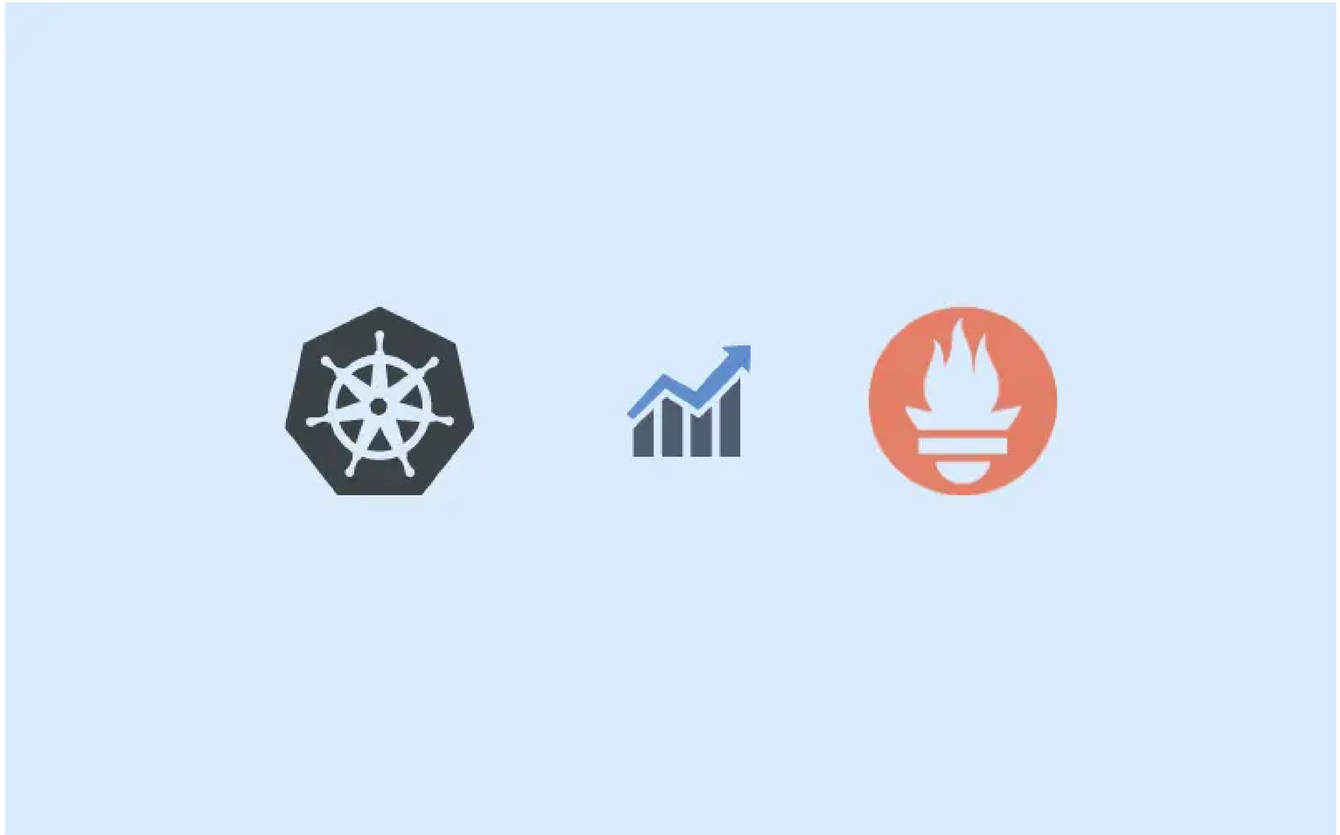
D — DEVOPS

How to Setup Prometheus Node Exporter on Kubernetes

by **devopscube** · April 6, 2021



If you want to know how the Kubernetes nodes perform or monitor system-level insights of kubernetes nodes, you need to set up a Prometheus node exporter on Kubernetes cluster.

This guide will walk you through the node-exporter setup on a Kubernetes cluster and integrate Prometheus scrape config to scrape the node metrics.

What is Prometheus Node Exporter?

It collects all the hardware and Operating System level metrics that are exposed by the kernel.

You can use the node exporter to collect the system metrics from all your Linux systems. Check this article on node monitoring using node-exporter.

Why do we need Node Exporter on Kubernetes?

By default, most of the Kubernetes clusters expose the metric server metrics (Cluster level metrics from the summary API) and Cadvisor (Container level metrics). It does not provide detailed node-level metrics.

To get all the kubernetes node-level system metrics, you need to have a node-exporter running in all the kubernetes nodes. It collects all the Linux system metrics and exposes them via `/metrics` endpoint on port `9100`

Similarly, you need to install Kube state metrics to get all the metrics related to kubernetes objects.

Kubernetes Manifests

The Kubernetes manifest used in this guide is present in the Github repository. Clone the repo to your local system.

```
git clone https://github.com/bibinwilson/kubernetes-node-exporter
```

Setup Node Exporter on Kubernetes

> **Note:** If you don't have the Prometheus setup, please follow my guide on setting up Prometheus on kubernetes.

Here is what we are going to do.

- Deploy node exporter on all the Kubernetes nodes as a `daemonset`. Daemonset makes sure one instance of node-exporter is running in all the nodes. It exposes all the node metrics on port `9100` on the `/metrics` endpoint

- Create a service that listens on port `9100` and points to all the daemonset node exporter pods. We would be monitoring the service endpoints (Node exporter pods) from Prometheus using the endpoint job config. More explanation on this in the Prometheus config part.

Lest get started with the setup.

Step 1: Create a file name `daemonset.yaml` and copy the following content.

> **Note:** This Daemonset will be deployed in the monitoring namespace. If you wish to deploy it in a different namespace, change it in the following YAML

```
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: root
          readOnly: true
      volumes:
      - hostPath:
          path: /sys
        name: sys
      - hostPath:
          path: /
        name: root
```

Step 2: Deploy the daemonset using the kubectl command.

```
kubectl create -f daemonset.yaml
```

Step 3: List the `daemonset` in the monitoring namespace and make sure it is in the available state.

```
kubectl get daemonset -n monitoring
```

Step 4: 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 5: Create the service.

```
kubectl create -f service.yaml
```

Step 6: Now, check the service's endpoints and see if it is pointing to all the daemonset pods.

```
kubectl get endpoints -n monitoring
```

```
➜  ~ kubectl get daemonset -n monitoring
NAME            DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
node-exporter   3         3         3       3            3           <none>
➜  ~ kubectl get service -n monitoring
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
grafana             NodePort    10.245.13.187   <none>        3000:32000/TCP   15h
node-exporter       ClusterIP   10.245.72.166   <none>        9100/TCP         9h
prometheus-service  NodePort    10.245.2.190    <none>        8080:30000/TCP   15h
➜  ~ kubectl get endpoints -n monitoring
NAME                ENDPOINTS                                                     AGE
grafana             10.244.0.160:3000                                             15h
node-exporter       10.244.0.223:9100,10.244.1.183:9100,10.244.1.69:9100          9h
prometheus-service  10.244.0.195:9090                                             15h
➜  ~
```

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 Daemonset.

Node-exporter Prometheus Config

We have the node-exporter daemonset running on port 9100 and a service pointing to all the node-exporter pods.

You need to add a scrape config to the Prometheus config file to discover all the node-exporter pods.

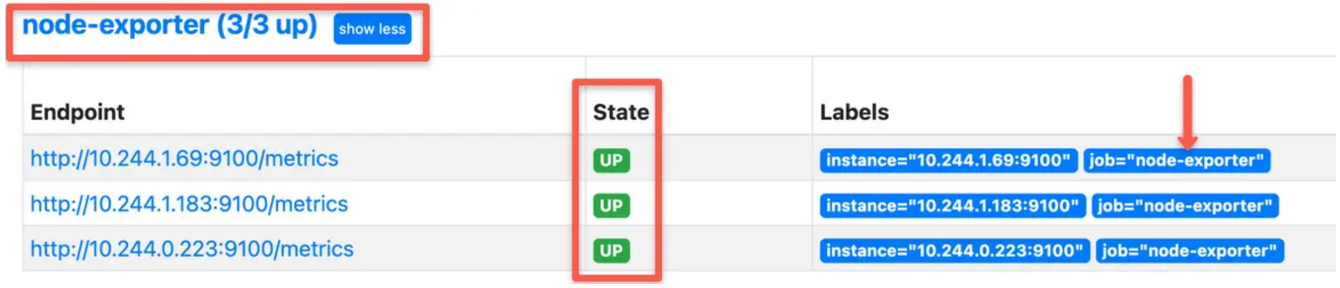
Let's take a look at the Prometheus scrape config required to scrape the node-exporter metrics.

```
      - job_name: 'node-exporter'
        kubernetes_sd_configs:
          - role: endpoints
        relabel_configs:
        - source_labels: [__meta_kubernetes_endpoints_name]
          regex: 'node-exporter'
          action: keep
```

In this config, we mention the role as endpoints to scrape the endpoints with the name `node-exporter` .

See Prometheus config map file I have created for the Kubernetes monitoring stack. It includes all the scrape configs for kubernetes components.

Once you add the scrape config to Prometheus, you will see the node-exporter targets in Prometheus, as shown below



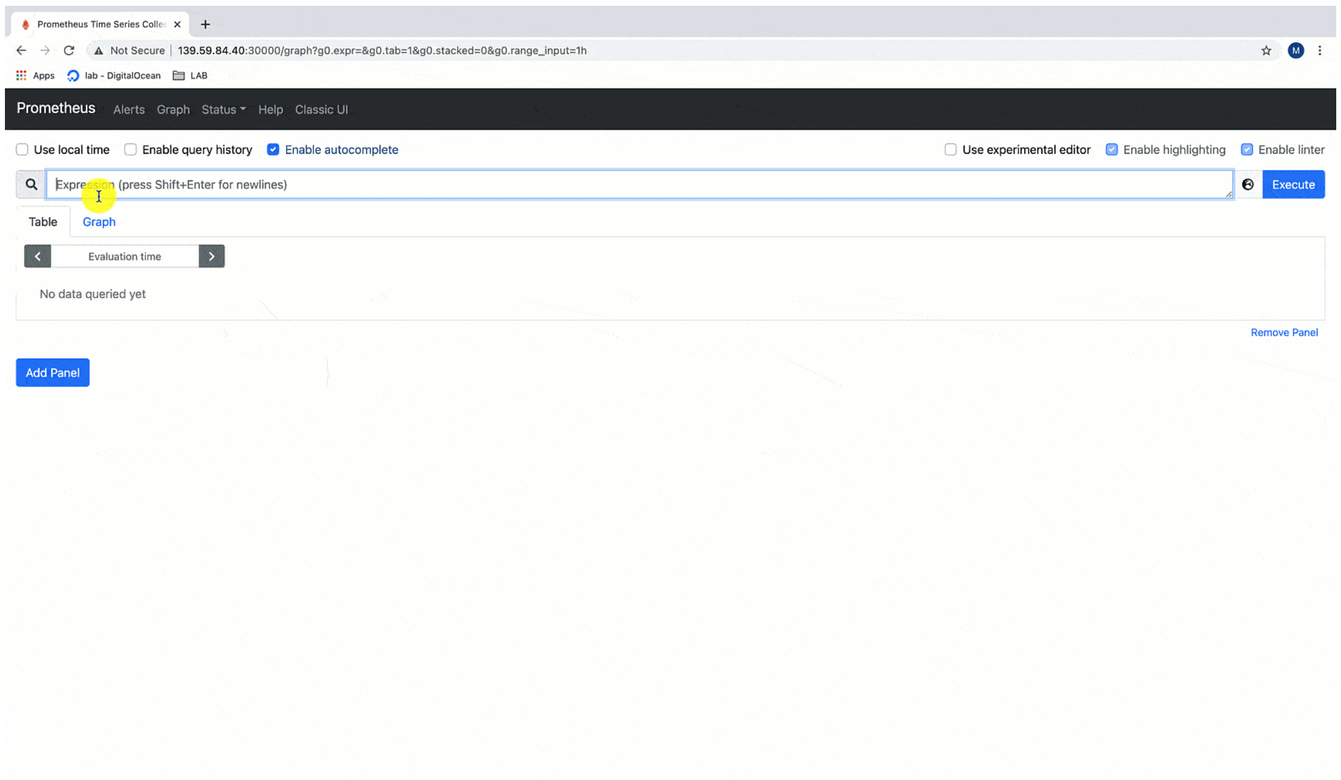
Querying Node-exporter Metrics in Prometheus

Once you verify the node-exporter target state in Prometheus, you can query the Prometheus dashboard's available node-exporter metrics.

All the metrics from node-exporter is prefixed with `node_`

You can query the metrics with different PromQL expressions. See querying basics to learn about PromQL queries.

If you type `node_` in the Prometheus dashboard, it will list all the available metrics as shown below.



Visualizing Prometheus Node Exporter Metrics as Grafana Dashboards

Visualising the node exporter metrics on Grafana is not difficult as you think.

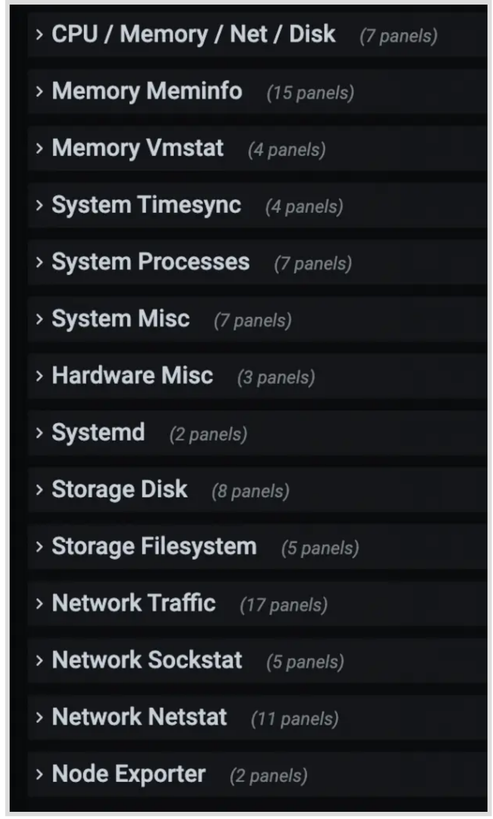
A community Grafana node exporter dashboard template has a predefined dashboard with all the supported node exporter metrics.

You can modify the template as per your project requirements.

If you don't know how to import a community template, please check my Grafana Prometheus integration article, where I have added the steps to import community dashboard templates.

So here is how the node-exporter Grafana dashboard looks for CPU/memory and disk statistics.

Once you have the dashboard, you will find the following sections. If you expand it, you will find all the metrics panel.



More References

1. Official Node exporter Github repository
2. Prometheus Linux host metrics guide
3. Prometheus Exporters

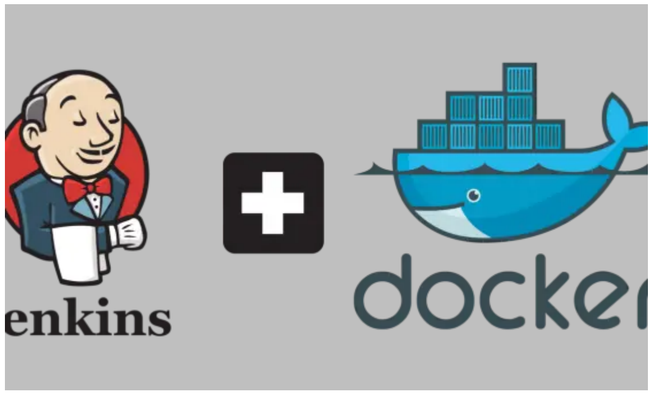
devopscube

Established in 2014, a community for developers and system admins. Our goal is to continue to build a growing DevOps community offering the best in-depth articles, interviews, event listings, whitepapers, infographics and much more on DevOps.

List of Open Source Tools Every DevOps Engineer Should Use

by **Bibin Wilson** · February 28, 2021

As a DevOps engineer or a developer, you have to shut down those distractions and deliver high-quality infrastructure…



D — DEVOPS

How to Setup AWS ECS Cluster as Build Slave for Jenkins

by **devopscube** · April 10, 2017

In our last post, we wrote about setting up docker containers as build slaves. Integrating docker into your…

D — DEVOPS

How to Configure SSL on Jenkins Server

by **devopscube** · April 9, 2020

It is very important to secure Jenkins by enabling SSL which runs in a project environment. This article…

D — DEVOPS

How To Setup a etcd Cluster On Linux – Beginners Guide

by **devopscube** · November 8, 2018

Introduction etcd is an open source key-value store for storing and retrieving configurations. It is a core component…

Setup NFS Server On Google Cloud – Managed Cloud FileStore Service

by **devopscube** · August 16, 2018

Google Cloud Filestore is a managed NFS implementation on google cloud. This is one of the awaited features...

D — DEVOPS

How to Setup Prometheus Monitoring On Kubernetes Cluster

by **Bibin Wilson** · April 7, 2021

This article will guide you through setting up Prometheus on a Kubernetes cluster for monitoring the Kubernetes cluster....