10 week

Resource Management

Resources of Nodes – 1/2

- 사용할 수 있는 Node의 Resource 확인

```
❯ kubectl describe nodes
Name:               master-stg
. . .


Name:               worker1
. . .
Capacity:
  cpu:                2
  ephemeral-storage:  25155844Ki
  hugepages-2Mi:      0
  memory:             4030620Ki
  pods:               110
Allocatable:
  cpu:                1900m
  ephemeral-storage:  23183625793
  hugepages-2Mi:      0
  memory:             3666076Ki
  pods:               110
. . .
```

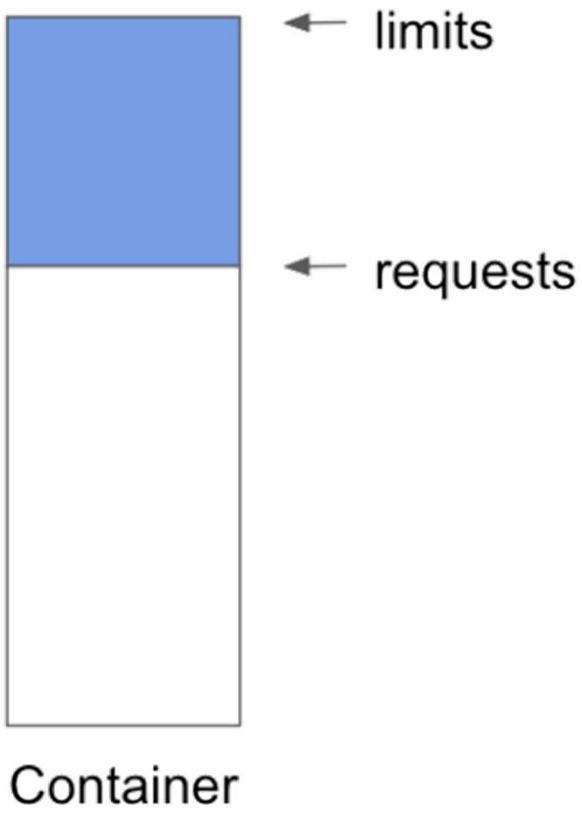
Resources of Nodes – 2/2

```
Non-terminated Pods:          (8 in total)
  Namespace                   Name                                        CPU Requests  CPU Limits  Memory Requests  Memory Limits  AGE
  ---------                   ----                                        ------------  ----------  ---------------  -------------  ---
  ingress-nginx               ingress-nginx-controller-4cf6w              0 (0%)        0 (0%)      0 (0%)           0 (0%)         51d
  kube-system                 calico-node-b5vl8                           150m (7%)     300m (15%)  64M (1%)         500M (13%)     51d
  kube-system                 coredns-657959df74-zvtrd                    100m (5%)     0 (0%)      70Mi (1%)        170Mi (4%)     38d
  kube-system                 kube-proxy-wssfs                            0 (0%)        0 (0%)      0 (0%)           0 (0%)         51d
  kube-system                 kubernetes-dashboard-7ddc76ff5f-69vvc       50m (2%)      100m (5%)   64M (1%)         256M (6%)      51d
  kube-system                 kubernetes-metrics-scraper-64db6db887-zkx7r  0 (0%)        0 (0%)      0 (0%)           0 (0%)         51d
  kube-system                 nginx-proxy-worker1                         25m (1%)      0 (0%)      32M (0%)         0 (0%)         51d
  kube-system                 nodelocaldns-69mkv                          100m (5%)     0 (0%)      70Mi (1%)        170Mi (4%)     51d
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests        Limits
  --------           --------        ------
  cpu                425m (22%)      400m (21%)
  memory             306800640 (8%)  1112515840 (29%)
  ephemeral-storage  0 (0%)          0 (0%)
  hugepages-2Mi      0 (0%)          0 (0%)
Events:              <none>


Name:               worker2
. . .
```

Requests & Limits

- requests : container가 생성될 때 요청하는 리소스

- limits : container가 생성되고 CPU/Memory가 더 필요한 경우 추가로 더 사용할 수 있는 리소스



※ 참고 : https://bcho.tistory.com/1291

Units

- CPU : ms (밀리 세컨드), 1000ms = 1 vCore (가상 CPU 코어)

  . 1 = 1000ms, 0.5 = 500ms

- Memory : Mi (MiB, 메비바이트), 1 MiB = 1024 KiB

※ 참고 : https://bcho.tistory.com/1291

resource requests

- dd : 최대 CPU 소비, but 1 thread

- 실행환경 2 cpu, ∴ 50%

→ requests cpu 200m, but use 1000m

01-requests-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: requests-pod
spec:
  containers:
  - image: busybox
    command: ["dd", "if=/dev/zero", "of=/dev/null"]
    name: main
    resources:
      requests:
        cpu: 200m
        memory: 10Mi
```

```
> kubectl create -f 01-requests-pod.yaml
pod/requests-pod created

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

```
Mem: 1977492K used, 2053128K free, 3400K shrd, 79856K buff, 1283004K cached
CPU: 12.1% usr 38.7% sys  0.0% nic 49.0% idle  0.0% io  0.0% irq  0.0% sirq
Load average: 1.34 1.33 1.13 4/637 12
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
    1     0 root     R     1308  0.0   0 50.8 dd if /dev/zero of /dev/null
    7     0 root     R     1316  0.0   1  0.0 top
```

resource limits

- 실행환경 2 cpu, limits cpu 200m

→ ∴ 10%

02-limits-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: limits-pod
spec:
  containers:
  - image: busybox
    command: ["dd", "if=/dev/zero", "of=/dev/null"]
    name: main
    resources:
      limits:
        cpu: 200m
        memory: 10Mi
```

```
❯ kubectl create -f 02-limits-pod.yaml
pod/limits-pod created

❯ kubectl exec -it limits-pod -- top
```

```
Mem: 1981580K used, 2049040K free, 3568K shrd, 82440K buff, 1284664K cached
CPU: 14.3% usr 47.9% sys  0.0% nic 37.2% idle  0.0% io  0.0% irq  0.3% sirq
Load average: 1.45 1.43 1.28 3/662 11
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
    1     0 root     R     1308  0.0   0  9.8 dd if /dev/zero of /dev/null
    6     0 root     R     1316  0.0   0  0.0 top
```

requests를 설정하지 않으면, limits 값으로 requests 값 설정됨

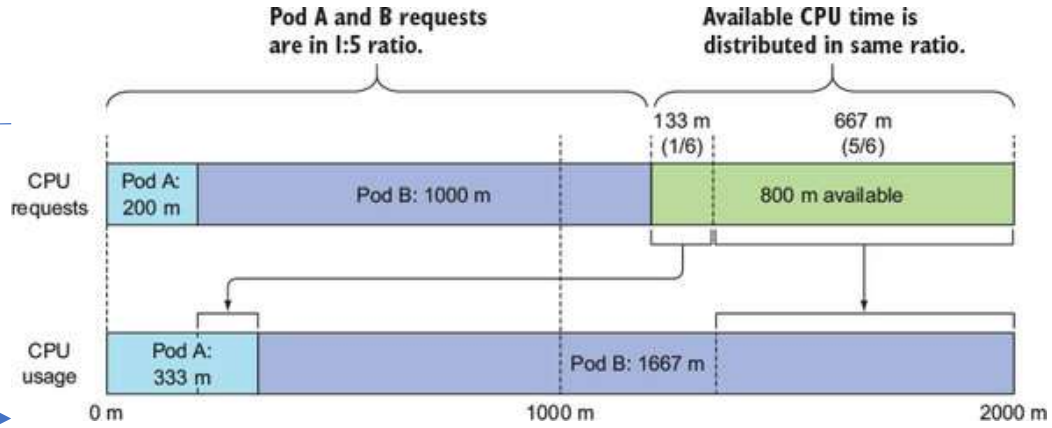
scheduling

- LeastRequestedPriority
- MostRequestedPriority

CPU time sharing

이렇게 요청했지만,

실제로는 전체 CPU를 비율만큼 사용한다.



※ 참고 : https://livebook.manning.com/book/kubernetes-in-action/chapter-14/60

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

- /sys/fs/cgroup/cpu/cpu.cfs_quota_us
- /sys/fs/cgroup/cpu/cpu.cfs_period_us

```
❯ kubectl exec -it requests-pod -- sh
/ # cat /sys/fs/cgroup/cpu/cpu.cfs_quota_us
-1
/ # cat /sys/fs/cgroup/cpu/cpu.cfs_period_us
100000
/ # exit
```

```
❯ kubectl exec -it limits-pod -- sh
/ # cat /sys/fs/cgroup/cpu/cpu.cfs_quota_us
20000
/ # cat /sys/fs/cgroup/cpu/cpu.cfs_period_us
100000
/ # exit
```

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

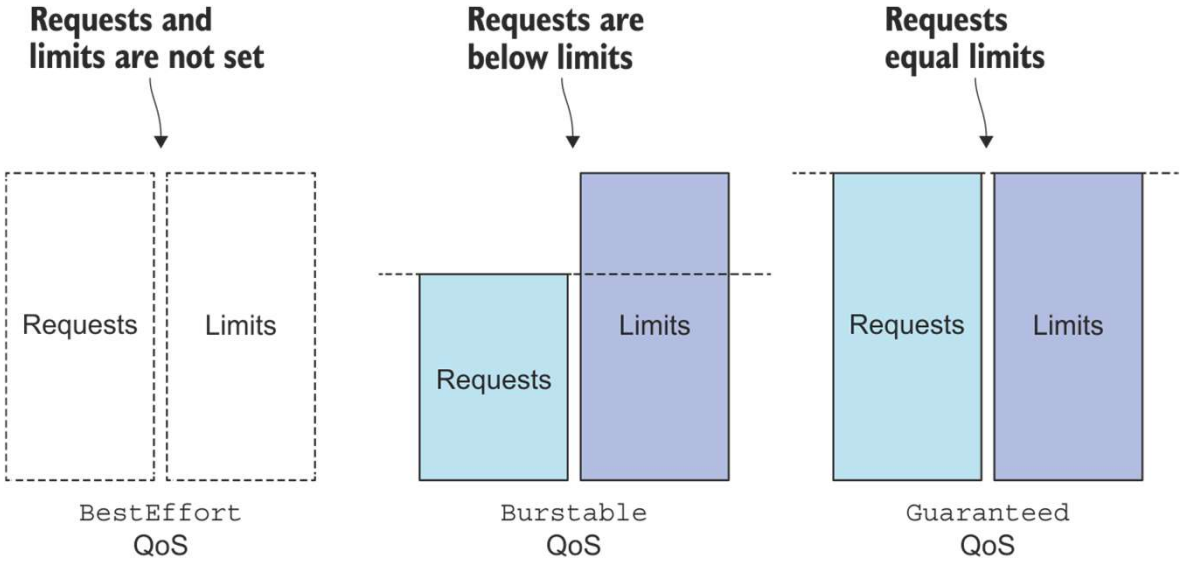
- **Guaranteed** : 최상위 우선순위

  . 3가지 조건 충족되어야 함

  ① requests/limits 모두 설정

  ② 각 container에 모두 설정

  ③ requests == limits



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

QoS (Quality of Service) – 2/2

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`

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

※ 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)

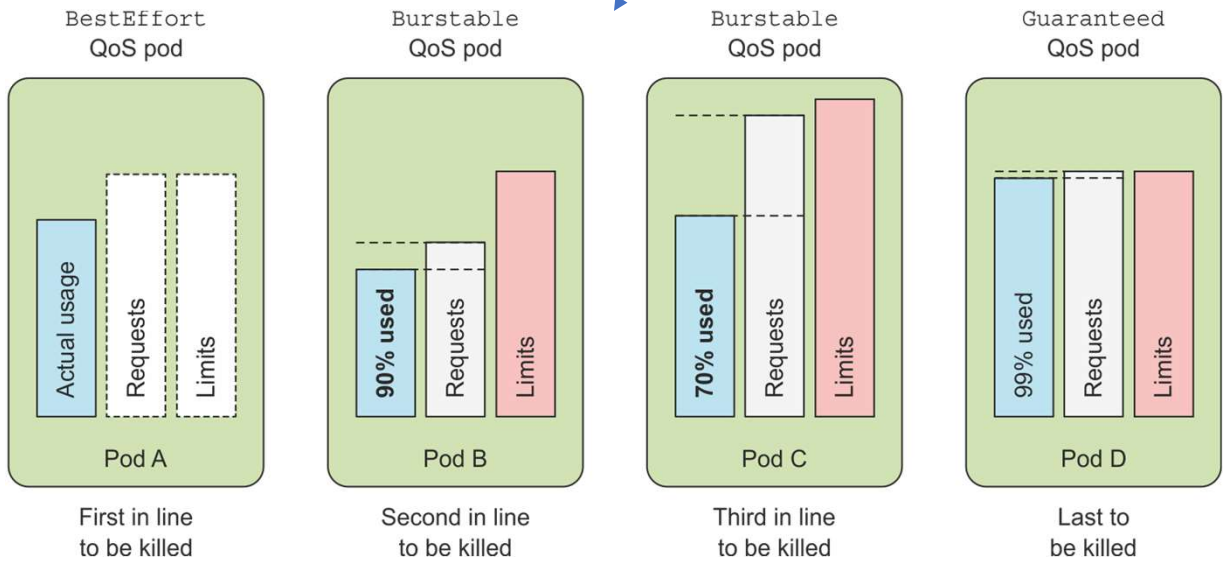
  . 아래 2가지 기준을 QoS 클래스를 기반으로 한 고정된 OOM Score 조정

  ① 프로세스가 소비하는 가용 메모리 비율

  ② requests Memory

Requests 대비하여
사용하는 비율이 높은 Pod가 먼저 종료

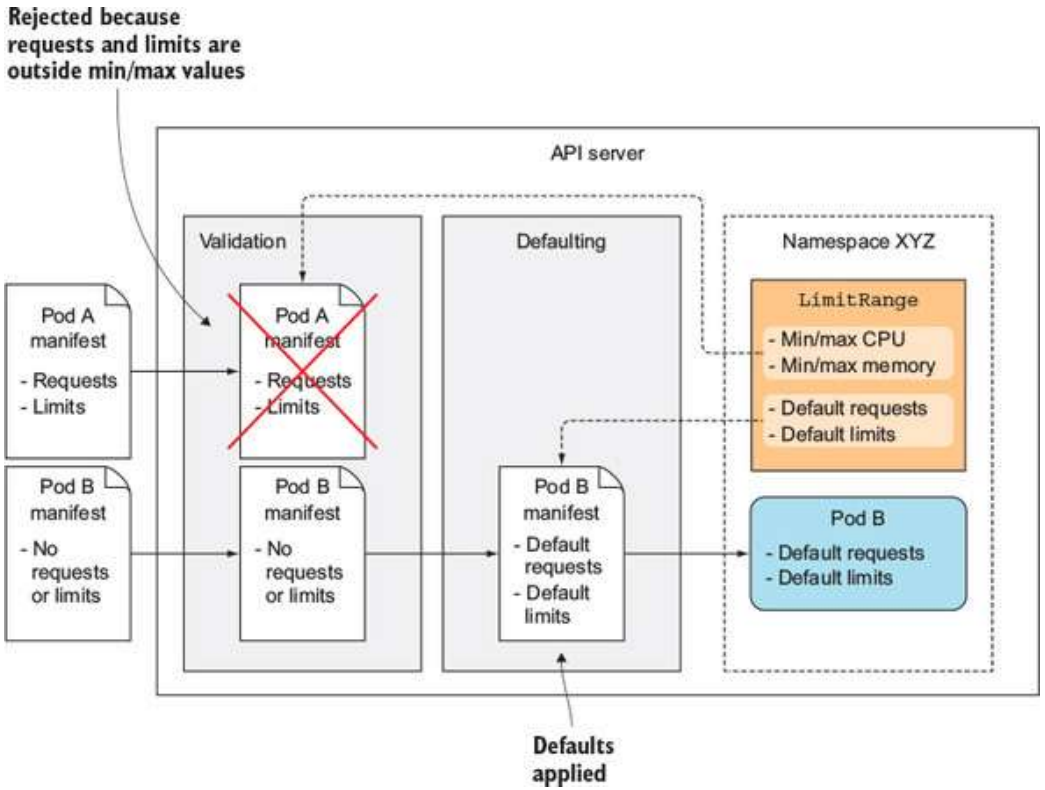
4개의 Pod에 대해서
어떤 것부터
프로세스 종료되는지 설명



※ 참고 : https://livebook.manning.com/book/kubernetes-in-action/chapter-14/151

LimitRange overview – 1/2

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



정해 놓은 *min/max* 값을 벗어난 요청은 거절

따로 명시하지 않아도 *default* 값 기본 적용

※ 참고 : https://livebook.manning.com/book/kubernetes-in-action/chapter-14/162

LimitRange overview – 2/2

03-limitrange.yaml

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: example

spec:

  limits:
  - type: Pod
    min:
      cpu: 50m
      memory: 5Mi

    max:
      cpu: 1
      memory: 1Gi

  - type: Container
    defaultRequest:
      cpu: 100m
      memory: 10Mi

    default:                    # 기본 limits 값
      cpu: 200m
      memory: 100Mi
```

```yaml
    min:
      cpu: 50m
      memory: 5Mi

    max:
      cpu: 1
      memory: 1Gi

    maxLimitRequestRatio:
      cpu: 4                    # limits : requests 최대 비율
      memory: 10

  - type: PersistentVolumeClaim
    min:
      storage: 1Gi

    max:
      storage: 10Gi
```

LimitRange 실습

```
❯ kubectl create namespace limitrange-test
namespace/limitrange-test created

❯ kubectl get namespaces
NAME              STATUS   AGE
default           Active   52d
ingress-nginx     Active   52d
kube-node-lease   Active   52d
kube-public       Active   52d
kube-system       Active   52d
limitrange-test   Active   3s

❯ kubectl create -f 03-limitrange.yaml --namespace limitrange-test
limitrange/example created

❯ kubectl get limitranges --namespace limitrange-test
NAME      CREATED AT
example   2021-06-26T00:57:59Z
```

namespace 하나 만들고 거기에 LimitRange 설정

04-nolimit-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nolimit-pod
spec:
  containers:
  - image: busybox
    command: ["dd", "if=/dev/zero", "of=/dev/null"]
    name: main
```

```
❯ kubectl create -f 04-nolimit-pod.yaml --namespace limitrange-test
pod/nolimit-pod created

❯ kubectl get pods --namespace limitrange-test
NAME          READY   STATUS    RESTARTS   AGE
nolimit-pod   1/1     Running   0          9s

❯ kubectl exec -it --namespace limitrange-test nolimit-pod -- top
```

```
Mem: 2270820K used, 1759800K free, 3380K shrd, 124872K buff, 1440620K cached
CPU:  3.7% usr  9.3% sys  0.0% nic 86.8% idle  0.0% io  0.0% irq  0.1% sirq
Load average: 0.58 0.51 0.32 2/696 10
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
    1     0 root     R     1308  0.0   0 10.1 dd if /dev/zero of /dev/null
    6     0 root     R     1316  0.0   1  0.0 top
```

default cpu 값이 200인데
2 cpu 이니까, 10 !!!

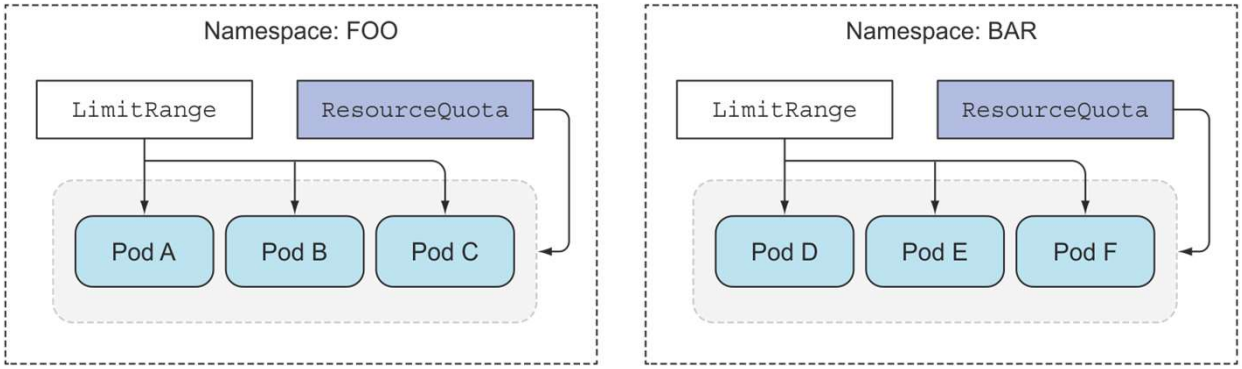
ResourceQuota

- namespace의 모든 pod에서 시용할 수 있는 전체 CPU 및 memory

04-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
    ssd.storageclass.storage.k8s.io/requests.storage: 300Gi
    standard.storageclass.storage.k8s.io/requests.storage: 1Ti
```

```
pods: 10
replicationcontrollers: 5
secrets: 10
configmaps: 10
persistentvolumeclaims: 5
services: 5
services.loadbalancers: 1
services.nodeports: 2
ssd.storageclass.storage.k8s.io/persistentvolumeclaims: 2
```



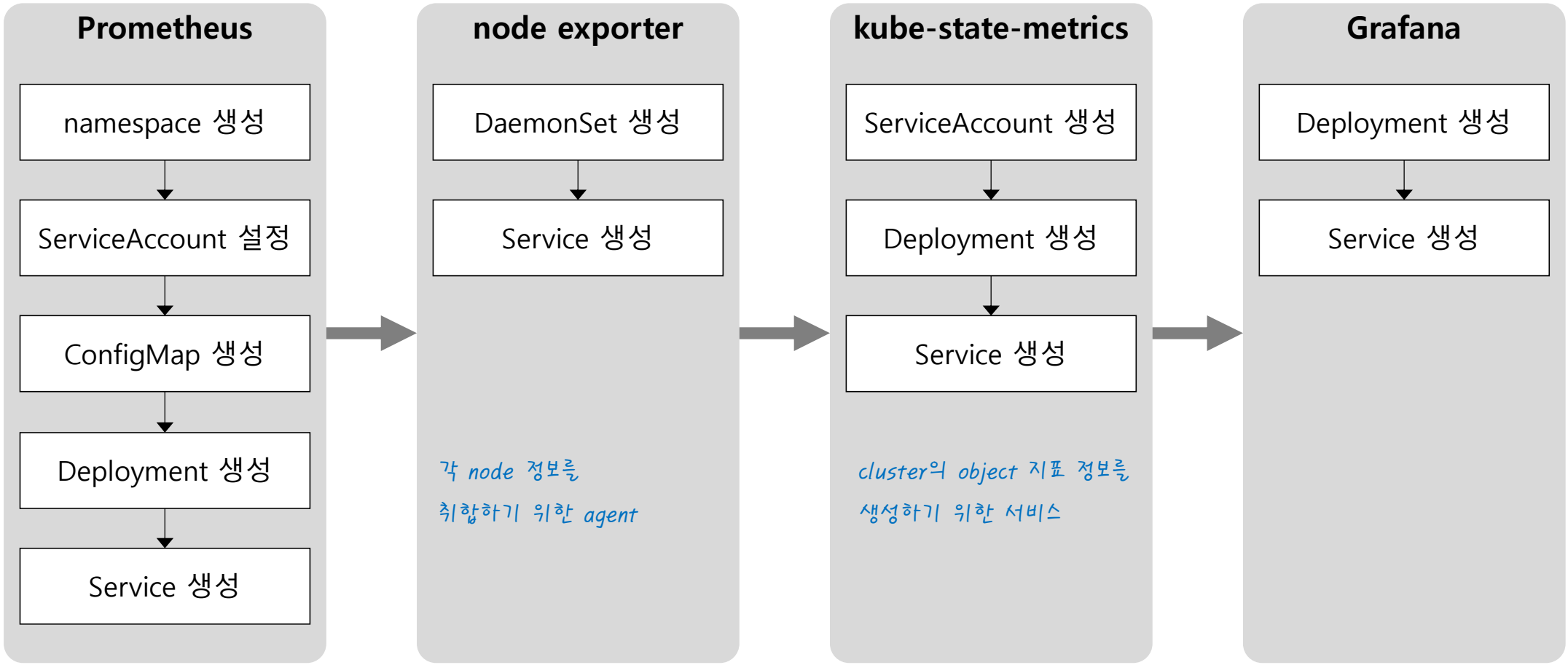
※ 참고 : https://livebook.manning.com/book/kubernetes-in-action/chapter-14/198

Resource Monitoring – metric server

- metric-server (heapster = 아재)

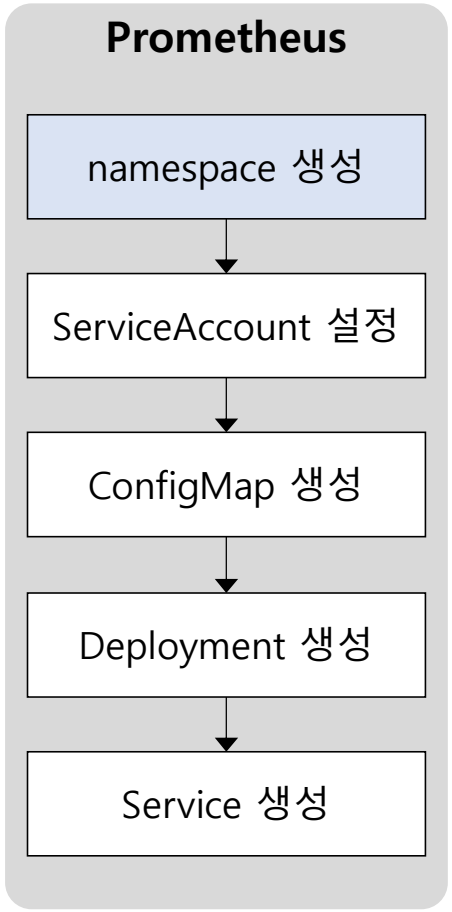
```
❯ kubectl get pods --namespace kube-system | grep metric
kubernetes-metrics-scraper-64db6db887-zkx7r   1/1     Running   16         52d
metrics-server-988c74c86-wwkr2                2/2     Running   34         52d

❯ kubectl top nodes
NAME         CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
master-stg   200m         11%    1497Mi          45%
worker1      82m          4%     1062Mi          29%
worker2      88m          4%     1110Mi          31%

❯ kubectl top pods
NAME       CPU(cores)   MEMORY(bytes)
mongod-0   3m           35Mi
mongod-1   3m           32Mi
mongod-2   3m           32Mi

❯ kubectl top pods --containers
POD        NAME               CPU(cores)   MEMORY(bytes)
mongod-0   mongod-container   3m           35Mi
mongod-1   mongod-container   3m           32Mi
mongod-2   mongod-container   3m           32Mi
```

Resource Monitoring – Prometheus & Grafana



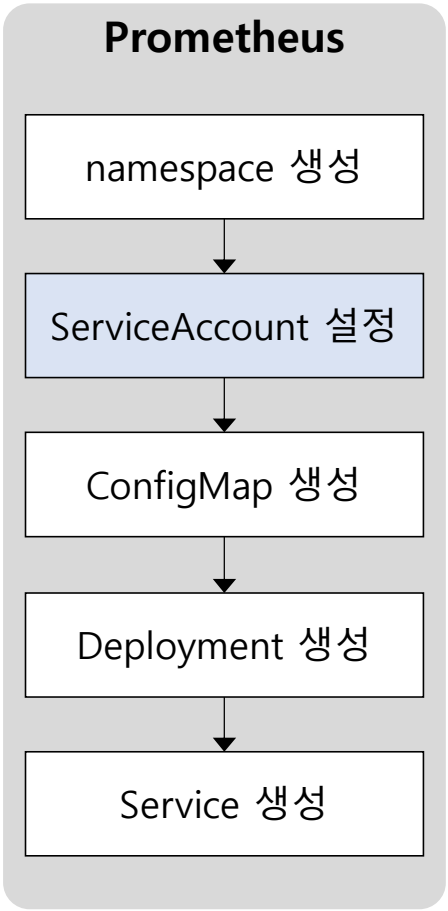
※ 참고 : https://velog.io/@pingping95/Kubernetes-Prometheus-Grafana-%EB%AA%A8%EB%8B%88%ED%84%B0%EB%A7%81-%EC%84%A4%EC%B9%98-KVM

Prometheus & Grafana – Prometheus




```
> kubectl create namespace monitoring
namespace/monitoring created
```

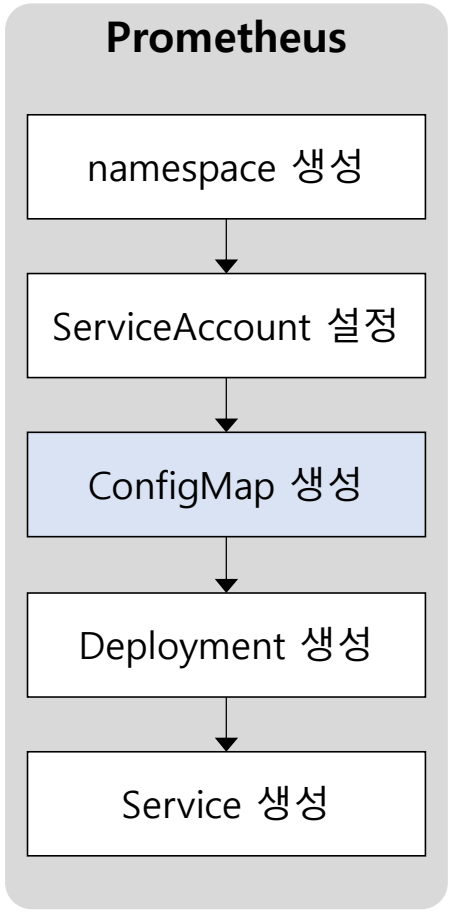
Prometheus & Grafana – Prometheus



default 계정에 ClusterRole Binding


```
❯ kubectl create -f 01-prometheus-cr.yaml
clusterrole.rbac.authorization.k8s.io/prometheus created

❯ kubectl create -f 02-prometheus-crb.yaml
clusterrolebinding.rbac.authorization.k8s.io/prometheus created
```

Prometheus & Grafana – Prometheus

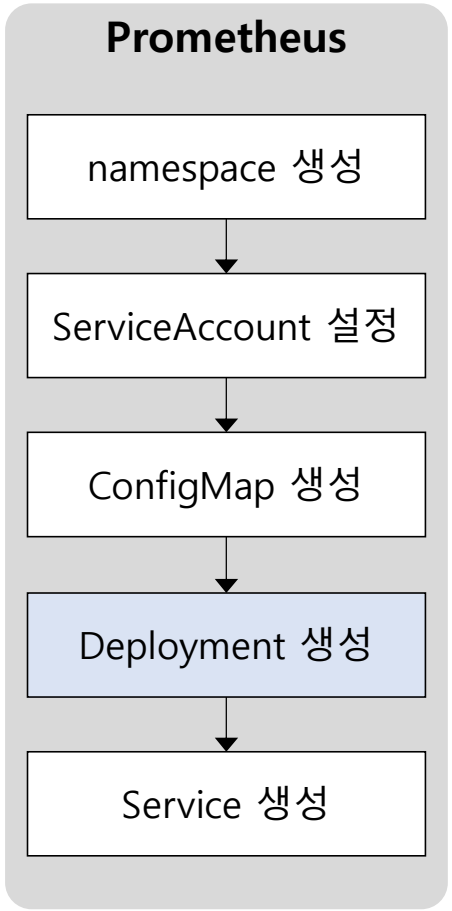


환경 설정 파일

- prometheus.rules — Metric에 대한 Alarm 조건
- prometheus.yml — Metric의 종류, 수집 주기

```
❯ kubectl create -f 03-prometheus-cm.yaml --namespace monitoring
configmap/prometheus-server-conf created
```

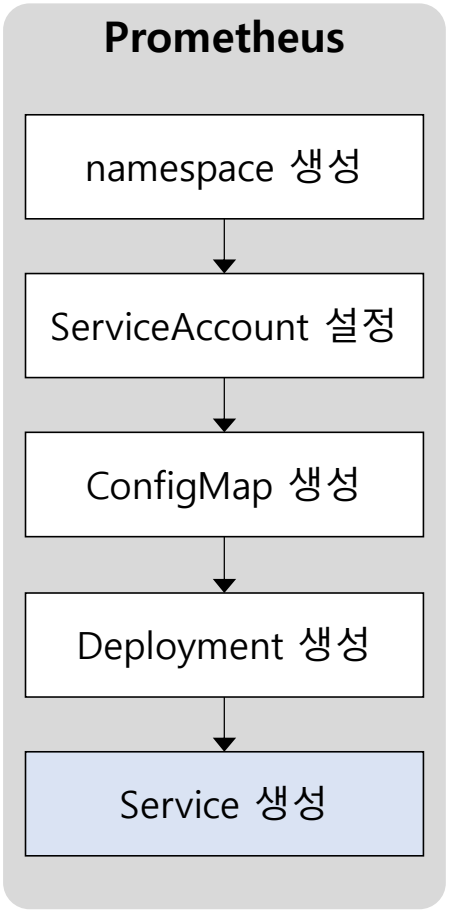
Prometheus & Grafana – Prometheus



Prometheus 띄우기

```
❯ kubectl create -f 04-prometheus-deployment.yaml --namespace monitoring
deployment.apps/prometheus-deployment created
```

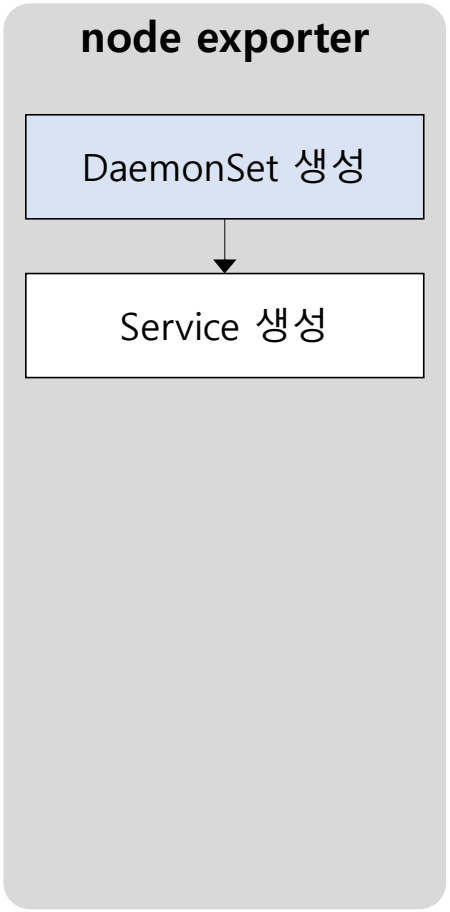
Prometheus & Grafana – Prometheus



NodePort 30003

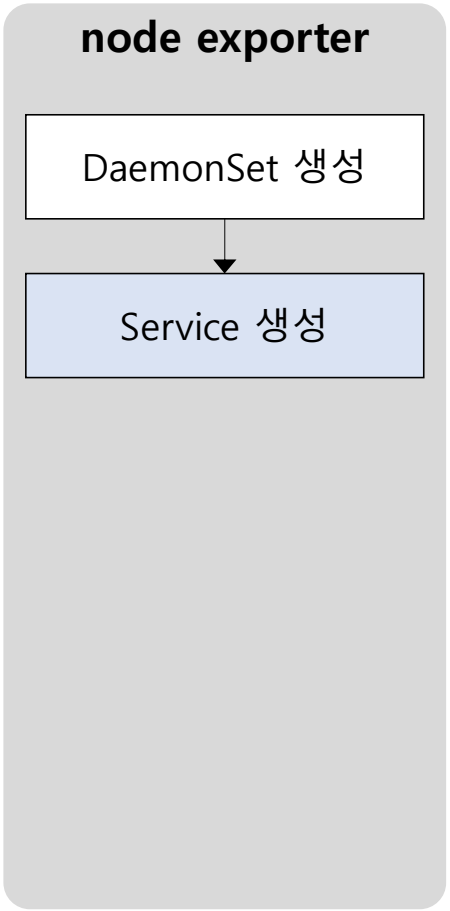
```
> kubectl create -f 05-prometheus-svc.yaml --namespace monitoring
service/prometheus-service created
```

Resource Monitoring – node exporter




```
❯ kubectl create -f 06-node-exporter-daemonset.yaml --namespace monitoring
daemonset.apps/node-exporter created
```

Resource Monitoring – node exporter

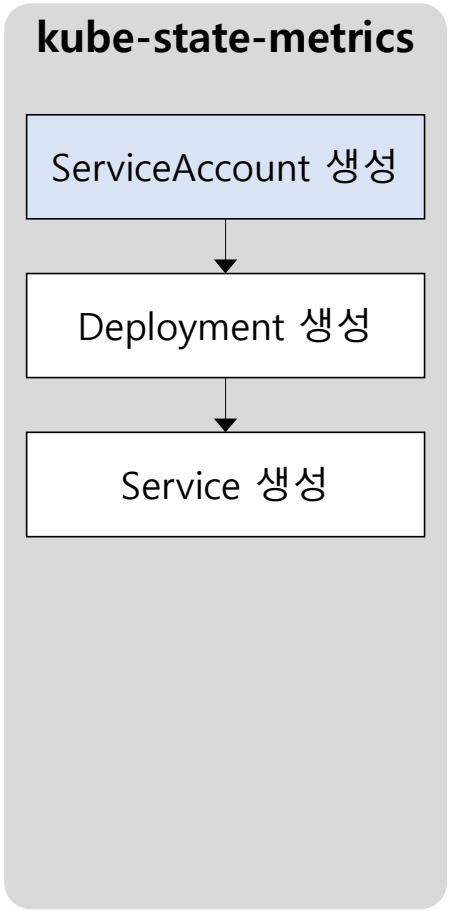


namespace = kube-system 으로 생성

NodePort 31672

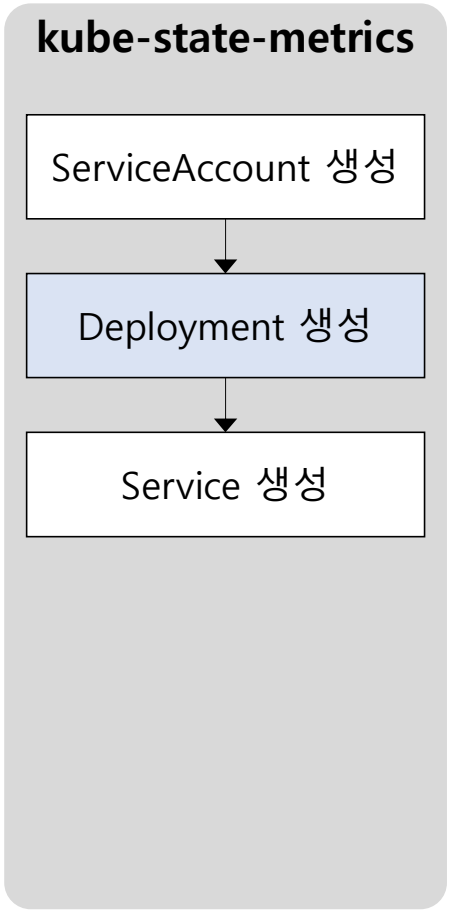
```
> kubectl create -f 07-node-exporter-svc.yaml --namespace kube-system
service/node-exporter created
```

Resource Monitoring – kube-state-metrics




```
❯ kubectl create -f 08-kube-state-sa.yaml
serviceaccount/kube-state-metrics created

❯ kubectl create -f 09-kube-state-cr.yaml
clusterrole.rbac.authorization.k8s.io/kube-state-metrics created

❯ kubectl create -f 10-kube-state-crb.yaml
clusterrolebinding.rbac.authorization.k8s.io/kube-state-metrics created
```

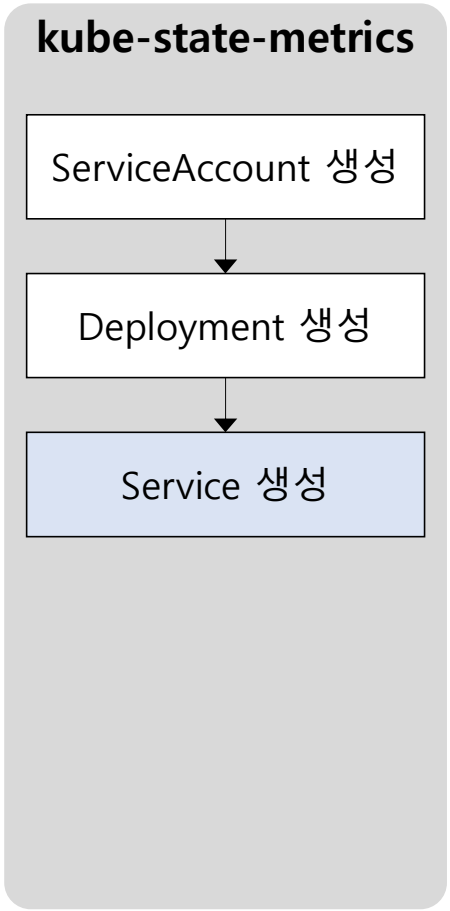
Resource Monitoring – kube-state-metrics



namespace = kube-system 으로 생성

```
❯ kubectl create -f 11-kube-state-deployment.yaml --namespace kube-system
deployment.apps/kube-state-metrics created
```

Resource Monitoring – kube-state-metrics

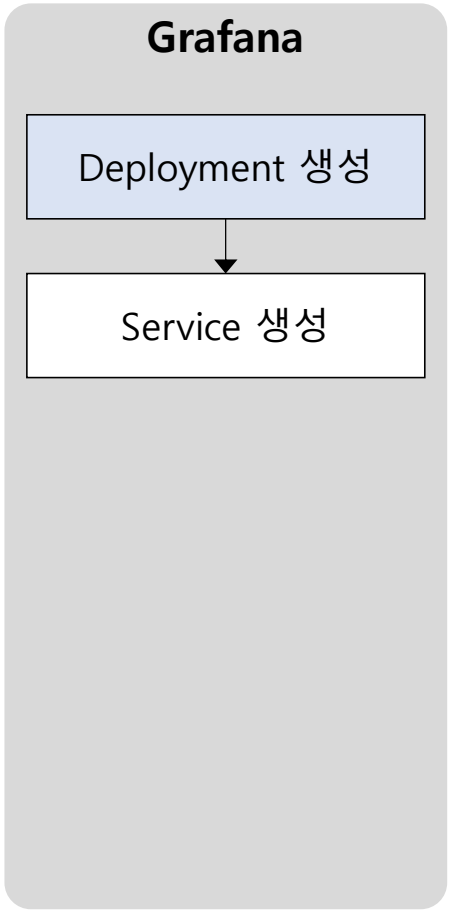


namespace = kube-system 으로 생성

headless service로 생성됨

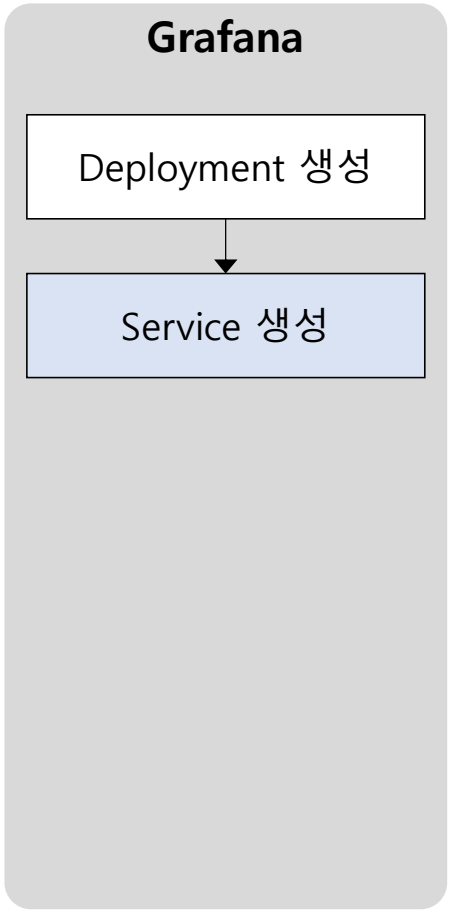
```
> kubectl create -f 12-kube-state-svc.yaml --namespace kube-system
service/kube-state-metrics created
```

Resource Monitoring – Grafana



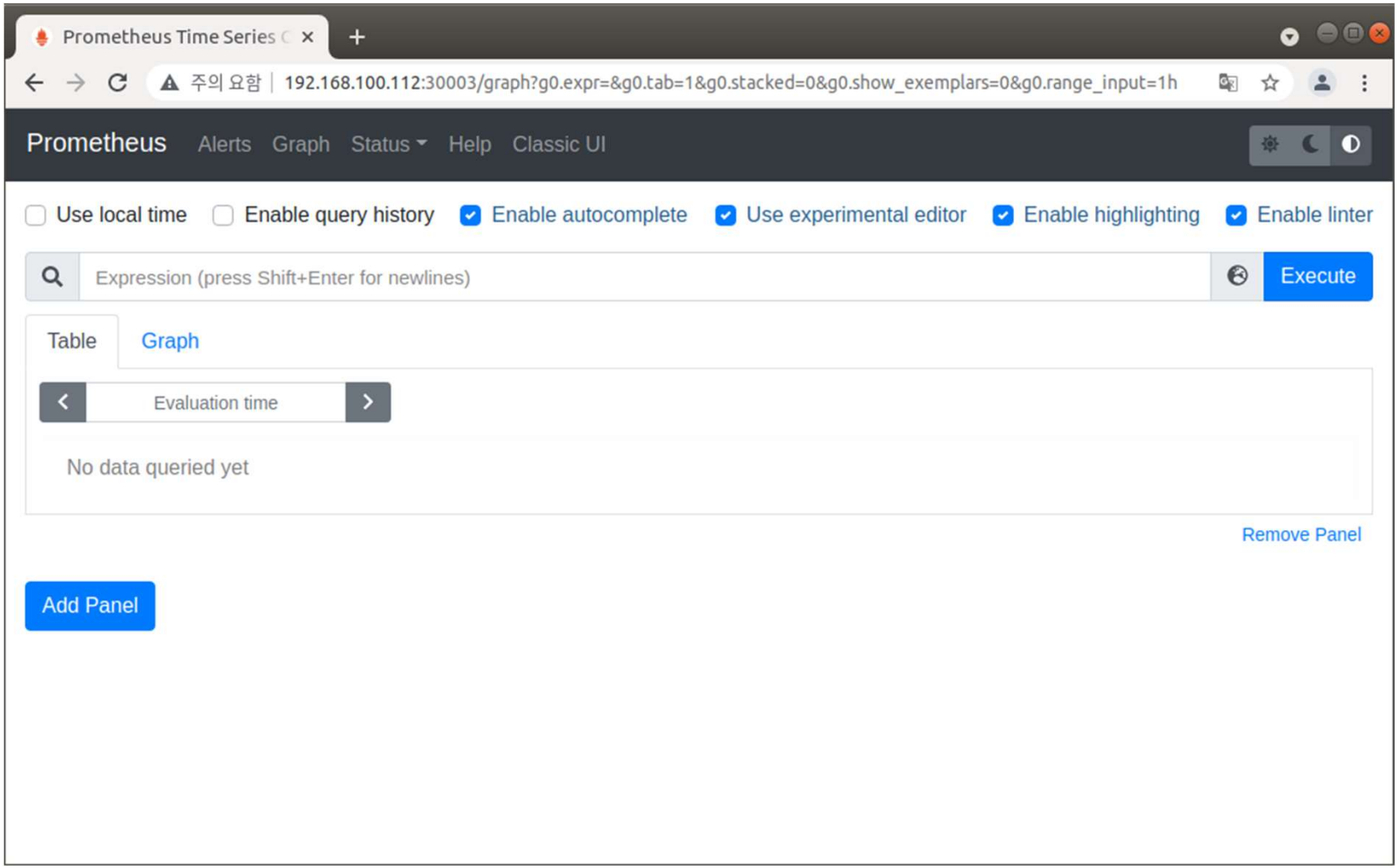

```
❯ kubectl create -f 13-grafana-deployment.yaml --namespace monitoring
deployment.apps/grafana created
```

Resource Monitoring – Grafana

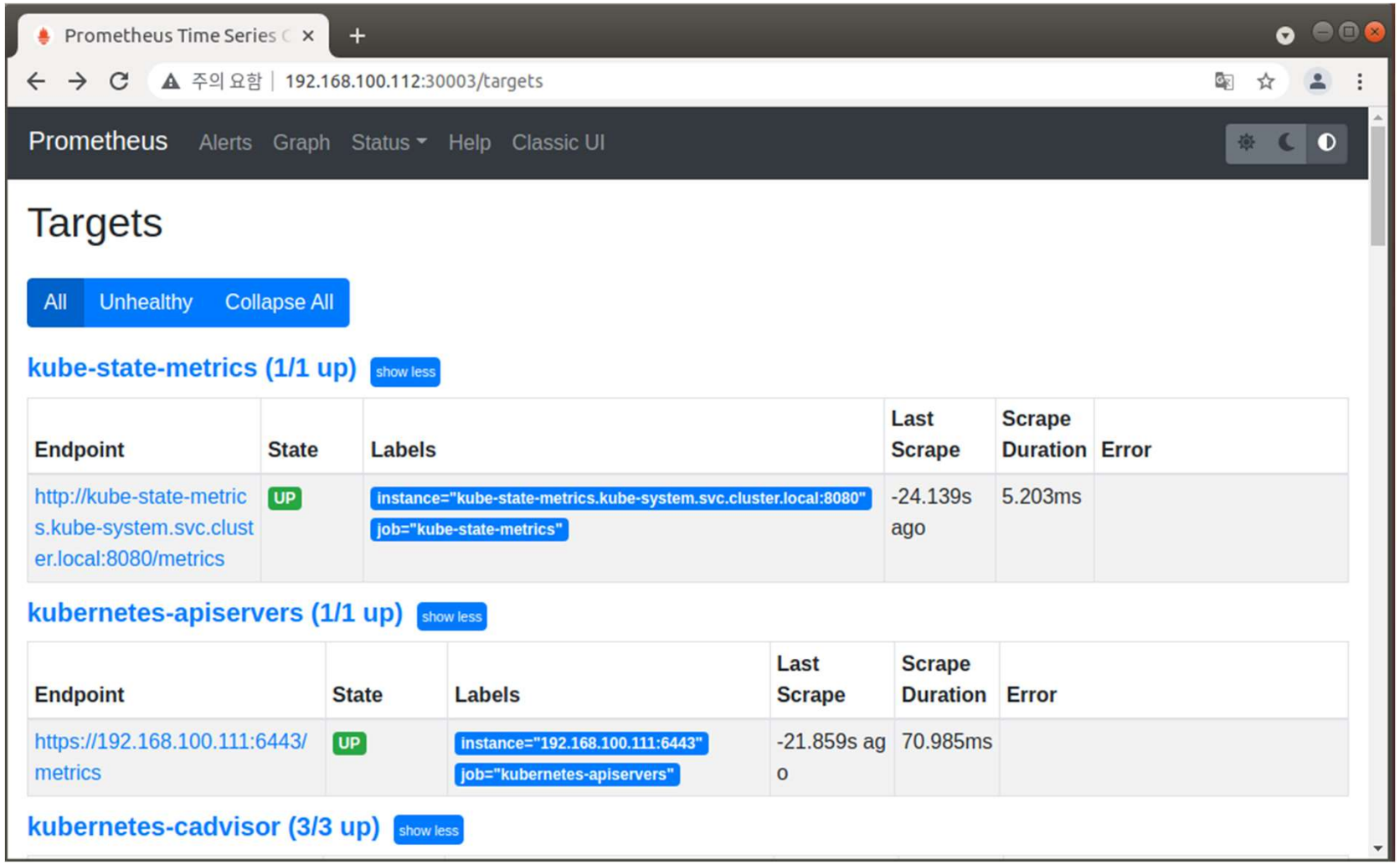



```
❯ kubectl create -f 14-grafana-svc.yaml --namespace monitoring
service/grafana created
```

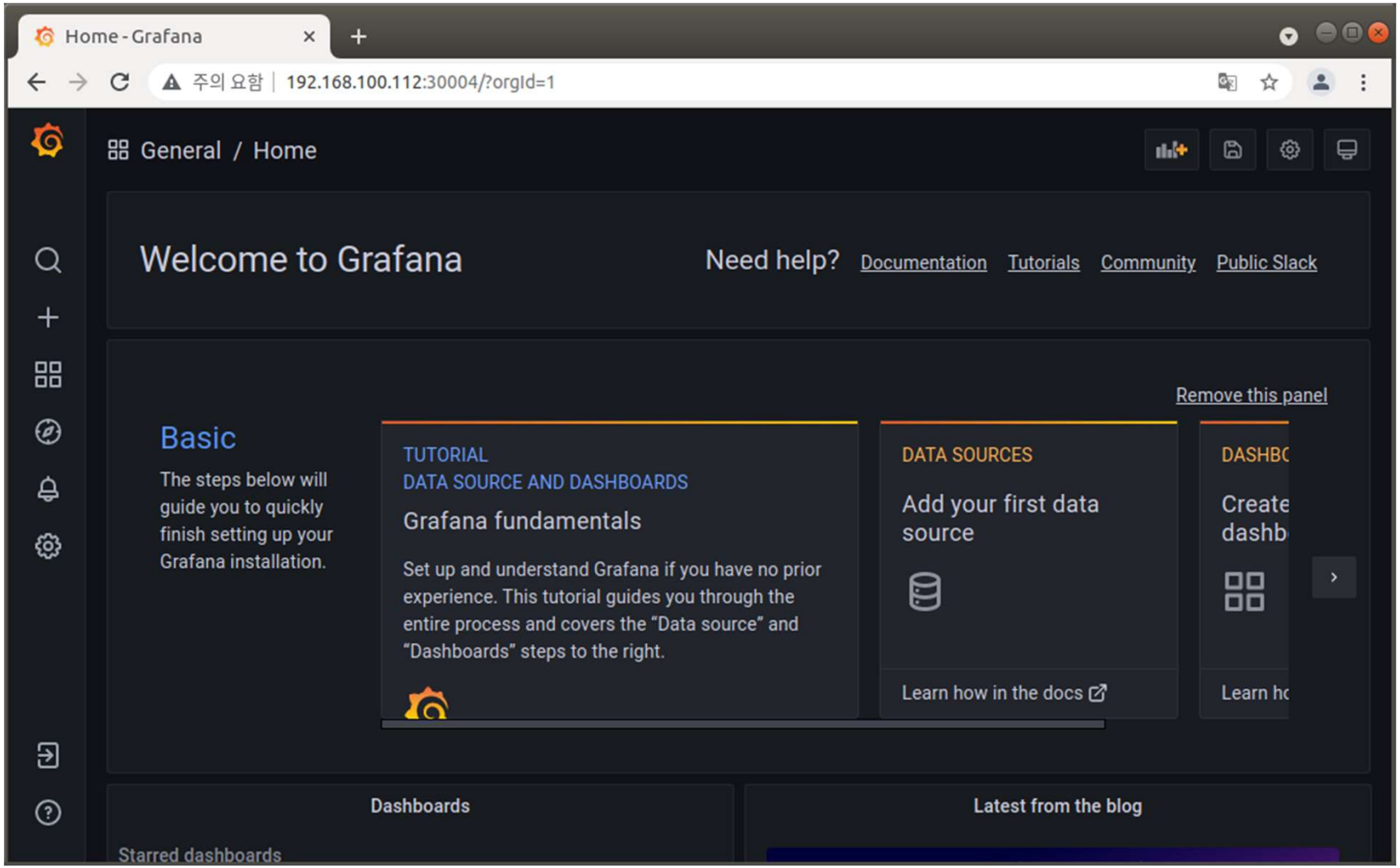
Prometheus


Prometheus Time Series
주의 요함 | 192.168.100.112:30003/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
Use experimental editor
Enable highlighting
Enable linter
Expression (press Shift+Enter for newlines)
Execute
Table
Graph
Evaluation time
No data queried yet
Remove Panel
Add Panel

Prometheus : Status – Targets 확인



Grafana

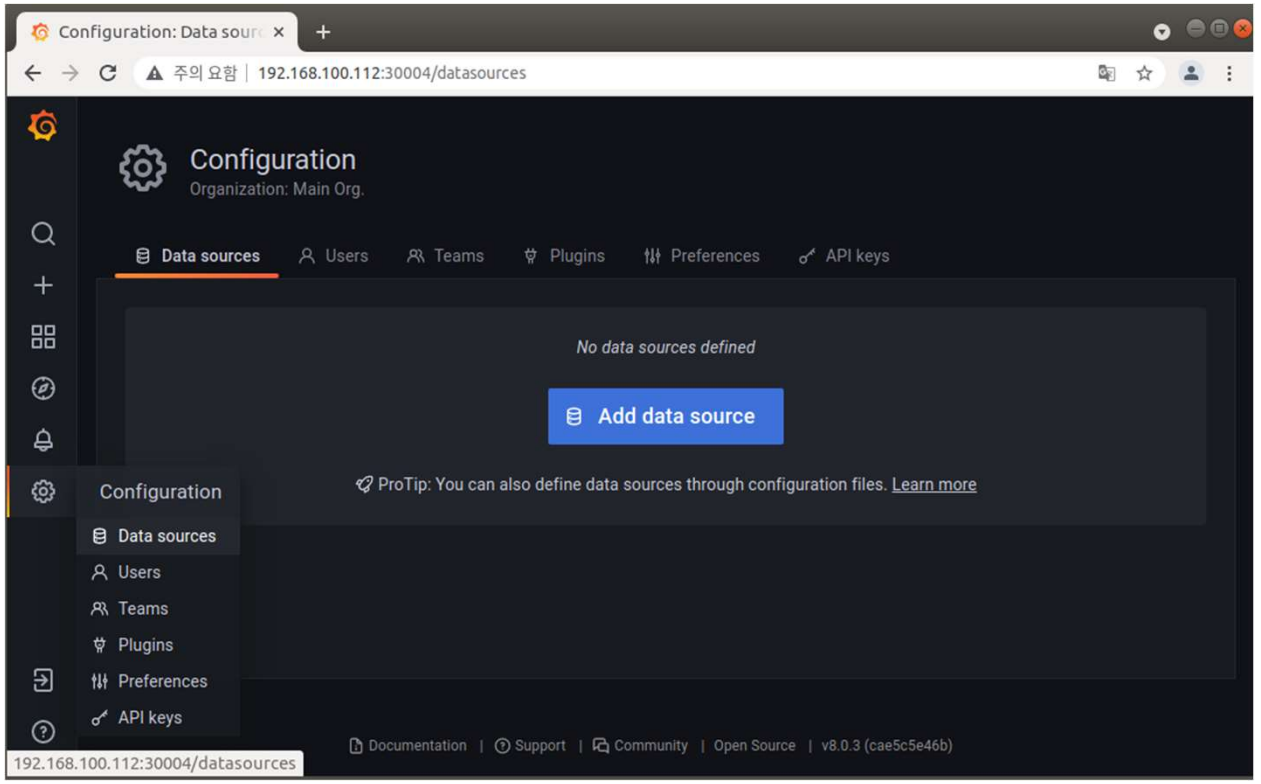

Home - Grafana
주의 요함 | 192.168.100.112:30004/?orgId=1
General / Home
Welcome to Grafana
Need help?
Documentation
Tutorials
Community
Public Slack
Remove this panel
Basic
The steps below will guide you to quickly finish setting up your Grafana installation.
TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals
Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.
DATA SOURCES
Add your first data source
Learn how in the docs
Dashboards
Starred dashboards
Latest from the blog

Grafana : Prometheus 연계 – 1/4

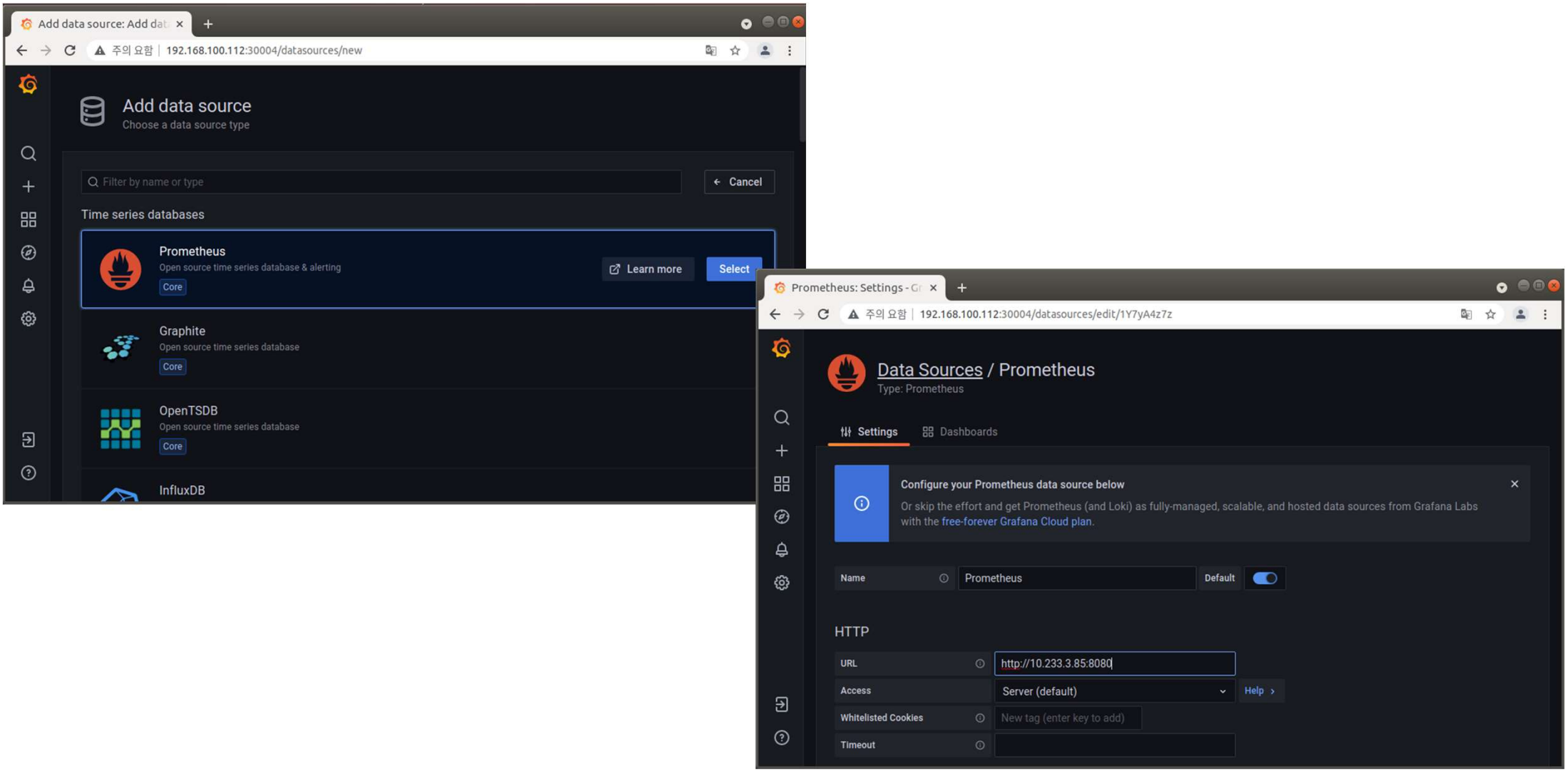
```
❯ kubectl get services --namespace monitoring
NAME                 TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
grafana              NodePort   10.233.36.113   <none>        3000:30004/TCP   12m
prometheus-service   NodePort   10.233.3.85     <none>        8080:30003/TCP   63m
```

Configuration – Data Sources 선택

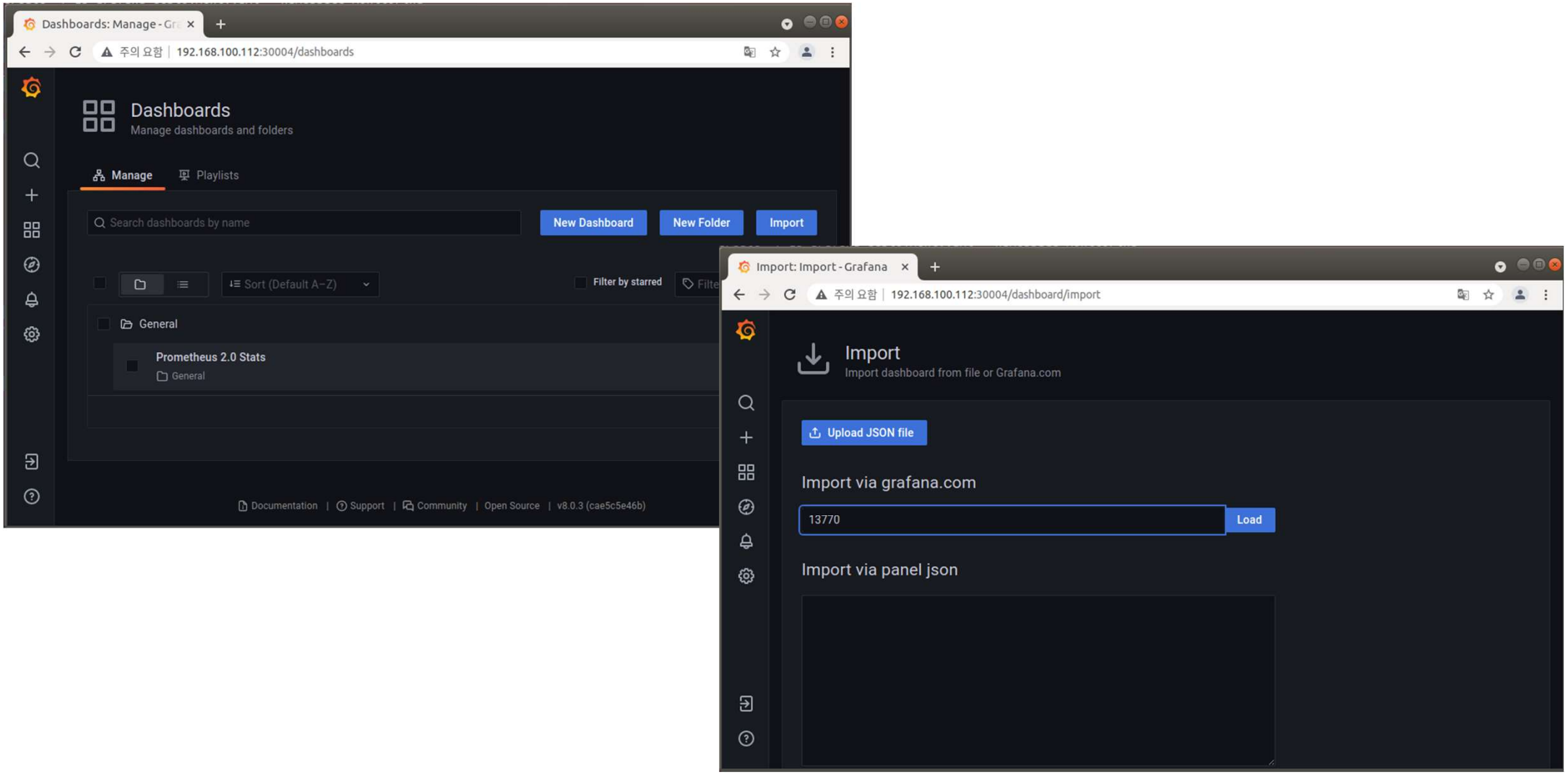
Add data source 클릭



Grafana : Prometheus 연계 – 2/4



Grafana : Prometheus 연계 – 3/4



Grafana : Prometheus 연계 – 4/4