# **Deploy Prometheus & Grafana on Kubernetes Cluster**

# **Introduction**

Monitoring is a crucial aspect of any jobs for technologies like Kubernetes. A robust monitoring setup can bolster your confidence to migrate production workloads from VMs to Containers.

There is an opensource monitoring solution. Prometheus and Grafana. This tool is a perfect combination for monitoring. Where Prometheus server is a metric storage or time-series database to store the metrics. Grafana is a visualization tool.

# **Requirements**

My environment uses 1CPU cores and 4GB of RAM per node.

1 master nodes. 2 worker nodes. 50GB NFS storage attached to worker nodes as a Persistent Volume. You can use your Kubernetes environment. I would suggest you use 8 CPUs and 8GB of RAM per node with 1 Master and 2 Worker or more.

# **Deployment of Prometheus on Kubernetes**

Let's get started with the setup.

First, we will create a Kubernetes namespace for all our monitoring components.

apiVersion: v1

kind: Namespace

metadata:

name: monitoring

monitoring namespace

We should create a config map with all the Prometheus scrape config, which will be mounted to the Prometheus container in /etc/prometheus as prometheus.yaml file.  
Below you can easily add your targets and job name for Prometheus configuration.

apiVersion: v1

kind: ConfigMap

metadata:

name: prometheus-config

namespace: monitoring

data:

prometheus.yml: |

# my global config

global:

scrape\_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.

evaluation\_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.

# scrape\_timeout is set to the global default (10s).

# Alertmanager configuration

alerting:

alertmanagers:

- static\_configs:

- targets:

# - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation\_interval'.

rule\_files:

# - "first\_rules.yml"

# - "second\_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:

# Here it's Prometheus itself.

scrape\_configs:

# The job name is added as a label `job=<job\_name>` to any timeseries scraped from this config.

- job\_name: 'prometheus'

# metrics\_path defaults to '/metrics'

# scheme defaults to 'http'.

static\_configs:

- targets: ['localhost:9090']

Prometheus conf

Create PV and PVC for Prometheus persistent volume

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv-nfs-data

namespace: monitoring

labels:

type: nfs

app: prometheus-deployment

spec:

storageClassName: managed-nfs

capacity:

storage: 1Gi

accessModes:

- ReadWriteMany

nfs:

server: 192.168.31.174

path: "/mnt/nfs/promdata

Persistent Volume

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: pvc-nfs-data

namespace: monitoring

labels:

app: prometheus-deployment

spec:

storageClassName: managed-nfs

accessModes:

- ReadWriteMany

resources:

requests:

storage: 500Mi

Persistent Volume Claim

Create a file named prometheus-deployment.yaml and copy the following contents onto the file. In this configuration, we are mounting the Prometheus config map as a file inside /etc/prometheus. It uses the official Prometheus image from the docker hub.

apiVersion: apps/v1

kind: Deployment

metadata:

name: prometheus

namespace: monitoring

labels:

app: prometheus

spec:

replicas: 1

strategy:

rollingUpdate:

maxSurge: 1

maxUnavailable: 1

type: RollingUpdate

selector:

matchLabels:

app: prometheus

template:

metadata:

labels:

app: prometheus

annotations:

prometheus.io/scrape: "true"

prometheus.io/port: "9090"

spec:

containers:

- name: prometheus

image: prom/prometheus

args:

- '--storage.tsdb.retention=6h'

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

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

ports:

- name: web

containerPort: 9090

volumeMounts:

- name: prometheus-config-volume

mountPath: /etc/prometheus

- name: prometheus-storage-volume

mountPath: /prometheus

restartPolicy: Always

volumes:

- name: prometheus-config-volume

configMap:

defaultMode: 420

name: prometheus-config

- name: prometheus-storage-volume

persistentVolumeClaim:

claimName: pvc-nfs-data

Deployment.yaml

To access the Prometheus dashboard over an IP or a DNS name, you need to expose it as Kubernetes service with NodePort or a Load Balancer.

apiVersion: v1

kind: Service

metadata:

name: prometheus-service

namespace: monitoring

annotations:

prometheus.io/scrape: 'true'

prometheus.io/port: '9090'

spec:

selector:

app: prometheus

type: NodePort

ports:

- port: 8080

targetPort: 9090

nodePort: 30000

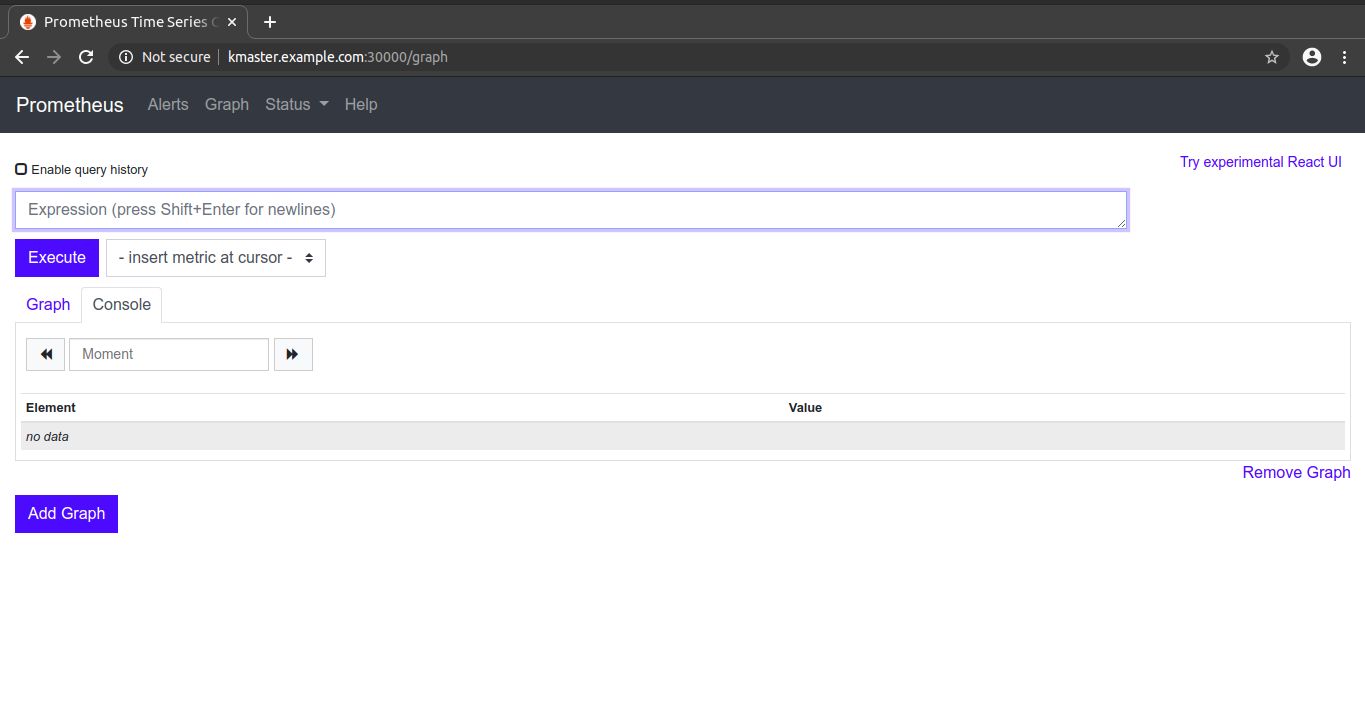
service

Creating all Prometheus related resources you can use Kustomization file and deploy it simply using bellow command.

kubectl apply -k . -n monitoring

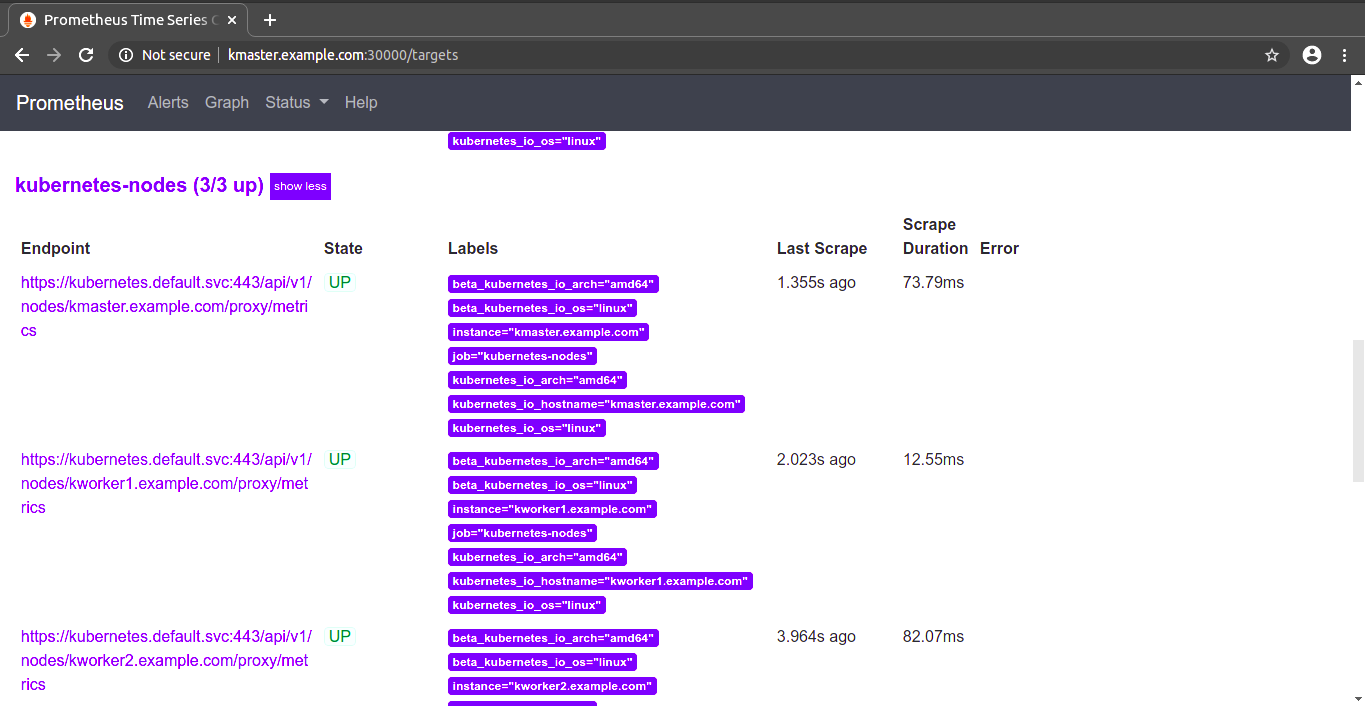
Prometheus Deployment

Now you can access the Prometheus dashboard using any Kubernetes node IP on port 30000.



Dashboard

Now if you browse to status --> Targets, you will see all the Kubernetes endpoints connected to Prometheus automatically using service discovery as shown below.



Prometheus Dashboard Targets View

# **Setting up Grafana for Viewing Metrics**

Grafana is an ****open**** platform for ****beautiful**** analytics and monitoring. Prometheus has a basic expression browser for debugging purpose but to have a good-looking dashboard, use Grafana. Grafana has a data source ready to query Prometheus.

Using Grafana you can create dashboards from Prometheus metrics to monitor the Kubernetes cluster.

Let's get started with the setup.

Creating ConfigMap for Grafana which consist of configuration information about data source Prometheus. If you have more data sources, you can add more data sources with different YAMLs under the data section.

apiVersion: v1

kind: ConfigMap

metadata:

name: grafana-datasources

namespace: monitoring

data:

prometheus.yaml: |-

{

"apiVersion": 1,

"datasources": [

{

"access":"proxy",

"editable": true,

"name": "prometheus",

"orgId": 1,

"type": "prometheus",

"url": "http://prometheus-service.monitoring.svc:8080",

"version": 1

}

]

}

data-source configuration

Create PV and PVC for Grafana persistent volume

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv-nfs-storage

namespace: monitoring

labels:

type: nfs

spec:

storageClassName: managed-nfs

capacity:

storage: 1Gi

accessModes:

- ReadWriteMany

nfs:

server: 192.168.31.174

path: "/mnt/nfs/grafana-storage"

Persistent Volume

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: pvc-nfs-storage

namespace: monitoring

labels:

app: grafana

spec:

storageClassName: managed-nfs

accessModes:

- ReadWriteMany

resources:

requests:

storage: 500Mi

Persistent Volume Claim

Deploy Grafan using deployment file

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

volumeMounts:

- mountPath: /var/lib/grafana

name: grafana-storage

- mountPath: /etc/grafana/provisioning/datasources

name: grafana-datasources

readOnly: false

volumes:

- name: grafana-storage

persistentVolumeClaim:

claimName: pvc-nfs-storage

- name: grafana-datasources

configMap:

defaultMode: 420

name: grafana-datasources

To access the Grafana dashboard over an IP or a DNS name, you need to expose it as Kubernetes service with NodePort or a Load Balancer.

apiVersion: v1

kind: Service

metadata:

name: grafana

namespace: monitoring

annotations:

prometheus.io/scrape: 'true'

prometheus.io/port: '3000'

spec:

selector:

app: grafana

type: NodePort

ports:

- port: 3000

targetPort: 3000

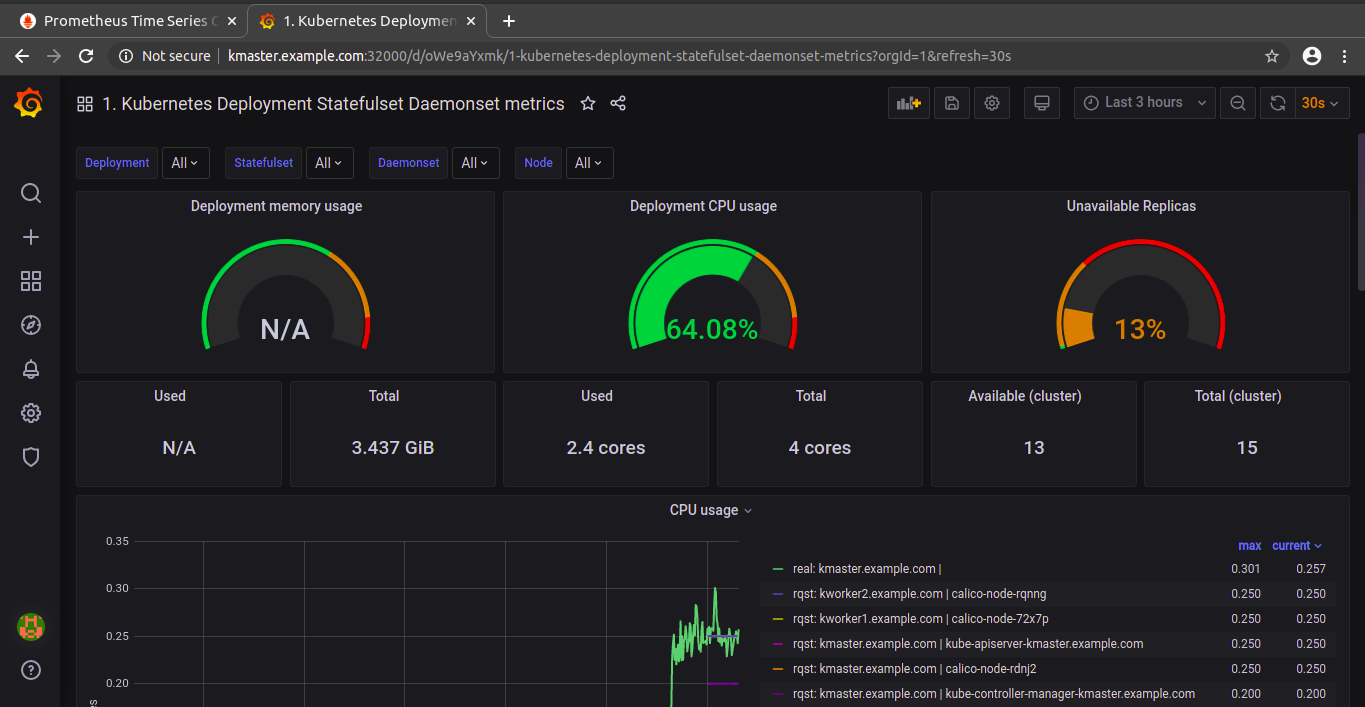
nodePort: 32000

Creating all Prometheus related resources you can use Kustomization file and deploy it simply using bellow command.

cd Grafana  
kubectl apply -k . -n monitoring

Let’s head to Grafana (http://yourhost.com:32000) and set up the Prometheus Dashboard.

I used the template ID 8588 from grafana public template list and import to grafana dashboard



Grafana Dashboard