Real time analysis using ClickHouse and Kafka

Introduction

This project focuses on the development and implementation of a real-time data processing and analysis pipeline using Kafka and ClickHouse. The objective is to build an end-to-end solution for collecting, transforming, and analyzing data from web server logs. The system leverages Kafka as a streaming platform to handle data ingestion and transformation, while ClickHouse is used for real-time data analysis and storage.

The pipeline begins with the ingestion of raw log data from an S3-compatible storage system into Kafka, where it is processed and transformed using ksql. The data is then structured to make it suitable for analysis, extracting key metrics such as IP addresses, user agents, and geolocation information. Through a series of queries, relevant insights are extracted to monitor web access patterns.

Next, the project integrates Kafka with ClickHouse to provide an infrastructure that supports efficient real-time analytics. ClickHouse is used to store and process the transformed data, enabling the generation of detailed reports and statistical insights about web traffic. By utilizing advanced SQL functions, such as window functions and aggregation combinators, the system can produce up-to-date statistics on web access at various levels of granularity, providing valuable insights into user behavior and system performance.

This project demonstrates the integration of powerful data processing tools in a unified pipeline, offering a scalable solution for handling and analyzing large volumes of web server log data in real time.

Data Loading

First, we load the data into the weblog topic using the ibd08-s3-source connector. In this step, we could omit the parameters for the key field, since all our data is located in the value field, as we can see below.

Using the print command, we can verify that the data is present in the topic.

```
ksql> print `weblog` from beginning limit 1;
Key format: ¯\_(ツ )_/¯ - no data processed
Value format: AVRO
rowtime: 2024/04/27 15:01:23.166 Z, key: <null>, value: {"GEOIP": {"IP": "5.188.62.140", "as": {"ORGANIZATION": {"NAME": "Petersburg
Internet Network ltd."}, "NUMBER": 34665}, "GEO": {"POSTAL_CODE": null, "CITY_NAME": null, "COUNTRY_NAME": "Russia", "REGION_ISO_CODE
": null, "TIMEZONE": "Europe/Moscow", "LOCATION": {"LON": 37.6068, "LAT": 55.7386}, "COUNTRY_ISO_CODE": "RU", "REGION_NAME": null, "C
ONTINENT_CODE": "EU"}}, "URL": "/administrator/index.php", "HTTP": {"RESPONSE": {"BODY": {"BYTES": 4494}, "STATUS_CODE": 200}, "REQUE
ST": {"METHOD": "POST", "REFERRER": null}, "VERSION": "1.1"}, "USER_AGENT": {"UAID": "1a184c9ea2806985af981dea8ffe25e9ce454f46", "ORI
GINAL": "Mozilla/5.0 (Windows NT 6.2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2227.1 Safari/537.36", "OS": {"VERSION": "8"
, "NAME": "Windows", "full": "Windows 8"}, "VERSION": "41.0.2227.1", "NAME": "Chrome", "DEVICE": {"NAME": "Other"}}, "APACHE_TS": 164
0998294010}, partition: 0
Topic printing ceased
```

Here we can see how our data contains the IP and UAID that we need.

Additionally, we can see the JSON schema with the following command:

py -c "import requests, json; response = requests.get('http://localhost:8081/schemas/'); schemas = response.json(); print(json.dumps(schemas, indent=4))"

```
PS C:\Users\edert> py -c "import requests, json; response = requests.get('http://localhost:8081/schemas/'); schemas = response.json(); print(json.dumps(schemas, indent=4)
)"
```


```
[
    {
        "subject": "weblog-value",
        "version": 1,
        "id": 1,
        "schema": "{\"type\":\"record\",\"name\":\"KsqlDataSourceSchema\",\"namespace\":\"io.confluent.ksql.avro_schemas\",\"fields\":[{\"name\":\"GEOIP\",\"type\":[\"nul
l\",{\"type\":\"record\",\"name\":\"KsqlDataSourceSchema_GEOIP\",\"fields\":[{\"name\":\"IP\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"as\",\"type\":[
\"null\",{\"type\":\"record\",\"name\":\"KsqlDataSourceSchema_GEOIP_as\",\"fields\":[{\"name\":\"ORGANIZATION\",\"type\":[\"null\",{\"type\":\"record\",\"name\":\"KsqlDat
aSourceSchema_GEOIP_as_ORGANIZATION\",\"fields\":[{\"name\":\"NAME\",\"type\":[\"null\",\"string\"],\"default\":null}],\"connect.name\":\"io.confluent.ksql.avro_schemas.K
sqlDataSourceSchema_GEOIP_as_ORGANIZATION\"}],\"default\":null},{\"name\":\"NUMBER\",\"type\":[\"null\",\"int\"],\"default\":null}],\"connect.name\":\"io.confluent.ksql.a
vro_schemas.KsqlDataSourceSchema_GEOIP_as\"}],\"default\":null},{\"name\":\"GEO\",\"type\":[\"null\",{\"type\":\"record\",\"name\":\"KsqlDataSourceSchema_GEOIP_GEO\",\"fi
elds\":[{\"name\":\"POSTAL_CODE\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"CITY_NAME\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"CO
UNTRY_NAME\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"REGION_ISO_CODE\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"TIMEZONE\",\"type
\":[\"null\",\"string\"],\"default\":null},{\"name\":\"LOCATION\",\"type\":[\"null\",{\"type\":\"record\",\"name\":\"KsqlDataSourceSchema_GEOIP_GEO_LOCATION\",\"fields\":
[{\"name\":\"LON\",\"type\":[\"null\",\"double\"],\"default\":null},{\"name\":\"LAT\",\"type\":[\"null\",\"double\"],\"default\":null}],\"connect.name\":\"io.confluent.ks
ql.avro_schemas.KsqlDataSourceSchema_GEOIP_GEO_LOCATION\"}],\"default\":null},{\"name\":\"COUNTRY_ISO_CODE\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"
REGION_NAME\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"CONTINENT_CODE\",\"type\":[\"null\",\"string\"],\"default\":null}],\"connect.name\":\"io.conflu
ent.ksql.avro_schemas.KsqlDataSourceSchema_GEOIP_GEO\"}],\"default\":null}],\"connect.name\":\"io.confluent.ksql.avro_schemas.KsqlDataSourceSchema_GEOIP\"}],\"default\":n
ull},{\"name\":\"URL\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"HTTP\",\"type\":[\"null\",{\"type\":\"record\",\"name\":\"KsqlDataSourceSchema_HTTP\",
\"fields\":[{\"name\":\"RESPONSE\",\"type\":[\"null\",{\"type\":\"record\",\"name\":\"KsqlDataSourceSchema_HTTP_RESPONSE\",\"fields\":[{\"name\":\"BODY\",\"type\":[\"null
\",{\"type\":\"record\",\"name\":\"KsqlDataSourceSchema_HTTP_RESPONSE_BODY\",\"fields\":[{\"name\":\"BYTES\",\"type\":[\"null\",\"int\"],\"default\":null}],\"connect.name
\":\"io.confluent.ksql.avro_schemas.KsqlDataSourceSchema_HTTP_RESPONSE_BODY\"}],\"default\":null},{\"name\":\"STATUS_CODE\",\"type\":[\"null\",\"int\"],\"default\":null}]
,\"connect.name\":\"io.confluent.ksql.avro_schemas.KsqlDataSourceSchema_HTTP_RESPONSE\"}],\"default\":null},{\"name\":\"REQUEST\",\"type\":[\"null\",{\"type\":\"record\",
\"name\":\"KsqlDataSourceSchema_HTTP_REQUEST\",\"fields\":[{\"name\":\"METHOD\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"REFERRER\",\"type\":[\"null\"
,\"string\"],\"default\":null}],\"connect.name\":\"io.confluent.ksql.avro_schemas.KsqlDataSourceSchema_HTTP_REQUEST\"}],\"default\":null},{\"name\":\"VERSION\",\"type\":[
\"null\",\"string\"],\"default\":null}],\"connect.name\":\"io.confluent.ksql.avro_schemas.KsqlDataSourceSchema_HTTP\"}],\"default\":null},{\"name\":\"USER_AGENT\",\"type\
":[\"null\",{\"type\":\"record\",\"name\":\"KsqlDataSourceSchema_USER_AGENT\",\"fields\":[{\"name\":\"UAID\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"
ORIGINAL\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"OS\",\"type\":[\"null\",{\"type\":\"record\",\"name\":\"KsqlDataSourceSchema_USER_AGENT_OS\",\"fie
lds\":[{\"name\":\"VERSION\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"NAME\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"full\",\"typ
e\":[\"null\",\"string\"],\"default\":null}],\"connect.name\":\"io.confluent.ksql.avro_schemas.KsqlDataSourceSchema_USER_AGENT_OS\"}],\"default\":null},{\"name\":\"VERSIO
N\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"NAME\",\"type\":[\"null\",\"string\"],\"default\":null},{\"name\":\"DEVICE\",\"type\":[\"null\",{\"type\"
:\"record\",\"name\":\"KsqlDataSourceSchema_USER_AGENT_DEVICE\",\"fields\":[{\"name\":\"NAME\",\"type\":[\"null\",\"string\"],\"default\":null}],\"connect.name\":\"io.con
fluent.ksql.avro_schemas.KsqlDataSourceSchema_USER_AGENT_DEVICE\"}],\"default\":null}],\"connect.name\":\"io.confluent.ksql.avro_schemas.KsqlDataSourceSchema_USER_AGENT\"
}],\"default\":null},{\"name\":\"APACHE_TS\",\"type\":[\"null\",{\"type\":\"long\",\"connect.version\":1,\"connect.name\":\"org.apache.kafka.connect.data.Timestamp\",\"lo
gicalType\":\"timestamp-millis\"}],\"default\":null}],\"connect.version\":2,\"connect.name\":\"io.confluent.ksql.avro_schemas.KsqlDataSourceSchema\"}"
    }
]
```


ETL. Data Processing Using KSQL

The first step of the ETL is to move our data from the weblog topic to a stream called ibd08_weblog. Here we can see that the stream has the same structure as the topic.

```
ksql> DESCRIBE RAW_IBD08 EXTENDED;

Name                 : RAW_IBD08
Type                 : STREAM
Timestamp field      : Not set - using <ROWTIME>
Key format           : KAFKA
Value format         : AVRO
Kafka topic          : weblog (partitions: 1, replication: 1)
Statement            : CREATE STREAM RAW_IBD08 (GEOIP STRUCT<IP STRING, `AS` STRUCT<ORGANIZATION STRUCT<NAME STRING>, NUMBER INTEGER>
, GEO STRUCT<POSTAL_CODE STRING, CITY_NAME STRING, COUNTRY_NAME STRING, REGION_ISO_CODE STRING, TIMEZONE STRING, LOCATION STRUCT<LON
DOUBLE, LAT DOUBLE>, COUNTRY_ISO_CODE STRING, REGION_NAME STRING, CONTINENT_CODE STRING>>, URL STRING, HTTP STRUCT<RESPONSE STRUCT<BO
DY STRUCT<BYTES INTEGER>, STATUS_CODE INTEGER>, REQUEST STRUCT<METHOD STRING, REFERRER STRING>, VERSION STRING>, USER_AGENT STRUCT<UA
ID STRING, ORIGINAL STRING, OS STRUCT<VERSION STRING, NAME STRING, `FULL` STRING>, VERSION STRING, NAME STRING, DEVICE STRUCT<NAME ST
RING>>, APACHE_TS TIMESTAMP) WITH (KAFKA_TOPIC='weblog', KEY_FORMAT='KAFKA', VALUE_FORMAT='AVRO');

 Field      | Type


------------------------------------------------------------------------------------------------------------------------------------
 GEOIP      | STRUCT<IP VARCHAR(STRING), AS STRUCT<ORGANIZATION STRUCT<NAME VARCHAR(STRING)>, NUMBER INTEGER>, GEO STRUCT<POSTAL_CODE
 VARCHAR(STRING), CITY_NAME VARCHAR(STRING), COUNTRY_NAME VARCHAR(STRING), REGION_ISO_CODE VARCHAR(STRING), TIMEZONE VARCHAR(STRING),
 LOCATION STRUCT<LON DOUBLE, LAT DOUBLE>, COUNTRY_ISO_CODE VARCHAR(STRING), REGION_NAME VARCHAR(STRING), CONTINENT_CODE VARCHAR(STRIN
G)>>
 URL        | VARCHAR(STRING)


 HTTP       | STRUCT<RESPONSE STRUCT<BODY STRUCT<BYTES INTEGER>, STATUS_CODE INTEGER>, REQUEST STRUCT<METHOD VARCHAR(STRING), REFERRE
R VARCHAR(STRING)>, VERSION VARCHAR(STRING)>

ksql>
 USER_AGENT | STRUCT<UAID VARCHAR(STRING), ORIGINAL VARCHAR(STRING), OS STRUCT<VERSION VARCHAR(STRING), NAME VARCHAR(STRING), FULL VA
ksql> STRING)>, VERSION VARCHAR(STRING), NAME VARCHAR(STRING), DEVICE STRUCT<NAME VARCHAR(STRING)>>


 APACHE_TS  | TIMESTAMP
```

Next, we create the ibd08_accesos stream and the ibd08_geoips table from the ibd08_weblog stream. From this part, we can mention that we use the AS operator to name the column in our new stream/table, the STRUCT operator to create a structure within a column of our new stream/table, the := operator to give a name to each element of this structure, the + operator for concatenations, and the -> operator to specify the path of our data in the source stream (ibd08_weblog). Additionally, to create the table in a single step, we use GROUP BY, including all the data that will make up this table, along with an aggregation function (these two operations are necessary to create the table this way).

We can see that both the stream and the table have been created according to the contents we want them to have by running the following queries.

```
ksql> SELECT * FROM ibd08_accesos LIMIT 3;
>SELECT * FROM ibd08_geoips LIMIT 3;
>
+------------------------+------------------------+------------------------+------------------------+------------------------+------------------------+
|IP                      |UAID                    |UTF8URL                 |UAINFO                  |HTTPINFO                |APACHE_TS               |
+------------------------+------------------------+------------------------+------------------------+------------------------+------------------------+
|5.188.62.140            |1a184c9ea2806985af981dea|L2FkbWluaXN0cmF0b3IvaW5k|{OSINFO=Windows:Windows |{MET=POST, REF=null, STC|2022-01-01T00:51:34.010 |
|                        |8ffe25e9ce454f46        |ZXgucGhw                |8:8, UANAME=Chrome, DEVN|=200}                   |                        |
|                        |                        |                        |AME=Other}              |                        |                        |
|94.130.219.236          |0b7f204dd0e94388df8b351f|L2luZGV4LnBocD9vcHRpb249|{OSINFO=Other:Other:null|{MET=GET, REF=null, STC=|2022-01-01T00:20:42.010 |
|                        |f9a359debaed7c81        |Y29tX3Bob2NhZ2FsbGVyeSZ2|, UANAME=BLEXBot, DEVNAM|200}                    |                        |
|                        |                        |aWV3PWNhdGVnb3J5JmlkPTI6|E=Other}                |                        |                        |
|                        |                        |d2ludGVyZm90b3MmSXRlbWlk|                        |                        |                        |
|                        |                        |PTUzJmxpbWl0c3RhcnQ9MjA=|                        |                        |                        |
|66.249.66.192           |59c3d4f2509c82627f0eeb8e|L3JvYm90cy50eHQ=        |{OSINFO=Other:Other:null|{MET=GET, REF=null, STC=|2022-01-01T00:54:59.010 |
|                        |0e13ae1254eb4065        |                        |, UANAME=Googlebot, DEVN|200}                    |                        |
|                        |                        |                        |AME=Spider}             |                        |                        |
Limit Reached
Query terminated
+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+
|IP        |NOMORG    |NUMORG    |POSTAL_COD|CITY_NAME |COUNTRY_NA|REGION_NAM|REGION_ISO|CONTINENT_|TIMEZONE  |LAT       |LON       |N         |
|          |          |          |E         |          |ME        |E         |_CODE     |CODE      |          |          |          |          |
+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+
|1.10.181.2|TOT Public|23969     |15000     |Lopburi   |Thailand  |Lopburi   |TH-16     |AS        |Asia/Bangk|14.8025   |100.6181  |1         |
|44        | Company L|          |          |          |          |          |          |          |ok        |          |          |          |
|          |imited    |          |          |          |          |          |          |          |          |          |          |          |
|1.20.235.1|TOT Public|23969     |95000     |Yala      |Thailand  |Yala      |TH-95     |AS        |Asia/Bangk|6.5409    |101.2806  |1         |
|53        | Company L|          |          |          |          |          |          |          |ok        |          |          |          |
|          |imited    |          |          |          |          |          |          |          |          |          |          |          |
|100.25.221|AMAZON-AES|14618     |20149     |Ashburn   |United Sta|Virginia  |US-VA     |NA        |America/Ne|39.0469   |-77.4903  |1         |
|.141      |          |          |          |          |tes       |          |          |          |w_York    |          |          |          |
Query terminated
ksql> |
```

Validation Queries

To validate our ETL, we will perform several queries. First, we create an intermediate stream called ibd08_adh_schema from ibd08_accesos, where we select the data and give it the structure required. We do this because from this stream, we build another stream called ibd08_adh that creates the ibd08.adh topic, which contains the IP and UAID in the key field of our data. The reason for creating this intermediate stream is that the PARTITION BY clause (which allows us to place our data in the key field of the topic) requires that the column passed to it belongs to the source stream.

Here we can see how we first give our data the structure we want.

```
ksql> DESCRIBE ibd08_adh_schema EXTENDED;

Name                 : IBD08_ADH_SCHEMA
Type                 : STREAM
Timestamp field      : Not set - using <ROWTIME>
Key format           : KAFKA
Value format         : AVRO
Kafka topic          : ibd08.adh.schema (partitions: 1, replication: 1)
Statement            : CREATE STREAM IBD08_ADH_SCHEMA WITH (KAFKA_TOPIC='ibd08.adh.schema', PARTITIONS=1, REPLICAS=1, VALUE_FORMAT='AVRO') AS SELECT
  STRUCT(IP:=IBD08_ACCESOS.IP, UAID:=IBD08_ACCESOS.UAID) K,
  FORMAT_TIMESTAMP(IBD08_ACCESOS.APACHE_TS, 'dd') DD,
  FORMAT_TIMESTAMP(IBD08_ACCESOS.APACHE_TS, 'HH') HH
FROM IBD08_ACCESOS IBD08_ACCESOS
EMIT CHANGES;

 Field | Type
----------------------------------------------------------
 K     | STRUCT<IP VARCHAR(STRING), UAID VARCHAR(STRING)>
 DD    | VARCHAR(STRING)
 HH    | VARCHAR(STRING)
----------------------------------------------------------
```

Next, we give the appropriate format to both the key and the value of the topic, and with PARTITION BY, we make the K column the key of the stream.

```
ksql> DESCRIBE ibd08_adh EXTENDED;

Name                 : IBD08_ADH
Type                 : STREAM
Timestamp field      : Not set - using <ROWTIME>
Key format           : AVRO
Value format         : AVRO
Kafka topic          : ibd08.adh (partitions: 1, replication: 1)
Statement            : CREATE STREAM IBD08_ADH WITH (KAFKA_TOPIC='ibd08.adh', KEY_FORMAT='AVRO', PARTITIONS=1, REPLICAS=1, VALUE_FORMAT='AVRO') AS SELECT *
FROM IBD08_ADH_SCHEMA IBD08_ADH_SCHEMA
PARTITION BY IBD08_ADH_SCHEMA.K
EMIT CHANGES;

 Field | Type
----------------------------------------------------------------
 K     | STRUCT<IP VARCHAR(STRING), UAID VARCHAR(STRING)> (key)
 DD    | VARCHAR(STRING)
 HH    | VARCHAR(STRING)
----------------------------------------------------------------
```

```
ksql> print `ibd08.adh` limit 1;
Key format: AVRO or HOPPING(KAFKA_STRING) or TUMBLING(KAFKA_STRING) or KAFKA_STRING
Value format: AVRO or KAFKA_STRING
rowtime: 2024/04/27 15:01:23.166 Z, key: {"IP": "5.188.62.140", "UAID": "1a184c9ea2806985af981dea8ffe25e9ce454f46"}, value: {"DD": "01", "HH": "00"}, partit
ion: 0
Topic printing ceased
```

Additionally, we can see that the result of the stream is correct.

```
ksql> SELECT * FROM ibd08_adh LIMIT 3;
+--------------------------------------------------+--------------------------------------------------+--------------------------------------------------+
|K                                                 |DD                                                |HH                                                |
+--------------------------------------------------+--------------------------------------------------+--------------------------------------------------+
|{IP=5.188.62.140, UAID=1a184c9ea2806985af981dea8ff|01                                                |00                                                |
|e25e9ce454f46}                                    |                                                  |                                                  |
|{IP=94.130.219.236, UAID=0b7f204dd0e94388df8b351ff|01                                                |00                                                |
|9a359debaed7c81}                                  |                                                  |                                                  |
|{IP=66.249.66.192, UAID=59c3d4f2509c82627f0eeb8e0e|01                                                |00                                                |
|13ae1254eb4065}                                   |                                                  |                                                  |
Limit Reached
Query terminated
```

In the second validation query for ibd08_accesos, we create the ibd08_adh10 table with the number of accesses per day and hour for the IP, UAID keys that record more than 10 accesses in each period. We do this in a similar way to how we created the ibd08_geoips table in our ETL, but using the HAVING clause in addition to GROUP BY, and applying the FORMAT_TIMESTAMP function.
With this query, we can see that we obtain the result we wanted.

```
ksql> select * from ibd08_adh10 where (DD = '01') LIMIT 3;
+------------------------------+------------------------------+------------------------------+------------------------------+------------------------------+
|IP                            |UAID                          |DD                            |HH                            |N                             |
+------------------------------+------------------------------+------------------------------+------------------------------+------------------------------+
|206.62.3.58                   |30f39198a85204522d1cf64ce5837|01                             |22                            |42                            |
|                              |2b3b1537878                   |                              |                              |                              |
|62.138.2.214                  |56fc8c32673de1f7add1e47c8a1b2|01                             |08                            |11                            |
|                              |3f95a9ce479                   |                              |                              |                              |
|77.119.174.167                |2c19ba4c72f613f9824bd037fccfd|01                             |09                            |71                            |
|                              |243ef928970                   |                              |                              |                              |
Query terminated
```

With these queries, we consider the ETL validated, as we can see that it is functioning correctly for our case.

Data Integration in Kafka

First, we create the ibd08 database, which will be used for our work. Then, we create the "accesos" table exactly as provided in the supporting material, which will store the transformed information through a materialized view obtained from Kafka. To retrieve the data from the ETL created in P1, we create the kafka_accesos table and specify the ibd08.accesos topic, which contains the data from the ETL. Additionally, we use the Clickhouse Tuple function to load the structured data coming from Kafka.
First, we create the ibd08.accesos table with the following query:

```sql
CREATE TABLE ibd08.accesos
(
    ip IPv4 NOT NULL,
    uaid String NOT NULL,
    url String NOT NULL,
    osinfo Nullable(String),
    uaname Nullable(String),
    devname Nullable(String),
    meth Nullable(String),
    uref Nullable(String),
    status_code Nullable(Int32),
    ts DateTime64
) ENGINE = MergeTree ORDER BY (ip, uaid, url, ts);
```

Next, we will create the ibd08.kafka_accesos table, which will load the data passing through the ibd08.accesos topic of our Kafka architecture.

```sql
CREATE TABLE ibd08.kafka_accesos
(
    IP String,
    UAID String,
    UTF8URL String,
    UAINFO Tuple(OSINFO Nullable(String), UANAME Nullable(String), DEVNAME Nullable(String)),
    HTTPINFO Tuple(MET Nullable(String), REF Nullable(String), STC Nullable(Int32)),
    APACHE_TS Nullable(DateTime64)
) ENGINE = Kafka()
SETTINGS
    kafka_broker_list = 'broker:9092',
    kafka_topic_list = 'ibd08.accesos',
    kafka_group_name = 'clickhouse',
    kafka_format = 'AvroConfluent',
    format_avro_schema_registry_url = 'http://schema-registry:8081';
```

To implement it in the course's project, we would need to modify the topic from "ibd08.accesos" to "accesos".
Finally, the materialized view is needed to connect our two tables:

```sql
CREATE MATERIALIZED VIEW ibd08.accesos_mv TO ibd08.accesos AS
SELECT
    IP as ip,
    UAID as uaid,
    UTF8URL as url,
    UAINFO.OSINFO as osinfo,
    UAINFO.UANAME as uaname,
    UAINFO.DEVNAME as devname,
    HTTPINFO.MET as meth,
    HTTPINFO.REF as uref,
    HTTPINFO.STC as status_code,
    APACHE_TS as ts
FROM ibd08.kafka_accesos;
```

At this point, we can load our Kafka data into Clickhouse.
We then decide to perform a small check to verify if the data has been loaded correctly and if we are following the correct steps. By running the query *SELECT * FROM ibd08.kafka_accesos LIMIT 3;*, we encounter the error *DB::Exception: Direct select is not allowed. To enable use setting 'stream_like_engine_allow_direct_select'. (QUERY_NOT_ALLOWED).*

To resolve this, we execute the command *SET stream_like_engine_allow_direct_select = 1;*. Now, we can see that the data has been successfully loaded.

```
clickhouse :) SELECT * FROM ibd08.kafka_accesos LIMIT 3;

SELECT *
FROM ibd08.kafka_accesos
LIMIT 3

Query id: 971628af-e69f-4048-a5bd-fe75078126fb

   ┌─IP─────────────┬─UAID─────────────────────────────┬─UTF8URL──────────────────────────────────────────────────────────────────────────┬─UAINFO──────────────────────────────────────┬─HTTPINFO───────────────┬──────────────APACHE_TS─┐
1. │ 5.188.62.140   │ 1a184c9ea2806985af981dea8ffe25e9ce454f46 │ /administrator/index.php                                                  │ ('Windows:Windows 8:8','Chrome','Other')    │ ('POST',NULL,200)      │ 2022-01-01 00:51:34.010 │
2. │ 94.130.219.236 │ 0b7f204dd0e94388df8b351ff9a359debaed7c81 │ /index.php?option=com_phocagallery&view=category&id=2:winterfotos&Itemid=53&limitstart=20 │ ('Other:Other:null','BLEXBot','Other') │ ('GET',NULL,200) │ 2022-01-01 00:20:42.010 │
3. │ 66.249.66.192  │ 59c3d4f2509c82627f0eeb8e0e13ae1254eb4065 │ /robots.txt
              │ ('Other:Other:null','Googlebot','Spider') │ ('GET',NULL,200)  │ 2022-01-01 00:54:59.010 │
   └────────────────┴──────────────────────────────────┴──────────────────────────────────────────────────────────────────────────────────┴─────────────────────────────────────────────┴────────────────────────┴─────────────────────────┘

3 rows in set. Elapsed: 0.573 sec. Processed 7.06 thousand rows, 1.75 MB (12.31 thousand rows/s., 3.05 MB/s.)
Peak memory usage: 785.86 KiB.
```

Now, we create the materialized view *accesos_mv* to pass the data to our *accesos* table and apply a series of changes. We can see that the data is loaded correctly with the requested format.

```
clickhouse :) SELECT * FROM ibd08.accesos limit 3;

SELECT *
FROM ibd08.accesos
LIMIT 3

Query id: 0f262666-c070-4221-9931-7b8248a57ade

   ┌─ip───────────┬─uaid─────────────────────────────────────┬─url────────────────────┬─osinfo──────────────────┬─uaname────────┬─devname─┬─meth─┬─uref──────────────────────────────┬─status_code─┬──────────────────────ts─┐
1. │ 1.10.181.244 │ da59a5cd1c7de8ce80fe549457054b6b0fb68a5c │ /apache-log/access.log │ Linux:Linux:null        │ Chrome        │ Other   │ GET  │ http://www.almhuette-raith.at/    │         200 │ 2022-01-11 04:50:46.010 │
2. │ 1.15.175.155 │ 3c920fe3fe9cd766f43ae032e374e60166eac74f │ /apache-log/access.log │ Android:Android 6.0:6.0 │ Chrome Mobile │ Nexus 5 │ GET  │ NULL                              │         200 │ 2022-01-03 06:22:41.010 │
3. │ 1.15.175.155 │ 3c920fe3fe9cd766f43ae032e374e60166eac74f │ /apache-log/access.log │ Android:Android 6.0:6.0 │ Chrome Mobile │ Nexus 5 │ GET  │ NULL                              │         200 │ 2022-01-03 06:23:21.010 │
   └──────────────┴──────────────────────────────────────────┴────────────────────────┴─────────────────────────┴───────────────┴─────────┴──────┴───────────────────────────────────┴─────────────┴─────────────────────────┘

3 rows in set. Elapsed: 0.006 sec. Processed 7.06 thousand rows, 1.61 MB (1.13 million rows/s., 257.53 MB/s.)
Peak memory usage: 384.75 KiB.
```

Regarding the materialized view, we use the "." operator to access the structured data (e.g., UAINFO.OSINFO).

Data Analysis

Exploratory analysis

To perform the exploratory data analysis, we have created a table called *informe*, where we will insert the result of a select query on the *accesos* table.
The `informe` table will contain the following columns: f (date), h (hour), url, di (distinct IPs), du (distinct URLs), au (access for URL). This table will be created as follows:

```sql
CREATE TABLE ibd08.informe (
    f Date NOT NULL,
    h UInt8 NOT NULL,
    url String NOT NULL,
    di Nullable(UInt32),
    du Nullable(UInt32),
    au Nullable(UInt32)
) ENGINE = ReplacingMergeTree() ORDER BY (f, h, url);
```

The query to insert the data we are interested in into this new table is as follows:

```sql
INSERT INTO ibd08.informe
SELECT
    toDate(ts) AS f,
    toUInt8(toHour(ts)) AS h,
    url,
    ip OVER (PARTITION BY h, f) AS di,
    url OVER (PARTITION BY h, f) AS du,
    COUNT(url) OVER (PARTITION BY f, h, url) AS au
FROM ibd08.accesos
ORDER BY f, h, url;
```

In it, we do the following:

1. Select the tuples "f," "h," and "url" that are unique, where *f* is the date in "YYYY-MM-DD" format, *h* is the time from the datetime, and *url* is the URL value.
2. 2. Select the number of unique IPs over a partition of the columns *h* and *f*, meaning the number of unique IPs for each distinct group of *h* and *f*.
3. 3. Similarly, select the number of unique URLs over a partition of the columns *h* and *f*, that is, the number of unique URLs for each distinct group of *h* and *f*.
4. 4. Finally, we select the number of times a URL appeared for each distinct group of *h*, *f*, and *url* to know how many times that URL appeared by date and time.

It is necessary to perform the insert once the entire infrastructure is set up and the data is in the *accesses* table; otherwise, we will not obtain any results. We can see that the results we obtain are as expected.

```
clickhouse :) SELECT * FROM ibd08.informe limit 3;

SELECT *
FROM ibd08.informe
LIMIT 3

Query id: 8cf87f70-d598-4f0e-aa4b-44e851749632

   ┌──────────f─┬─h─┬─url─────────────────────────────┬─di─┬─du─┬─au─┐
1. │ 2022-01-01 │ 0 │ /administrator/index.php        │  5 │  6 │  2 │
2. │ 2022-01-01 │ 0 │ /index.old                      │  5 │  6 │  1 │
3. │ 2022-01-01 │ 0 │ /index.php?format=feed&type=rss │  5 │  6 │  1 │
   └────────────┴───┴─────────────────────────────────┴────┴────┴────┘

3 rows in set. Elapsed: 0.009 sec. Processed 4.00 thousand rows, 278.62 KB (468.10 thousand rows/s., 32.61 MB/s.)
Peak memory usage: 231.05 KiB.
```

Hourly statistical analysis

In the hourly statistical analysis, our goal is to have a table composed of aggregates that, through a materialized view and using the "state" and "merge" utilities, allows us to keep this table updated despite the incoming data flow. To achieve this, we will first create a table using the *AggregatingMergeTree* storage engine, which allows us to define columns as aggregation functions. The *hstats* table is as follows:

```sql
create table ibd08.hstats
(
    f Date NOT NULL,
    h UInt8 NOT NULL,
    urls AggregateFunction(uniq, String),
    ips AggregateFunction(uniq, IPv4),
    uaids AggregateFunction(uniq, String),
    tot AggregateFunction(count, UInt32)
) ENGINE = AggregatingMergeTree ORDER BY (f, h);
```

Where *f* is the date, *h* is the hour, *urls* is an aggregation function of the type *uniq*, as are *ips* and *uaids*, and *tot* is an aggregation function of the type *count*. We combine this table with the following materialized view:

```sql
create MATERIALIZED view ibd08.hstats_mv
TO ibd08.hstats
as select
    toDate(ts) as f,
    toHour(ts) as h,
    uniqState(url) as urls,
    uniqState(ip) AS ips,
    uniqState(uaid) AS uaids,
    countState(h) AS tot
from ibd08.accesos
group by f, h
order by f, h, urls;
```

In this materialized view, we begin by defining the states we will use to solve our task. Since we use the aggregation functions *uniq* and *count* in our aggregation table, we will use *uniqState* and *countState* to store the state in our materialized view. Similarly, we convert the *DateTime* from the *accesses* table into date and time.

Finally, we execute the query to obtain the result:

```sql
select f, h, uniqMerge(urls) AS urls, uniqMerge(ips) AS ips,
    uniqMerge(uaids) AS uaids, countMerge(tot) AS tot
    from ibd08.hstats  GROUP BY f, h ORDER BY f, h;
```

In this query, just as we used *uniqState* and *countState* to store the state, we use *uniqMerge* and *countMerge* on the corresponding columns to retrieve the values.

We obtain the correct results (many more than can be displayed in the screenshot):

```
SELECT
    f,
    h,
    uniqMerge(urls) AS urls,
    uniqMerge(ips) AS ips,
    uniqMerge(uaids) AS uaids,
    countMerge(tot) AS tot
FROM hstats
GROUP BY
    f,
    h
ORDER BY
    f ASC,
    h ASC

Query id: 56dff49d-228f-45ed-b96d-0f604e02955d
```

	f	h	urls	ips	uaids	tot
1.	2022-01-01	0	6	5	5	8
2.	2022-01-01	1	2	4	3	4
3.	2022-01-01	2	26	14	6	29
4.	2022-01-01	3	2	1	1	2
5.	2022-01-01	4	11	11	4	11
6.	2022-01-01	5	4	7	5	12
7.	2022-01-01	7	6	4	4	10
8.	2022-01-01	8	12	3	3	14
9.	2022-01-01	9	67	4	4	75
10.	2022-01-01	10	38	3	3	44
11.	2022-01-01	11	4	8	4	11
12.	2022-01-01	12	6	6	6	11

Monthly statistical analysis

This task is entirely analogous to the previous one, except that instead of working with hours, we will work with months. With that said, we only need to make minor changes in the queries to obtain the correct results.

The aggregation table *mstats* would be as follows:

```
create table ibd08.mstats
(
    f Date NOT NULL,
    m UInt8 NOT NULL,
    urls AggregateFunction(uniq, String),
    ips AggregateFunction(uniq, IPv4),
    uaids AggregateFunction(uniq, String),
    tot AggregateFunction(count, UInt32)
) ENGINE = AggregatingMergeTree ORDER BY (f, m);

create MATERIALIZED view ibd08.mstats_mv
TO ibd08.mstats
as select
    toDate(ts) as f,
    toMonth(ts) as m,
    uniqState(url) as urls,
    uniqState(ip) AS ips,
    uniqState(uaid) AS uaids,
    countState(m) AS tot
from ibd08.accesos
group by f, m
order by f, m, urls;
```

And the query to select the data:

```sql
select m, uniqMerge(urls) AS urls, uniqMerge(ips) AS ips,
    uniqMerge(uaids) AS uaids, countMerge(tot) AS tot
    from ibd08.mstats GROUP BY m ORDER BY m;
```

As can be seen, by simply replacing the hour with the month, we obtain the results corresponding to this new time interval. Therefore, if we wanted to do this for years, quarters, semesters, or any other new time interval, we would just need to modify the statements and define the column as this new interval instead of the month. We obtain a similar result to the previous one, but now by months:

m	urls	ips	uaids	tot
1	375	999	401	7.059