Overview of the infrastructure and usage report

Fink Collaboration meeting
16/07/2025

Fink & the technics

Fink is way more than a bunch of codes. But we should not forget what runs behind.

3 main ingredients:

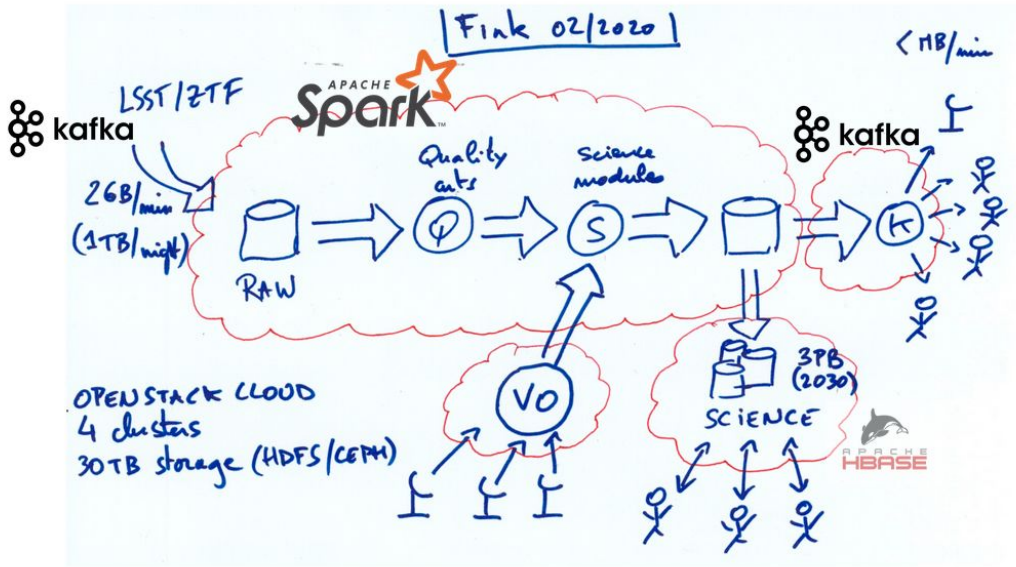
- Computing infrastructure (compute, volumes, network, …)
- Core software (manipulating alerts, filling database, providing live connection & web services, …): e.g. *fink-broker, fink-science-portal, fink-object-api, …*
- User-defined codes (providing the *scientific surplus*): e.g. *fink-science, fink-filters*

3 main services to interact with the data:

- Science Portal (+ API): *daily work, quick search, visualisation*
- Data Transfer: *Bulk download, complex analyses, …*
- Livestream: *follow-up analyses*

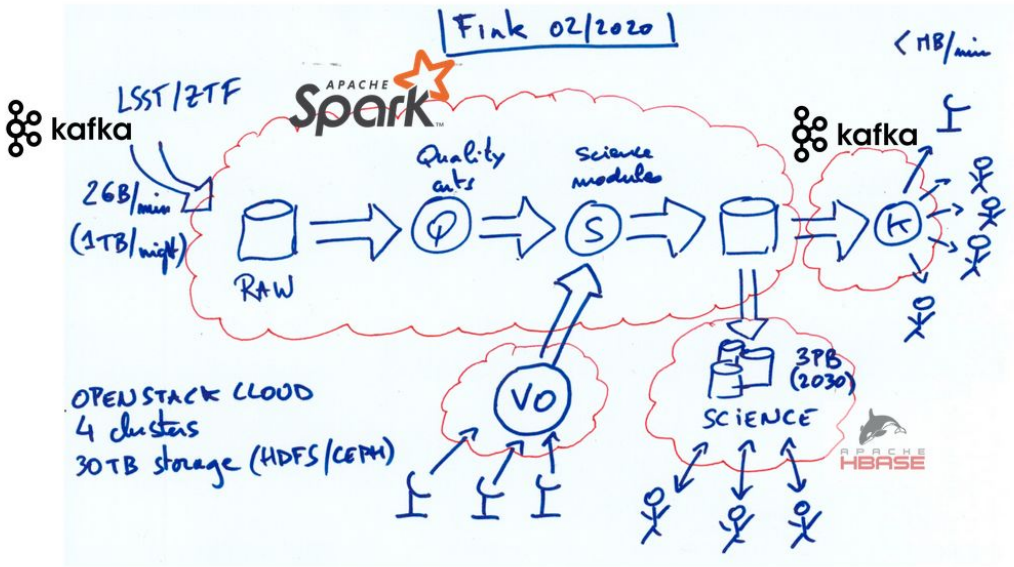
Fink in the early days…

Fink alert processing



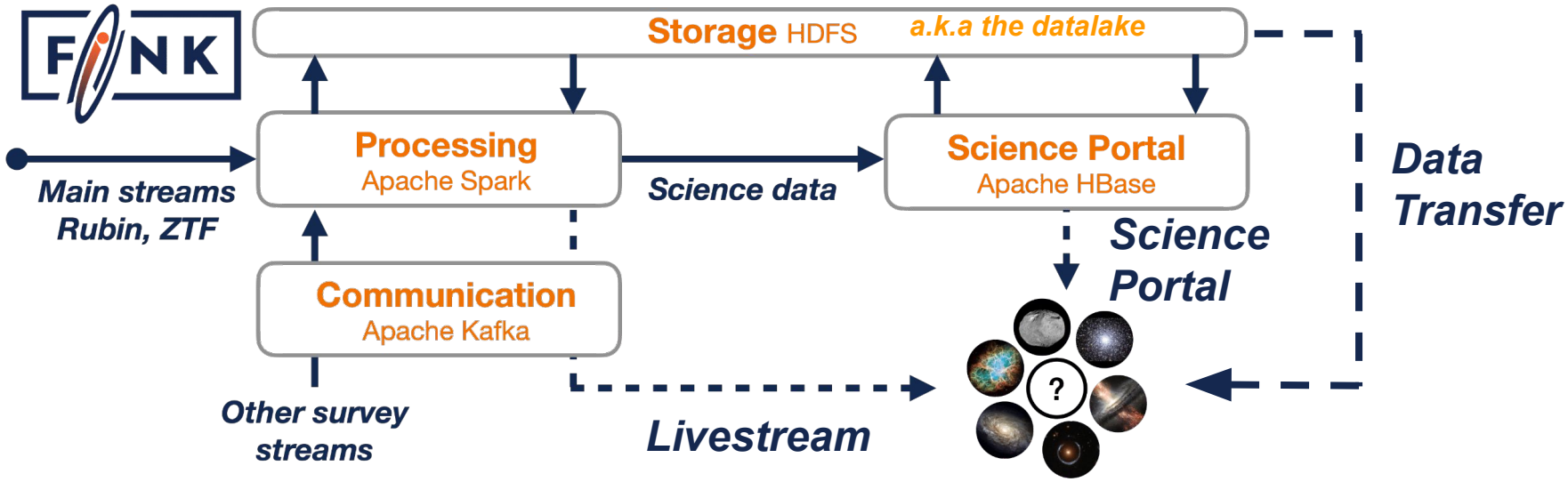
Fink today

Fink alert processing



(almost…)

High level architecture



Datalake vs database

Why two different data sources in Fink? Because there are many usage for the data (many *different questions to be asked to the data*), imposing different constraints that can be hardly solved by one single system.

The **Fink Datalake** contains processed **alerts**, partitioned by **time**. It is distributed over many machines.

- Pro: Easy to load large chunks of data ordered by time (last month, last year, …). Easy to interface with a distributed computing framework for additional analyses. SQL syntax works under certain circumstances.
- Cons: Basic data structure is **alert**. If you want objects (i.e. a collection of alerts sharing the same ID), or something else than time-ordered, data is slow to access.

The **Fink Database** contains processed **objects**, indexed primarily by **ID**. It is distributed over machines.

- Pro: Easy to access full data for an **object**. Easy to connect with external services.
- Cons: difficult to get away the primary key (ID). Hard to debug (compared to a datalake).

And you will hear about Graphs tomorrow by Julius.

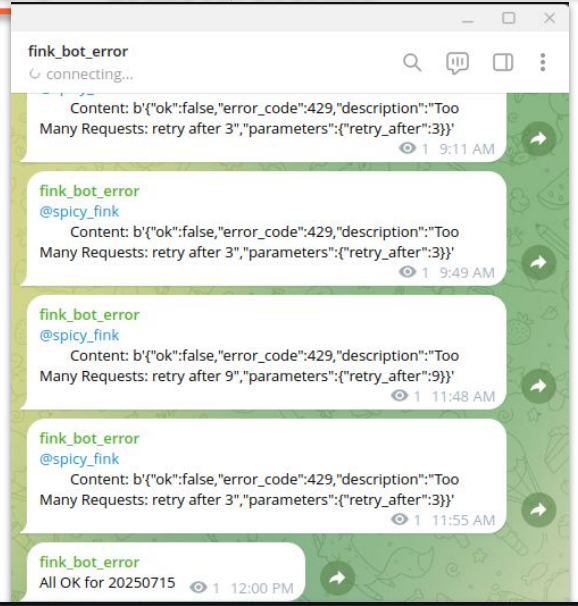
Real-time system

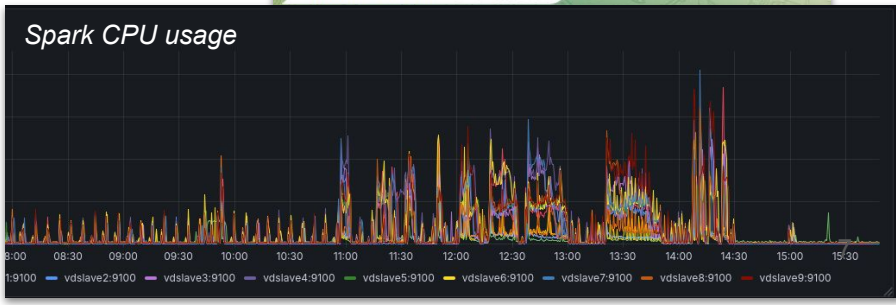
```
vdmaster1:~$ hdfs dfs -du -h
archive
12.4 T  archive/raw
8.1 T   archive/science
```

Fink processes *autonomously* data in **3 stages**:

- **Decoding & backing up the incoming stream**
  - 4 vCPUs, 8GB RAM
- **Processing the stream & backing up**
  - 8 vCPUS, 16GB RAM
  - Quality cuts + science modules
- **Filtering the stream**
  - 4 vCPUs, 8GB RAM
  - Fink filters, Slack/Telegram bots

Monitored via Grafana & Telegram bots





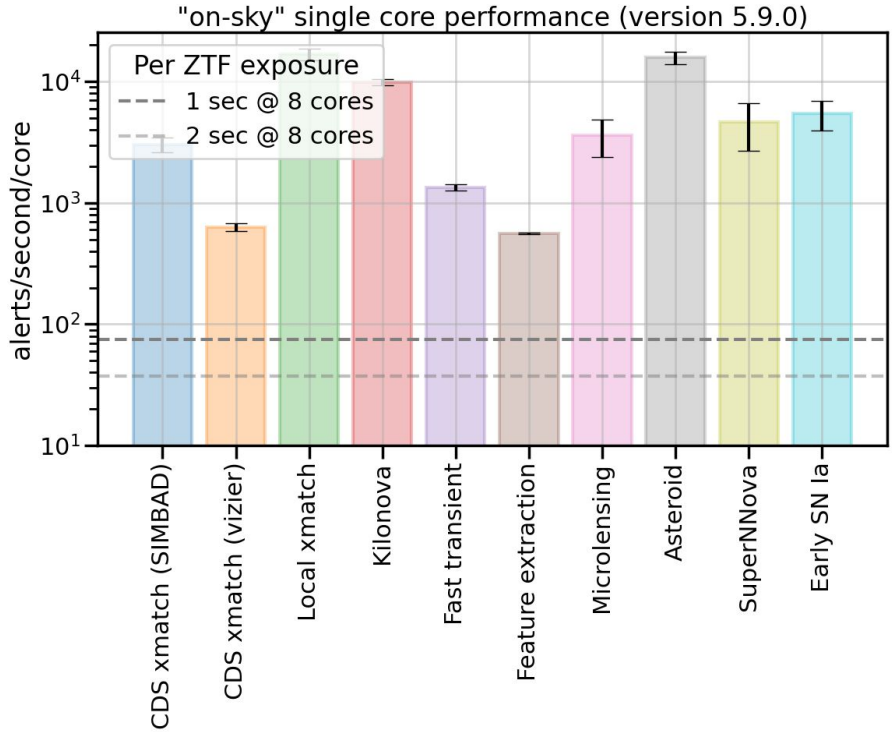
Science modules

We currently host 12+ science modules:

- ML-based
- Simple feature-based (decision tree, fit, etc.)
- Crossmatch with external catalogs

They are all tested, profiled, curated, modified as dependencies change (67 Python deps!).

Scaling is horizontal → more workers, bigger throughput.



On the ML side

We know how to deploy ML models at scale. But we don’t do it in a clean way…

Here is the current story of `Toto`, doing ML in Fink:

- `Toto` asks Julien to deploy a model, by opening a PR on fink-science
- Julien is busy. After days (weeks?), the model finally is deployed.
- Then `toto` wants to change the model. `Toto` needs to ask Julien again. Delay again.
- Then `toto` wants to try different models in parallel to compare outputs, and perform active learning loops. Julien is unfortunately on holidays. `Toto` gives up…

This could be solved simply by using the right tools. Here is an alternative world:

- `Toto` trains one or several models, uploads them to a platform without the need of Julien. Fink collects new models ready for production*, without Julien. `Toto` & Fink are happy, Julien has disappeared from the loop!

Stay tuned!

**to be defined*

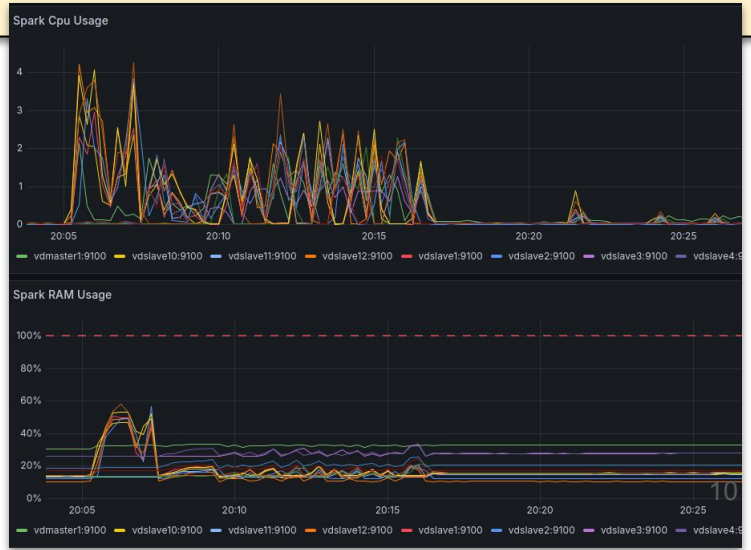
After the night operations

Several (autonomous) operations performed at 8pm Paris time:

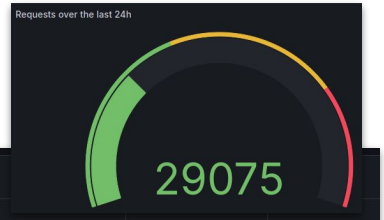
- Data consolidation (80 vCPUs)
- Once-in-night filters (8 vCPUs)
- Database aggregation (9 vCPUs)

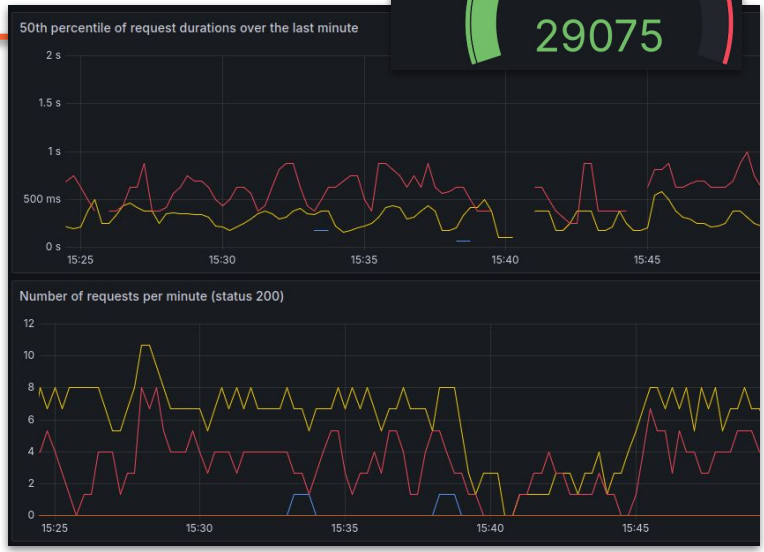
Problems for months about database push for large nights… Largely solved by decoupling cutouts from the rest (and by my capability to better understand how HBase works…).

```
vdmaster1:~$ fink_db -h

fink_db is a wrapper around fink to control the database operations.

Usage: fink_db [OPTIONS]

...

Examples:
fink_db -s ztf --merge                # merge files
fink_db -s ztf --main_table           # Push data to HBase
fink_db -s ztf --index_tables         # Push data to HBase index tables
fink_db -s ztf --clean_night          # Clean temp files for the next
night
```



Science Portal / API





Key points:

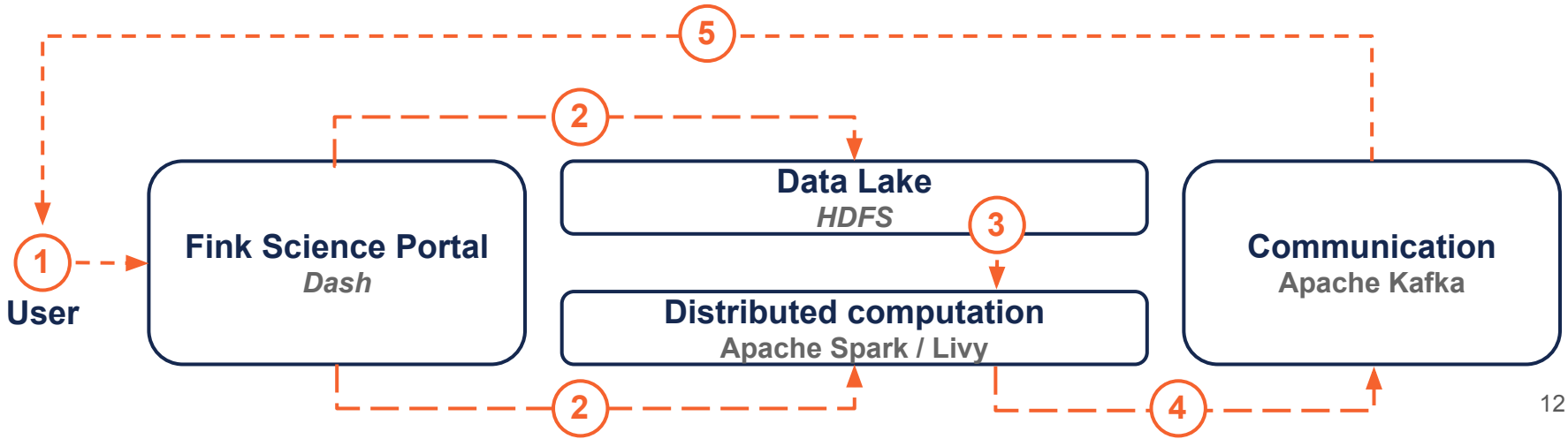
- https://fink-portal.org (doc)
- https://api.fink-portal.org (doc)
- Quick search based on **name, position, class**
- Data source is **the Fink database**
- **100+ users** every day (not counting robots!)
- **~30k requests/day** (½ `/api/v1/objects`)
- **Anonymous**, no limits, but usage heavily monitored!

Data Transfer

"Not formally our job, but we fill a crucial gap in the alert community"

Key points:

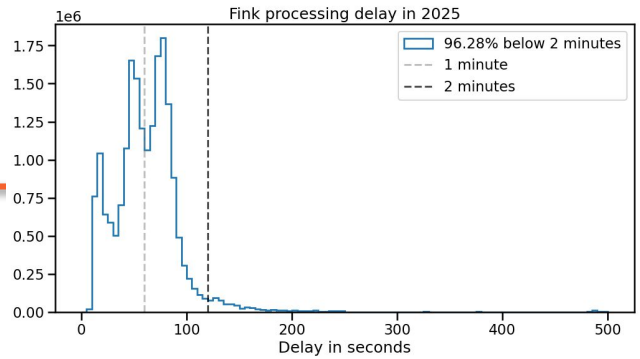
- https://fink-portal.org/download (documentation)
- Access to 200M+ alerts from ZTF from the **Fink Data Lake**
- Each job is allocated **8 vCPUs** on the Spark cluster.
- Customisation: dates, filters (**updated!**), content (**new!**)

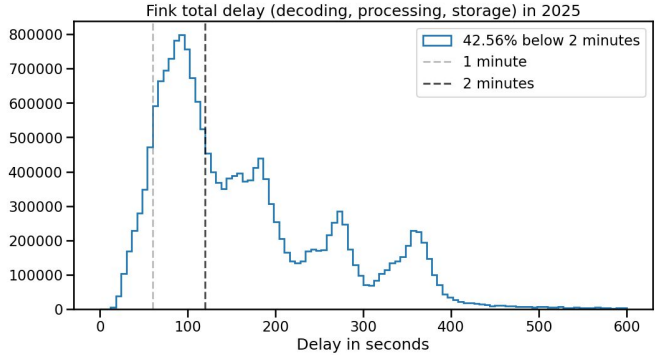


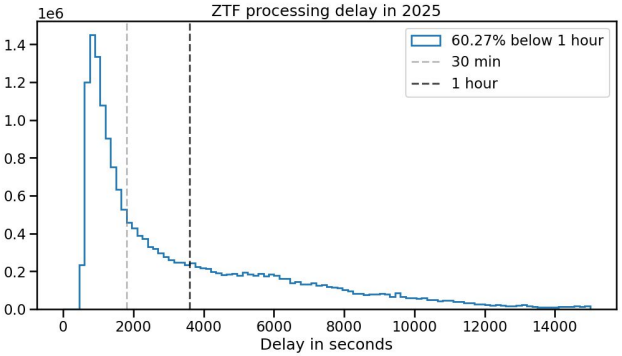
Livestream

Key points:

- Facilitate follow-up observations or heavy analyses
- Rely on fink-filters (documentation)
- Output: Kafka stream, Slack/TG channels
- It gives all flexibility to the users about the definition of cuts and filters, and it should be de-facto the primary choice for real-time analyses. **But it is not so much used… Why?**





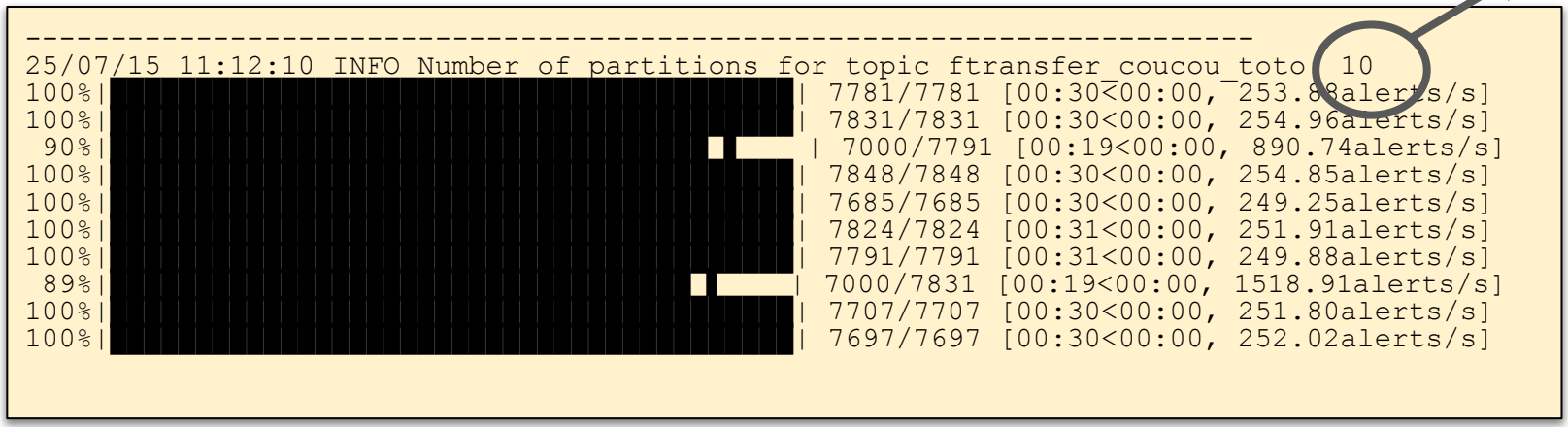


Fink client

`fink-client` is a package to manipulate alerts issued from Fink programmatically

- It is a convenient wrapper around Kafka low-level functionalities.
- It is used to retrieve data from the **Data Transfer** (`fink_datatransfer`) & the **Livestream** (`fink_consumer`) services
- Multithreaded to increase the throughput, but sensitive to timeouts.

Important!



After 6 years, Fink is still alive

Main ingredients to get an evolving & sustainable system (and happy Julien)

- Code quality! 5% code, 95% quality checks
  - Incl. versioning, documentation, tests, continuous integration
  - Open source code, documented (users, developers)
- Monitoring is not an option
  - Performances (profiling) & logs. Set up easy-to-use interface.
- Continuous deployment
  - Release fast, take into account feedback quickly
  - Digest release notes available for all components (example)
- Develop easy-to-use sandbox
  - Make distributed computing easy (cf Fabrice's talk)

And what is next for LSST?

Same infrastructure, but at CC-IN2P3 cloud (the beauty of clouds!), and bigger.

Operation Rehearsals 4 & 5 already comfort us in our capability to **meet real-time requirements**.

Elasticc was a **first test for ML classification.**

Database & web services are still TBD.