GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

Scalable logging and log-file analysis

Linus Weber

MatrNr: 21418903

Supervisor: Christoph Hottenroth

Georg-August-Universität Göttingen
Institute of Computer Science

March 28, 2023

Abstract

Servers and applications generate large amounts of logging data that is useful for debugging and also for monitoring services.

However, this data resides on the machines themselves which makes it difficult to analyze or recover in case of an incident.

A number of architectures and software stacks exist that tackle these problems, ranging from custom architectures developed in-house to service components used in a variety of contexts.

Standardized approach was chosen that allows the organization to adapt and expand the logging and log-file analysis processes to their needs.

An 'Elastic stack' was set up on a virtual machine from a GWDG-hosted cluster, and a couple of machines that run a number of services were set up and connected to the logging service. First, different kinds of log-files were ingested into the logging service in order to explore its capabilities. Then, it was investigated if it is possible to control the detail with which logging-data was collected, analysed and visualized.

Because there is a wide range of 'Beats' file-shippers available and users can also write their own shippers to suit their requirements, it is possible to collect and analyze all kinds of log-files. Controlling the detail-level of log-collection and -analysis, however, proved to be difficult, especially because there is no way to issue these commands from within the web-interface.

Contents

List of Figures

List of Abbreviations

HPC High-Performance Computing

GWDG Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen

SaaS Software-as-a-Service

HTTP Hypertext Transport Protocol

HTTPS Hypertext Transport Protocol Secure

TLS Transport Layer Security

XML Extensible Markup Language

1 Introduction

Servers and applications generate large amounts of logging data that is useful for debugging and also for monitoring services.

However, this data resides on the machines themselves which makes it difficult to analyze or recover in case of an incident.

A number of architectures and software stacks exist that tackle these problems, ranging from custom architectures developed in-house to service components used in a variety of contexts. Most of the custom-tailored solutions, however, are employed to serve an infrastructure that is unique to the company that built it and usually comprises of services that are all part of one product which is not the kind of environment that is comparable with the GWDG.

Instead, a more standardized approach was chosen that allows the organization to adapt and expand the logging and log-file analysis processes to their needs. The Elastic stack includes log-file shippers called 'Beats', a file-aggregator called 'Logstash', a database called 'Elasticsearch' and a visualization platform called 'Kibana'. This technology stack was chosen as its components are also used in a range of other stacks and custom-tailored solutions and promise to be well adaptable to the needs of the organization.

The Elastic stack was set up on a virtual machine from a GWDG-hosted cluster, and a couple of machines that run a number of services were set up and connected to the logging service. First, different kinds of log-files were ingested into the logging service in order to explore its capabilities. Then, it was investigated if it is possible to control the detail with which logging-data was collected, analysed and visualized.

Because there is a wide range of 'Beats' file-shippers available and users can also write their own shippers to suit their requirements, it is possible to collect and analyze all kinds of log-files with small amounts of effort. The 'Kibana' web-interface offers many options to filter, sort, analyze and aggregate data. Controlling the detail-level of log-collection and -analysis, however, proved to be difficult, especially because there is no way to issue these commands from within the web-interface.

1.1 Contributions

Documentation for how to set up a functioning Elastic stack in a GWDG-hosted cluster that can be used for further studies and to implement features that are needed to fulfill the organization's requirements is provided in section 3. Example configuration files for Docker and an nginx proxy can be found in appendix A.

1.2 Outline

Section 2 gives a summary of the initial search for candidate logging and log-file analysis solutions. A brief description of how the service was set up is presented in section 3. Then, section 4 describes how the chosen solution was tested and evaluated. Section 5 contains the results of that evaluation and they are discussed in section 6. Finally, a conclusion summarizes the findings of this report in section 7.

2 Research

2.1 Log files

A wide range of approaches to logging and log-file analysis are employed in different organizations or even within organizations.

Most commonly, services running on Linux servers print log messages to files in the

`/var/log`

directory. This directory includes logs from system services such as the package manager, low level system services, but also services that are installed natively on the machine.

Some log files, however, reside in other directories, depending on the installation method. For example, dockerized applications might store their log files in the docker volume that was assigned to them which typically resides in

`/var/lib/docker/volumes`.

2.2 Manual logging

Most commonly, the log files are analyzed using tools and filters such as grep in order to collect relevant information. This can be done with minimal effort, but provides little options for log aggregation and analysis. Moreover, all logs remain on the machines themselves which means that they may be lost in case of an incident.

2.3 Semi-automated logging

One way to automate the log-file analysis process is to use scripts that employ the same tools used in the manual process. It is feasible to send alerts to system administrators if there are critical error messages in the logs or certain metrics such as the number of active connections to a web-service exceed a predefined threshold. Log-files can also be transferred to a central server for safekeeping, but techniques such as log-file-rotation make it very time consuming to code a custom-tailored solution for the organization.

2.4 Why scalable logging

The extend of logging and log-file analysis is constrained firstly by the capability of the infrastructure and secondly by the extend of information the engineers that use the logging system need and are able to process. There are a few examples of when an organization might decide to change how much information from the logs is collected and analyzed.

- Different environments: depending on whether the monitored server is a development, test, staging, or production machine, different levels of logging might be appropriate. Development servers should make logs available that include low level messages that can be helpful to developers but would be unnecessary noise for the operations team.

- New instance: when a new system is set up, engineers might require a a higher level of logging in order to spot deficiencies early on. Introducing a new component to an environment could affect other services as well if, for example, integration with another microservice fails.

- System update: similar to adding a new instance, updating systems introduces change to an infrastructure. Often, these updates are rolled out rapidly with a large number of systems migrating to the new version in a short amount of time.

- Traffic peaks: there might be a regular increase in traffic during a certain time of the day, or there might be unexpected peaks due to external events. In some cases, these circumstances could push the logging infrastructure to its limits as the might not be enough network-bandwidth or processing power to process the increased amount of logs generated.

- Unexpected failure: it is expected that an infrastructure will generate lots of error messages in its logs when a component fails. For the engineers it would be beneficial to have a lot of messages to gather information from as long as they have the ability to filter out unnecessary ones.

2.5 Scalable parameters

A range of methods exist in order to increase or decrease the resolution with which logging of an infrastructure is done.

- File selection: each machine generates a range of log files for services like the package manager, font collection or applications that are run on the server. Some of those files might or might not be relevant to the engineers. Moreover, services like the nginx web-server generate an `access.log` file which contains information about incoming requests and an `error.log` file which contains information on internal problems.

- Log level: there is no one standardized format for log-files, but there is - to a degree - a shared terminology for how to label levels of log messages: `debug`, `info`, `warning`, `error`. While `debug` messages are relevant to developers in their own development environments, `warning` and `error` messages are important for operations engineers as well.

- Filters: log messages can be excluded based on filters like regular expressions or other operations. That is useful in situations where specific types of errors should be ignored or, conversely, singled out because they are being worked on.

- Aggregation: Instead of storing details for every single log-message, individual values can be aggregated into a single value. For example, counting the number of accesses from a continent instead of accesses from an IP address, or counting the number of accesses from a certain device. While aggregation can provide valuable information, if it is done to reduce the storage space required it will inevitably reduce the resolution of information that can be gathered from the aggregated data.

- Metrics polling interval: in cases where network bandwidth or computing power of the logging-service is limited, a way to reduce usage of those resources is to increase

the time between collecting new data from the log-files. However, the more time there is between rounds the less up to date the information in the logging-service is.

- Retention period: Data can be deleted from the machine hosting the logging-service after an amount of time that fits the requirements of the organization. A short lifespan of the data could undermine the engineers' ability to track medium- to long term trends.

2.6 Types of Services

Organizations operating a server infrastructure have a number of options available to them when it comes to logging services.

Software-as-a-Service models allow the organization to outsource part of their infrastructure to another service provider. A SaaS solution could comprise all parts of the logging service, or just single components could be hosted by the service provider so that the service can be integrated into the organization's logging pipeline. Examples for such service providers include Google Cloud Logging[1], Sumo Logic[2] and Loggly[3].

One alternative to the Software-as-a-Service model would be self-hosted applications - some of which could also be available in a SaaS model but allow hosting private instances under an open license. Some example applications are explored in section 2.8.

2.6.1 Custom solutions

The third alternative are custom-tailored solutions that employ existing services such as Redis[4], Apache Kafka[5], Elasticsearch[6], InfluxDB[7] or applications that are developed in-house.

One such example can be found in a case-study[8] from the company Carousell who developed their own log-file shipper for the worker servers and a log-service using Redis that collects the log-messages before passing them on to an Apache Kafka instance and then ingesting them into the Google Cloud Logging platform which is provided using the SaaS model.

2.7 Typical Architecture

A logging architecture typically consists of four major components:

1. Shipper: an agent running on the worker server finding files that should be passed on to the next stage, performing some initial filtering operations. Often, shippers parse the log-files to turn them into a format that can be processed by the later stages such as json or XML.

[1] https://cloud.google.com/logging
[2] https://www.sumologic.com/
[3] https://www.loggly.com/
[4] https://redis.io/
[5] https://kafka.apache.org/
[6] https://www.elastic.co/de/elasticsearch/
[7] https://www.influxdata.com/
[8] https://medium.com/carousell-insider/scalable-logging-for-microservices-1043d3d17c18

2. Collector: a software component that serves as the target for the submissions from the shipper agents. These components are able to integrate data from a variety of sources including but not limited to the shipper agents. Data from log-files can be aggregated and filtered in this step.

3. Storage: specialized databases that hold data for further analysis. There are some storage systems that are especially good at searching the data in the system, while others specialize in ingesting a large amount of input streams simultaneously. At this point, the file retention period may be specified to achieve scalability of the logging system.

4. Visualization & Analysis: a web-interface that allows engineers to chart log-data as graphs, display, sort and search log-files from individual servers or even groups of servers at once and apply advanced analyses such as machine-learning techniques to the logging data. This is usually the main interface between human users and the logging infrastructure.

Figure 1 shows the typical generalized architecture of a logging service.

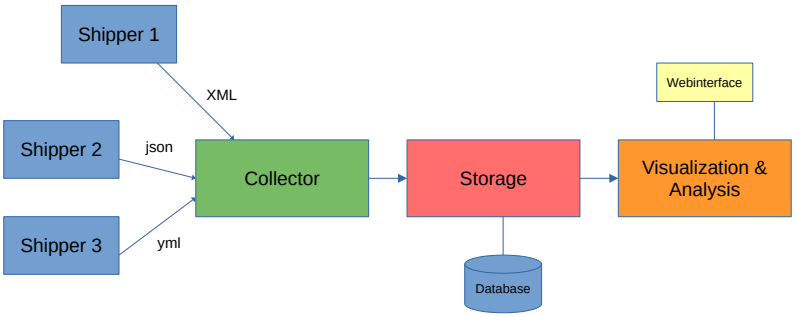

Figure 1: Typical architecture of a logging service.

2.7.1 Buffering

A situation may arise in which the network or individual machines are overloaded with work. Therefore, it is commonly seen that buffer services are employed in order to prevent data loss as messages that cannot be processed are lost. The first position where a buffer could be used is between the individual shippers and the collector, and the second position is between the collector and the storage system. Depending on the complexity of the infrastructure, even multiple collectors and multiple buffers might be used.

An example of a generic logging architecture employing buffers is shown in figure 2.

2.8 Available self-hosted services

A number of software components that can be used in logging and log-file analysis services have already been listed in the sections above.

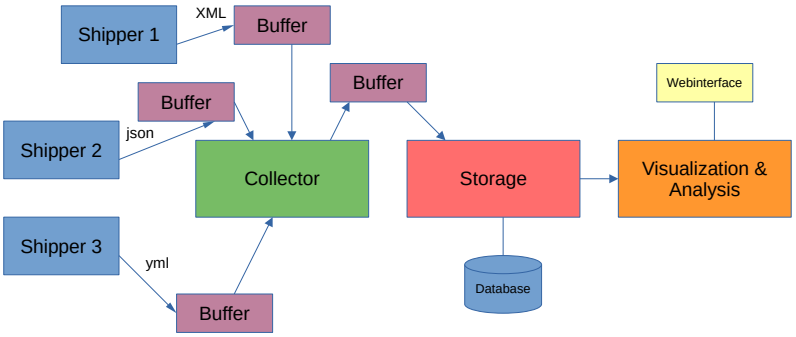

Figure 2: Architecture of a logging service including buffers.

Nagios Log Server[9] is a service that requires a paid license starting at 4,000 $ per year for a single instance to operate. It provides an extensive set of capabilities and integrates with other solutions from the same provider.

Another self-hosted solution that provides a variety of capabilities as described in section 2.7 and integrates with other services such as InfluxDB, Elasticsearch and others is Icinga[10].

2.8.1 Elastic stack

The Elastic stack[11], formerly known as the ELK stack, has one software service for each of the components described in section 2.7.

1. Beats: shippers are called 'Beats' in the Elastic stack. There a many different kinds of Beats used for different purposes: Filebeat sends data from log-files, Metricbeats sends server metrics such as system load or memory usage, Packetbeat sends data about individual network packages and Winlogbeat can be used to send log data from Windows systems.

2. Logstash: 'Logstash' is a data collector that can ingest data from many different sources such as Beats, Kafka, redis or `.csv` files. Logstash is also able to perform certain operations such as filtering or aggregation on the data it collects.

3. Elasticsearch: this is a no-SQL database designed specifically to provide powerful search features such as full-text search, fuzzy searches and offers results based on scores that gauge how likely an item is to fit to a given query. Elasticsearch is used not only in logging services but also in many other fields that make us of its search features.

4. Kibana: this is the interface that allows users to discover the data they have collected and build dashboards based on logs and also aggregated and analyzed data from Logstash. Some functionalities for the other components are integrated into the Kibana web-interface as they don't offer graphical user interfaces themselves.

[9] https://www.nagios.com/products/nagios-log-server/
[10] https://icinga.com/products/metrics-and-logs/
[11] https://www.elastic.co/de/elastic-stack/

Single components of the Elastic stack are used not only by various custom-built solutions, but also serve as the foundation for services such as Graylog[12]. It is built on Elasticsearch and can integrate other Elastic stack components such as Beats and Logstash. However, it comes with its own user-interface that offers capabilities for processes such as reporting.

3 Setup

The Elastic stack was set up on a virtual machine using Docker[13]. An nginx[14] reverse proxy that is in the same docker network as the Elastic stack components forwards web traffic to the Kibana application and redirects HTTP requests to use TLS. A certbot[15] client checks daily (cronjob) if the TLS certificates need to be renewed. The certificates are mounted into the nginx docker container in a volume.

Figure 3 shows a simplified version of how the logging system and test systems that feed log-files were set up.

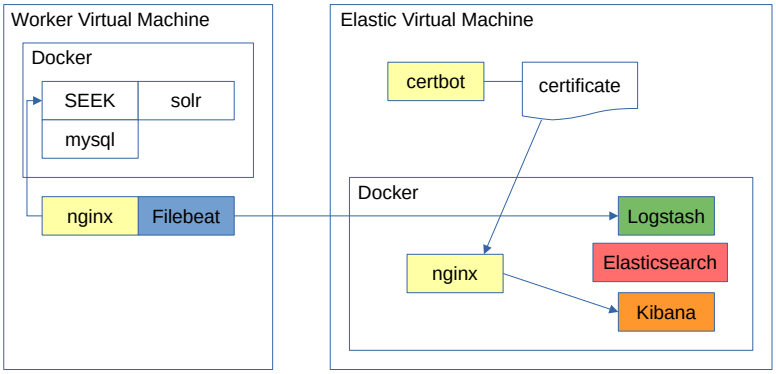

Figure 3: Simplified overview of the setup used in the practical part of the seminar.

The cronjob used to check the validity of the TLS certificate:

```
@daily certbot renew -pre-hook "docker compose -f \\
    /opt/docker/docker-elk/docker-compose.yml down" \\
    --post-hook "docker compose -f \\
    /opt/docker/docker-elk/docker-compose.yml up -d"
```

The dockerized application must be shut down when renewing the certificates.

The following snippet shows how the nginx proxy was set up. The config file and the certificates are mapped into the nginx container. Ports 80 for HTTP and 443 for HTTPS are mapped to the host system ports, and the container is in the 'elk' network - the same as the other applications in the stack.

[12]https://www.graylog.org/
[13]https://www.docker.com/
[14]https://nginx.org/en/
[15]https://certbot.eff.org/

```
nginx:
    image: nginx:latest
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - /etc/letsencrypt/:/etc/letsencrypt/
    ports:
      - 80:80
      - 443:443
    networks:
      - elk
    restart: unless-stopped
```

The nginx proxy is configured to redirect incoming HTTP traffic to HTTPS and encrypt all traffic using the TLS certificate acquired using certbot.

```
server {
    server_name c111-042.cloud.gwdg.de;
    listen 80;

    return 301 https://$host$request_uri;
}
```

For testing purposes, an nginx webserver and a dockerized instance of FAIRDOM-SEEK[16] was set up on a separate virtual machine. FAIRDOM-SEEK is a Ruby on Rails[17] appication that uses MySQL[18] as a database (although substitutes may also be used) and Apache Solr[19] for indexing and searching.

Different Filebeat modules (manual, nginx, mysql) were installed to collect data both from the webserver and FAIRDOM-SEEK.

The application in this case was not secured by HTTPS because they are only temporary instances used for testing and no sensitive data is transmitted.

At times, components of the applications or even virtual machines were deleted and set up again in order to reset them to a clean and working state.

Dashboards and other overviews were added automatically when setting up the filebeat assets. They depend on the modules that were enabled.

4 Evaluation

For evaluating the Elastic stack a few approaches were chosen:

- Webserver: basic data from an nginx webserver's `access.log` and `error.log` were ingested into the logging system. This was done to explore the systems, their capabilities and get a general overview of its inner workings.

[16] https://seek4science.org/
[17] https://rubyonrails.org/
[18] https://www.mysql.com/de/
[19] https://solr.apache.org/

- Provoked errors: for example, config files that contain errors were loaded by the webserver in order to provoke error messages in the logs. This was done specifically to see whether information displayed in Kibana would be useful for figuring out mistakes and fixing them.
- Regular use: The FAIRDOM-SEEK application was used regularly (uploading files, creating projects and institutions, etc.) in order to simulate normal usage and see how many and which kinds of logs would be collected.
- API calls: Using FAIRDOM-SEEK's API, a lot more log messages could be generated, and some specific errors could be provoked (known bug when the logging service cannot generate an error message if a request has an unsupported HTTP method specified in its header).

Configuration options and capabilities of the stack in regards to scalable logging were explored with the parameters specified in section 2.5 in mind. Possible ways to automatically change those parameters and interfaces to integrate them into the Elastic stack were considered.

5 Results

Figure 4 illustrates how investigation access and error log-files from an nginx webserver looks like.

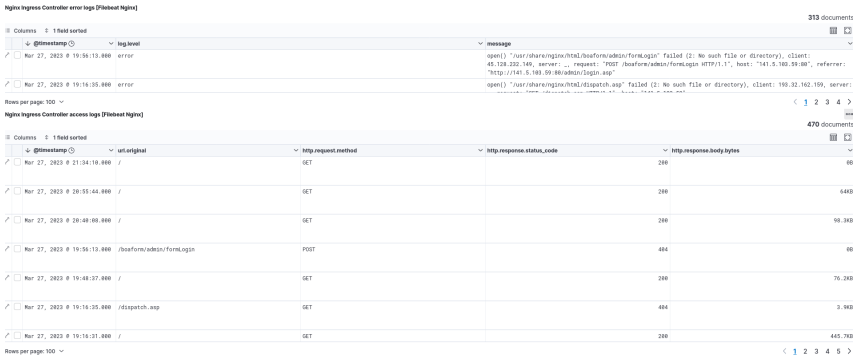

Figure 4: Example screenshot from the Kibana web-interface when inspecting access and error log-files from an nginx webserver.

It is easy to collect data from new sources because a variety of Filebeat modules are available that cover most use-cases. There's also the possibility to define custom Filebeats in order to integrate self-developed applications into the logging infrastructure. Pre-defined processes can be reused as part of the data extraction pipeline.

The Kibana web-interface offers easy to use tools to explore the data sources and inspect errors and log-files in detail. Log files can be sorted and filtered so that only relevant data ist displayed. Through the charts, users can interactively select the timeframe from which data should be displayed, allowing better focused investigations.

Both Beats and Logstash offer many configuration options that allow engineers to modify each server individually according to their needs. Together with the options

available on the Kibana web-interface and the Elasticsearch database, all parameters defined in section 2.5 can be set to the desired levels.

There is, however, no convenient way to change these settings from within the user interface of the Elastic stack. Users may only select which data is displayed in the Kibana web-interface. This does help engineers cut down the mass of messages that are delivered to them. But it does not help with avoiding network or computing bottlenecks in the entire infrastructure.

External programs would be necessary to modify the configuration files for the Beats, for Logstash and for Elasticsearch, and subsequently restart the services. Additionally, a convenient user interface is needed that would allow engineers to control how much information individual servers provide and how it is aggregated, stored and analyzed.

6 Discussion

The Elastic stack provides many options and features for log-file analysis. However, it does not offer convenient ways to modify the level of logging of the worker servers and the way in which the collected data is aggregated, stored and analyzed. That might not be the case with alternative solutions presented in sections 2.6 and 2.8. They could be explored in further projects.

The capabilities of Logstash were explored only on the surface and might offer more potential for scalable logging.

While there is currently no way to control individual or groups of Beats from within the Kibana web-interface, one option for further work could be to add those features as an extension to the interface.

Alternatively, a custom solution to scalable logging could be developed that would fully address the challenge. However, this would probably be to the detriment many log-file analysis features that the Elastic stack or similar solutions offer.

7 Conclusion

This project report gives an overview of the capabilities and limitations of the Elastic stack for scalable logging and log-file analysis. Different available options for logging infrastructure are listed and general concepts and patterns are presented. It is explained how the Elastic stack and an application server for testing purposes can be set up, and ideas for further work aiming to overcome the limitations of existing logging solutions for scalable logging are provided.

A Code samples

Elastic stack docker-compose.yml (modified from https://github.com/deviantony/docker-elk):

```
version: '3.7'

services:

  # The 'setup' service runs a one-off script which initializes users inside
  # Elasticsearch - such as 'logstash_internal' and 'kibana_system' - with the
  # values of the passwords defined in the '.env' file.
  #
  # This task is only performed during the *initial* startup of the stack. On all
  # subsequent runs, the service simply returns immediately, without performing
  # any modification to existing users.
  setup:
    build:
      context: setup/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    init: true
    volumes:
      - ./setup/entrypoint.sh:/entrypoint.sh:ro,Z
      - ./setup/lib.sh:/lib.sh:ro,Z
      - ./setup/roles:/roles:ro,Z
      - setup:/state:Z
    environment:
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD:-}
      LOGSTASH_INTERNAL_PASSWORD: ${LOGSTASH_INTERNAL_PASSWORD:-}
      KIBANA_SYSTEM_PASSWORD: ${KIBANA_SYSTEM_PASSWORD:-}
      METRICBEAT_INTERNAL_PASSWORD: ${METRICBEAT_INTERNAL_PASSWORD:-}
      FILEBEAT_INTERNAL_PASSWORD: ${FILEBEAT_INTERNAL_PASSWORD:-}
      HEARTBEAT_INTERNAL_PASSWORD: ${HEARTBEAT_INTERNAL_PASSWORD:-}
      MONITORING_INTERNAL_PASSWORD: ${MONITORING_INTERNAL_PASSWORD:-}
      BEATS_SYSTEM_PASSWORD: ${BEATS_SYSTEM_PASSWORD:-}
    networks:
      - elk
    depends_on:
      - elasticsearch

  elasticsearch:
    build:
      context: elasticsearch/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    volumes:
      - ./elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/elast
```

```
      - elasticsearch:/usr/share/elasticsearch/data:Z
    ports:
      - 9200:9200
      - 9300:9300
    environment:
      node.name: elasticsearch
      ES_JAVA_OPTS: -Xms512m -Xmx512m
      # Bootstrap password.
      # Used to initialize the keystore during the initial startup of
      # Elasticsearch. Ignored on subsequent runs.
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD:-}
      # Use single node discovery in order to disable production mode and avoid bootst
      # see: https://www.elastic.co/guide/en/elasticsearch/reference/current/bootstrap
      discovery.type: single-node
    networks:
      - elk
    restart: unless-stopped

  logstash:
    build:
      context: logstash/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    volumes:
      - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml:ro,Z
      - ./logstash/pipeline:/usr/share/logstash/pipeline:ro,Z
    ports:
      - 5044:5044
      - 50000:50000/tcp
      - 50000:50000/udp
      - 9600:9600
    environment:
      LS_JAVA_OPTS: -Xms256m -Xmx256m
      LOGSTASH_INTERNAL_PASSWORD: ${LOGSTASH_INTERNAL_PASSWORD:-}
    networks:
      - elk
    depends_on:
      - elasticsearch
    restart: unless-stopped

  kibana:
    build:
      context: kibana/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    volumes:
      - ./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml:ro,Z
    expose:
```

```
      - 5601
    environment:
      KIBANA_SYSTEM_PASSWORD: ${KIBANA_SYSTEM_PASSWORD:-}
    networks:
      - elk
    depends_on:
      - elasticsearch
    restart: unless-stopped

  nginx:
    image: nginx:latest
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - /etc/letsencrypt/:/etc/letsencrypt/
    ports:
      - 80:80
      - 443:443
    networks:
      - elk
    restart: unless-stopped

networks:
  elk:
    driver: bridge

volumes:
  setup:
  elasticsearch:
```

Config for nginx reverse-proxy serving Kibana web-interface:

```
# For more information on configuration, see:
#   * Official English Documentation: http://nginx.org/en/docs/
#   * Official Russian Documentation: http://nginx.org/ru/docs/

user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

http {
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
```

```
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
    keepalive_timeout   650;
    types_hash_max_size 2048;
    client_max_body_size 20m;

    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;

    # Load modular configuration files from the /etc/nginx/conf.d directory.
    # See http://nginx.org/en/docs/ngx_core_module.html#include
    # for more information.
    #include /etc/nginx/conf.d/*.conf;

    server {
        server_name c111-042.cloud.gwdg.de;
        listen 80;

        return 301 https://$host$request_uri;
    }

    server {
        server_name c111-042.cloud.gwdg.de;
        listen 443 ssl;
        ssl_certificate /etc/letsencrypt/live/c111-042.cloud.gwdg.de/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/c111-042.cloud.gwdg.de/privkey.pem;

        location / {
            proxy_pass http://kibana:5601;
        }
    }
}
```