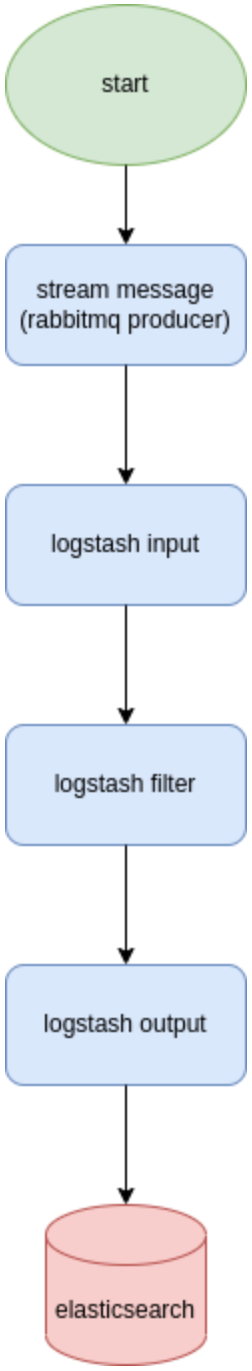
Elasticsearch data stream

Introduction

Having reasonable logging messages in the production helps us discover logs that would otherwise be undiscoverable. ELK (Elasticsearch Logstash Kibana) is one of the best platforms for handling desirable logs.

Abstract

We use elasticsearch as a central system where data can be kept and we can search among them easily. There are two ways to configure elasticsearch, single node and multi nodes. We use logstash for collecting data from multiple systems into our central system, in this case our central system is elasticsearch. We could save system logs directly in elasticsearch but we didn't because logstash can have multiple inputs and multiple outputs. Whenever We need, We can add a new input or output with a few configurations. In our case We use rabbitmq for input and elasticsearch for output. Why do We use rabbitmq?
We have multiple services and we don't want to gather logs in every services separately we just write a direct message in rabbitmq and send this message to a special exchange that is connected to input of our logstash and logstash filter and change our data in our desirable way and sending them to make a new document in elasticsearch. Another advantage of using rabbitmq is we won't have data lost with increasing requests because rabbitmq handles all requests in a queue.



Docker compose

We don't want to configure all these tools on our server, so we wrote a docker-compose.yml file that includes all these tools. We use elasticsearch, kibana, logstash and rabbitmq containers in our docker-compose file.

It is generally recommended that you use the same major version of elasticsearch, kibana and logstash. This will ensure that your data is compatible and can be easily searched and analyzed.

Start a single-node with docker compose

Setup

We set a **container_name**. In this container, we need a **command** to set our desired password for the kibana user. Setting a password for the kibana user is necessary for versions above 8.

```
setup:
    image: elasticsearch:${STACK_VERSION}
    container_name: setup
    command: >
      bash -c '
        echo "Setting kibana_system password";
        until curl -s -X POST -u
"${ELASTIC_USER:-}:${ELASTIC_PASSWORD:-}" -H "Content-Type:
application/json"
http://elasticsearch:9200/_security/user/kibana_system/_password -d
"{\"password\":\"${KIBANA_PASSWORD:-}\"}"; do sleep 10; done;
        echo "All done!";
      '
```

Elasticsearch

We set a **container_name** so other containers can know Elasticsearch with that exact name. This node should be **dependent on** the setup container. We set a **volume** to keep data after stopping or restarting this container. Elasticsearch uses two **ports** 9200 and 9300. Port 9200 is used for all API calls over HTTP. This includes search and aggregations, monitoring and anything else that uses a HTTP request. All client libraries will use this port to talk to Elasticsearch. Port

9300 is a custom binary protocol used for communications between nodes in a cluster. For things like cluster updates, master elections, nodes joining/leaving, shard allocation.

We need to set some **environment** variables.

- ES_JAVA_OPTS: This field considers the max memory size of elasticsearch that can be used and the value of this field is really related to the hardware.
- xpack.security.enabled: We don't turn security off so we should send the username and password of our elasticsearch.
- ELASTIC_USER: This user is used as a super admin of elasticsearch. We need this variable if security settings are enabled.
- ELASTIC_PASSWORD: This password is used as a super admin password of elasticsearch. We need this variable if security settings are enabled.
- network.publish.host: We can reach our elasticsearch with localhost:9200 address in its network but if we want to use it from another network we can set a host for it.
- discovery.type: In this case we use a single node.

We set a **health check** condition for this container too. This healthcheck checks the starting position of this container. curl to check one time per interval period time and try up to 120 times.

```
elasticsearch:
    image: elasticsearch:${STACK_VERSION}
    container_name: elasticsearch
    volumes:
      - elasticsearch:/usr/share/elasticsearch/data:z
    ports:
      - 9200:9200
      - 9300:9300
    environment:
      - ES_JAVA_OPTS=-Xms512m -Xmx512m
      - ELASTIC_USER=${ELASTIC_USER:-}
```

```
      - ELASTIC_PASSWORD=${ELASTIC_PASSWORD:-}
      - network.publish.host=${ELASTIC_HOST:-}
      - discovery.type=single-node
   healthcheck:
      test:
        [
          "CMD-SHELL",
          "curl -s http://localhost:9200 | grep -q 'missing
authentication credentials'"
        ]
      interval: 10s
      timeout: 10s
      retries: 120
```

Logstash

We set a **container_name** so other containers can know Logstash with that exact name. Logstash uses elasticsearch as an output so it **depends on** the health of elasticsearch container for starting. We set a **volume** to keep data after stopping or restarting this container. Logstash uses **port** 5044 mapping to access the services running inside a docker container. We need to set some **environment** variables.

- ES_JAVA_OPTS: This field considers the max memory size of logstash that can be used and the value of this field is really related to the hardware.
- xpack.monitoring.enabled: With this field we can specify that logstash uses our configuration for connecting to elasticsearch not its default configuration.
- CONFIG_STRING: we write our input and output config (pipeline configuration) in this field. We can write this config clearly in a separated file and set the path of that file in the volume part. If you use configuration file in a separated file do not forget to set configuration path in environment variable like PATH_CONFIG: "/etc/logstash/*.conf". This part

has three main parts: Input, Filter, Output. Variables in input and output are really related to the tools that are chosen. We can have multiple inputs and outputs. We set the exchange name and routing key in input. If we don't consider these fields, rabbitmq will listen to all messages. In the filter part we deserialize our variables and remove extra fields from the body. In the output we index our document dynamically. We consider whenever we receive a message from rabbitmq we receive a key with the name of the *domain* to know what is the new document index. For the filtering part there are so many options.

```
logstash:
    image: logstash:${STACK_VERSION}
    container_name: logstash
    depends_on:
      - elasticsearch
    volumes:
      - /var/log/GDPR/myapplication:/var/log/GDPR/myapplication:Z
    ports:
      - 5044:5044
    environment:
      - ES_JAVA_OPTS=-Xmx256m -Xms256m
      - xpack.monitoring.enabled=false
      - CONFIG_STRING=input { rabbitmq { host => "rabbitmq" port =>
5672 vhost => "/" password => "password" user => "username" queue =>
"projectname.log.queue" exchange => "projectname.log.exchange" key =>
"projectname.log.routing" exchange_type => "direct" type =>
"rabbitmq" durable => true codec => "json" } } filter { json { source
=> "message" } mutate { remove_field => ["@version", "type", "event",
"@timestamp"] } } output { elasticsearch { hosts =>
"elasticsearch:9200" user => "elastic" password => "elastic_password"
index => "%{domain}" action => "index" } }
```

Kinana

We set a **container_name** so other containers can know Kibana with that exact name. Kibana illustrates elasticsearch so it **depends on** the health of the

elasticsearch container for starting. We set a **volume** to keep data after stopping or restarting this container. By default, kibana uses **port** 560. Kibana needs elasticsearch password, elasticsearch username and elasticsearch hosts for internal connection to connect elasticsearch. Kibana can use the *kibana_system* username that exists in elasticsearch but we should set a password for that in elasticsearch before using the command is:

docker exec -it cd cd

The latest version for ELK is 8 in this but we use version 7 because in version 8 we can not use *elastic* username in kibana and we should set a password for *kibana_system* and we don't want to use the command line for docker compose configuration so we did this part in setup container. We need to set some **environment** variables.

- ELASTICSEARCH_HOSTS: Kibana uses this host to make connections with elasticsearch.
- ELASTICSERACH_USERNAME: This user is used to make an inner connection between elasticsearch and kibana of elasticsearch.
- ELASTICSEARCH_PASSWORD: This password is used to make an inner connection between elasticsearch and kibana of elasticsearch. We need this variable if security settings are enabled.

```
kibana:
    image: kibana:${STACK_VERSION}
    container_name: kibana
    depends_on:
      elasticsearch:
        condition: service_healthy
    volumes:
      - kibanadata:/usr/share/kibana/data:z
    ports:
      - 5601:5601
    environment:
```

```
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
      - ELASTICSEARCH_USERNAME=kibana_system
      - ELASTICSEARCH_PASSWORD=${KIBANA_PASSWORD:-}
```

Rabbitmq

We set a **container_name** so other containers can know rabbitmq with that exact name. We set a **volume** to keep data after stopping or restarting this container. Rabbitmq uses **port** 5672 for ip version 4 and 15672 for ip version 6. We set **restart** to start the container after stopping. We need to set some **environment** variables to change the default username and password of rabbitmq.

- RABBITMQ_DEFAULT_USER: A username for connecting to rabbitmq.
- RABBITMQ_DEFAULT_PASS: A password for connecting to rabbitmq.

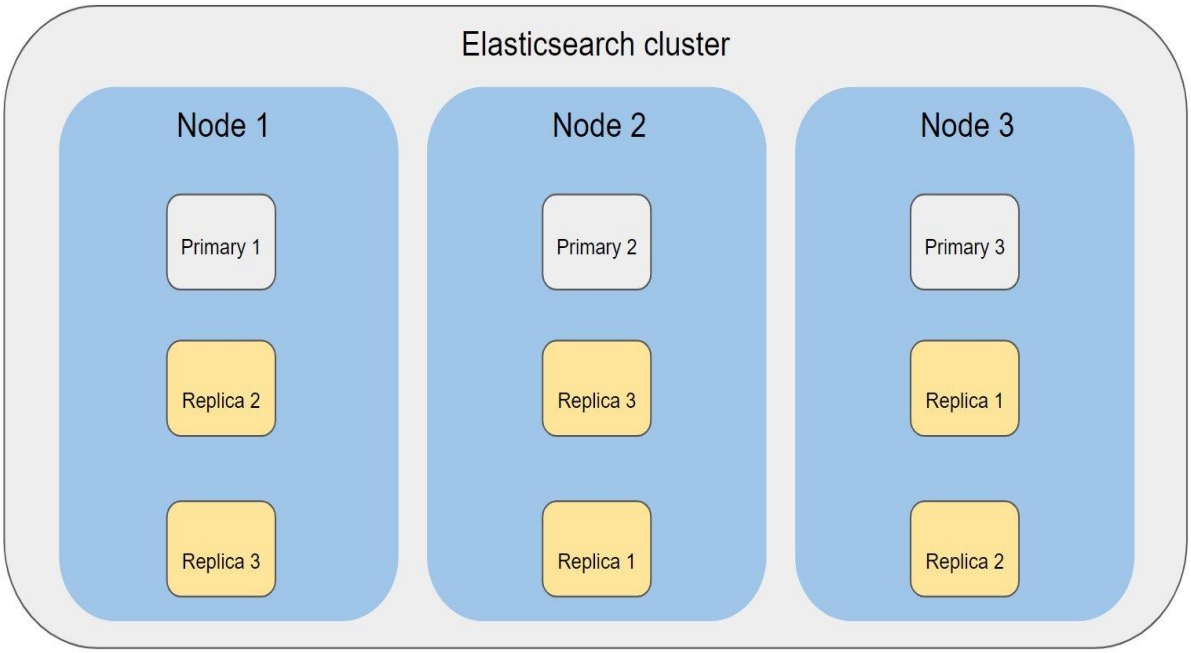
```
rabbitmq:
    image: rabbitmq:3-management
    container_name: rabbitmq
    volumes:
      - rabbitmq_data:/var/lib/rabbitmq/
      - rabbitmq_log:/var/log/rabbitmq/
    ports:
        - 5672:5672
        - 15672:15672
    restart: always
    environment:
      - RABBITMQ_DEFAULT_USER=${RABBITMQ_USER:-}
      - RABBITMQ_DEFAULT_PASS=${RABBITMQ_PASSWORD:-}
```

Overall tips

In input and output of logstash for elasticsearch and rabbitmq host we use the name of their container because all of them run in the same service with the same network and know each other by their container name.
If logstash faces an error for example for parsing data as a json it ignores that request.

Start a multi-node cluster with docker compose

Multi-node clustering in elasticsearch can have so many benefits. With that we can keep some replica shards. Replica shard is the copy of primary Shard , to prevent data loss in case of hardware failure. Elasticsearch allows you to make one or more copies of your index's shards into what are called replica shards. Replica shards should be kept in different nodes where the primary node is kept. The schema of keeping nodes should be like this:



If we want to use multi-clustering in elasticsearch, our docker compose should have some changes. We should consider a new container from elasticsearch image per node. We should use ssl connection if we want to use security in multi-node state. For this some configurations are added. Please consider that we can not use ssl connection in the free version until we use a version higher than 7. In our case we want to use two nodes so we have two containers from elasticsearch image. We have another image from elasticsearch to just set up ssl configurations.

Setup

We set a **container_name**. This container has a **volume** to keep the configs that other containers can use this config too. This container should have access to write in the config file so we set the root **user** for this container. In this container, we need a **command** because we should create a total certificate and a certificate per each node for this. First we check if these total certificates exist or not. If they don't exist we create them. We set permissions for these files so our containers can use these certificates. After that we send a request that checks if our first node is up. After that we can set our desired password for the kibana user. Setting a password for the kibana user is necessary for versions above 8. This container has a volume to keep these configs that other containers can use this config too. We set a **health check** condition for this container too. This healthcheck checks the starting position of this container. curl to check one time per interval period time and try up to 120 times.

```
setup:
    image: elasticsearch:${STACK_VERSION}
    container_name: setup
    volumes:
      - certs:/usr/share/elasticsearch/config/certs
    user: "0"
    command: >
      bash -c '
        if [ ! -f config/certs/ca.zip ]; then
          echo "Creating CA";
          bin/elasticsearch-certutil ca --silent --pem -out
config/certs/ca.zip;
          unzip config/certs/ca.zip -d config/certs;
        fi;
        if [ ! -f config/certs/certs.zip ]; then
          echo "Creating certs";
          echo -ne \
          "instances:\n"\
```

```
          "  - name: elasticsearch1\n"\
          "    dns:\n"\
          "      - elasticsearch1\n"\
          "      - localhost\n"\
          "    ip:\n"\
          "      - 127.0.0.1\n"\
          "  - name: elasticsearch2\n"\
          "    dns:\n"\
          "      - elasticsearch2\n"\
          "      - localhost\n"\
          "    ip:\n"\
          "      - 127.0.0.1\n"\
          > config/certs/instances.yml;
          bin/elasticsearch-certutil cert --silent --pem -out
config/certs/certs.zip --in config/certs/instances.yml --ca-cert
config/certs/ca/ca.crt --ca-key config/certs/ca/ca.key;
          unzip config/certs/certs.zip -d config/certs;
        fi;
        echo "Setting file permissions"
        chown -R root:root config/certs;
        find . -type d -exec chmod 750 \{\} \;;
        find . -type f -exec chmod 640 \{\} \;;
        echo "Waiting for Elasticsearch availability";
        until curl -s --cacert config/certs/ca/ca.crt
https://elasticsearch1:9200 | grep -q "missing authentication
credentials"; do sleep 30; done;
        echo "Setting kibana_system password";
        until curl -s -X POST --cacert config/certs/ca/ca.crt -u
"elastic:${ELASTIC_PASSWORD}" -H "Content-Type: application/json"
https://elasticsearch1:9200/_security/user/kibana_system/_password -d
"{\"password\":\"${KIBANA_PASSWORD}\"}"; do sleep 10; done;
        echo "All done!";
      '
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "[ -f config/certs/elasticsearch1/elasticsearch1.crt ]"
        ]
```

```
      interval: 1s
      timeout: 5s
      retries: 120
```

elasticsearch1

This container is our first node. We set a **container_name**. This node should be **dependent on** the setup container if it works correctly. We set a **volume** to keep data after stopping or restarting this container and configs. **Port** 9200 is used for all API calls over HTTP. In the **environment**, we should add some new variables. Variables with *xpack.security.http.ssl* prefix are related to connection of our node with http requests and variables with *xpack.security.transport.ss*l prefix are related to connection of our node with other nodes.

- ES_JAVA_OPTS: This field considers the max memory size of elasticsearch that can be used and the value of this field is really related to the hardware.
- ELASTIC_PASSWORD: This password is used as a super admin password of elasticsearch. We need this variable if security settings are enabled.
- node.name: It is our unique name for this node.
- cluster.name: It is our cluster name.
- cluster.initial_master_nodes: names of all nodes that exist in this cluster.
- discover.seed_hosts: names of other nodes.
- xpack.security.enabled: with this variable we turn on security in our system.
- xpack.security.http.ssl.enabled: with this variable we turn on ssl connection for http requests.
- xpack.lisence.self_generated.type: this variable declares type of license. We do not have any license so we set ***basic***.
- xpack.security.http.ssl.key and xpack.security.http.ssl.certificate: we address those certificates that we made in the setup container.

- xpack.security.http.ssl.verification_mode and xpack.security.transport.ssl.verification_mode: with this variable this container understands that the certificate that it uses is trusted.
- xpack.security.transport.ssl.enabled: with this variable we turn on ssl connection between our nodes.
- xpack.security.transport.ssl.key and xpack.security.transport.ssl.certificate: we address those certificates that we made in the setup container.

Ulimits establishes that our process is entitled to an unlimited amount of locked memory. Without establishing ulimits you will receive a container exit.

We set a **health check** condition for this container too. This healthcheck checks the starting position of this container. curl to check one time per interval period time and try up to 120 times.

```
elasticsearch1:
    image: elasticsearch:${STACK_VERSION}
    container_name: elasticsearch1
    depends_on:
      setup:
        condition: service_healthy
    volumes:
      - certs:/usr/share/elasticsearch/config/certs
      - elasticsearch1:/usr/share/elasticsearch/data:z
    ports:
      - 9200:9200
    environment:
      - ES_JAVA_OPTS=-Xms512m -Xmx512m
      - ELASTIC_PASSWORD=${ELASTIC_PASSWORD}
      - node.name=elasticsearch1
      - cluster.name=docker-cluster
      - cluster.initial_master_nodes=elasticsearch1,elasticsearch2
      - discovery.seed_hosts=elasticsearch2
      - xpack.security.enabled=true
      - xpack.license.self_generated.type=basic
```

```
      - xpack.security.http.ssl.enabled=true
      -
xpack.security.http.ssl.key=certs/elasticsearch1/elasticsearch1.key
      -
xpack.security.http.ssl.certificate=certs/elasticsearch1/elasticsearc
h1.crt
      -
xpack.security.http.ssl.certificate_authorities=certs/ca/ca.crt
      - xpack.security.http.ssl.verification_mode=certificate
      - xpack.security.transport.ssl.enabled=true
      -
xpack.security.transport.ssl.key=certs/elasticsearch1/elasticsearch1.
key
      -
xpack.security.transport.ssl.certificate=certs/elasticsearch1/elastic
search1.crt
      -
xpack.security.transport.ssl.certificate_authorities=certs/ca/ca.crt
      - xpack.security.transport.ssl.verification_mode=certificate
    ulimits:
      memlock:
        soft: -1
        hard: -1
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "curl -s --cacert config/certs/ca/ca.crt
https://localhost:9200 | grep -q 'missing authentication
credentials'"
        ]
      interval: 10s
      timeout: 10s
      retries: 120
```

elasticsearch2

This container is our second node. We set a **container_name.** This node should be dependent on the elasticsearch1 container. We set a **volume** to keep data

after stopping or restarting this container and configs. **Port** 9200 is used for all API calls over HTTP. In the **environment**, we should add some new variables. Variables with *xpack.security.http.ssl* prefix are related to connection of our node with http requests and variables with *xpack.security.transport.ss*l prefix are related to connection of our node with other nodes.

- ES_JAVA_OPTS: This field considers the max memory size of elasticsearch that can be used and the value of this field is really related to the hardware.
- node.name: It is our unique name for this node.
- cluster.name: It is our cluster name.
- cluster.initial_master_nodes: names of all nodes that exist in this cluster.
- discover.seed_hosts: names of other nodes.
- xpack.security.enabled: with this variable we turn on security in our system.
- xpack.security.http.ssl.enabled: with this variable we turn on ssl connection for http requests.
- xpack.lisence.self_generated.type: this variable declares type of license. We do not have any license so we set ***basic***.
- xpack.security.http.ssl.key and xpack.security.http.ssl.certificate: we address those certificates that we made in the setup container.
- xpack.security.http.ssl.verification_mode and xpack.security.transport.ssl.verification_mode: with this variable this container understands that the certificate that it uses is trusted.
- xpack.security.transport.ssl.enabled: with this variable we turn on ssl connection between our nodes.
- xpack.security.transport.ssl.key and xpack.security.transport.ssl.certificate: we address those certificates that we made in the setup container.

Ulimits establishes that our process is entitled to an unlimited amount of locked memory. Without establishing ulimits you will receive a container exit.

We set a **health check** condition for this container too. This healthcheck checks the starting position of this container. curl to check one time per interval period time and try up to 120 times.

```
elasticsearch2:
    image: elasticsearch:${STACK_VERSION}
    container_name: elasticsearch2
    depends_on:
      - elasticsearch1
    volumes:
      - certs:/usr/share/elasticsearch/config/certs
      - elasticsearch2:/usr/share/elasticsearch/data:z
    environment:
      - ES_JAVA_OPTS=-Xms512m -Xmx512m
      - node.name=elasticsearch2
      - cluster.name=docker-cluster
      - cluster.initial_master_nodes=elasticsearch1,elasticsearch2
      - discovery.seed_hosts=elasticsearch1
      - xpack.security.enabled=true
      - xpack.security.http.ssl.enabled=true
      -
xpack.security.http.ssl.key=certs/elasticsearch2/elasticsearch2.key
      -
xpack.security.http.ssl.certificate=certs/elasticsearch2/elasticsearc
h2.crt
      -
xpack.security.http.ssl.certificate_authorities=certs/ca/ca.crt
      - xpack.security.http.ssl.verification_mode=certificate
      - xpack.security.transport.ssl.enabled=true
      -
xpack.security.transport.ssl.key=certs/elasticsearch2/elasticsearch2.
key
      -
xpack.security.transport.ssl.certificate=certs/elasticsearch2/elastic
search2.crt
      -
```

```
xpack.security.transport.ssl.certificate_authorities=certs/ca/ca.crt
      - xpack.security.transport.ssl.verification_mode=certificate
      - xpack.license.self_generated.type=basic
    ulimits:
      memlock:
        soft: -1
        hard: -1
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "curl -s --cacert config/certs/ca/ca.crt
https://localhost:9200 | grep -q 'missing authentication
credentials'"
        ]
      interval: 10s
      timeout: 10s
      retries: 120
```

logstash

Logstash doesn't have too many differences from a single node state.
We set a **container_name** so other containers can know Logstash with that exact name. Logstash uses elasticsearch as an output so it **depends on** the health of elasticsearch1 and elasticsearch2 container for starting. We set a **volume** to keep data and configs after stopping or restarting this container. Logstash uses **port** 5044 mapping to access the services running inside a docker container. We need to set some **environment** variables.

- ES_JAVA_OPTS: This field considers the max memory size of logstash that can be used and the value of this field is really related to the hardware.
- xpack.monitoring.enabled: With this field we can specify that logstash uses our configuration for connecting to elasticsearch not its default configuration.

- CONFIG_STRING: we write our input and output config (pipeline configuration) in this field. We can write this config clearly in a separated file and set the path of that file in the volume part. If you use configuration file in a separated file do not forget to set configuration path in environment variable like PATH_CONFIG: "/etc/logstash/*.conf". This part has three main parts: Input, Filter, Output. Variables in input and output are really related to the tools that are chosen. We can have multiple inputs and outputs. We set the exchange name and routing key in input. If we don't consider these fields, rabbitmq will listen to all messages. In the filter part we deserialize our variables and remove extra fields from the body. In the output we index our document dynamically. We consider whenever we receive a message from rabbitmq we receive a key with the name of the *domain* to know what is the new document index. For the filtering part there are so many options.

```
logstash:
    image: logstash:${STACK_VERSION}
    container_name: logstash
    depends_on:
      elasticsearch1:
        condition: service_healthy
      elasticsearch2:
        condition: service_healthy
    volumes:
      - /var/log/GDPR/myapplication:/var/log/GDPR/myapplication:ro,Z
      - certs:/usr/share/logstash/config/certs
    ports:
      - 5044:5044
    environment:
      - ES_JAVA_OPTS=-Xmx256m -Xms256m
      - xpack.monitoring.enabled=false
      - CONFIG_STRING=input { rabbitmq { host => "rabbitmq" port =>
5672 vhost => "/" password => "password" user => "username" queue =>
"projectname.log" exchange => "projectname.log" key => "log.routing"
```

```
exchange_type => "direct" type => "rabbitmq" durable => true codec =>
"json" } } filter { json { source => "message" } mutate {
remove_field => ["@version", "type", "event", "@timestamp"] } }
output { elasticsearch { ssl => "true" ssl_certificate_verification
=> "false" hosts => ["elasticsearch1:9200"] user => "elastic"
password => "elastic_password" index => "%{domain}" action => "index"
} }
      -
ELASTICSEARCH_SSL_CERTIFICATEAUTHORITIES=config/certs/ca/ca.crt
```

kibana

Kibana doesn't have too many differences from a single node state. We set a **container_name** so other containers can know Kibana with that exact name. Kibana illustrates elasticsearch so it **depends on** the health of elasticsearch1 and elasticsearch2 container for starting. We set a **volume** to keep data and configs after stopping or restarting this container. By default, kibana uses **port** 560. Kibana needs elasticsearch password and elasticsearch username and elasticsearch hosts for internal connection to connect elasticsearch. Kibana can use the *kibana_system* username that exists in elasticsearch. The latest version for ELK is 8 in this but we use version 7 because in version 8 we can not use *elastic* username in kibana and we should set a password for *kibana_system* and we don't want to use the command for docker compose configuration. We need to set some **environment** variables.

- SERVERNAME: name of kibana server.
- ELASTICSEARCH_HOSTS: Kibana uses this host to make connections with elasticsearch.
- ELASTICSERACH_USERNAME: This user is used to make an inner connection between elasticsearch and kibana of elasticsearch.

- ELASTICSEARCH_PASSWORD: This password is used to make an inner connection between elasticsearch and kibana of elasticsearch. We need this variable if security settings are enabled.
- ELASTICSEARCH_SSL_CERTIFICATEAUTHORITIES: add path of ssl certificate.

```
kibana:
    image: kibana:${STACK_VERSION}
    container_name: kibana
    depends_on:
      elasticsearch1:
        condition: service_healthy
      elasticsearch2:
        condition: service_healthy
    volumes:
      - certs:/usr/share/kibana/config/certs
      - kibanadata:/usr/share/kibana/data
    ports:
      - 5601:5601
    environment:
      - SERVERNAME=kibana
      - ELASTICSEARCH_HOSTS=https://elasticsearch1:9200
      - ELASTICSEARCH_USERNAME=kibana_system
      - ELASTICSEARCH_PASSWORD=${KIBANA_PASSWORD}
      -
ELASTICSEARCH_SSL_CERTIFICATEAUTHORITIES=config/certs/ca/ca.crt
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "curl -s -I http://localhost:5601 | grep -q 'HTTP/1.1 302
Found'",
        ]
      interval: 10s
      timeout: 10s
      retries: 120
```

Rabbitmq

Rabbitmq doesn't have too many differences from a single node state. We set a **container_name** so other containers can know rabbitmq with that exact name. We set a **volume** to keep data after stopping or restarting this container. Rabbitmq uses **port** 5672 for ip version 4 and 15672 for ip version 6. We set **restart** to start the container after stopping. We need to set some **environment** variables to change the default username and password of rabbitmq.

- RABBITMQ_DEFAULT_USER: A username for connecting to rabbitmq.
- RABBITMQ_DEFAULT_PASS: A password for connecting to rabbitmq.

```
rabbitmq:
    image: rabbitmq:3-management
    container_name: rabbitmq
    volumes:
      - rabbitmq_data:/var/lib/rabbitmq/
      - rabbitmq_log:/var/log/rabbitmq/
    ports:
        - 5672:5672
        - 15672:15672
    restart: always
    environment:
      - RABBITMQ_DEFAULT_USER=${RABBITMQ_USER:-}
      - RABBITMQ_DEFAULT_PASS=${RABBITMQ_PASSWORD:-}
```

Conclusion

In this article, we learned how to successfully deploy a multi-node Elasticsearch cluster or single-node Elasticsearch cluster, along with a companion Kibana UI and Logstash and using rabbitmq for input of our logs. We illustrated how nodes can communicate with one another.