**Prometheus and features set-up for continuous monitoring**

To monitor the target machine before the service migration, a machine learning (ML) model was trained to return the result whether we should start the migration process based on the collected metrics. To retrieve these metrics, the *Prometheus* tool is used together with the *Prometheus Node Exporter* and Metrics Visualisation tool *Grafana*. However, these tools are kept running continuously even while the service is being migrated. The aim of the continuous monitoring is to stop the migration process if the condition becomes bad, for example if the connection is down or the target machine's file system is not working properly.

This chapter gives a brief introduction to the tools used, followed by the configuration of the entire monitoring system.

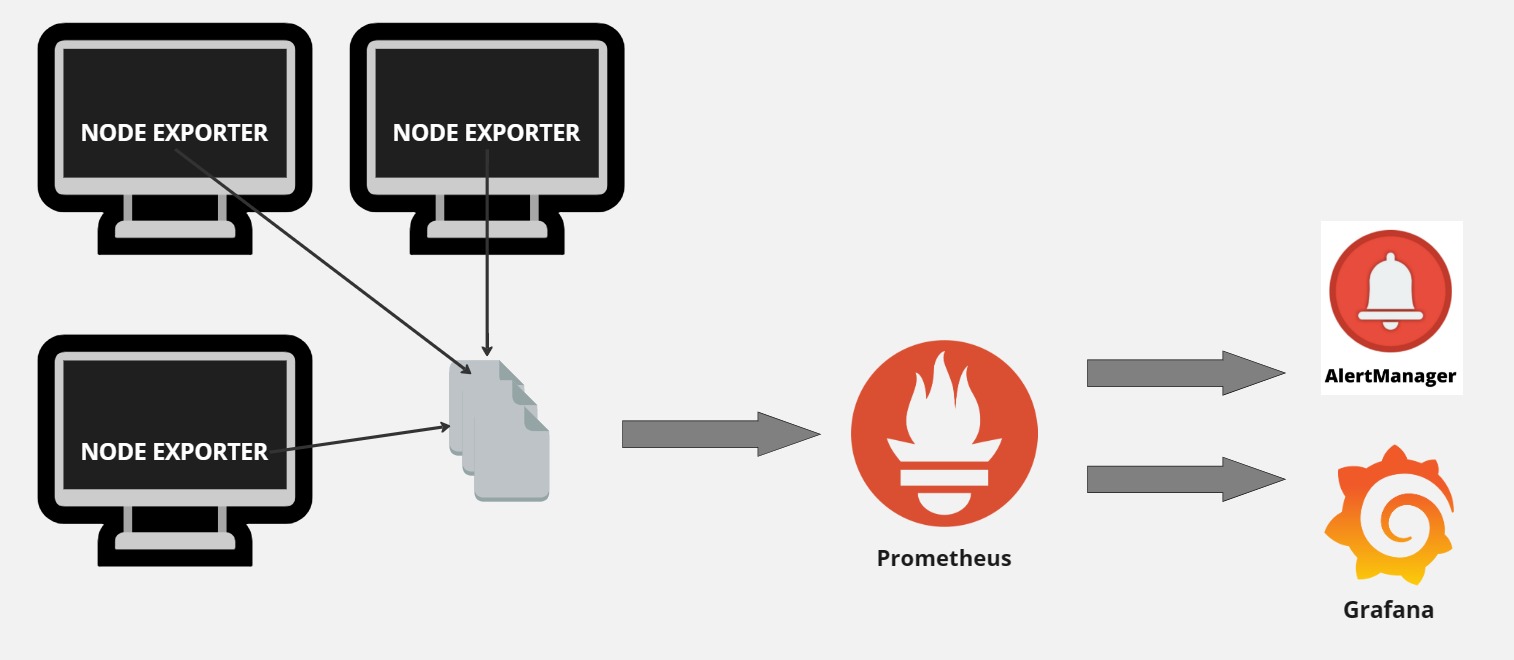


Figure 1 Prometheus monitoring system

# **Monitoring system**

Figure 1 illustrates the structure of our monitoring system. First, the *Prometheus Node Exporter* is run on the host to retrieve hardware-based metrics. The metrics are then collected by *Prometheus*. As well as using Prometheus to monitor our nodes, there are a number of features that Prometheus also provides to facilitate the management work. In our case, we use *Alert Manager* and *Grafana*. In the later chapters, each tool involved in the monitoring system will be described in more detail.

## **Prometheus**

There are quite a lot of options for monitoring tools currently on the market, such as Sematext, SolarWinds Server, Dynatrace, Datadog, Nagios and so on (Src: [10 Best Server Monitoring Tools & Software [2023 Review] - Sematext](https://sematext.com/blog/server-monitoring-tools/)). Among other tools, we decided to use Prometheus. The reasons for setting up a monitoring system with Prometheus are that it is an open source monitoring tool. Together with Grafana, they are the well-known combination used for server tracking (Src: [10 Best Server Monitoring Tools & Software [2023 Review] - Sematext](https://sematext.com/blog/server-monitoring-tools/)).

Prometheus monitoring tool is commonly used due to its following key features (Src: [Prometheus Monitoring: The Pull Approach (network-insight.net)](https://network-insight.net/2022/06/29/prometheus-monitoring-the-pull-approach/#:~:text=Prometheus%20operates%20on%20a%20pull,the%20data%20at%20regular%20intervals.), [A Comprehensive Guide to Prometheus Monitoring (weave.works)](https://www.weave.works/blog/a-comprehensive-guide-to-prometheus-monitoring#:~:text=Prometheus%20Monitoring%20is%20a%20powerful,your%20systems%20for%20maximum%20efficiency.)):

* **Time-series database:** The metrics are stored in a Prometheus-customised time-series database, making it convenient to query and analyse the data. The stored data is timestamped, so it is possible to graph its change to get a visual overview of how the data changes over time.
* **PromQL query language:** Prometheus uses PromQL, which provides various operations to extract meaningful information from the collected metrics.
* **Pull-based system:** Prometheus actively pulls metrics from the node it is monitoring. The fetching process is done regularly after a time interval and this brings the advantage of timely handling of any negative changes.
* **Supported components:** Various features such as Grafana and Alert Manager are available to help monitor the node. They are also simple to set up together.
* **Alerting:** The user can define rules for the metrics, for example, to control when the CPU utilisation is higher than 95%. Alert rule configuration is described in a later chapter.

Prometheus does not monitor the server itself, but needs the help of another module to collect the metrics, and this is where Node Exporter comes in.

### **1.1.1. Prometheus Node Exporter**

Our monitoring system is designed for Linux operating system hosts, so Node Exporter is used to collect system information of the machine it is running on (Src: [Node exporter | GitLab](https://docs.gitlab.com/ee/administration/monitoring/prometheus/node_exporter.html)). For the Windows operating system, the WMI Exporter is used instead. The metrics are in the format that is utilized by the Prometheus (Src: [Guide To The Prometheus Node Exporter : OpsRamp](https://www.opsramp.com/guides/prometheus-monitoring/prometheus-node-exporter/#:~:text=A%20Node%20Exporter%20is%20needed,sub%2Dpath%20on%20port%209100.)).

With Node Exporter as the low-level system information collector, Prometheus' role is to store and display the metrics. This separation allows scalability. One Prometheus can collect and analyse metrics from multiple nodes (Src: [Prometheus Monitoring: The Pull Approach (network-insight.net)](https://network-insight.net/2022/06/29/prometheus-monitoring-the-pull-approach/#:~:text=Prometheus%20operates%20on%20a%20pull,the%20data%20at%20regular%20intervals.)).

### **1.1.2. Prometheus and Node Exporter setup**

*Node Exporter* does not require any special configuration. For the *Linux* operating system, all we need to do is download the *Node Exporter* zip file to the machine we want to monitor, unpack it, run it and we are ready to go. By default, *Node Exporter* runs and updates the host machine's metrics continuously (Src: [Prometheus: Using Node Exporter (stackhero.io)](https://www.stackhero.io/en/services/Prometheus/documentations/Using-Node-Exporter)).

As I mentioned earlier, *Prometheus* will actively pull the metrics from the *Node Exporter* after a time interval. And *Prometheus* can capture multiple nodes at the same time. To make all this work, we need to register these nodes in Prometheus ([Prometheus: Using Node Exporter (stackhero.io)](https://www.stackhero.io/en/services/Prometheus/documentations/Using-Node-Exporter)).

*# Global configuration*global:  
 […] *# Alertmanager configuration*alerting:  
 […]

rule\_files:  
 […]  
  
*# A scrape configuration containing exactly one endpoint to scrape:*scrape\_configs:

*# The Prometheus itself.* - job\_name: "prometheus"  
 scrape\_interval: 5sstatic\_configs:  
 - targets: ["localhost:9090"]

*# Prometheus Node Exporter*  
 - job\_name: "node"  
 scrape\_interval: 5s  
 static\_configs:  
 - targets: ["localhost:9100"]  
  
 […]

In the scrape\_configs section above, there is one job that captures the Prometheus itself and the other is the Node Exporter. The default port of Prometheus is 9090 and the Node Exporter exposes metrics on port 9100. Both jobs update every five seconds.

## **1.2. Grafana**

Up to now, we have been able to retrieve, query and analyse the metrics. The data is presented as plain text and in some cases this way of presenting data is difficult to perceive or takes longer to perceive. It is quite common that the *Grafana* tool is used together with Prometheus, because with *Grafana* we can display the metrics in a chart. In our use case, we use Grafana mainly to visualise the metrics and to extract the metrics that will later be used to train the ML model. In addition, Grafana also provides various functionalities, such as data query, alerts, metrics exploration, logs, and so on. Grafana treats Prometheus as a data source, as any other source can be named here, such as MySQL, Graphite, InfluxDB, etc (Src: [What Is Grafana? Why Use It? Everything You Should Know About It - Scaleyourapp](https://scaleyourapp.com/what-is-grafana-why-use-it-everything-you-should-know-about-it/)).

Grafana gives us the freedom to design the dashboard. A dashboard can contain histograms, different types of maps (heat maps, geomaps) and charts. In addition, there are also pre-built dashboards that are available so that we can include them in our use case and not have to build ours from scratch (Src: [What Is Grafana? Why Use It? Everything You Should Know About It - Scaleyourapp](https://scaleyourapp.com/what-is-grafana-why-use-it-everything-you-should-know-about-it/)).

## **1.3. Prometheus Alert Manager**

After all of the steps above, we continue extend our monitoring system with an alert manager system. An alert mechanism has been configured to send an email if the parameter exceeds the limit. We use Prometheus Alert Manager for alerting but Grafana provides also the same mechanism and can be used instead.

Prometheus provides alert feature, which contains a list of rules, when the metrics exceed the limit value, the rule is triggered. The status of rules can be controlled using Prometheus UI. The usage of Alert Manager gives us more flexibility since it provides integration to various communication platforms: Slack, Microsoft Teams, Google Mail, OpsGenie, PageDuty (Src: [Alertmanager | Prometheus](https://prometheus.io/docs/alerting/latest/alertmanager/), [The Guide To Prometheus Alerting : OpsRamp](https://www.opsramp.com/guides/prometheus-monitoring/prometheus-alerting/#:~:text=Notification%20Channels,with%20any%20event%20management%20system.)).

### **1.3.1. Alert Rules Configuration**

We have configured a list of rules to monitor the target machine. There are some parameters that are critical to control and handle in time when monitoring a server in general. These are CPU, disk and memory usage. For each of these parameters, I defined two rules with different alert flags, one with a warning flag and the other with a critical flag. The list of rules has been created with reference to the source “Awesome Prometheus Alerts” (Src: [Awesome Prometheus alerts | Collection of alerting rules (samber.github.io)](https://samber.github.io/awesome-prometheus-alerts/)). The limit for these parameters is set as follow:

* Disk usage:
  + Warning: If the disk is possessed more than 80%
  + Critical: If CPU is possessed more than 90%
* CPU usage:
  + Warning: If the CPU is utilized more than 75%
  + Critical: If the CPU is utilized more than 80%
* Memory usage:
  + Warning: If the memory is utilized more than 75%
  + Critical: If the CPU is utilized more than 80%

In addition to the above parameters, based on the purpose of monitoring the health of the target host, our list of rules also includes setting limits to check the state of the network, read and copy speed of data, and so on. These rules are as follows:

* Server down:
  + Critical: If server is down for more than one minute.
* Host File System Error:
  + Warning: If file system does not work.
* Disk Read Latency:
  + Warning: If read operations take more than 100 milliseconds.
* Disk Write Latency:
  + Warning: If the write operations take more than 100 milliseconds.
* Network Transmit Errors:
  + Warning: Report the number of transmission errors in the last two minutes.
* Network Receive Errors:
  + Warning: Report the number of reception errors in the last two minutes.
* Prometheus Alert Manager Missing:
  + Warning: If the *Alert Manager* job is inactive.
* Prometheus Alert Manager Notification Failing:
  + Warning: If the *Alert Manager* has failed to send the alert email.

The code snippet below is a disk usage rule configuration, it gives an example of what a rule configuration might look like.

- name: DiskUsage90  
 rules:  
 - alert: 'Low Data Disk Space'  
 expr: ceil(((node\_filesystem\_size\_bytes{mountpoint!="/boot"} - node\_filesystem\_free\_bytes{mountpoint!="/boot"}) / node\_filesystem\_size\_bytes{mountpoint!="/boot"} \* 100)) > 90  
 labels:  
 severity: critical  
 annotations:  
 title: "Disk Usage"  
 description: 'Partition: {{$labels.mountpoint}}'  
 summary: "Disk usage is `{{humanize $value}}%`"  
 host: "{{$labels.instance}}"

### **1.3.2. Alert Manager Configuration**

For the alert rules we have two different alert flags: warning and critical. Having two different alert levels is not only useful when we control the server using the Prometheus UI, but also when we set up the *Alert Manager*.

route:  
 group\_by: ['alertname']  
 group\_wait: 30s  
 group\_interval: 10s  
 repeat\_interval: 1h  
 receiver: 'email-notifications'  
  
receivers:  
- name: 'email-notifications'  
 email\_configs:  
 - to: 'minhtrang190199@gmail.com'  
 from: 'alertmanagertestprometheus@gmail.com'  
 smarthost: 'smtp.gmail.com:587'  
 auth\_username: 'alertmanagertestprometheus@gmail.com'  
 auth\_password: ‘<alert manager pwd>’  
 send\_resolved: true  
 require\_tls: true  
 match:  
 severity: critical

The code snippet above is the *Alert Manager* configuration that allows an email to be sent when a rule with a critical flag is triggered. The alert email is resent after one hour.