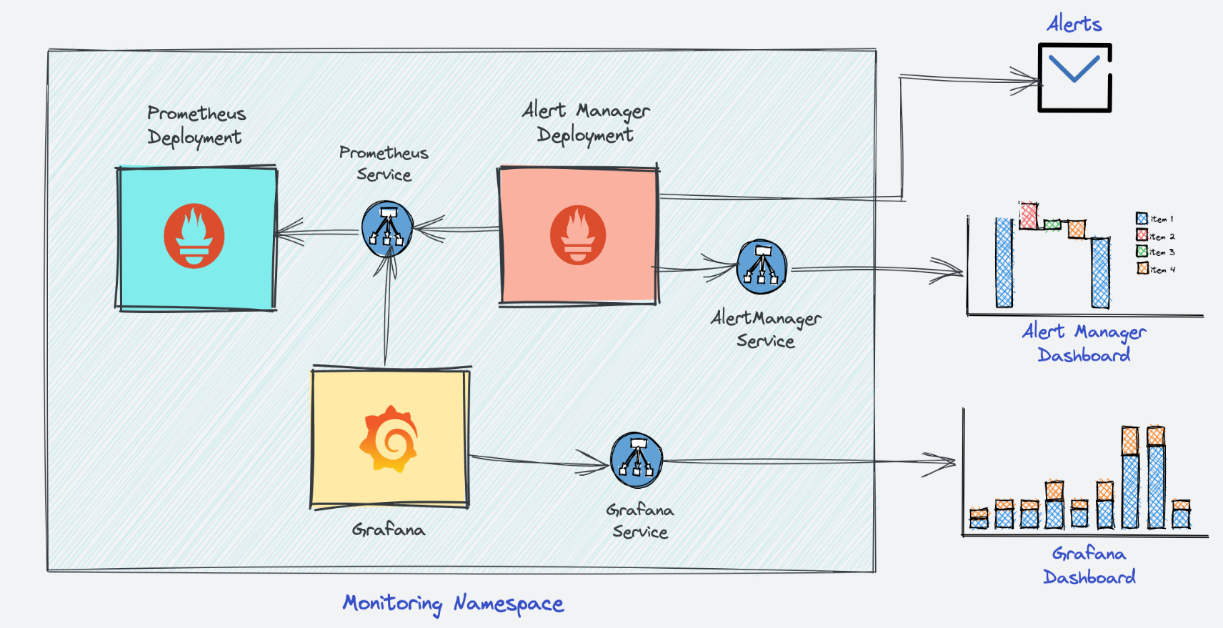
# Setup Prometheus Monitoring On Kubernetes Cluster

Prometheus is a high-scalable open-source monitoring framework. It provides out-of-the-box monitoring capabilities for the Kubernetes [container orchestration platform](https://devopscube.com/docker-container-clustering-tools/). Also, In the [observability](https://devopscube.com/what-is-observability/) space, it is gaining huge popularity as it helps with metrics and alerts.

Prometheus Architecture: - The Kubernetes Prometheus monitoring stack has the following components.

1. Prometheus Server
2. Alert Manager
3. Grafana



Step 1: clone the git for Prometheus Kubernetes Manifest Files

$ git clone <https://github.com/techiescamp/kubernetes-prometheus>

## Step 2: Create a Namespace & Cluster Role

$ kubectl create namespace monitoring  
  
Step 3: Create a file named clusterRole.yaml and copy the following RBAC role.

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"]

---

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

Step 4: Create the role using the following command.

$ kubectl create -f clusterRole.yaml

Step 5: create a config map to externalize Prometheus configurations

All configurations for Prometheus are part of prometheus.yaml file and all the alert rules for Alert manager are configured in prometheus.rules.

prometheus.yaml: This is the main Prometheus configuration which holds all the scrape configs, service discovery details, storage locations, data retention configs, etc)

prometheus.rules: This file contains all the Prometheus alerting rules

Step 6:Create a file called config-map.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: prometheus-server-conf

labels:

name: prometheus-server-conf

namespace: monitoring

data:

prometheus.rules: |-

groups:

- name: devopscube demo alert

rules:

- alert: High Pod Memory

expr: sum(container\_memory\_usage\_bytes) > 1

for: 1m

labels:

severity: slack

annotations:

summary: High Memory Usage

prometheus.yml: |-

global:

scrape\_interval: 5s

evaluation\_interval: 5s

rule\_files:

- /etc/prometheus/prometheus.rules

alerting:

alertmanagers:

- scheme: http

static\_configs:

- targets:

- "alertmanager.monitoring.svc:9093"

scrape\_configs:

- job\_name: 'node-exporter'

kubernetes\_sd\_configs:

- role: endpoints

relabel\_configs:

- source\_labels: [\_\_meta\_kubernetes\_endpoints\_name]

regex: 'node-exporter'

action: keep

- job\_name: 'kubernetes-apiservers'

kubernetes\_sd\_configs:

- role: endpoints

scheme: https

tls\_config:

ca\_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

bearer\_token\_file: /var/run/secrets/kubernetes.io/serviceaccount/token

relabel\_configs:

- source\_labels: [\_\_meta\_kubernetes\_namespace, \_\_meta\_kubernetes\_service\_name, \_\_meta\_kubernetes\_endpoint\_port\_name]

action: keep

regex: default;kubernetes;https

- job\_name: 'kubernetes-nodes'

scheme: https

tls\_config:

ca\_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

bearer\_token\_file: /var/run/secrets/kubernetes.io/serviceaccount/token

kubernetes\_sd\_configs:

- role: node

relabel\_configs:

- action: labelmap

regex: \_\_meta\_kubernetes\_node\_label\_(.+)

- target\_label: \_\_address\_\_

replacement: kubernetes.default.svc:443

- source\_labels: [\_\_meta\_kubernetes\_node\_name]

regex: (.+)

target\_label: \_\_metrics\_path\_\_

replacement: /api/v1/nodes/${1}/proxy/metrics

- job\_name: 'kubernetes-pods'

kubernetes\_sd\_configs:

- role: pod

relabel\_configs:

- source\_labels: [\_\_meta\_kubernetes\_pod\_annotation\_prometheus\_io\_scrape]

action: keep

regex: true

- source\_labels: [\_\_meta\_kubernetes\_pod\_annotation\_prometheus\_io\_path]

action: replace

target\_label: \_\_metrics\_path\_\_

regex: (.+)

- source\_labels: [\_\_address\_\_, \_\_meta\_kubernetes\_pod\_annotation\_prometheus\_io\_port]

action: replace

regex: ([^:]+)(?::\d+)?;(\d+)

replacement: $1:$2

target\_label: \_\_address\_\_

- action: labelmap

regex: \_\_meta\_kubernetes\_pod\_label\_(.+)

- source\_labels: [\_\_meta\_kubernetes\_namespace]

action: replace

target\_label: kubernetes\_namespace

- source\_labels: [\_\_meta\_kubernetes\_pod\_name]

action: replace

target\_label: kubernetes\_pod\_name

- job\_name: 'kube-state-metrics'

static\_configs:

- targets: ['kube-state-metrics.kube-system.svc.cluster.local:8080']

- job\_name: 'kubernetes-cadvisor'

scheme: https

tls\_config:

ca\_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

bearer\_token\_file: /var/run/secrets/kubernetes.io/serviceaccount/token

kubernetes\_sd\_configs:

- role: node

relabel\_configs:

- action: labelmap

regex: \_\_meta\_kubernetes\_node\_label\_(.+)

- target\_label: \_\_address\_\_

replacement: kubernetes.default.svc:443

- source\_labels: [\_\_meta\_kubernetes\_node\_name]

regex: (.+)

target\_label: \_\_metrics\_path\_\_

replacement: /api/v1/nodes/${1}/proxy/metrics/cadvisor

- job\_name: 'kubernetes-service-endpoints'

kubernetes\_sd\_configs:

- role: endpoints

relabel\_configs:

- source\_labels: [\_\_meta\_kubernetes\_service\_annotation\_prometheus\_io\_scrape]

action: keep

regex: true

- source\_labels: [\_\_meta\_kubernetes\_service\_annotation\_prometheus\_io\_scheme]

action: replace

target\_label: \_\_scheme\_\_

regex: (https?)

- source\_labels: [\_\_meta\_kubernetes\_service\_annotation\_prometheus\_io\_path]

action: replace

target\_label: \_\_metrics\_path\_\_

regex: (.+)

- source\_labels: [\_\_address\_\_, \_\_meta\_kubernetes\_service\_annotation\_prometheus\_io\_port]

action: replace

target\_label: \_\_address\_\_

regex: ([^:]+)(?::\d+)?;(\d+)

replacement: $1:$2

- action: labelmap

regex: \_\_meta\_kubernetes\_service\_label\_(.+)

- source\_labels: [\_\_meta\_kubernetes\_namespace]

action: replace

target\_label: kubernetes\_namespace

- source\_labels: [\_\_meta\_kubernetes\_service\_name]

action: replace

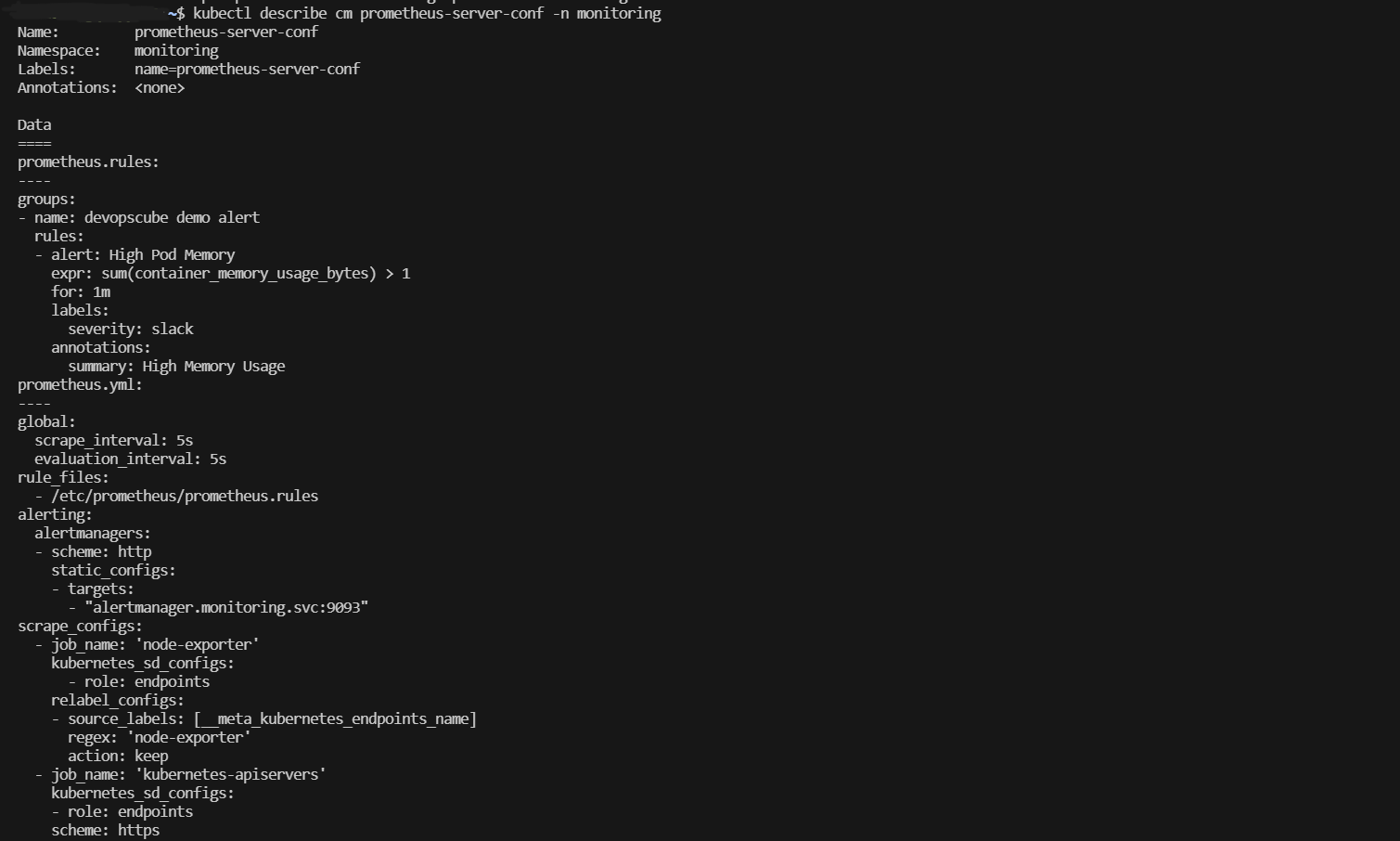
target\_label: kubernetes\_name

Step 7: Execute the following command to create the config map in Kubernetes.

$ kubectl create -f config-map.yaml

Step 8:Check wheather it is created or not

$ kubectl describe cm prometheus-server-conf –n monitoring



Step 9: Create a file named **prometheus-deployment.yaml** and copy the following contents onto the file.

apiVersion: apps/v1

kind: Deployment

metadata:

name: prometheus-deployment

namespace: monitoring

labels:

app: prometheus-server

spec:

replicas: 1

selector:

matchLabels:

app: prometheus-server

template:

metadata:

labels:

app: prometheus-server

spec:

containers:

- name: prometheus

image: prom/prometheus

args:

- "--storage.tsdb.retention.time=12h"

- "--config.file=/etc/prometheus/prometheus.yml"

- "--storage.tsdb.path=/prometheus/"

ports:

- containerPort: 9090

resources:

requests:

cpu: 500m

memory: 500M

limits:

cpu: 1

memory: 1Gi

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: {}

Step 10: Create a deployment on monitoring namespace using the above file.

$ kubectl create -f prometheus-deployment.yaml

Step 11: You can check the created deployment using the following command.

NAME READY UP-TO-DATE AVAILABLE AGE

Prometheus-deployment 1/1 1 1 5d20h

Step 12: Create a file named **prometheus-service.yaml** and copy the following contents. We will expose Prometheus on all kubernetes node IP’s on port 30000.

apiVersion: v1

kind: Service

metadata:

name: prometheus-service

namespace: monitoring

annotations:

prometheus.io/scrape: 'true'

prometheus.io/port: '9090'

spec:

selector:

app: prometheus-server

type: NodePort

ports:

- port: 8080

targetPort: 9090

nodePort: 30000

Step 13: Create the service using the following command.

$ kubectl create -f prometheus-service.yaml --namespace=monitoring

Step 14: Now we can access to use IP address and Nodeport check in the Browers

Step 15: **kube state metrics setup** and Clone the Github repo

$ git clone <https://github.com/devopscube/kube-state-metrics-configs.git>

Step 16:Create all the objects by pointing to the cloned directory.

$ kubectl apply -f kube-state-metrics-configs/

Step 17: Check the deployment status using the following command.

$ kubectl get deployments kube-state-metrics -n kube-system

NAME READY UP-TO-DATE AVAILABLE AGE

kube-state-metrics 1/1 1 1 5d4h

Step 18: Setup Node Exporter on KubernetesCreate a file name **daemonset.yaml** and copy the following content.

apiVersion: apps/v1

kind: DaemonSet

metadata:

labels:

app.kubernetes.io/component: exporter

app.kubernetes.io/name: node-exporter

name: node-exporter

namespace: monitoring

spec:

selector:

matchLabels:

app.kubernetes.io/component: exporter

app.kubernetes.io/name: node-exporter

template:

metadata:

labels:

app.kubernetes.io/component: exporter

app.kubernetes.io/name: node-exporter

spec:

containers:

- args:

- --path.sysfs=/host/sys

- --path.rootfs=/host/root

- --no-collector.wifi

- --no-collector.hwmon

- --collector.filesystem.ignored-mount-points=^/(dev|proc|sys|var/lib/docker/.+|var/lib/kubelet/pods/.+)($|/)

- --collector.netclass.ignored-devices=^(veth.\*)$

name: node-exporter

image: prom/node-exporter

ports:

- containerPort: 9100

protocol: TCP

resources:

limits:

cpu: 250m

memory: 180Mi

requests:

cpu: 102m

memory: 180Mi

volumeMounts:

- mountPath: /host/sys

mountPropagation: HostToContainer

name: sys

readOnly: true

- mountPath: /host/root

mountPropagation: HostToContainer

name: root

readOnly: true

volumes:

- hostPath:

path: /sys

name: sys

- hostPath:

path: /

name: root

Step 19: Deploy the daemon set using the kubectl command.

$ kubectl create -f daemonset.yaml

Step 20: List the daemon set in the monitoring namespace and make sure it is in the available state.

$ kubectl get daemonset -n monitoring

Step 21:Create a file names service.yaml and copy the following contents.

kind: Service

apiVersion: v1

metadata:

name: node-exporter

namespace: monitoring

annotations:

prometheus.io/scrape: 'true'

prometheus.io/port: '9100'

spec:

selector:

app.kubernetes.io/component: exporter

app.kubernetes.io/name: node-exporter

ports:

- name: node-exporter

protocol: TCP

port: 9100

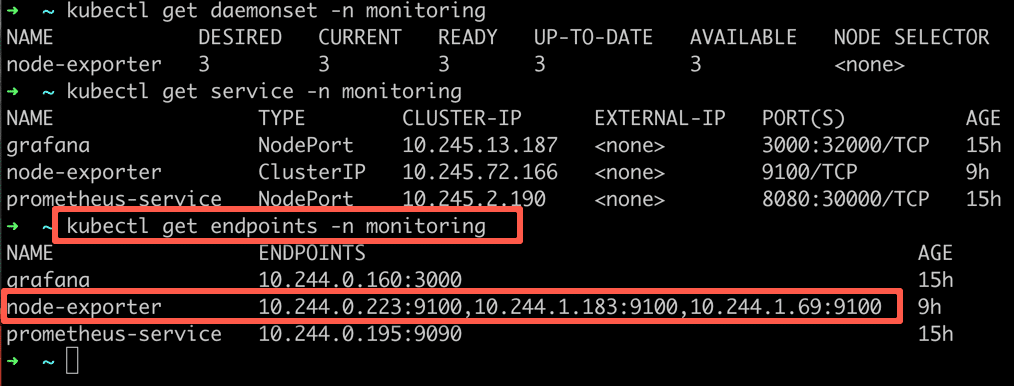
targetPort: 9100

Step 22: Create the service.

$ kubectl create -f service.yaml

Step 23: Now, check the service’s endpoints and see if it is pointing to all the daemon set pods.

$ kubectl get endpoints -n monitoring



As you can see from the above output, the node-exporter service has three endpoints. Meaning three node-exporter pods running on three nodes as part of Daemon set. Grafana

# Grafana Setup on standalone server

Step 1: Install Required Packages

$ sudo apt-get install -y apt-transport-https software-propertiescommon wget

### Step 2**:** Create Directory for GPG Key

Create a directory to store the GPG key used to authenticate the Grafana repository.

$ sudo mkdir -p /etc/apt/keyrings/

Step 3: Download and Store Grafana GPG Key

$ wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor | sudo tee /etc/apt/keyrings/grafana.gpg > /dev/null

### Step 4**: Add Grafana Stable Repository**

Add the stable Grafana repository to your system’s sources list.

$ echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main" | sudo tee -a /etc/apt/sources.list.d/grafana.list

### Step 5: (Optional) Add Grafana Beta Repository

$ echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com beta main" | sudo tee -a /etc/apt/sources.list.d/grafana.list

### Step 6: Verify the Sources List

Navigate to the sources list directory and review the newly added Grafana repositories.

$ cd /etc/apt/sources.list.d/

sudo vim grafana.list

Step 7: Update Package Lists

$ sudo apt-get update

### Step 8: Install Grafana

Now, install the Grafana package.

$ sudo apt-get install grafana

### Step 10: Start Grafana Server

Start the Grafana server service.

$ sudo systemctl start grafana-server.service

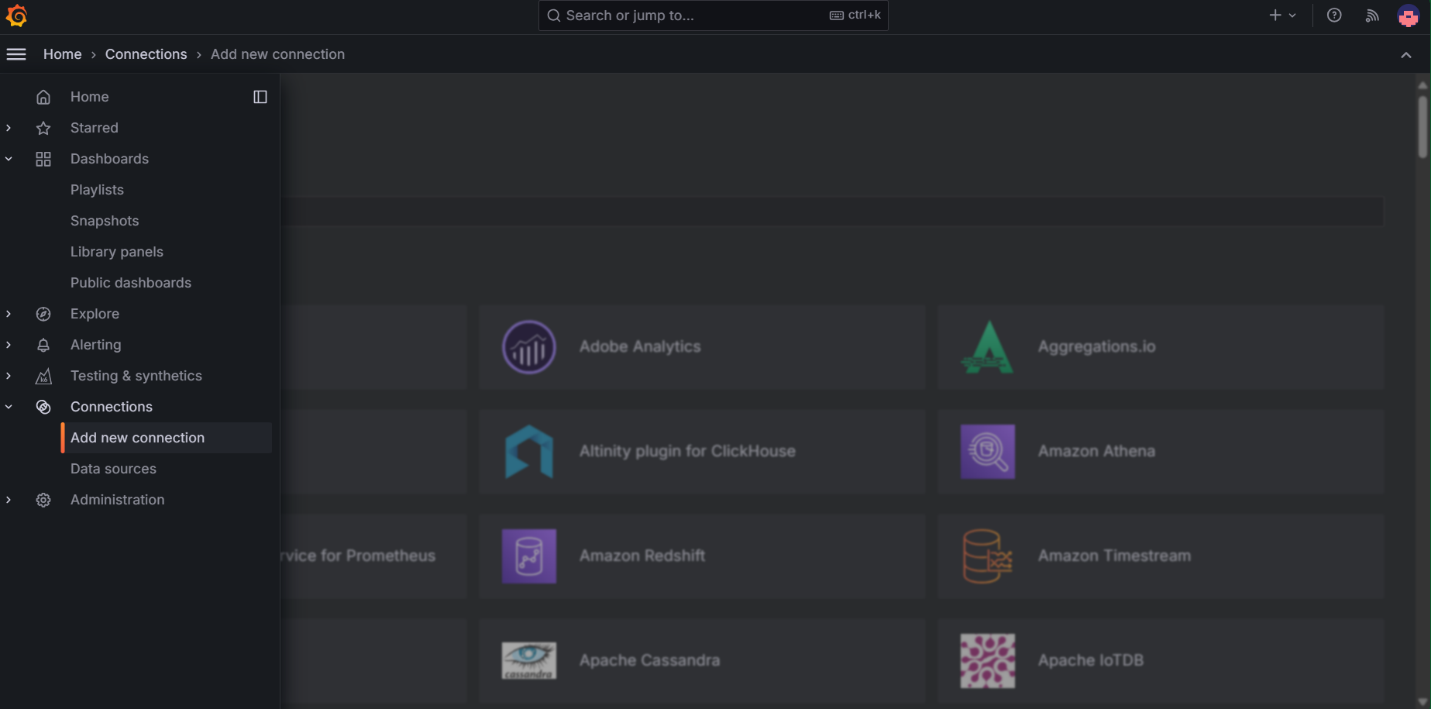
### Step 11: Check Grafana Server Status

Check the status of the Grafana server to ensure it’s running correctly.

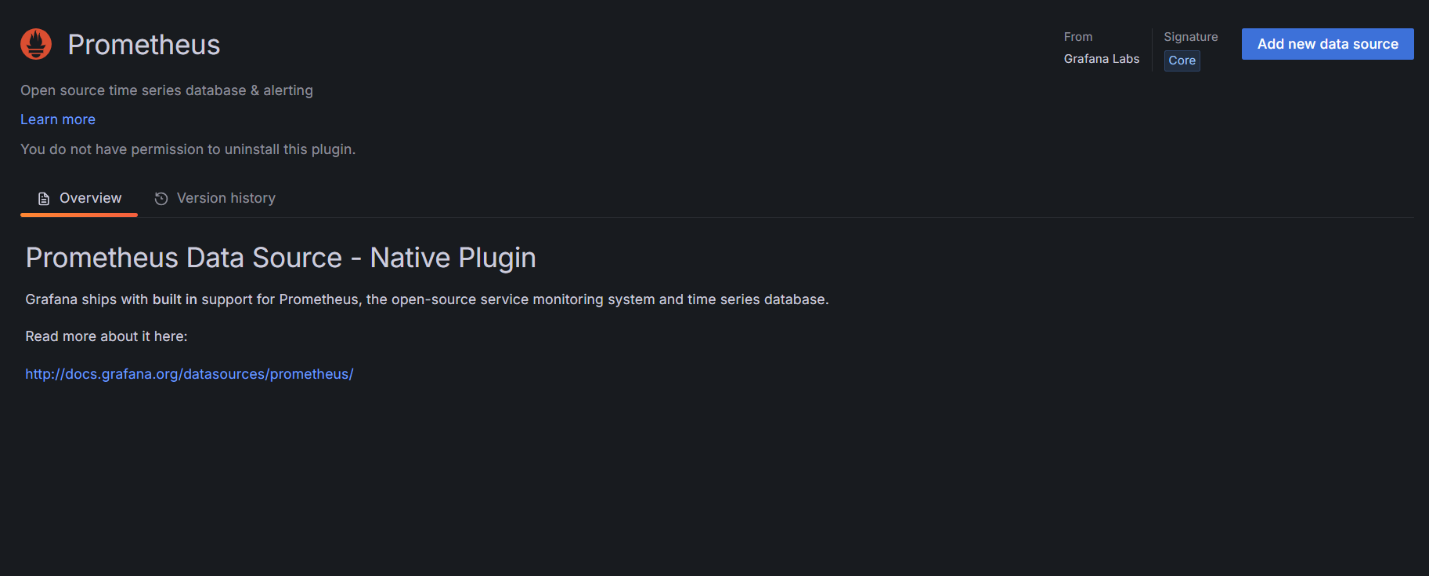
$ sudo systemctl status grafana-server.service

Step 12 : login with user and password

Step 13: add connection search the Prometheus and it



Step 14: add data source in Prometheus



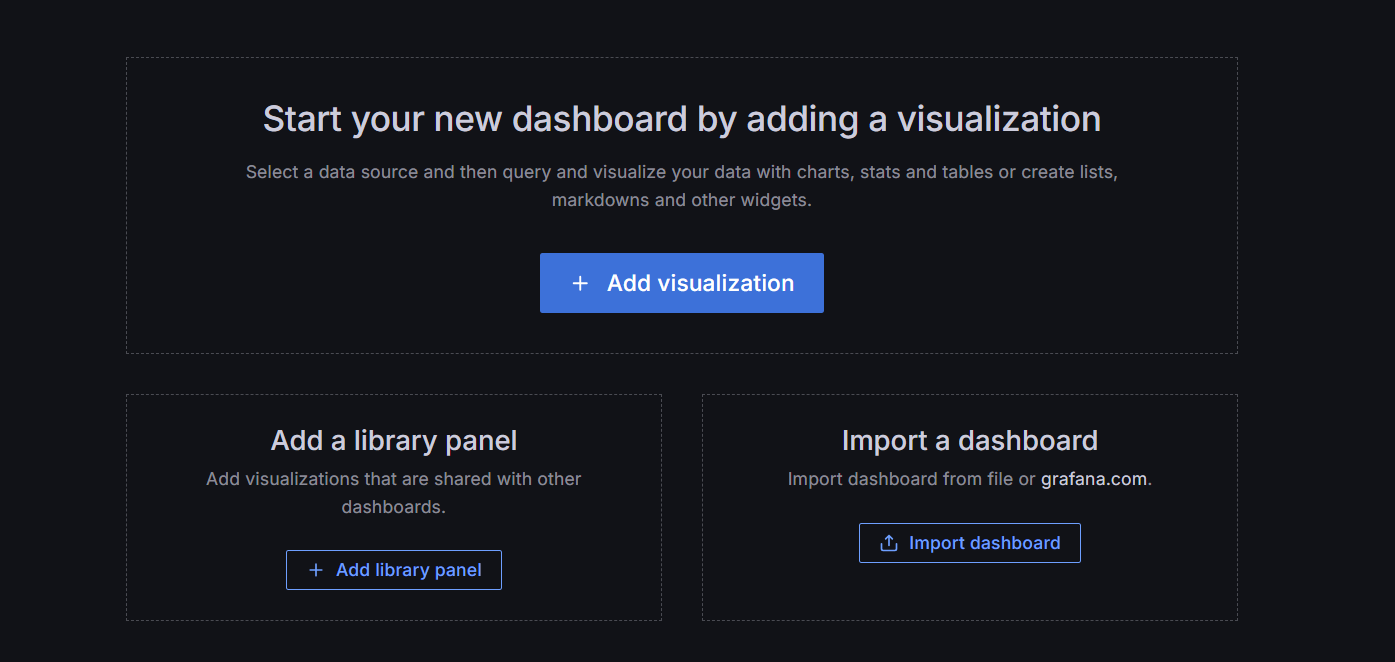
Enter the fill details regarding Prometheus

### Connection

Prometheus server URL \*

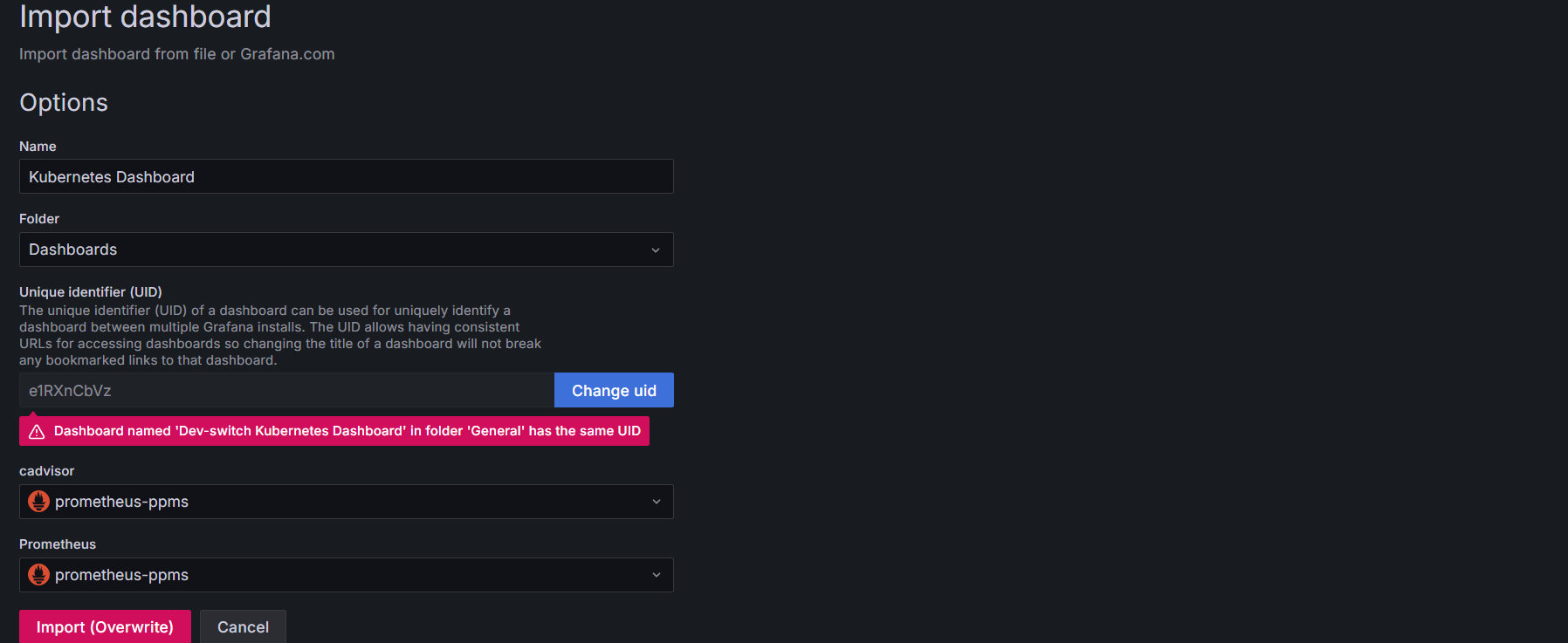
Step 15: Goto dashboard add now dashboard

Step 16: go to import dashboard

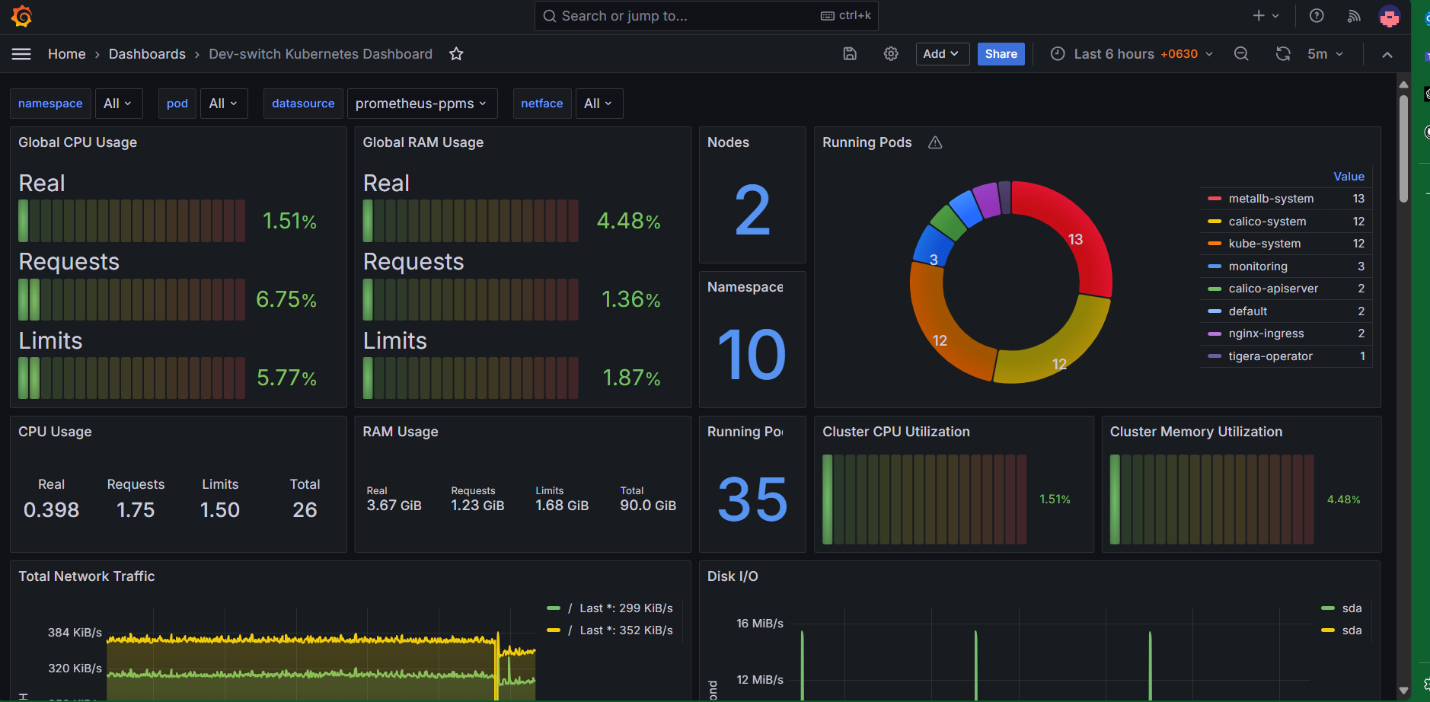


Step 17: In google search Grafana dashboard and download the dashboard as per requirement

Step 18: import the dashboard from download give Prometheus connection in dashboard



Step 19:- we can see grafana dashboard

  
  
  
  
<https://devopscube.com/setup-prometheus-monitoring-on-kubernetes/>

[https://devopscube.com/setup-kube-state-metrics/](https://devopscube.com/setup-kube-state-metrics/" \o "https://devopscube.com/setup-kube-state-metrics/" \t "_blank)

[How To Setup Prometheus Node Exporter On Kubernetes (devopscube.com)](https://devopscube.com/node-exporter-kubernetes/" \o "https://devopscube.com/node-exporter-kubernetes/" \t "_blank)