Prometheus configuration:

## 🚀 Phase 1: Install and Configure Prometheus Server (on Debian 12)

### ✅ ****Step 1: Create a Dedicated Prometheus User****

sudo useradd --no-create-home --shell /usr/sbin/nologin prometheus

🔍 **Why?**

Security best practice: runs Prometheus with limited permissions

Prevents it from accessing unnecessary system files

Avoids running it as root (unsafe in production)

### ✅ ****Step 2: Create Required Directories****

sudo mkdir /etc/prometheus

sudo mkdir /var/lib/prometheus

🔍 **Why?**

/etc/prometheus: stores config files (like prometheus.yml)

/var/lib/prometheus: stores time-series metrics database

Keeping configs and data separate improves organization and backup handling

### ✅ ****Step 3: Download the Latest Prometheus Binary****

cd /tmp

wget https://github.com/prometheus/prometheus/releases/download/v2.52.0/prometheus-2.52.0.linux-amd64.tar.gz

🔍 **Why?**

You’re getting the official, latest stable version

Using /tmp keeps your system organized — it's a temporary directory for downloads

### ✅ ****Step 4: Extract and Install Prometheus Binaries****

tar xvf prometheus-2.52.0.linux-amd64.tar.gz

cd prometheus-2.52.0.linux-amd64

Then move the binaries:

sudo cp prometheus /usr/local/bin/

sudo cp promtool /usr/local/bin/

🔍 **Why?**

prometheus: the main server binary

promtool: a helper tool to validate configs

/usr/local/bin: standard location for custom binaries (makes them available system-wide

### ✅ ****Step 5: Copy Console Files and Example Config****

sudo cp -r consoles /etc/prometheus

sudo cp -r console\_libraries /etc/prometheus

sudo cp prometheus.yml /etc/prometheus/

🔍 **Why?**

The web UI needs these console files for visual dashboards

You copy the example prometheus.yml to start with a working config

### ✅ ****Step 6: Set File Ownership to Prometheus User****

sudo chown -R prometheus:prometheus /etc/prometheus /var/lib/prometheus

🔍 **Why?**

Ensures that only the prometheus user can read/write its own files

Improves security and avoids permission errors at runtime

### ✅ ****Step 7: Create a Systemd Service for Prometheus****

sudo nano /etc/systemd/system/prometheus.service

Paste this:

[Unit]

Description=Prometheus Monitoring

Wants=network-online.target

After=network-online.target

[Service]

User=prometheus

Group=prometheus

Type=simple

ExecStart=/usr/local/bin/prometheus \

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

--storage.tsdb.path=/var/lib/prometheus/ \

--web.console.templates=/etc/prometheus/consoles \

--web.console.libraries=/etc/prometheus/console\_libraries

[Install]

WantedBy=multi-user.target

🔍 **Why?**

This defines how Prometheus starts, stops, and runs as a background service

Runs as prometheus user with correct paths

Allows you to manage it using systemctl (start, stop, enable)

### ✅ ****Step 8: Reload Systemd and Start Prometheus****

sudo systemctl daemon-reexec

sudo systemctl daemon-reload

sudo systemctl enable prometheus

sudo systemctl start prometheus

daemon-reexec and daemon-reload reload systemd so it recognizes your new service

enable starts it on every reboot

start launches Prometheus immediately

### ✅ ****Step 9: Check Prometheus Status and Web UI****

Check the service status:

sudo systemctl status prometheus

Then open the web UI in a browser:

[http://192.168.50.10:9090](http://<your-prometheus-ip>:9090)

✅ **Step 10: (Optional) Open Firewall Port 9090**

If using UFW or nftables:

sudo ufw allow 9090/tcp

Prometheus listens on port 9090 by default

Installation is done now adding nodes to the prometehus:

For it we have to install node exporter on machines which prometehus is going to monitor

1. Email server :

We will add node exporter and postfix exporter: node exporter will export metrics of system like cpu usage and all ,and postfix exporter will export metrics like postfix\_queue \_length that is total number of mails in queue,postfix\_message\_sent\_total that is number of successfully sent messages.postfix\_message\_delivery\_seconds that is time taken to deliver email. It collects these metrics by **reading Postfix logs** (/var/log/mail.log, /var/log/maillog) and **queue directories**.

**{1}node exporter for mail server:**

# Create user

sudo useradd --no-create-home --shell /bin/false node\_exporter

# Download latest

cd /tmp

wget https://github.com/prometheus/node\_exporter/releases/latest/download/node\_exporter-1.9.1.linux-amd64.tar.gz

tar -xvzf node\_exporter-1.9.1.linux-amd64.tar.gz

cd node\_exporter-1.9.1.linux-amd64

# Move binary

sudo cp node\_exporter /usr/local/bin/

sudo chown node\_exporter:node\_exporter /usr/local/bin/node\_exporter

# Creating systemd service:

sudo nano /etc/systemd/system/node\_exporter.service

Paste following line-

[Unit]

Description=Node Exporter

After=network.target

[Service]

User=node\_exporter

Group=node\_exporter

Type=simple

ExecStart=/usr/local/bin/node\_exporter

[Install]

WantedBy=default.target

# Enable and start service

sudo systemctl daemon-reexec

sudo systemctl daemon-reload

sudo systemctl enable node\_exporter

sudo systemctl start node\_exporter

sudo systemctl status node\_exporter

# Test

<http://192.168.40.10:9100/metrics>

**{2} Install postfix\_exporter on mail server**

#Install go (#needed for )

sudo apt install golang -y

# Building exporter: getting binary

cd /opt

sudo git clone https://github.com/kumina/postfix\_exporter.git

cd postfix\_exporter

sudo go build

# Creating service

sudo nano /etc/systemd/system/postfix\_exporter.service

Paste following lines:

[Unit]

Description=Postfix Exporter

After=network.target

[Service]

ExecStart=/opt/postfix\_exporter/postfix\_exporter

User=prometheus

Restart=always

[Install]

WantedBy=multi-user.target

# Starting and enabling service

sudo systemctl daemon-reexec

sudo systemctl daemon-reload

sudo systemctl enable postfix\_exporter

sudo systemctl start postfix\_exporter

# Accessing

<http://192.168.40.10:9154/metrics>

#Now add this machine to prometehus - have to edit prometheus.yml file

sudo nano /etc/prometheus/prometheus.yml

- job\_name: 'mail\_node'

static\_configs:

- targets: ['192.168.40.10:9100']

- job\_name: 'postfix\_node'

static\_configs:

- targets: ['192.168.40.10:9154']

1. Installing node exporter on wazuh server

Same steps that we have done for mail server node\_exporter

Add following lines to /etc/prometheus/prometheus.yml

- job\_name: 'wazuh\_node'

static\_configs:

- targets: ['192.168.50.20:9100']

1. **Configuring node\_exporter on k8s cluster**

We will need to configure exporters and metric endpoints on every node of cluster

| **Node** | **Role** | **Metrics to Collect** | **How to Export** |
| --- | --- | --- | --- |
| 192.168.40.20 | Worker Node 1 (App Pods) | Node system metrics + Pod metrics | node\_exporter + kubelet |
| 192.168.50.30 | Worker Node 2 (DB Pods) | Node system metrics + Pod metrics | node\_exporter + kubelet + DB exporter |
| 192.168.50.50 | Master Node | Control plane + Node metrics | node\_exporter + kube-apiserver + etcd |

### ✅ Phase 1: Enable System Metrics on All K8s Nodes

Same steps as given above.

### ✅ Phase 2: Expose Kubelet and Pod Metrics

Prometheus needs to scrape metrics **from each node and kubelet** in the cluster, because:

Pods don’t expose metrics by default.

The **Kubelet** on each node provides system/pod/container stats.

#### Step 1: Enable kubelet read-only metrics (if required)

Most kubelet endpoints are exposed at:

<https://<node-ip>:10250/metrics>

Here by default when kublet expose metrics it uses tls(https),so to scrape these we need create service account with RABC or we can expose metrics via prometehus kubelet service monitor if we are using prometheus operator.

#### Step 2: Edit Prometheus config to scrape kubelet and pods

This allows Prometheus to scrape node and pod-level stats from the kubelet running on each node.

Open /etc/prometheus/prometheus.yml on your Prometheus server.

Paste following lines in it:

- job\_name: 'kubernetes\_nodes'

scheme: https

tls\_config:

insecure\_skip\_verify: true # Skip TLS verification (lab only)

metrics\_path: /metrics

static\_configs:

- targets:

- 192.168.40.20:10250 # App node kubelet

- 192.168.50.30:10250 # DB node kubelet

- 192.168.50.50:10250 # Master node kubelet

sudo systemctl restart prometheus

http://<prometheus-server-ip>:9090/targets

### ✅ ****Phase 3: Monitor Kubernetes-specific Metrics (Pods, cAdvisor, kubelet, etc.)****

**Enabling Kubernetes API access** so Prometheus can discover components.

**Configuring Prometheus** to scrape:

**kubelet metrics**

**cAdvisor (container stats)**

**Kubernetes API server metrics**

(Optional) **kube-state-metrics** for high-level Kubernetes object status

| **Component** | **Why We Monitor It** |
| --- | --- |
| kubelet | Gives node and pod status, runtime metrics, container usage |
| cAdvisor | Gives detailed container-level resource usage (CPU, memory, etc.) |
| API server | Helps you monitor health, latency, and usage of K8s API |
| kube-state-metrics | High-level state of deployments, pods, nodes, etc. (useful for alerting/logics) |

### ****Step 1:Ensure kubelet metrics and cAdvisor are exposed- run following command on all nodes of k8s****

ps aux | grep kubelet

Shows output like this:-

--read-only-port=10255

--authentication-token-webhook

--authorization-mode=Webhook

--cgroup-driver=systemd

--container-runtime-endpoint=unix:///var/run/containerd/containerd.sock

--enable-server

--address=0.0.0.0

Step 2: From your **Prometheus machine**, try accessing the following endpoints:

curl http://192.168.40.20:10255/metrics # worker1 (application pods)

curl http://192.168.50.30:10255/metrics # worker2 (database pods)

curl http://192.168.50.50:10255/metrics # master node

### ****🔹 Step 3: Edit**** prometheus.yml ****to scrape these metrics****

sudo nano /etc/prometheus/prometheus.yml

Paste these line:

- job\_name: 'kubernetes\_nodes'

static\_configs:

- targets:

- '192.168.40.20:10255' # worker1

- '192.168.50.30:10255' # worker2

- '192.168.50.50:10255' # master node

It gives Pod/container resource usage via cAdvisor (built into kubelet)

sudo systemctl restart prometheus

## ✅ Phase 4: Setting Up kube-state-metrics

Pod status (Pending, Running, Failed)

Deployment & ReplicaSet health

StatefulSet & DaemonSet metrics

Node readiness

Resource requests vs limits

### ****Why we need this:****

Prometheus + node\_exporter + kubelet metrics give you system-level and container-level metrics.

But they **do not give object-level** info like:

“How many pods are in CrashLoop?”

“Is the replica count of this deployment satisfied?”

“How many nodes are NotReady?

This is where kube-state-metrics comes in. It exposes these as Prometheus metrics.

We will **deploy** kube-state-metrics **as a pod** in your cluster.

Before deploying Make sure: you have kubectl access to your cluster. And Your cluster is running fine (DNS and RBAC working).

## 💡 What does “official manifests from Kubernetes Sigs GitHub repo” mean?

### ✅ ****Manifest = YAML configuration file****

In Kubernetes, a **manifest** is just a YAML file that **describes how to deploy something**. For example:

A Deployment manifest defines how many pods to run, which image to use.

A Service manifest exposes those pods over the network.

### ✅ ****Official = Maintained by trusted Kubernetes SIG (Special Interest Group)****

The kube-state-metrics tool is developed by the **Kubernetes SIG instrumentation team**.

These YAMLs (manifests) are hosted on the **official GitHub repository** maintained by them:  
👉 <https://github.com/kubernetes/kube-state-metrics>

### ✅ ****Why use them?****

### Secure, stable, production-ready

You don’t have to write all YAMLs yourself

Includes best practices (resource limits, RBAC, namespaces, ports, labels)

### When we run this:

kubectl apply -f https://github.com/kubernetes/kube-state-metrics/releases/latest/download/kube-state-metrics.yaml

We are telling Kubernetes:

“Download the **official YAML file** that defines all the needed resources for kube-state-metrics, and deploy it.”

This YAML will create:

A Deployment

A Service

ClusterRole, RoleBinding (RBAC)

Namespace (if need)

**Manifest** = config file that tells Kubernetes what to deploy and how.

**Official** = written and maintained by the trusted developers of the tool.

**We use it** to avoid errors, save time, and follow best practices

Step 1:kubectl apply -f https://github.com/kubernetes/kube-state-metrics/releases/latest/download/kube-state-metrics.yaml

Step 2:checking pod of kube state metric is running or not.

kubectl get pods -n kube-system | grep kube-state-metrics

#It will expose the pod internally in the cluster, typically on:

kube-state-metrics.kube-system.svc.cluster.local:8080

Step 3: our prometheus is on external so we have to expose these metrics outside/externally so that it can scrape.that is giving access of that metrics to prometheus.We’ll expose the kube-state-metrics service using a NodePort:

kubectl edit svc kube-state-metrics -n kube-system

Make changes in yaml file:

from=

type: ClusterIP

To =

type: NodePort

Save exit and then run next command

kubectl get svc kube-state-metrics -n kube-system

✅ Note the NodePort number it’s assigned to (e.g., 31892)

http://<worker\_node\_ip>:<node\_port>

### Step 4: Add it to Prometheus

Edit /etc/prometheus/prometheus.yml

- job\_name: 'kube\_state\_metrics'

static\_configs:

- targets: ['192.168.50.50:31892'] # adjust IP and NodePort as per your output

sudo systemctl restart prometheus

Note: kube-state-metrics is a **cluster-level** monitoring component, not a node-level one.so we can run it on any one pod and it will expose metrics of all pods on all nodes.

---------------------------------------------------------------------

#####mysqld\_exporter on **Worker Node 2** (your DB server – 192.168.50.30)##

mysqld\_exporter exposes internal MySQL/MariaDB server metrics (connections, queries per second, slow queries, cache hit ratio, etc.) in Prometheus format.

Prerequisites:

MySQL/MariaDB should already be installed and running.

You need a dedicated MySQL user **with limited privileges** for Prometheus.

### ****Step 1: Create Prometheus MySQL User****

🔹 **Why:** Prometheus needs read-only access to MySQL internals.

CREATE USER 'mysqld\_exporter'@'localhost' IDENTIFIED BY 'securepassword';

GRANT PROCESS, REPLICATION CLIENT, SELECT ON \*.\* TO 'mysqld\_exporter'@'localhost';

FLUSH PRIVILEGES;

### ****Step 2: Download and Install**** mysqld\_exporter

cd /opt/

wget https://github.com/prometheus/mysqld\_exporter/releases/latest/download/mysqld\_exporter-0.15.1.linux-amd64.tar.gz

tar -xzf mysqld\_exporter-0.15.1.linux-amd64.tar.gz

sudo mv mysqld\_exporter-\*/mysqld\_exporter /usr/local/bin/

### ****Step 3: Set MySQL Credentials for Exporter****

Prometheus mysqld\_exporter uses a .my.cnf file to read DB credentials.

Create it in /etc/.mysqld\_exporter.cnf:

sudo nano /etc/.mysqld\_exporter.cnf

Paste following lines:

[client]

user=mysqld\_exporter

password=securepassword

sudo chmod 600 /etc/.mysqld\_exporter.cnf

🔹 **Why:** This is a secure way to store DB credentials for exporters.

### ****Step 4: Create a Systemd Service.****

sudo nano /etc/systemd/system/mysqld\_exporter.service

[Unit]

Description=MySQL Exporter

After=network.target

[Service]

User=root

Environment="DATA\_SOURCE\_NAME=mysqld\_exporter:securepassword@(localhost:3306)/"

ExecStart=/usr/local/bin/mysqld\_exporter \

--config.my-cnf=/etc/.mysqld\_exporter.cnf \

--web.listen-address=":9104"

[Install]

WantedBy=multi-user.target

sudo systemctl daemon-reload

sudo systemctl enable mysqld\_exporter

sudo systemctl start mysqld\_exporter

sudo systemctl status mysqld\_exporter

curl http://localhost:9104/metrics

Sample output:

# HELP mysql\_global\_status\_threads\_connected Number of currently open connections.

# TYPE mysql\_global\_status\_threads\_connected gauge

mysql\_global\_status\_threads\_connected 2

Step 7: Add to Prometheus Targets

Go to your **Prometheus server**, open prometheus.yml, and add:

- job\_name: 'mysqld\_exporter'

static\_configs:

- targets: ['192.168.50.30:9104']

sudo systemctl restart prometheus

Perfect — now let’s configure kube-apiserver and etcd monitoring on your **Kubernetes master node (**192.168.50.50**)**.

## ✅ Phase: Monitor kube-apiserver and etcd on Master Node

These are **critical control plane components**, and monitoring them helps you ensure your Kubernetes cluster is healthy.

### PART 1: Monitor kube-apiserver Metrics

#### Why?

The API Server is the heart of Kubernetes. It handles:

Cluster state reads/writes

Auth, RBAC, controller communication

Prometheus can scrape metrics like:

Request rate

Latency

Error rate

Auth failure

### Step 1: Ensure API Server Metrics Are Exposed

By default, kube-apiserver exposes metrics at:

https://<master-node-ip>:6443/metrics

But Prometheus can't scrape HTTPS endpoints with client auth unless configured.

➡️ **Option 1 (Recommended for simplicity):** Use the local loopback (from the same master node), and run a small exporter or use a scrape config that disables TLS verification (only in internal network).

### Step 2: Add Job to Prometheus (prometheus.yml)

From your Prometheus server:

- job\_name: 'kube-apiserver'

scheme: https

tls\_config:

insecure\_skip\_verify: true

metrics\_path: /metrics

static\_configs:

- targets: ['192.168.50.50:6443']

🔐 insecure\_skip\_verify: true is used only if you don’t want to manually set up TLS CA cert verification (OK in isolated lab setups like yours).

Then reload Prometheus:

curl -X POST http://localhost:9090/-/reload

Step 3: Check in Browser

Visit your Prometheus UI → **Status > Targets**, and you should see kube-apiserver as UP.

PART 2: Monitor etcd (Kubernetes Backend Store)

#### Why?

etcd stores **all Kubernetes state**. You want to monitor:

Disk usage

Latency

DB size

Raft performance

### 🔹 Step 1: Check if etcd is Running and Metrics Are Available

Run on the **master node**:

curl -k https://127.0.0.1:2379/metrics

✅ You should see metrics like etcd\_server\_has\_leader, etcd\_disk\_wal\_fsync\_duration\_seconds, etc.

### 🔹 Step 2: Add Prometheus Job for etcd

On Prometheus server:

- job\_name: 'etcd'

scheme: https

metrics\_path: /metrics

tls\_config:

insecure\_skip\_verify: true

static\_configs:

- targets: ['192.168.50.50:2379']

Then reload config again:

curl -X POST http://localhost:9090/-/reload

Now, Prometheus will monitor:

| **Component** | **Port** | **Metrics Included** |
| --- | --- | --- |
| kube-apiserver | :6443 | API request rates, auth stats, latency, error count |
| etcd | :2379 | DB size, latency, Raft status, storage health |

**Setting grafana for prometheus:**

|  |
| --- |
| Support for **community plugins** for data sources and panels |

|  |
| --- |
|  |
| | **Tool** | **Description** | **Key Differences from Grafana** | | --- | --- | --- | | **Kibana** | Part of the **ELK stack** (Elasticsearch, Logstash, Kibana); used for logs | Best for **log analytics**, tightly integrated with **Elasticsearch** | | **Chronograf** | UI for the **TICK stack** (Telegraf, InfluxDB, Chronograf, Kapacitor) | Optimized for **InfluxDB**, simpler than Grafana | | **Tableau** | Enterprise-grade data visualization software | Proprietary, very powerful but **not free** and not focused on monitoring | | **Power BI** | Microsoft’s BI tool | More for **business intelligence**, not infrastructure monitoring | | **Zabbix** | Includes monitoring + alerting + visualization | All-in-one tool, but **not as flexible** in visualization as Grafana | | **Datadog** | SaaS observability platform | Powerful but **paid**, provides more built-in metrics than Grafana | |

**Step 1: Update Your System**

sudo apt update && sudo apt upgrade -y

**Step 2: Install Grafana**

### 2.1 Add Grafana APT repository:

sudo apt install -y software-properties-common

sudo apt install -y apt-transport-https

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://packages.grafana.com/gpg.key | sudo gpg --dearmor -o /etc/apt/keyrings/grafana.gpg

### 2.2 Add repository to sources list:

echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://packages.grafana.com/oss/deb stable main" | \

sudo tee /etc/apt/sources.list.d/grafana.list

### 2.3 Update and install Grafana:

sudo apt update

sudo apt install grafana -y

**Step 3: Start and Enable Grafana Server**

sudo systemctl daemon-reexec

sudo systemctl start grafana-server

sudo systemctl enable grafana-server

Check status:

sudo systemctl status grafana-server

**Step 4: Access Grafana Web Interface**

Open browser

Go to: http://<grafana\_server\_ip>:3000

Default Login:

**Username**: admin

**Password**: admin (you'll be prompted to change it)

**Step 5: Add Prometheus as a Data Source**

After login, click on ⚙️ **(gear icon) → Data Sources**

Click **Add data source**

Choose **Prometheus**

In **URL** field, enter:

http://<prometheus\_server\_ip>:9090

Example: http://192.168.50.10:9090

Click **Save & Test**

You should see: ✅ Data source is working

**Step 6: Import Pre-built Dashboards**

Click on the **+ (plus icon) → Import**

You can import a dashboard using:

**Dashboard ID** from Grafana website (like 1860 for node\_exporter dashboard)

Or upload JSON files

Select Prometheus as the data source

Click **Import**

**Likewise import dashboards for mysql and kubernetes cluster monitoring.**

**Dashboard codes=mysql 7362**

**Kubernetes kube state metrics 13332**

**K8s cluster monitoring 315**