.9% sys  0.0% nic 90.0% idle  0.0% io  0.0% irq  0.2% sirq
Load average: 0.44 0.31 0.28 3/383 24
```

PID	PPID	USER	STAT	VSZ	%VSZ	CPU	%CPU	COMMAND
1	0	root	R	1312	0.1	0	9.9	dd if /dev/zero of /dev/null
16	0	root	R	1320	0.1	0	0.0	top

Limits 값이 200인데
2 cpu 이니까, 10 ???

```
remote > kubectl describe --namespace limitrange-test pods nolimit
```

```
...
Limits:
  cpu:      200m
  memory:   100Mi
Requests:
  cpu:      100m
  memory:   10Mi
...
```




ResourceQuota

resourcequota.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota-all

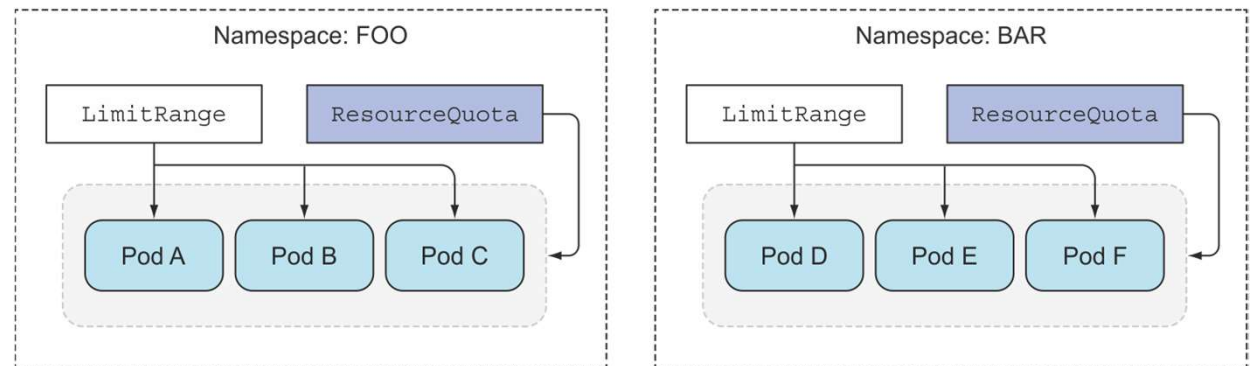
spec:
  scopes:
    - BestEffort
    - NotTerminating

  hard:
    requests.cpu: 400m
    requests.memory: 200Mi
    limits.cpu: 600m
    limits.memory: 500Mi

    requests.storage: 500Gi
    standard.storageclass.storage.k8s.io/requests.storage: 1Ti
    ssd.storageclass.storage.k8s.io/requests.storage: 300Gi
    ssd.storageclass.storage.k8s.io/persistentvolumeclaims: 2
    pods: 10
    replicationcontrollers: 5
    secrets: 10
    configmaps: 10
    persistentvolumeclaims: 5
    services: 5
    services.loadbalancers: 1
    services.nodeports: 2
```

- API 서버에 --enable-admission-plugins= 플래그 인수로 ResourceQuota가 있는 경우 활성화
- 각 Namespace에서 동작. 관리자는 각 네임스페이스에 대해 하나의 리소스 쿼터를 생성
- Namespace에 쿼터가 활성화된 경우 사용자는 CPU, MEMORY값에 request, limit을 지정해야 함
- 리소스 요구사항이 없는 Pod를 기본값으로 설정하려면, LimitRange admission controller를 사용하

※ 참고 : <https://velog.io/@idnnbi/kubernetes-Resource-Quotas>



※ 참고 : <https://livebook.manning.com/book/kubernetes-in-action/chapter-14/198>



Kubernetes

Monitoring

metrics

```
remote > kubectl get componentstatuses
```

Warning: v1 ComponentStatus is deprecated in v1.19+

NAME	STATUS	MESSAGE	ERROR
controller-manager	Healthy	ok	
scheduler	Healthy	ok	
etcd-0	Healthy	{"health":"true","reason":""}	

```
remote > kubectl get --namespace kube-system pods | grep metrics
```

kubernetes-metrics-scraper-6d49f96c97-9clml	1/1	Running	7 (5h34m ago)	12d
metrics-server-6978dd689f-dgst9	1/1	Running	25 (5h34m ago)	46d

```
remote > kubectl top nodes
```

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
master	139m	7%	1282Mi	93%
worker1	60m	3%	691Mi	79%
worker2	259m	13%	619Mi	71%

```
remote > kubectl top pods -A
```

NAMESPACE	NAME	CPU(cores)	MEMORY(bytes)
ingress-nginx	ingress-nginx-controller-778574f59b-pw25s	1m	95Mi
...			
limitrange-test	nolimit	201m	1Mi
metallb-system	controller-7dcc8764f4-565sc	1m	11Mi
metallb-system	speaker-87zsg	3m	18Mi
metallb-system	speaker-m4rg4	4m	20Mi
metallb-system	speaker-xr847	4m	21Mi

[별첨] ComponentStatus – 오류 해결

- `kubectl get componentstatuses` 결과가 정상적으로 나오지 않는 경우 아래와 같이 조치를 취하면 된다.

```
master > sudo nano /etc/kubernetes/manifests/kube-controller-manager.yaml
```

```
master > sudo nano /etc/kubernetes/manifests/kube-scheduler.yaml
```

```
...  
#- --port=0  
...
```

2개 파일 모두 `--port` 구문을 주석 처리하면 된다

master node에서 진행해야 한다.

```
remote > kubectl get componentstatuses
```

Warning: v1 ComponentStatus is deprecated in v1.19+

NAME	STATUS	MESSAGE	ERROR
controller-manager	Healthy	ok	
scheduler	Healthy	ok	
etcd-0	Healthy	{"health":"true","reason":""}	



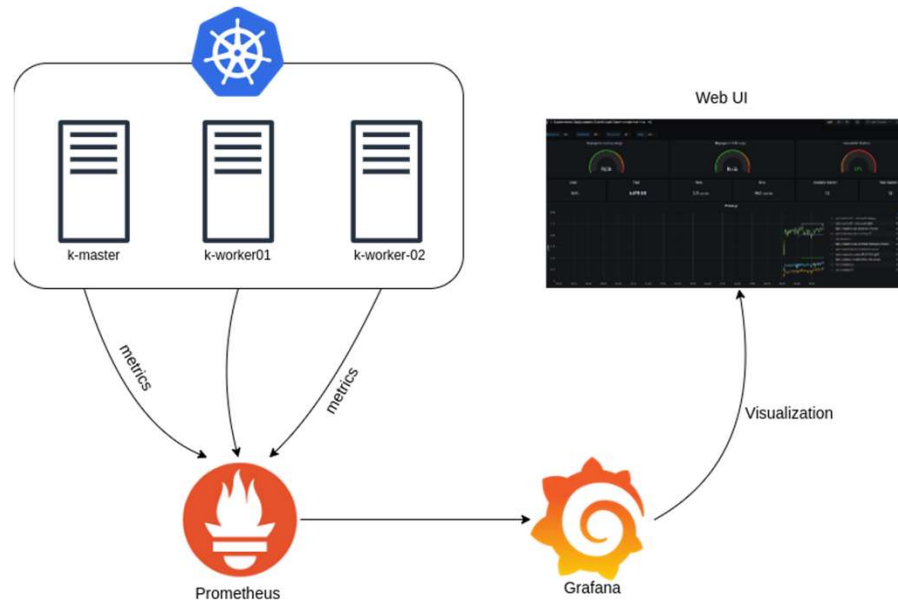
Break



Prometheus & Grafana

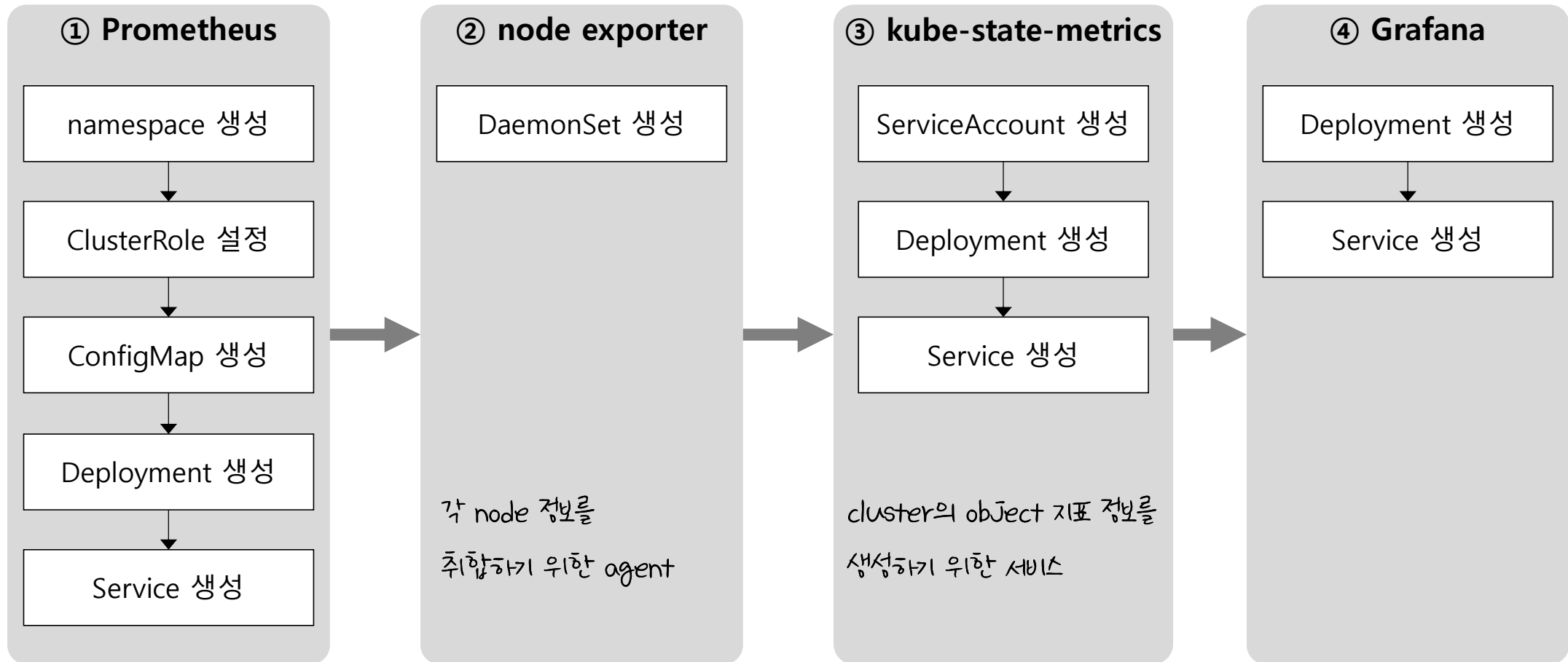
Prometheus

- SoundCloud社에서 만든 오픈소스 모니터링 툴
- go언어로 만들어졌으며, 지금은 독립된 오픈소스 프로젝트로 개발
- Kubernetes 환경에서 모니터링 하기 원하는 리소스로부터 metric을 수집하고 해당 metric을 이용해서 모니터링하는 기능을 제공
- 이상 증세가 발생했을 때 slack이나 여타 다른 webhook을 이용해서 알림을 주는 등 다양한 기능을 제공



※ 참고 : <https://velog.io/@pingping95/Kubernetes-Prometheus-Grafana-모니터링-설치-KVM>

Workflow - Overview





Workflow - ① Prometheus

① Prometheus

namespace 생성



ClusterRole 설정



ConfigMap 생성



Deployment 생성



Service 생성

```
remote > kubectl create namespace monitoring
```

```
namespace/monitoring created
```

Workflow - ① Prometheus

① Prometheus

namespace 생성

ClusterRole 설정

ConfigMap 생성

Deployment 생성

Service 생성

01-clusterrole.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus

rules:
  - apiGroups: [""]
    resources: ["nodes", "nodes/proxy", "services", "endpoints", "pods"]
    verbs: ["get", "list", "watch"]

  - apiGroups: ["extensions"]
    resources: ["ingresses"]
    verbs: ["get", "list", "watch"]

  - nonResourceURLs: ["/metrics"]
    verbs: ["get"]
```

02-clusterrolebinding.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus

roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus

subjects:
  - kind: ServiceAccount
    name: default
    namespace: monitoring
```

```
remote > git clone https://github.com/whatwant-school/advanced-kubernetes.git
remote > cd advanced-kubernetes
```

```
remote > kubectl create -f 08-week/monitoring/01-clusterrole.yaml
```

```
clusterrole.rbac.authorization.k8s.io/prometheus created
```

```
remote > kubectl create -f 08-week/monitoring/02-clusterrolebinding.yaml
```

```
clusterrolebinding.rbac.authorization.k8s.io/prometheus created
```


Workflow - ① Prometheus

① Prometheus

namespace 생성

ClusterRole 설정

ConfigMap 생성

Deployment 생성

Service 생성

03-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-server-conf
  labels:
    name: prometheus-server-conf
  namespace: monitoring

data:
  prometheus.rules: |-
    groups:
      - name: container memory alert
        rules:
          - alert: container memory usage rate is very high( > 55%)
    ...

  prometheus.yml: |-
    global:
      scrape_interval: 5s
      evaluation_interval: 5s
    ...
```

Metric에 대한 Alarm 조건

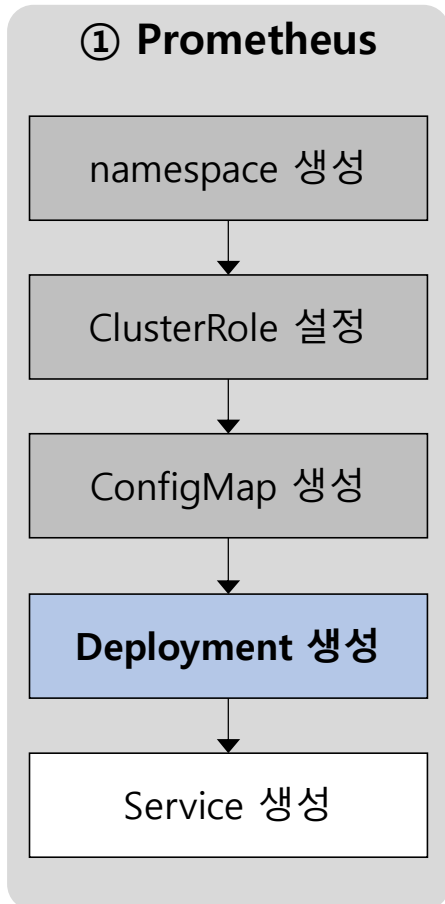
Metric의 종류, 수집 주기

```
remote > git clone https://github.com/whatwant-school/advanced-kubernetes.git
remote > cd advanced-kubernetes
```

```
remote > kubectl create -f 08-week/monitoring/03-configmap.yaml
```

```
configmap/prometheus-server-conf created
```

Workflow - ① Prometheus



04-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus-deployment
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-server
  template:
    metadata:
      labels:
        app: prometheus-server
    spec:
      containers:
        - name: prometheus
          image: prom/prometheus:latest
```

```
args:
  - "--config.file=/etc/prometheus/prometheus.yml"
  - "--storage.tsdb.path=/prometheus/"

ports:
  - containerPort: 9090

volumeMounts:
  - name: prometheus-config-volume
    mountPath: /etc/prometheus/
  - name: prometheus-storage-volume
    mountPath: /prometheus/

volumes:
  - name: prometheus-config-volume
    configMap:
      defaultMode: 420
      name: prometheus-server-conf
  - name: prometheus-storage-volume
    emptyDir: {}
```

```
remote > git clone https://github.com/whatwant-school/advanced-kubernetes.git
remote > cd advanced-kubernetes
```

```
remote > kubectl create -f 08-week/monitoring/04-deployment.yaml
```

```
deployment.apps/prometheus-deployment created
```

Workflow - ① Prometheus

① Prometheus

namespace 생성



ClusterRole 설정



ConfigMap 생성



Deployment 생성



Service 생성

05-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: prometheus
  namespace: monitoring

annotations:
  prometheus.io/scrape: 'true'
  prometheus.io/port: '9090'

spec:
  type: LoadBalancer

  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 9090

  selector:
    app: prometheus-server
```

```
remote > git clone https://github.com/whatwant-school/advanced-kubernetes.git
remote > cd advanced-kubernetes
```

```
remote > kubectl create -f 08-week/monitoring/05-service.yaml
```

```
service/prometheus created
```



Workflow - ② node exporter

② node exporter

DaemonSet 생성

06-daemonset.yaml

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: node-exporter
  labels:
    k8s-app: node-exporter
  namespace: monitoring
spec:
  selector:
    matchLabels:
      k8s-app: node-exporter
  template:
    metadata:
      labels:
        k8s-app: node-exporter
    spec:
      containers:
        - image: prom/node-exporter
          name: node-exporter
```

```
remote > git clone https://github.com/whatwant-school/advanced-kubernetes.git
remote > cd advanced-kubernetes
```

```
remote > kubectl create -f 08-week/monitoring/06-daemonset.yaml
```

```
daemonset.apps/node-exporter created
```



Workflow - ③ kube-state-metrics

③ kube-state-metrics

ServiceAccount 생성



Deployment 생성



Service 생성

08-serviceaccount.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kube-state-metrics
  namespace: kube-system
```

10-clusterrolebinding.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kube-state-metrics

roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kube-state-metrics
subjects:
- kind: ServiceAccount
  name: kube-state-metrics
  namespace: kube-system
```

09-clusterrole.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kube-state-metrics

rules:
- apiGroups: [""]
  resources: ["configmaps", "secrets", "nodes", "pods", "services", ...]
  verbs: ["list", "watch"]

- apiGroups: ["extensions"]
  resources: ["daemonsets", "deployments", "replicasets", "ingresses"]
  verbs: ["list", "watch"]

- apiGroups: ["apps"]
  resources: ["statefulsets", "daemonsets", "deployments", "replicasets"]
  verbs: ["list", "watch"]

- apiGroups: ["batch"]
  ...
```

```
remote > kubectl create -f 08-week/monitoring/08-serviceaccount.yaml
serviceaccount/kube-state-metrics created
```

```
remote > kubectl create -f 08-week/monitoring/09-clusterrole.yaml
clusterrole.rbac.authorization.k8s.io/kube-state-metrics created
```

```
remote > kubectl create -f 08-week/monitoring/10-clusterrolebinding.yaml
clusterrolebinding.rbac.authorization.k8s.io/kube-state-metrics created
```

Workflow - ③ kube-state-metrics

③ kube-state-metrics

ServiceAccount 생성



Deployment 생성



Service 생성

11-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment

metadata:
  name: kube-state-metrics
  labels:
    app: kube-state-metrics
  namespace: kube-system
```

```
spec:
  replicas: 1
```

```
selector:
  matchLabels:
    app: kube-state-metrics
```

```
template:
  metadata:
    labels:
      app: kube-state-metrics
```

```
spec:
  containers:
    - image: quay.io/coreos/kube-state-metrics:v1.8.0
      name: kube-state-metrics
```

```
ports:
  - containerPort: 8080
    name: http-metrics
  - containerPort: 8081
    name: telemetry
```

```
livenessProbe:
  httpGet:
    path: /healthz
    port: 8080
  initialDelaySeconds: 5
  timeoutSeconds: 5
```

```
readinessProbe:
  httpGet:
    path: /
    port: 8081
  initialDelaySeconds: 5
  timeoutSeconds: 5
```

```
nodeSelector:
  kubernetes.io/os: linux
```

```
serviceAccountName: kube-state-metrics
```

```
remote > kubectl create -f 08-week/monitoring/11-deployment.yaml
```

```
deployment.apps/kube-state-metrics created
```


Workflow - ③ kube-state-metrics

③ kube-state-metrics

ServiceAccount 생성



Deployment 생성



Service 생성

12-service.yaml

```
apiVersion: v1
kind: Service
```

```
metadata:
  name: kube-state-metrics
  labels:
    app: kube-state-metrics
  namespace: kube-system
```

```
spec:
  clusterIP: None
```

Headless Service로 생성

```
ports:
- name: http-metrics
  port: 8080
  targetPort: http-metrics
```

```
- name: telemetry
  port: 8081
  targetPort: telemetry
```

```
selector:
  app: kube-state-metrics
```

```
remote > kubectl create -f 08-week/monitoring/12-service.yaml
```

```
service/kube-state-metrics created
```



Workflow - ④ Grafana

④ Grafana

Deployment 생성



Service 생성

13-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment

metadata:
  name: grafana
  namespace: monitoring

spec:
  replicas: 1

  selector:
    matchLabels:
      app: grafana

  template:
    metadata:
      name: grafana
      labels:
        app: grafana

    spec:
      containers:
        - name: grafana
          image: grafana/grafana:latest
```

```
ports:
  - name: grafana
    containerPort: 3000

env:
  - name: GF_SERVER_HTTP_PORT
    value: "3000"
  - name: GF_AUTH_BASIC_ENABLED
    value: "false"
  - name: GF_AUTH_ANONYMOUS_ENABLED
    value: "true"
  - name: GF_AUTH_ANONYMOUS_ORG_ROLE
    value: Admin
  - name: GF_SERVER_ROOT_URL
    value: /
```

```
remote > kubectl create -f 08-week/monitoring/13-deployment.yaml
```

```
deployment.apps/grafana created
```

Workflow - ④ Grafana

14-service.yaml

④ Grafana

Deployment 생성



Service 생성

```
apiVersion: v1
kind: Service
metadata:
  name: grafana
  annotations:
    prometheus.io/scrape: 'true'
    prometheus.io/port: '3000'
  namespace: monitoring
spec:
  type: LoadBalancer

  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 3000

  selector:
    app: grafana
```

```
remote > kubectl create -f 08-week/monitoring/14-service.yaml
```

```
service/grafana created
```

```
remote > kubectl get services --namespace monitoring
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
grafana	LoadBalancer	10.233.33.18	192.168.100.241	80:30119/TCP	48s
prometheus	LoadBalancer	10.233.50.69	192.168.100.240	80:31407/TCP	132m



Prometheus

The screenshot shows the Prometheus Time Series web interface. The browser tab is titled "Prometheus Time Series" and the address bar shows the URL "192.168.100.240/graph?g0.expr=&g0.tab=1&g0.stacked=0&g0.show_exemplars=0&g0.range_input=1h". The interface has a dark header with the "Prometheus" logo and navigation links for "Alerts", "Graph", "Status", "Help", and "Classic UI". On the right of the header are icons for settings, a dark mode toggle, and a user profile. Below the header, there are several checkboxes: "Use local time" (unchecked), "Enable query history" (unchecked), "Enable autocomplete" (checked), "Enable highlighting" (checked), and "Enable linter" (checked). A search bar with a magnifying glass icon contains the placeholder text "Expression (press Shift+Enter for newlines)". To the right of the search bar is a blue "Execute" button. Below the search bar, there are two tabs: "Table" and "Graph", with "Graph" being the active tab. Below the tabs is a range selector with left and right arrow buttons and the text "Evaluation time". The main content area is empty and displays the message "No data queried yet". In the bottom right corner of the main area is a blue link "Remove Panel". In the bottom left corner is a blue button "Add Panel".

Prometheus Time Series

192.168.100.240/graph?g0.expr=&g0.tab=1&g0.stacked=0&g0.show_exemplars=0&g0.range_input=1h

Prometheus Alerts Graph Status Help Classic UI

☐ Use local time ☐ Enable query history ☒ Enable autocomplete ☒ Enable highlighting ☒ Enable linter

Search Expression (press Shift+Enter for newlines) Execute

Table Graph

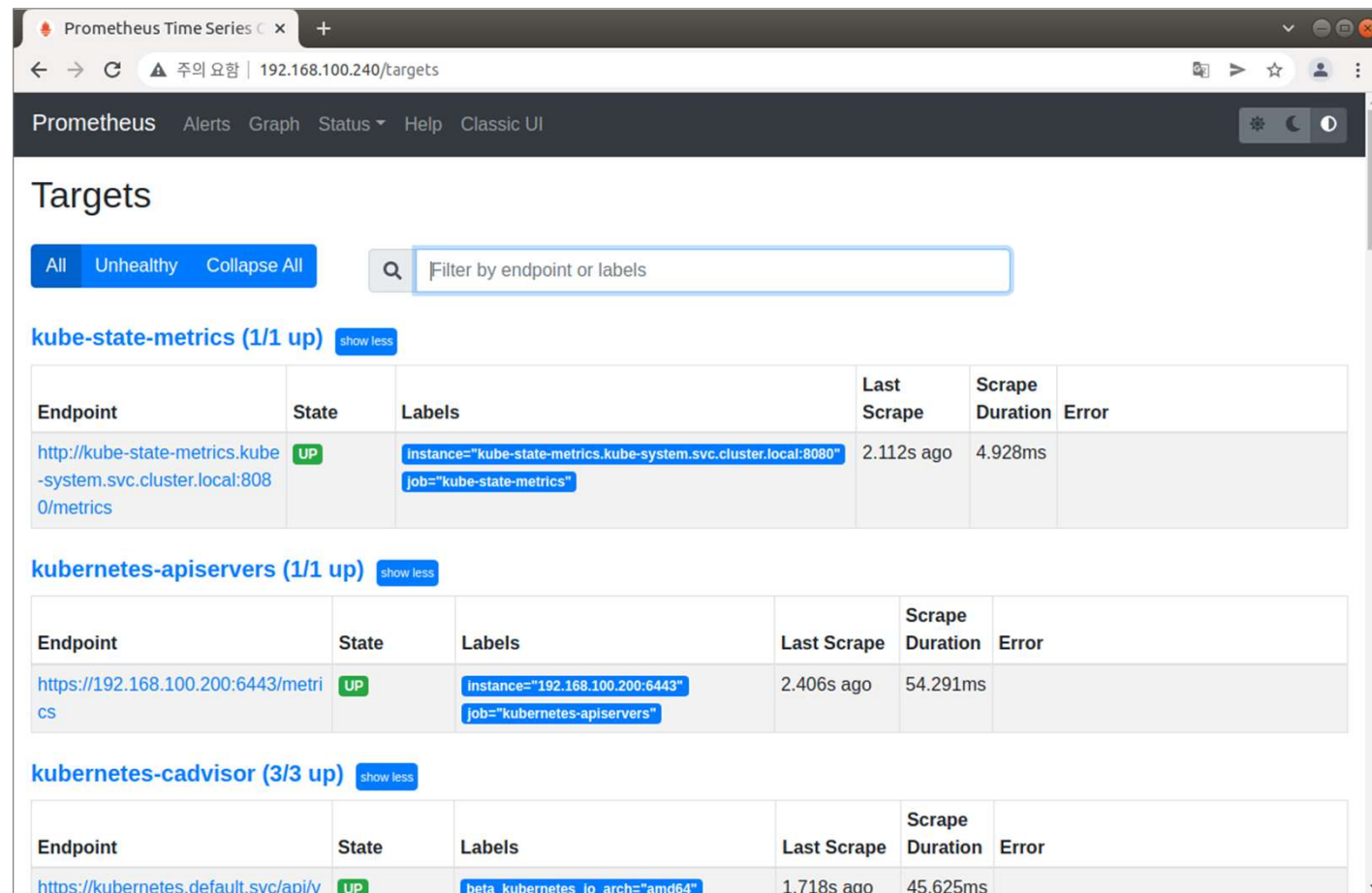
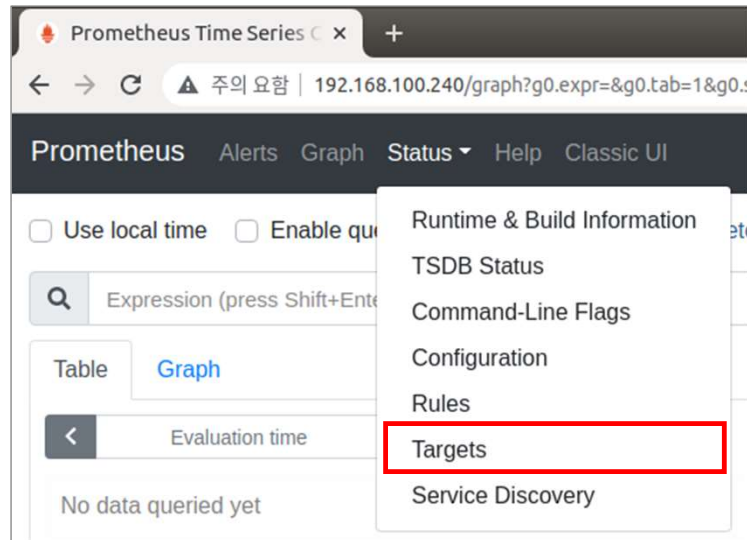
Evaluation time

No data queried yet

Remove Panel

Add Panel

Prometheus : status - Targets

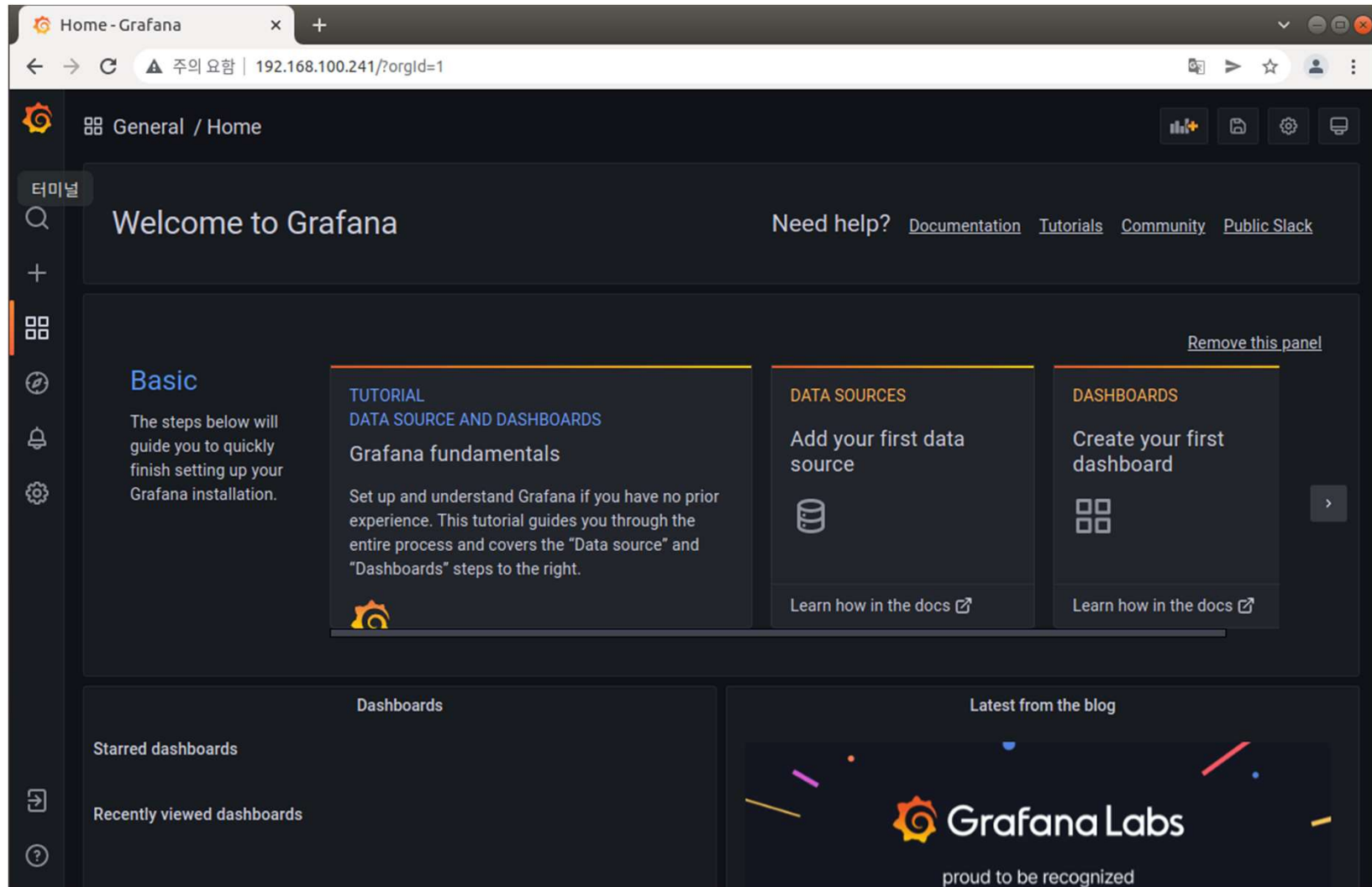


A screenshot of the Prometheus 'Targets' page. The page shows a list of targets with columns for Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error. The targets are grouped by job name: kube-state-metrics (1/1 up), kubernetes-apiservers (1/1 up), and kubernetes-cadvisor (3/3 up). Each group has a 'show less' button. The 'kube-state-metrics' group shows one target with a green 'UP' state. The 'kubernetes-apiservers' group shows one target with a green 'UP' state. The 'kubernetes-cadvisor' group shows one target with a green 'UP' state.

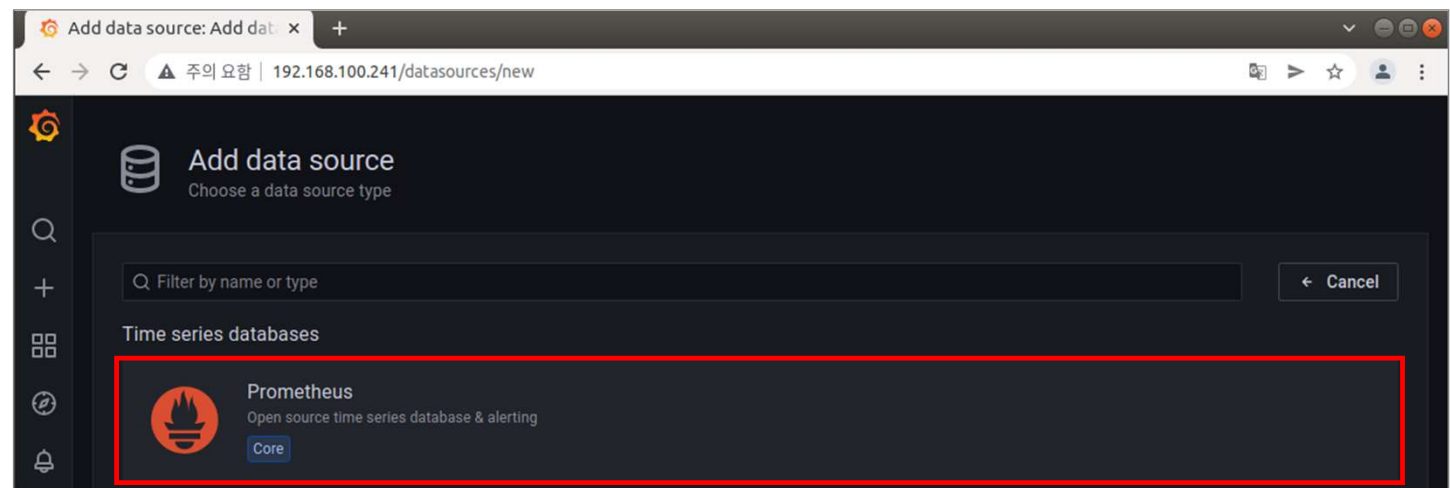
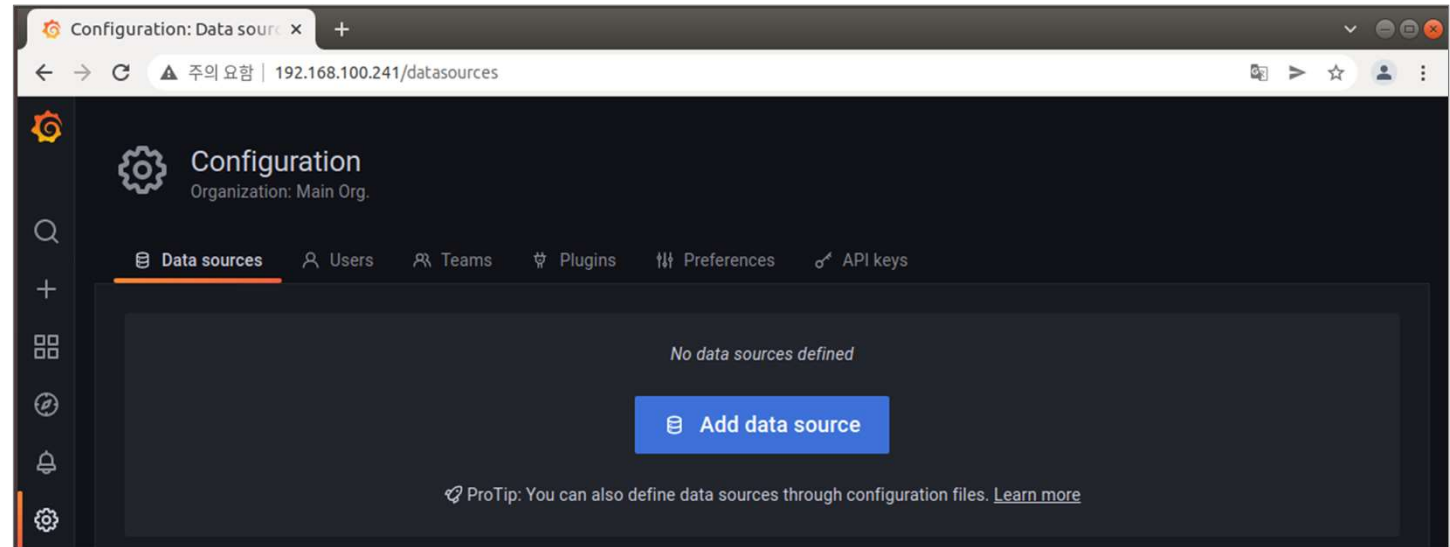
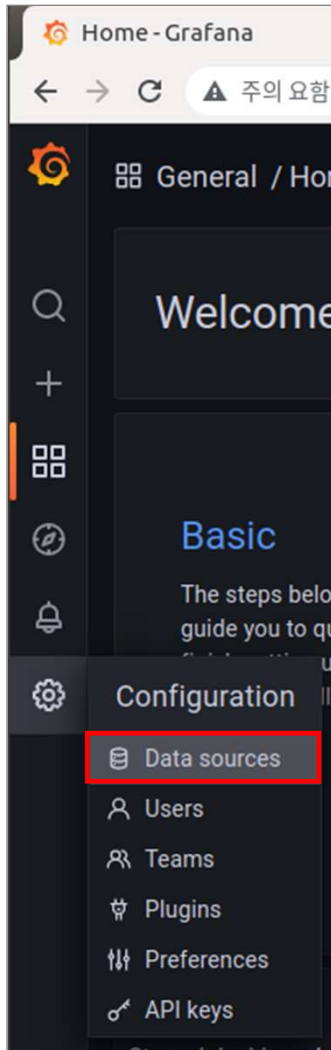
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
kube-state-metrics (1/1 up) show less					
http://kube-state-metrics.kube-system.svc.cluster.local:8080/metrics	UP	instance="kube-state-metrics.kube-system.svc.cluster.local:8080" job="kube-state-metrics"	2.112s ago	4.928ms	
kubernetes-apiservers (1/1 up) show less					
https://192.168.100.200:6443/metrics	UP	instance="192.168.100.200:6443" job="kubernetes-apiservers"	2.406s ago	54.291ms	
kubernetes-cadvisor (3/3 up) show less					
https://kubernetes.default.svc/api/v1	UP	beta kubernetes.io arch="amd64"	1.718s ago	45.625ms	



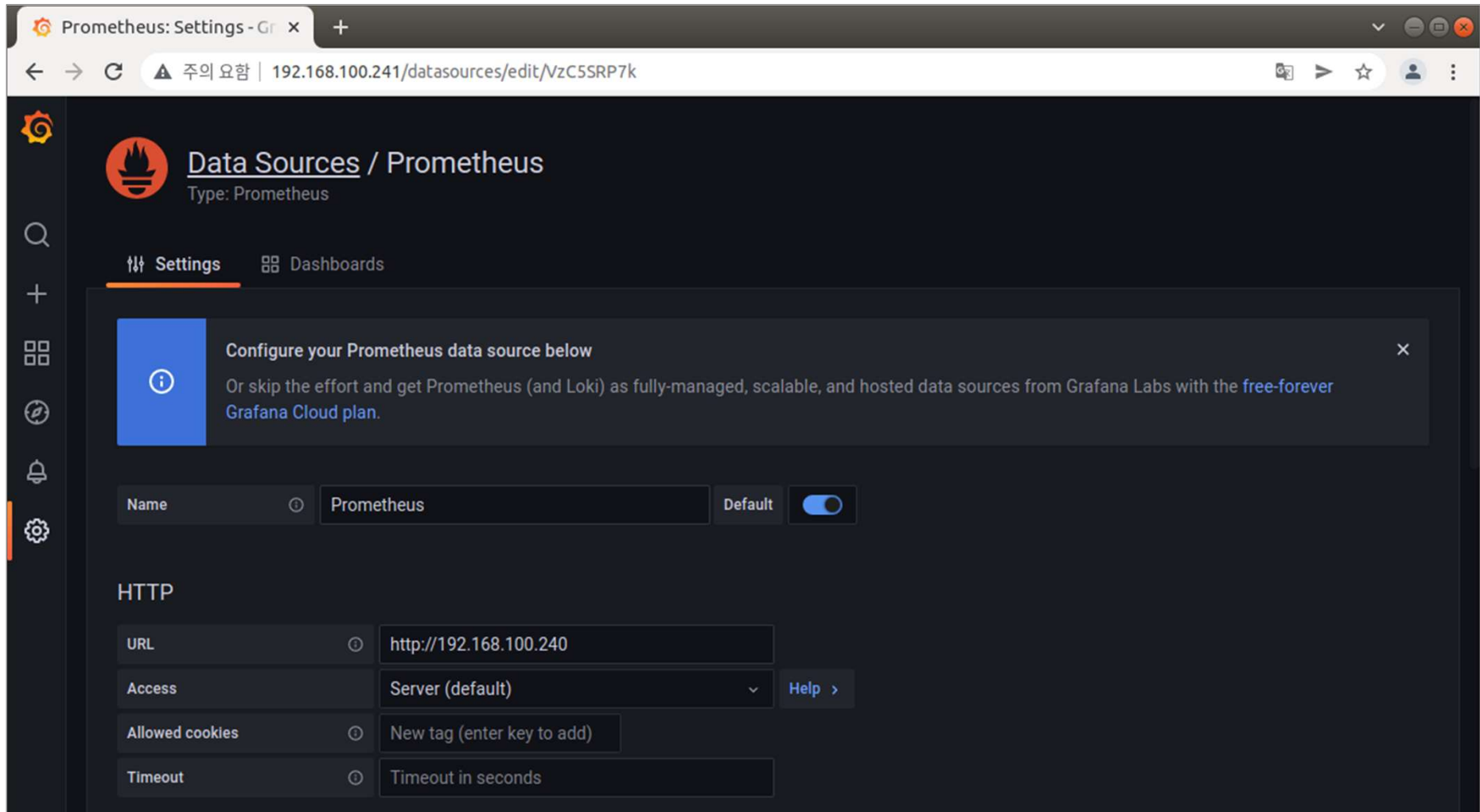
Grafana



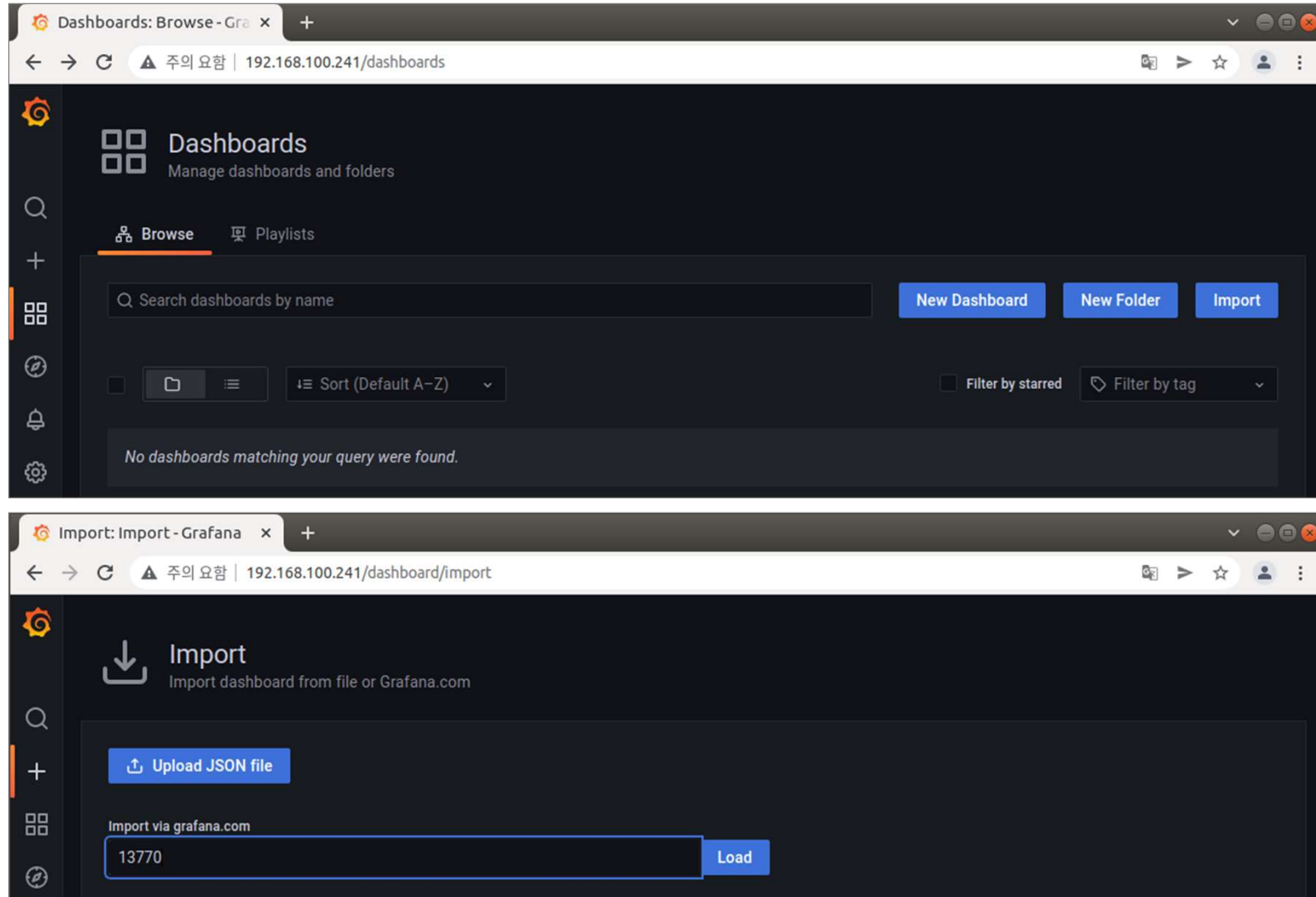
Grafana : Configuration - Data sources - 1/2



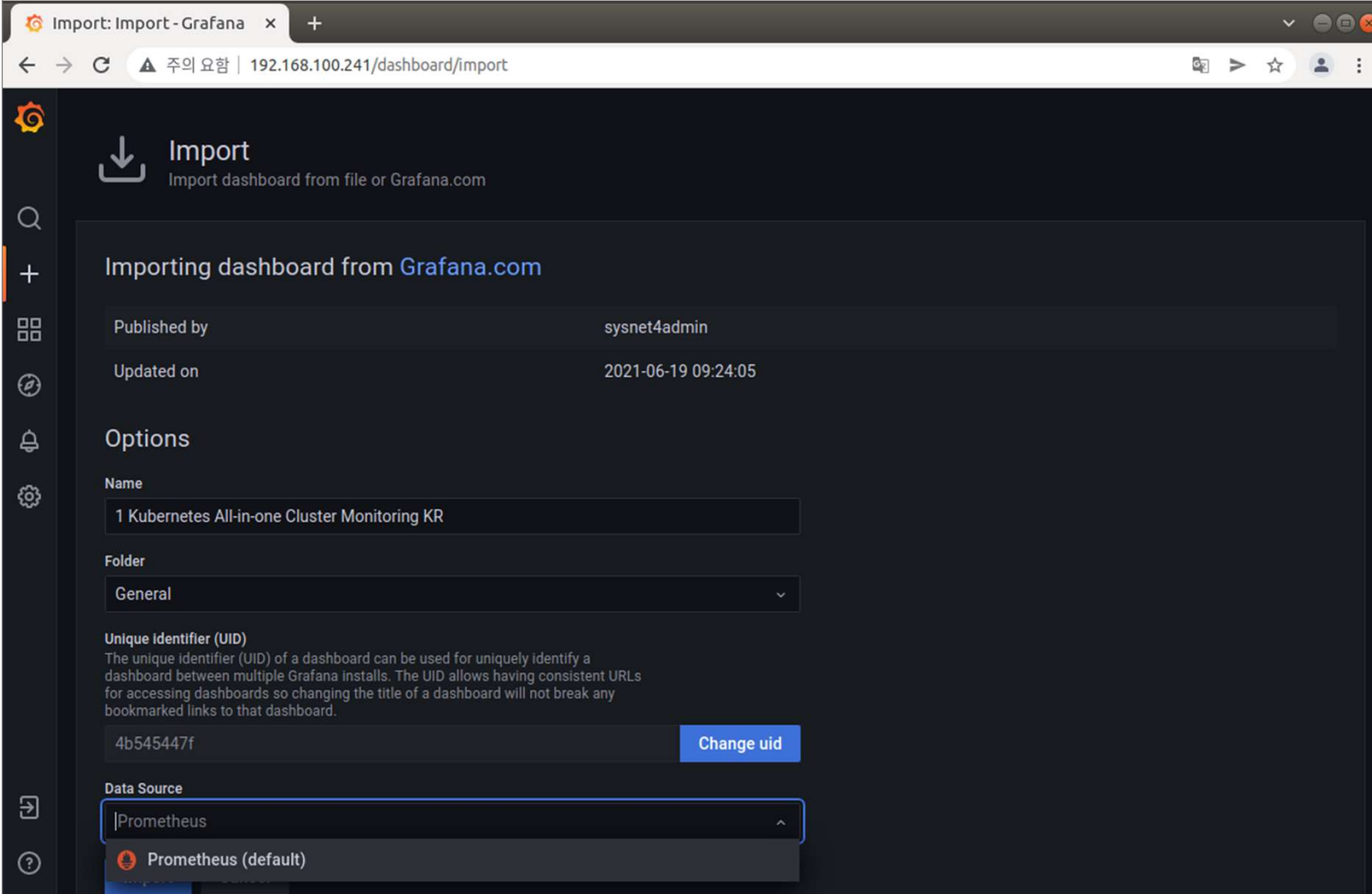
Grafana : Configuration - Data sources - 2/2



Grafana : Dashboard - Import - 1/2



Grafana : Dashboard - Import - 2/2



The screenshot shows the Grafana 'Import' page in a web browser. The browser's address bar displays '192.168.100.241/dashboard/import'. The page title is 'Import: Import - Grafana'. The main heading is 'Import', with a subtext 'Import dashboard from file or Grafana.com'. Below this, the section 'Importing dashboard from Grafana.com' is active. It shows metadata for a dashboard: 'Published by' is 'sysnet4admin' and 'Updated on' is '2021-06-19 09:24:05'. The 'Options' section includes a 'Name' field with the value '1 Kubernetes All-in-one Cluster Monitoring KR', a 'Folder' dropdown menu set to 'General', and a 'Unique Identifier (UID)' field with the value '4b545447f'. A 'Change uid' button is next to the UID field. Below the UID field, there is a 'Data Source' dropdown menu with 'Prometheus' selected, and a list of available data sources including 'Prometheus (default)'.

Import

Import dashboard from file or Grafana.com

Importing dashboard from [Grafana.com](#)

Published by sysnet4admin

Updated on 2021-06-19 09:24:05

Options

Name

1 Kubernetes All-in-one Cluster Monitoring KR

Folder

General

Unique Identifier (UID)

The unique identifier (UID) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The UID allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

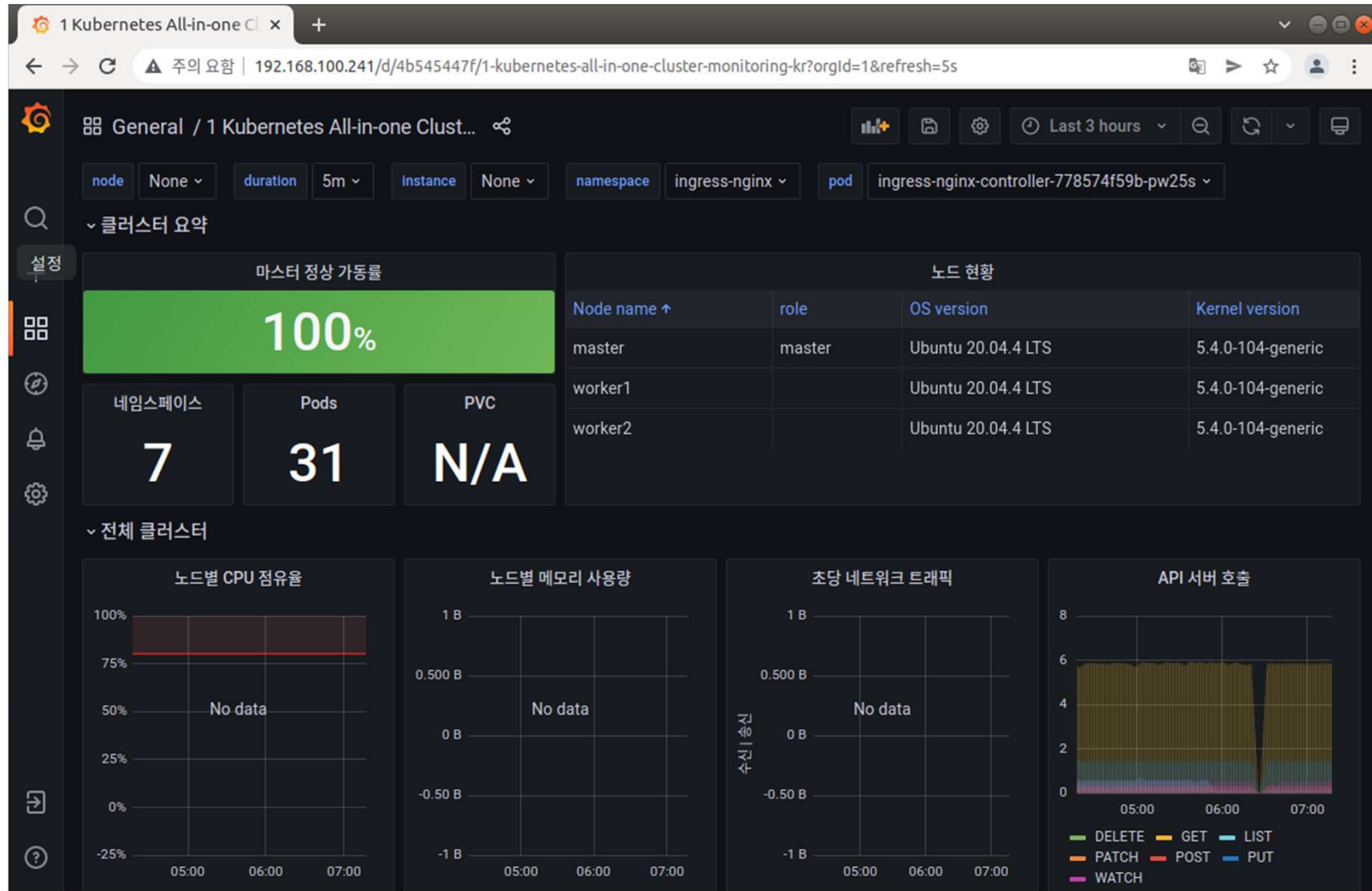
4b545447f [Change uid](#)

Data Source

Prometheus

Prometheus (default)

Grafana : Dashboard







Kubernetes

PodSecurityPolicy

PodSecurityPolicy is ...

- Pod의 Security 관련 항목들을 제어하는 cluster-레벨의 리소스
- K8s v1.21 deprecated / v1.25 removed

Control Aspect	Field Names
특권을 가진(privileged) 컨테이너의 실행	<code>privileged</code>
호스트 네임스페이스의 사용	<code>hostPID</code> , <code>hostIPC</code>
호스트 네트워킹과 포트의 사용	<code>hostNetwork</code> , <code>hostPorts</code>
볼륨 유형의 사용	<code>volumes</code>
호스트 파일시스템의 사용	<code>allowedHostPaths</code>
특정 FlexVolume 드라이버의 허용	<code>allowedFlexVolumes</code>
파드 볼륨을 소유한 FSGroup 할당	<code>fsGroup</code>
읽기 전용 루트 파일시스템 사용 필요	<code>readOnlyRootFilesystem</code>
컨테이너의 사용자 및 그룹 ID	<code>runAsUser</code> , <code>runAsGroup</code> , <code>supplementalGroups</code>
루트 특권으로의 에스컬레이션 제한	<code>allowPrivilegeEscalation</code> , <code>defaultAllowPrivilegeEscalation</code>
리눅스 기능	<code>defaultAddCapabilities</code> , <code>requiredDropCapabilities</code> , <code>allowedCapabilities</code>
컨테이너의 SELinux 컨텍스트	<code>seLinux</code>
컨테이너에 허용된 Proc 마운트 유형	<code>allowedProcMountTypes</code>
컨테이너가 사용하는 AppArmor 프로파일	어노테이션
컨테이너가 사용하는 seccomp 프로파일	어노테이션
컨테이너가 사용하는 sysctl 프로파일	<code>forbiddenSysctls</code> , <code>allowedUnsafeSysctls</code>

※ 참고 : <https://kubernetes.io/ko/docs/concepts/policy/pod-security-policy/>

K8s : PodSecurityPolicy Hands-On

PodSecurityPolicy Create

PodSecurityPolicy Admission Controller를 활성화 해야 하는데,
활성화 되었을 때 PodSecurityPolicy 리소스가 없을 경우 Pod 생성 및 업데이트 할 때 정상적으로 동작하지 않을 수 있다.
즉, PodSecurityPolicy Admission Controller 활성화 하기 전에 PodSecurityPolicy를 1개 이상 미리 등록해 놓는 것이 좋다.

podsecuritypolicy.yaml

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy

metadata:
  name: privileged
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'

spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
    - '*'
  volumes:
    - '*'
  hostNetwork: true
  hostPorts:
    - min: 0
      max: 65535
  hostIPC: true
  hostPID: true
```

```
runAsUser:
  rule: 'RunAsAny'
seLinux:
  rule: 'RunAsAny'
supplementalGroups:
  rule: 'RunAsAny'
fsGroup:
  rule: 'RunAsAny'
```

```
remote > cd advanced-kubernetes/inbox/psp

remote > kubectl create -f ./podsecuritypolicy.yaml

remote > kubectl get podsecuritypolicies -o wide
```

ClusterRole / ClusterRoleBinding Create

대부분 RBAC 권한 체계를 사용하고 있을 것이고,

그렇다면, 정책 사용 권한을 부여하기 위해 ClusterRole / ClusterRoleBinding 리소스를 생성해 놓자 !

clusterrole.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole

metadata:
  name: privileged-psp

rules:
- apiGroups: ['policy']
  resources: ['podsecuritypolicies']
  verbs:     ['use']
  resourceNames:
  - privileged
```

clusterrolebinding.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding

metadata:
  name: privileged-psp-system-authenticated

roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: privileged-psp

subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:authenticated
```

```
remote > cd advanced-kubernetes/inbox/psp
```

```
remote > kubectl create -f ./clusterrole.yaml
```

```
remote > kubectl create -f ./clusterrolebinding.yaml
```

```
remote > kubectl get clusterroles -o wide
```

```
remote > kubectl get clusterrolebindings -o wide
```

PodSecurityPolicy Admission Controller Enable - 1/2

Admission Controller 중에서 'PodSecurityPolicy'를 추가해주어야 한다(enable).
master node로 접속해서 YAML 파일을 수정하고 반영해주자 !

```
remote > ssh vagrant@192.168.100.200
```

```
master > cd /etc/kubernetes/manifests/
```

```
master > sudo nano ./kube-apiserver.yaml
```

```
...
spec:
  containers:
    - command:
      - kube-apiserver
      - --advertise-address=192.168.100.200
      - --allow-privileged=true
      - --anonymous-auth=True
      - --apiserver-count=1
      - --authorization-mode=Node,RBAC
      - --bind-address=0.0.0.0
      - --client-ca-file=/etc/kubernetes/ssl/ca.crt
      - --default-not-ready-toleration-seconds=300
      - --default-unreachable-toleration-seconds=300
      - --enable-admission-plugins=NodeRestriction,PodSecurityPolicy
      - --enable-aggregator-routing=False
      - --enable-bootstrap-token-auth=true
      - --endpoint-reconciler-type=lease
    ...
```

PodSecurityPolicy Admission Controller Enable - 2/2

```
master > sudo kubectl apply -f kube-apiserver.yaml
```

```
master > kubectl describe pod --namespace kube-system kube-apiserver-master
```

```
Name:                kube-apiserver-master
Namespace:           kube-system
...
Containers:
  kube-apiserver:
    Container ID:  docker://4c7d91c3358b4e770b0d3ddc5e3a742ebfd28c775d897ab3062afb9c24b9bbe7
    Image:         k8s.gcr.io/kube-apiserver:v1.20.7
    Image ID:      docker-pullable://k8s.gcr.io/kube-apiserver@sha256:5ab3d676c426bfb272fb7605e6978b90d5676913636a6105688862849961386f
    Port:         <none>
    Host Port:    <none>
    Command:
      kube-apiserver
      --advertise-address=192.168.100.200
      --allow-privileged=true
      --anonymous-auth=True
      --apiserver-count=1
      --authorization-mode=Node,RBAC
      --bind-address=0.0.0.0
      --client-ca-file=/etc/kubernetes/ssl/ca.crt
      --default-not-ready-toleration-seconds=300
      --default-unreachable-toleration-seconds=300
      --enable-admission-plugins=NodeRestriction,PodSecurityPolicy
      --enable-aggregator-routing=False
      --enable-bootstrap-token-auth=true
    ...
```

Privileged Example - 1/2

privileged 정책이 잘 적용이 되는지 확인 해 보자

pod-non-privileged.yaml

```
apiVersion: v1
kind: Pod

metadata:
  name: pod-non-privileged

spec:
  containers:
    - name: main
      image: alpine
      command: ["/bin/sleep", "999999"]
```

pod-privileged.yaml

```
apiVersion: v1
kind: Pod

metadata:
  name: pod-privileged

spec:
  containers:
    - name: main
      image: alpine
      command: ["/bin/sleep", "999999"]

  securityContext:
    privileged: true
```

```
remote > cd advanced-kubernetes/inbox/psp

remote > kubectl create -f ./pod-non-privileged.yaml

remote > kubectl create -f ./pod-privileged.yaml
```


Privileged Example - 2/2

'privileged: true' 적용한 경우, host의 내역까지 모두 나오고 있는 것을 볼 수 있다.

```
remote > kubectl exec -it pod-non-privileged -- ls /dev
```

core	null	shm	termination-log
fd	ptmx	stderr	tty
full	pts	stdin	urandom
mqueue	random	stdout	zero

```
remote > kubectl exec -it pod-privileged -- ls /dev
```

autofs	sda2	tty41	ttyS28
bsg	sda3	tty42	ttyS29
btrfs-control	sg0	tty43	ttyS3
bus	sg1	tty44	ttyS30
core	shm	tty45	ttyS31
cpu	snapshot	tty46	ttyS4
cpu_dma_latency	snd	tty47	ttyS5
cuse	sr0	tty48	ttyS6
dm-0	stderr	tty49	ttyS7
dri	stdin	tty5	ttyS8
ecryptfs	stdout	tty50	ttyS9
fb0	termination-log	tty51	ttyprintk
fd	tty	tty52	udmabuf
full	tty0	tty53	uhid
fuse	tty1	tty54	uinput
hidraw0	tty10	tty55	urandom
hpet	tty11	tty56	userio
hwrng	tty12	tty57	vboxguest
i2c-0	tty13	tty58	vboxuser
input	tty14	tty59	vcs
kmsg	tty15	tty6	vcs1
lightnvm	tty16	tty60	vcs2
loop-control	tty17	tty61	vcs3
...			



<https://kahoot.it/>

[Score]

이민준 (11)

김남형 (6)

이혜정 (6)

정현찬 (4)

박남준 (3)

김상호 (2)

이원준 (2)

김정은 (1)