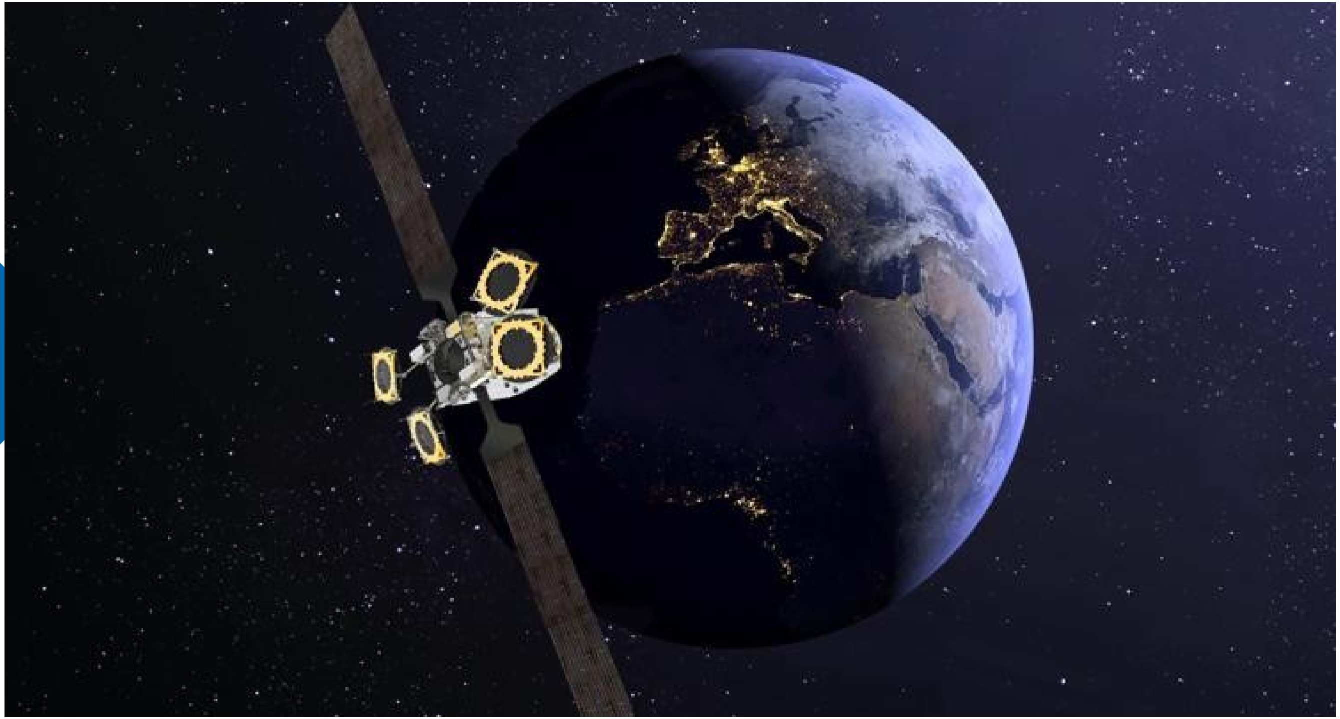
Using **ClickHouse** Open Source columnar database for satellite communication data

OSA CON 2021

Introduction

A real case project using

Clickhouse Open Source database



eutelsat

Eutelsat

Eutelsat is one of the most innovative operators in the commercial satellite business.

Eutelsat Group offers capacity on 36 satellites in geostationary orbit that provide premium coverage of Europe, Africa, the Middle East, Asia and the Americas. The support team has over 1,000 industry professionals from 46 countries located at offices and teleports around the world, ensuring the highest quality of service.

Eutelsat is using **Clickhouse** as a core database component for some of its most recent projects.

Clickhouse

Clickhouse is an Open Source columnar database and is fast, very fast

Clickhouse is a great choice when You need an on-line analysis (OLAP) database

Let’s see why!

Clickhouse

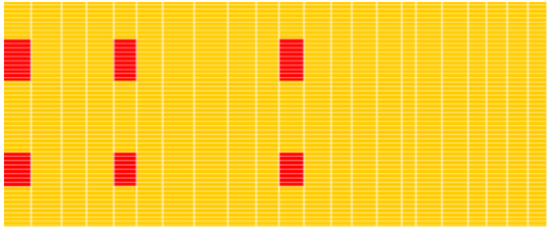
Good reasons to start using Clickhouse:

- Apache 2 license
- Easy to install
- Scale well (from docker to hundreds of nodes in cluster)
- Good SQL (and growing better)
- MySQL like DCL
- Easy to integrate with external data sources
- Very, very fast on OLAP (near real time)

Clickhouse

Very, very fast:

- Column Storage
- Parallel execution
- Vectorized algorithms
- *Delayed* data merge



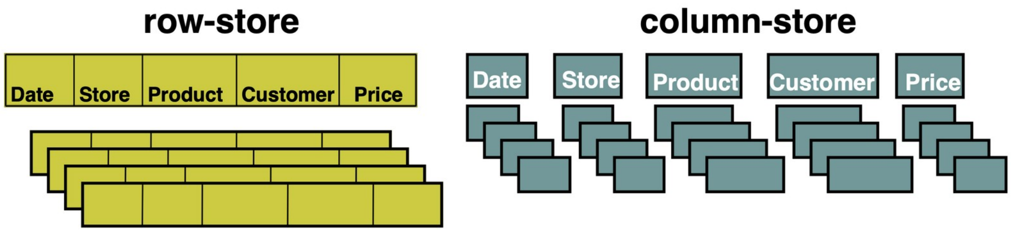
Clickhouse

Let’s create a table:

```
CREATE TABLE invoice (
    date DateTime,
    store UInt32,
    product String,
    customer String,
    price Float32,
    ...
)
ENGINE = MergeTree
PARTITION BY toYYYYMM(date)
ORDER BY (customer, date);
```

Clickhouse

Column-store is quite different from classic row-store



- etc
- var
  - lib
    - clickhouse
      - data
        - default
        - system
        - my_db
          - my_table1
            - 201801_1_8_2 (part)
              - columns.txt
              - minmax_my_pkey.txt
              - primary.idx
              - my_column1.bin
              - my_column1.mrk
              - my_column2.bin
              - my_column2.mrk
              - my_column3.bin
              - my_column3.mrk
            - detached
          - my_table2

Satellite data

Clickhouse is a fast, very fast database designed for OLAP usage.

Clickhouse does not have / does not support: transactions, stored procedure/functions, CRUD operations, optimizer, efficient joins, ...
Clickhouse is an analytical DBMS and has some limitations:
"we recommend expecting a maximum of 100 (short) queries per second"
"we recommend inserting data in packets of at least 1000 rows, or no more than a single request per second"

Can Clickhouse be used for real case data collection project?
Yes... keep reading!

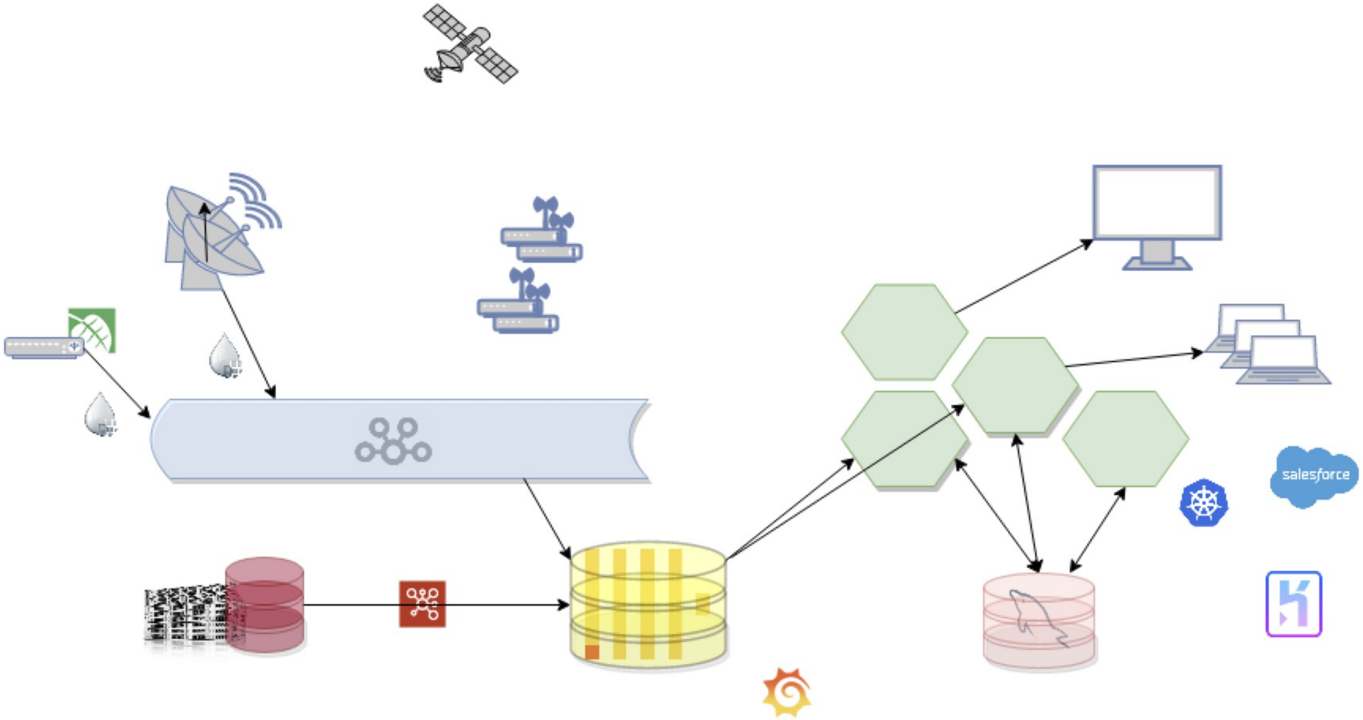
Satellite data

Satellite communication plays a key role in the global connectivity ecosystem, connecting rural and remote populations. In many countries and for many communities satellites are the only connectivity option.

Each satellite/platform is different, there are continuos upgrades, change requests, ...

Most important data are traffic (for billing!) and terminals state (to optimize bandwidth).

A complex ecosystem...



A complex ecosystem…

Hughes satellite platform as data source (CSV)
Newtec Dialog satellite platform as data source (InfluxDB)
Legacy database (Oracle, MySQL) for historical data

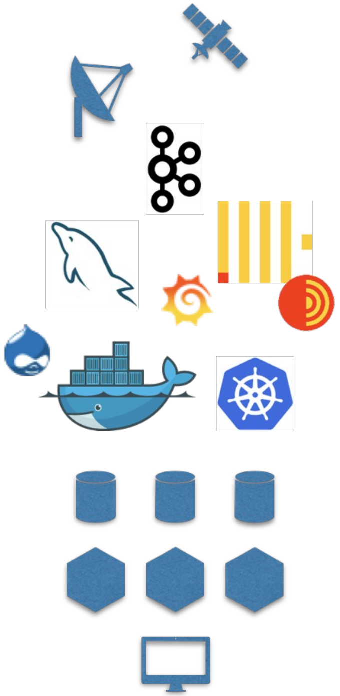
Kafka messaging system as communication bus

Several external services in Cloud

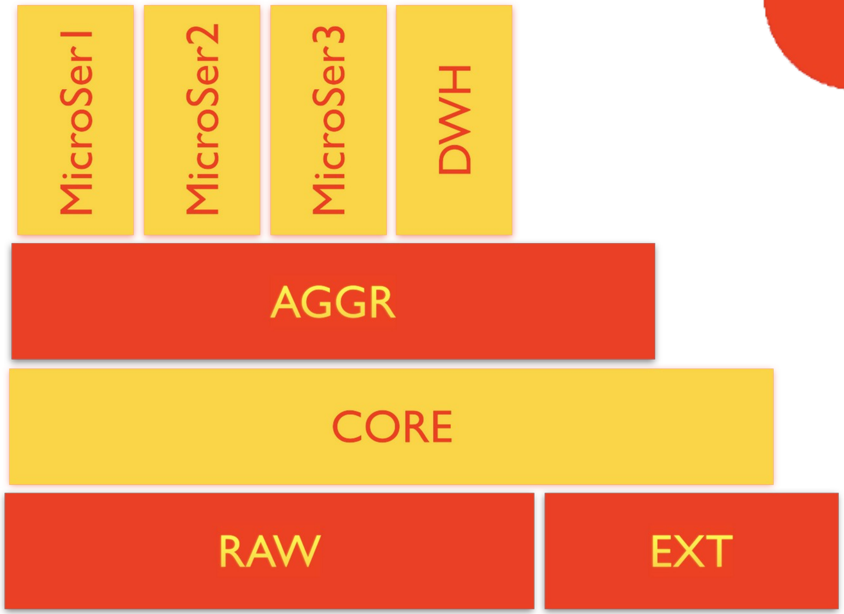
Applications built with Microservices in Java

MySQL as transactional database

ClickHouse for all data ingestion and complex aggregations



Layered schema design



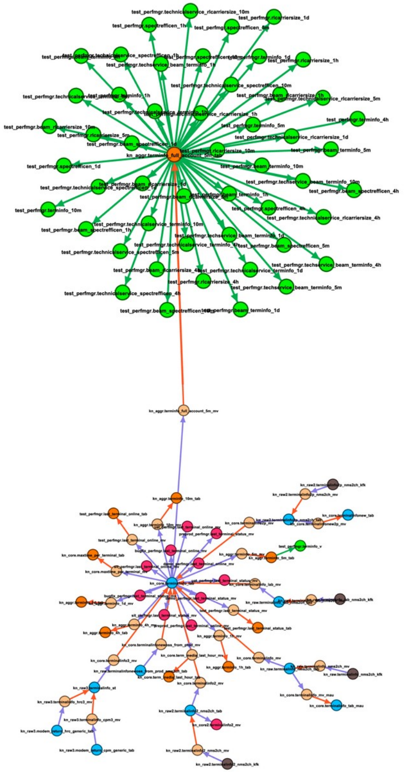
Layered schema design

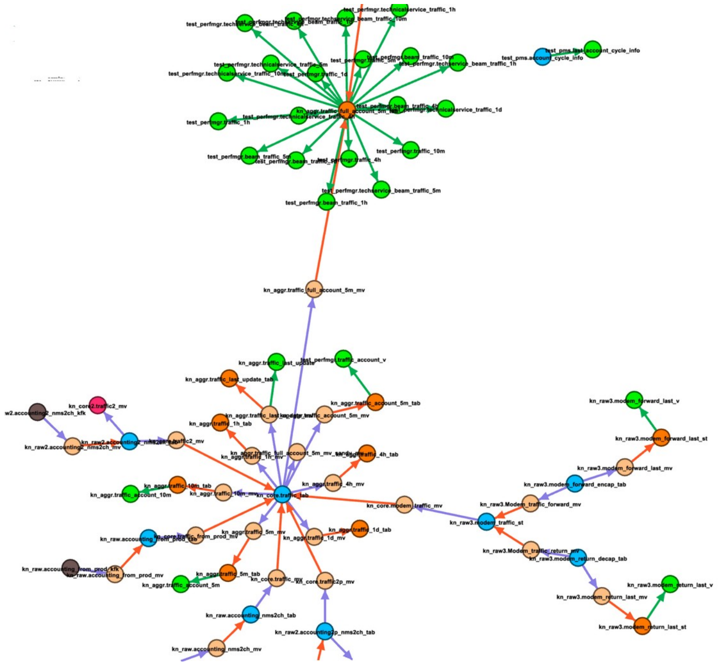
RAW database receives data from the satellite platforms
CORE database contains only useful, checked, optimized data
AGGR database is used for data aggregation

Each application microservice, class of end users, … has a dedicated database
Data *moves* thanks to Views and Materialized Views
Other databases, EXT, COMMON, …

The layered design hides complexity, differences and some “optimizations tricks” to upper levels

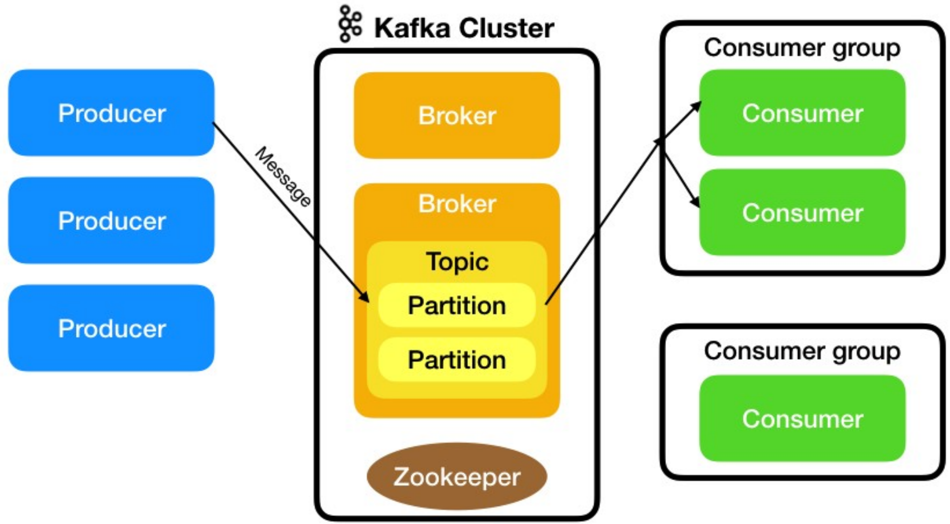
Layered schema design



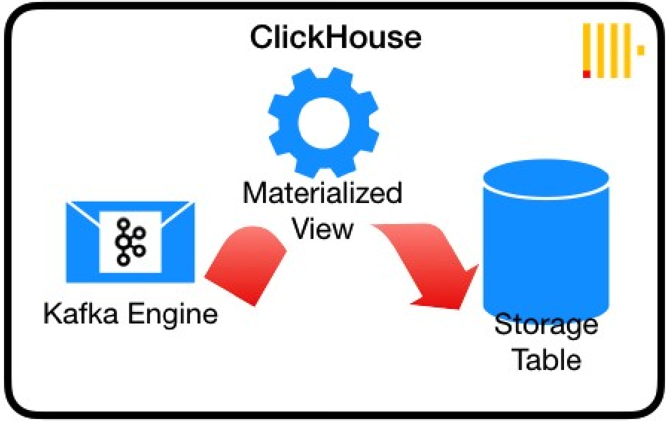


eutelsat

Data ingestion



Data ingestion




```
CREATE TABLE accounting_kfk
(
    `Device_ID` String,
    `Actual_Gateway_ID` String,
    `IPGW_ID` String,
    `Last_Association_Time` String,
    `Collection_Date` String,
    `Collection_Start_Time` String,
    `Collection_End_Time` String,
    `Minutes_Used` String,
... more than 150 fields ...
)
ENGINE = Kafka()
SETTINGS kafka_broker_list = '{kafka_cluster}',
kafka_topic_list = 'xxx', kafka_group_name =
'{replica}_xxx', kafka_format = 'CSV', ...
```

Data ingestion

```
CREATE TABLE accounting_tab
(
    `Device_ID` String,
    `Actual_Gateway_ID` String,
    `IPGW_ID` String,
    `Last_Association_Time` String,
    `Collection_Date` String,
...
)
ENGINE = MergeTree
...
```

```
CREATE MATERIALIZED VIEW accounting_mv
       TO accounting_tab
(
    `Device_ID` String,
    `Actual_Gateway_ID` String,
    `IPGW_ID` String,
...
) AS
SELECT
    _timestamp AS timestamp_k,
    now() AS timestamp_ch,
    Device_ID,
    Actual_Gateway_ID,
    IPGW_ID,
...
FROM accounting_kfk;
```

Optimizations

Partitioning, ordering/indexing:

```
CREATE TABLE aggr.traffic_1h_tab (
    timestamp DateTime,
    external_id Int32,
    duration UInt16,
    hub LowCardinality(String),
    fwc_volume UInt64,
    rtc_volume UInt64
) Engine = MergeTree
PARTITION BY toYYYYMM(timestamp)
ORDER BY (external_id, timestamp);
```

Optimizations

Clickhouse has several features tipical of a Time Series Database: TTL

```
CREATE TABLE traffic_tab
(
    `timestamp` DateTime,
    `external_id` String,
    `source_id` LowCardinality(String) DEFAULT 'KONNECT',
    `rtc_volume` UInt64,
    `fwc_volume` UInt64
)
ENGINE = MergeTree
PARTITION BY toYYYYMM(timestamp)
ORDER BY (external_id, timestamp)
TTL timestamp + toIntervalMonth(6) TO DISK 'slow',
    timestamp + toIntervalMonth(61) DELETE
SETTINGS ttl_only_drop_parts = 1;
```

Optimizations

Compression (default LZ4) and codec:

```
CREATE TABLE aggr.traffic_1h_tab (
    time_ref DateTime Codec(Delta, ZSTD),
    external_id Int32,
    duration Float64 Codec(Gorilla, ZSTD),
    hub LowCardinality(String),
    fwc_volume UInt64 Codec(T64, ZSTD(22)),
    rtc_volume UInt64 Codec(T64, LZ4)
) Engine = MergeTree
PARTITION BY toYYYYMM(time_ref)
ORDER BY (external_id, time_ref);
```

Optimizations

Materialized Views are a distinguishing feature of ClickHouse.

Materialized Views are implemented as an insert trigger on the source table. The MV conditions are applied only to the batch of freshly inserted data.

The can used to collect data from Kafka, to move data to a differently optimized table, to aggregate data, to implement “last point queries”, ...

Optimizations

Materialized Views: creating the base table

```
CREATE TABLE last_coordinate_tab
(
    terminal_id    String,
    timestamp_max  AggregateFunction(max, DateTime),
    latitude       AggregateFunction(argMax,Float32, DateTime),
    longitude      AggregateFunction(argMax,Float32, DateTime)
)
ENGINE = AggregatingMergeTree()
PARTITION BY tuple()
ORDER BY terminal_id;
```

Optimizations

Materialized Views: populating the base table with the MV

```
CREATE MATERIALIZED VIEW last_coordinate_mv
    TO last_coordinate_tab AS
SELECT terminal_id
      ,maxState(timestamp) AS timestamp_max
      ,argMaxState(latitude, timestamp) as latitude
      ,argMaxState(longitude, timestamp) as longitude
  FROM geolocation_tab
 GROUP by terminal_id;
```

Optimizations

Materialized Views: querying the base table with a view

```
CREATE VIEW last_coordinate AS
SELECT terminal_id
      ,maxMerge(timestamp_max) as timestamp
      ,argMaxMerge(latitude) as latitude
      ,argMaxMerge(longitude) as longitude
  FROM last_coordinate_tab
 GROUP BY terminal_id;
```

Optimizations

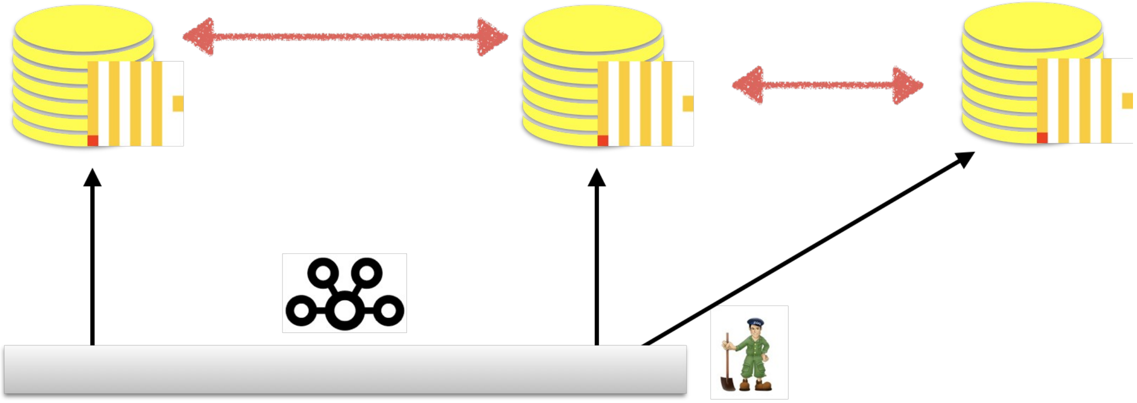
Dictionaries, PREWHERE:

```
SELECT timestamp, external_id,
       dictGet('get_account', 'account_id', external_id) AS account,
...
  FROM traffic_tab
 PREWHERE timestamp > now() - INTERVAL 7 day
 WHERE ...
```

HA

→ Data loading

↔ Replica



OSA CON 2021

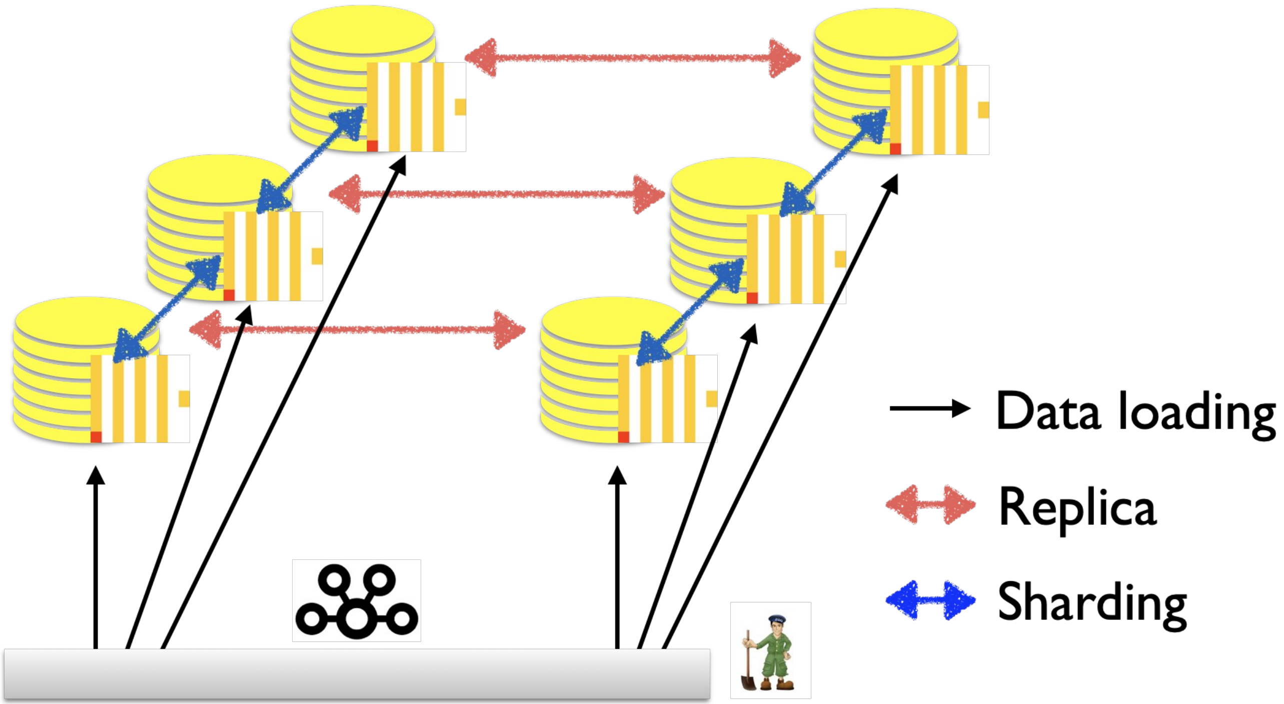
eutelsat

HA

Replica:

```
CREATE TABLE alarm_clock
(
    `timestamp` DateTime,
    `source_id` LowCardinality(String),
    `type` LowCardinality(String) DEFAULT 'TRAFFIC'
)
ENGINE = ReplicatedMergeTree('/clickhouse/{cluster}/tables/{shard}/alarm_clock', '{replica}')
PARTITION BY toYYYYMM(timestamp)
ORDER BY (source_id, timestamp);
```

Scalability



OSA CON 2021

eutelsat

/etc

Some numbers:

- Metrics / Kafka topics: 50
- Source data #fields: 600
- Tables: 450
- Columns: 6000
- Upper level views: 100
- Day merges: 1 TB
- Data: 7 TB
- Biggest table: 0.5 TB
- QPS: 100
- Version: 20.4

Some results:

- More frequent data collection
- Much more metrics
- Less time to production
- Very, very fast on analytic queries
- Cost savings

Analytics

The presentation focus was on Clickhouse Open Source database…

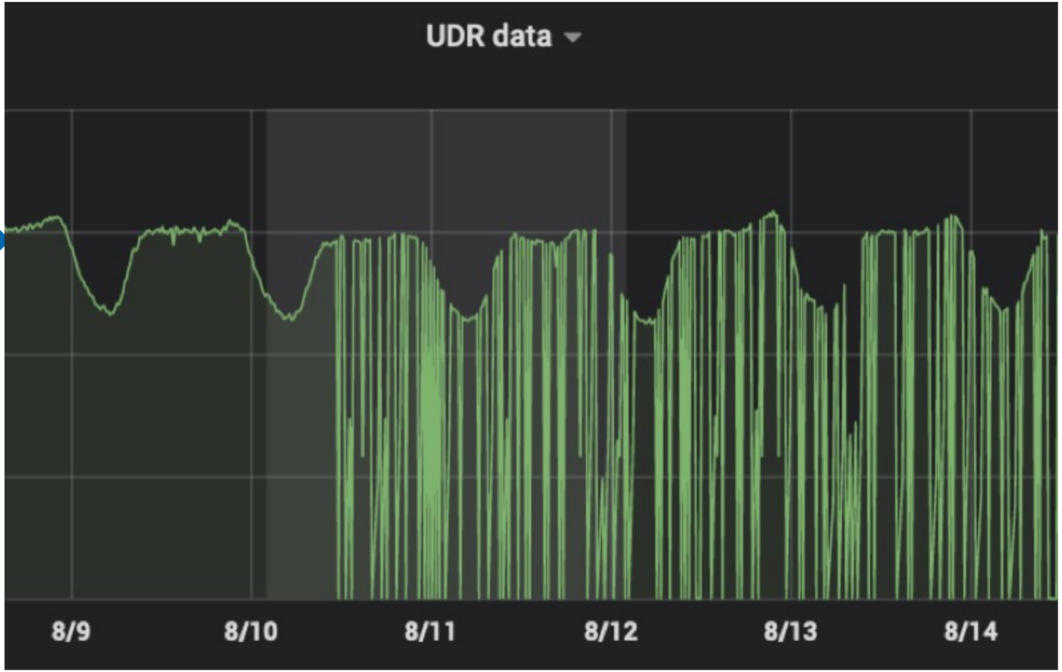
But let's present a couple of examples on why analytics is important!

What happened after installing CH v.19.11.3.11 ?

```
select toStartOfFiveMinute(timestamp), sum(fwc_volume)
  from traffic
 group by toStartOfFiveMinute(timestamp)
 order by toStartOfFiveMinute(timestamp) desc
 limit 20;
```

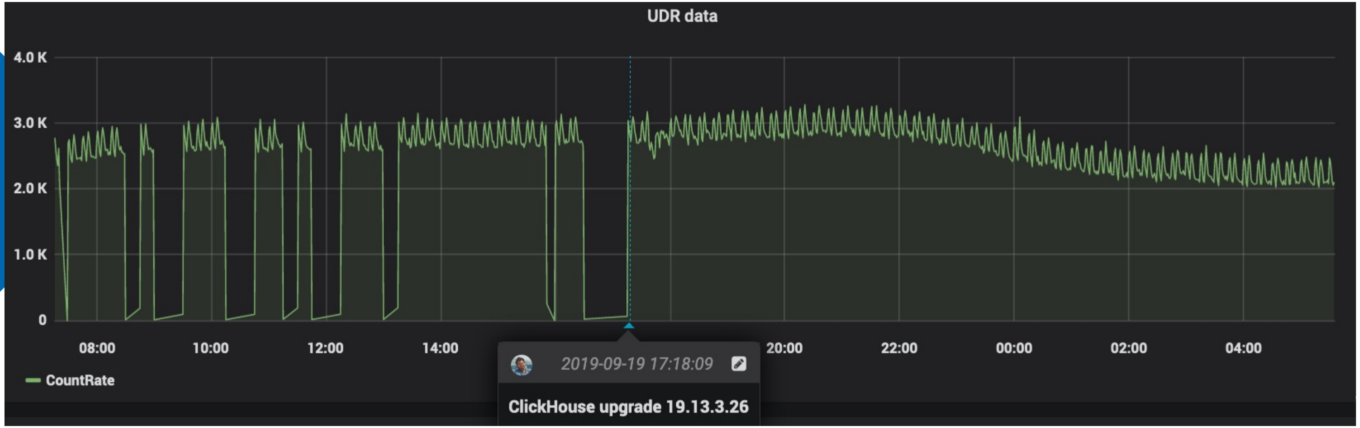
toStartOfFiveMinute(time_ref)	sum(fwc_volume)
2019-08-11 21:30:00	4955854184
2019-08-11 21:25:00	329491077829
2019-08-11 21:20:00	160244921732
2019-08-11 21:15:00	341716444958
2019-08-11 21:10:00	175404086307
2019-08-11 21:05:00	333417505956

What happened …



eutelsat

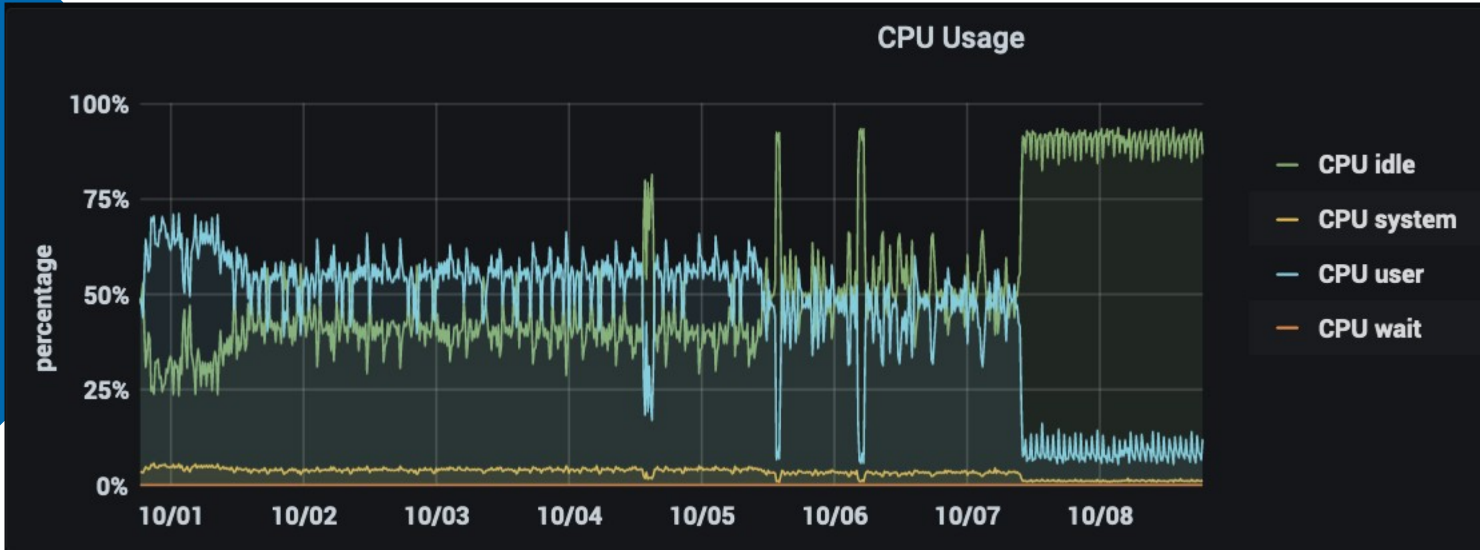
Problem... fixed!



Is the performance problem solved?

We found some slow queries and we optimized with a PREWHERE clause…

Yes: the problem is… fixed!



eutelsat

Analytics

A wise graphical data presentation can be immediately understood by some the oldest Deep Learning tools we have: our eyes and our brain!

Thank You!

mail [AT] meo.bogliolo.name