Bachelor’s Thesis

Information and Communications Technology

2021

Tiia Leppänen

Data visualization and monitoring with Grafana and Prometheus

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor’s Thesis | Abstract

Information and Communications Technology

2021 | 49 pages

Tiia Leppänen

Data visualization and monitoring with Grafana and Prometheus

Visual monitoring raises awareness of changes in the IT environment. In visual monitoring, the monitoring of targets must have a clear meaning and the visualization must be easy to understand.

The purpose of this thesis was to implement a visual monitoring system in the client's network environment.

Two monitoring solutions, Zabbix and LibreNMS, were already in use in the client's network environment. A solution had to be found for presenting the monitored data, for which the Grafana visualization system was selected. Prometheus was chosen as Grafana's counter part for the monitoring tool.

In order to gain the best possible insight into the client's needs, differences of the monitoring systems in the data collection and the differences in the data visualization of the different visualization methods were compared. The strengths and advantages of these two systems (Prometheus and Grafana) were examined and compared to Zabbix and LibreNMS’ features. The intention was not to replace the old monitoring systems, but to utilize the advantages of the new systems in Tietopartio's network environment.

A Prometheus server was installed on a Unix-based virtual server to collect data from that virtual server and from a switch located in Tietopartio’s data center. The Grafana visualization platform was also installed on the same virtual server. Grafana's visualization capabilities were tested by importing a ready-made dashboard for monitoring Unix operating systems. A completely new dashboard was created to monitor the switch.

The final result was a functional example of Prometheus and Grafana’s working in different environments. Issues to be taken into account in the further development of the monitoring system include implementation of an encryption method and alarms.

Keywords:

Grafana, Prometheus, monitoring, visualization

Opinnäytetyö (AMK) | tiivistelmä

Tieto- ja viestintätekniikka

2021 | 49 sivua

Tiia Leppänen

Datan visualisointi ja monitorointi Grafanan ja Prometheuksen avulla

Visuaalinen monitorointi lisää tietoisuutta IT-ympäristössä tapahtuvista muutoksista. Visuaalisessa monitoroinnissa kohteiden monitoroinnilla tulee olla selkeä merkitys sekä visualisoinnin pitää olla helposti ymmärrettävä.

Tämän opinnäytetyön tavoitteena oli visuaalisen monitoroinnin toteuttaminen toimeksiantajan laiteympäristöön.

Toimeksiantajan laiteympäristössä oli jo käytössä kaksi monitorointiratkaisua, Zabbix ja LibreNMS. Monitoroidun datan esittämiseen oli löydettävä ratkaisu, johon valikoitui visualisointijärjestelmä Grafana. Grafanan pariksi monitorointityökaluksi valikoitui Prometheus.

Jotta saatiin mahdollisimman hyvä kuva toimeksiantajan tarpeista, vertailtiin monitorointijärjestelmien välisiä eroavaisuuksia datan keruussa sekä eri visualisointialustojen datan visualisoinnin eroavaisuuksia. Lisäksi tarkasteltiin, mitä vahvuuksia näillä kahdella järjestelmällä (Prometheus ja Grafana) on Zabbixiin ja LibreNMS:ään verrattuna. Tarkoituksena ei ollut syrjäyttää vanhoja monitorointijärjestelmiä, vaan hyödyntää uusien järjestelmien hyviä puolia toimeksiantajan laiteympäristössä.

Datan kerryttämistä varten Unix-pohjaiselle virtuaalipalvelimelle asennettiin Prometheus-palvelin keräämään dataa kyseisestä virtuaalipalvelimesta sekä konesalissa sijaitsevasta kytkimestä. Samalle virtuaalipalvelimelle asennettiin myös visualisointialusta Grafana. Grafanan visualisointiominaisuuksia testattiin tuomalla valmis, Unix-käyttöjärjestelmien monitorointia varten tehty kojelauta. Kytkimen monitorointia varten luotiin täysin uusi kojelauta.

Loppuratkaisuna saatiin toimiva esimerkki Prometheuksen ja Grafanan toiminnasta eri ympäristöissä. Järjestelmän jatkokehityksessä huomioon otettavia asioita ovat mm. salausjärjestelmän ja hälytysten käyttöönotto.

Asiasanat:

Prometheus, Grafana, monitorointi, visualisointi

Content

List of abbreviations

ARP	Address Resolution Protocol
CPU	Central Processing Unit
DEB	the format, as well as extension of the software package format for the Debian Linux distribution and its derivatives
Debian	Also known as Debian GNU/Linux, is a GNU/Linux distribution composed of free and open-source software (Wikipedia, Debian)
HP	Hewlett Packard
HTTP	Hypertext Transfer Protocol
IT	Information Technology
iTerm2	A terminal emulator for macOS
Linux	A family of open-source Unix-like operating systems based on the Linux kernel
MariaDB	Open source relational database
Metrics	A set of numbers that give information about a particular process or activity (Cambridge dictionary)
MIB	Management information base
MySQL	Open-source relational database management system (SQL = Structured Query Language)
OID	Object Identifier
OSPF	Open Shortest Path First
PHP	General-purpose scripting language

PNG	Portable Network Graphics
PromQL	Prometheus query language
RRD	Round-robin database
SNMP	Simple Network Management Protocol
SQLite	Relational database management system
SSH	Secure Shell
STP	Spanning Tree Protocol
Time series	sequence taken at successive equally spaced points in time (Wikipedia, Time series)
TSDB	Timeseries Database
UI	User Interface
Unix	A family of multitasking, multiuser computer operating systems (Wikipedia, Unix)
URL	Uniform Resource Locator
YAML	Human-readable data-serialization language

1 Introduction

A functional and well-maintained monitoring system has become a crucial part of almost any company and business in this era of Internet and IoT where computers are basic work environment tools, and any device is and can be connected to the Internet. It is important to have knowledge of the performance, network traffic and storage capacity of companies' infrastructure. Monitored targets can vary from physical devices to services and applications and to virtual machines. Visual representation and alerting are key elements of a proper monitoring system. Alerts can be implemented for different occasions. For example, the monitoring system will send an email notification whenever a certain service is down. In this way the IT personnel can act as fast as possible to fix the problem. By visualizing the devices and services' data, a good overall view of the environment can be given. Displayed information can be great for preventive monitoring by showing past incidents and trends. Therefore, IT specialist can make predictions for future events or fix what could become a problem later on and send out an alert. There are numerous monitoring solutions available for download free as open-source or paid licenses.

This thesis was implemented as a commission for Tietopartio Oy, an IT solutions provider based in Turku. Tietopartio was looking for a data visualizing platform to display valuable data from their data center. A more flexible monitoring tool was also needed to accompany this visualization system. The employer had their mind set on Grafana as the visualization software and Prometheus as the monitoring tool. The provider already had two monitoring systems, Zabbix and LibreNMS, monitoring their own network environment and their customers' backups. These systems were mainly used for alerting and not for graphical representation of the monitored targets.

The theoretical part of this thesis consists of research of the differences, and advantages of the new and old graphical interfaces and monitoring software. Zabbix and LibreNMS will be briefly introduced and compared with Grafana and Prometheus.

There was not much to debate on what softwares to use in this project, so the questions came to be: What made Grafana more special than the old systems' visualization platforms? Why was Prometheus chosen to be paired up with Grafana when there already were two systems for collecting data? What advantages did these two monitoring systems have against the old systems?

1.1 Objectives

The main objective of this thesis was the implementation of a new monitoring system and graphical interface on the side of the current monitoring systems. There has not been a proper solution to display the client's own and their customers' data. The goal was to install and configure a visualization software, Grafana, for creating informational and aesthetical dashboards for Tietopartio's network environment which could also be used for customers' monitoring needs in the future. Prometheus was installed and configured as the monitoring tool to accompany Grafana.

A Unix-based server, where Grafana and Prometheus were installed, and a HP network switch were chosen as the example environments to monitor with Prometheus' different data gathering techniques. These techniques were node exporter and SNMP exporter. Two example dashboards were created in Grafana based on the collected data from the devices.

2 Used technologies, softwares, and environments

The main methods and softwares used in this project are explained in the following sections. The monitored devices are introduced briefly to give a better perception of the used backround in the client’s environment. Prometheus and two of Prometheus’ data collecting methods are introduced here, as well as Grafana and Grafana’s data visualization methods.

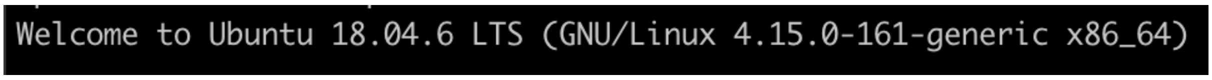
For this project, a new virtual server was created in one of the client’s servers in their data center. On that virtual server, a Unix-based Ubuntu 18.04 operating system was installed and iTerm2 was used to connect to the server via SSH.

2.1 Monitored devices

The monitored devices were the Ubuntu 18.04 virtual machine and a HP network switch in the client’s data center.

2.1.1 Ubuntu 18.04

Ubuntu is a Debian-based Linux operating system with free and open-source software. There are three three different editions of Ubuntu: desktop, server and core. For this project a generic Linux kernel Image was downloaded and installed to the virtual server (Ubuntu packages 2021) (Image 1). That means that there was no graphical interface and all the configuration had to be done in terminal with command line. (Wikipedia – The free encyclopedia 2021)



```
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-161-generic x86_64)
```

Image 1. Installed Ubuntu operation system.

2.1.2 HP Network Switch

Tietopartio’s data center had many possible candidates for monitoring purposes. HP 1810-24G network switch was selected for this project’s example SNMP monitoring subject (Image 2).

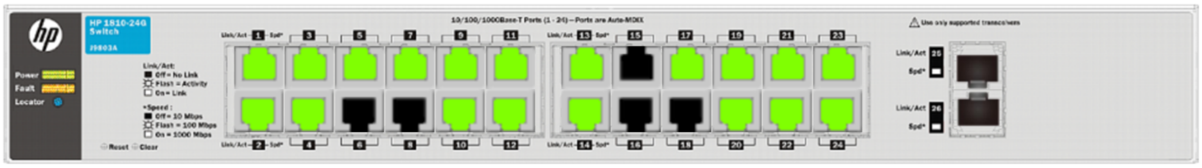


Image 2. HP 1810-24G switch ports in use.

2.2 Prometheus

Prometheus is an open-source monitoring and alerting toolkit based on collecting time series by scraping metrics over HTTP. The scrape is done by pulling the time series data from the targets. Prometheus is highly customizable, and the base server consists of only of the Prometheus server, database where timeseries data is stored locally and an expression browser where you can access and visualize collected data (Image 3). Other Prometheus’ features are completely optional, but you won’t get too far in monitoring without adding these features to the server. These optional features are client libraries, exporters, plugins and an alertmanager to handle alerts, which help you monitor whatever devices and services you have in your network environment. Prometheus supports various client libraries and exporters, but only a handful are under Prometheus’ own management. Most of the features are developed and maintained by third party operators, so caution should always be taken into consideration when applying them into your Prometheus server. If for some reason, the user cannot find an exporter or a client library for their needs, they can always make their own with the help of Prometheus’ documentation on how to write exporters and client libraries (Prometheus Authors 2014-2021a; Prometheus Authors 2014-2021c).

Prometheus also comes with its own query language, PromQL, which helps selecting and analyzing the time series data. (Brazil 2018; Prometheus Authors 2014-2021g)

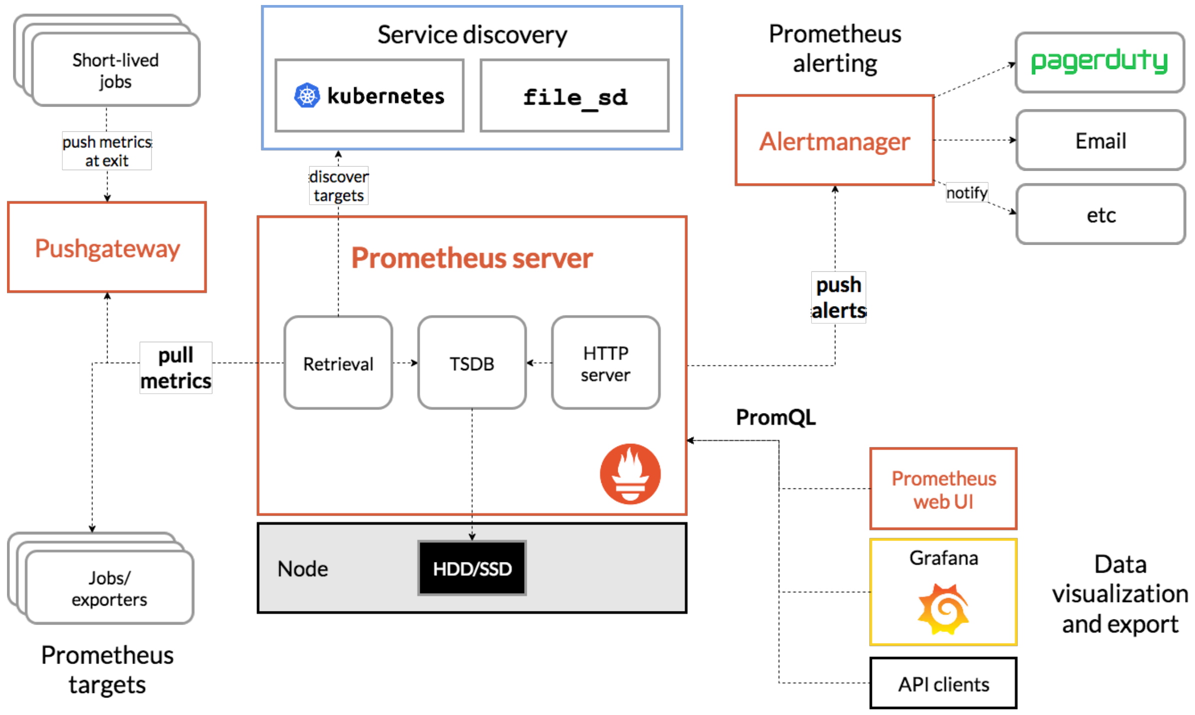


Image 3. Prometheus' architecture (screenshot from Prometheus' Overview page)

2.2.1 PromQL

Prometheus provides its own query language PromQL for querying metrics. PromQL significantly improves viewing and graphing metrics with different query methods. It allows the user to select, filter, group and aggregate metrics using functions and operators. The results can be viewed in a tabular form or shown as a graph in Prometheus' expression browser or in an external system. To avoid slow graphing or time outs, it is advised to construct the query in expression browser's tabular view before using it in a graph. (Brazil 2018; Prometheus Authors 2014-2021h)

2.3 Used methods for collecting data

Prometheus supports a few ways to collect data from various devices: client libraries, pushgateway, exporters and integrations. Client libraries enable monitoring by implementing lines of code to the target device or service and a matching client library installation to the Prometheus server (Prometheus Authors 2014-2021a). Pushgateway allows Prometheus to scrape targets that don’t support the pull mechanism for data collection. Pushgateway transforms pushed data into a temporary job that Prometheus can scrape.

Since most of Tietopartio’s environment consists of network devices, the main method of collecting data was using SNMP and snmp exporter. The second method that was used, was node exporter to collect time series data from the virtual server where Grafana and Prometheus servers were installed. Both exporters are part of the official Prometheus GitHub organization officially supported methods of collecting data.

2.3.1 Exporter

Not all applications and devices are accessible or controlled by the user, so exporters are the answer. Exporter is a piece of software that can be deployed to the application or a device where monitoring is required. Prometheus sends requests to the exporter, exporter takes those requests, gathers the requested data from the target and transforms the data into a format that Prometheus can read and returns the data to Prometheus server. There are a numerous of exporters supported by Prometheus. In the following section, two of these exporters are introduced; node exporter and snmp exporter. These are the two methods used with Prometheus in this project. (Prometheus Authors 2014-2021c)

2.3.2 Node exporter

Node exporter is designed to collect data from Unix environments. It exposes hardware- and kernel-related metrics from the target machine, for example CPU, memory, and disk space metrics. The Node exporter is intended only to monitor the machine itself, not individual processes or services on it (Brazil 2018). The exporter can be downloaded from Prometheus' download page, and it should be installed to the target machine. The node exporter runs on port 9100 as a default. (Github Inc. 2021a)

2.3.3 SNMP exporter

The SNMP exporter is especially useful when the user is trying to monitor network devices such as switches, routers or firewalls. SNMP function needs to be enabled on the target machine before the exporter is able to pull metrics from it. Using SNMP exporter requires knowledge of MIBs and OIDs. MIBs and OIDs define the information that is possible to pull from the devices. OID defines certain metrics in a MIB tree. There are universal MIBs that almost all the network devices support, but some devices require the manufactures own MIBs for specific information. OIDs are inputted to snmp.yml file where the exporter gets the information of the data to be searched and translates the data for Prometheus to understand. (Github Inc. 2021b; Prasath 2018)

2.4 Grafana

Grafana is an open-source database analytics and visualization tool. The purpose of Grafana is visualizing your most important data in an organized and informative manner. Grafana doesn't have any means of collecting data itself, but it supports numerous different data sources and has a unique query editor for all those data sources (Grafana Labs 2021b). Grafana is a running process on your computer or server and the interface is accessed through a web browser. Visual alerts are also possible to deploy in Grafana's graphs and

Grafana can send notifications about alerts to your email, for example (Grafana Labs 2021j).

2.4.1 Dashboards

Visually pleasing and useful dashboards can be either created or imported from Grafana’s dashboard download page. Different kinds of panels are used to showcase the most important data (Image 4). Grafana’s website defines a dashboard as “A set of one or more panels, organized and arranged into one or more rows, that provide an at-a-glance view of related information.” (Grafana Labs 2021d). Panels are dashboard’s basic building blocks. They consist of a query and a graphical representation of the query results. The time range of the dashboard can be controlled by the user. Time ranges can vary from a specific time and date to, for example, the last two hours. Grafana also has a feature called “variables”. Variables enable a selection of a target in the dashboard so that same dashboard can be used by multiple targets. There is a playlist option that lets the user to create a playlist containing already made dashboards. When the playlist is played, the dashboards change like in a slide show. The time between the dashboards can be altered (Grafana Labs 2021k).

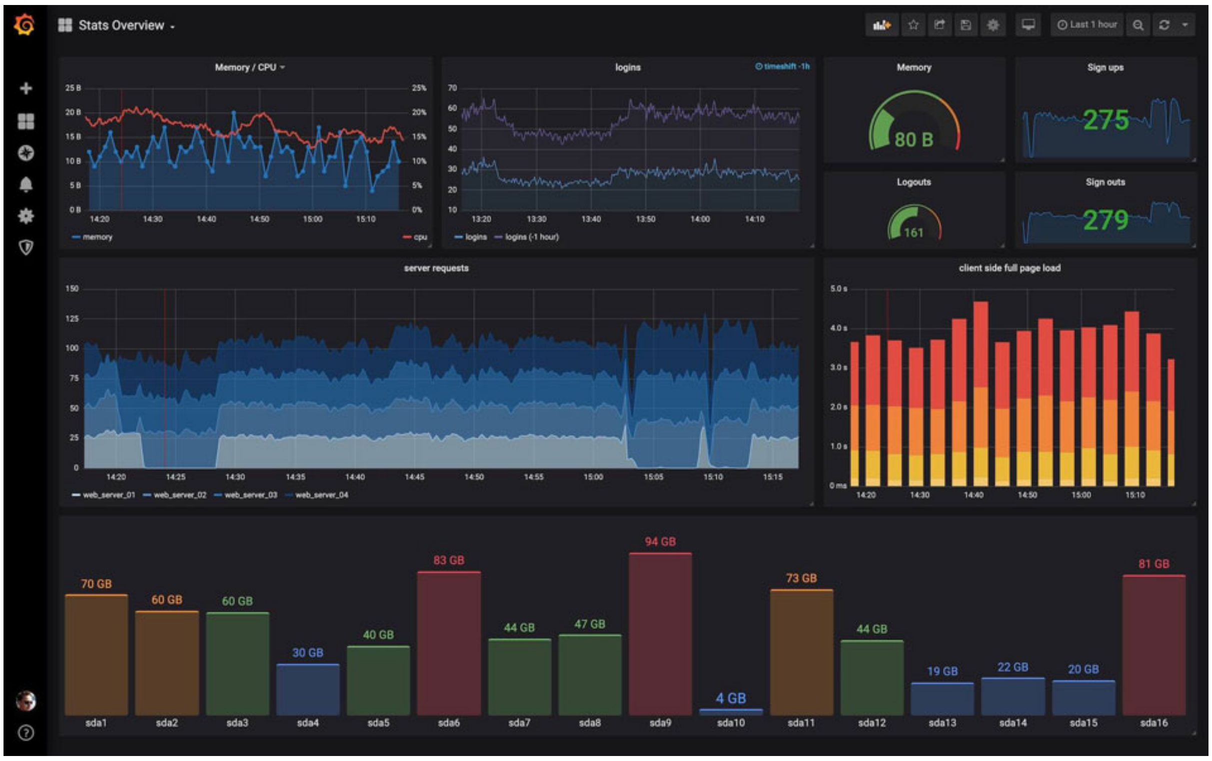


Image 4. Example dashboard with different panels. (Copied image from Grafana's Dashboards page via Google image search, Grafana Labs 2021c)

3 Comparison of the systems

Existing monitoring systems' basic functions are introduced briefly here. After the introduction, the existing systems are put side by side with the new monitoring systems to find out more of what each of them are good at, what sets them apart form each other and what kind of similarities they have.

This section compares Prometheus and Grafana to Zabbix and LibreNMS to determine why Prometheus was selected as the data collector and Grafana was selected as the visual representation for this project and what are the advantages of using these two monitoring tools.

3.1 Existing monitoring systems

Currently there are two monitoring systems actively monitoring Tietopartio's network environment. These are Zabbix and LibreNMS. Both are configured to alert on various of events that require an IT-specialist's actions. Zabbix and LibreNMS also offer their own visualization system, but these visualization systems haven't been fully utilized in the company for displaying data.

3.1.1 Zabbix

Zabbix is an enterprise-level monitoring tool, and the first beta version was released as early as 2001. Zabbix is an open-source software, which means it is free to download and the source code is accessible to anyone.

Zabbix consists of several components, but the major ones are the Zabbix server, database, Web interface and an Agent. With these components, you can easily and effectively collect metrics from your devices and services. The Agent is deployed on the target device or application to retrieve metrics for the Zabbix server. Metrics, as well as configurations and statistics, are stored in the Zabbix database. The Web interface allows the data being examined and viewed through dashboard visualizations.

Zabbix also offers alerting and notification system to stay on top of the network's performance. It can be configured to send e-mail notifications of different events. (Zabbix SIA 2001-2021d; Zabbix SIA 2001-2021f; Zabbix SIA 2001-2021h)

3.1.2 LibreNMS

LibreNMS is an open-source monitoring software that has many useful features for operating system and network monitoring. LibreNMS is based on PHP and uses multiple protocols for data collection (SNMP, STP, OSPF, etc). Some of LibreNMS' supported features are auto discovery, alerting, API, Syslog and many more. LibreNMS also has its own dashboard system for utilizing the data in a graphical manner (LibreNMS Docs c).

3.2 Comparing the monitoring systems

In comparing the data collector tools, the focus was on the data collection method, data storage and server structure. The goal was to gain better insight on how the systems worked and what are their primary use and strengths.

3.2.1 Data collection

Prometheus uses pull mechanism to collect data from its targets. It is also possible to use push mechanism when needed. The data is collected in the form of time series. Prometheus can be configured to scrape various kinds of devices and applications by downloading and installing exporters or using client libraries. Some of the exporters need to be installed on the target machine (i.e., node exporter) to extract metrics like an agent and some, for example SNMP, only need to be enabled on the target machine without any software installation on the target machine. The targets are configured in Prometheus' prometheus.yml file and sometimes in exporters own YAML-files.

Zabbix gathers data by deploying an agent to a monitored device, called host. Agents perform active and passive checks. Active checks are the ones where the Zabbix agent requests a list of items to gather form the host and in passive checks the Zabbix server is the one requesting the data (Zabbix SIA 2001-2021c). Zabbix also supports agentless monitoring, like Prometheus, when installing an agent to a host device is not possible. Zabbix uses both pull and push mechanism to collect metrics depending on the use case. Hosts can be added easily from the web interface. The user can define the collection method (Agent, SNMP) in the Add hosts -form (Zabbix SIA 2001-2021a).

LibreNMS supports Unix agent to collect data from Unix-based systems. It also supports various protocols like SNMP, OSPF and ARP to gather metrics from devices which don’t support agent installation. Monitored devices can be added from the Web UI, like in Zabbix, and auto discovery of devices is also possible.

3.2.2 Data storage

Prometheus stores the collected metrics data in a time series database (TSDB) on a local disk. An external storage is also supported. The data is stored 15 days as a default but can be overwritten to be a shorter or a longer time period. Log files are not supported in Prometheus, but there is a way to include logs into Prometheus monitoring by using Loki, a Grafana plugin for monitoring logs. (Prometheus Authors 2014-2021i)

In Zabbix the data is stored in an external database, for example MySQL, SQLite or Oracle. Data is stored in a form of history and trends. History keeps all the collected values which is resource consuming. Trends only keep minimum, maximum, average and total number of values making it less resource consuming way of storing data (Zabbix SIA 2001-2021g). The user can decide for how long to keep the acquired data. Besides time series monitoring, Zabbix also supports monitoring text and log files. (Zabbix LLC 2001-2021i)

LibreNMS uses external database for data storage, like Zabbix. Supported databases are MySQL, MariaDB et cetera. The metrics are stored as RRD (round-robin database) files, and they can be exported to other monitoring systems as well (i.e., Prometheus, Graphite). With LibreNMS, the user can monitor logs (LibreNMS Docs b; LibreNMS Docs d).

3.2.3 Server structure

Prometheus server consists of a time series database (TSDB), prometheus.yml configuration file, a script to run the Prometheus software and an HTTP server with web UI (see Image 3 from page 12).

Zabbix server is made of three core components, the server, agent and web interface. The data and configuration files are stored in the server part in a database that is configured during the installation. (Zabbix SIA 2001-2021e)

LibreNMS's basic installation contains a device poller/discovery, a time series data storage for RRD, an external database and a webserver (LibreNMS Docs b).

3.3 Advantages of Prometheus

It is safe to say that all these three monitoring systems have advantages of their own. But what are the advantages of using Prometheus?

Its level of customization is a clear advantage. Depending on the monitored systems and services, the user can install the right exporter or client library to match that system or service. And if there is no right data collector, the user can always create a new one with the help of Prometheus' documentation.

Prometheus has two powerful features: PromQL and the pre-installed TSDB. There is no need to install any external databases because Prometheus provides one automatically during installation. PromQL is a rich query language that can be used to analyze collected time series.

Pull mechanism, that Prometheus uses to gather data, is an efficient way to collect metrics. The monitoring server actively requests data form the targets, rather than the server passively waiting for targets to push their metrics to the monitoring server.

Prometheus provides an extensive community that offers support and help to Prometheus users. Documentation, blog post and courses also help Prometheus users to develop their skills in using the monitoring platform. (Prometheus Authors 2014-2021b)

3.4 Comparing visualization platforms

Tietopartio needed a system that is both eye-catching and informative for data display and data analytic purposes.

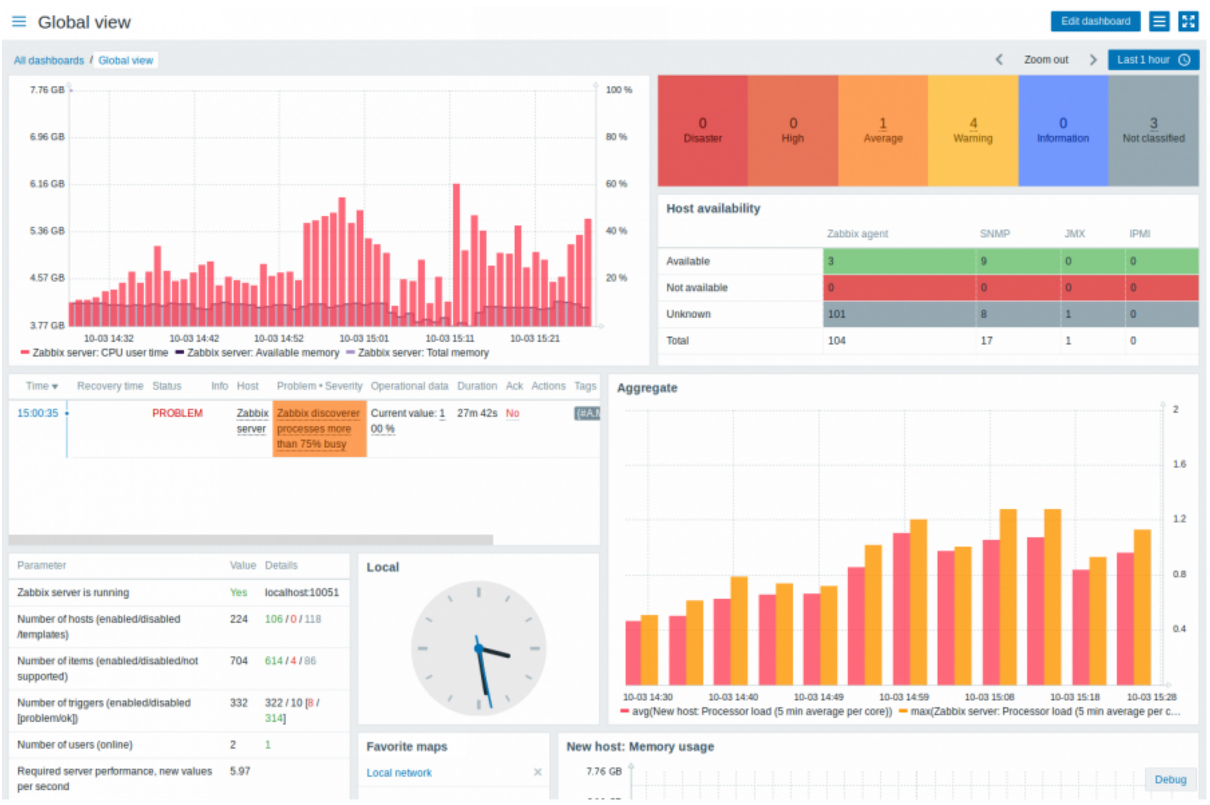
This section compares Grafana’s graphical interface to Zabbix’s and LibreNMS’ ways to visualize data. The purpose is to look at the dashboard features, functionalities, and appearances. There are of course, a numerous amount of other open source and paid graphical solutions, but since Zabbix and LibreNMS are already used in Tietopartio’s network environment, the comparison is done between these systems.

Grafana’s dashboard features and functions were explained earlier beginning from the page 14.

3.4.1 Data visualization in Zabbix

Like Grafana, Zabbix dashboard is also accessed via web UI. Zabbix dashboard consists of panels that are called widgets (Image 5). The user selects type of widget and fills the configuration form and then Zabbix creates the widget based on the supplied information. Graph widgets can be customized and configured in multiple ways, for example an aggregation function can be added, and the colors of the graph can be changed. Widgets can be resized and moved around in the dashboard editing mode. Dashboards can have multiple pages and they can be displayed in a slide show. Pages can be reordered and copied to a new dashboard or to an existing page. Graph widgets can be downloaded as PNG image.

It is also possible to configure a “Dynamic Item”, which enables selecting a host for a dashboard. The benefit is that the user can have the same dashboard for multiple hosts, the data changes in the widgets when selecting a different host, just like in Grafana. (Zabbix SIA 2001-2021b)



Host availability

	Zabbix agent	SNMP	JMX	IPMI
Available	3	9	0	0
Not available	0	0	0	0
Unknown	101	8	1	0
Total	104	17	1	0

Time ▾	Recovery time	Status	Info	Host	Problem • Severity	Operational data	Duration	Ack	Actions	Tags
15:00:35		PROBLEM		Zabbix server	Zabbix discoverer processes more than 75% busy	Current value: 100 %	27m 42s	No		{#A.I

Parameter	Value	Details
Zabbix server is running	Yes	localhost:10051
Number of hosts (enabled/disabled /templates)	224	106 / 0 / 118
Number of items (enabled/disabled/not supported)	704	614 / 4 / 86
Number of triggers (enabled/disabled [problem/ok])	332	322 / 10 [8 / 314]
Number of users (online)	2	1
Required server performance, new values per second	5.97	

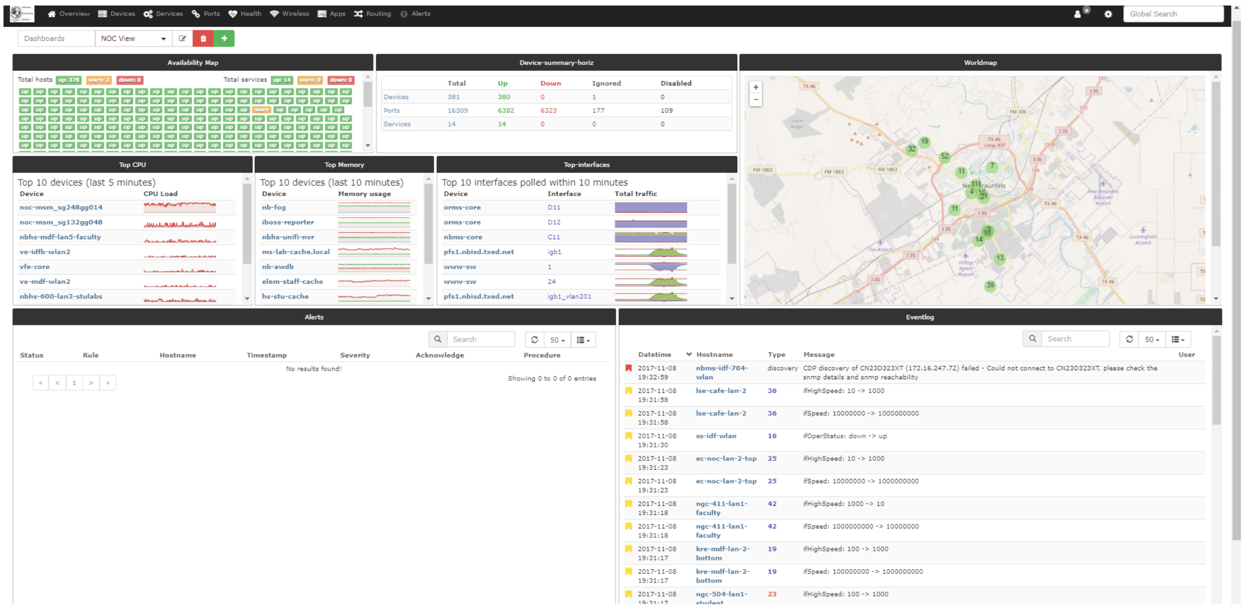
Image 5. Example Zabbix dashboard with different widgets. (Image copied from the Zabbix Dashboard page, Zabbix SIA 2001-2021b)

3.4.2 Data visualization in LibreNMS

LibreNMS dashboards are accessible through web UI. Dashboards are created by using various widgets. LibreNMS widgets offer easy to use overall view of your network. Selected widgets transform into panels in the dashboard environment (Image 6). The panels can be rearranged, and the size is adjustable. You can also name the panels as you please, and if you don't name the panel, LibreNMS creates an automatic title.

A standout feature in LibreNMS is the automatically created graphs from device discoveries. For example, LibreNMS creates an overall view of the selected device with system information (name, hardware, operation system, etc) overall traffic, processors, and memory. For closer inspection of the devices' performance, the user can browse graphs throughout different time periods.

There is a downside for everything being automatic. The user is not given so much freedom when creating the panels. The user cannot change visualizations or colors of the panels. Widgets/panels or dashboards can't be copied from one dashboard to the other. You also cannot copy automatically created panels for your own dashboard. LibreNMS dashboards support only its own data source (LibreNMS Docs a).



Device-summary-horiz

	Total	Up	Down	Ignored	Disabled
Devices	381	380	0	1	0
Ports	16309	6382	6323	177	109
Services	14	14	0	0	0

Top CPU

Top 10 devices (last 5 minutes)

Device	CPU Load
noc-msm_sg248gg014	
noc-msm_sg132gg048	
nbhs-mdf-lan5-faculty	
ve-idfb-wlan2	
vfe-core	
ve-mdf-wlan2	
nbhs-600-lan3-stulabs	

Top Memory

Top 10 devices (last 10 minutes)

Device	Memory usage
nb-fog	
iboss-reporter	
nbhs-unifi-nvr	
ms-lab-cache.local	
nb-awdb	
elem-staff-cache	
hs-stu-cache	

Top-interfaces

Top 10 interfaces polled within 10 minutes

Device	Interface	Total traffic
orms-core	D11	
orms-core	D12	
nbms-core	C11	
pfs1.nbisd.txed.net	igb1	
www-sw	1	
www-sw	24	
pfs1.nbisd.txed.net	igb1_vlan201	

Alerts

Status	Rule	Hostname	Timestamp	Severity	Acknowledge	Procedure

No results found!

Showing 0 to 0 of 0 entries

Eventlog

Datetime	Hostname	Type	Message	User
2017-11-08 19:32:59	nbms-idf-704-wlan	discovery	CDP discovery of CN23D323XT (172.16.247.72) failed - Could not connect to CN23D323XT, please check the snmp details and snmp reachability	
2017-11-08 19:31:58	lse-cafe-lan-2	36	ifHighSpeed: 10 -> 1000	
2017-11-08 19:31:58	lse-cafe-lan-2	36	ifSpeed: 10000000 -> 1000000000	
2017-11-08 19:31:30	ss-idf-wlan	16	ifOperStatus: down -> up	
2017-11-08 19:31:23	ec-noc-lan-2-top	25	ifHighSpeed: 10 -> 1000	
2017-11-08 19:31:23	ec-noc-lan-2-top	25	ifSpeed: 10000000 -> 1000000000	
2017-11-08 19:31:18	ngc-411-lan1-faculty	42	ifHighSpeed: 1000 -> 10	
2017-11-08 19:31:18	ngc-411-lan1-faculty	42	ifSpeed: 1000000000 -> 10000000	
2017-11-08 19:31:17	kre-mdf-lan-2-bottom	19	ifHighSpeed: 100 -> 1000	
2017-11-08 19:31:17	kre-mdf-lan-2-bottom	19	ifSpeed: 100000000 -> 1000000000	
2017-11-08 19:31:17	ngc-504-lan1-student	23	ifHighSpeed: 100 -> 1000	

Image 6. Example LibreNMS dashboard with different panels. (Image copied from LibreNMS' Dashboard page, LibreNMS Docs a)

3.5 Advantages of Grafana

Compared to Zabbix and LibreNMS, Grafana has some advantages. Zabbix and LibreNMS support only one data source, their own, whereas Grafana supports several data sources. Some of the data sources, for example, Prometheus and Graphite, are built-in features in Grafana and other data sources can be added by installing plugins. Because of this, Grafana has a specific Query Editor for all the data sources. Creation of multiple panels with different data sources in the same dashboard is also supported in Grafana.

All these monitoring systems offer versatile visualization types and modifications to dashboards. Grafana, on the other hand, is specialized in visualization and modification of dashboards. Even in the monitoring world, looks matter. Grafana's dashboards are aesthetically pleasing while still being informative, and therefore Grafana is so popular among monitoring communities.

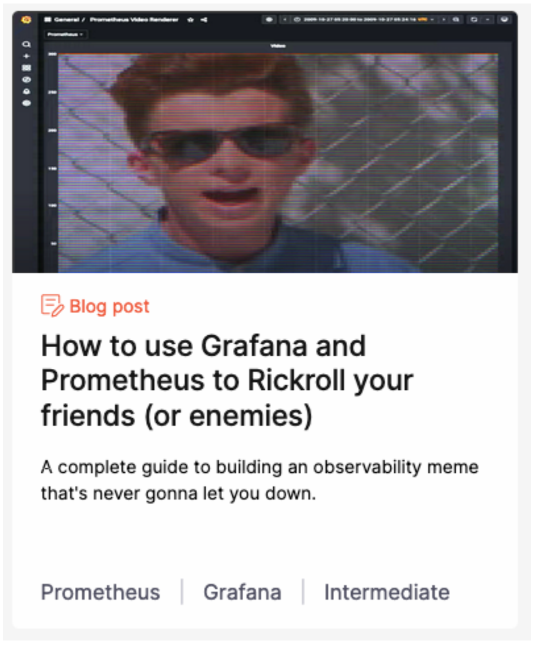
What sets Grafana aside from other visualization tools, is the ability to share and import dashboards. There is a whole page in Grafana's web site dedicated to dashboards where the user can search and import dashboard into their own Grafana. Also anyone can submit their dashboards to that page.

One impressive feature is automatic rendering of panels as PNG images. Images can also be downloaded straight from the dashboard (Grafana Labs 2021h).

Grafana is constantly evolving with newer versions that are trying to make monitoring easier and more versatile. With several visualization choices, the user can make their dashboards look outstanding while still being easy to understand and informative.

Communities are very important and a big part of open source softwares. Grafana community offers a comprehensive documentation, guides, tutorials and webinars and videos about the software and its usage (Image 7).

Community page offers support and help for users' problems and the users can discus all things related to Grafana.



Blog post

How to use Grafana and Prometheus to Rickroll your friends (or enemies)

A complete guide to building an observability meme that's never gonna let you down.

Prometheus | Grafana | Intermediate

Image 7. Example of a tutorial offered in Grafana's website. (Screenshot from Grafana's tutorials page, Grafana Labs 2021g)

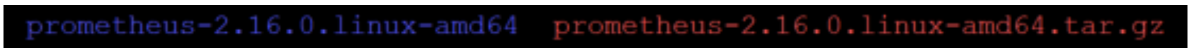
4 Implementation of Prometheus and Grafana

Implementation of Prometheus, exporters, Grafana and dashboards are reviewed in this part. The Images show the Unix operating system, Prometheus' expression browser and Grafana's web interface in use. Installations of each software are written separately and explained in depth for better understanding of the process.

4.1 Prometheus

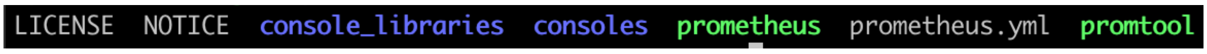
A new virtual server was created to one of Tietopartio's servers and Unix 18.04 operating system was installed in the virtual server. A user and a password for that user were created to the server. To gain access to the virtual server, SSH connection in iTerm2 software was used.

Using 'wget' command enabled the download of the latest installation media from Prometheus' web site and with 'tar' command the media file was installed on the server. Image 8 shows the downloaded and installed Prometheus server. Image 9 shows the contents of the installed Prometheus server.



```
prometheus-2.16.0.linux-amd64  prometheus-2.16.0.linux-amd64.tar.gz
```

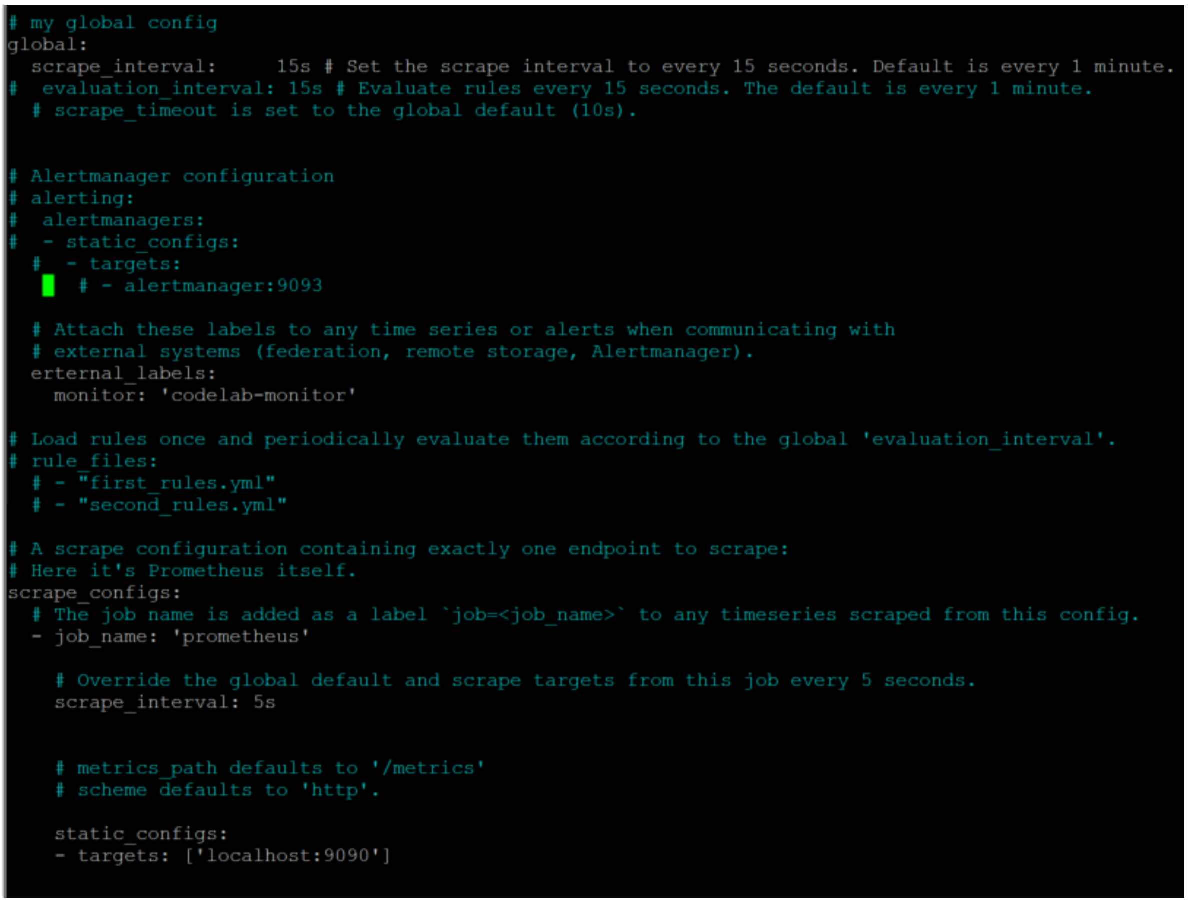
Image 8. Highlighted in red is the downloaded Prometheus server. In blue, is the installed Prometheus server.



```
LICENSE  NOTICE  console_libraries  consoles  prometheus  prometheus.yml  promtool
```

Image 9. Contents of the Prometheus server.

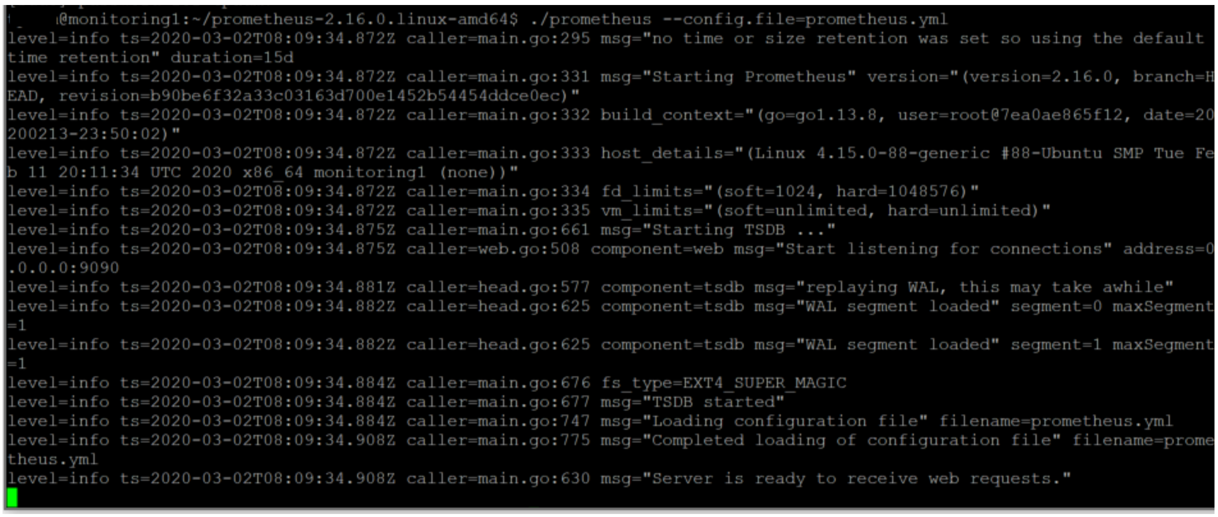
The prometheus.yml file is where targets are specified and what Prometheus will monitor. The most important lines in the YAML file are scrape_interval, and the ones starting with scrape_configs. The prometheus.yml file can be modified as long as the YAML syntax stays right, otherwise the code doesn’t work. So far, there is only one job, named ‘prometheus’, with a scrape interval of 15 seconds and a target ‘localhost:9090’. This job monitors Prometheus itself (Image 10).



```
# 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

  # Attach these labels to any time series or alerts when communicating with
  # external systems (federation, remote storage, Alertmanager).
  erternal_labels:
    monitor: 'codelab-monitor'

# 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'

    # Override the global default and scrape targets from this job every 5 seconds.
    scrape_interval: 5s


    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
    - targets: ['localhost:9090']
```

Image 10. Default prometheus.yml. The lines marked with # are comments and therefore don’t affect to the execution of the program.

To start up Prometheus, the ‘prometheus’ scipt was run (Image 11).

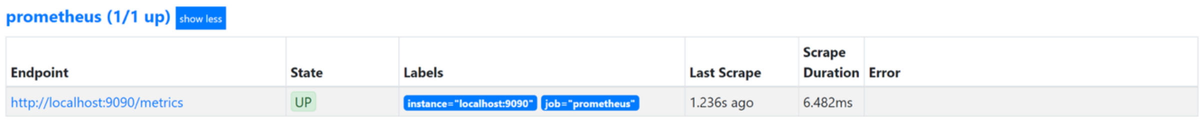


```
@monitoring1:~/prometheus-2.16.0.linux-amd64$ ./prometheus --config.file=prometheus.yml
level=info ts=2020-03-02T08:09:34.872Z caller=main.go:295 msg="no time or size retention was set so using the default
time retention" duration=15d
level=info ts=2020-03-02T08:09:34.872Z caller=main.go:331 msg="Starting Prometheus" version="(version=2.16.0, branch=H
EAD, revision=b90be6f32a33c03163d700e1452b54454ddce0ec)"
level=info ts=2020-03-02T08:09:34.872Z caller=main.go:332 build_context="(go=go1.13.8, user=root@7ea0ae865f12, date=20
200213-23:50:02)"
level=info ts=2020-03-02T08:09:34.872Z caller=main.go:333 host_details="(Linux 4.15.0-88-generic #88-Ubuntu SMP Tue Fe
b 11 20:11:34 UTC 2020 x86_64 monitoring1 (none))"
level=info ts=2020-03-02T08:09:34.872Z caller=main.go:334 fd_limits="(soft=1024, hard=1048576)"
level=info ts=2020-03-02T08:09:34.872Z caller=main.go:335 vm_limits="(soft=unlimited, hard=unlimited)"
level=info ts=2020-03-02T08:09:34.875Z caller=main.go:661 msg="Starting TSDB ..."
level=info ts=2020-03-02T08:09:34.875Z caller=web.go:508 component=web msg="Start listening for connections" address=0
.0.0.0:9090
level=info ts=2020-03-02T08:09:34.881Z caller=head.go:577 component=tsdb msg="replaying WAL, this may take awhile"
level=info ts=2020-03-02T08:09:34.882Z caller=head.go:625 component=tsdb msg="WAL segment loaded" segment=0 maxSegment
=1
level=info ts=2020-03-02T08:09:34.882Z caller=head.go:625 component=tsdb msg="WAL segment loaded" segment=1 maxSegment
=1
level=info ts=2020-03-02T08:09:34.884Z caller=main.go:676 fs_type=EXT4_SUPER_MAGIC
level=info ts=2020-03-02T08:09:34.884Z caller=main.go:677 msg="TSDB started"
level=info ts=2020-03-02T08:09:34.884Z caller=main.go:747 msg="Loading configuration file" filename=prometheus.yml
level=info ts=2020-03-02T08:09:34.908Z caller=main.go:775 msg="Completed loading of configuration file" filename=prome
theus.yml
level=info ts=2020-03-02T08:09:34.908Z caller=main.go:630 msg="Server is ready to receive web requests."
```

Image 11. Starting up Prometheus server. The top line in the picture is the command to start up the server. The bottom line indicates that the server is up and running and ready to pull data.

The server is working properly, as can be seen form the bottom line "Server is ready to receive web requests." (Image 11)

After letting Prometheus scrape its target, the metrics were viewed in Prometheus' expression browser. Image 12 confirms that the Prometheus server is working and getting metrics from its target 'prometheus' job.



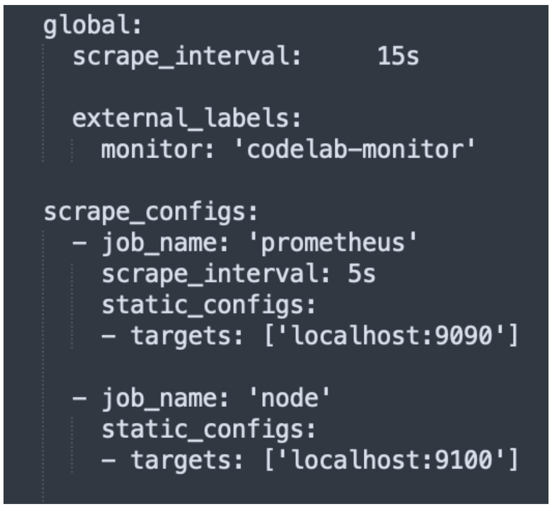
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	1.236s ago	6.482ms	

Image 12. Prometheus expression browser's overview of the targets.

Prometheus' graphical interface is very basic and is not useful for the long run. It is better to use another software for graphical purposes. Prometheus' expression browser is great for ad-hoc quaries. (Prometheus Authors 2014-2021d; Prometheus Authors 2014-2021e)

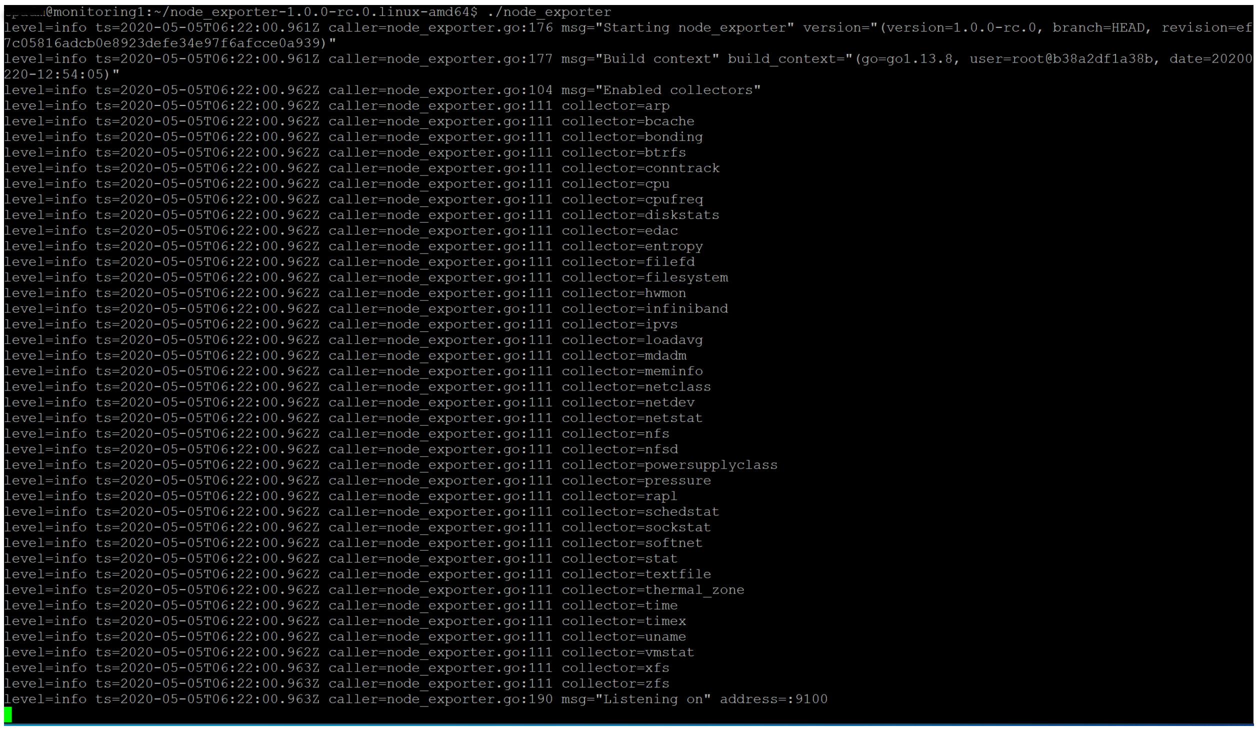
4.2 Node exporter

To monitor the virtual server where Prometheus was installed, the latest node exporter pakage was downloaded from node exporter Github page using ‘wget’ command and it was installed using ‘tar’ command, just like previously for Prometheus server. To get the node exporter to monitor the server, a job using node exporter needed to be configured in prometheus.yml file. A ’node’ job was added in the scrape_configs section of the YAML file and targeted to the localhost on port 9100 (Image 13). The node exporter was started from the script with ‘./node_exporter’ command (Image 14).



```yaml
global:
  scrape_interval:     15s

  external_labels:
    monitor: 'codelab-monitor'

scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 5s
    static_configs:
    - targets: ['localhost:9090']

  - job_name: 'node'
    static_configs:
    - targets: ['localhost:9100']
```

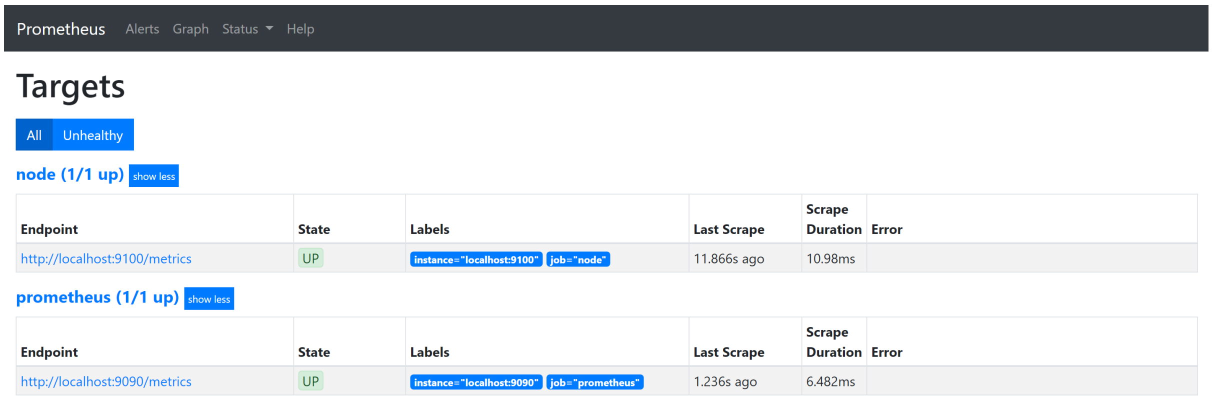
Image 13. prometheus.yml configuration with ‘node’ job.



```
[illegible]@monitoring1:~/node_exporter-1.0.0-rc.0.linux-amd64$ ./node_exporter
level=info ts=2020-05-05T06:22:00.961Z caller=node_exporter.go:176 msg="Starting node_exporter" version="(version=1.0.0-rc.0, branch=HEAD, revision=ef
7c05816adcb0e8923defe34e97f6afcce0a939)"
level=info ts=2020-05-05T06:22:00.961Z caller=node_exporter.go:177 msg="Build context" build_context="(go=go1.13.8, user=root@b38a2df1a38b, date=20200
220-12:54:05)"
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:104 msg="Enabled collectors"
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=arp
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=bcache
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=bonding
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=btrfs
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=conntrack
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=cpu
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=cpufreq
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=diskstats
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=edac
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=entropy
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=filefd
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=filesystem
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=hwmon
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=infiniband
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=ipvs
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=loadavg
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=mdadm
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=meminfo
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=netclass
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=netdev
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=netstat
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=nfs
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=nfsd
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=powersupplyclass
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=pressure
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=rapl
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=schedstat
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=sockstat
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=softnet
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=stat
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=textfile
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=thermal_zone
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=time
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=timex
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=uname
level=info ts=2020-05-05T06:22:00.962Z caller=node_exporter.go:111 collector=vmstat
level=info ts=2020-05-05T06:22:00.963Z caller=node_exporter.go:111 collector=xfs
level=info ts=2020-05-05T06:22:00.963Z caller=node_exporter.go:111 collector=zfs
level=info ts=2020-05-05T06:22:00.963Z caller=node_exporter.go:190 msg="Listening on" address=:9100
```

Image 14. Node exporters running sequence. All the collectors are listed what kind of data node exporter is collecting from the target. Last line tells the port which node exporter is using to listen the target.

The working of the node exporter was verified in the Prometheus expression browser (Image 15). (Github Inc. 2021a ; Prometheus Authors 2014-2021f)



Targets

node (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9100/metrics	UP	instance="localhost:9100" job="node"	11.866s ago	10.98ms	

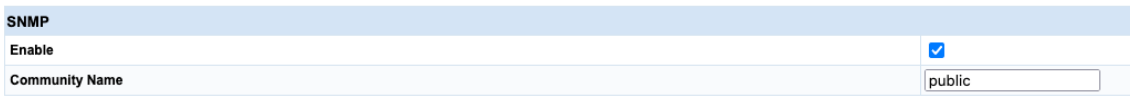
prometheus (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	1.236s ago	6.482ms	

Image 15. From the Prometheus expression browser Targets tab with ‘prometheus’ and ‘node’ jobs.

4.3 SNMP exporter

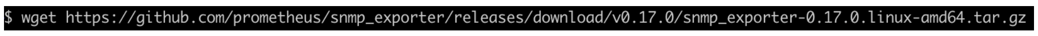
To be able to use SNMP to monitor devices, SNMP has to be enabled from the target device (Image 16). Since SNMPv2 was used in this project, the community string needed to be known for identification of the device. ‘public’ is the default community string and it is adviced to use something stronger and safer string in the actual use.



SNMP	
Enable	✓
Community Name	public

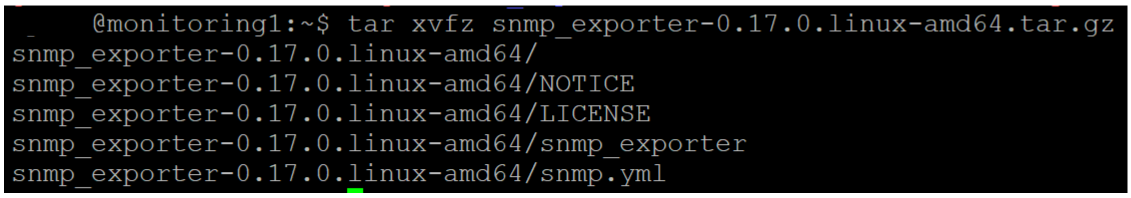
Image 16. SNMP enabled on the network switch and the switche’s community string.

SNMP expoter was downloaded and istalled the same way as Prometheus and node exporter by using ‘wget’ (Image 17) and ‘tar’ (Image 18) commands.



```
$ wget https://github.com/prometheus/snmp_exporter/releases/download/v0.17.0/snmp_exporter-0.17.0.linux-amd64.tar.gz
```

Image 17. Using wget command to download snmp exporter from Github web page.

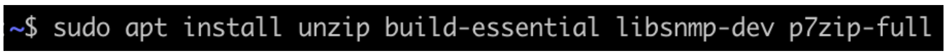


```
[redacted]@monitoring1:~$ tar xvfz snmp_exporter-0.17.0.linux-amd64.tar.gz
snmp_exporter-0.17.0.linux-amd64/
snmp_exporter-0.17.0.linux-amd64/NOTICE
snmp_exporter-0.17.0.linux-amd64/LICENSE
snmp_exporter-0.17.0.linux-amd64/snmp_exporter
snmp_exporter-0.17.0.linux-amd64/snmp.yml
```

Image 18. Using tar command to extract the snmp exporter file.

In order to get the SNMP exporter to work, SNMP needed to be enabled on the server.

‘libsnmp-dev’ a Debian-based distribution of SNMP development files were installed on the server as shown in the Image 19.

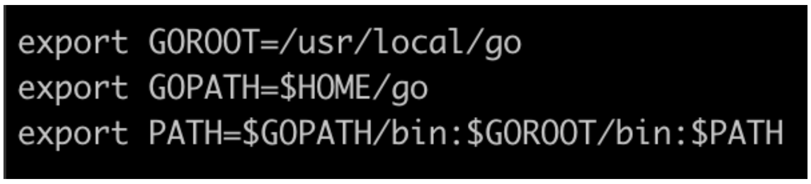


```
~$ sudo apt install unzip build-essential libsnmp-dev p7zip-full
```

Image 19. Debian-based distribution download command.

Because Prometheus is written in Go language, the snmp exporter needed a Go environment, therefore Go was downloaded and installed to the server.

Go was downloaded and installed on the server and copied to /usr/local/go. The /etc/profile file was edited by implementing a path variable for the system to find Go’s executable binaries (Image 20).

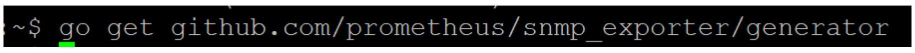


```
export GOROOT=/usr/local/go
export GOPATH=$HOME/go
export PATH=$GOPATH/bin:$GOROOT/bin:$PATH
```

Image 20. Path variable, Go root and Go path for Go.

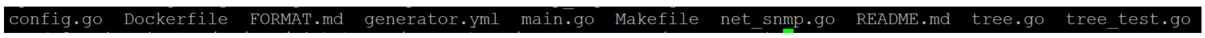
A network switch in Tietopartio’s data center was selected as an example for the SNMP exporter. In order to configure prometheus.yml with the SNMP target, the snmp.yml needed to be configured first. It is not advised to configure snmp.yml manually and therefore a generator has been developed to do the work. The generator can be downloaded from the snmp exporter Github page.

With ‘go get’ command the generator was downloaded to the server (Image 21). Contents of the downloaded generator are displayed in the Image 22.



```
~$ go get github.com/prometheus/snmp_exporter/generator
```

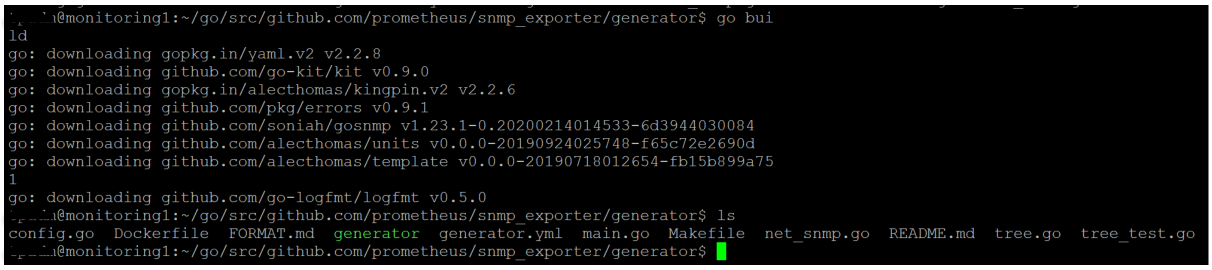
Image 21. Generator download.



```
config.go  Dockerfile  FORMAT.md  generator.yml  main.go  Makefile  net_snmp.go  README.md  tree.go  tree_test.go
```

Image 22. Contents of the generator directory.

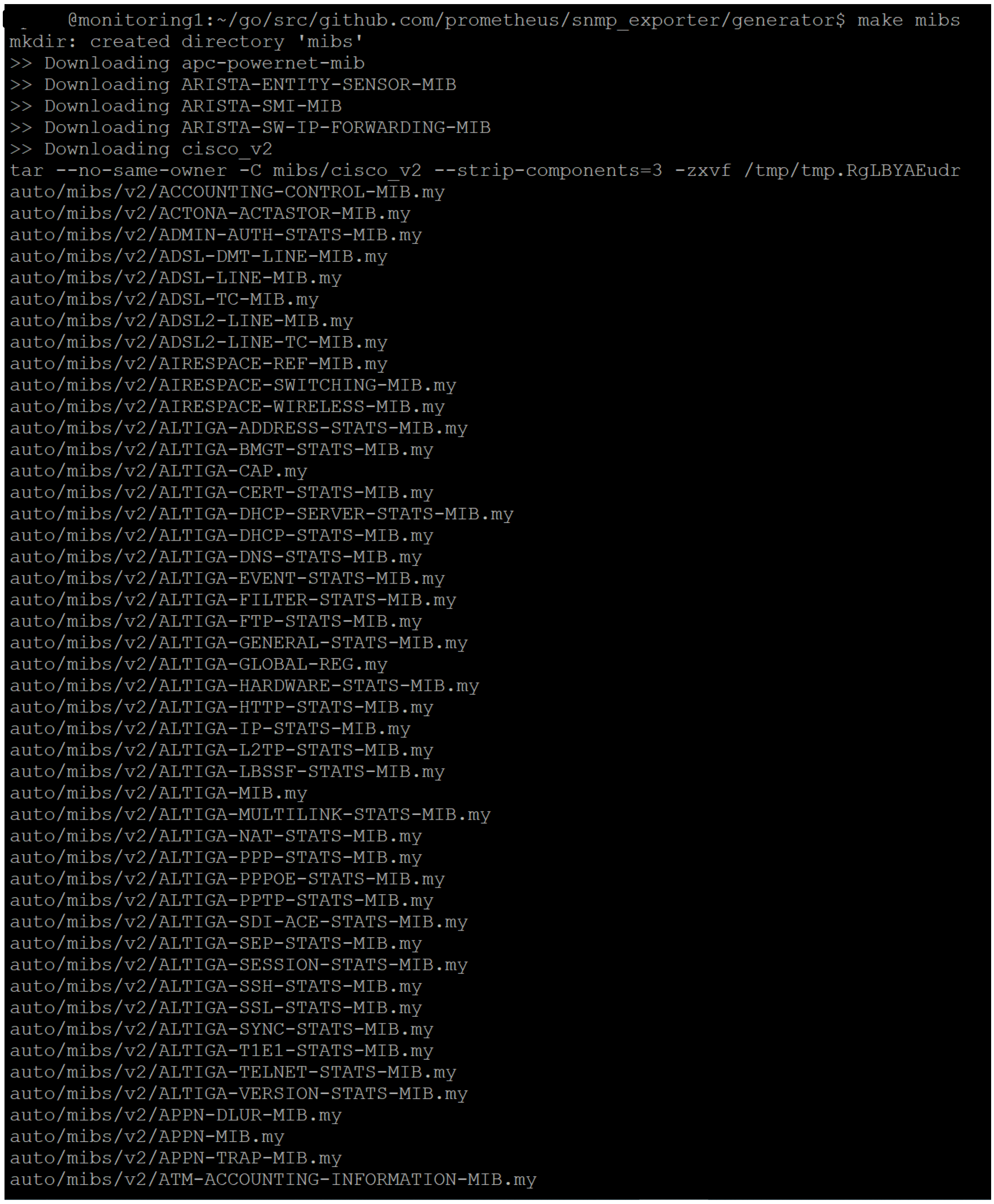
'go build' command created the generator script that was needed for running the program (Image 23).



```
[redacted]@monitoring1:~/go/src/github.com/prometheus/snmp_exporter/generator$ go bui
ld
go: downloading gopkg.in/yaml.v2 v2.2.8
go: downloading github.com/go-kit/kit v0.9.0
go: downloading gopkg.in/alecthomas/kingpin.v2 v2.2.6
go: downloading github.com/pkg/errors v0.9.1
go: downloading github.com/soniah/gosnmp v1.23.1-0.20200214014533-6d3944030084
go: downloading github.com/alecthomas/units v0.0.0-20190924025748-f65c72e2690d
go: downloading github.com/alecthomas/template v0.0.0-20190718012654-fb15b899a75
1
go: downloading github.com/go-logfmt/logfmt v0.5.0
[redacted]@monitoring1:~/go/src/github.com/prometheus/snmp_exporter/generator$ ls
config.go  Dockerfile  FORMAT.md  generator  generator.yml  main.go  Makefile  net_snmp.go  README.md  tree.go  tree_test.go
[redacted]@monitoring1:~/go/src/github.com/prometheus/snmp_exporter/generator$
```

Image 23. Building of the generator script which can be seen on the green.

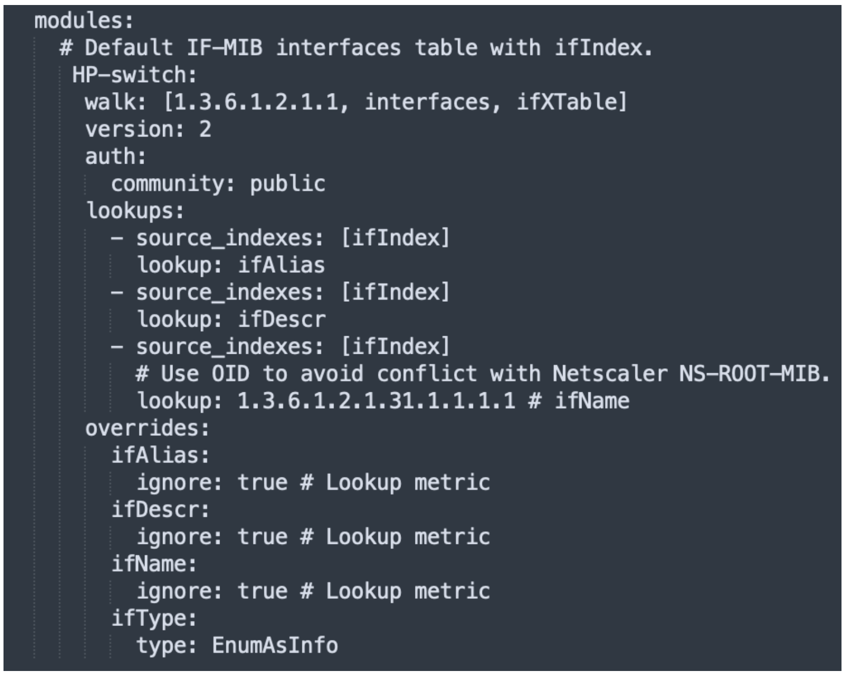
The 'make mibs' command created a directory called 'mibs' to the generator directory and it downloaded a bunch of different MIB files to the 'mibs' directory (Image 24). (Bradley 2020-2021; Github Inc. 2021b; Github Inc. 2021c)



```
[redacted]@monitoring1:~/go/src/github.com/prometheus/snmp_exporter/generator$ make mibs
mkdir: created directory 'mibs'
>> Downloading apc-powernet-mib
>> Downloading ARISTA-ENTITY-SENSOR-MIB
>> Downloading ARISTA-SMI-MIB
>> Downloading ARISTA-SW-IP-FORWARDING-MIB
>> Downloading cisco_v2
tar --no-same-owner -C mibs/cisco_v2 --strip-components=3 -zxvf /tmp/tmp.RgLBYAEudr
auto/mibs/v2/ACCOUNTING-CONTROL-MIB.my
auto/mibs/v2/ACTONA-ACTASTOR-MIB.my
auto/mibs/v2/ADMIN-AUTH-STATS-MIB.my
auto/mibs/v2/ADSL-DMT-LINE-MIB.my
auto/mibs/v2/ADSL-LINE-MIB.my
auto/mibs/v2/ADSL-TC-MIB.my
auto/mibs/v2/ADSL2-LINE-MIB.my
auto/mibs/v2/ADSL2-LINE-TC-MIB.my
auto/mibs/v2/AIRESPACE-REF-MIB.my
auto/mibs/v2/AIRESPACE-SWITCHING-MIB.my
auto/mibs/v2/AIRESPACE-WIRELESS-MIB.my
auto/mibs/v2/ALTIGA-ADDRESS-STATS-MIB.my
auto/mibs/v2/ALTIGA-BMGT-STATS-MIB.my
auto/mibs/v2/ALTIGA-CAP.my
auto/mibs/v2/ALTIGA-CERT-STATS-MIB.my
auto/mibs/v2/ALTIGA-DHCP-SERVER-STATS-MIB.my
auto/mibs/v2/ALTIGA-DHCP-STATS-MIB.my
auto/mibs/v2/ALTIGA-DNS-STATS-MIB.my
auto/mibs/v2/ALTIGA-EVENT-STATS-MIB.my
auto/mibs/v2/ALTIGA-FILTER-STATS-MIB.my
auto/mibs/v2/ALTIGA-FTP-STATS-MIB.my
auto/mibs/v2/ALTIGA-GENERAL-STATS-MIB.my
auto/mibs/v2/ALTIGA-GLOBAL-REG.my
auto/mibs/v2/ALTIGA-HARDWARE-STATS-MIB.my
auto/mibs/v2/ALTIGA-HTTP-STATS-MIB.my
auto/mibs/v2/ALTIGA-IP-STATS-MIB.my
auto/mibs/v2/ALTIGA-L2TP-STATS-MIB.my
auto/mibs/v2/ALTIGA-LBSSF-STATS-MIB.my
auto/mibs/v2/ALTIGA-MIB.my
auto/mibs/v2/ALTIGA-MULTILINK-STATS-MIB.my
auto/mibs/v2/ALTIGA-NAT-STATS-MIB.my
auto/mibs/v2/ALTIGA-PPP-STATS-MIB.my
auto/mibs/v2/ALTIGA-PPPOE-STATS-MIB.my
auto/mibs/v2/ALTIGA-PPTP-STATS-MIB.my
auto/mibs/v2/ALTIGA-SDI-ACE-STATS-MIB.my
auto/mibs/v2/ALTIGA-SEP-STATS-MIB.my
auto/mibs/v2/ALTIGA-SESSION-STATS-MIB.my
auto/mibs/v2/ALTIGA-SSH-STATS-MIB.my
auto/mibs/v2/ALTIGA-SSL-STATS-MIB.my
auto/mibs/v2/ALTIGA-SYNC-STATS-MIB.my
auto/mibs/v2/ALTIGA-T1E1-STATS-MIB.my
auto/mibs/v2/ALTIGA-TELNET-STATS-MIB.my
auto/mibs/v2/ALTIGA-VERSION-STATS-MIB.my
auto/mibs/v2/APPN-DLUR-MIB.my
auto/mibs/v2/APPN-MIB.my
auto/mibs/v2/APPN-TRAP-MIB.my
auto/mibs/v2/ATM-ACCOUNTING-INFORMATION-MIB.my
```

Image 24. ‘make mibs’ command and few examples of downloaded MIBs.

With all the preparations done, it was time to configure a generator.yml file for a generator script to use. The generator script uses generator.yml to create snmp.yml. The generator came with a pre-configured generator.yml file full of ready-made modules. Instead using the default generator.yml file, a new YAML file was made with only one module for the network switch (Image 25).



```yaml
modules:
  # Default IF-MIB interfaces table with ifIndex.
   HP-switch:
    walk: [1.3.6.1.2.1.1, interfaces, ifXTable]
    version: 2
    auth:
      community: public
    lookups:
      - source_indexes: [ifIndex]
        lookup: ifAlias
      - source_indexes: [ifIndex]
        lookup: ifDescr
      - source_indexes: [ifIndex]
        # Use OID to avoid conflict with Netscaler NS-ROOT-MIB.
        lookup: 1.3.6.1.2.1.31.1.1.1.1 # ifName
    overrides:
      ifAlias:
        ignore: true # Lookup metric
      ifDescr:
        ignore: true # Lookup metric
      ifName:
        ignore: true # Lookup metric
      ifType:
        type: EnumAsInfo
```

Image 25. Configured generator.yml file

The layout of one the premade modules was copied to the new generator.yml file and was filled with the information from the network switch.

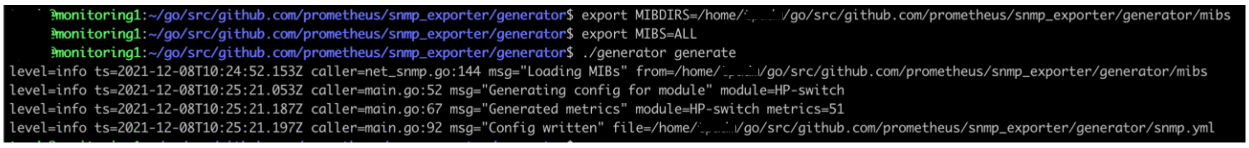
Generator.yml consists of modules and the modules in the simplest form can consist of only module name and list of OIDs to walk. In the Image 25 example, the module consists of:

- Module name
- OIDs to walk
- Version
- Auth
- Lookups
- Overrides

The user can define module name to be whatever, preferably the devices name or something to distinguish from other modules if there are a lot of them. OIDs can be listed in their numerical form or their SNMP object name, for example

1.3.6.1.2.1.2 and ‘interfaces’ mean the same thing. Version defines what SNMP version to use. Auth, short for authentication, uses a community string to authenticate the device. Lookups can create a different label and keep the same value from the table it is looking from, for example ifIndex will lookup ifAlias table entry and create a ifAlias label. Overrides allow bits of MIBs to be overridden. Ignore: true drops the metric from the output and type: EnumAsInfo overrides the metric type as an enum for which a single timeseries is created. (Github Inc. 2021c; Github Inc. 2021d)

The generator needed to be told where to find the MIB files before starting up the process. The commands ‘export MIBDIRS=<location of the MIBs>’ and ‘MIBS=ALL’ defined the location of the MIB files for generator script to read them. The generator script was run and it didn’t have any errors as can be seen from the last line of the Image 26. The snmp.yml file was created and copied to the SNMP exporter directory with ‘sudo cp snmp.yml /home/tpamd/snmp_exporter-0.17.0.linux-amd64’ command (Image 26).



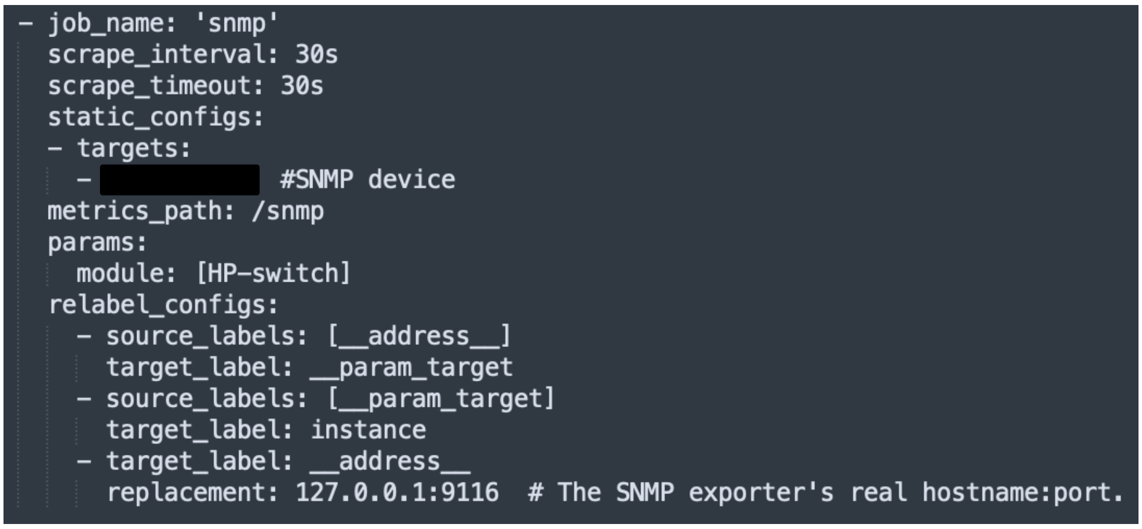
```
[redacted]@monitoring1:~/go/src/github.com/prometheus/snmp_exporter/generator$ export MIBDIRS=/home/[redacted]/go/src/github.com/prometheus/snmp_exporter/generator/mibs
[redacted]@monitoring1:~/go/src/github.com/prometheus/snmp_exporter/generator$ export MIBS=ALL
[redacted]@monitoring1:~/go/src/github.com/prometheus/snmp_exporter/generator$ ./generator generate
level=info ts=2021-12-08T10:24:52.153Z caller=net_snmp.go:144 msg="Loading MIBs" from=/home/[redacted]/go/src/github.com/prometheus/snmp_exporter/generator/mibs
level=info ts=2021-12-08T10:25:21.053Z caller=main.go:52 msg="Generating config for module" module=HP-switch
level=info ts=2021-12-08T10:25:21.187Z caller=main.go:67 msg="Generated metrics" module=HP-switch metrics=51
level=info ts=2021-12-08T10:25:21.197Z caller=main.go:92 msg="Config written" file=/home/[redacted]/go/src/github.com/prometheus/snmp_exporter/generator/snmp.yml
```

Image 26. Creating a snmp.yml file via generator.

Next step was to configure prometheus.yml file to receive SNMP requests from the network switch.

In this example, the job was simply named ‘snmp’ and were given a little bit longer scrape time since network switched can take longer to scrape the assigned metrics. Target is the IP address of the device (Image 27). Module needs to be exactly the same as the one determined in the generator.yml file. If the user has multiple modules, they need to have a unique name otherwise

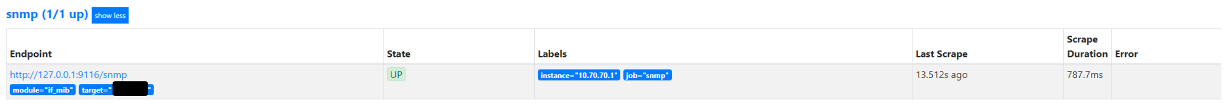
Prometheus cannot determine which module it needs to check and therefore the scrape will fail.



```
- job_name: 'snmp'
  scrape_interval: 30s
  scrape_timeout: 30s
  static_configs:
  - targets:
    - [redacted]  #SNMP device
  metrics_path: /snmp
  params:
    module: [HP-switch]
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: 127.0.0.1:9116  # The SNMP exporter's real hostname:port.
```

Image 27. Prometheus.yml configuration for the network switch.

After the prometheus.yml file configurations, the SNMP and Prometheus server scripts were run to start up the monitoring. Image 28 defines that the 'snmp' job is up and running.



snmp (1/1 up) show less

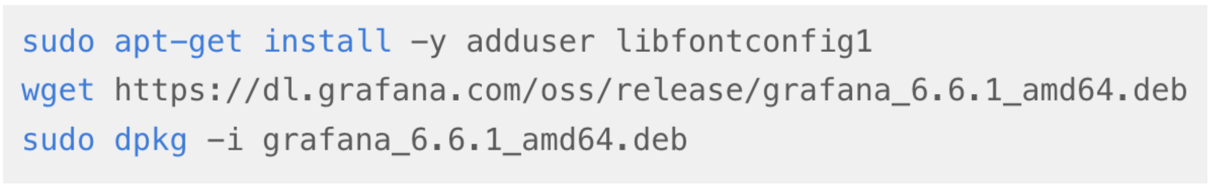
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://127.0.0.1:9116/snmp module="if_mib" target="[redacted]"	UP	instance="10.70.70.1" job="snmp"	13.512s ago	787.7ms	

Image 28. Prometheus' expression browser shows that 'snmp' job is up and running.

4.4 Grafana

After configuring Prometheus server to scrape metrics via node exporter and SNMP exporter, Grana server was ready to be downloaded and installed on the same server as Prometheus.

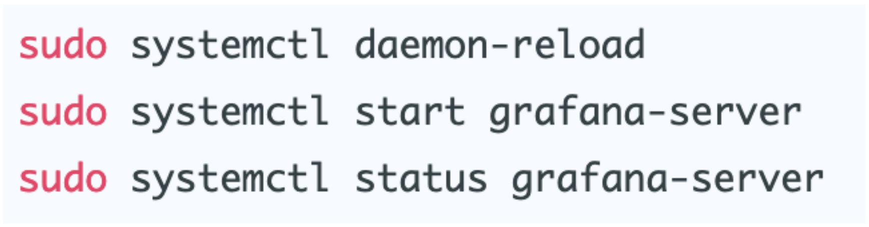
For contrast to all above downloads and installations, the latest Grafana server DEB package from Grafana’s download page was downloaded and installed by following the installation instructions (Grafana Labs 2021e) (Image 29).



```
sudo apt-get install -y adduser libfontconfig1
wget https://dl.grafana.com/oss/release/grafana_6.6.1_amd64.deb
sudo dpkg -i grafana_6.6.1_amd64.deb
```

Image 29. Download and installation of Grafana server. (Screenshot from Grafana’s Downloads page, Grafana Labs 2021e)

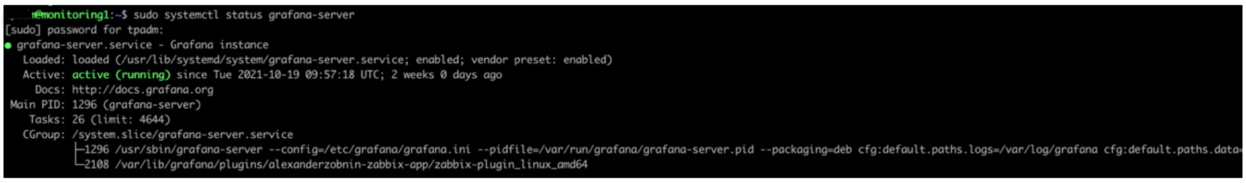
After the installation the Grafana server was started with systemd using the following commands listed in the Image 30:



```
sudo systemctl daemon-reload
sudo systemctl start grafana-server
sudo systemctl status grafana-server
```

Image 30. Grafana startup with systemd. (Screenshot from Grafana’s Downloads page, Grafana Labs 2021i)

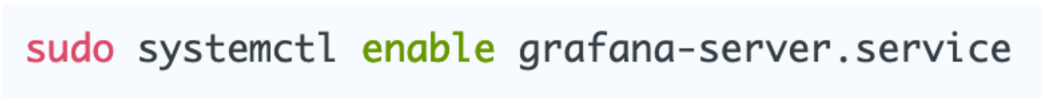
‘sudo systemctl status grafana-server’ command shows if the server is running or not (Image 31).



```
[redacted]n@monitoring1:~$ sudo systemctl status grafana-server
[sudo] password for tpadm:
● grafana-server.service - Grafana instance
   Loaded: loaded (/usr/lib/systemd/system/grafana-server.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2021-10-19 09:57:18 UTC; 2 weeks 0 days ago
     Docs: http://docs.grafana.org
 Main PID: 1296 (grafana-server)
    Tasks: 26 (limit: 4644)
   CGroup: /system.slice/grafana-server.service
           ├─1296 /usr/sbin/grafana-server --config=/etc/grafana/grafana.ini --pidfile=/var/run/grafana/grafana-server.pid --packaging=deb cfg:default.paths.logs=/var/log/grafana cfg:default.paths.data=
           └─2108 /var/lib/grafana/plugins/alexanderzobnin-zabbix-app/zabbix-plugin_linux_amd64
```

Image 31. Example of running Grafana server.

The Grafana server was configured to start at boot (Image 32).

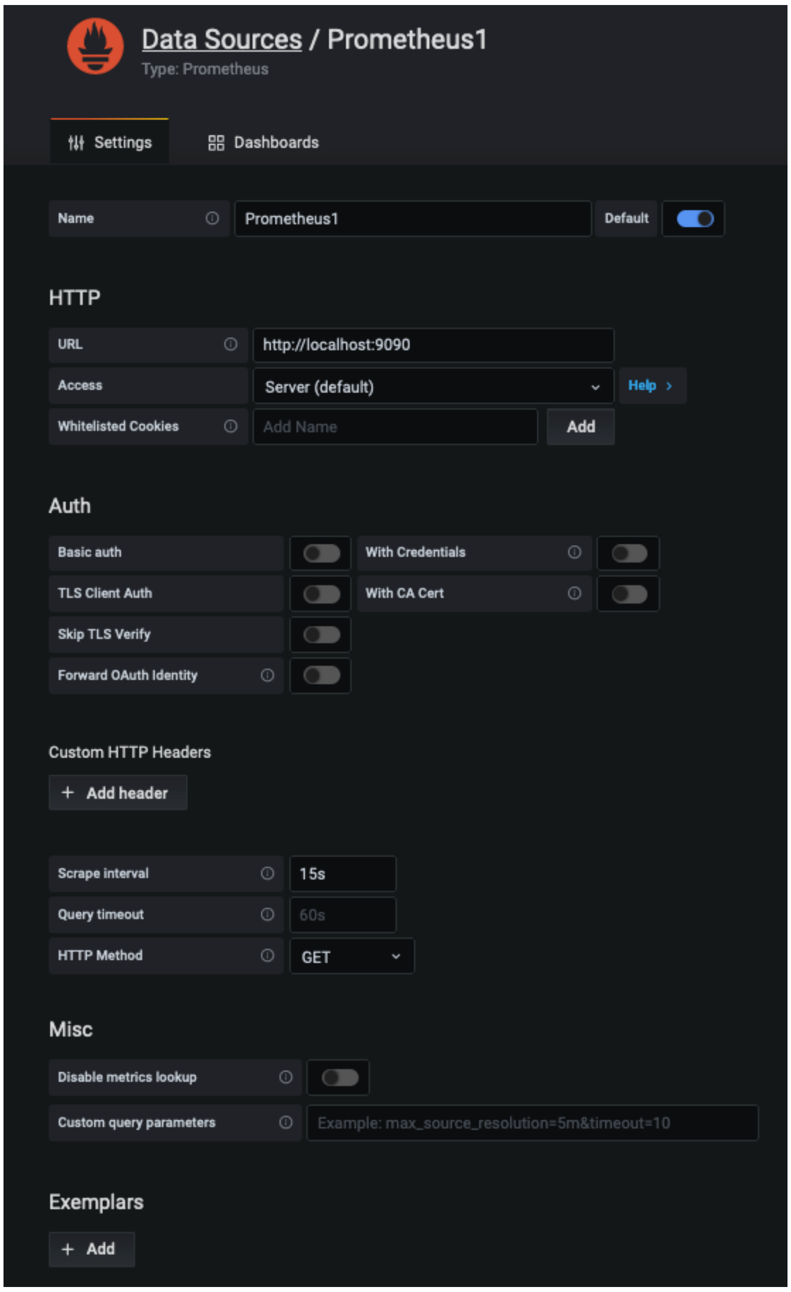


```
sudo systemctl enable grafana-server.service
```

Image 32. Making a service for Grafana to start at boot. (Screenshot from Grafana's Downloads page, Grafana Labs 2021i)

The Grafana server didn't need much configuration in the server. Grafana was accessed through a web browser at http://localhost:3000/. First time visiting the address, the user is asked to change the password for login. The default username and password are 'admin'.

To be able to make dashboards for visual representation of the collected data, a data source needed to be assigned. Data source name and URL were configured to point to the created Prometheus server (Grafana Labs 2021a). Image 33 depicts the data source configuration in Grafana's web UI.



Data Sources / Prometheus1
Type: Prometheus

Settings | Dashboards

Name	Prometheus1	Default	on

HTTP

URL	http://localhost:9090
Access	Server (default)
Whitelisted Cookies	Add Name

Auth

Basic auth	off	With Credentials	off
TLS Client Auth	off	With CA Cert	off
Skip TLS Verify	off		
Forward OAuth Identity	off		

Custom HTTP Headers

+ Add header

Scrape interval	15s
Query timeout	60s
HTTP Method	GET

Misc

Disable metrics lookup	off
Custom query parameters	Example: max_source_resolution=5m&timeout=10

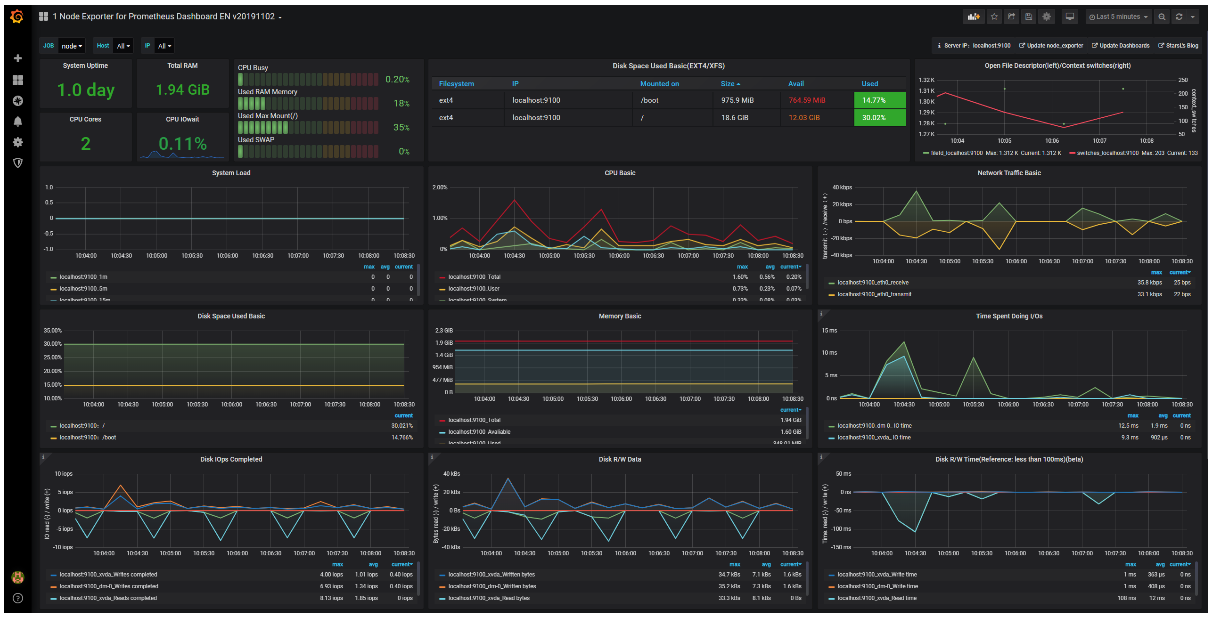
Exemplars

+ Add

Image 33. Data source configuration page.

4.5 Node exporter dashboard

A premade dashboard, that somebody else had made to be used with node exporter, was imported from Grafana's dashboard download page (Grafana Labs 2021f). The dashboard had all the vital information about Unix environment, and it was a good starting place to getting to know how Grafana and Prometheus work together before trying to make new dashboards from scratch Image 34).



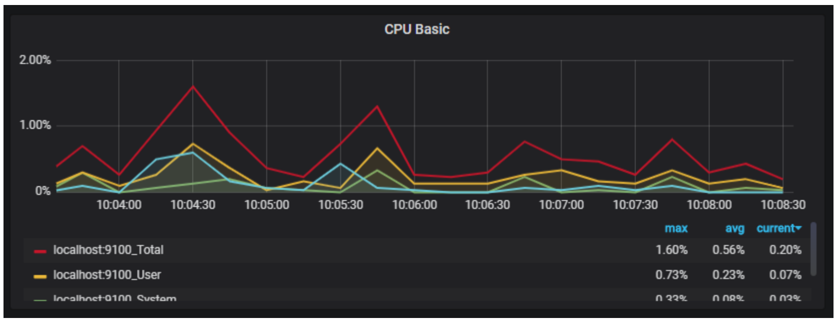
Disk Space Used Basic(EXT4/XFS)

Filesystem	IP	Mounted on	Size	Avail	Used
ext4	localhost:9100	/boot	975.9 MiB	764.59 MiB	14.77%
ext4	localhost:9100	/	18.6 GiB	12.03 GiB	30.02%

Image 34. Imported node exporter dashboard with different panels.

The imported dashboard gave a good idea of how dashboards and panels work. What makes this a good dashboard, is that the panels state clearly what they are used for, and they fit into the screen without having to scroll down the page. From the above Image 34 information about the monitored system can be seen. For example, how much memory and storage the system has and how much is used and how much is still available. The network traffic that system is receiving and transmitting can also be monitored with node exporter.

For example in the Image 35, the CPU Basic panel is telling how much the system is using the processor cores in each state. Total is all the works minus the idle state, user means the time spent in user's processes and system is the time spent in kernel mode.



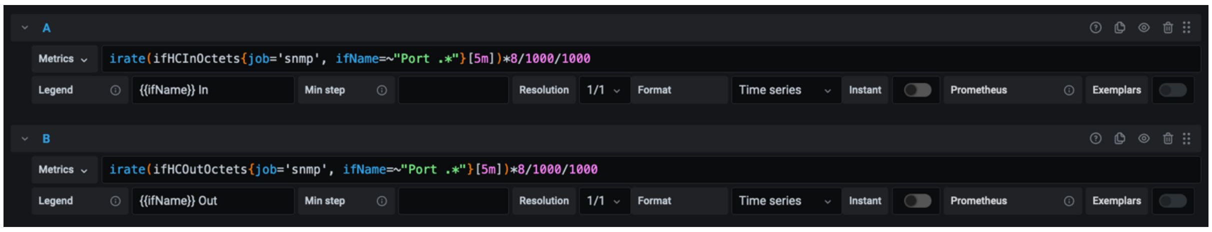
	max	avg	current
localhost:9100_Total	1.60%	0.56%	0.20%
localhost:9100_User	0.73%	0.23%	0.07%
localhost:9100_System	0.33%	0.08%	0.03%

Image 35. Panel about processor usage in percentage.

4.6 SNMP exporter dashboard

For SNMP exporter target, the network switch, an entirely new dashboard was created. The wanted information for the network switch were network traffic, system up time and up panels. The panels work by implementing a PromQL query about wanted information.

Below, Image 36 represents the queries used in defining the incoming and outgoing network traffic from the switch. IfHCInOctets and IfHCOutOctets return the total number of octets received or transmitted through the switch's interfaces in 64-bit form. 'job' and 'ifName' labels were used to define what the queries are looking for. Irate function with [5m] returns the per-second rate of the two most recent data points per time series within the five-minute time range. A little math was also needed, because the results wanted to be displayed in megabits, so the query had to be multiplyed by 8 because of octets and to get megabits the query needed to be divided by 1000 and then again by 1000. (Prometheus Authors 2014-2021h)


A

Metrics
```
irate(ifHCInOctets{job='snmp', ifName=~"Port .*"}[5m])*8/1000/1000
```
Legend {{ifName}} In Min step Resolution 1/1 Format Time series Instant Prometheus Exemplars

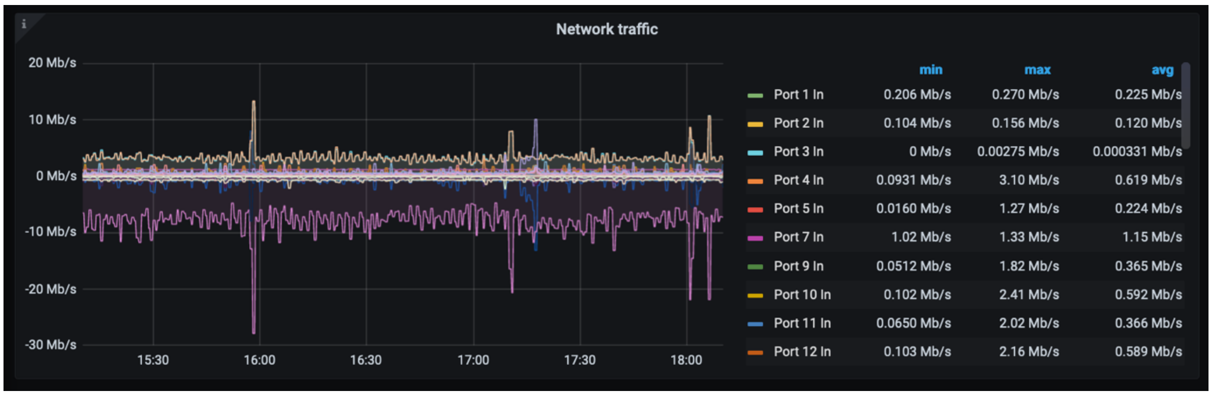
B

Metrics
```
irate(ifHCOutOctets{job='snmp', ifName=~"Port .*"}[5m])*8/1000/1000
```
Legend {{ifName}} Out Min step Resolution 1/1 Format Time series Instant Prometheus Exemplars

Image 36. PromQL query about incoming and outgoing traffic.

The below graph in the Image 37 displays the incoming and outgoing traffic of a network switch from the last three hours. On the right side of the panel are all the switch's interfaces and their minimum, maximum and average values.

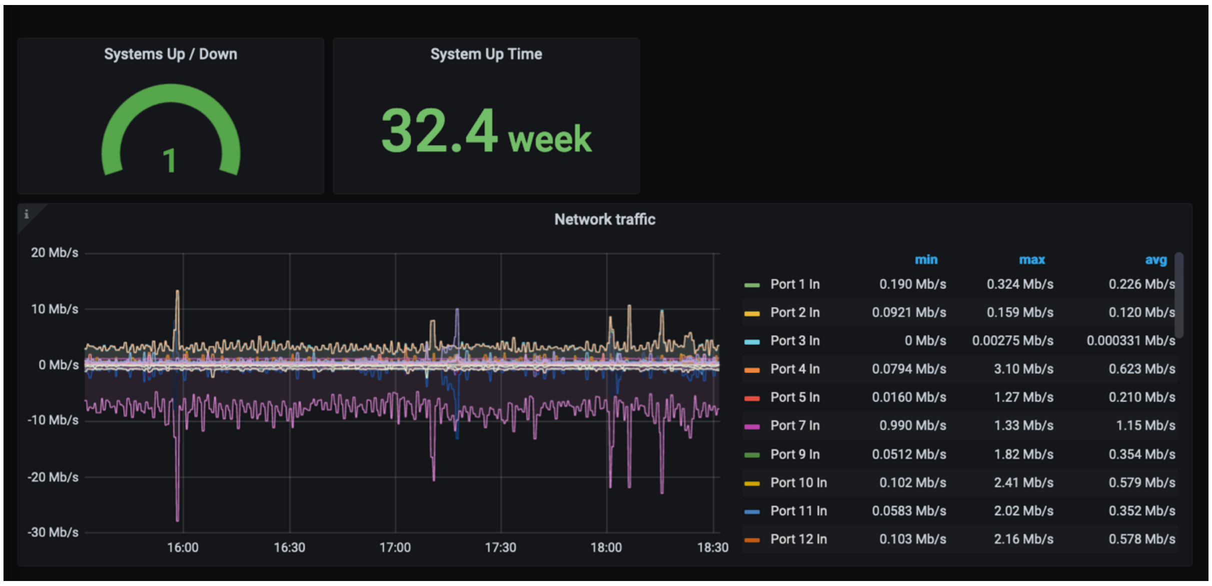
The end result looked like this:



	min	max	avg
🟩 Port 1 In	0.206 Mb/s	0.270 Mb/s	0.225 Mb/s
🟨 Port 2 In	0.104 Mb/s	0.156 Mb/s	0.120 Mb/s
🟦 Port 3 In	0 Mb/s	0.00275 Mb/s	0.000331 Mb/s
🟧 Port 4 In	0.0931 Mb/s	3.10 Mb/s	0.619 Mb/s
🟥 Port 5 In	0.0160 Mb/s	1.27 Mb/s	0.224 Mb/s
🟪 Port 7 In	1.02 Mb/s	1.33 Mb/s	1.15 Mb/s
🟩 Port 9 In	0.0512 Mb/s	1.82 Mb/s	0.365 Mb/s
🟨 Port 10 In	0.102 Mb/s	2.41 Mb/s	0.592 Mb/s
🟦 Port 11 In	0.0650 Mb/s	2.02 Mb/s	0.366 Mb/s
🟧 Port 12 In	0.103 Mb/s	2.16 Mb/s	0.589 Mb/s

Image 37. The result of the network traffic monitoring query.

'Up' and 'System up time' panels were added to the dashboard as can be seen on the Image 38. The 'System Up / Down' panel is displayed in 'gauge' form and 'System up time' panel is shown as 'stats'. 'Graph' form is used in the network traffic panel. The panels are named clearly and informatively. The 'System Up / Down' panel shows as a green gauge when the device is 'up' or the value is 1. When the value is 0 the gauge turns red indicating the device is down.



	min	max	avg
🟩 Port 1 In	0.190 Mb/s	0.324 Mb/s	0.226 Mb/s
🟨 Port 2 In	0.0921 Mb/s	0.159 Mb/s	0.120 Mb/s
🟦 Port 3 In	0 Mb/s	0.00275 Mb/s	0.000331 Mb/s
🟧 Port 4 In	0.0794 Mb/s	3.10 Mb/s	0.623 Mb/s
🟥 Port 5 In	0.0160 Mb/s	1.27 Mb/s	0.210 Mb/s
🟪 Port 7 In	0.990 Mb/s	1.33 Mb/s	1.15 Mb/s
🟩 Port 9 In	0.0512 Mb/s	1.82 Mb/s	0.354 Mb/s
🟨 Port 10 In	0.102 Mb/s	2.41 Mb/s	0.579 Mb/s
🟦 Port 11 In	0.0583 Mb/s	2.02 Mb/s	0.352 Mb/s
🟧 Port 12 In	0.103 Mb/s	2.16 Mb/s	0.578 Mb/s

Image 38. The network switch dashboard.

5 Conclusion

Tietopartio already had a vision about a monitoring system that could be put on display in their office environment and what could be shown to customers. The work carried out in this thesis was the realization of this vision.

Since the client knew exactly what systems they wanted to be used in the project, this thesis focused on comparing the suitable options to carry out the project. The client had previously implemented two other monitoring systems to their environment. Therefore, it was obvious that the next step was to compare the old systems to the new ones to gain a better understanding of why there was a need for additional monitoring systems.

The objective of this thesis was the installation, the configuration and the testing of the new visual monitoring system Grafana and the new data monitoring tool Prometheus. These objectives were met, and both systems continued to be used in Tietopartio’s environment. Data was successfully collected from the targets, therefore, demonstrating that the installation and configuration of the systems were successful. The visual representations of the data were shown in clearly constructed dashboards and panels. Information of the targets’ performance could be viewed and analyzed through the dasboards. With the setup done and with the gained knowledge of the monitoring systems, new targets can be added to the monitoring domain with ease.

5.1 Challenges

Working with Prometheus was a bit challenging since Prometheus had a quite steep learning curve. There is so much that the user can do with Prometheus. A first time user needs some time to study to fully understand and grasp all the information of the capabilities of the monitoring tool. I had some difficulties in Using the PromQL was challenging as it requires mathematical thinking and mastery of its features.

SNMP was also a new protocol and this thesis provided an opportunity for the author to gain better understanding and experience of MIBs and OIDs. MIB Browswer and ‘snmpwalk’ method were useful tools in the learning process of understanding how SNMP works.

5.2 Future improvements and developments

The idea was that Tietopartio would install monitors to their office walls to display the monitored data so that the IT specialists could see, for example how much free space a specific server has, and react to it, before another monitoring system would send an alert about disk space being low. The next logical step would be implementing the monitoring system to all Tietopartio’s devices in the office and data center.

Setting up alerts either to Prometheus or Grafana would also be beneficial for the overall monitoring purpose. Alerting needs would first be assessed since there already are two alert systems on the same devices.

Implementing proper security measurements would be crucial to have at some point and is definitely an important improvement step.

Joining Zabbix and LibreNMS to Grafana or deactivating them completely and relaying only on Prometheus as the means of data collecting would also be taken into consideration. It is also important to determine if there is a need for three different monitoring tools and whether they all should be monitoring the same targets.

Lastly, when the monitoring service is made clear and safe, it could also be deployed to Tietopartio’s customers environments.

References

Bradley, S. 2020-2021. SBCODE. SNMP Exporter Configuration Generator. Accessed 30.11.2021. https://sbcode.net/prometheus/snmp-exporter-generator/

Brazil, B.; Robust Perception Ltd. 2018. Prometheus: Up & Running - Infrastructure and application performance monitoring. 1005 Gravenstein Highway North, Sebastopol, CA 95472. O'Reilly Media, Inc.

Github Inc. 2021a. Node exporter. Accessed 30.11.2021. https://github.com/prometheus/node_exporter

Github Inc. 2021b. Prometheus SNMP exporter. Accessed 30.11.2021. https://github.com/prometheus/snmp_exporter

Github Inc. 2021c. SNMP Exporter Config Generator. Accessed 30.11.2021. https://github.com/prometheus/snmp_exporter/tree/main/generator

Github Inc. 2021d. SNMP Config Generator. Accessed 30.11.2021. https://pkg.go.dev/github.com/prometheus/snmp_exporter/generator#section-readme

Grafana Labs 2021a. Add a data source. Accessed 30.11.2021. https://grafana.com/docs/grafana/latest/datasources/add-a-data-source/

Grafana Labs 2021b. Add a data source. Accessed 30.11.2021. https://grafana.com/docs/grafana/latest/datasources/

Grafana Labs 2021c. Dashboards. Google search result. Accessed 30.11.2021. https://grafana.com/static/img/grafana/showcase_visualize-954.jpg

Grafana Labs 2021d. Dashboard overview. Accessed 30.11.2021. https://grafana.com/docs/grafana/latest/dashboards/

Grafana Labs 2021e. Download Grafana. Accessed 30.11.2021. https://grafana.com/grafana/download

Grafana Labs 2021f. Export and import. Accessed 30.11.2021. https://grafana.com/docs/grafana/latest/dashboards/export-import/

Grafana Labs 2021g. Grafana tutorials. Accessed 30.11.2021. https://grafana.com/tutorials/

Grafana Labs 2021h. Image rendering. Accessed 30.11.2021. https://grafana.com/docs/grafana/latest/Image-rendering/

Grafana Labs 2021i. Install on Debian or Ubuntu. Accessed 30.11.2021. https://grafana.com/docs/grafana/latest/installation/debian/

Grafana Labs 2021j. Introduction to Grafana. Accessed 30.11.2021 https://grafana.com/docs/grafana/latest/introduction/

Grafana Labs 2021k. Panel overview. Accessed 30.11.2021. https://grafana.com/docs/grafana/latest/panels/

LibreNMS Docs a. Dashboards. Accessed 30.11.2021. https://docs.librenms.org/Extensions/Dashboards/

LibreNMS Docs b. Distributed Poller. Accessed 30.11.2021. https://docs.librenms.org/Extensions/Distributed-Poller/

LibreNMS Docs c. Features. Accessed 30.11.2021. https://docs.librenms.org/Support/Features/

LibreNMS Docs d. Metric storage. Accessed 30.11.2021. https://docs.librenms.org/Extensions/Metric-Storage/

Prasath, M. 2018. SNMP monitoring and easing it with Prometheus. Article. Accessed 30.11.2021. https://medium.com/@openmohan/snmp-monitoring-and-easing-it-with-prometheus-b157c0a42c0c

Prometheus Authors 2014-2021a. Client libraries. Accessed 30.11.2021. https://prometheus.io/docs/instrumenting/clientlibs/

Prometheus Authors 2014-2021b. Comparison to alternatives. Accessed 30.11.2021. https://prometheus.io/docs/introduction/comparison/

Prometheus Authors 2014-2021c. Exporters and integrations. Accessed 30.11.2021. https://prometheus.io/docs/instrumenting/exporters/

Prometheus Authors 2014-2021d. First steps with Prometheus. Accessed 30.11.2021. https://prometheus.io/docs/introduction/first_steps/

Prometheus Authors 2014-2021e. Getting started. Accessed 30.11.2021. https://prometheus.io/docs/prometheus/latest/getting_started/

Prometheus Authors 2014-2021f. Monitoring Linux host metrics with the node exporter. Accessed 30.11.2021. https://prometheus.io/docs/guides/node-exporter/

Prometheus Authors 2014-2021g. Overview. Accessed 30.11.2021. https://prometheus.io/docs/introduction/overview/

Prometheus Authors 2014-2021h. Querying Prometheus. Accessed 30.11.2021. https://prometheus.io/docs/prometheus/latest/querying/basics/

Prometheus Authors 2014-2021i. Storage. Accessed 30.11.2021. https://prometheus.io/docs/prometheus/latest/storage/

Ubuntu packages 2021. Linux Image generic. Accessed 30.11.2021. https://packages.ubuntu.com/bionic/linux-Image-generic

Wikipedia – The free encyclopedia 2021. Ubuntu. Wiki platform. Accessed 30.11.2021. https://en.wikipedia.org/wiki/Ubuntu

Zabbix SIA 2001-2021a. 1 Configuring a host. Accessed 30.11.2021. https://www.zabbix.com/documentation/current/manual/config/hosts/host

Zabbix SIA 2001-2021b. 1 Dashboard. Accessed 30.11.2021. https://www.zabbix.com/documentation/current/en/manual/web_interface/frontend_sections/monitoring/dashboard

Zabbix SIA 2001-2021c. 2 Agent. Accessed 30.11.2021. https://www.zabbix.com/documentation/current/manual/concepts/agent

Zabbix SIA 2001-2021d. 2 What is Zabbix. Accessed 30.11.2021. https://www.zabbix.com/documentation/5.2/en/manual/introduction/about

Zabbix SIA 2001-2021e. 3 Components. Accessed 30.11.2021. https://www.zabbix.com/documentation/1.8/manual/installation/components

Zabbix SIA 2001-2021f. 3 Zabbix features. Accessed 30.11.2021. https://www.zabbix.com/documentation/5.2/en/manual/introduction/features

Zabbix SIA 2001-2021g. 4 History and trends. Accessed 30.11.2021. https://www.zabbix.com/documentation/current/manual/config/items/history_and_trends

Zabbix SIA 2001-2021h. 4 Zabbix overview. Accessed 30.11.2021. https://www.zabbix.com/documentation/5.2/en/manual/introduction/overview

Zabbix LLC 2001-2021i. Explore Zabbix features. Accessed 30.11.2021. https://www.zabbix.com/features#data_customization