Project Ember

CINI 2017 Smart Cities

Apache Flink based implementation for a smarter city illumination control system.

Alessio Moretti
Federico Vagnoni

version 1.0.0 - RELEASE CANDIDATE

Introduction | a smarter city

Problem statement*: provide an energetically optimal and efficient solution to power a grid of smart lamps, capable to adapt according to weather, light and traffic conditions. The system must be able to reduce the consumptions, monitor the lamps status and provide a safe illumination.

Efficiency to reduce lamps consumption.

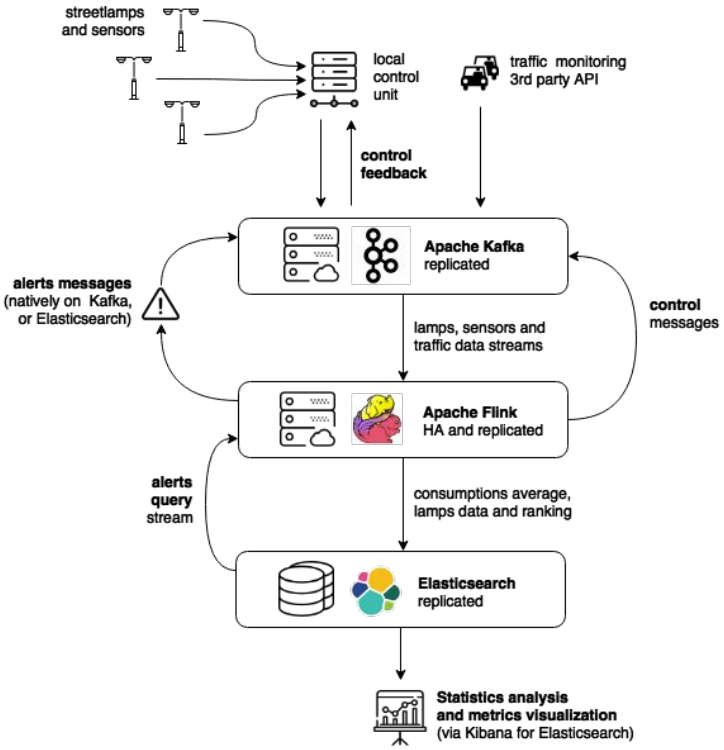
Awareness to weather conditions and light levels.

Safety to not compromise vehicles and pedestrians traffic.

Assumptions: we will assume sensors on lamps sending every 10 s the lamp status and lumen level via co-located sensors, third party APIs to monitor traffic levels.
All the records are sent as a JSON-formatted string.

* concept was originally created for the CINI 2017 challenge about smart cities illumination

Architecture | standing on the shoulders of giants



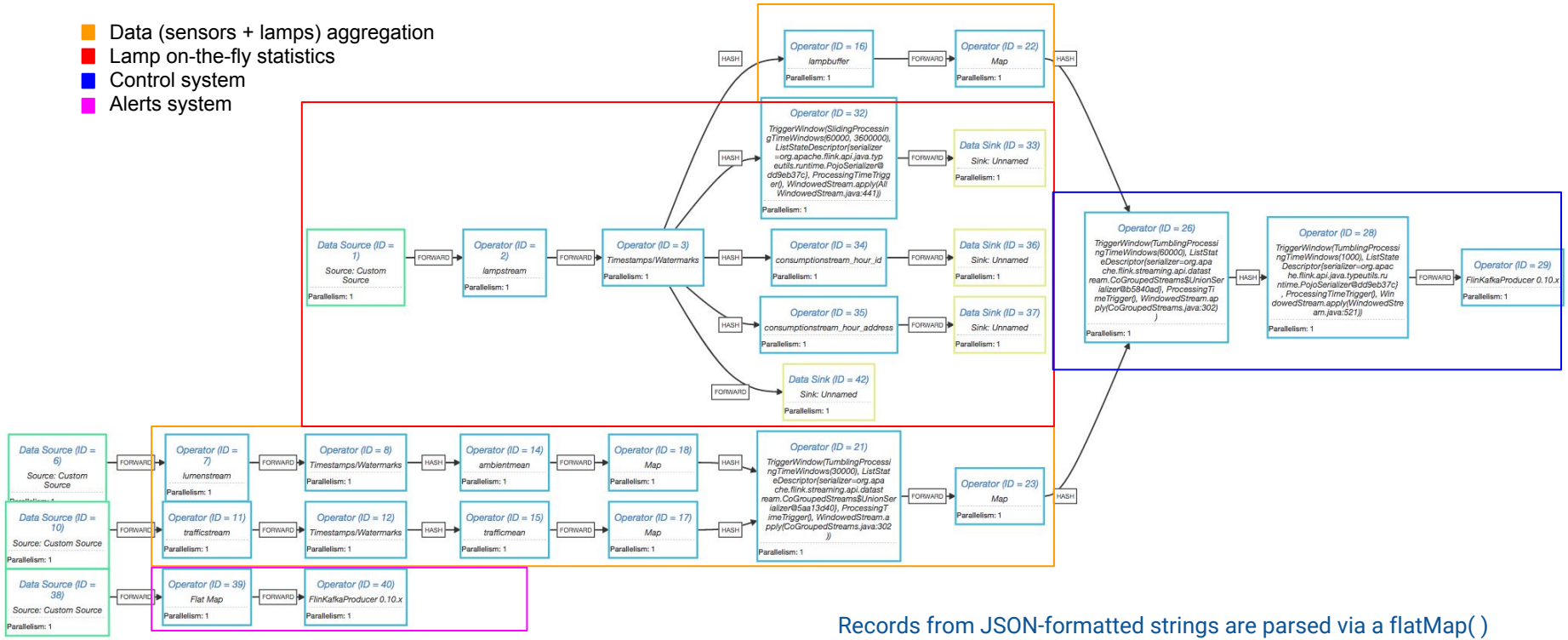
A general overview of the architecture: **Apache Kafka** to handle data from sensors and to get controller output delivered, **Apache Flink** for near real-time data processing and **Elasticsearch** to store and visualize data statistics and metric.

It is interesting to observe we are working on the edge of the network, following the **fog computing paradigm**.

** Note that it is irrelevant to define streetlamps by address or by parking cells to operate our control system.

The Grid | (near) real time data processing

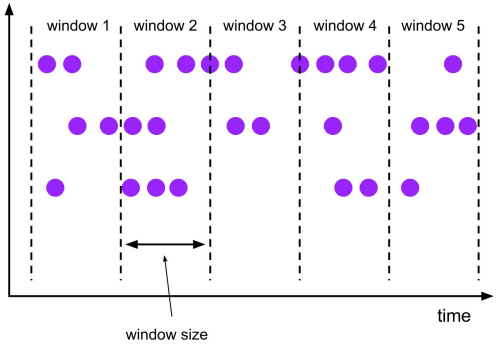
- Data (sensors + lamps) aggregation
- Lamp on-the-fly statistics
- Control system
- Alerts system



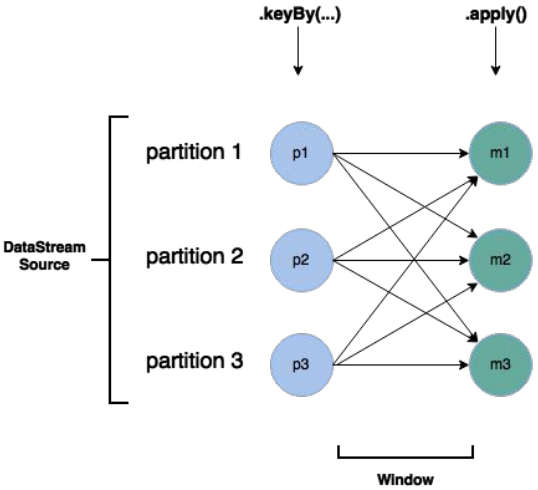
Records from JSON-formatted strings are parsed via a flatMap( ) function to be processed as complex records - accessing their attributes to perform data aggregation and produce punctual control output.

The Grid | data aggregation

In order to create useful groups of data we used aggregations by windows.
A window can be of different type: **EventTime/ProcessingTime triggered, Tumbling/Sliding**.

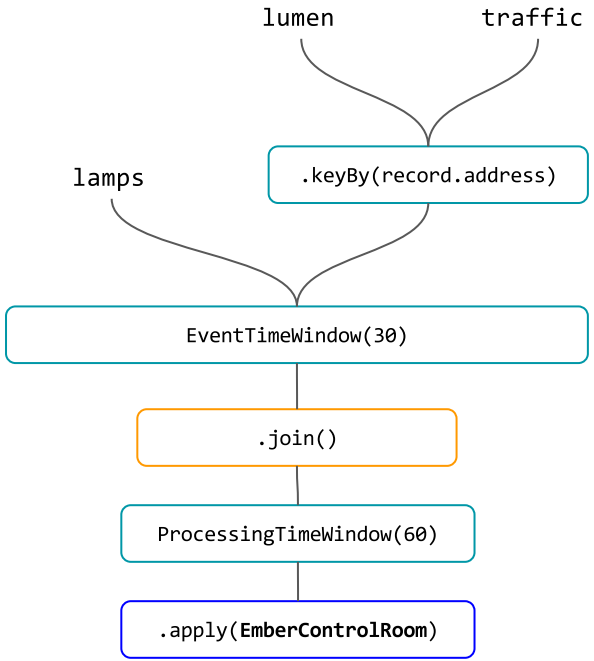


An **element that fall in a window shares with the others the key by which they will be aggregated**
and the timing property
of the defined window.



To ensure performances, different aggregations were made.
In particular, a **1m 30s total buffering is achieved using a combination of ProcessingTimeWindows in cascade from EventTimeWindows to collect and produce a control output from the records streams**.

The Grid | control system



The control system is powered by a parallelized set of operators whose computer the optimal control value after a cascade of windows to **aggregate by address the sensors DataStreams**. Then a **output is computed for each lamp** using the following formula, returning the **optimal light level to be set by the lamp controller**:

$$I = \frac{L_l * C_u}{A_l} \rightarrow L_l = I * A_l$$

provided by the National Optical Astronomy Observatory

We use windows to buffer records before control operator, respectively to aggregate different records streams and to buffer in order to optimize computation.

The Grid | lamps (on-the-fly) statistics

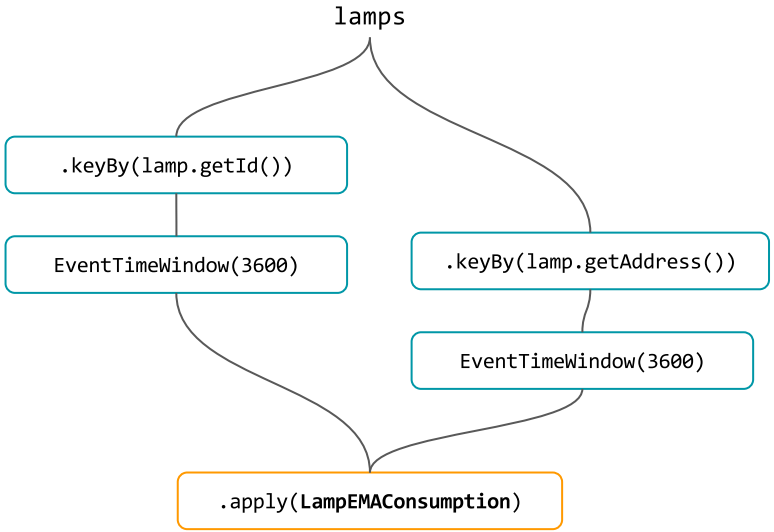
To compute the consumption average, we will split the data from the lamps stream into two keyed streams - **by address and by id**.

$$S_t = \alpha * Y_t + (1 - \alpha) * S_{t-1}$$

alpha = 0.8 - the latest data are the most significative

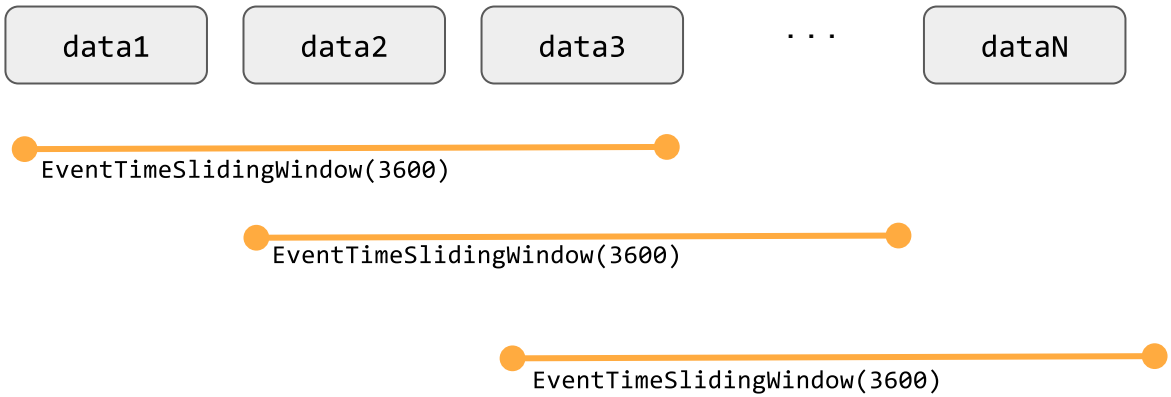
We will use the **moving average to compute, on a 1h long window, the consumption mean value**.

The resulting streams are stored on Elasticsearch for future analysis (as for example last day, week average) across the entire dataset (by single district, by city, by address and so on…)



The Grid | lamps lifetime ranking

To compute the real-time ranking of the lamps whose lifetime is near (or over) the expiration limit configured by the operators, we chose to use the **EventTimeSlidingWindow** iterating on a per-hour step over the lamps arriving into the system.



The Grid | alerts system

Query to retrieve the lamps which are showing any anomalies*

Stream to parse query results and produce correct output messages according the anomaly detected

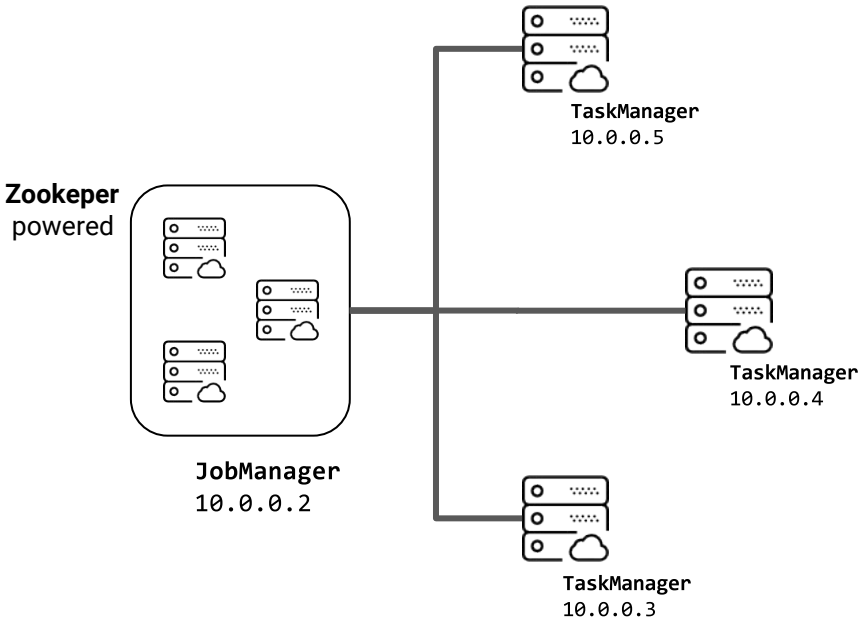
Alerts produced on Kafka (the system can be configured to produce alerts on Elasticsearch)

* anomalies are states of the lamp where control level is out of bounds, expected lifetime is expired or an electric failure is happening (lamp does not respond) - please note that the lamps are continuously updated into Elasticsearch by Flink sinks when new records are retrieved from Kafka MOM.

“Everything fails, all the time”

Werner Vogels, CTO @ Amazon

The Grid | deploy at scale



Apache Flink can be deployed in high-availability mode, adding at runtime TaskManagers to the main JobManager. The system utilizes each core as a new task slot to run the topology.

Note:

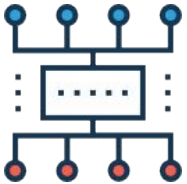
- **Apache Kafka** can be deployed on a **cluster of nodes running the same instance** (Zookeper powered)
- **Elasticsearch**, instead, allows both the **replica set** and the **sharding**.

AWS EMR friendly!

The Grid | our metrics

406

Control throughput
(records per second)



180

Kafka consumer operator
(records per second)

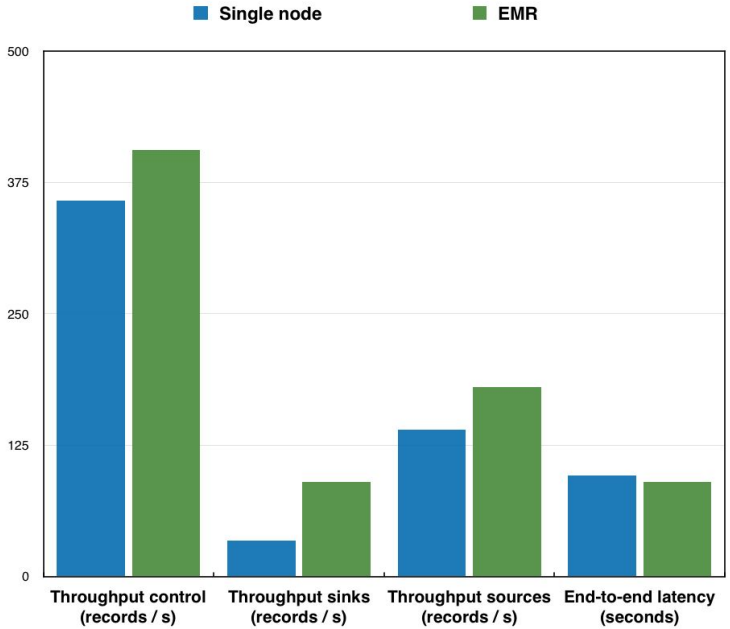
60

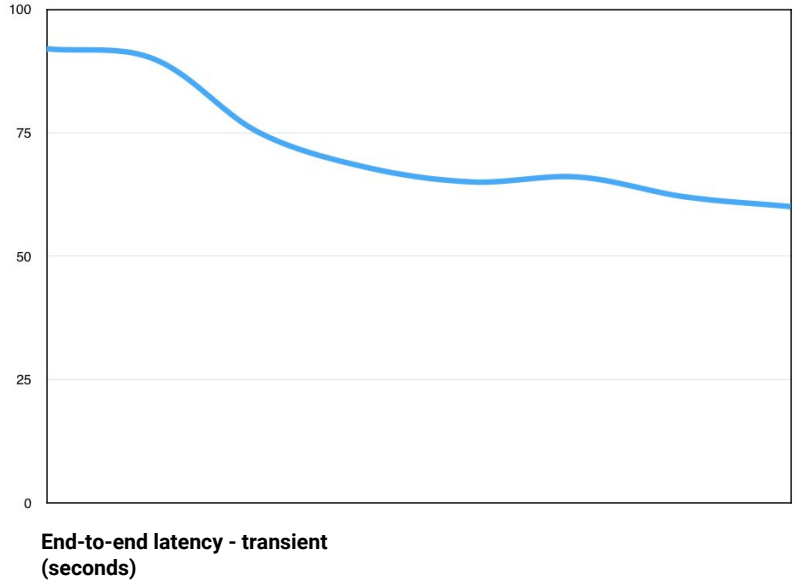
End-to-end latency
(seconds)

Testing environment: Apache Kafka and Elasticsearch replicated on 3 x m4.large EC2 instances, Apache Flink on a five-node m4.large in an AWS EMR cluster.

2000 lamps for 100 different streets.

The Grid | our metrics (pt. 2)





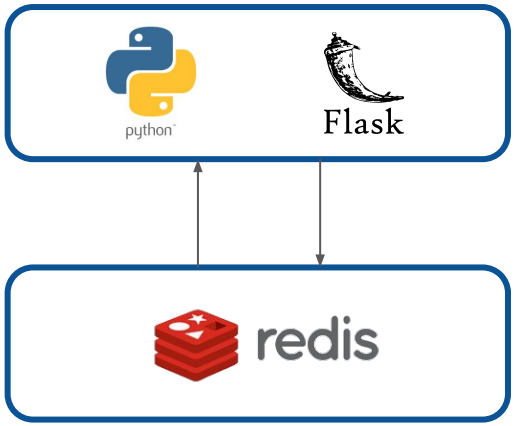
During the test it appeared clearly that the latency in operators forwarding was insignificant (under 0.001 seconds), our bottleneck is the control operator output's sinks - this is why a multilevel windows buffering is necessary to optimize the computation.

“Simplicity favors regularity”

MIPS projectual principle

Edge of the network | control units and messaging

The local control unit is a portable, lightweight and API oriented component for reading data from street lamps, as well as forwarding control inputs. It provides also API endpoints to manage the entire control units network. At this level we can add modules and extensions to make the system behave according to CINI challenge.



Python + Flask (+ Kafka)

API endpoints, Kafka topics consumers and producers driver, street lamps management and control routing, first security level for operations on lamps.

Redis

Persistence level to filter streetlamps inputs / outputs and manage CRUD operations.

Docker friendly!

Edge of the network | security and interoperability

When a lamp is placed, it has to be set up. The operator, in fact, is the only person that has the access to the **control unit register,** powered by **AWS Lambda**, so he can interface directly to the control units API endpoints, querying their IPs on a per-area or per-city basis.

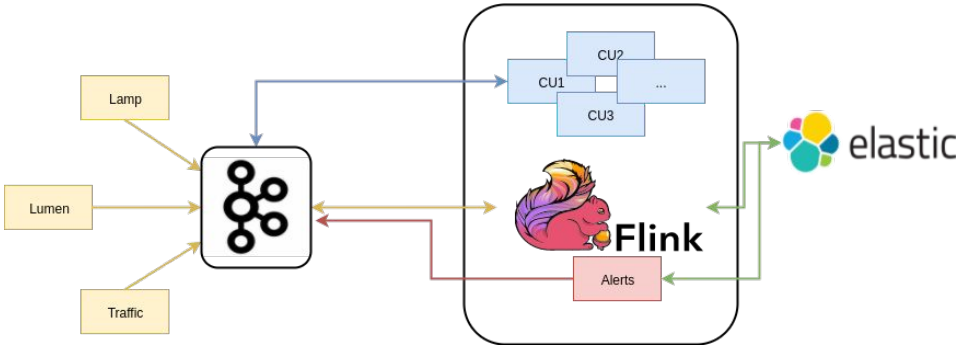
We can provide API Keys to operators for lamps registration i**mproving security** and **safety** for the whole system.

The control unit **guarantees** for the new registered lamp reinforcing the whole structure.

Edge of the network | a message oriented approach

Lamp, Control, Lumen, Traffic and Alerts (configurable) are the topics managed by Kafka so the entire system can work.

How does Flink contact a lamp
that needs to be updated?
It should know each lamp ip addresses
in the network, or use one topic
for lamp!



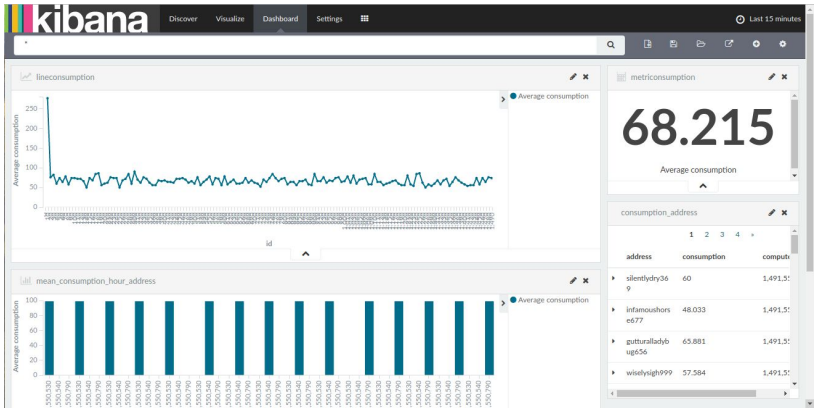
The control unit act as an indirection level

Docker friendly!

Store and Visualize | elasticsearch loves kibana

The persistency level is managed by **Elasticsearch**. It lets us combine many types of searches (structured, unstructured, even geo and metric) taking care of the dirty work for us.

Kibana comes as the perfect choice, as being developed as part of the Elastic Stack:
we can visualize and explore the data stored in Elasticsearch using the customizable **dashboard** provided.



Store and Visualize | monitoring system

Elasticsearch is our source for alert events thanks to the full-text queries we can submit, all powered by the Apache Lucene machine learning core of ES.

Flink processes records sent by the sensor network and sends lamp records, means and ranks to Elasticsearch.

We built a custom SourceFunction that performs a query which control the updated lamps records and check if:

- a lamp is not working (e.g.: the record timestamp refers to a moment too far in time)
- a lamp is not responding (e.g: the lamp should be powered on but it’s not)
- a lamp should be replaced (e.g: the last replacement timestamp refers to a no more tollerable amount of time)

The records gathered are then routed to Kafka or to an alerting system of choice.

Fork us on GitHub

```
// explore the project
Repository repo = “https://github.com/projectember”;
Git projectEmber = new Git(repo);

// drop us a line
String alessioMail  = “alessio.moretti@alumni.uniroma2.eu”;
String federicoMail = “federico.vagnoni@alumni.uniroma2.eu”;

// thx for your attention
projectEmber.addSink(new ThankYouFunction());
```