UNIVERSITY OF ŽILINA
Faculty of Management Science and Informatics

BRAIN:IT

PROJECT

Multi-tenant cybersecurity operations centre designed as an open cloud service with machine learning elements

Linked to work package KPB1 – Design of the virt. solution

Linked to output V1, the result of the analysis of available solutions

Funded by the European Union
NextGenerationEU

RECOVERY AND RESILIENCE PLAN

GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

Contents

SOLUTION OPTIONS 3

1 ELK 3

2 Canonical Observability Stack (COS) 4

3 Monasca 5

4 Ceilometer 7

5 Grafana Loki 8

6 Prometheus 11

SOLUTION OPTIONS

Several solution options were analysed. Each was given one subsection in this document. The selected ones include: ELK, COS, Monasca, Ceilometer, Grafana Loki, Prometheus.

1 ELK

One of the popular solutions for centralized logging and monitoring is the ELK Stack. It is a combination of several systems merged together. These components are Elasticsearch, Logstash and Kibana. For a long time, this solution was one of the preferred ones and built up a community; recently, however, other solutions have also been coming to the fore.

Individual components used:

- **Elasticsearch** serves as a tool for searching, analysing and storing logs; it excels in search speed and in its ability to analyse large volumes of logs.
- **Logstash** is used to aggregate, filter and transform logs from various sources (including OpenStack components) and then send them to Elasticsearch for storage and analysis. Log transformation is performed by so-called pattern mapping using Regex and Grok filters.
- **Kibana** provides an interface for visualizing the data stored in Elasticsearch.

When integrating OpenStack with ELK, logs are sent to Logstash. This sending is ensured either by using an exporter such as filebeat, or the logs are sent via network protocols such as syslog. Input plugins for Logstash are used to receive logs efficiently. The logs are then filtered and transformed using Regex and Grok patterns, whose task is to remove unwanted information. They can add additional metadata and labels and convert the logs as a whole into a structured format such as JSON, which is suitable for storage and analysis. ElasticSearch takes care of storing, indexing and analysing logs for fast searching and efficient storage. The data are then visualized in Kibana, where dashboards can be created and the performance, security and availability of services can be monitored.

Advantages of integrating OpenStack logging with the ELK Stack:

- **Fast analysis and search:** Elasticsearch enables fast searching and analysis of logs, which makes it easier to identify problems and optimize the performance of the monitored infrastructure.
- **Flexible visualization:** Kibana provides a wide range of tools for visualizing logs and metrics and at the same time makes it possible to adapt dashboards to specific monitoring needs.
- **Extensibility and scalability:** The ELK Stack can be scaled and adapted to process large amounts of data, which enables its use in large-scale cloud environments.
- **Log parsing:** Transforming logs with Logstash allows us to reshape the incoming data into a form that is more readable for computers, which leads to more efficient searching and analysis. The Grok mapping tool helps us to add additional data or, conversely, to remove redundant information from the logs.

Some of the disadvantages:

- **Resource intensity:** The ELK Stack requires a significant amount of computing and storage resources, especially when processing large volumes of data.
- **Complexity of deployment and management:** Correct configuration and optimization of the ELK Stack can be complicated and require an advanced understanding of the whole system.
- **Latency:** When processing a large number of logs with insufficient resources, and owing to the high requirements of this system, increased latency may occur when searching data in Kibana.

Despite these challenges, ELK Stack integration offers a powerful logging tool for OpenStack, helping to improve the performance, security and availability of cloud services.

2 Canonical Observability Stack (COS)

COS is an integrated solution for monitoring, logging and tracing services in cloud as well as on-premise environments. It is developed directly by Canonical, the creators of MaaS, Juju and charms. COS is designed to provide deep insight into infrastructure and applications, with an emphasis on simplicity, automation and open standards. This solution consists of several open technologies combined into multiple charms. COS forms a monitoring solution precisely for infrastructure deployed using Juju and MaaS, in order to cover a wide range of monitoring technologies.

COS consists of:

Prometheus: collection and storage of metrics on the performance and availability of services.

Grafana: a visualization tool for creating dashboards and alerts from data collected by Prometheus and other sources.

Loki: for aggregation and efficient searching in application and infrastructure logs.

Jaeger and OpenTelemetry: distributed tracing and analysis of the latency and performance of services.

Grafana agent: an exporter of logs and metrics.

These components are integrated into a unified charms solution that can be deployed as part of a Kubernetes cluster or in more traditional server environments.

Integrating and deploying this solution requires infrastructure in Juju with a prepared Juju cloud into which the services will be deployed. COS supports both deployment options in Juju: either into a bootstrapped k8s cluster with a Juju controller, or into a machine cloud. The COS Juju charms deploy all the necessary applications, such as Prometheus, Grafana, Loki and Jaeger. The necessary configuration is then performed in the Juju charms, relations are created, and exporters are deployed for sending metrics and logs. Also, after the relations with Grafana have been created, specific dashboards for visualizing metrics and logs are created in it, and alerting rules are configured. To monitor applications, the COS stack must be integrated using Juju add-relation, specifically by connecting the grafana-agent charm to the applications we

want to monitor. After a relation with a given application has been created, the grafana-agent charm is installed alongside the application as a subordinate. A subordinate charm/application is meant to run alongside another application. It is a type of charm designed precisely for logging and monitoring. The grafana-agent stays with the application throughout its entire lifecycle, during scaling out, scaling down, etc.

Advantages of COS integration:

- **Use of charms:** Supports the integrity and uniformity of the solution, since the entire existing OpenStack infrastructure has already been built using Juju and charms.
- **Open standards and technologies:** It uses popular and proven open-source solutions, which ensures flexibility, extensibility and a community.
- **Integration:** Native, built-in integration for Kubernetes or a machine cloud, and support for cloud-native applications.
- **Easy scalability:** COS components support easy horizontal scaling.

Disadvantages:

- **Complexity:** Integrating multiple components requires advanced technical knowledge; in addition, it contains extra technologies such as Jaeger and OpenTelemetry.
- **Poor documentation:** This is quite a serious problem for new users. Juju and charms are nice tools for managing and deploying applications, but they are not very well tested or documented, which prevents their full potential from being exploited.

Canonical Observability Stack is a powerful and flexible solution for monitoring and logging infrastructure and applications. Its integration of open-source tools such as Prometheus, Grafana, Loki and Jaeger offers a comprehensive overview of the performance, security and availability of services. Despite a certain complexity of deployment and the need to scale resources according to requirements, COS provides a good basis for effective monitoring in modern cloud-native applications as well as in traditional IT environments. However, the weaker community and documentation remain a major gap.

3 Monasca

Monasca is an open-source project designed for monitoring and logging, primarily for OpenStack but also for other cloud environments. It offers high scalability and performance and is designed to process large volumes of data efficiently in real time. Monasca enables users to monitor the health, performance and availability of cloud resources and applications, all through the MonascaAPI. The Monasca architecture consists of several main components that together provide comprehensive monitoring services.

Monasca components:

- **Monasca API:** Serves as the gateway for all monitoring requests and enables the configuration of alarms, metrics and notifications.
- **Monasca Agent:** Deployed on the monitored servers, it collects metrics, which it then sends to the system via the API.
- **Kafka:** Used as a message broker for storing and transferring metrics between components. Integration with an existing message broker already in use is also possible.
- **Monasca Persister:** Receives metrics from the Kafka system and stores them for later processing in a database, a so-called time series database, such as InfluxDB.
- **Keystone:** Requires authentication and authorization in the OpenStack environment.
- **Grafana with the Monasca plugin:** Can be used to search and visualize metrics and alarms.

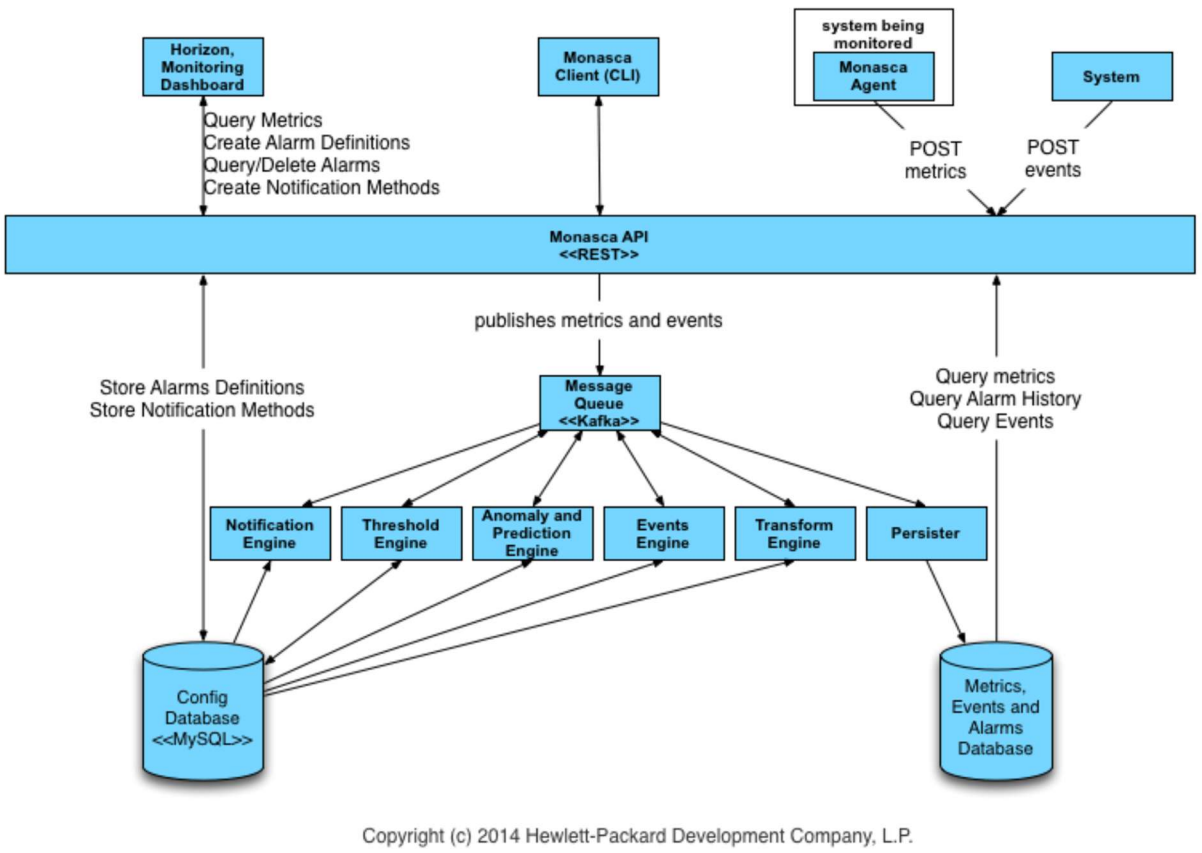


Figure 1 Monasca architecture

Monasca supports 4 different deployment methods using Docker, k8s, ansible and pupet. To set it up, the MonascaAPI must be connected to keystone and Apache Kafka, since it is a system created directly for monitoring OpenStack. Monasca supports taking its configuration directly from a database such as MySql. It further requires the deployment of a time series database, the so-called Monasca Persister, such as InfluxDB. This is followed by the connection to grafana, either using the Grafana plugin or by connecting Grafana directly to InfluxDB, which holds the monitoring data for visualization and further processing. Finally, the

Monasca agent is installed on the nodes and VMs from which we want to send monitoring data.

Advantages:

- **Scalability and performance:** Some Monasca deployment methods also support scaling.
- **Integration with OpenStack:** Monasca was primarily developed for monitoring OpenStack and therefore integrates easily with other OpenStack services and modules.

Disadvantages:

- **Need for administration:** It requires constant administration and maintenance to ensure performance and reliability.
- **Community:** Monasca is not one of the frequently demanded solutions, and the quality of its documentation develops accordingly.
- **Non-existent charms:** As a project intended directly for OpenStack, its disadvantage is that no deployment in the form of Juju charms has been built for this software.

Monasca is a robust and scalable solution intended for monitoring in cloud environments, which is ideal for the OpenStack platform. Its architecture and components provide a detailed overview of the performance and health of cloud resources. Although its deployment and administration can be more demanding and it lacks deployment via a charm, in some cases it may be a suitable monitoring tool.

4 Ceilometer

Ceilometer is an OpenStack component that provides resource usage data collection and monitoring functions for cloud environments. It was designed to enable users and administrators to obtain detailed information on the resource usage of their infrastructure in cloud environments. It is designed to facilitate billing, benchmarking and capacity planning. It is often used in conjunction with other modules, such as CloudKitty, to verify the performance of the entire cloud infrastructure. [12] Ceilometer records information about virtual machines, storage, network services and other resources, thus providing a rich source of data for the analysis and management of the cloud environment.

Ceilometer consists of:

- **Ceilometer Agent:** Collects metrics and usage data from various sources within OpenStack. Agents can be deployed on hypervisors, network devices or other nodes.
- **Ceilometer Collector:** Receives data from the agents and forwards them to the data store.
- **Datastore:** Storage for the metrics and events collected by Ceilometer. It can use various databases, for example MongoDB or MySQL, to store data. With newer versions of OpenStack, the use of another OpenStack module, Gnocchi (TSDB), is recommended.
- **Ceilometer API:** Provides an interface for retrieving the data and metrics stored in Ceilometer. It enables access to the collected data via the API.

Deployment of Ceilometer into OpenStack charms:

To deploy Ceilometer, the Gnocchi TSDB module must be available as the data store. Ceilometer must be integrated, with relations created with keystone, rabbitmq and the above-mentioned Gnocchi. Subsequently, the Ceilometer-agent must be deployed on each nova-compute node, and likewise relations must be created with the message broker for interaction with the other parts of this software.

Advantages:

- **Integration with OpenStack:** As part of the OpenStack ecosystem, Ceilometer integrates with other services such as keystone, rabbitmq and other third-party tools.
- **Flexibility:** It supports multiple databases as the data store, which enables flexible data processing. TSDBs can be inserted directly as a data source into a Grafana dashboard using the Grafana plugin.
- **Extensibility:** thanks to its modular architecture and the possibility of deployment using charms, Ceilometer can easily be extended with new data sources and metrics.

Disadvantages:

- **Performance:** Collecting and processing large amounts of data can burden system resources, which may affect the overall performance of the system, and it is necessary to keep in mind the availability of the resources provided to CC users.

Ceilometer is part of the monitoring framework in the OpenStack ecosystem, which provides an overview of the performance of the entire cloud and of the user instances created within the cloud. It can be integrated with other tools such as ManageIQ to display resource usage by users. Ceilometer is a module worth deploying to show performance, consumption and resource usage to end users. [13]

5 Grafana Loki

As the name suggests, Grafana Loki is a product of the company behind the development of the Grafana dashboard. This logging system is scalable, efficient and easy to integrate, which makes it ideal for monitoring cloud and microservice environments. Loki focuses on efficient storage and searching of logs, with an emphasis on simplicity, low storage costs and performance. Inspired by Prometheus, Loki uses the same approach to data labelling (labels), which simplifies working with metrics and logs in the same ecosystem.

Grafana Loki components:

Promtail: An agent deployed on target devices to collect logs and send them to Loki. Promtail collects logs and assigns them labels, which help with their organization and searching.

Grafana Loki: The central component that receives, stores and indexes logs. Loki is designed primarily to be resource-efficient in storage and fast in searching.

Grafana: Used to visualize and search the logs stored in Loki. Grafana enables users to search logs by labels and display them in context with metrics from Prometheus.

Figure 2 Grafana Loki architecture

Distributor

The distributor receives log data from clients and distributes them to multiple Ingesters. Distribution is based on consistent hashing, which enables the scalability and resilience of the system.

Ingester

The Ingester is responsible for processing incoming log data, storing them temporarily in memory and periodically writing these "chunks" of data to long-term storage, such as the index and chunks storage.

Ruler

The Ruler evaluates rules defined by users and creates alerts or records based on these rules. It can also query the Ingesters or the long-term storage directly.

Query Frontend

The Query Frontend is a layer that performs caching and sharding, which optimizes query execution. It stores (caches) queries and their results in internal memory and also enables parallel processing of large queries by splitting them into smaller parts.

Querier

The Querier processes queries on log data. It communicates with the Ingesters to obtain data from the cache and with the storage for older data. Query results can be used directly or displayed in Grafana.

Memcaches

Memcaches are used to cache various parts of the data – results (R), indexes (I) and chunks (C). These caches increase system performance by reducing repeated reading of data from storage.

Index

The Index stores metadata about chunks, such as identifiers and timestamps. It enables fast searching for relevant data when answering queries.

Chunks

Chunks contain the logged data themselves, divided either into segments or chunks. These chunks are stored in object storage, such as Amazon S3, Google Cloud Storage, or a locally compatible solution such as MinIO.

Grafana Loki deployments:

Grafana Loki is software that can be deployed in multiple ways; it supports deployment using various tools into cloud infrastructure in containerized form. Grafana Loki can be deployed in various modes, namely monolithic, simple scalable and microservices mode. It supports configuring the backend storage to S3, to files, TSDB and others. The requirements for the backend storage also differ depending on the deployment. Although the configuration may differ for different types of deployment, the basis remains the same: Grafana Loki receives, stores and searches labelled logs, which are most often visualized using a Grafana dashboard. Exporting logs from target devices is handled by the Promtail log shipper, which also handles their labelling. In Grafana, Loki can then be added as a datasource, and the data can be accessed directly from Grafana for display and further analysis.

Advantages:

- **Cost efficiency:** Like Prometheus, Loki minimizes log storage costs by storing indexes and already compressed logs separately.
- **Integration with Grafana:** Simple integration with Grafana enables easy searching and visualization of logs together with metrics.
- **Simplicity and speed:** Loki is designed for simplicity, speed and efficiency, which makes it an attractive choice for solutions where efficient resource usage is a priority.
- **Community and diversity:** Grafana Loki is becoming increasingly popular, which accelerates its development, extensibility and plugins.

Disadvantages:

- **Limited log searching:** Loki focuses on label-based searching, which may be limiting for some use cases compared with full-text searching.

- **Comprehensibility of configuration:** Some deployment types are difficult to understand; Loki itself consists of several smaller services, which we listed above.

Grafana Loki is thus an excellent choice for solutions that need an efficient logging system with low resource requirements and at the same time use Grafana for monitoring and visualizing metrics. Its integration with the Prometheus ecosystem provides a powerful tool for log management using labels. Although searching based exclusively on labels may be limiting in some cases, the advantages that Loki offers in terms of simplicity, speed and integration with Grafana make it a great choice for many solutions.

6 Prometheus

Prometheus is an open-source monitoring and alerting system that was originally developed by SoundCloud. Today it is widely used in cloud environments and microservices thanks to its performance, flexibility and reliability. Prometheus enables organizations to collect and process metrics from a wide range of sources, from operating systems to applications running in cloud or containerized environments. With its robust data model and powerful query language (PromQL), Prometheus provides deep insight into the performance of applications and infrastructure.

Prometheus components:

- **Prometheus server:** Collects metrics from configured sources at specified intervals, stores them and executes rules for evaluating alerts.
- **Client libraries:** Provide interfaces for applications to send metrics to Prometheus.
- **Push gateway:** Enables temporary storage of metrics for jobs that exist only for a short time.
- **Exporters:** Collect metrics from standard services or servers and translate them into a format suitable for processing by Prometheus.
- **Alertmanager:** Processes the alerts generated by the rules defined in the Prometheus server and ensures their delivery via a so-called web hook channel.

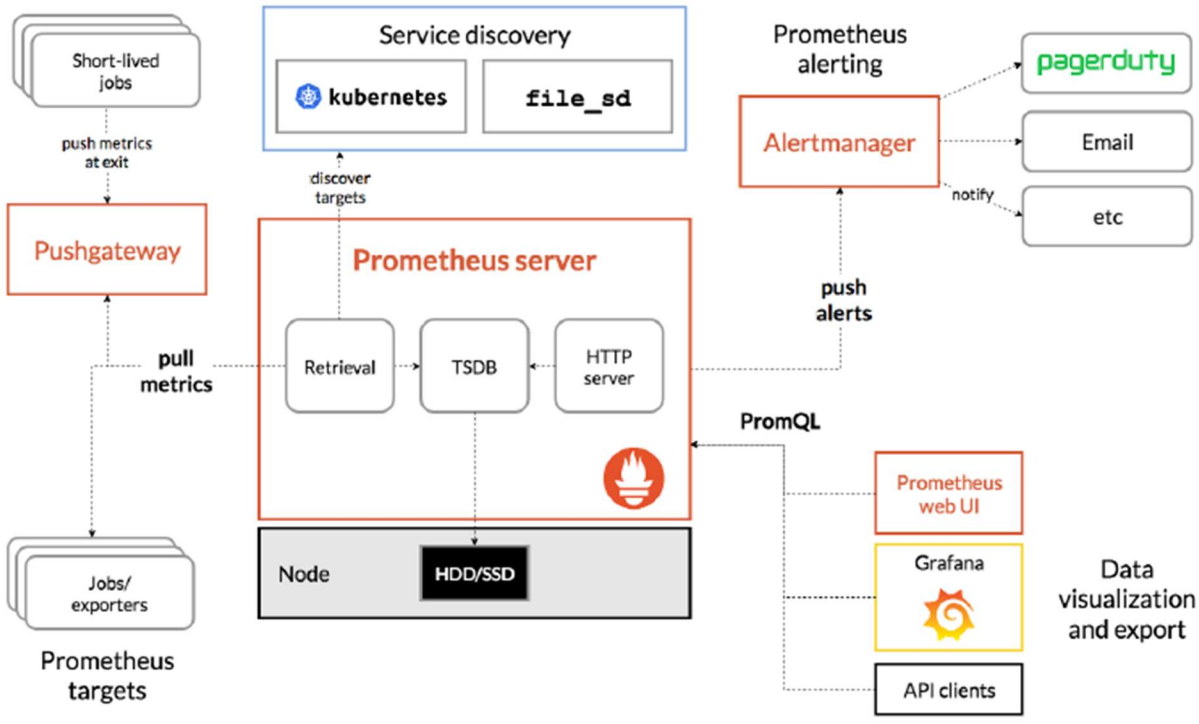


Figure 3 Prometheus architecture

Prometheus deployment:

Deployment into containers or on bare metal is possible by installing and configuring the Prometheus server, followed by installing the necessary exporters for monitoring specific information. Prometheus also includes the use of Alertmanager, which is configured for the specific metrics collected, with critical values and the applications to which we want to send notifications being set. Prometheus also offers a simple web interface where it is possible to test, display the collected metrics and verify other settings; however, this interface is simple, and for deeper analysis the data are displayed using Grafana.

Advantages:

- **Multidimensional data:** Prometheus stores metrics with labels, i.e. it makes it possible to identify metrics by a name and key/value pairs.
- **Active pull model:** Prometheus actively collects metrics from its targets, which simplifies management, ensures data consistency and reduces the resource usage of its targets.
- **Powerful query language (PromQL):** Enables complex queries on metrics and their aggregation.
- **Built-in alerting:** Integration of alerting directly in Prometheus eliminates the need for external services for monitoring and alarming.
- **Automatic target discovery:** Use in cloud infrastructures such as k8s.

Disadvantages:

- **Limited long-term data storage:** Without integration with external storage, long-term data storage in Prometheus can be challenging.

Prometheus is a high-performance and flexible monitoring and alerting tool that has become a standard in many cloud and microservice architectures. Its strengths, such as the multidimensional data model, its own query language and the built-in alerting system, make it an ideal solution for modern monitoring requirements. Despite some challenges, such as long-term data storage and the complexity of administration, Prometheus offers a robust solution for organizations seeking deep insight into their systems and applications.