IoT Practicum
WS 17/18

Sensors’ data storage & processing layer

Yesika Ramirez

Moawiah Assali

Atakan Yenel

Sergey Nasonov

Content

1. Layer description

The sensor's data storage & processing layer aims to receive streaming data from sensor's boards such as a Raspberry PI to process, format and storage it.

The configuration of the layer ensures the reception of the messages without duplication. The data is formatted and stored according to the customer's needs. In order to query and analyze the sensor's data, it can be accessed in a streaming fashion as well as in chunks.

2. Solution Architecture

2.1 Component Diagram

Figure 1 shows an overview of the solution architecture for the implementation of the Layer II. A detailed description of each component and instructions about deployment are described in next sections.

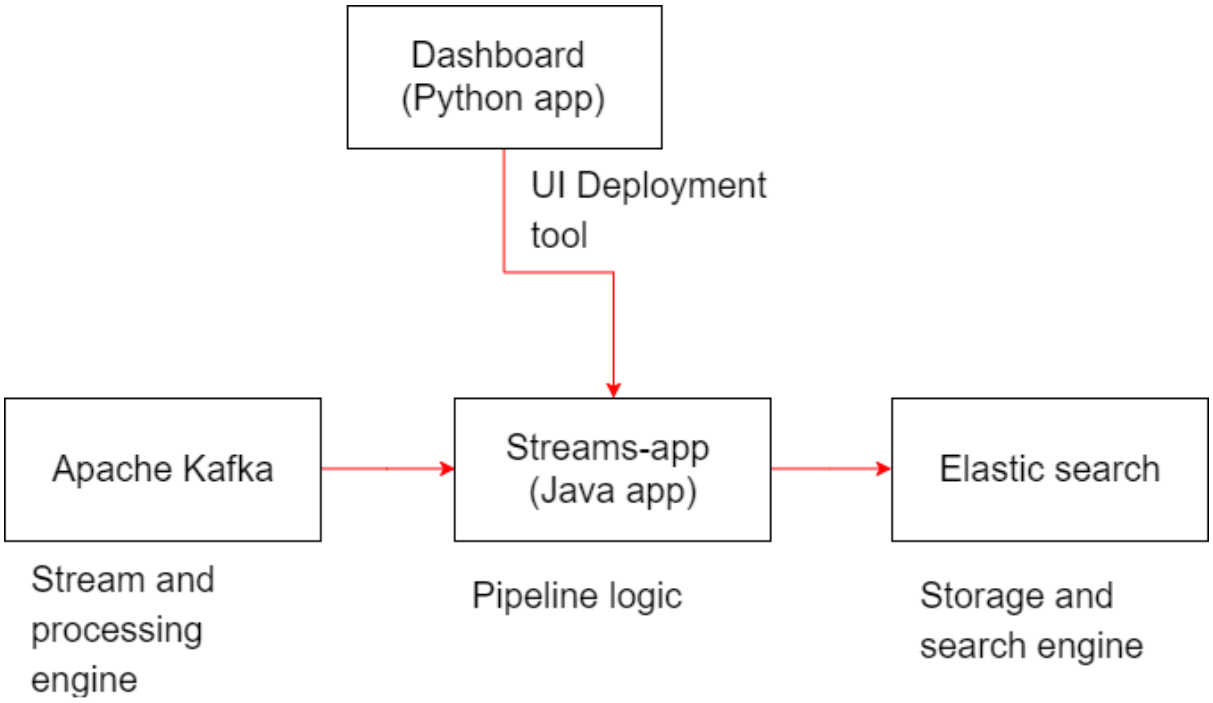


Figure 1.

2.2 Advantages and disadvantages

The designed architecture provides the following benefits:

- Offer a deployment strategy out of the box by the execution of one single script that takes care of open required ports, download and install the software and libraries required previous the installation our solution.

- User friendly dashboard, a custom application to deploy on demand new streaming applications.
- The streaming and processing engine support fault tolerance and scalability by the deployment of a cluster using docker containers which allows a faster recovery in case that one of the kafka nodes fails.
- Regarding the storage component, an elasticsearch cluster is configured to distribute the data storage workload.

Some drawbacks of the current solution.

- Currently there is support only for the version 2.3 of elasticsearch which is a constraint to use the features included in the latest version.
- The master node of elasticsearch is a single point failure for the system, so in case it fails the recovery time could affect the quality of the service.

3. Components

3.1. Apache Kafka

Is an open-source stream processing platform to handle real-time data feeds and uses a pub/sub message queue architecture.

Within the IoT platform, it is used to publish and subscribe to streams of sensor data and to build the streaming pipeline that reliably get data from the sensor's boards to the storage component.

The streaming of data is via channels or topics which are created on demand per customers. Additionally, Kafka has four core API's:

- Producer API: Allows an application to publish a stream of records to one or more Kafka topic. This API is used by the sensor's boards implemented in the Layer I.
- Consumer API: Allows an application to subscribe to one or more topics. This API is used by the Stream-app component of our solution to receive the data sent by the Layer I of the platform.
- Streams API: Allows the stream processing functionality which after the data has been received and processed is streamed to one or more output topics. This API is used to provide processed data in real time and not only in batches.
- Connector API: Used to connect Kafka to other applications. This API is used in our solution to connect the streaming solution to the storage and search engine solution Elasticsearch.

3.2. Elastic Search

Concepts & Description:

Our solution uses Elasticsearch as the database that will collect documents sent from layer 1. Elasticsearch attracted us to be used because:

- Elasticsearch is developed with Java, so its compatible with many platforms and systems.
- RESTful Database.
- Operates on all OSs that have Java VM installed.

Using ElasticSearch requires at first clarifying some main concepts:

- **Index**: collection of different type of documents and document properties.

- **Shards**: Shards represents the horizontal scaling ability of elasticsearch. A shard contains all the properties of a specific documents but with lower amount than an index. A shard is also an independent node which can be stored at any node.

- **Replicas**: these are copies of shards that assist in the following
  - Offering higher availability of data in case of failure.
  - Performance improvement when carrying out parallel search processes using them.

- **Clusters**: collection of nodes that work together and provide search options on the entire data. Each cluster contains a master which orchestrates the work among nodes (slaves). Slaves are the nodes where data is actually saved. We will explain how to configure these into more detail later.

3.3. Kibana

Kibana is a visualization tool that can be used with with ElasticSearch to view stored data as well as providing a high variety of charts and diagrams. Unfortunately, this tool works with ES version 6 and higher.

Installing and using Kibana is simple and straightforward:

1. You need to install an Elastic Search on your system → https://www.elastic.co/start
2. Install Kibana tool → https://www.elastic.co/start
3. Go to the Kibana installed folders and change the IP in the bin/kibana.yml configuration file to your machine.
4. Connect Kibana with ES following the instruction here → https://www.elastic.co/guide/en/kibana/current/connect-to-elasticsearch.html

3.4. Streaming application

The streaming application is a Java project using Maven as project management. While developing a streaming application for Apache Kafka, there are only two development languages available for usage, either Java or Scala. The reason is , Kafka only provides libraries that can do streaming through those languages. Of course it is possible to develop in other languages using a Kafka Consumer to get messages from Kafka, but if that happens , the full power of Kafka cannot be used and the application is no more streaming application.

The application requires at least one Kafka and one ElasticSearch instance to connect to. Configuration of the application can be done through environment variables without updating code. The main flow of the application is in *KafkaNode.java*, where the Kafka Stream object is created and configured. The application flow uses an Interface for easier configuration and is highly modular.

The application also supports Docker deployment on top of a maven image.

The additional documentation can be found inside *docs* folder, it is created using javadoc and explains the individual function of all classes and variables inside code.

3.5. Dashboard

Dashboard is a Python application using Flask as web framework. The objective of the dashboard is relatively easy , it is just to monitor docker containers and to create new containers using given configurations.

Dashboard uses Vue.js for it's frontend. The whole system is written as a web service so in practice in doesn't require a frontend to operate and it's only there for convenience. Dashboard requires access to docker daemon so make sure that the ports opened.

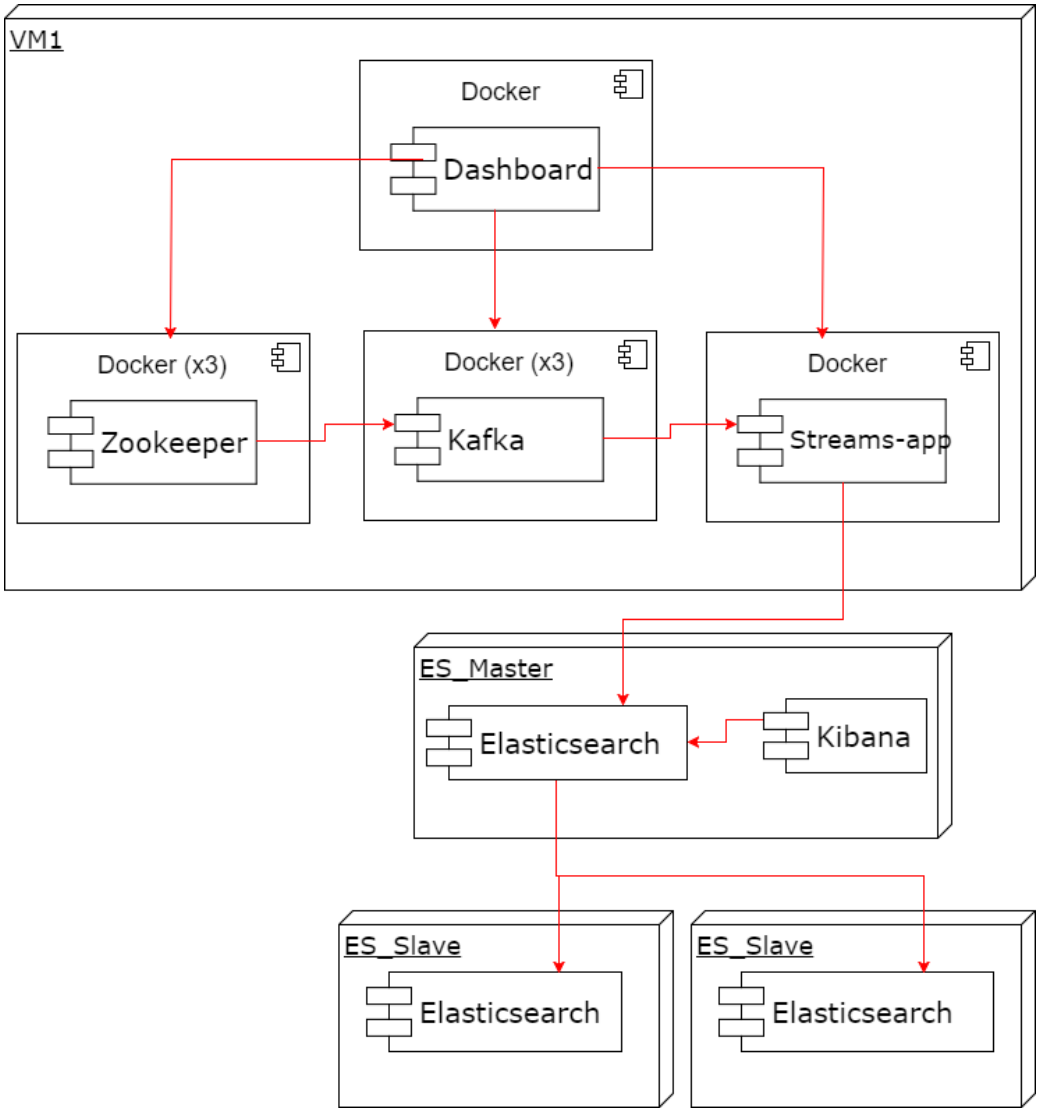
Dashboard has only one resource, docker containers, and it manages them in REST format. You can get , create , delete containers. There is no update container option as this is not possible with Docker.

There is also a scheduler API , which deploys multiple containers for a given time. It is still Work In Progress.

4. Deployment

The designed architecture allows deployment out of the box, as it is based on the docker images.

4.1 Deployment Diagram



4.2 Streaming-Processing deployment (Zookeeper, Kafka, Dashboard, Streams-app)

The whole Infrastructure can be deployed using a single *docker-compose.yml.* The IPs in the docker-compose file can be edited using different options, e.g env. variables, shell scripts. There are also comments on docker-compose file to show where to look for certain docker images.

All images in docker-compose use a personal docker hub ID and that may change in the future. It is advised to fork all images to a different docker hub profile at the start of the project. It takes around 4 minutes to download the images and start containers in LRZ cloud.

Make sure to open ports so that different containers can connect to each other. They are using external connection so, ports should be opened.

Individual deployment is also possible for every software part either by deploying their respective *DockerFile* or by using traditional ways and installing them to OS directly.

4.3 Storage deployment (Elasticsearch)

This section will describe how to configure the main points for an ES cluster to run on your system. It will also suggest some solutions to the main troubleshooting issues that you may face supporting the cluster.

Configuration:

When building a cluster using Elasticsearch, you need to configure at least one master node, and one data node (our cluster consists of 1 master and 2 slave nodes). There are many configuration Items that can be done but we will mention the main ones that will make you cluster up and working ready for the stream data to be pushed.
we will start by configuring a master node:

Configuring a node to operate as an ES master starts by installing Java, then installing a suitable version of ES → https://www.elastic.co/downloads/past-releases
After that, we need to change the main ES configuration file (.../config/elasticsearch.yml). As this file sometimes differ between versions, we should mention here that we used ES 2.3.2 for our cluster. Moreover, this file contains some basic sectioning and values, any needed value that is not there can be manually added.

The following table shows the main items that we need to change for our master to start working.

Field	***Description***	***Notes***
cluster.name: elasticsearch	as the name indicates, it specify the name of the cluster that we need to create. Default value is elasticsearch.	Be careful here that this value should be synchronized with the value mentioned in the code connecting to this cluster.
node.name: iotLab_master	Node Name is specified here, you will see this name in the logs as you start your cluster.	
node.master: true	This value stated the role of the node in the cluster. here it is a master.	

node.data: false	This value states that this node is not a data node, so its not saving documents.	
index.number_of_shards: 5	Here we can specify how many shards do we need our data to reside in.	This value may not work directly like this in other ES versions. There is the option to specify the number of shards using commands. There is no exact rule for the number of shards that one must have → we recommend 2 shards per node. Watch out that a shard introduces another set of performance considerations. A shard should be initiated and managed by ES → so a large number is NOT recommended.
index.number_of_replicas: 3	This value specifies the number of replicas that we need to have in our cluster.	If one node have a shard, the other would have a replica to overcome failures. the default value is "1", which means that each shard has one replica.
path.data: …./elasticsearch-2.3.2/data/elasticsearch	This values specifies the location of the data.	This value is checked when the cluster is initiated.
path.logs: .../elasticsearch-2.3.2/logs	This value specifies the logs of the ES node.	
network.host: 141.40.254.44	This value binds the ES to a specific IP which we can access for information.	
discovery.zen.ping.unicast.hosts: ["141.40.254.44","141.40.254.38","141.40.254.46"]	This value takes the value of other nodes in the cluster - and sometimes the nodes that are eligible to be a master.	This list of IPs is used for pinging other nodes and election process.

Meanwhile, configuring a slave is similar by using the following values:

Field	***Description***	***Notes***
cluster.name: elasticsearch	as the name indicates, it specify the name of the cluster that we need to create. Default value is elasticsearch.	Be careful here that this value should be synchronized with the value mentioned in the code connecting to this cluster.
node.name: iotLab_Slave1	Node Name is specified here, you will see this name in the logs as you start your cluster.	
node.master: false	This value stated the role of the node in the cluster. here it is a master.	This value here can be used to instruct that this node can be elected as a master if set to true.
node.data: true	This value states that this node is not a data node, so its not saving documents.	
path.data: …./elasticsearch-2.3.2/data/elasticsearch	This values specifies the location of the data.	This value is checked when the cluster is initiated.
path.logs: .../elasticsearch-2.3.2/logs	This value specifies the logs of the ES node.	
network.host: 141.40.254.44	This value binds the ES to a specific IP which we can access for information.	
discovery.zen.ping.unicast.hosts: ["141.40.254.44","141.40.254.38","141.40.254.46"]	This value takes the value of other nodes in the cluster - and sometimes the nodes that are eligible to be a master.	This list of IPs is used for pinging other nodes and election process.

Troubleshooting Issues:

Issue	***Description***	***Solution***
No Ping	Master and Slaves are	Don’t forget to open ES ports. Default

	not able to ping each other, and you get that pinging master is not working.	ones are 9200/9300
No Ping	Same as the first issue	Dont forget to shut the firewall down Example: sudo ufw disable
Many files are Open	If you run the cluster with default values → Many people accessing the host may end up with this error and the cluster fails.	change the default limit by the following for example: ulimit -Hn 65536
Virtual Memory Warning	Cluster warns that the virtual memory is low and sometimes does not start at all	Increase the VM for ES: Example: sysctl -w vm.max_map_count=262144

Starting the ES cluster is simple → navigate to the bin folder and run
./elasticsearch

4.4 Basic Visualization

This section will describe a quick visualization stack that you can use to visualize the nodes after the deployment of layer 2. A detailed explanation of visualizing the whole platform is found in layer 4 description.

The visualization stack here consists mainly of three components:

- Graphite
- Grafana
- CollectD

You can start by configuring graphite on your node so you can start collecting some basic information about your CPU usage and memory. Configuring graphite has some steps to be done which are mentioned here:
https://www.digitalocean.com/community/tutorials/how-to-install-and-use-graphite-on-an-ubuntu-14-04-server

After that, you can do something OPTIONAL which is to build CollectD upon Graphite. The idea here is that Graphite power is limited and it does not have a broad access to offer a wide range of useful data for your node → CollectD empowers graphite with more power.

CollectD can be configured after Graphite as follows:
https://www.digitalocean.com/community/tutorials/how-to-configure-collectd-to-gather-system-metrics-for-graphite-on-ubuntu-14-04

As a last step, and in order to obtain good looking curves and visualizations, you can configure Grafana upon Graphite. Grafana is a powerful visualization tool that can be used in a wide range of combinations. Configuring this with graphite is as follows:
http://docs.grafana.org/features/datasources/graphite/

5. Challenges

- The resources of the Virtual Machines were very limited. In particular, it was difficult to allocate a high amount of memory (over 8GB) and CPUs(1+). In fact, we tried to increase the number of CPUs to 4, and while the number of physical CPU increased, however, the number of virtual CPUs stayed the same. This fact created a difficulty for the monitoring solutions team as the software they were using could detect only one CPU while 4 were present. The same value was reported using bash commands inside of the OS.
- We found that some versions of Kafka, together with the connectors to Flink were incompatible. Additionally, not all versions of Flink, together with the connectors worked with ElasticSearch. The reasons were mainly were bugs in connectors, changed data flow logic between versions and general ambiguity with a lack of proper guidance from the connector providers as well as Kafka, Flink and ElasticSearch development community for earlier versions of their software (v2). For this reason we decided to discontinue our usage of Flink as it would decrease the complexity of the solution as well as provide more stability.
- LRZ cloud provides every VM with an external IP upon request. It is however, a rule, when a VM gets undeployed or shut down, it loses the IP address. This fact created a difficulty because ElasticSearch nodes need to know the ip addresses of other nodes and in case one of the node crashes and a new one is deployed, the old nodes cannot

find the new node as it got a new ip address. The issue was solved with the custom ip-tagger that was described earlier.

6. Lessons learnt

In order to decide about the stack of technologies to design and implement the solution, a set of proof of concepts were needed. A specific case was the selection of the processing component where Kafka Streams and Flink were evaluated for this purpose. On the one hand Kafka Streams offers the possibility of processing the streaming data and publish it at the same time to a defined output channel. On the other hand, Flink has been longer in the market which gives a sign of robustness. However in order to ensure high availability of the streaming processing component a Flink cluster should be setted up which increase the complexity of the general solution.

The result of these proof of concepts concluded that Kafka Streams would be a better option since the integration of streaming and processing would be already done and the system could still ensure high availability implementing a Kafka cluster without increasing the overall complexity of the system.

A summary of the lessons learn include:

- In addition to the literature review and revision of technologies trending in the market, we highlight the importance of proof of concepts before take final design decisions.
- The design and implementation of a solution has to take into account the limitation of resources.(Eg. max capacity of storage)
- In a project where different teams are responsible for different functionalities it is important to establish efficient communication channels, establish clear responsibilities and agreements to avoid misunderstandings.

7. References

- https://www.tutorialspoint.com/elasticsearch/elasticsearch_basic_concepts.htm
- https://db-engines.com/en/system/Elasticsearch%3BMongoDB
- https://www.elastic.co/guide/index.html