

How To Setup Prometheus Monitoring On Kubernetes Cluster

[Prometheus](#) is an open source monitoring framework. Explaining Prometheus is out of the scope of this article. In this article, I will guide you to setup Prometheus on a Kubernetes cluster and collect node, pods and services metrics automatically using Kubernetes service discovery configurations. If you want to know more about Prometheus, You can watch all the Prometheus related videos [from here](#).

Prometheus Monitoring On Kubernetes

I assume that you have a kubernetes cluster up and running with kubectl setup on your workstation. If you don't have a kubernetes setup, you can set up a cluster on google cloud by [following this article](#).

Latest Prometheus is available as a [docker image](#) in its official docker hub account. We will use that image for the setup.

Let's get started with the setup.

All the configuration files I mentioned in this guide is hosted on [Github](#). You can clone the repo using the following command. Thanks to [James](#) for contributing to this repo. Please don't hesitate to contribute to the repo for adding features.

Create A Namespace:-

First, we will create a Kubernetes namespace for all our monitoring components. Execute the following command to create a new namespace called monitoring.

```
kubectl create namespace monitoring
```

You need to assign cluster reader permission to this namespace so that Prometheus can fetch the metrics from kubernetes API's.

1. Create a file named clusterRole.yaml and copy the content of this file → [ClusterRole Config](#)
2. Create the role using the following command

```
# kubectl create -f clusterRole.yaml
```

Create A Config Map:-

We should create a config map with all the [prometheus scrape config](#) and alerting rules, which will be mounted to the Prometheus container in /etc/prometheus as prometheus.yaml and prometheus.rules files. The prometheus.yaml contains all the configuration to dynamically discover pods and services running in the kubernetes cluster. prometheus.rules will contain all the alert rules for sending alerts to alert manager.

1. Create a file called config-map.yaml and copy the contents of this file → [Prometheus Config File](#)

2. Execute the following command to create the config map in kubernetes.

```
kubectl create -f config-map.yaml -n monitoring
```

Create A Prometheus Deployment

1. 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 docker hub.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: prometheus-deployment
  namespace: monitoring
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: prometheus-server
    spec:
      containers:
        - name: prometheus
          image: prom/prometheus:v2.1.0
          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: {}
```

4. Create a deployment on monitoring namespace using the above file.

```
kubectl create -f prometheus-deployment.yaml --namespace=monitoring
```

5. You can check the created deployment using the following command.

```
kubectl get deployments --namespace=monitoring
```

You can also get details from the kubernetes dashboard like shown below.

The screenshot displays the Kubernetes dashboard interface. On the left, a sidebar menu shows the 'Workloads' section selected. The main panel is divided into three sections: Deployments, Pods, and Replica Sets. The 'Deployments' section shows a single deployment named 'prometheus-monitoring' with 1 pod. The 'Pods' section shows a single pod named 'prometheus-monitoring-3331088907-h...' in a 'Running' state. The 'Replica Sets' section shows a single replica set named 'prometheus-monitoring-3331088...' with 1 pod.

Name	Labels	Pods	Age	Images
prometheus-monitoring	app: prometheus-server	1 / 1	35 seconds	prom/prometheus

Name	Status	Restarts	Age	CPU (cores)	Memory (bytes)
prometheus-monitoring-3331088907-h...	Running	0	35 seconds	-	2.980 Mi

Name	Labels	Pods	Age	Images
prometheus-monitoring-3331088...	app: prometheus-server pod-template-hash: 3331088...	1 / 1	35 seconds	prom/prometheus

Connecting To Prometheus

You can connect to the deployed Prometheus in two ways.

1. Using Kubectl port forwarding
2. Exposing the Prometheus deployment as a service with NodePort or a Load Balancer.

We will look at both the options.

Using Kubectl Port Forwarding

Using kubectl port forwarding, you can access the pod from your workstation using a selected port on your localhost.

1. First, get the Prometheus pod name.

```
# kubectl get pods --namespace=monitoring
```

The output will look like the following.

```
# kubectl get pods --namespace=monitoring
NAME                                READY    STATUS    RESTARTS    AGE
prometheus-monitoring-3331088907-hm5n1  1/1      Running   0
```

2. Execute the following command with your pod name to access Prometheus from localhost port 8080.

Note: Replace prometheus-monitoring-3331088907-hm5n1 with your pod name.

```
# kubectl port-forward prometheus-monitoring-3331088907-hm5n1 8080:9090 -n monitoring
```

3. Now, if you access <http://localhost:8080> on your browser, you will get the Prometheus home page.

Exposing Prometheus As A Service

To access the Prometheus dashboard over a IP or a DNS name, you need to expose it as kubernetes service.

1. 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
spec:
  selector:
    app: prometheus-server
  type: NodePort
  ports:
    - port: 8080
      targetPort: 9090
      nodePort: 30000
```

Note: If you are on AWS or Google Cloud, You can use Loadbalancer type, which will create a load balancer and points it to the service.

2. Create the service using the following command.

```
# kubectl create -f prometheus-service.yaml --namespace=monitoring
```

3. Once created, you can access the Prometheus dashboard using any Kubernetes node IP on port 30000. If you are on the cloud, make sure you have the right firewall rules for accessing the apps.

[Prometheus](#) [Alerts](#) [Graph](#) [Status ▾](#) [Help](#)

Expression (press Shift+Enter for newlines)

Execute

- insert metric at cursor - ▴ ▾

Graph

Console

Element

no data

Add Graph

- Now if you go to status → Targets, you will see all the Kubernetes endpoints connected to Prometheus automatically using service discovery as shown below. So you will get all kubernetes container and node metrics in Prometheus.

Prometheus	Alerts	Graph	Status ▾	Help
------------	--------	-------	----------	------

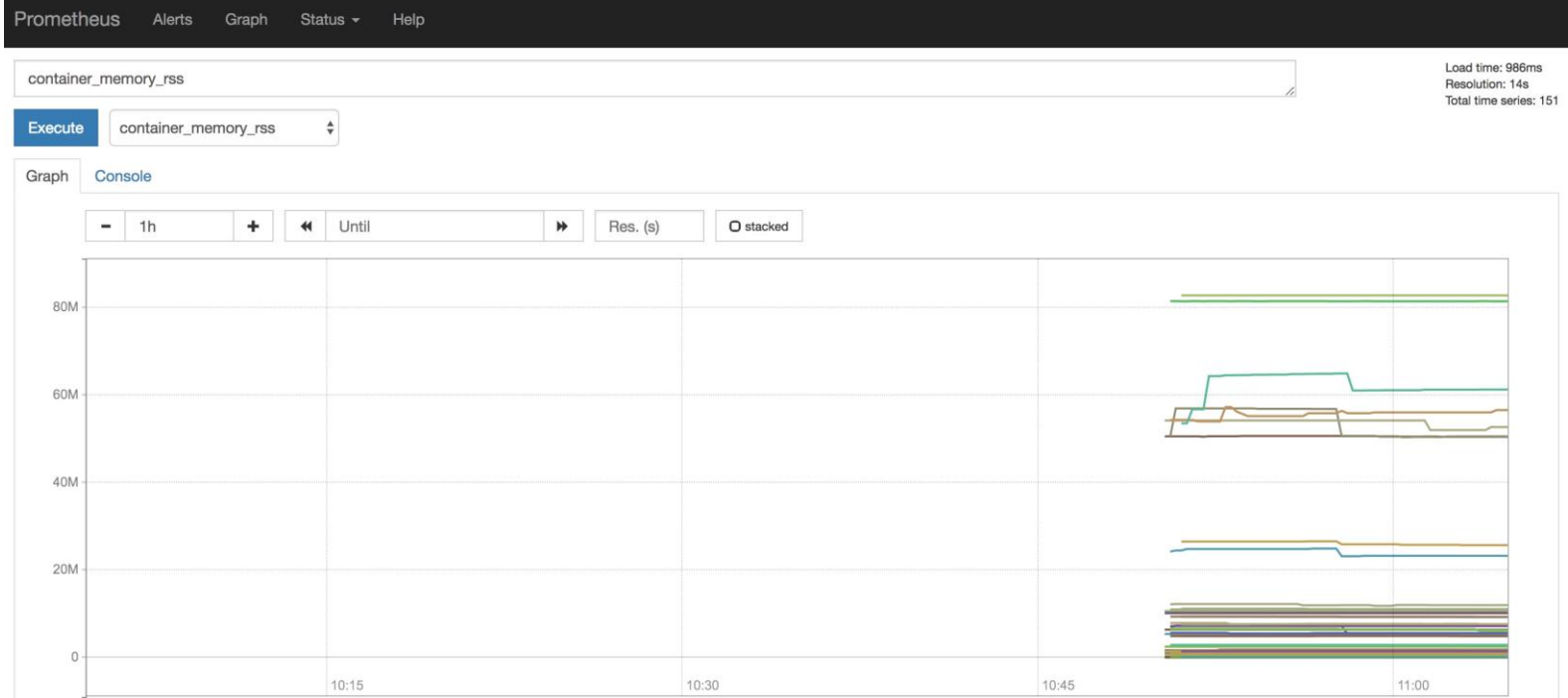
Targets

kubernetes-apiservers (1/1 up)				
Endpoint	State	Labels	Last Scrape	Error
https://35.193.112.250:443/metrics	UP	instance="35.193.112.250:443"	4.793s ago	

kubernetes-cadvisor (2/2 up)				
Endpoint	State	Labels	Last Scrape	Error
https://kubernetes.default.svc:443/api/v1/nodes/gke-deveopscube-demo-default-pool-142449aa-mjl9/proxy/metrics/cadvisor	UP	beta_kubernetes_io_arch="amd64" beta_kubernetes_io_fluentd_ds_ready="true" beta_kubernetes_io_instance_type="n1-standard-1" beta_kubernetes_io_os="linux" cloud_google_com_gke_nodepool="default-pool" failure_domain_beta_kubernetes_io_region="us-central1" failure_domain_beta_kubernetes_io_zone="us-central1-a" instance="gke-deveopscube-demo-default-pool-142449aa-mjl9" kubernetes_io_hostname="gke-deveopscube-demo-default-pool-142449aa-mjl9"	3.512s ago	
https://kubernetes.default.svc:443/api/v1/nodes/gke-deveopscube-demo-default-pool-142449aa-n80t/proxy/metrics/cadvisor	UP	beta_kubernetes_io_arch="amd64" beta_kubernetes_io_fluentd_ds_ready="true" beta_kubernetes_io_instance_type="n1-standard-1" beta_kubernetes_io_os="linux" cloud_google_com_gke_nodepool="default-pool" failure_domain_beta_kubernetes_io_region="us-central1" failure_domain_beta_kubernetes_io_zone="us-central1-a" instance="gke-deveopscube-demo-default-pool-142449aa-n80t" kubernetes_io_hostname="gke-deveopscube-demo-default-pool-142449aa-n80t"	2.24s ago	

kubernetes-nodes (2/2 up)				
Endpoint	State	Labels	Last Scrape	Error
https://kubernetes.default.svc:443/api/v1/nodes/gke-deveopscube-demo-default-pool-142449aa-mjl9/proxy/metrics	UP	beta_kubernetes_io_arch="amd64" beta_kubernetes_io_fluentd_ds_ready="true" beta_kubernetes_io_instance_type="n1-standard-1" beta_kubernetes_io_os="linux" cloud_google_com_gke_nodepool="default-pool" failure_domain_beta_kubernetes_io_region="us-central1" failure_domain_beta_kubernetes_io_zone="us-central1-a" instance="gke-deveopscube-demo-default-pool-142449aa-mjl9" kubernetes_io_hostname="gke-deveopscube-demo-default-pool-142449aa-mjl9"	950ms ago	
https://kubernetes.default.svc:443/api/v1/nodes/gke-deveopscube-demo-default-pool-142449aa-n80t/proxy/metrics	UP	beta_kubernetes_io_arch="amd64" beta_kubernetes_io_fluentd_ds_ready="true" beta_kubernetes_io_instance_type="n1-standard-1" beta_kubernetes_io_os="linux" cloud_google_com_gke_nodepool="default-pool" failure_domain_beta_kubernetes_io_region="us-central1" failure_domain_beta_kubernetes_io_zone="us-central1-a" instance="gke-deveopscube-demo-default-pool-142449aa-n80t" kubernetes_io_hostname="gke-deveopscube-demo-default-pool-142449aa-n80t"	1.374s ago	

- You can head over the homepage and select the metrics you need from the drop-down and get the graph for the time range you mention. An example graph for container memory utilization is shown below.



5.