LAB University of Applied Sciences

Implementation of log collection pipeline and visualization for a CI/CD environment

LAB University of Applied Sciences
Bachelor of Engineering, Information and Communications Technology
2022
Henri Vatto

Abstract

Author(s)	Publication type	Completion year
Henri Vatto	Thesis, UAS	2022
	Number of pages	
	24	

Title of the thesis

Implementation of log collection pipeline and visualization for a CI/CD environment

Degree, Field of Study

Bachelor's Degree Programme in Information and Communications Technology

Organisation of the client

GE Healthcare Finland Oy

Abstract

The goal of the thesis was to create a log collection pipeline for a continuous integration environment. The unified logging layer would collect logs from servers running containerised software tests on embedded Linux devices. This data would be collected to a database and the data would be visualised on a web-based user interface. The project was conducted for GE Healthcare Finland, which is a medical technology company specialized in patient monitoring.

EFK stack is a collection of tools made for collecting, monitoring, and analysing log data. The stack consists of Elasticsearch for storing and indexing the data, Fluentd for aggregating the logs, and Kibana for visualization.

The EFK stack was implemented by containerizing all the parts of it for easier distribution and deployment. The logs were collected from systemd journal and sent to Fluentd through systemd journal upload. Elasticsearch was configured to run as a single node cluster with daily creation of indices. The Kibana dashboard was created with visualizations displaying information on the latest errors and warnings as well as general information of the test side logs.

The log collection pipeline was set up as planned and test servers were able to send their logs directly to it. As the pipeline performs adequately on the continuous integration testing environment, it can be extended to collect logs from the build systems as well.

Keywords

Elasticsearch, Fluentd, Kibana

Contents

1 Introduction

As the infrastructure and the number of machines grows so do the number of logs generated by them. Managing these logs can be a challenging task if there is no centralised system in place for them. If a programmer wants to know why their software tests are stuck and wants to receive logs related to this, having to manually search through different hosts chasing that specific log file with the required data can be a hassle. Also picking up on these errors ahead of time, before users start to complain the broken continuous integration (CI), is vital for keeping a healthy infrastructure.

GE Healthcare Finland Oy is a subsidiary of GE Healthcare which is part of General Electric. It specialises in producing and developing patient monitoring devices. In Finland, the company has two sites: Helsinki site and Kuopio site. While the Kuopio site only focuses on the clinical information systems of anesthesia and intensive care unit, the Helsinki site has a more diverse assortment of operations such as assembly lines, product and software development, and maintenance all for patient monitoring solutions. GE Healthcare Finland employs around 700 employees between the two sites. The sales of the company reached 197.5 million euros in 2021

The goal of this thesis is to create a pipeline where logs can be collected to a single centralised server and to visualize the data in a way that provides essential information about the health of the CI infrastructure. The main tools to be used in this project are Fluentd, Elasticsearch and Kibana. These tools are chosen because of their flexibility and efficiency. All the software running on the logging server will be containerized. The logs will be collected from the test servers running Schedy.

2 Logging Systems

2.1 The Fundamentals of Logs

Logs are reports generated from processes running on the operating system. Reports contain information about the event. The information includes valuable information, such as the user, group, syslog facility and the time when the logging occurred. Analysing these logs gives information on what went wrong in cases when problems occur. (Loggly.)

Logs are also important in continuous integration environments. When a build fails or test start to produce irregular results, there can be problems hiding in the backend. Finding the problematic services or applications using logs is easy when you know where to look. When the logs are spread around multiple hosts in a distributed system, finding the correct log can take time. (Assaraf 2022.)

2.2 Syslog

Syslog is a protocol for sending logs. It is the old way of creating log files. The common implementations, such as rsyslog, generate log files as normal text files. These files lack structure causing finding information from them to be slow. (Schäfer 2016.)

2.3 Systemd journal

Systemd is a part of most modern Linux distributions. It handles the system settings and service management. Systemd handles most of the events running on the system and as a result produces its own logs. Systemd journal, or journald, is the logging system used by systemd. Journald was created as a replacement for Syslog, and being part of systemd, is the default logging system on many Linux distributions. (Schäfer 2016.)

Since Syslog is missing some important functionality, journald was created to implement these features which include structured messages, indexed logs and access control. The structured message files are created in a binary format with the message itself being in plain text. The structured and indexed messages allow for fast searches and filtering. While being the Syslog replacement, journald retains full Syslog compatibility since it provides the full Syslog API. (Systemd.) Figure 1 displays the contents of a log from the system journal using the journalctl utility. The log is taken from a failed authentication attempt. The _COMM field shows the name of the process where the journal entry came from, MESSAGE field prints out a human-readable message of the log, and PRIORITY field represents the severity of the log message, 5 standing a notice.

```
Sun 2022-11-13 21:43:25.546139 EET [s=eddfbc7cd31048b2b57336556e304548;i=f4559;
b=66b43996e8874d15a5ffdb8fd45d4474;m=6ed69563;t=5ed5f5616871e;x=f330cbc4a01536fb]
    _UID=1000
    _GID=1000
    _SELINUX_CONTEXT=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
    _AUDIT_SESSION=3
    _AUDIT_LOGINUID=1000
    _SYSTEMD_OWNER_UID=1000
    _SYSTEMD_UNIT=user@1000.service
    _SYSTEMD_SLICE=user-1000.slice
    _MACHINE_ID=9c4bc5eb4f19497c9e136ffc6bffc82d
    _HOSTNAME=fedora
    _TRANSPORT=syslog
    PRIORITY=5
    _BOOT_ID=66b43996e8874d15a5ffdb8fd45d4474
    SYSLOG_FACILITY=10
    SYSLOG_IDENTIFIER=sudo
    _COMM=sudo
    _EXE=/usr/bin/sudo
    _CMDLINE=sudo su
    _CAP_EFFECTIVE=1ffffffffff
    _SYSTEMD_USER_SLICE=app-org.gnome.Terminal.slice
    SYSLOG_TIMESTAMP=Nov 13 21:43:25
    MESSAGE=pam_unix(sudo:auth): authentication failure; logname= uid=1000 euid=0
    tty=/dev/pts/1 ruser=henrivatto rhost=  user=henrivatto
    _PID=3623
    _SYSTEMD_CGROUP=/user.slice/user-1000.slice/user@1000.service/app.slice/app-org.
    gnome.Terminal.slice/vte-spawn-6cfaf5dd-f191-4bfd-a31a-c6c36ffdd9f7.scope
    _SYSTEMD_USER_UNIT=vte-spawn-6cfaf5dd-f191-4bfd-a31a-c6c36ffdd9f7.scope
    _SYSTEMD_INVOCATION_ID=e16b9b5a8dc540ca90f66efdd52a0f40
    _SOURCE_REALTIME_TIMESTAMP=1668368605546139
```

Figure 1. Log from journald

Weaknesses of journald include the lack of remote functionality, while Syslog implementations, such as rsyslog, come with it. Journald requires a separate system for forwarding these log messages. (Gheorghe 2020.)

3 Continuous Integration and Delivery

3.1 Introduction to CI/CD

Continuous Integration and Continuous Delivery are a way of automating the stages of application development. Continuous Integration, CI, focuses on automating building and testing the code (Red Hat. a). Figure 2 shows a traditional CI/CD pipeline. As a programmer pushes code to their repository, the build system pulls these changes and downloads all the required dependencies. The build system attempts to build the code into an executable software. If the build process succeeds, the software is sent to be tested in the testing environment. The testing environment runs unit tests and integration tests on the software to gain information on if the latest changes have been integrated successfully and no unforeseen issues have emerged. Throughout this process the programmer is given feedback on all the stages of the procedure. (GitLab.)

Figure 2. Example of a traditional CI/CD pipeline

Together CI and CD increase the efficiency and effectiveness of software development. They allow programmers to focus on programming and automating the work of the operations team.

The continuous integration environment used by the GE Healthcare team makes use of Gerrit as version control, Open Build Service, known as OBS, as the build system and Schedy as their testing environment.

OBS gets notified about the changes in the version control and pulls the latest versions from Gerrit. It then compiles the code and produces binaries. If the build process succeeds, OBS creates repositories for the binaries and pushes them forwards. These repositories get listed in a file that gets sent forwards to Schedy.

3.2 Schedy

Schedy is the main tool used by the testing environment. It is a task scheduling server, which runs all the tests, called tasks, for software development. Schedy is built with Ruby on Rails framework and uses Podman for building containers. It has a front-end user interface for developers which shows the progress of all the tasks. Every time a developer wants

to run tests, they create an execution, which is a collection of tasks. The version of Schedy in use is an in-house fork of the open-source application.

The main Rails application runs on the main Schedy server while the tasks run on distributed test servers. The test servers have resources, physical devices that the software test run on connected to them. The tests can require a selection of multiple resources, so tests can get queued up waiting for a free slot to open. Dealer is part of Schedy that handles dealing tasks to test servers. Test servers generate data on their capabilities of completing tasks, which the dealer uses to deal the tasks.

The tests in the execution start with container image builder. This image builder builds the images which will be used to run the tests on. It uses information gained from the request made for creating the execution, repositories made by the software build system and a Dockerfile template. When the test container images are built, they are published in the image repository and pulled by the selected test servers. Schedy then spawns containers using the test container images and runs the tests on the resources. After the tests finish, test results get uploaded back to Schedy with other generated artifacts.

4 Software Used for Creating a Centralized Logging Server

4.1 Podman

Podman is an open-source container engine. Podman makes use of Open Container Initiative (OCI) projects. These open-source projects work towards implementing the open industry standards set by OCI. These projects include applications such as OCI runtime tools for creating OCI compliant runtime configurations, slirp4netns for creating rootless networks and Buildah for building container images. The use of Open Container Initiative standards allows Podman to be mostly a drop-in replacement for Docker. Podman is capable of running images from the Docker Hub container image library and can build standard Dockerfiles. While Podman and Docker can mostly be used interchangeably, as seen from the documentation suggesting the usage of alias-command for renaming Podman to Docker, there are some major differences between the two. (Podman.)

Podman supports running containers without elevated privileges. This means users don't need to be added to a group that allows root privileges. Docker requires these privileges because Docker uses a daemon for managing all its containers. The ability to run rootless allows for increased security. All the containers run on namespaces of the users meaning different users can run the same containers at the same time without the containers interfering with each other. (Red Hat. b.)

The daemonless architecture of Podman means instead of relying on a daemon for handling the containers, it executes and runs the containers straight on the operating system. While Docker runs on all platforms, Podman is Linux native due to usage of Linux based tools. Podman can still technically be ran on Windows through the Windows Subsystem for Linux, a virtualized Linux environment. (Red Hat. b.)

Containers are software that has been packed up to a single easily distributable and deployable images. These containers run on the container engine on the host operating system in isolated packages. Generic virtual machines require a complete operating system running inside each one of them. These operating systems require considerable amounts of resources from the host running the hypervisor. The operating systems also take time to boot up, due to having to start all the processes inside it. Figure 3 highlights the overhead caused by running normal virtual machines compared to running containers, in this case using Docker as the container engine. Containers only include the libraries and dependencies required by the software inside of them and share the operating systems kernel and host system, so containers are lightweight and fast to deploy. (Docker.)

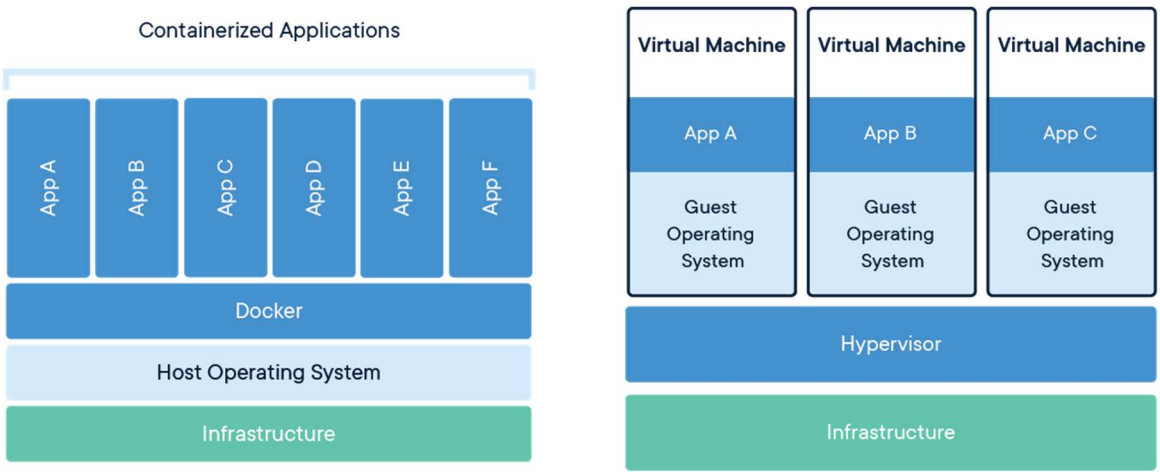

Figure 3. Container applications compared to virtual machines (Docker.com)

Containers are initiated from container images. These images are built from Dockerfiles. The Dockerfiles are text files which contain a collection of rules set for building the container image (Butler 2015). Building this file to a container image packages the application and downloads all the dependencies inside the image. Dockerfiles always use a container image as a base and append the required changes to it. Running the image executes the set rules in the container image and assembles the instance for the container. Container images can be distributed to online repositories such as Docker Hub. As a result of this, using containers is easy and the ease of distributing them makes them fast to deploy. (Docker.)

4.2 Systemd-journal-remote

Systemd journal remote is a package which includes both system journal remote and system journal upload services. These services allow transferring logs from journald over HTTP and HTTPS streams. The upload daemon attempts to send all the logs collected and stored by the journald, while the remote daemon receives the packages and stores them in the specified directory. The logs are sent over in journalds binary format.

4.3 Fluentd

4.3.1 Fluentd Operating Principals

Fluentd is a data collection software which receives and distributes data from clients. This data comes in form of logs. Fluentd is an open-source application and relies on many community-driven plugins. Fluentd is written in Ruby with the more performance centric parts written in C programming language. (Fluentd.)

Fluentd is a middleware application in log collection pipelines. It creates a unified logging layer which takes advantage of converting logs to JSON format for better machine readability and allows for better usability with different programming languages and frameworks.

Up on receiving data from a client, Fluentd analyses the data and run all the specified operations on it. The data is then automatically forwarded to given hosts. This automatic handling and forwarding are the main advantages of a unified logging layer. Figure 4 shows this automatic data handling with its input and output plugins. Fluentd supports a concept where log data is made for machines instead of users, since machines are capable of dissecting log data much more efficiently than humans. (Tamura 2014.)

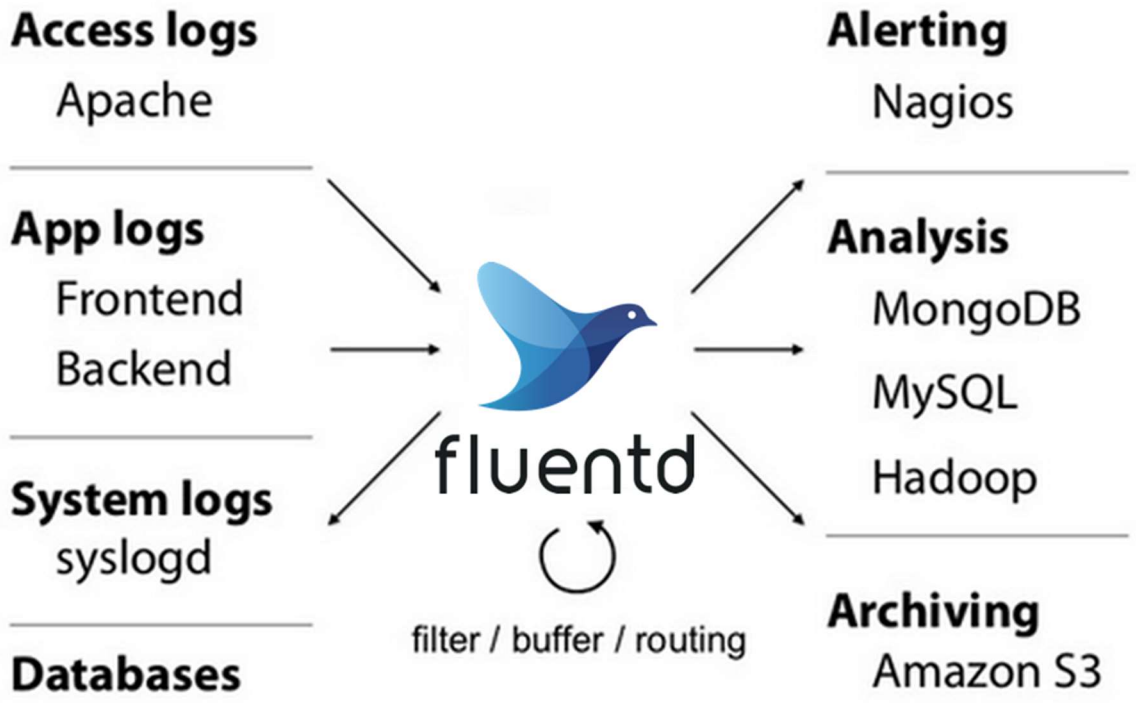

Figure 4. Fundamentals of Fluentd (Fluentd.)

4.3.2 Plugins

Plugins extend Fluentds capabilities. Plugins come in form of ruby gems. The following plugins are currently available:

- Input plugins get logs from external or internal sources, such as receiving http packages or reading from log files. Input plugins are required.
- Output plugins tell Fluentd what to do with the received and processed data. These plugins include functionality such as printing the output to the standard output, writing to files, and sending structured data to various kinds of databases. Buffering types

- Filter plugins read through input streams from given tags.
- Parser plugins are for parsing through data which comes in with a custom format. These parsers can parse through formats such as Apache logs, csv format, regular expression and more.
- Formatter plugins automatically parse the contents of the incoming data to the wanted format.
- Buffer plugins are used by the output plugins to set buffering settings for operations such as sending data to database.
- Storage plugins are for storing Fluentds internal memory either momentarily or permanently. Third-party plugins enable features such as saving to a local database.

Community plugins allow the use of many other services, which are not supported by default. These plugins are available by default in Ruby gem repositories.

4.4 Elasticsearch

Elasticsearch is a search engine, released in 2010, that is created for searching through large datasets. It is essentially a NoSQL JSON based database. Its use cases can vary from storing and searching for logfiles to application performance monitoring. Communicating with Elasticsearch is established through a REST API, which enables usage with numerous programming languages and frameworks. Elasticsearch is based on Apache Lucene, an open-source search engine software library. (Elastic b.)

Data is stored in Elasticsearch as indices. Each index contains a set of types which are generated by the documents pushed to it. In a use case of saving log files, Elasticsearch generates a new index for each day. This approach is advantageous for logs, since majority of the logs will never be looked at and having logs indexed in daily indices offers higher performance for searching. (Elastic a.)

Elasticsearch indices are built on shards and replicas. Every index is divided to the given number of shards. Every shard is a fully functional index. These shards can be divided among the nodes of a cluster, allowing for high scalability.

Replicas are copies of existing shards. They work as a form of backup. In cluster set ups, replicas are stored in different nodes than shards, this increases the resilience of the index and allows for greater search performance, due to replicas ability to serve read requests. While the number of shards is fixed during the creation of the index, replicas can be added at any time during the lifetime of the index.

4.5 Kibana

Kibana is a front-end software made for visualizing data from Elasticsearch. Kibana allows the creation of custom visualizations through an intuitive user interface with scripting capabilities. These visualizations can show various kinds of graphs, metrics and charts. The visualizations are then saved and added to a visualization library. Dashboards are pages created for building up a collection of visualizations to convey information about a specific collection of data. (Elastic c.)

Kibana includes tools for editing Elasticsearch through its user interface. The dev tools allow usage of Elasticsearches own scripting language, Painless script. The main usage of these dev tools is to send HTTP requests to Elasticsearches REST API endpoints through Kibanas text editor. (Elastic b.)

5 Implementing a Log Collection Pipeline

5.1 Containers

The centralized logging server runs on three main containers. Figure 5 displays the simple form of the logging pipeline. The host runs the containers and clients send their logs to it. Fluentd container forwards its logs to Elasticsearch container in JSON format. Elasticsearch container communicates with Kibana container through the use of its REST API.

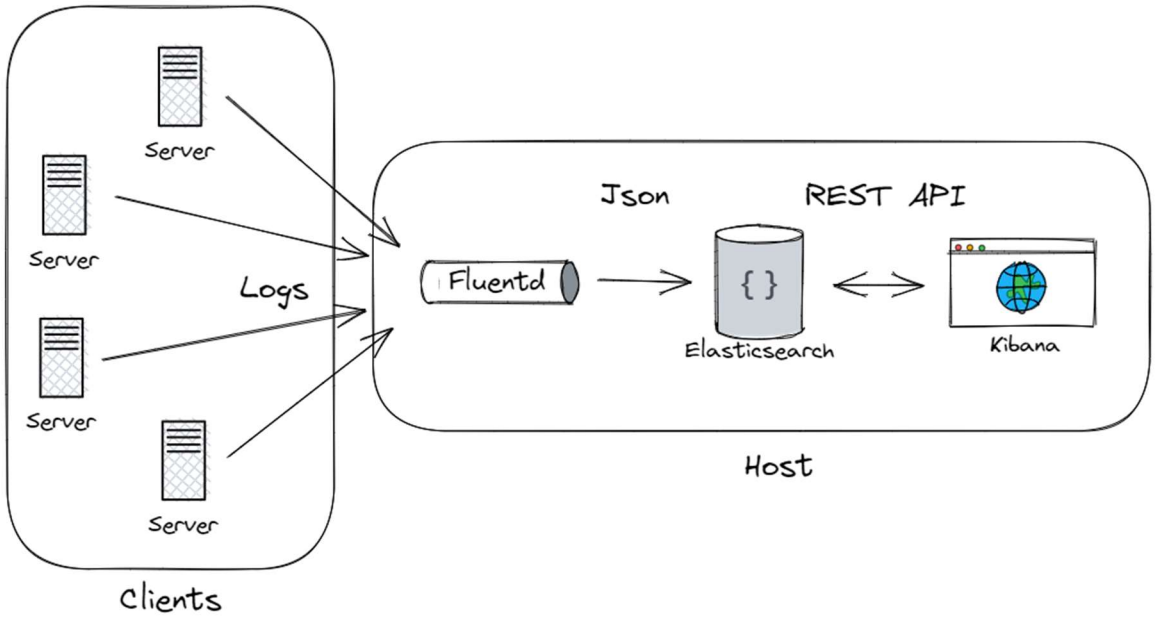

Figure 5. Illustration of the logging pipeline infrastructure

Elasticsearch and Kibana have suitable docker images already distributed on Docker hub. These images are good for use but require a few small configuration changes. Fluentd requires more specific configurations for its image, so it is created manually. The used plugins with their specific versions are listed in the Dockerfile. The custom Fluentd image uses the official release v1.15-1 as its base image.

Dockerfile includes a set of rules on how the image will be built. It is pulled from the official Fluentd Dockerfile GitHub repository. There are a couple of changes that are made to it: the Alpine Linux repository had to be given manually due to the strict nature of the network and all the third party Fluentd plugins are installed as Ruby gems. Figure 6 displays the contents of the Dockerfile.

When building the image with Podman, the commands inside the container are ran from top to bottom. It specifies the base image and sets the user as root. On the line five the specified build dependencies required for adding the Fluentd plugins are added. It then installs the

plugins in their specified versions. After adding the plugins, it removes the build dependencies and copies over the Fluentd configuration file and entry point shell script.

```
FROM fluent/fluentd:v1.15-1

USER root

RUN apk add --no-cache --repository "http://dl-cdn.alpinelinux.org/alpine/v3.15/main" \
       --update --virtual .build-deps \
       sudo build-base ruby-dev

RUN gem install elasticsearch:7.17.1 elasticsearch-api:7.17.1 \
 elasticsearch-transport:7.17.1 fluent-plugin-elasticsearch elastic-transport:8.0.1 \
 fluent-plugin-filter-logs fluent-plugin-journal-parser \
 && gem sources --clear-all \
 && apk del .build-deps \
 && rm -rf /tmp/* /var/tmp/* /usr/lib/ruby/gems/*/cache/*.gem

COPY fluent.conf /fluentd/etc/
COPY entrypoint.sh /bin/

USER fluent

```

Figure 6. Dockerfile for Fluentd

The shell script starts the Fluentd process inside the container with the given config and plugins. It is downloaded from the official Fluentd GitHub repository. Figure 7 shows the command used for building the container image. Running the build command requires Fluentds configuration file, the entrypoint shell script and the Dockerfile all to be in the same directory as where the command will be run. Network flag is used to allow Podman to use the same proxy for downloading dependencies as the user running the command.

```
podman build --network host -t custom-fluentd:latest ./
```

Figure 7. Podman build command

Figure 8 displays the entire Fluentd configuration file. Fluentd configuration file includes all the Fluentd rules. The rules are set inside directives. The source directive holds all the configurations for inbound data. The type sets the input plugin to be HTTP, since the logs will be sent as HTTP packages. Port is set as the default journald remote port. Bind with the given address sets Fluentd to listen to all client addresses. Body size limit sets the maximum size of a POST request element. Setting a large size is important since the default size is very small and the way journald Upload operates, is when the system daemon gets started, all the logs previously collected and stored by journald will be sent as a single request. After the initial collection of logs is sent logs are sent as they get created. Keepalive timeout is setting the time limit for how long the connection should be alive. Format is using

the journal format from one of the external plugins. This plugin allows Fluentd to convert the binary format of the journald files to a readable JSON format. Label is a required tag given to a specific input source. This tag groups various sources for use with specified label directives.

The filter directive is used to implement an event processing pipeline. This filter is used to alter the __REALTIME_TIMESTAMP field of the logs. This field is the timestamp used in Kibana, because the default timestamp is created when logs enter Elasticsearch instead of when the logs are created. The time format is changed since the time is in microseconds since the epoch in UTC and Elasticsearch uses a format of milliseconds since the epoch in UTC instead. Record transformer plugin enables transformation of the incoming event streams. Enable Ruby allows the use of full Ruby syntax inside the record directives expression. The record directive is for creating new key-value pairs. A new value is added in the logs with a name of realtimestamp with a value of __REALTIME_TIMESTAMP without the last three digits.

The match directive is used to determine the output for outbound data. It matches data with the Fluentd tag of upload which is given automatically in the source directive. The Elasticsearch plugin is used to create a connection between Fluentd and Elasticsearch. Because the host is running both containers, the host field is set to localhost. Logstash format configuration sets the data to be sent to Elasticsearch indices in logstash format. Logstash is Elasticsearches own log aggregation applications so mimicking the format allows Fluentd to communicate with Elasticsearch easily. This is also required in order to use Kibana. Logstash prefix sets the name of the index which the data will be sent to. The index automatically generates a date after the prefix. Include tag key includes the Fluentd tag in the JSON record. Elasticsearch 7 requires the use of type name _doc.

```
<source>
  @type http
  port 19532
  bind 0.0.0.0
  body_size_limit 256m
  keepalive_timeout 10s
  format journal
  @label @TEST
</source>

<label @TEST>

  <filter>
    @type record_transformer
    enable_ruby
    <record>
      realtimestamp ${record["__REALTIME_TIMESTAMP"][0...-3]}
    </record>
  </filter>

  <match upload>
     @type elasticsearch
     host localhost
     port 9200
     logstash_format true
     logstash_prefix datetesting
     verify_es_version_at_startup false
     include_tag_key true
     type_name _doc
  </match>

</label>

```

Figure 8. Fluentd configuration file

All the images are pushed to the local docker image registry. The registry can't be reached through the default proxy, so it requires the use of reverse SSH tunnelling through a host with access to it. In Figure 9 reverse tunnelling with SSH is used to create local port forwarding to the registry. The local port 5000 will be set to the port 80 on the endpoint, allowing traffic to the previously unreachable registry.

```
ssh user@hostname -L 5000:registry_hostname:80
```

Figure 9. Reverse tunnelling with SSH

With the connection to the registry established, the container image is pushed as seen in Figure 10. Disabling TLS verification is used since the registry is local and doesn't use SSL verification. The Elasticsearch and Kibana images are pushed to the registry with the same command.

```
podman push --tls-verify=false localhost/fluentd-custom:latest \
    docker://localhost:5000/fluentd-custom:1
```

Figure 10. Podman push command

The rest of the implementation is done on the host machine running the logging pipeline. The host has access to the docker image registry so reverse tunnelling is no longer required. All the container images are pulled on to the host machine using the Podman pull command as shown in Figure 11.

```
podman pull --tls-verify=false docker://registry_hostname/fluentd-custom:1
```

Figure 11. Podman pull command

5.2 Connecting the Containers Together

To enable communication between the containers, net flag is given on the containers startup command as seen in Figure 12. This flag allows the containers to use all the hosts network interfaces. This allows greater network access for the containers. With the host network, the containers can communicate freely with each other. This also disables any port mappings given with the startup command. The containers are also given the k8s-file as the log driver because the default journald implementation of Podman was not working. The d flag starts the containers in detached mode, in which they run in the background. The e flag is for setting environment variables for the containers. Elasticsearch run on Java and is given 1 gigabyte of memory through the environment variable. Both Elasticsearch and Kibana containers are started with the single node discovery type. This means that the Elasticsearch cluster is set to only have a single node.

```
podman run -d --net=host --log-driver=k8s-file --name fluentd \
    registry_hostname/custom-fluentd:1

podman run -d --net=host -e ES_JAVA_OPTS="-Xms1g -Xmx1g" \
    -e discovery.type="single-node" -e node.store.allow_mmap="false" \
    --log-driver=k8s-file --name elasticsearch registry_hostname/elasticsearch:1

podman run -d --net=host -e discovery.type="single-node" --log-driver=k8s-file \
    --name kibana registry_hostname/kibana:1
```

Figure 12. Podman run commands

After Kibana container is running it requires the hostname and port of Elasticsearch to be set in its configuration file. Due to the containers being highly optimised, they don't ship with any traditional text editors, so stream editor is used instead. Figure 13 shows the command for starting a Bash session inside the Kibana container.

```
podman exec -it kibana bash
```

Figure 13. Podman exec command

The command in Figure 14 displays the usage of the stream editor. This command is executed inside of the config directory of the Kibana container. It is set to edit a file and create a backup of the original file. It looks for the first occurrence of the default Elasticsearch hostname and port and replaces them with the correct address, which in this case leads to localhost.

```
sed -i.bak 's/elasticsearch:9200/127.0.0.1:9200/' kibana.yml
```

Figure 14. Stream editor command

Kibana container requires a restart after editing the configuration file. Figure 15 shows the Podman command for restarting containers.

```
podman restart kibana
```

Figure 15. Podman restart command

After restarting, Kibana starts up and connects to Elasticsearch. Kibana is accessible through a browser using the hostname and Kibanas default port of 5601. As seen in Figure 16, reverse tunnelling has to be used because of the proxy so localhost port 8080 was set to the hosts 5601.

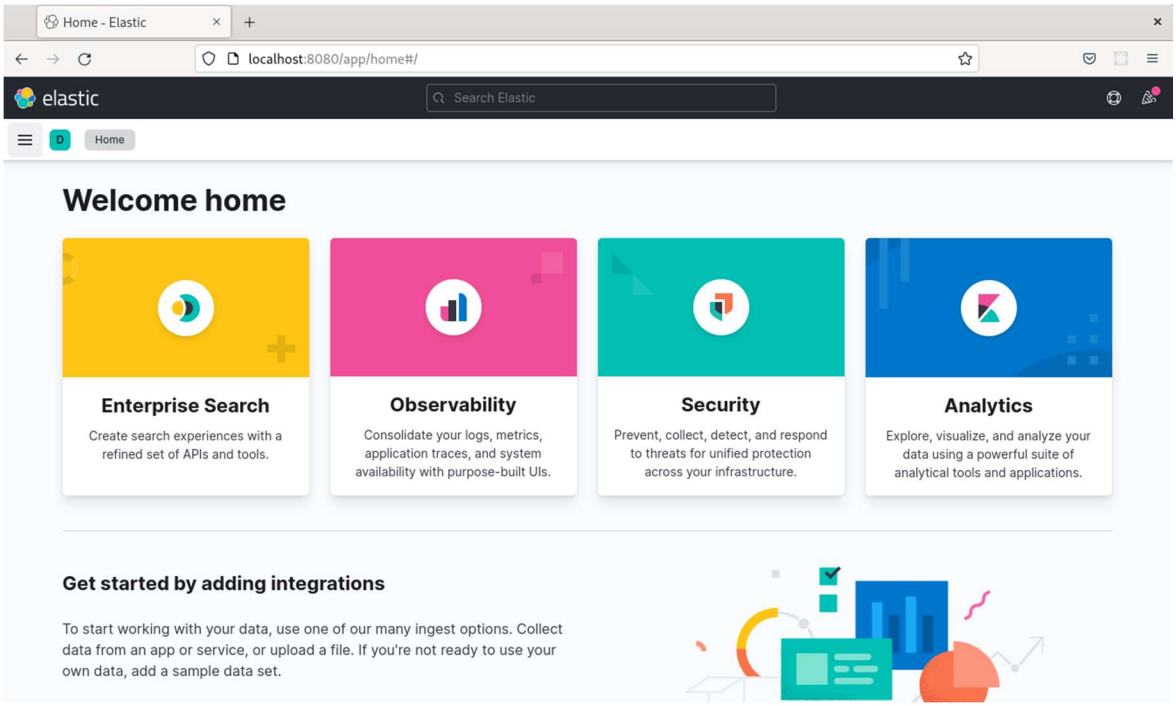

Figure 16. Kibana frontpage

Elasticsearch indices require an index template in order to map the custom timestamp created in Fluentd as a date type. The template is created using the Kibana Dev Tools. Dev Tools are used to send simple HTTP requests to Elasticsearches REST API. The template is used on indices which match the index pattern. Index pattern is set in Fluentd configuration file under the name logstash prefix. The number of shards and replicas is set to one because of the Elasticsearch only running on a single node. Figure 17 shows the creation of the index template in Kibana Dev Tools.

```
PUT _index_template/test_logs_template
{
  "index_patterns": ["testlogs*"],
  "priority": 1000,
  "template": {
    "mappings": {

      "properties": {
        "realtimestamp": {
          "type": "date"
        }
      }
    },
    "settings":{
      "number_of_shards":1,
      "number_of_replicas":1
    },
    "aliases":{
    }
  }
}

```

Figure 17. Index template creation

Fluentd container is started after Kibana and Elasticsearch are running and communicating with each other. Once Fluentd is up and running, it can begin to receive logs from clients. Systemd journal remote is installed to the clients, in this case the test servers, using a local Zypper repository. Figure 18 shows the command used for installing the package. The package contains both journald remote and journald upload services.

```
zypper install systemd-journal-remote
```

Figure 18. Zypper install command

The journald upload configuration file is located in /etc/systemd/journald-upload.conf. The URL field is the only line edited in the file as seen in Figure 19. It is set to the logging servers hostname and the port which is given in Fluentd configuration file. It is important to set HTTP manually because it automatically defaults to HTTPS.

```
15  [Upload]
16  URL=http://log_server_hostname:19532
17  # ServerKeyFile=/etc/ssl/private/journal-upload.pem
18  # ServerCertificateFile=/etc/ssl/certs/journal-upload.pem
19  # TrustedCertificateFile=/etc/ssl/ca/trusted.pem
20
```

Figure 19. Journal-upload.conf file

The system daemon is started, after the configuration file is updated. Starting the daemon requires root access. Figure 20 shows the systemctl command for starting the daemon.

```
systemctl start systemd-journal-upload.service
```

Figure 20. Starting daemons from systemctl

The log indices are located in Kibana in the Index Management under Stack Management. An index pattern is created to be able to retrieve data from all the indices when using the dashboard. The index pattern is created in Index Patterns sub-menu. Figure 21 shows the index pattern creation process. The name of the index pattern must match the names of the indices generated by Fluentd. The timestamp field sets the default time for the indices.

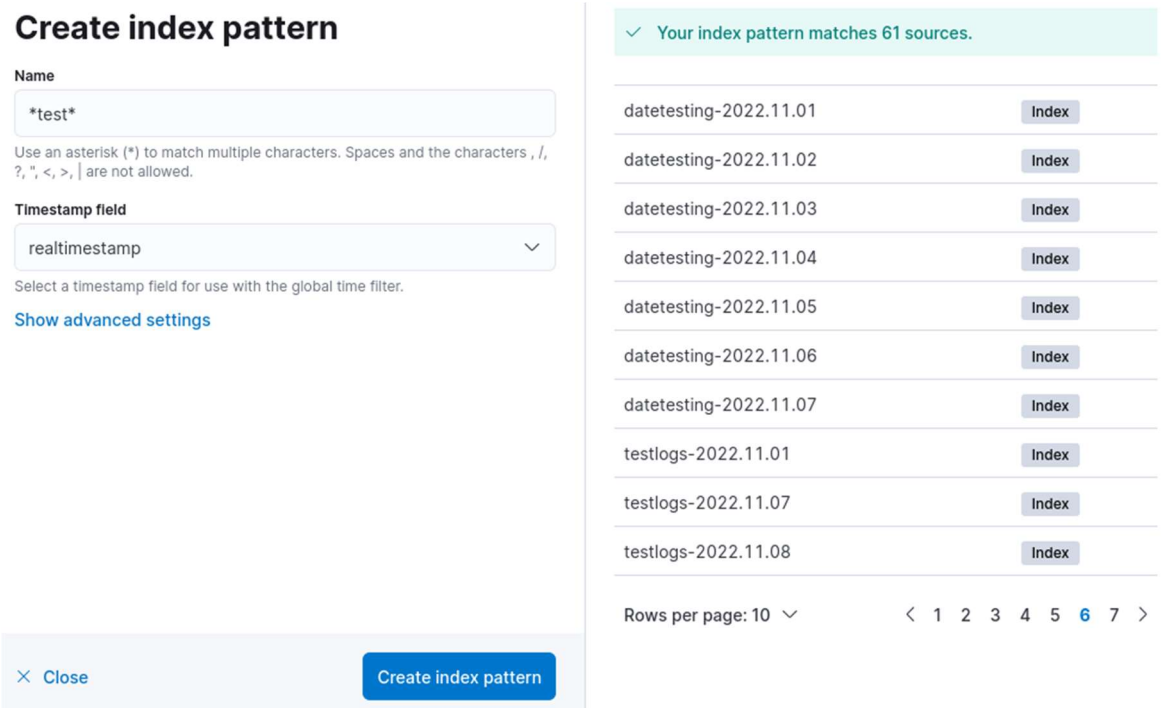

Figure 21. Index pattern creation

5.3 Visualization

Visualization is done on Kibana using its analytics dashboard. The views inside the dashboard are created through the visualize library. The library holds all the created visualizations. Visualizations can be graphs or other general views of data coming from indices, plain text and pictures or controls for changing elements on the dashboard.

Figure 22 shows the creation of the metric-based counter for displaying the number of tests ran in the given time. Usage of the Formula function allows for use of custom scripts. The function searches for unique log messages with the given search parameters given in Kibana Query Language (KQL) format.

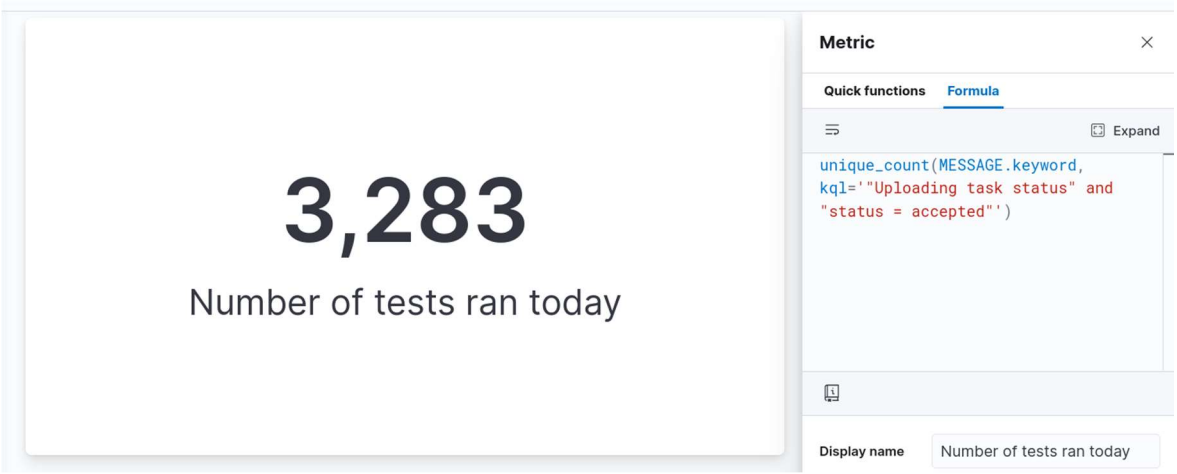

Figure 22. Metrics of tests

Figure 23 shows the control visualization. It allows users to select a test server from a dropdown list which gets automatically populated when new hostnames are inserted to the database inside logs. The Kibana dashboard displays the logs from the selected test servers or from all of them when none are selected.

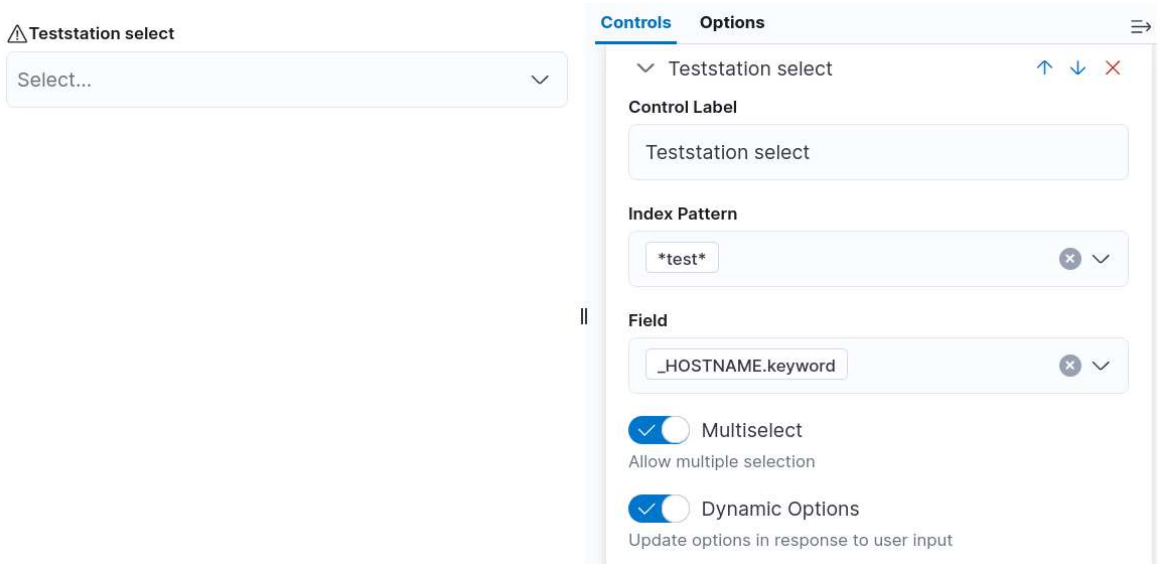

Figure 23. Control block

Figure 24 shows a graph of all the logs inside the system and their system user units. The graph can be used to spot anomalies within the logs. If there are a low number of logs from a time when there should be numerous tests running or a large number of logs when there's nothing happening, the graph makes it clear.

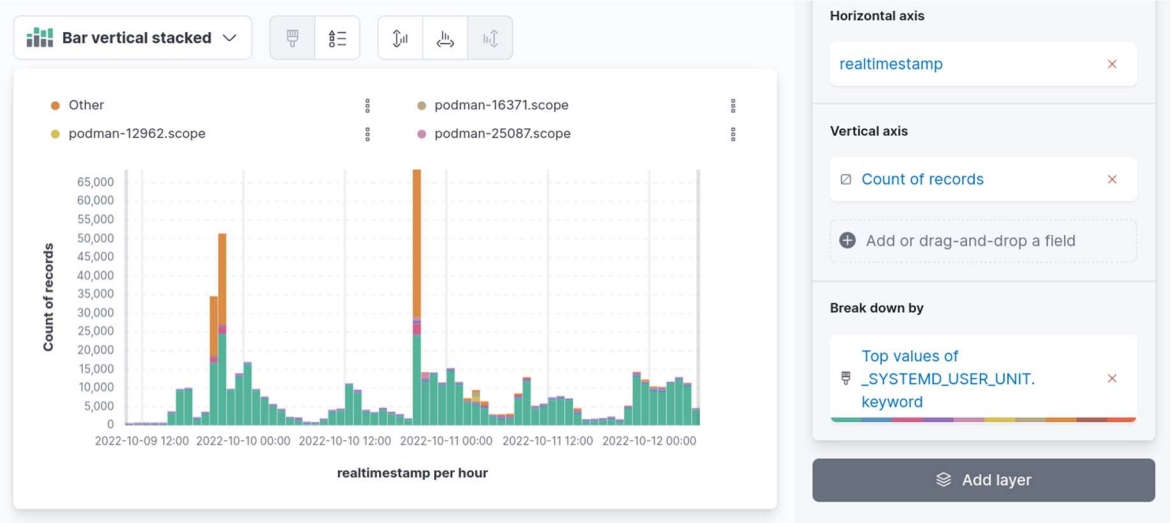

Figure 24. Bar graph

Kibana has a discover functionality for doing searches inside the indices of an index pattern. These search results are customisable and can be saved in order to use them in the dashboard. Figure 25 shows the search of latest warnings. It uses KQL to select the warning logs and the fields are selected and arranged manually.

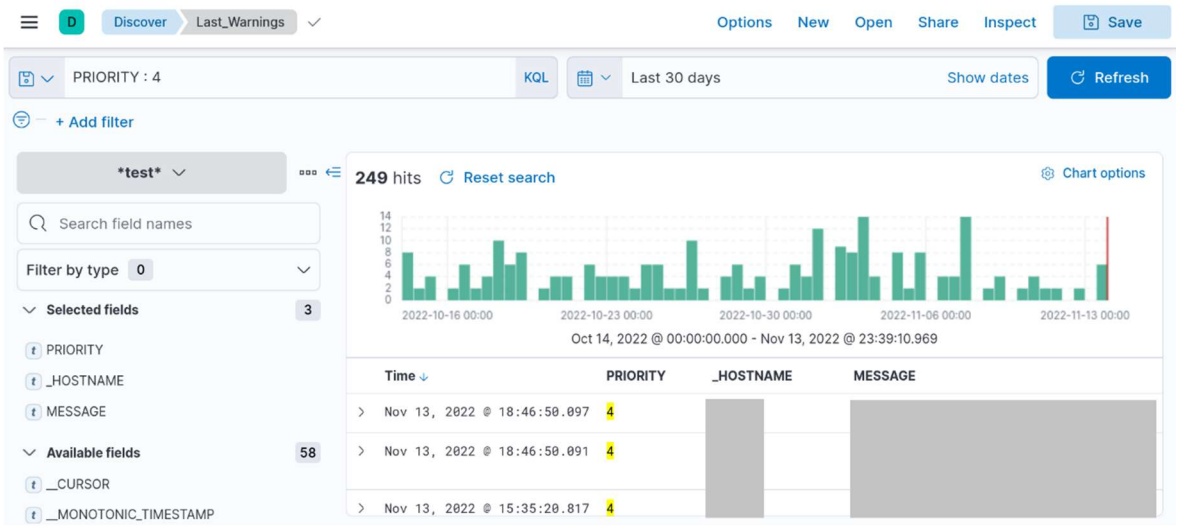

Figure 25. Customised search

All of the created visualizations are used to create the final dashboard. Figure 26 shows the end result of the dashboard. Users can select specific test servers to display their log information from. Clicking on a log inside one of the search windows opens all the data inside of it. Users can also edit the current time frame from the dashboard settings or by selecting a segment of time from the bar graph.

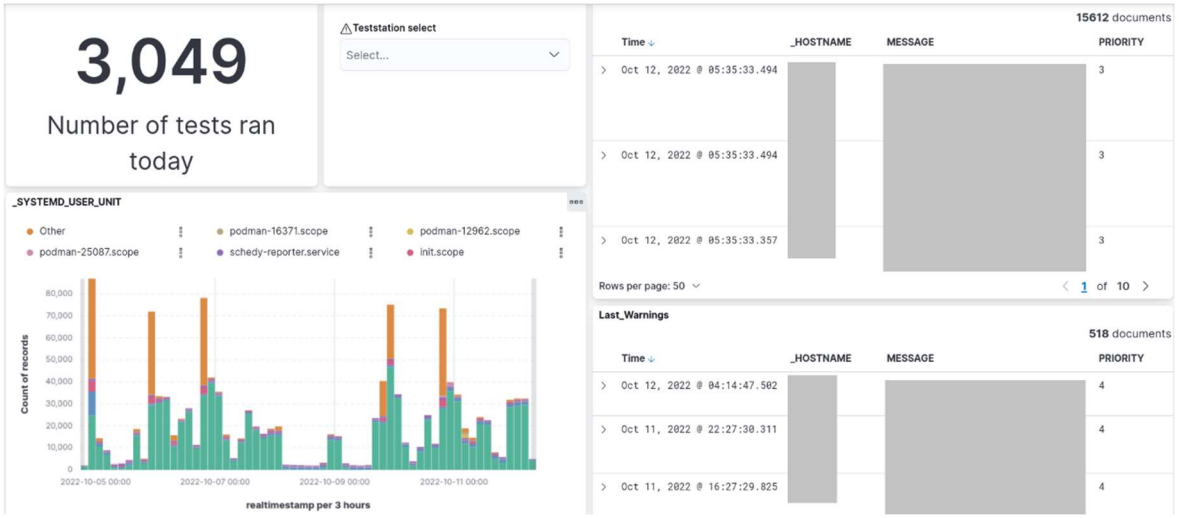

Figure 26. Final dashboard

6 Results and Conclusion

The objective of the thesis was to create a unified logging layer, a pipeline, for a continuous integration environment with a focus on the testing side. The log data was meant to be visualized through to use of a front-end user interface which would show important information, such as latest error messages and overall log activity through a dashboard. The pipeline was to be easily deployable and distributable by the means of containerization. The transportation of the logs was also planned to be done using journald upload.

The result of the thesis is a complete unified logging layer created using the EFK stack. This stack consists of Elasticsearch as the database for the logs, Fluentd as the log aggregator and Kibana as the user interface. The logs are sent from servers running tests through Schedy task schedulers systemd journal through systemd journal upload. These logs get sent as HTTP packages to Fluentd, which uses a collection of third-party plugins to mould the logs to JSON format suitable for Elasticsearch and Kibana and sends them to Elasticsearch indices where the data gets indexed to daily indices.

The pipeline could be improved by automating the implementation of systemd journal upload to the servers. Kibana also has some weaknesses, which come through in their decision to split functionality such as performance monitoring and live log stream to a different view, instead of allowing them to be added to the same dashboard. In the future the project can be improved by also implementing the log collection to the build side of the continuous integration environment and creating a separate dashboard and index pattern for visualization side.

References

Assaraf, A. 2022. How Logs Can Empower Your CI/CD Delivery Pipeline. Retrieved on 7 November 2022. Available at https://devops.com/how-logs-can-empower-your-ci-cd-delivery-pipeline/

Butler, T. 2015 What Is a Dockerfile? Retrieved on 2 November 2022. Available at https://www.cloudbees.com/blog/what-is-a-dockerfile

Docker. What is a container? Retrieved on 14 October 2022. Available at https://www.docker.com/resources/what-container/

Docker.com. Retrieved on 1 November 2022. Available at https://commons.wikimedia.org/wiki/File:Docker-containerized-and-vm-transparent-bg.png

Elastic. a. What is an Elasticsearch Index? Retrieved on 20 October 2022. Available at https://www.elastic.co/blog/what-is-an-elasticsearch-index

Elastic. b. What is Elasticsearch? Retrieved on 16 October 2022. Available at https://www.elastic.co/what-is/elasticsearch

Elastic. c. What is Kibana? Retrieved on 16 October 2022. Available at https://www.elastic.co/what-is/kibana

Fluentd. What is Fluentd? Retrieved on 14 November 2022. Available at https://www.fluentd.org/architecture

Gheorghe R. 2020. Tutorial: Loggin with journald. Retrieved on 15 November 2022. Available at https://sematext.com/blog/journald-logging-tutorial/

GitLab. What is CI/CD? Retrieved on 15 October 2022. Available at https://about.gitlab.com/topics/ci-cd/

Loggly. Using journalctl. Retrieved on 14 October 2022. Available at https://www.loggly.com/ultimate-guide/using-journalctl/

Podman. What is Podman? Retrieved on 16 October 2022. Available at https://docs.podman.io/en/latest/

Red Hat. a. What is CI/CD? Retrieved on 15 October 2022. Available at https://www.redhat.com/en/topics/devops/what-is-ci-cd

Red Hat. b. What is Podman? Retrieved on 10 November 2022. Available at https://www.redhat.com/en/topics/containers/what-is-podman

Schäfer, J. 2016. Why Journald? Retrieved on 15 November 2022. Available at https://www.loggly.com/blog/why-journald/

Systemd. Journal File Format. Retrieved on 10 November 2022. Available at https://systemd.io/JOURNAL_FILE_FORMAT/

Tamura, K. 2014. Unified Logging Layer: Turning Data into Action. Retrieved on 2 November 2022. Available at https://www.fluentd.org/blog/unified-logging-layer