TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Nikita Ratškov 206195IAAB

Centralized Logging System for a Multi-Tenant Environment

Bachelor's thesis

Supervisor: Mohammad Tariq Meeran
PhD

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Nikita Ratškov 206195IAAB

Keskne klientide vahel jagatud logihaldussüsteem

Bakalaureusetöö

Juhendaja: Mohammad Tariq Meeran PhD

Tallinn 2023

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Nikita Ratškov

10.04.23

Abstract

Company X lacks a proper solution for log collection and aggregation leading to manual processing. In case of any related problem or customer request it takes extra time to do everything manually causing the unnecessary delays for customers also leading to dissatisfaction with the Company X service.

The thesis aims to solve the problem faced by Company X. Solution should be able to collect and process logs from different WAF (Web Application Firewall) sensor locations and provide different storage options.

Work includes an in-depth literature review, which was conducted to identify the available technologies and find the most suitable and effective solutions among them. To achieve the goal and solve the problem, the Author implemented the experimental setup using all the data obtained from the literature overview and following all the requirements requested by Company X.

The solution development steps are described in 2 major parts: Experimental Setup and Implementation. The Experimental Setup part consists of chosen options and proposed configurations. The Implementation part consists of building of the solution and its integration with the existing tools.

The result of the thesis is a logging collection and aggregation solution for Company X which provides all the requested features and meets the requirements. The goal was achieved by building the fully working integrated logging solution and implementing it as Infrastructure as Code.

This thesis is written in English and is 46 pages long, including 7 chapters, 19 figures and 2 tables.

Annotatsioon

Keskne klientide vahel jagatud logihaldussüsteem

Ettevõttel X puudub korralik lahendus logide kogumiseks ja koondamiseks, mis viib käsitsi töötlemiseni. Mis tahes seotud probleemi või kliendi soovi korral kulub lisaaega kõige käsitsi tegemiseks, mis põhjustab klientidele tarbetuid viivitusi, mis omakorda toob kaasa ka rahulolematuse Ettevõte X teenusega.

Lõputöö eesmärk on lahendada ettevõtte X probleem. Lahendus peaks suutma koguda ja töödelda logisid erinevatest WAF-anduritest ning pakkuma erinevaid salvestusvõimalusi.

Töö sisaldab põhjalikku kirjanduse ülevaadet, mis viidi läbi olemasolevate tehnoloogiate väljaselgitamiseks ning nende hulgast sobivaimate ja tõhusamate lahenduste leidmiseks. Ülesande saavutamiseks ja probleemi lahendamiseks rakendas autor eksperimentaalse seadistuse, kasutades kõiki kirjanduse ülevaatest saadud andmeid ja järgides kõiki ettevõtte X nõudeid.

Lahenduse arendamise etappe kirjeldatakse üksikasjalikult kahes suures osas: eksperimentaalne seadistamine ja juurutamine. Eksperimentaalse seadistuse osa koosneb valitud valikutest ja pakutud konfiguratsioonidest. Rakendamise osa koosneb lahenduse ehitamisest ja integreerimisest olemasolevate tööriistadega.

Lõputöö tulemuseks on ettevõtte X logide kogumise ja koondamise lahendus, mis pakub kõiki nõutud funktsioone ja vastab nõuetele. Eesmärk saavutati täielikult töötava integreeritud logimise lahenduse ehitamisega ja kogu ehituse rakendamisega infrastruktuur koodina.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 46 leheküljel, 7 peatükki, 19 joonist, 2 tabelit.

List of abbreviations and terms

API	Application Programming Interface
AWS	Amazon Web Services
CA	Certification Authority
DB	Database
DNS	Domain Name System
ECS	Elastic Container Service
HCL	HashiCorp Configuration Language
HTTP	Hypertext Transfer Protocol
IaC	Infrastructure as Code
IT	Information Technology
JDK	Java Development Kit
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
Regex	Regular expression
SaaS	Software as a Service
SAN	Subject Alternative Name
SOC	Security Operation Center
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOML	“Tom's Obvious, Minimal Language”
TSDB	Time Series Database
VRL	Vector Remap Language
WAF	Web Application Firewall
XSS	Cross-Site Scripting
YAML	“Yet Another Markup Language”

Table of contents

List of figures

List of tables

1 Introduction

Logging is one of the most essential components in any modern IT system operation. A proper logging solution should be able to provide any essential information about various system events like system activity, errors, and performance, making it easier to identify and resolve any issue or incident. For cloud-based companies, logs can be challenging to manage due to enormous amounts of data from various locations.

Lack of the logging system would lead to serious problems for Company X, including degraded customer service quality and increased downtimes. The inability to find the proper logs to identify the root of the problem leads to further delays in resolving issues. Additionally, unfiltered and unaggregated logs require more storage space, which can lead to higher costs and impact the company's budget.

All the problems mentioned can be solved by using a proper and complete logging solution which covers all essential aspects: collection, aggregation, correlation, analysis, reporting, and storage.

1.1 Problem statement

Company X is getting a lot of different logs from separately located WAF sensors. Currently these logs exist only on WAF sensors located in a multi-tenant configuration, which makes it hard to collect all of them manually by going to each of the sensors separately with plain SSH (Secure Shell). What is more, Company X uses mostly cloud based solutions for their products and tools, therefore logs are also stored in the cloud. Unfiltered and unaggregated logs may take up a lot of space and lead to increased costs of data storage.

1.2 Goal of the Thesis

The main goal of the research is to find suitable options for building the complete logging collection and aggregation system for Company X and integrate it into the production environment. Solution must follow Company X's security requirements and measures, provide seamless integration with currently used systems and solutions, provide scalability and high availability options to ensure that no data or services are lost.

This new solution will be integrated with existing internal logging tools to provide automatic collection, forwarding, and visualizing the logs for production environment and increase work efficiency of the SOC (Security Operation Center) team. Tool must be able to parse and aggregate different types of logs automatically without any additional observation from the user. The whole solution should be as lightweight as possible while meeting the functional and security requirements provided.

1.3 Objectives of the research

- To analyze and find the most suitable logging solutions
- To meet all the defined requirements
- To implement and integrate the solution

1.4 Research questions

- What logging options are available?
- What requirements should be met?
- What can be an appropriate solution for Company X?

2 Literature review

In today's digital age, companies are looking for ways to improve their IT infrastructure and services quality. To achieve this, many organizations have decided to use various logging solutions.

This section presents a review and analysis of tools and technologies which can be used for building the proper solution for the logging problem of Company X. The purpose of this review is to examine the current situation with the proposed options. Review of the existing literature should provide insights into best practices for solving the problem.

2.1 Organization

US-based cybersecurity Company X is providing WAF services for their clients around the world. Company's solutions are designed to protect web applications and APIs (Application Programming Interface) from different types of attacks, including SQL (Structured Query Language) injection, XSS (Cross-Site Scripting) and other common types.

They have asked the author to conduct research and find a proper solution for their logging problem. They are using cloud-based multi-tenant system with services provided by companies like Amazon or Internap and located in different geo locations.

2.2 Multitenancy

Tenant is a group of users sharing the same view of an application they use. This view includes the data they can access, configurations, user management and some other functional and non-functional options. Usually, groups are members of different legal entities.

Multitenancy is an approach to share an instance of an application between multiple tenants by providing every tenant a dedicated "share" of the instance, which is isolated from other shares regarding performance and security.

2.3 Syslog

Syslog is a standard protocol for message logging. It allows various applications, servers, databases, and network devices to send event messages to a predefined, usually centralized, server or service. Generated messages can include different information about system errors, security alerts, application events and so on. Syslog protocol works by client-server model. The client is usually a device or application that generates log messages, while the server is the logging system that collects and stores such messages. Syslog messages can be sent over the network using UDP (User Datagram Protocol) and TCP (Transmission Control Protocol) protocols or between applications on the same host with UNIX sockets. [1]

2.4 Nginx

Nginx is open-source software which can be used as web server, reverse proxy, and load balancer. Nginx is a high-performance solution which can handle many concurrent connections and requests [2]. Company X uses it in reverse proxy mode as a main component of sensor instances, therefore it acts as a main source of the access and error logs for this research.

2.5 Common logging solutions

As logging is an important aspect of every IT related activity, therefore a lot of solutions are available like: Grafana Loki, ELK (Elastic, Logstash, Kibana) Stack, Splunk and Graylog.

Grafana Loki is horizontally scalable, highly available and multi-tenant logging system designed to be cost-effective and easy to operate. It is an open-source solution which can be self-hosted on premises or in the cloud. Loki is a highly customizable tool which allows users to store logs in a way that suits their specific requirements. Additionally, it can be easily integrated with other solutions like Fluentd or Vector.

ELK is a widely used open-source log and analytics stack. Elasticsearch is a search and analytics engine that stores and indexes data, also providing fast and efficient data search and analysis. Logstash is server-side data processing pipeline that collects, parses, and sends it to another predefined place for investigation and examination. Kibana is a data

visualization tool, like Grafana, that allows users to explore, visualize data and build custom dashboards, charts, and reports. [3]

Splunk is a platform that collects, indexes and analyses logs, metrics, and other types of data. Splunk is not an open-source solution, and it is provided only as a paid service. Splunk also supports different data sources, like cloud instances or network devices, it has a big plugins and integrations ecosystem.

Fluentd, Promtail, Vector are all open-source log collection, processing and forwarding tools. These tools can be used with different frontends like Kibana or Grafana to visualize and structure the data.

2.6 High availability

High availability is the ability of a system or application to remain operational and accessible for users for an extended period without any disruptions or downtime.

A highly available system must be designed with redundancy and fault tolerance in mind. System must be able to provide uninterrupted services during essential time periods, most hours of the day and most days of the week throughout the year. [4]

2.7 Cloud

AWS (Amazon Web Services) and Digital Ocean are both cloud computing platforms that provide tools, resources, and infrastructure options for deploying, scaling, and managing applications.

AWS is a complete cloud platform that provides a wide range of different services like computing, storage, networking, and security. It also offers more enterprise-oriented features like high availability, failover, and recovery.

Digital Ocean, on the other hand, is a simpler solution that provides virtual servers and a few other important services such as storage, databases, and load balancing. All this makes it an option for enthusiasts, developers or small businesses who need an uncomplicated and cheaper platform than AWS. [5]

2.8 Portability

Portability is the ability to move applications between different environments without any significant configuration changes or adaptations. It means that an application can be deployed to different platforms, operating systems, or cloud environments without changes. [6]

Docker is one of the most popular open-source container technologies. It is a software platform that allows to create, deploy, and run applications using containerization technology. Docker provides portability option which allows to pack applications with all the required dependencies into a single container and help users or customers to deploy software faster and with less issues.

2.9 Infrastructure as Code

IaC (Infrastructure as Code) is an approach that involves managing IT infrastructure resources such as virtual machines, network and storage with software code and templates. IaC enables developers to automate the process of deploying and managing infrastructure to make it faster and more consistent. [7]

Terraform is an open-source IaC tool for building, changing, and versioning IT infrastructure. Terraform focuses on the abstraction of the datacenter and associated services allowing users to define various infrastructure resources such as virtual hosts, databases, and load balancers across multiple cloud providers like AWS, Azure or Digital Ocean and on-premises data centers. It works with infrastructure definition by providing a configuration file created by user using a high-level language called HCL (HashiCorp Configuration Language). [8]

Ansible is an open-source automation tool for application deployment and configuration management. It can automate a wide range of management tasks across different environments. Ansible uses SSH for changes and does not require any agents or daemons to be installed on remote hosts making it easier to integrate with the existing environment.

The main difference between both is their configuration language and use cases. Terraform uses HCL for configuration files and its aim is to provide infrastructure, while

Ansible uses plain YAML (Yet Another Markup Language) for its playbooks and its main purpose is configuration management on already provided hosts. [9]

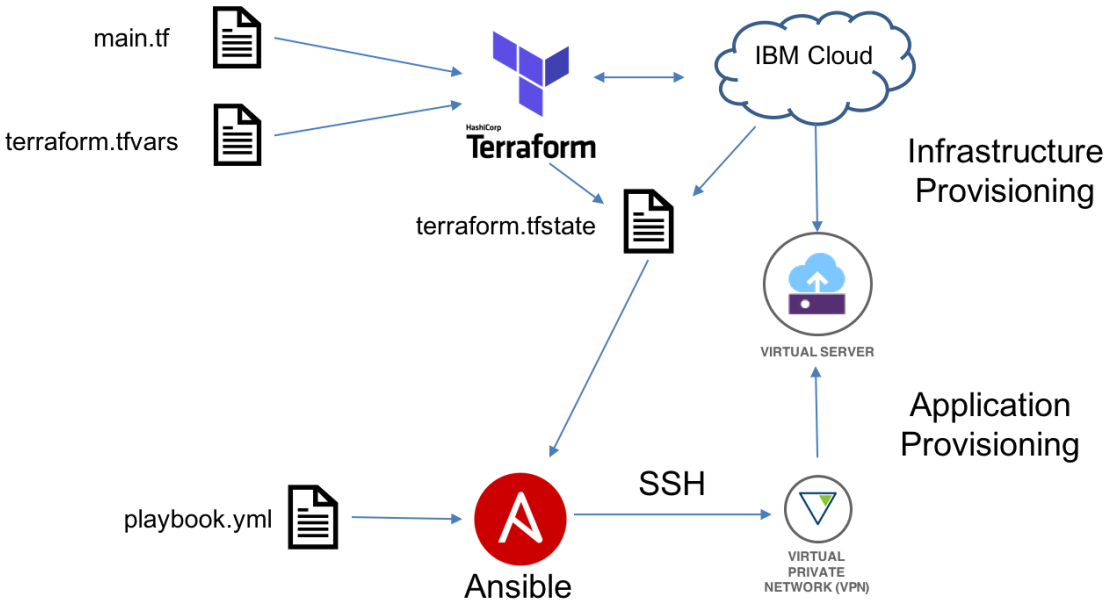

Figure 1. Example of using Terraform with Ansible (Source: [10])

3 Methodology

In this section, the author presents research method, data collection and testing environment used in the study.

3.1 Research method

The choice of research methods may vary based on a variety of factors, including the object of research, settings, and timeframes. For this work, the most suitable methods are analytical review and empirical search.

An analytical review process involves reviewing of the existing literature, research studies, and other relevant works.

Empirical research involves collecting and analyzing through the experiments to test the solution against company functional and security requirements.

3.2 Data collection

The observation method is the main data collection method for this work. Data is being collected from various open sources and with the information from the running corporate environment including on-site interviews with IT (Information Technology) Engineers of the Company X, observing and collection of the data regarding security and functional aspects of IT systems. IT related scientific articles, documents and previous research have also been reviewed to get data about various options and possibilities to analyze.

3.3 Testing environment

To conduct the research without affecting the corporate production environment and provide a testing stage of the proposed solutions before implementing them into production, the testing cloud environment will be implemented and used for all the development and testing purposes.

4 Experimental Setup

Experimental setup, also known as a testing environment, is critical for testing and validating the changes before implementing into production environment. This setup allows developers and engineers to test changes in a controlled environment to not cause any unexpected problems with production.

4.1 Requirements

Log collection and analysis is always relevant to IT, Engineering and Security. What makes this unique is the various limitations the solution should run within. The simplest solution would be to add a hook into every sensor that streams logs to a centralized location. However, it adds unwanted network overhead, and processing power for the entire system. Similarly, there are sensors regionally deployed and load balanced, so collection of the logs for specific investigations is cumbersome since it is impossible to connect to each instance separately. Also, various solutions that exist have known security vulnerabilities, this is why Company X tries not to use Java based solutions for example. Regarding the deployment options, the solution should be open-source to allow the self-hosting option, be independent of the service provider which also decreases costs and allows moving between instances or even different providers.

Main requirements for the logging solution:

- Open-Source
- Self-Hosted
- Not Java based
- Lightweight with less dependencies
- Mature product with support
- Resource efficient

The number of options has been narrowed down to 3: Grafana Loki, Graylog and ELK. All these products are open source, they can be self-hosted, mature enough to be used in production environment and have good community support.

However, ELK stack and Graylog seems to be slower and less lightweight for our usage cases, also they are using Java as the main project programming language. Java applications require JRE (Java Runtime Environment) or JDK (Java Development Kit) packages to be installed on the server, which take up a lot of space and they also do not meet the security requirements, therefore the author decided to skip any Java based solutions.

The author has found that the most suitable solution for our environment is Grafana Loki. Loki instance uses Grafana as dashboard that is already used in production environment, so it will allow to reuse the resources and avoid additional costs by not running any separate dashboards like Kibana.

Additionally, Loki can work with various logging collectors and processors like Vector, FluentBit, Fluentd and Promtail, which gives a variety of options to choose from. Grafana is based on Golang and does not require Java packages, therefore it does not have any Java specific security issues.

The comparison table was composed to provide a visual interpretation of the available options with their advantages and disadvantages for our usage cases:

Table 1. Logging solutions comparison

Solution	**Source**	**Free options**	**Deployment**	**Main PLs**	**Community support**	**Collectors**
Graylog	Open	Self-Hosted	SaaS and self-hosted	Java	Yes	Syslog or other community options
ELK (Logstash)	Open	Self-Hosted	SaaS and self-hosted	Java and Go	Yes	Filebeat
Grafana Loki	Open	Self-Hosted	SaaS and self-hosted	Go	Yes	Promtail, Fluentd, Fluentbit and Vector
Datadog	Partially open	Trial	SaaS	-	Yes, closed.	Datadog agent
GoAccess	Open	Free	Self-hosted	C and JS	Yes	-
Splunk	Partially open	Trial	Saas	-	Yes	Syslog
SigNoz	Open	Self-Hosted	Saas and self-hosted	TS and Go	Yes	Open Telemetry

Falcon LogScale	Partially open	Limited	SaaS and self-hosted	Java	Yes, closed.	Internal agent
New Relic	Partially open	Limited	SaaS	-	Yes, low activity.	Internal agent
Mezmo/LogDNA	Partially open	Limited	SaaS	-	No	Internal agent
Loggly	Partially open	Limited	SaaS	-	No	Internal agent

4.2 Log collection

It is not possible to hook into every sensor and collect the logs directly from them due to specific infrastructure realization. Moreover, any log collectors like Vector or Fluentbit cannot be added to the sensor's images, as it needs to pass implementation, testing, integration, and some other phases to be done, which will take a lot of time. Therefore, a fully automatic logging solution cannot be implemented now.

However, Company X developed tools for the internal SOC team to collect the logs, this tool could be modified for new purposes to send logs directly to the parser using the syslog protocol.

Internal tool kit must collect the logs and send them to the Vector instance, for this purpose some changes should be implemented: Vector host must be connected with tools with internal jump host through SSH tunnel and new function to collect and send all the logs must be implemented to allow users to run through the entire process with only one command.

4.3 Log processing

Vector, Promtail, Fluentd are all popular log collection and aggregation tools. Each tool has its own advantages and disadvantages, and the choice of the right tool to use depends on a variety of factors, including the specific use cases, volume of data, acceptable performance, and resource consumption.

The main reason Vector is faster than Promtail or Fluentd is that it was designed to be high-performance and low-latency tool for collecting and processing logs. Vector is built with modern and fast Rust language, which is known for its efficiency and low memory consumption.

As a result, Vector can process massive amounts of logs efficiently. Additionally, Company X development team is using Rust language frequently for internal tool set, therefore they can modify the code if needed. Furthermore, Vector provides a big variety of options for data output such as Vector itself to be used as separated collector and aggregator or Apache Kafka to attach even more log consumers to it.

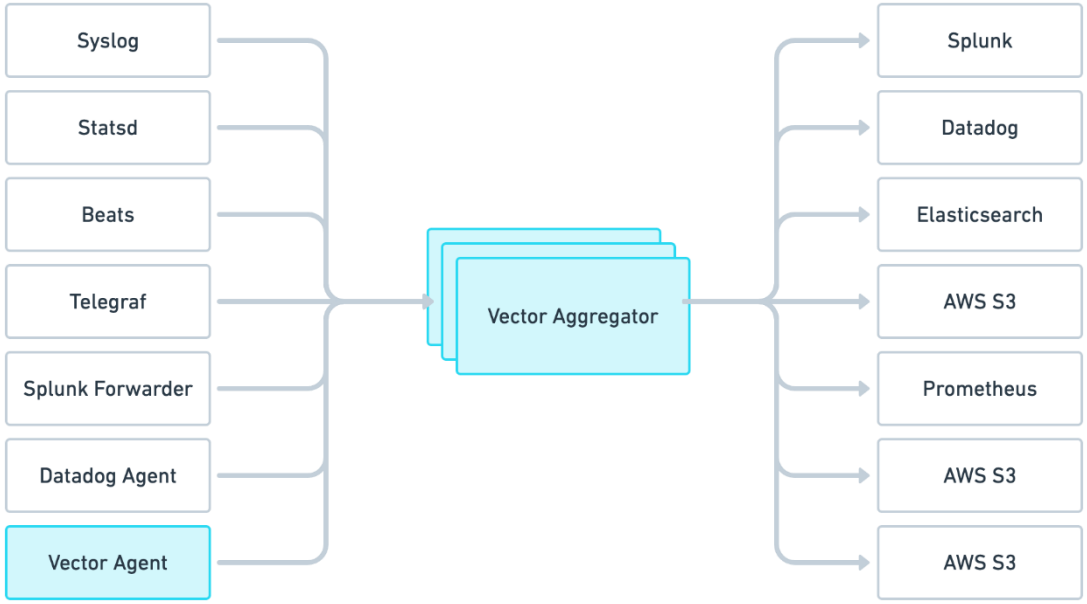

Figure 2. Vector deployment options (Source: [11])

The comparison table of log collectors and processors was created to show main differences between the options:

Table 2. Log collectors' comparison

Name	Purpose	PL	Dependencies
Fluentd	Log collector, processor, and aggregator	C and Ruby	Ruby Gems
Fluentbit	Processor and Forwarder	C	None
Promtail	Build-in log agent for Loki	Go	Loki
Vector	Collect, transform, and route all logs	Rust	None

Based on various reviews, the author found that Vector usually provides better performance, especially in heavy workload conditions. Some reviews use Logs Per Second metric which shows how many logs can be processed in a second, based on the data provided, Vector offers 2 times better performance in comparison with Fluentbit and around 4 times better than Fluentd with less memory consumption. [12] [13] [14]

Furthermore, Vector uses Adaptive Request Concurrency feature by default which automatically optimizes HTTP (Hypertext Transfer Protocol) concurrency limits based on downstream service responses to improve performance and reliability. [15]

4.4 Security

There is no defined security policy about encryption of connections between nodes. However, the SOC team asked the author to implement encryption for logs travelling between logging instances.

TLS (Transport Layer Security) cryptographic protocol is the most common way for such cases, and it can be used on all the stages without any problems, Grafana, Loki and Vector support it natively. It is the protocol used to secure communications over the internet. TLS is used to encrypt data transmissions between servers or clients to prevent unauthorized access and data theft. It is commonly used with secured HTTPS (Hypertext Transfer Protocol Secure) connections.

Internally developed tools do not have the option to use TLS by default, but as we need it for syslog transmissions this option can be added with Python library: "rfc5424-logging-handler", which adds TLS enabled handler to standard syslog library. [16]

Self-signed TLS certificates can be used, they should not cause any security problems in this case as the production environment is in internal network and is not exposed to the internet directly.

4.5 Infrastructure as Code

IaC tools must be used to provide a consistent and repeatable way to build the whole setup for the experimental and testing purposes, and later in the production environment to improve scalability of the deployment. Provided Infrastructure as Code solution must provide a consistent and automated option to deploy and manage infrastructure resources, which can increase the quality of a services provided by a Company X by reducing manual efforts and ensuring that whole logging infrastructure is deployed consistently across all the environments.

5 Implementation

This chapter provides an overview of the logging solution implementation workflow.

5.1 DNS

Internal resources of Company X do not have a DNS (Domain Name System) server. Therefore, static DNS records can be used to avoid regeneration of the certificates, records can be added separately to each docker container by “etc_hosts” parameter. This option let us use the same certificates with the same SAN (Subject Alternative Name) DNS records in them.

All the DNS entries should have the same format containing name of the host and domain part: “.companyx.internal”. Domain name “.internal” was used instead of “.local” due to potential problems with Multicast DNS as per RFC6762 Appendix G [17].

5.2 Certificates

Self-signed TLS certificates are used to encrypt connections between all the steps in chain: Jump Host – Vector – Loki – Grafana. Grafana, Loki and Vector support TLS certificates by default, therefore no additional changes are needed there, only to enable TLS mode and to set the required certificate and key files info configurations.

For the ease of use the following bash scripts were created to generate required CA (Certificate Authority) and individual keys and certificates in a more convenient way:

```
#!/usr/bin/env bash

while getopts s:e:a: flag
do
    case "${flag}" in
        s) subject=${OPTARG};;
        e) expiration=${OPTARG};;
        a) addext=${OPTARG};;
        *) echo "Usage: $0 [-s] [-e] [-a]" >&2
               exit 1 ;;
    esac
done

set -e

ca_cert="ca.crt"
ca_key="ca.key"

echo "Subject: $subject";
echo "Expiration: $expiration";
echo "AddExt: $addext";

# Generate private key
openssl genrsa -out $ca_key 2048

echo "Generating root certificate..."
echo "Subject: ${subject}"
echo "Expiration: ${expiration} days"

# Generate root certificate
openssl req -x509 -new -nodes -subj "$subject" -addext "$addext" \
       -key $ca_key -sha256 -days "$expiration" -out $ca_cert

chmod 644 ca_key

echo -e "Success!"
echo "The following files have been written:"
echo -e "  - $ca_cert"
echo -e "  -  $ca_key"
```

Figure 3. Script to generate CA

```
#!/usr/bin/env bash

while getopts c:s:e:a: flag
do
    case "${flag}" in
        c) client=${OPTARG};;
        s) subject=${OPTARG};;
        e) expiration=${OPTARG};;
        a) addext=${OPTARG};;
        *) printf "Usage: %s [-s] [-e] [-a]\n""$0" >&2
               exit 1 ;;
    esac
done

set -e

ca_cert="ca.crt"
ca_key="ca.key"

echo "Client: $client";
echo "Subject: $subject";
echo "Expiration: $expiration";
echo "AddExt: $addext";

openssl genrsa -out "$client.key" 2048

openssl req -new -subj "$subject" -addext "$addext" -key "$client.key" \
       -out "$client.csr"

cat > "$client.ext" <<-EOF
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth, clientAuth
$addext
EOF

openssl x509 -req -in "$client.csr" -extfile "$client.ext" -CA "$ca_cert" \
    -CAkey "$ca_key" -out "$client.crt" -days "$expiration" -sha256

rm "$client.csr"
rm "$client.ext"

chmod 644 "$client.crt"
chmod 644 "$client.key"
```

Figure 4. Script to generate client certificates

5.3 IaC

Ansible was integrated with Terraform to provide all the variables for Ansible on the initialization and deployment steps dynamically. The simplest solution to integrate Terraform with Ansible is using “local-exec” provisioner to run ansible playbook as a plain command on local machine. However, despite its simplicity, this solution has a lot of potential drawbacks that should be considered. The most important problem is that this way provides limited error handling, this means that in case of any playbook failure, Terraform may not be able to handle the problem properly and the state of the managed resources may be left uncertain. Furthermore, the code of the whole solution becomes unclear with all the additional shell commands and parameters.

Therefore, official Ansible provider for Terraform and Terraform Collection for Ansible were used for this purpose to provide a seamless experience and better integration with both tools. [18]

All this allows to run the entire process with only 3 commands:

- terraform init
- terraform apply -var-file=do_main.tfvars
- ansible-playbook -i ansible/inventory.yaml ansible/infra.yaml

The most essential parts of the Terraform configuration are provided below:

```
terraform {
  required_providers {
    digitalocean = {
      source = "digitalocean/digitalocean"
      version = "~> 2.0"
    }
    ansible = {
      version = "~> 1.0.0"
      source  = "ansible/ansible"
    }
  }
}
```

Figure 5. Terraform providers

```
resource "digitalocean_droplet" "vector" {
  image    = "centos-stream-9-x64"
  name     = "vector.company.internal"
  region   = "lon1"
  size     = "s-1vcpu-1gb"
  ssh_keys = [digitalocean_ssh_key.def_pk.fingerprint]


  provisioner "remote-exec"{
    inline = [
      "echo Vector server is running!",
    ]

    connection {
      host        = self.ipv4_address
      user        = "root"
      type        = "ssh"
      private_key = file(var.do_private_key)
      timeout     = "2m"
    }
  }
}
```

Figure 6. Terraform DO resource

This resource part also includes “remote-exec” provisioner to make sure that host is created before exiting the program.

```
resource "ansible_host" "vector-0" {
  name   = digitalocean_droplet.vector.name
  groups = ["vector"]
  variables = {
    ansible_host = digitalocean_droplet.vector.ipv4_address,
    ansible_user = "root",
    ansible_ssh_private_key_file = var.do_private_key,
    ansible_ssh_extra_args = "-o StrictHostKeyChecking=no"
  }
}
```

Figure 7. Terraforn Ansible resource

“ansible_host” resource provides all the variable values to the ansible.

```
resource "ansible_group" "vector" {
  name      = "vector"
  variables = {
    vector_port = var.vector_port,
    loki_port = var.loki_port,
    vector_image = var.vector_image
  }
}
```

Figure 8. Terraform Ansible group

The latest resource part provides a group to ansible to assign each separate similar instances to one group, also it some important variable values to it.
Provided configuration snippets are the same for Loki, excluding some variable values.

5.4 Digital Ocean

Digital Ocean was taken as a testing platform due to its simplicity and cheapness compared to AWS. Three separate Ubuntu 22.04 Linux instances were created in distinct regions (LON, AMS and FRA) to simulate various locations for production environment. All the instances, except Grafana, were created using only Terraform and provisioned with Ansible. Each machine runs docker containers with appropriate images. The Loki host is two times more powerful than other ones, for Grafana and Vector, as Loki requires more resources for fast operation.

5.5 Vector

Vector supports a set of different sources and sinks for receiving and transmitting the data, it allows the use of Syslog as a log source and Loki as a sink natively without any additional plugins. For its configurations, it can use different file types of its configuration files: YAML, TOML (Tom's Obvious Minimal Language) and JSON (JavaScript Object Notation), but it is recommended to stick with TOML, as it was used from the start of a Vector development process, and it has better support from the community and developers.

For data transformation purposes Vector provides multiple transform options including VRL (Vector Remap Language) to define event transformation logic. VRL offers a wide range of data-specific functions that map directly to the desired use cases. The VRL compiler also performs multiple compile-time checks for the provided code to ensure that

it is correct and does not contain any unhandled errors. As the main purpose is to process the nginx logs which have the same format all the time, the remap transform function could be used with regex (regular expression) specific rules for proper separation and aggregation of the log's fields.

Nginx provides two main types of logs: access and error logs, therefore two main regex rules should be used to process them separately due to different formats. Access and error logs must be separated before the regex processing steps, it can be implemented with filter transformation function which should check for a log's severity field options and forward each log entry to the desired point. Main remap function should also remove unused fields and unnest additional syslog fields as they have more information about the tenant and source of the log entry to put them together with main syslog message field.

The last step is to forward all processed logs entries to the Loki instance. This task can be done with Loki sink module which also supports TLS protocol and Syslog source module. Loki labels could be set up manually or be taken from each syslog entry automatically.

To accomplish the tasks the following ansible configuration template was written:

```
[sources.syslog]
type = "syslog"
mode = "tcp"
address = "0.0.0.0:{{ vector_port }}"
tls.enabled = true
tls.crt_file = "/etc/vector/certs/vector.crt"
tls.key_file = "/etc/vector/certs/vector.key"
tls.ca_file = "/etc/vector/certs/ca.crt"
tls.verify_certificate = true
tls.verify_hostname = true
```

Figure 9. Vector syslog source

Syslog source collects all the logs going from outside to the specified port.

```
[transforms.nginx_filtered]
type = "filter"
inputs = ["syslog"]
condition = '.appname == "nginx"'

[transforms.nginx_access_filtered]
type = "filter"
inputs = ["nginx_filtered"]
condition = '.severity == "info"'

[transforms.nginx_err_filtered]
type = "filter"
inputs = ["nginx_filtered"]
condition = '.severity == "err"'
```

Figure 10. Vector filters

Logs are being filtered by application name and Nginx log types. This is made to avoid parsing unwanted logs going with wrong syntax or from incorrect application.

```
[transforms.nginx_access_processed]
inputs = ["nginx_access_filtered"]
type = "remap"
source = '''
.message = parse_regex!(.message, r'^(?P<source_ip>\d+\.\d+\.\d+\.\d+) \-
(?P<user>-|[a-z_][a-z0-9_]{0,30}) \[(?P<log_timestamp>[^]]+)]
"(?P<method>GET|POST|HEAD|PUT|DELETE|CONNECT|OPTIONS|TRACE|PATCH)
(?P<request_uri>/[^\s]*) (?P<http_version>HTTP/\d\.\d)" (?P<status_code>\d{3})
(?P<body_bytes_sent>\d+) "(?P<server>[^\s]+)" "(?P<user_agent>[^"]+)"
"(?P<forward_for>[^"]+)" "(?P<request_id>[a-z0-9]+)" "(?P<ssl_protocol>[^\s]+)"
"(?P<ssl_cipher>[^\s]+)" "(?P<upstream_addr>[^\s]+)"
"(?P<upstream_resp_time>[^\s]+)" "(?P<upstream_status>[^\s]+)"$')

.customer_name = get!(value: ."additional@0", path: ["customer_name"])
.sensor_name = get!(value: ."additional@0", path: ["sensor_name"])
.cloud_provider = get!(value: ."additional@0", path: ["cloud_provider"])
.log_timestamp = to_timestamp!(get!(value: .message, path: ["log_timestamp"]))

.int_status_code = to_int!(get!(value: .message, path: ["status_code"]))

if .int_status_code >= 500 && .int_status_code < 600 {
    .level = "critical"
} else if .int_status_code >= 400 && .int_status_code < 500 {
    .level = "error"
} else if .int_status_code >= 300 && .int_status_code < 400 {
    .level = "warning"
} else {
    .level = "info"
}

.additionals = {"log_timestamp": .log_timestamp, "type": "access",
"customer_name": .customer_name, "sensor_name": .sensor_name, "cloud_provider":
.cloud_provider, "level": .level}

. = merge(.message, .additionals)
'''
```

Figure 11. Vector Nginx access log transform

On the transform module logs are being parsed by specific regex rule and some additional fields are added. Access logs does not have any log level by default; therefore, it is going to be added by checking the HTTP response codes.

```
[transforms.nginx_error_processed]
inputs = ["nginx_err_filtered"]
type = "remap"
source = '''
.message = parse_regex!(.message, r'^(?P<log_timestamp>.+) \[(?P<level>\w+)]
(?P<pid>\d+#\d+): \*(?P<tid>\d+) (?P<cid_message>[^,]+), client:
(?P<client_ip>\d+\.\d+\.\d+\.\d+), server: (?P<server>[^,]+)(?:, request:
"(?P<method>GET|POST|HEAD|PUT|DELETE|CONNECT|OPTIONS|TRACE|PATCH)
(?P<request_uri>/[^\s]+) (?P<http_version>HTTP/\d\.\d)")?(?:, upstream:
"(?P<upstream_url>[^"]+)")?(?:, host: "(?P<host>[^"]+)")?(?:, refferer:
"(?P<refferer>[^"]+)")?$')

.customer_name = get!(value: ."additional@0", path: ["customer_name"])
.sensor_name = get!(value: ."additional@0", path: ["sensor_name"])
.cloud_provider = get!(value: ."additional@0", path: ["cloud_provider"])
.log_timestamp = to_timestamp!(replace!(get!(value: .message, path:
["log_timestamp"]), "/", "-"))

.additionals = {"log_timestamp": .log_timestamp, "type": "error", "customer_name":
.customer_name, "sensor_name": .sensor_name, "cloud_provider": .cloud_provider}
. = merge(.message, .additionals)
'''
```

Figure 12. Vector Nginx error log transform

```
[sinks.loki]
type = "loki"
inputs = [ "nginx_access_processed", "nginx_error_processed" ]
endpoint = "https://loki.threatx.internal:{{ loki_port }}"
encoding.codec = "json"
tls.crt_file = "/etc/vector/certs/loki.crt"
tls.key_file = "/etc/vector/certs/loki.key"
tls.ca_file = "/etc/vector/certs/ca.crt"
tls.verify_certificate = true
tls.verify_hostname = true
healthcheck.enabled = false

[sinks.loki.labels]
service = "sensor"
app = "nginx"
type = "{{ type }}"
customer_name = "{{ customer_name }}"
server = "{{ server }}"
sensor_name = "{{ sensor_name }}"
```

Figure 13. Vector Loki sink

Loki sink module sends logs next to the Loki host, providing required headers.

The provided configuration uses multiple filter functions to ensure that logs are going from Nginx and to separate them before processing with regex function. The filters and regex rules were tested multiple times with a lot of different kinds of logs, the code was optimized and all the found issues were solved to parse all the correct log records properly. Incorrect log entries are being skipped as they do not contain any useful information.

The configuration was written with help of official Vector documentation and tested with VRL Playground as it has specific regex syntax which cannot be properly tested with common tools like regex101 or RegExr.

5.6 Tool Set

As mentioned before the internal tool set should be used to forward logs from sensor instances to Vector host. Tools are integrated with all the production and development systems through a set of internal jump hosts which provide SSH tunnels to allow secure connection to remote and internal resources for management and debugging purposes. This channel can be used for sending the logs from Nginx on the sensors to Vector.

Currently log collection can be triggered only manually and all the logs are saved to the directory on the local machine of the tool user, therefore this function can be reused to forward logs back to the jump host and to the Vector instance. The new function should send logs as syslog messages to the Vector using TLS protocol for encryption, so the additional Python library was used: “rfc5424-logging-handler”.

The following class was written to provide log forwarder functionality:

```
import logging
from socket import SOCK_STREAM

from rfc5424logging import Rfc5424SysLogHandler

class VectorLogsSender:
    def __init__(self, address: tuple, utf8: bool = False, enterprise_id: str =
"0"):
        self.address = address
        self.utf8 = utf8
        self.enterprise_id = enterprise_id

        self.logger = logging.getLogger('nginx')
        self.handler = Rfc5424SysLogHandler(
            address=self.address,
            msg_as_utf8=self.utf8,
            enterprise_id=self.enterprise_id,
            socktype=SOCK_STREAM,
            tls_enable=True,
            tls_verify=True,
            tls_client_cert="/vector.crt",
            tls_client_key="/vector.key"
            )

        self.logger.addHandler(self.handler)

    def send_logs(self, log_type: str, log_list: list, extra: dict):
        for log in log_list:
            if log_type == "error":
                self.logger.setLevel(logging.ERROR)
                self.logger.error("%s", log, extra=extra)
            else:
                self.logger.setLevel(logging.INFO)
                self.logger.info("%s", log, extra=extra)
```

Figure 14. Vector Python class code

Additional function was added to tool set to provide new command for sending logs to the Vector instance:

```
for sensor_name in sensor:
            selected_sensor = sensorops.find_one_sensor(sensor_name)
            if not selected_sensor:
                logger.warn(f"Sensor was not found: {sensor_name}")
                continue

            sensor_instance = SensorConnFactory().create_sensor(
                selected_sensor, ctx.tty_jump)

            with console.status(f"Fetching contents of log file '{rfile}' from
{sensor_name}."):
                try:
                    for log_list_gen in sensor_instance.get_logs(rfile):
                        logs_list = log_list_gen.split("\n")

                        customer_name = selected_sensor.get("customer_name")
                        cloud_provider = selected_sensor.get("cloud_provider")

                        extra_info = {
                            "structured_data": {
                                "additional": {
                                    "sensor_name": sensor_name,
                                    "customer_name": customer_name,
                                    "cloud_provider": cloud_provider
                                }
                            }
                        }

                        vector = VectorLogsSender(('localhost', 9000))
                        vector.send_logs(log_type, logs_list, extra_info)

                except Exception as err:
                    logger.warn(err.__str__())
                    logger.debug("Error", exc_info=err)
                finally:
                    logger.info(f"Output has been sent.")
```

Figure 15. Tool set Python code snippet

In this example the localhost port is forwarded by another internal function to the Vector host located behind the jump host.

5.7 Grafana and Loki

Loki does not require any specific configuration to work with Vector, therefore the typical "local" configuration example was used with some changes to allow TLS and to increase performance of the Loki instance. As the current implementation does not have any external DB (Database) like Cassandra or AWS S3 to use and 24h logs period is enough, the TSDB (Time Series Database) as a new database option introduced in Loki version 2.8 option is used to store data locally with "filesystem" object storage without any additional dependencies. [19]

Also, some values of variables like "frontend.max_outstanding_per_tenant" and "query_scheduler.max_outstanding_requests_per_tenant" were increased to avoid "too many outstanding requests" issue which could appear when Grafana dashboard generates too many queries with data panels.

Two separate Grafana dashboards were created to provide all the essential information about logs and allow to examine in a better way:

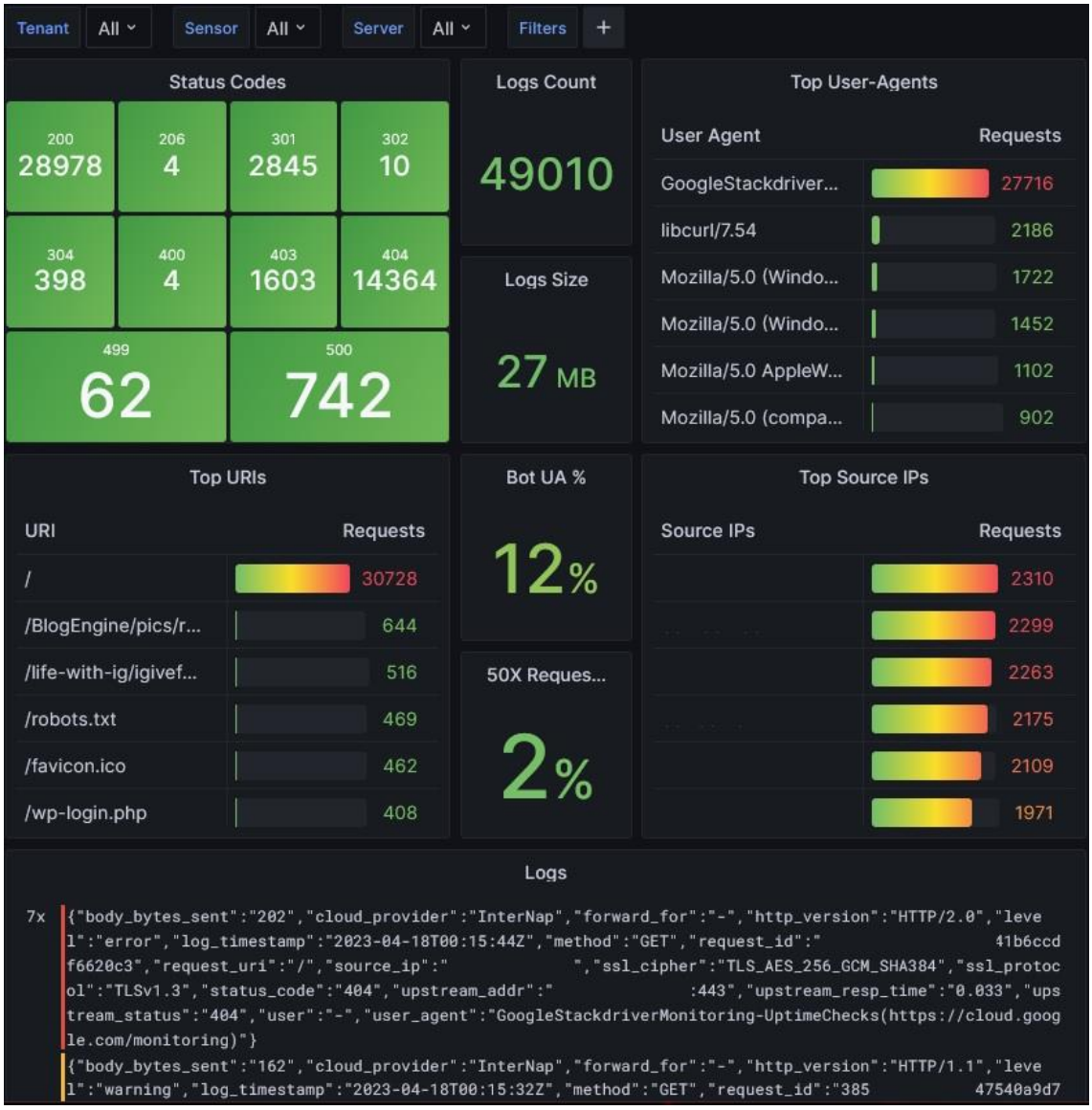

Figure 16. Grafana dashboard for Nginx access logs

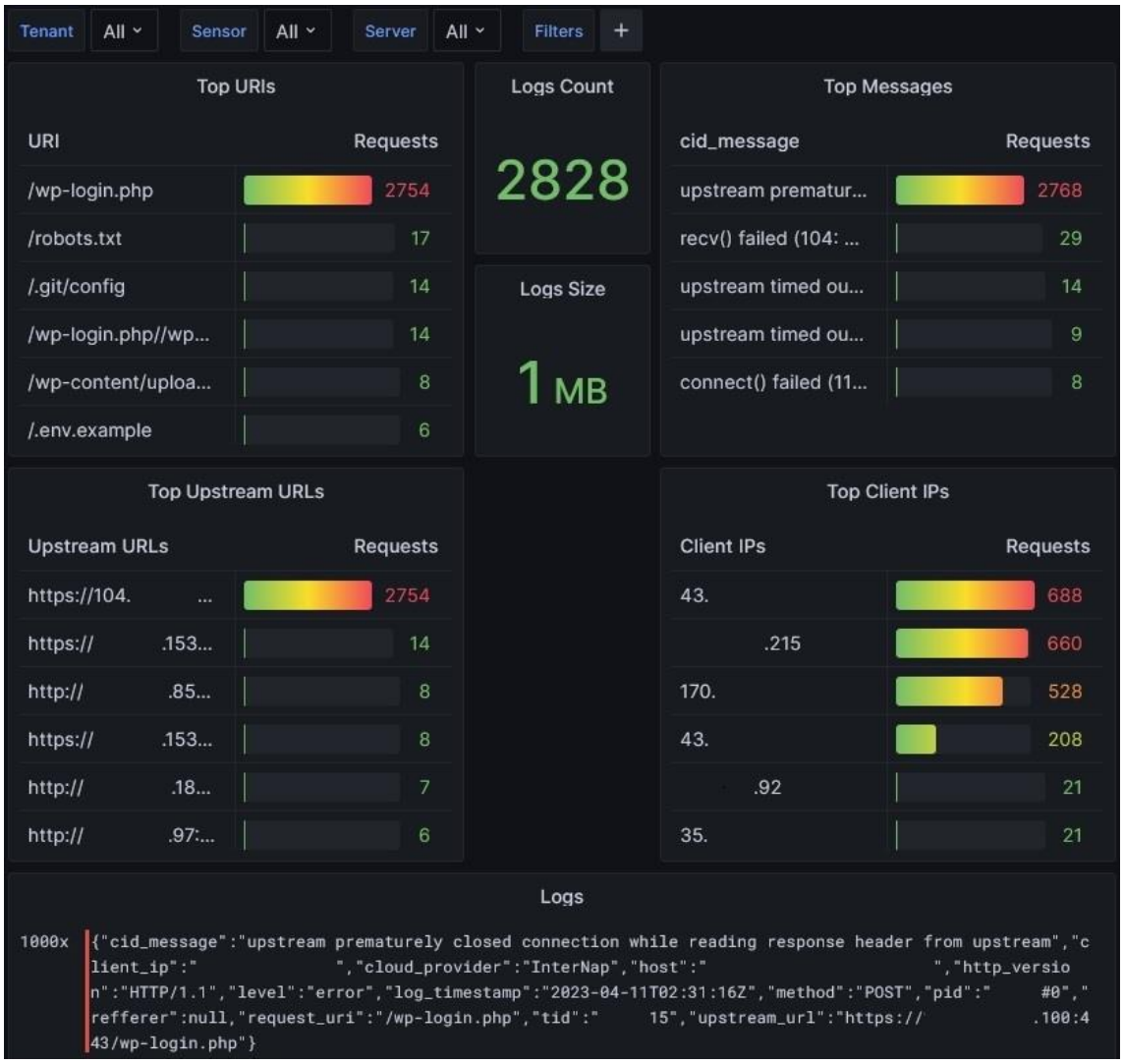

Figure 17. Grafana dashboard for Nginx error logs

As TLS certificates are used, the Grafana Loki data source was configured with additional Auth parameters enables and TLS/SSL (Secure Sockets Layer) Auth Details provided with certificates, keys, and ServerName of the Loki host.

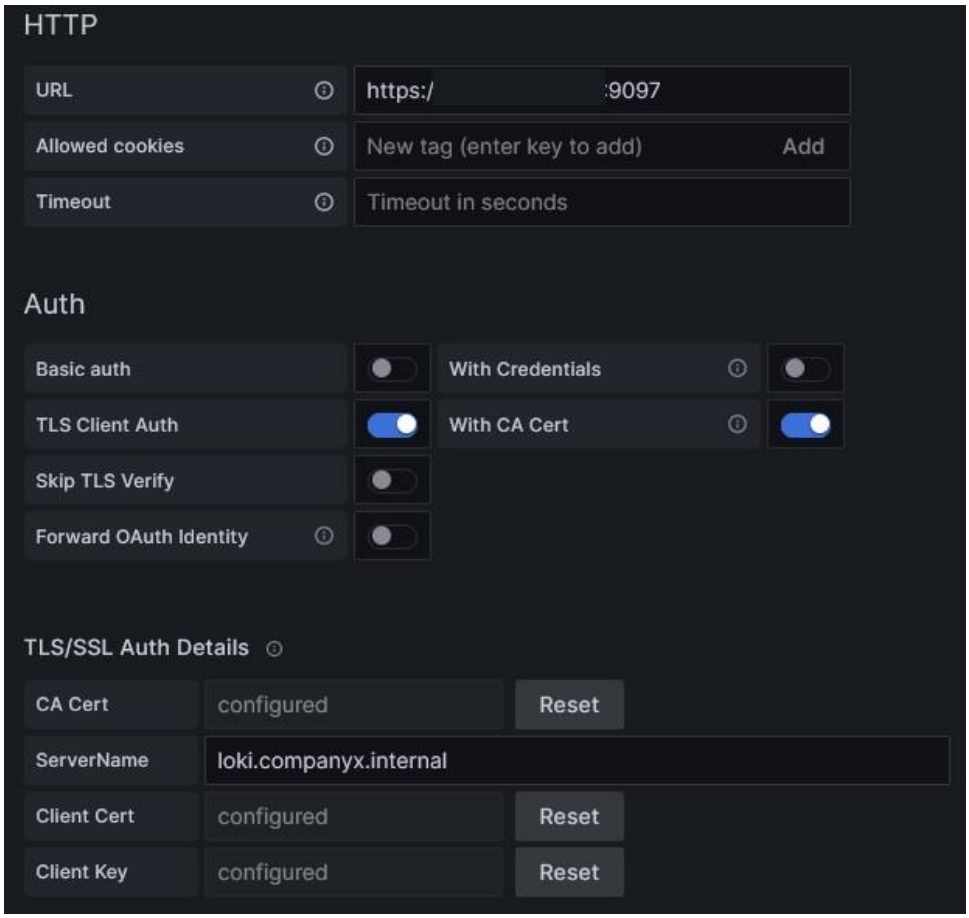

Figure 18. Grafana Loki data source settings

6 Review of the implemented setup

The implemented setup consists of three hosts based on Digital Ocean droplets service. Loki and Vector hosts are deployed using developed Terraform and Ansible playbooks, Grafana is standalone host as it does not require any specific changes and the internal Grafana host can be used instead.

Vector and Loki hosts are running docker containers with corresponding images and specific Jinja2 configuration templates provided by Ansible.

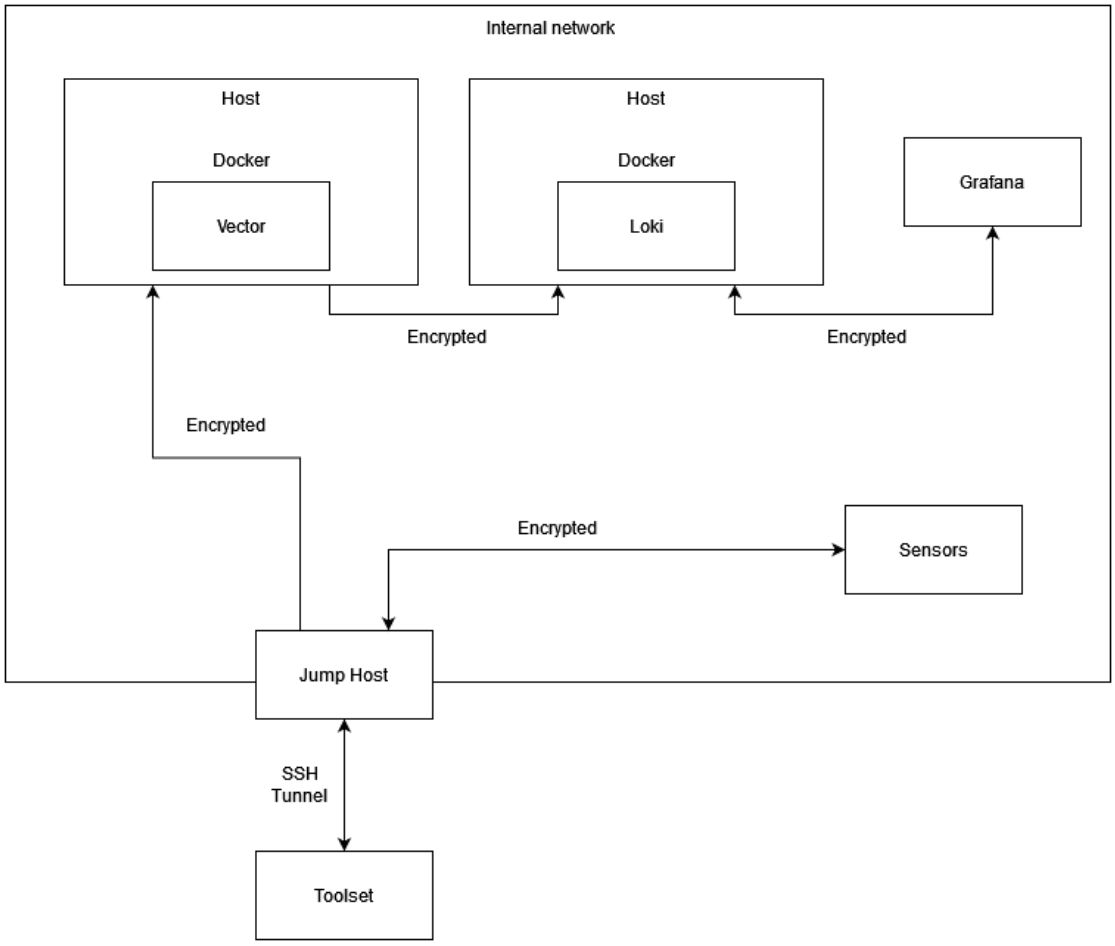

Figure 19. Implemented setup scheme

7 Conclusion

The main goal of this thesis was to find and build a proper logging solution for Company X. By analyzing available options and conducting experiments to find the most suitable solution, the author deems that this goal has been met. The author has fulfilled all the requirements and implemented the working solution using the testing environment. The solution was successfully integrated with existing tools and can already be used by an SOC team to improve their services quality. Some parts and principles of the created solution can be reused by other individuals or companies with similar environments to implement logging solutions reusing the existing tools.

Limitations of this research are related with the development and implementation of new features; the following factors limited the author's scope of work:

- Agreements: Company X has hierarchical structure as any other company, and it means that any significant changes, like changing the Sensor image, can't be made without any agreement on multiple levels like software developers and infrastructure engineers. This slowed down the whole process of implementation and significantly narrowed the variety of available options to and how to implement.

- Testing: Testing is an important phase of the implementation of any new solution, and it should be done properly to avoid any further problems like incompatibility with the existing environment.

- New tools: The author did not have much experience with tools like Vector, Loki, Digital Ocean, and Terraform, therefore the author had to learn from scratch how these tools work and how to configure them to make the solution work.

Future work in this topic can be focused on providing a more comprehensive view of the Sensors behavior and allow SOC team to investigate problems even faster. Proposed improvements are listed as follows:

- Move from Digital Ocean to production AWS hosts

- Implement high availability options for AWS installation using the Consul or ECS (Elastic Container Service) tools
- Add Vector as collector to each Sensor image to collect logs directly in real-time without any additional triggers from SOC team members
- Add more and improve existing Grafana dashboards

References

[1] A. Deveriya, "Using Syslog," in *Network Administrators Survival Guide* , Cisco, 2005, pp. 181-223.

[2] Nginx, "What Is NGINX?," [Online]. Available: https://www.nginx.com/resources/glossary/nginx/. [Accessed 18 April 2023].

[3] D. Tiede, "Big-Data Solutions for Manufacturing Health Monitoring and Log Analytics," 23 August 2022. [Online]. Available: https://d-nb.info/1272863476/34. [Accessed 19 April 2023].

[4] Oracle, "High Availability Overview and Best Practices," March 2023. [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/19/haovw/high-availability-overview-and-best-practices.pdf. [Accessed 18 April 2023].

[5] D. Manoor, "AWS or DigitalOcean - Which cloud platform is the best fit for you?," DigitalOcean, 8 December 2022. [Online]. Available: https://www.digitalocean.com/blog/aws-vs-digitalocean-cloud-platform. [Accessed 18 April 2023].

[6] J. D. Mooney, "Developing Portable Software," [Online]. Available: https://link.springer.com/content/pdf/10.1007/1-4020-8159-6_3.pdf. [Accessed 18 April 2023].

[7] K. Morris, Infrastructure as Code: Dynamic Systems for the Cloud Age, O'Reilly Media, Inc., 2021.

[8] HashiCorp, "What is Terraform?," [Online]. Available: https://developer.hashicorp.com/terraform/intro. [Accessed 18 April 2023].

[9] Red Hat, "Ansible vs. Terraform, clarified," 28 September 2022. [Online]. Available: https://www.redhat.com/en/topics/automation/ansible-vs-terraform. [Accessed 18 April 2023].

[10] A. I. C. S. S. Steve Strutt, "End-to-End Application Provisioning with Ansible and Terraform," IBM, 21 November 2018. [Online]. Available: https://www.ibm.com/cloud/blog/end-to-end-application-provisioning-with-ansible-and-terraform. [Accessed 18 April 2023].

[11] Vector, "Deployment," [Online]. Available: https://vector.dev/docs/setup/deployment/. [Accessed 18 April 2023].

[12] E. R. Ajay Gupta, "Who is the winner — Comparing Vector, Fluent Bit, Fluentd performance," IBM Cloud, 9 September 2021. [Online]. Available: https://medium.com/ibm-cloud/log-collectors-performance-benchmarking-8c5218a08fea. [Accessed 18 April 2023].

[13] Vector, "Vector. Comparisons: Performance," [Online]. Available: https://github.com/vectordotdev/vector#performance. [Accessed 18 April 2023].

[14] СберМегаМаркет, "Как мы искали свой Vector в построении высоконагруженной системы логирования," 1 November 2022. [Online]. Available: https://habr.com/ru/companies/sbermegamarket/articles/696844/. [Accessed 18 April 2023].

[15] Vector, "Adaptive request concurrency (ARC)," [Online]. Available: https://vector.dev/docs/about/under-the-hood/networking/arc/. [Accessed 18 April 2023].

[16] J. Beckers, "Python rfc5424 syslog logging handler," 2020. [Online]. Available: https://rfc5424-logging-handler.readthedocs.io/en/latest/. [Accessed 18 April 2023].

[17] M. K. A. I. S. Cheshire, "Multicast DNS," February 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6762#appendix-G.

[18] Ansible, "Terraform Provider for Ansible," [Online]. Available: https://github.com/ansible/terraform-provider-ansible. [Accessed 18 April 2023].

[19] G. L. Team, "Grafana Loki 2.8 release: TSDB GA, LogQL enhancements, and a third target for scalable mode," 6 April 2023. [Online]. Available: https://grafana.com/blog/2023/04/06/grafana-loki-2.8-release-tsdb-ga-logql-enhancements-and-a-third-target-for-scalable-mode/. [Accessed 18 April 2023].

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Nikita Ratškov

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Centralized Logging System for a Multi-Tenant Environment”, supervised by Mohammad Tariq Meeran
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

21.04.2023

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.