# CS 237 PROJECT REPORT

# LOGGING AS A SERVICE (LaaS)

Group 10

Aditya Harit
Akshay V. Bhandiwad
Shweta Iyer

## 1. KEY OBJECTIVE AND PROPOSED WORK:

In recent times, there has been a huge increase in the amount of data produced, a significant fraction of which are logs. Logs are used for running the services producing consumer-centric data. As more and more organizations adopt micro-service architecture, this leads to a scenario wherein we have multiple logs, coming from different services for the same applications. This necessitates the need for log parsing, transformation and storage systems.

Syslog log messages give a visual history of everything happening on a Linux machine. These logs can be generated by various user space applications or even by the Linux kernel. They follow a standard format containing the event details (like timestamp, severity etc.) and can be used to debug any faults and failures which might have occured in the system. It is, thus, important for a system administrator or a customer support executive (in case of a B2B set up) to get notified of any errors, failures or faults.

We build a real-time log server system to provide *Logging as a Service*. Our logging service notifies the system administrator when a fault or failure occurs in the system. The administrator can set the rules for notifications. We use syslog-ng as the main service to send the log messages from the clients to the servers. We also use **Kafka** and **KSQL** to process and store logs, and give real-time notifications to the administrator via **Email** or as **Slack** notifications. Further, this log data is sent to **ELK** stack (Elasticsearch, Logstash and Kibana) to get a real-time visualization of the events in the system to be monitored. This can be useful in areas where applications are running in a distributed fashion in a cluster of servers. It can be used to monitor a cluster of independent applications as well. Thus, logging is provided as a service.

## 2. RELATED WORK:

The Syslog protocol is a global standard for message logging. This protocol helps to separate the various attributes involved in logging such as the process generating the logs, the system which stores the logs and the system which analyzes the logs. Each Syslog message represents an event or a part of an event occurring in the system, which may be a successful event or a fault/failure. Syslog uses a client-server architecture where a Syslog server listens for and logs messages coming from clients. **Syslog-ng** is an open source implementation of the Syslog protocol for Linux and other Unix based systems. We use the Kafka based destination of Syslog-ng in our implementation to send the logs from the various client machines to the Kafka cluster. The communication between the syslog client and server can be through TCP or UDP. Our implementation uses TCP based communication. Syslog-ng also allows filtering of logs received from the source based on the process-name, severity, string matching etc. We use the process-name filter to forward the logs belonging to a particular application to its own Kafka topic.

**Apache Kafka** is a publish-subscribe based durable messaging system exchanging data between processes, applications, and servers. It can store streams of record in a fault tolerant way and process them as they occur. These streams of records are stored in categories called ‘topics’ in the Kafka cluster. Producers publish data of their choice to a given topic. Each record published to a topic is delivered to the subscribed consumer group. Kafka is run as a cluster on one or more servers that can span multiple datacenters. Kafka is a distributed messaging system designed to collect and deliver high volumes of log data with low latency. We employ this aspect of Kafka, feeding it with logs from the remote Syslog, and deliver these logs to ELK stack for better visualization of the same.

The **ELK** stack is made up of three applications: *Elasticsearch*, *Logstash* and ***Kibana***. Kibana is a visualization tool built on top of Elasticsearch, which lets a user visualize the data pictorially, using charts and graphs. Its main view is divided into four parts. The *discover* and *visualize* parts are used to query and present data, while the *dashboard* part combines them. The *management* part is responsible for configuring Kibana internals, like index patterns.

## 3. DESIGN AND ARCHITECTURE:

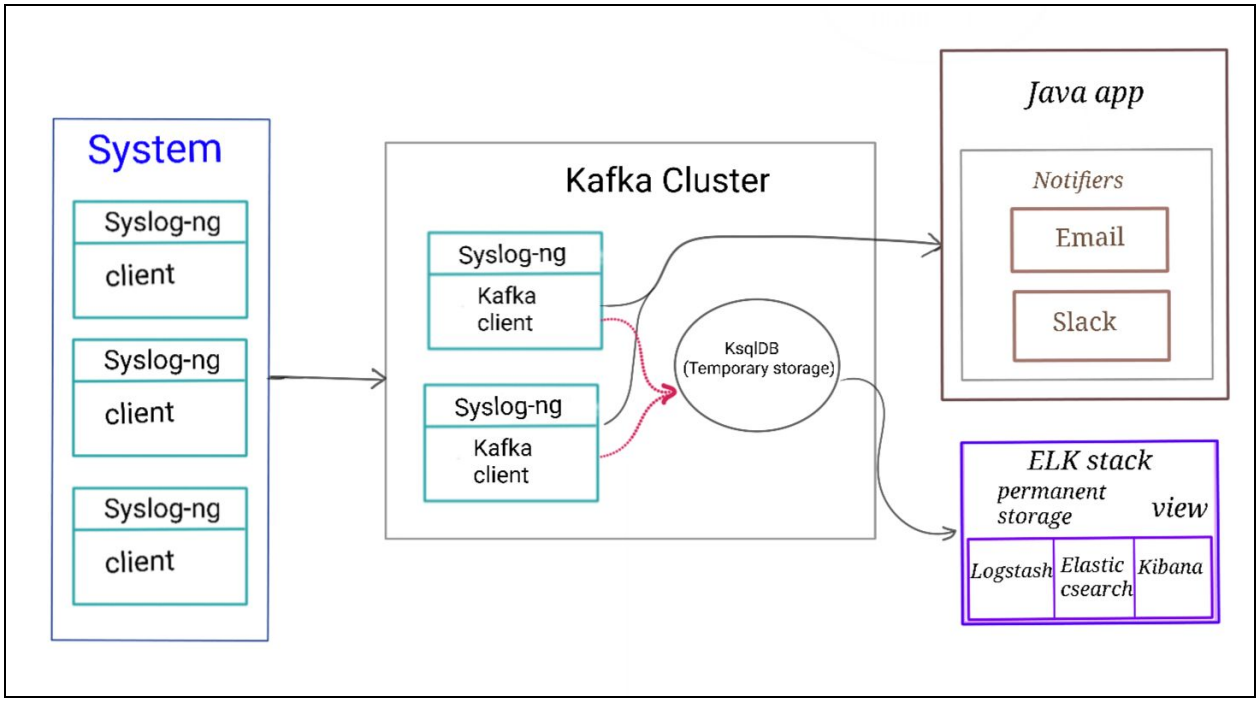


**Fig. 1**: Architecture of LaaS

In our architecture, we have a cluster of log servers. We adopt the Syslog-ng implementation of Syslog. The other alternative includes rSyslog which is also widely used. The flexibility and the ease of configuration of Syslog-ng is the primary reason it was chosen. Each log server receives logs from the clients via the remote Syslog method. The Syslog-ng in the log server then forwards the logs to a Kafka destination on the same machine.

Our Kafka cluster is made up of multiple servers running on Google cloud which receive logs from the clients via the remote Syslog method. We have one topic for each application to which events relating to that application are pushed by the Syslog-ng on the log server. This is achieved using the filter functionality of Syslog-ng. After the logs are pushed to the cluster, we use KSQL database to process them. In this step, the logs are parsed, and predefined keywords are extracted from them. After the logs have been modified by the KSQL database, they are sent to the ELK stack. Here, Logstash receives the logs, and simply forwards them to Elasticsearch. After the logs are sent to Elasticsearch, we can create indices on them to enable faster searching. The searching in our application is done primarily by Kibana, leveraging the powerful JSON-based Query DSL that Elasticsearch provides. We also create some widgets which focus on certain events, like CRITICAL or ERROR logs.

Simultaneously with the forwarding of the log data to Elasticsearch, we have subscribers to particular topics in the KSQL database which are given a callback when changes occur in the subscribed topics. The subscriber can take various actions on callback such as sending an Email notification or a Slack message to the system administrator. These prove to be very useful services required for monitoring and debugging a system at the time of failure.

## 4. IMPLEMENTATION:

We have deployed our implementation on a cluster of Google cloud machines. We have a total of 3 cloud machines which serve the following purposes:

Google Cloud VM details:

- OS - Ubuntu 18.04
- 1 vCPU

Instance 1 and Instance 2

- Act as a server
- Receive log traffic via remote syslog from the client
- Send the received remote syslog message to a Kafka topic
- Send Slack and Email notifications on receipt of error messages in the Kafka topic
- Send the log messages to KSQL and then to the ELK stack for visualization
- Open a port 5601 for Kibana dashboard to be accessed

Instance 3

- Acts as a client
- A python script is used to send a bunch of log messages.
- Sends these messages to the servers using a round robin load balancing mechanism.

The Syslog messages are simulated using a Python script run on a cloud instance, serving as a client. The syntax of the Syslog messages is:
**<Timestamp>; <Severity>; <Hostname>; <Process name>; <Message>**
Example:
**2020-06-11T06:09:39+00:00; err; instance-2; user; New Log message sample 2**

The client generates a large number of Syslog messages, which are distributed across two server instances using the Round Robin technique for load balancing via the remote syslog destination feature of syslog-ng. The servers have both Syslog as well as Kafka consumers running. Each server receives the log messages through remote syslog and sends these messages to the Kafka daemon running on the same server by using the Kafka destination feature of syslog-ng. The message received is parsed and stored in the KSQL where the tags are generated for these messages. Simultaneously, we also send the messages in a set format from Kafka to Slack and Email, if the severity of the log is either ERR (error), CRIT (critical) or EMERG (emergency). The messages are sent from the KSQL to the ELK stack for visualization.

## 5. RESULTS:

Here are some sample screen shots from our application:

```
>Jun 10 18:00:00; err; instance1; systemd; test message
>Jun 10 18:00:00; err; instance1; systemd; test message234
>Jun 10 18:00:00; err; instance1; systemd; test message testinggg
>Jun 10 18:00:00; err; instance1; systemd; test message testinggg email
>Jun 10 18:00:00; err; instance1; systemd; test message testinggg 123
>Jun 10 18:00:00; err; instance1; systemd; test message testinggg email 12
>Jun 10 18:00:00; err; instance1; systemd; test message email test
>Jun 10 18:00:00; err; instance1; systemd; test message test email one
>Jun 10 18:00:00; err; instance1; systemd; test message
>Jun 10 18:00:00; err; instance1; systemd; test message123
>Jun 10 18:00:00; err; instance1; systemd; test message email
>Jun 10 18:00:00; err; instance1; systemd; test message testinggg email1
```

**Fig. 2**: Syslog messages sent from client

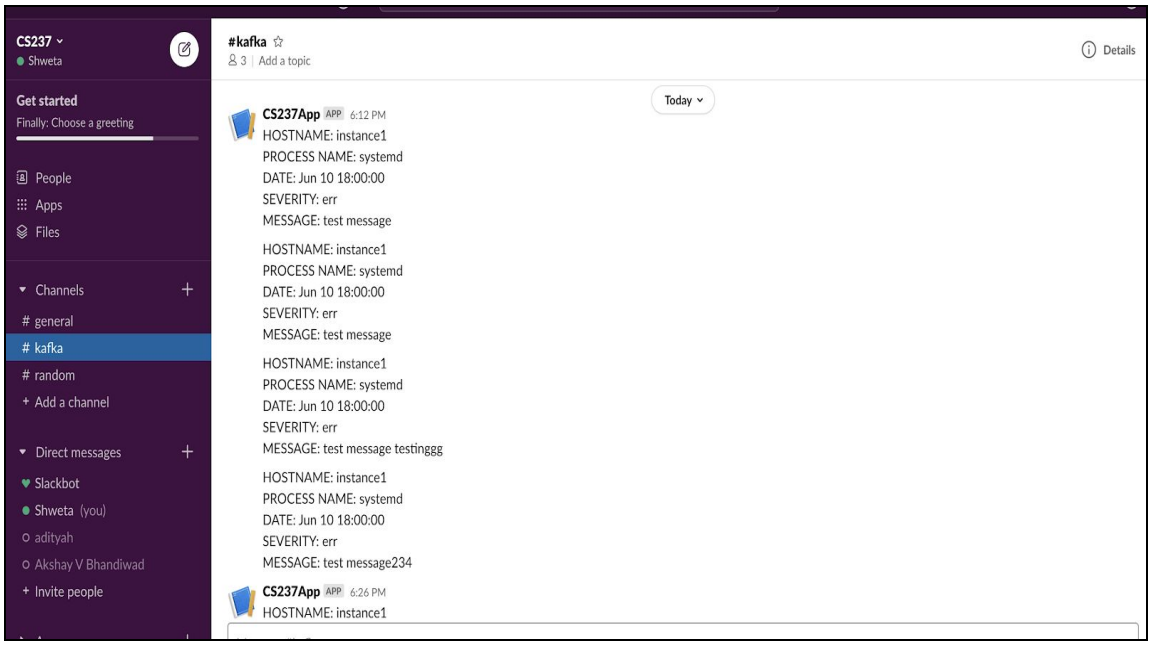


**Fig. 3**: Slack channel ‘Kafka’ receiving messages from Kafka instance

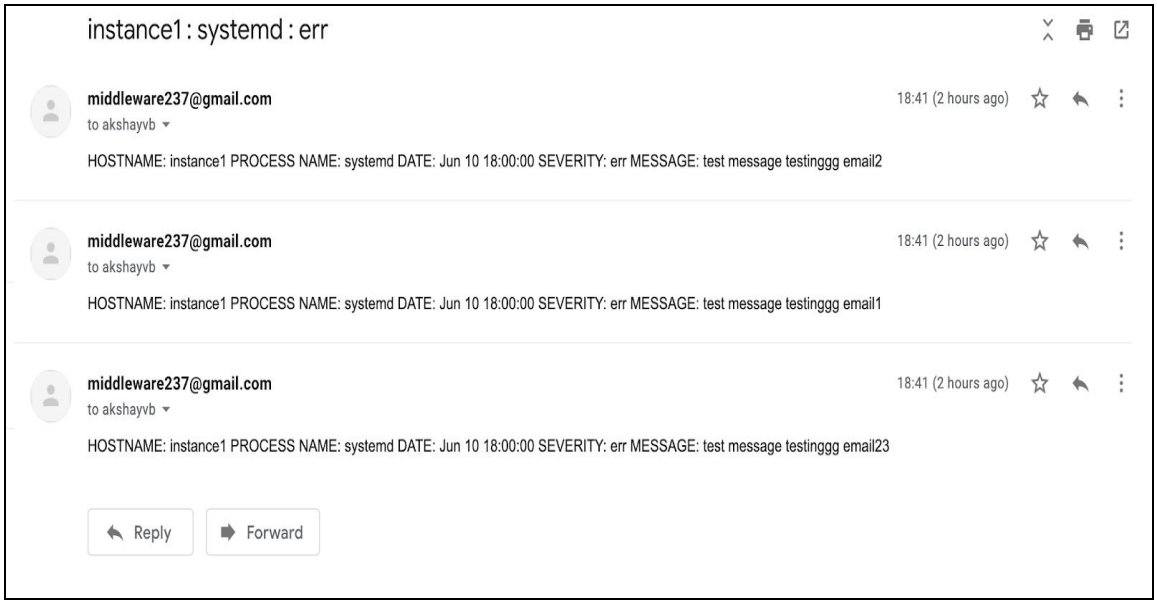


**Fig. 4**: Email notifications received from Kafka instance

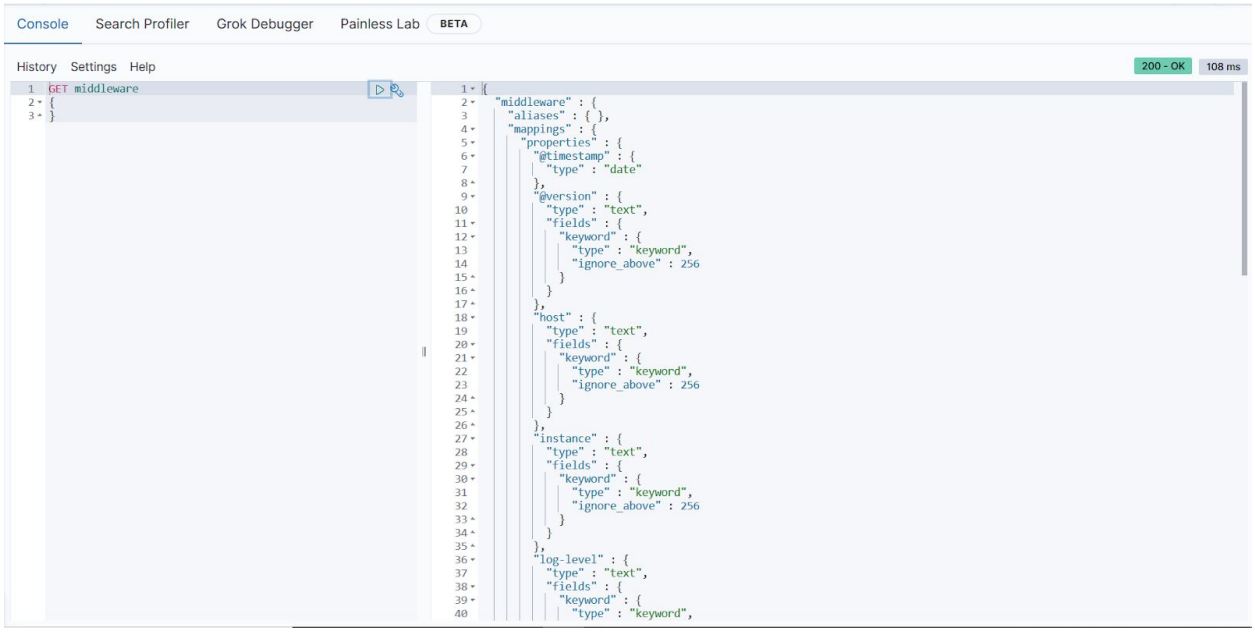


**Fig 5**: Kibana index

**Implementations issues faced and how we solved them**

1. Default syslog-ng version of Ubuntu repo did not have the Kafka destination feature.
   Solution: We had to build syslog-ng from source code and install it on the Ubuntu machines as the syslog-ng Github source code had the Kafka destination feature
2. Running Kafka and Zookeeper every time using a script was tedious.
   Solution: We made Kafka and Zookeeper as systemd services which always run in the machine as daemons

3. Syslog messages by default were not showing severity on Ubuntu.
   Solution: We modified the syslog-ng conf file to change the template of the message to a custom format as mentioned above.

## 6. CONCLUSION:

Our implementation provides real time notifications to the system administrators through Email and Slack. It also can be used to trigger support data collection scripts or recovery scripts on the client at the time of failure. In addition, the Kibana dashboard provides a real time visualization of these logs which can be used to monitor any system. So, our cluster of log servers effectively provides *Logging as a Service* to any client.

Some possible future extensions that we think of include addition of an encryption mechanism while sending the messages from the clients to the servers. In addition, we can also add more notifiers which can trigger a support data collection service or any other recovery service on the client cluster.

**Source code**: *https://github.com/akshayvb23/middleware/tree/master*

## 7. REFERENCES:


1. https://en.wikipedia.org/wiki/Syslog
2. https://www.rsyslog.com
3. https://www.syslog-ng.com
4. https://tools.ietf.org/html/rfc3164
5. https://kafka.apache.org/documentation
6. Jay Kreps, Neha Narkhede, Jun Rao. “Kafka: a Distributed Messaging System for Log Processing”
7. https://www.elastic.co
8. N. Kuduz, S. Salapura, "Building a Multitenant Data Hub System using Elastic Stack and Kafka for Uniform Data Representation" *2020 19th International Symposium INFOTEH-JAHORINA (INFOTEH)*, East Sarajevo, Bosnia and Herzegovina, 2020
9. Philippe Dobbelaere Kyumars Sheykh Esmaili. “Kafka vs RabbitMQ”
10. Vineet John, Xia Liu. “A Survey of Distributed Message Broker Queues”
11. Bajer, Marcin. “Building an IoT Data Hub with Elasticsearch, Logstash and Kibana”