Network and Systems Monitoring with Prometheus and Grafana

João Sanches[1] and Paulo Rogério Pereira[2][0000-0002-0192-2504]

[1] Instituto Superior Técnico, Universidade de Lisboa, Portugal
[2] INESC INOV, Instituto Superior Técnico, Universidade de Lisboa, Portugal
Paulo.Pereira@tecnico.ulisboa.pt

Abstract. This paper describes the implementation of a monitoring system aimed for universities, enterprises or on prem hosting services. We survey the most relevant time series database systems and visualization tools available and selected a Prometheus time series database to collect the metrics from the equipment and to serve them to Grafana, a visualization tool that allows to build dashboards for a holistic observability. Then we integrated Prophet, a machine learning tool able to detect pattern anomalies in the collected data and trigger warnings automatically. We evaluated the system for performance and scalability and concluded it is very lightweight in resource consumption which means it could scale to large networks.

Keywords: Network Management, Time series database, Observability, Metric Scrapping, Machine Learning.

1 Introduction

Companies nowadays are struggling with the maintenance of their services since IT infrastructures and getting larger and more load demanding with a fast and reliable accessibility and usability for the user being a concern. The Agile methodologies [1] implemented in companies right now, like Scrum and Kanban, enforce quickly and dynamic deployments which sometimes can lead to small bugs that have implications on the system's reliability. So, it is important to have monitoring (processes to collect system's metrics) and observability methods (process of analysing the collected information) in place in order to quickly notice errors that may be happening and act fast to fix them.

Having this in mind, this paper presents a study about network and systems management and performance solutions that could be used to achieve that goal. This paper presents the following contributions:

First, we assess the state of the art on the available software to build such systems, specially open-source. Concepts are presented in section 2 and solutions in section 3.

A second goal is to establish a monitoring system on three university labs, sharing the same subnet, at our university's computer department. We should be able to collect data about any of the computers connected inside each lab, as well as the ethernet

switches that route traffic inside the network. The corresponding architecture is presented in section 4 and its evaluation in section 5.

Concomitantly, we would like to understand if it would be possible to implement machine learning on this system to automatically detect anomalies in the collected values.

2 Network Management

The Simple Network Management Protocol (SNMP) [2] is an Internet standard protocol used for manager applications to collect and modify management data from agents in network devices such as routers, switches, modems, servers, hubs, workstations, even printers or cameras, and so on. Devices store management information in a Management Information Base (MIB) which is an object database installed in the managed devices that SNMP uses to store and expose the management data. The MIBs are defined using the Structure of Management Information (SMI), which defines the MIB's format (i.e. the data types that are used and the information transfer rules), in a machine-readable format, resorting to ASN.1 macros. The MIB is a hierarchical tree structure where each object has an Object Identifier (OID). The OIDs are standardized, and vendors are allowed to have their own MIB definitions to suit each specific device. There are MIBs [3] for information and statistics about devices, its interfaces, protocols and resources.

3 Available Solutions

3.1 Cacti

Cacti [4] is an open-source LAMP (Linux, Apache as server, MySQL as RDBM, and PHP, Pearl or Python as scripting language) network monitoring and graphing tool designed to provide a visual representation of various network-related metrics. It is particularly well-suited for monitoring and graphing the performance of network devices, servers, and other infrastructure components. Cacti mainly uses SNMP as its data collection method.

Cacti employs a polling mechanism to collect data at defined intervals, which is then stored in a database, typically using RRDtool (Round Robin Database tool). RRDtool is a powerful tool for efficiently storing time-series data and creating graphs. RRDtool implements a Round Robin Archive which is a fixed sized data structure, where, when full, the oldest data element of the structure will be deleted to make space for the newest sample. Stored data is consolidated using average, maximum or minimum mathematical functions. The application of both these techniques allows having a data history while simultaneously guaranteeing that the size of the database does not increase indefinitely.

The application is handled through a web interface served by an Apache HTTP Server and that interface allows users to configure, manage, and view graphs in a

user-friendly way given the access to various configuration options, templates, and graph management. It also supports setting up triggers and alerts.

Cacti benefits from a strong community of users and developers in the form of forums, documentation, and a large community library with contributions that enhance the tool's capabilities, like templates for the desired devices, and plug-ins for the Cacti application.

3.2 Prometheus

Prometheus [5][6] is an open-source monitoring tool originated at SoundCloud, a music streaming service, in 2012, to address the specific monitoring and alerting needs of SoundCloud: they needed a modern monitoring system that could adapt to its microservices architecture, dynamic environment, and containerized infrastructure. The project was initiated by a team led by ex-Google engineers who were experienced in dealing with large-scale systems who drew inspiration from Google's internal monitoring tools, especially Borgmon.

Prometheus was designed with several key principles in mind, including a dimensional data model with labels, a flexible query language (PromQL), a time series database, reliability, and scalability.

An overview of the Prometheus architecture can be seen in Fig. 1.

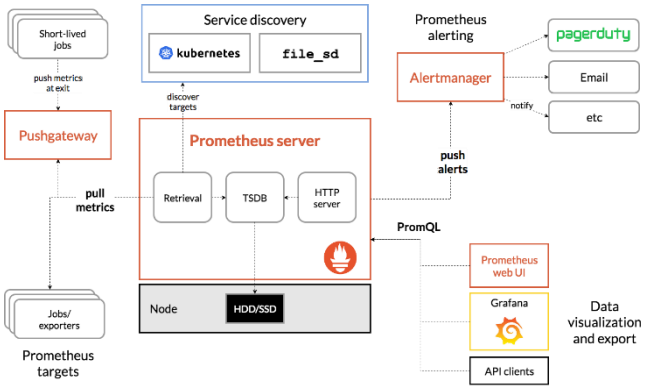

Fig. 1. Prometheus Architecture [6].

The core component of Prometheus is the Prometheus server. It is mainly made of an HTTP server that allows Prometheus to expose its web-based user interface, metrics, and other functionalities, providing an interface for querying, managing, and monitoring Prometheus itself and responsible for data collection and storage. It also contains an embedded Time-Series Database (TSDB), that stores the collected metrics and compresses time-series data, allowing for efficient querying and analysis.

Prometheus supports service discovery mechanisms to dynamically discover and monitor targets. Common service discovery methods include static configuration, DNS-based discovery, file-based discovery, and integrations with orchestration platforms like Kubernetes.

The data is collected from targets which can be applications, services, or infrastructure components. The tool supports various types of exporters that act as agents to

scrape and expose metrics from different systems. Examples include the Node Exporter for system-level metrics and the Blackbox Exporter for probing endpoints. The Prometheus server periodically scrapes metrics from the configured targets, which expose a *metrics* endpoint that provides metric data in a text-based format.

Metrics collected by Prometheus have a dimensional data model. Each metric is identified by a unique name and a set of key-value pairs called labels, which allow for a multi-dimensional representation of data.

The Pushgateway is a component that allows short-lived jobs or batch processes to expose their metrics to Prometheus. In the typical Prometheus pull model, the Prometheus server scrapes metrics from configured targets. However, in some scenarios where jobs are short-lived or not directly reachable by Prometheus, a different approach is needed. This is where the Pushgateway comes into play. Instead of pulling metrics from targets, jobs or processes push their metrics to the Pushgateway. Metrics pushed to the Pushgateway include labels for job and instance that help to identify and differentiate the metrics coming from different sources. Metrics pushed to the Pushgateway are associated with a specific expiration time, and after that time, they are considered stale and will not be scraped by Prometheus.

Prometheus provides a query language called PromQL for querying and analysing metrics data. PromQL allows users to aggregate, filter, and transform time series data. It supports a wide range of functions for complex queries.

There are also alerting features. Users are allowed to define alerting rules, specifying conditions that, when met, trigger alerts, using its configuration file. The alerting system is built into Prometheus, and it can send alerts to external systems via the Alertmanager. The Alertmanager is a separate component that handles alerts sent by Prometheus and that can also perform additional actions, such as grouping, deduplicating, and routing alerts to the appropriate receivers: it supports various notification channels like email, Slack, and others.

Prometheus does not have a built-in graphical user interface for creating graphs and visualization, so it is often used in conjunction with other visualization tools with Grafana [10] being the most popular open-source tool used. Grafana can connect to Prometheus as a data source (Fig. 1), allowing users to create customized dashboards.

In terms of the network architecture, Prometheus supports federation, allowing multiple Prometheus servers to be linked together. This is useful for scalability and high availability. Federated Prometheus servers can scrape metrics from each other and aggregate data.

In matters of storage, there is also a feature that supports remote write and remote read capabilities, allowing the integration with remote storage systems, useful for long-term storage and analysis of metrics data.

3.3 Grafana

Grafana [10] is an open-source platform for monitoring and observability that allows users to query, visualize, alert on, and understand metrics. It integrates with a variety of data sources, including time-series databases, logging systems, and other data platforms, making it a versatile tool for creating interactive and customizable dashboards to view data. Grafana supports a wide range of data sources, including Prometheus,

InfluxDB, Graphite, Elasticsearch, MySQL, PostgreSQL, and many others. That is due to the support of different query languages depending on the data source (for example, PromQL for Prometheus, but SQL in the case of relational databases, etc). This flexibility allows users to consolidate data from various sources into a single dashboard.

To connect Grafana to their desired data sources, users can use the Grafana's web interface to configure the connections, where the connection details, authentication credentials, and other settings specific to each data source should be provided.

Grafana provides a wide range of visualization options, including line charts, bar graphs, tables, heatmaps, and more. Users can choose the visualization type that best represents their data. Customizable options include axis labels, legends, and color schemes. The graphs can be organized in dashboards to visualize and analyse data. Dashboards are composed of panels, each representing a different type of visualization or data source query and can be customized to suit the monitoring needs. It is even possible to recur to templating for creating reusable and parametrized dashboards, even allowing users to create dynamic dashboards with variables. Annotations such as marking events, outages or any other relevant information can be added on the dashboards and provide a way to add contextual information.

3.4 Available Solutions Comparison

We have studied the following software: Cacti [4], Prometheus [5], InfluxDB [7], Nagios [8] and Zabbix [9]. They can be divided in two major categories: while InfluxDB and Prometheus are monitoring applications (their core component is a Time-Series Database Engine which fetches, processes and stores nodes' data), Cacti, Nagios and Zabbix are monitoring and presentation applications, that not only collect and store data as well (they have databases too), but also have user interfaces to manage the monitored systems and present that data graphically to end users. Prometheus and InfluxDB are often paired up with visualization tools such as Grafana, hence constituting a monitor and presentation system as well.

For a fair comparison between Prometheus and InfluxDB, we must consider Kapacitor as well, which is a data process engine from InfluxData and that endows InfluxDB with some features that Prometheus has, fundamentally regarding data processing and alerting. Prometheus and InfluxDB use a similar data model both using key-value pairs as labels (called tags on InfluxDB) appended to the actual data values for indexing purposes. Regarding storage, InfluxDB uses a Time-Structured Merge Tree (TSM) to store its data, and Prometheus an append-only file per time series, but both have retention policies to better manage storage space and keep historical data. InfluxDB supports higher data resolution, up to nanoseconds while Prometheus only supports resolutions up to milliseconds. Both Prometheus and InfluxDB run servers independently, relying only on local storage for engine tasks like data scraping, rule processing and alerting. In matters of redundancy and high availability, Prometheus offers strategies like federation or using external components for distributed setups, which are possible but harder to setup on InfluxDB. In summary, Prometheus is tailored for monitoring and alerting and InfluxDB for event logging.

Now we will compare Cacti, Nagios and Zabbix. As mentioned previously, they are software tools to monitor networks and systems. They all rely on SNMP for data gathering but they also can be extended through plugins to use other protocols. Cacti stores monitoring data on RRD files while Nagios resorts to text files (there are plugins to use RRD as well) and Zabbix to external relational databases such as MySQL or PostgreSQL. Regarding their alerting capabilities, Cacti has only basic threshold-based alerts (although plugins can extend those capabilities), while the others possess more complex capabilities, allowing to define more advanced conditions to trigger alerts. The visualization part is taken care of by RRDTool graphing capabilities on Cacti, and Nagios needs plugins to be able to draw graphs, while Zabbix has its own internal graphing features. On scalability, neither Cacti nor Nagios (on its free tier) are easy to setup on a multi-NMS distributed architecture, but on the other hand Zabbix has native support for that.

From all the open source choices seen previously, Prometheus is the one that provides more flexibility while being entirely open-source: it is a more recent solution when compared with Cacti, Nagios or Zabbix, provides HTTP data pull from sources and can also offer SNMP pulling, and has a built-in time series database designed for metrics collecting and storing rather than the InfluxDB's one which is more suited for event logging (collect data that is not well represented with key-value pairs).

3.5 Data Forecasting Tools

Collected data in the time-series database can also be analysed to predict evolution or detect problems.

Facebook's Prophet [15] is a tool implemented in R and Python for forecasting time-series data [11]. It simplifies the process of time series forecasting by providing a user-friendly interface and powerful modelling capabilities. It runs on a model that automatically detects patterns and outliers in the data, allowing users to focus on interpreting the results rather than fine-tuning model parameters, even providing intuitive methods for visualizing forecasts and evaluating model performance, making it accessible to users with varying levels of expertise in time series analysis. Prophet has been gaining popularity among data scientists, analysts, and researchers seeking reliable forecasting solutions for a wide range of applications, from sales forecasting and demand planning to resource allocation and capacity planning.

Grafana Labs also developed a predictive time series model for Grafana called Grafana Machine Learning [12]. It is designed to help users extract valuable insights, identify patterns, and make data-driven decisions by leveraging machine learning algorithms. Grafana ML enables users to explore and analyse time series data to uncover trends, anomalies, and correlations. Users can leverage forecasting algorithms to predict future values based on historical data, enabling them to anticipate trends and plan accordingly and to automatically identify unusual patterns or outliers in time series data, helping in anomalies detection. Those insights and predictions can be directly visualized using Grafana's dashboarding capabilities, allowing users to gain actionable insights at a glance. The main drawback is that this feature is only available for Pro users (paid subscription).

Weka (Waikato Environment for Knowledge Analysis) [13] is a collection of open-source machine learning algorithms and data analysis tools developed by the University of Waikato in New Zealand. It has tools for all kinds of data analytics purposes: data preparation, classification, regression, clustering, association rules mining, and visualization. Weka is implemented in Java which makes it a portable software capable of running on practically any platform.

Other Python tools could be considered such as the ones mentioned in [14] with them being, besides Prophet, Pmdarima, Ludwig, DeepAR, TFT, FEDOT, AutoTs and Sktime, which are all freely available Python libraries. While comparing Prophet with Weka and FEDOT we come to the conclusion that Prophet is easier to use and offers a simpler interface for the given goal that we want to achieve; even though its prediction time is the worse when comparing with the other mentioned tools, its milliseconds prediction time mark is acceptable for the desired task of sending alarms, where the training time being lower is more relevant.

4 Implementation

4.1 Network Architecture to be Monitored

The network topology is depicted in Fig. 2. Connection to the outside internet is done through a gateway built on a computer with ClearOS, which does address translation and applies firewall policies to the inbound and outbound traffic. The gateway is connected to one of the four existing ethernet switches, which is connected to the other three. Laboratory LSD1 has one unmanaged TP-Link TL-SG1016 switch connecting the computers within. Laboratories LSD2 and LSD3 have managed D-Link switches model DGS-1210-24 and in the printer room there is a managed D-Link switch model DGS-1520-28. Managed switches offer security and monitoring features through SNMP, while the unmanaged do not. The managed switches can be added to the pool of nodes to be monitored.

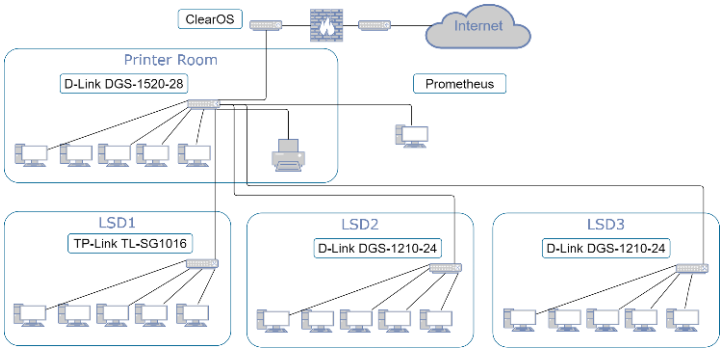

Fig. 2. Laboratories network topology to be monitored.

After analysing this network topology, we decided we would like to gather information regarding all devices on it, either computers, the managed switches and even

the gateway. From the computers, including the gateway, we want to extract operative system's metrics regarding uptime, CPU, memory, storage, network traffic, etc., while from the switches we want to have traffic information, since the managed ones offer this feature through SNMP and RMON.

4.2 Monitoring Solution Architecture

The monitoring solution, that seemed more suited, was to have a dedicated computer, connected to the same network as the labs, running Prometheus for data gathering and Grafana to display the gathered data. To extract OS information from the network nodes, we used the available Node Exporters from Prometheus GitHub page [17]. They can be installed as a service either on Windows or Linux and what they do is translate the OS performance information to Prometheus data model and make it available through HTTPS on a specific port. Then the Prometheus agent will only need to scrape data from that port and store the data on its database. To cover SNMP devices, like switches, we will need to use another Prometheus component, which is the SNMP exporter: it acts as a proxy between Prometheus monitoring node and the SNMP monitored nodes, so it receives HTTPS requests from Prometheus monitoring agent to scrape a given host and a given tree and translates the SNMP tree to Prometheus data model so that it can be promptly stored on the database.

The presented solution is depicted on Fig. 3. We have a central node running both Prometheus and Grafana. Prometheus' role is to extract data from the system and store it and for that it relies on a scrapping agent that is pointed towards the nodes in the network which sends periodic HTTPS requests to collect the desired performance information from their Node Exporters. A Node Exporter was installed in every monitored host as a `systemctl` service. Note that the monitoring node is also running a Node Exporter: it is being used to self-monitor the Prometheus machine. The SNMP Exporter receives, from the core agent, HTTPS requests containing the address of the monitored device and the MIB tree objects that we want to query as parameters, performs the SNMP query to the device, translates the tree information and answers to the agent with information ready to store. Finally, Grafana is periodically consuming data from the Prometheus API to present it, updated, on the dashboards that were set up by the users.

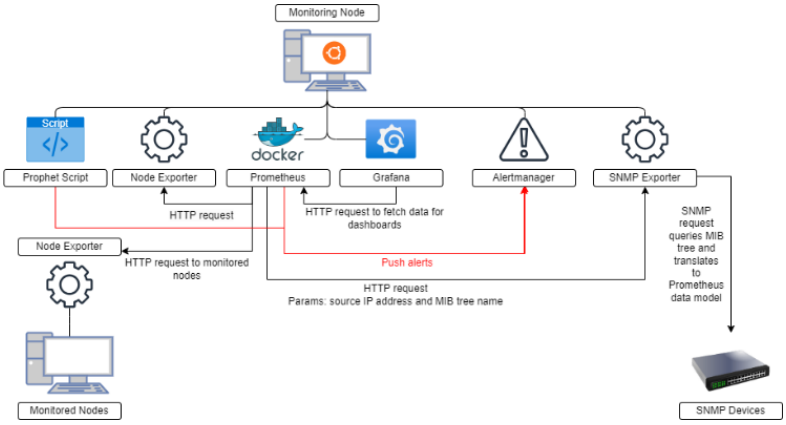

Fig. 3. Monitoring solution architecture.

If we wanted to further expand monitoring to a larger scale network, we could use the federation feature from Prometheus. There are two ways to use Prometheus federation, one is in a hierarchical set-up and the other is cross-service federation. For our use case, we would like to apply hierarchical federation since our goals are scalability and data aggregation. The main idea behind hierarchical federation is to have a central general Prometheus server collecting aggregated metrics, lacking some detail, from child servers that would be collecting data directly from subsets of devices or a cluster. In this way, we could have an aggregated view on the parent server, and we could still drill down locally on the child servers, if needed. Thanos [16] may assist in this, but this was left for future work.

4.3 Anomalies Detection with Prophet

For our specific case we are not so interested in doing data forecasts but more on finding abnormalities in the data that is collected. Our use case will be exactly to trigger an alarm when we find out any data point that goes out of the estimated uncertainty interval.

Alerting and machine learning were added to the system. A Python script running the Prophet [15] library collects data from Prometheus to make data forecasting and detect data points that are not within the predictions' uncertainty intervals. This triggers Prometheus Alertmanager, to have the system throwing alerts. The Prometheus instance itself can also be configured to send alerts to the Alertmanager when certain criteria is met.

5 Tests and Results

5.1 Monitoring Solution

Fig. 4 displays the data being gathered on the monitored computer nodes shown in the Grafana "Node Exporter Full" dashboard. On the top left corner, we have two dropdown boxes: one for choosing the Prometheus job that is querying the host we want to analyse and the second box to choose the hostname (the options provided on the "host" box depends on what is chosen on the "job" box). On the top right corner, we have two parameters that are general to all dashboards: time window, here defined to show the last fifteen minutes, and the refresh period, set up as five seconds. This means each five seconds the dashboard will update its values to show the most recent fifteen minutes. That parameter can be easily tweaked by clicking on that drop-down and choosing any time frame (since Prometheus, by default, only holds data from the last 15 days, if you choose a previous point in time, the graphs will become empty).

On Fig. 4 we can observe expanded tabs, for example where "Basic CPU/ Mem/Disk" is being displayed. This tab contains graphs that give a quick overview about CPU, memory and disk. On it we have tiles or graphs displaying data gathered. This is the way that dashboards can be organized. We can see six tiles with meters showing CPU, memory and disk usage in current time. We then have other smaller five tiles showing "more permanent" metrics, like the number of cores, installed memory and swap and free root filesystem storage.

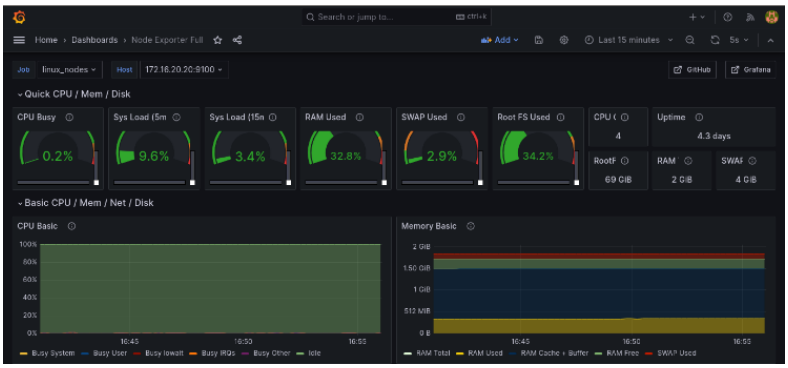

Fig. 4. Grafana Node Exporter Dashboard.

5.2 Anomalies Detection Solution

Now we will show how Prophet works. The anomalies detection starts with Prophet querying the Prometheus database. To get the best possible fitting and data prediction we query all the Prometheus data, corresponding to the last fifteen days. That data is then converted to a DataFrame (object type) and used as input to instantiate the Prophet model. Then, we proceed to apply the fitting to that data. After that, we make Prophet calculate a prediction. Since we are not, in this case, looking for a future trend and just want to analyse points that may be giving any clue about any abnormal behaviour, the prediction will only return the original data plus an uncertainty interval where values should be contained. Fig. 5 shows the graphical representation of that result, for the instant rate of increase of the total number of received bits (5 seconds delta of received bits) on the Prometheus host machine, on its interface "enp1s0". The Prometheus instance is collecting values every 5 seconds, and we use every data point available. Notice as well that there is a blue margin around the dots marking the data points (dark blue) and the fitting line. This margin is the uncertainty corresponding to a 95 percent confidence interval.

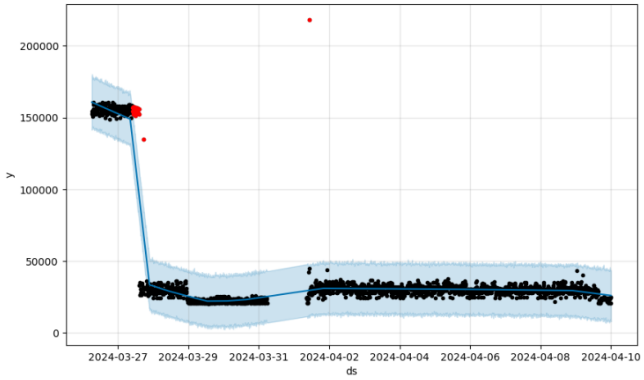

Fig. 5. Prophet Anomalies in red for the received data rate.

After we obtain the "forecast", we compare these values, considering the uncertainty, with the values that were given to the model for its instantiation. Those points are shown in red on Fig. 5.

In case there are any data point over the limit (in the span of all the used data points), an alarm request is sent to the Alertmanager and it redirects the alert to the proper recipients. The script is sending a POST HTTP request to the Alertmanager API `/v2/alerts` with a body containing the alert details. The alert will automatically be cleared if after five minutes there is no request refreshing the previous one.

5.3 Resources consumption

The Prometheus server is running four different processes: Prometheus, Grafana, Node Exporter and SNMP Exporter. The host is a computer operating GNU/Linux (Linux Mint 21), kernel version 5.15.0-48-generic, with an Intel(R) Core(TM) i5-6600K CPU @ 3.50GHz x86_64 (four cores) and 8 GB DDR4 memory.

The performance gathered data shows that Prometheus uses approximately 1.5% of CPU divided by seven threads. It only uses around 200 MiB of resident memory. On average, the database size keeps oscillating between 1.6 GB and 1.2 GB. Grafana uses around 0.2% of CPU when the web-app is not being accessed and higher, from 1% to 2%, when we were viewing the graphs and doing activity on the web-app. Grafana's memory usage is nearly constant, with 150 MiB of resident memory. Node Exporter's CPU consumption is steady and between 0.7% and 0.8%. Its resident memory was approximately 20 MiB. SNMP Exporter's CPU consumption is almost constant around 1.3% and uses around 22 MiB of resident memory.

6 Conclusions

Prometheus revealed to be a very flexible solution, since it has already great support for different protocols and data formats. Its community libraries offer lots of adaptation for common monitoring specific cases, and its open-source nature even allows for modification if we need to modify any tool (either official or community made) to fit a more restrictive case. Our measurements show that Prometheus has a very lightweight resource consumption which opens room for easy scaling. Besides that, the federation feature would facilitate scaling in cases of bigger and more complex architectures. Tools like Thanos can help Prometheus scale, providing the system with larger capacity for historic data and centralized querying.

Grafana was selected as visualization tool, as it is a standard tool to integrate with Prometheus nowadays and it is very straightforward to use. We imported several dashboards from Grafana Labs repository, regarding Linux machines, Windows machines and ethernet switches. After installing they were ready to use.

Prophet was used as a machine learning tool to analyse gathered data and discover anomalous data points. Prophet has a simple configurable model that can predict time series trends with an uncertainty range. We used it within a Python script that, after looking for anomalies, triggers an alarm in case any is found.

Future work should address a larger geographically disperse network, and measure how that would impact resources consumption. Another interesting study would be to develop the anomaly detection process and even include some automatic actions that could fix the detected issues.

References

1. Beck, K. et al., “Manifesto for agile software development,” The Agile Alliance, pp. 2002-04, 2001.
2. Case, J., Fedor, M., Schoffstall, M., Davin, J., “Simple Network Management Protocol (SNMP),” IETF RFC 1157, May 1990.
3. McCloghrie, K., Rose, M., “Management Information Base for Network Management of TCP/IP-based internets: MIB-II”, IETF RFC 1213, March 1991.
4. Berry, I., Roman, T., Adams, L., Pasnak, J. P., Conner, J., Scheck, R., Braun, A., “The Cacti Manual”, The Cacti Group, 2017. [Online] Available: https://files.cacti.net/docs/pdf/manual.pdf
5. Rabenstein, B., Volz, J., “Prometheus: A Next-Generation monitoring system (talk)”, Dublin: USENIX Association, 2015.
6. “Prometheus”, [Online]. Available: https://prometheus.io/
7. InfluxData, Inc., InfluxDB OSS v2, 2023. [Online]. Available: https://docs.influxdata.com/influxdb/v2/
8. Nagios Core Development Team and Community Contributors, “Nagios Core 4 Documentation”, Nagios Enterprises, 2018. [Online]. https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/index.html
9. Zabbix SIA, “Zabbix Manual”, Zabbix SIA, 2023. [Online]. Available: https://www.zabbix.com/documentation/current/en/manual
10. Grafana Labs, “Grafana Documentation” [Online]. Available: https://grafana.com/docs/grafana/latest/
11. Taylor, S. J., Letham, B., “Forecasting at scale”, PeerJ Preprints, 2018.
12. Grafana Labs, “Grafana Machine Learning Documentation,” [Online], https://grafana.com/docs/grafana-cloud/alerting-and-irm/machine-learning/
13. Weka Git Repository, [Online] Available: https://git.cms.waikato.ac.nz/weka/weka/-/tree/main
14. Pereira, P. J. et al., “A comparison of automated time series forecasting tools for smart cities”, Progress in Artificial Intelligence, G. Marreiros, et al. Eds. Cham: Springer International Publishing, pp. 551-562, 2022.
15. Facebook, “Facebook Prophet”, [Online] Available: https://github.com/facebook/prophet/
16. “Thanos,” [Online] Available: https://thanos.io/
17. Prometheus Authors. Prometheus GitHub Repository. https://github.com/prometheus/prometheus