Quarkus - Centralized log management (Graylog, Logstash, Fluentd)

This guide explains how you can send your logs to a centralized log management system like Graylog, Logstash (inside the Elastic Stack or ELK - Elasticsearch, Logstash, Kibana) or Fluentd (inside EFK - Elasticsearch, Fluentd, Kibana).

There are a lot of different ways to centralize your logs (if you are using Kubernetes, the simplest way is to log to the console and ask you cluster administrator to integrate a central log manager inside your cluster). In this guide, we will expose how to send them to an external tool using the `quarkus-logging-gelf` extension that can use TCP or UDP to send logs in the Graylog Extended Log Format (GELF).

The `quarkus-logging-gelf` extension will add a GELF log handler to the underlying logging backend that Quarkus uses (jboss-logmanager). By default, it is disabled, if you enable it but still use another handler (by default the console handler is enabled), your logs will be sent to both handlers.

Example application

The following examples will all be based on the same example application that you can create with the following steps:

- Create an application with the `quarkus-logging-gelf` extension. You can use the following Maven command to create it:

```
mvn io.quarkus:quarkus-maven-plugin:1.3.0.Alpha1:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=gelf-logging \
    -DclassName="org.acme.quickstart.GelfLoggingResource" \
    -Dpath="/gelf-logging" \
    -Dextensions="logging-gelf"
```

- For demonstration purposes, we create an endpoint that does nothing but log a sentence. You don't need to do this inside your application.

```java
import javax.enterprise.context.ApplicationScoped;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

import org.jboss.logging.Logger;

@Path("/gelf-logging")
@ApplicationScoped
public class GelfLoggingResource {
    private static final Logger LOG = Logger.getLogger
(GelfLoggingResource.class);

    @GET
    public void log() {
        LOG.info("Some useful log message");
    }

}
```

- Configure the GELF log handler to send logs to an external UDP endpoint on the port 12201:

```
quarkus.log.handler.gelf.enabled=true
quarkus.log.handler.gelf.host=localhost
quarkus.log.handler.gelf.port=12201
```

Send logs to Graylog

To send logs to Graylog, you first need to launch the components that compose the Graylog stack:

- MongoDB

- Elasticsearch

- Graylog

You can do this via the following docker-compose file that you can launch via `docker-compose run -d`:

```yaml
version: '3.2'

services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch-oss:6.8.2
    ports:
      - "9200:9200"
    environment:
      ES_JAVA_OPTS: "-Xms512m -Xmx512m"
    networks:
      - graylog

  mongo:
    image: mongo:4.0
    networks:
      - graylog

  graylog:
    image: graylog/graylog:3.1
    ports:
      - "9000:9000"
      - "12201:12201/udp"
      - "1514:1514"
    environment:
      GRAYLOG_HTTP_EXTERNAL_URI: "http://127.0.0.1:9000/"
    networks:
      - graylog
    depends_on:
      - elasticsearch
      - mongo

networks:
  graylog:
    driver: bridge
```

Then, you need to create a UDP input in Graylog. You can do it from the Graylog web console (System → Input → Select GELF UDP) available at http://localhost:9000 or via the API.

This curl example will create a new Input of type GELF UDP, it uses the default login from Graylog (admin/admin).

```
curl -H "Content-Type: application/json" -H "Authorization: Basic
YWRtaW46YWRtaW4=" -H "X-Requested-By: curl" -X POST -v -d \
'{"title":"udp
input","configuration":{"recv_buffer_size":262144,"bind_address":"0
.0.0.0","port":12201,"decompress_size_limit":8388608},"type":"org.g
raylog2.inputs.gelf.udp.GELFUDPInput","global":true}' \
http://localhost:9000/api/system/inputs
```

Launch your application, you should see your logs arriving inside Graylog.

Send logs to Logstash / the Elastic Stack (ELK)

Logstash comes by default with an Input plugin that can understand the GELF format, we will first create a pipeline that enables this plugin.

Create the following file in $HOME/pipelines/gelf.conf:

```
input {
  gelf {
    port => 12201
  }
}
output {
  stdout {}
  elasticsearch {
    hosts => ["http://elasticsearch:9200"]
  }
}
```

Finally, launch the components that compose the Elastic Stack:

- Elasticsearch

- Logstash

- Kibana

You can do this via the following docker-compose file that you can launch via `docker-compose run -d`:

```yaml
# Launch Elasticsearch
version: '3.2'

services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch-oss:6.8.2
    ports:
      - "9200:9200"
      - "9300:9300"
    environment:
      ES_JAVA_OPTS: "-Xms512m -Xmx512m"
    networks:
      - elk

  logstash:
    image: docker.elastic.co/logstash/logstash-oss:6.8.2
    volumes:
      - source: $HOME/pipelines
        target: /usr/share/logstash/pipeline
        type: bind
    ports:
      - "12201:12201/udp"
      - "5000:5000"
      - "9600:9600"
    networks:
      - elk
    depends_on:
      - elasticsearch

  kibana:
    image: docker.elastic.co/kibana/kibana-oss:6.8.2
    ports:
      - "5601:5601"
    networks:
      - elk
    depends_on:
      - elasticsearch

networks:
  elk:
    driver: bridge
```

Launch your application, you should see your logs arriving inside the Elastic Stack; you can use Kibana available at http://localhost:5601/ to access them.

Send logs to Fluentd (EFK)

First, you need to create a Fluentd image with the needed plugins: elasticsearch and input-gelf. You can use the following Dockerfile that should be created inside a `fluentd` directory.

```
FROM fluent/fluentd:v1.3-debian
RUN ["gem", "install", "fluent-plugin-elasticsearch", "--version",
"3.7.0"]
RUN ["gem", "install", "fluent-plugin-input-gelf", "--version",
"0.3.1"]
```

You can build the image or let docker-compose build it for you.

Then you need to create a fluentd configuration file inside `$HOME/fluentd/fluent.conf`

```
<source>
  type gelf
  tag example.gelf
  bind 0.0.0.0
  port 12201
</source>

<match example.gelf>
  @type elasticsearch
  host elasticsearch
  port 9200
  logstash_format true
</match>
```

Finally, launch the components that compose the EFK Stack:

- Elasticsearch

- Fluentd

- Kibana

You can do this via the following docker-compose file that you can launch via `docker-compose run -d`:

```yaml
version: '3.2'

services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch-oss:6.8.2
    ports:
      - "9200:9200"
      - "9300:9300"
    environment:
      ES_JAVA_OPTS: "-Xms512m -Xmx512m"
    networks:
      - efk

  fluentd:
    build: fluentd
    ports:
      - "12201:12201/udp"
    volumes:
      - source: $HOME/fluentd
        target: /fluentd/etc
        type: bind
    networks:
      - efk
    depends_on:
      - elasticsearch

  kibana:
    image: docker.elastic.co/kibana/kibana-oss:6.8.2
    ports:
      - "5601:5601"
    networks:
      - efk
    depends_on:
      - elasticsearch

networks:
  efk:
    driver: bridge
```

Launch your application, you should see your logs arriving inside EFK: you can use Kibana available at http://localhost:5601/ to access them.

Fluentd alternative: use Syslog

You can also send your logs to Fluentd using a Syslog input. As opposed to the GELF input, the Syslog input will not render multiline logs in one event, that's why we advise to use the GELF input that we implement in Quarkus.

First, you need to create a Fluentd image with the elasticsearch plugin. You can use the following Dockerfile that should be created inside a `fluentd` directory.

```
FROM fluent/fluentd:v1.3-debian
RUN ["gem", "install", "fluent-plugin-elasticsearch", "--version",
"3.7.0"]
```

Then, you need to create a fluentd configuration file inside `$HOME/fluentd/fluent.conf`

```
<source>
  @type syslog
  port 5140
  bind 0.0.0.0
  message_format rfc5424
  tag system
</source>

<match **>
  @type elasticsearch
  host elasticsearch
  port 9200
  logstash_format true
</match>
```

Then, launch the components that compose the EFK Stack:

- Elasticsearch

- Fluentd

- Kibana

You can do this via the following docker-compose file that you can launch via `docker-compose run -d`:

```yaml
version: '3.2'

services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch-oss:6.8.2
    ports:
      - "9200:9200"
      - "9300:9300"
    environment:
      ES_JAVA_OPTS: "-Xms512m -Xmx512m"
    networks:
      - efk

  fluentd:
    build: fluentd
    ports:
      - "5140:5140/udp"
    volumes:
      - source: $HOME/fluentd
        target: /fluentd/etc
        type: bind
    networks:
      - efk
    depends_on:
      - elasticsearch

  kibana:
    image: docker.elastic.co/kibana/kibana-oss:6.8.2
    ports:
      - "5601:5601"
    networks:
      - efk
    depends_on:
      - elasticsearch
networks:
  efk:
    driver: bridge
```

Finally, configure your application to send logs to EFK using Syslog:

```
quarkus.log.syslog.enable=true
quarkus.log.syslog.endpoint=localhost:5140
quarkus.log.syslog.protocol=udp
quarkus.log.syslog.app-name=quarkus
quarkus.log.syslog.hostname=quarkus-test
```

Launch your application, you should see your logs arriving inside EFK: you can use Kibana available at http://localhost:5601/ to access them.

Configuration Reference

Configuration is done through the usual `application.properties` file.

▤ Configuration property fixed at build time - ⚙ Configuration property overridable at runtime

Configuration property	Type	Default
⚙ `quarkus.log.handler.gelf.enabled` Determine whether to enable the GELF logging handler	boolean	`false`
⚙ `quarkus.log.handler.gelf.host` Hostname/IP-Address of the Logstash/Graylog Host By default it uses UDP, prepend tcp: to the hostname to switch to TCP, example: "tcp:localhost"	string	`localhost`
⚙ `quarkus.log.handler.gelf.port` The port	int	`12201`
⚙ `quarkus.log.handler.gelf.version` GELF version: 1.0 or 1.1	string	`1.1`
⚙ `quarkus.log.handler.gelf.extract-stack-trace` Whether to post Stack-Trace to StackTrace field.	boolean	`true`
⚙ `quarkus.log.handler.gelf.stack-trace-throwable-reference` Only used when `extractStackTrace` is `true`. A value of 0 will extract the whole stack trace. Any positive value will walk the cause chain: 1 corresponds with exception.getCause(), 2 with exception.getCause().getCause(), … Negative throwable reference walk the exception chain from the root cause side: -1 will extract the root cause, -2 the exception wrapping the root cause, …	int	`0`
⚙ `quarkus.log.handler.gelf.filter-stack-trace` Whether to perform Stack-Trace filtering	boolean	`false`

⚙ `quarkus.log.handler.gelf.timestamp-pattern` Java date pattern, see `java.text.SimpleDateFormat`	string	`yyyy-MM-dd HH:mm:ss,SSS`
⚙ `quarkus.log.handler.gelf.level` The logging-gelf log level.	Level	`ALL`
⚙ `quarkus.log.handler.gelf.facility` Name of the facility.	string	`jboss-logmanager`
⚙ `quarkus.log.handler.gelf.include-full-mdc` Whether to include all fields from the MDC.	boolean	`false`
Post additional fields	**Type**	**Default**
⚙ `quarkus.log.handler.gelf.additional-field."field-name".value` Additional field value.	string	required ⊗
⚙ `quarkus.log.handler.gelf.additional-field."field-name".type` Additional field type specification. Supported types: String, long, Long, double, Double and discover. Discover is the default if not specified, it discovers field type based on parseability.	string	`discover`

This extension uses the `logstash-gelf` library that allow more configuration options via system properties, you can access its documentation here: https://logging.paluch.biz/ .