Webinar

Using the Schema-Agnostic Design Pattern on ClickHouse for Product Analytics at Sumsub

Speaker: Olga Silyutina

Product Analytics Lead

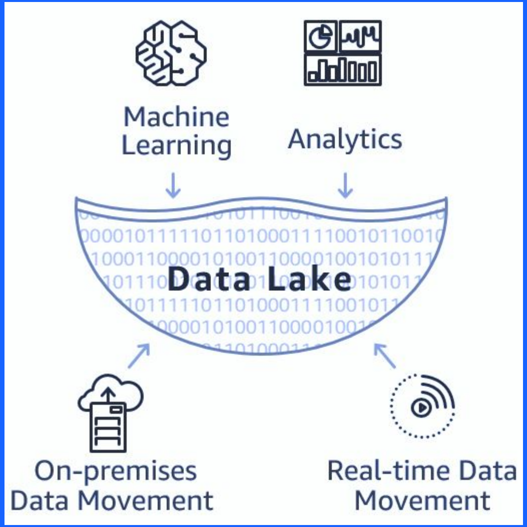
20 July
10:00 CET

sumsub

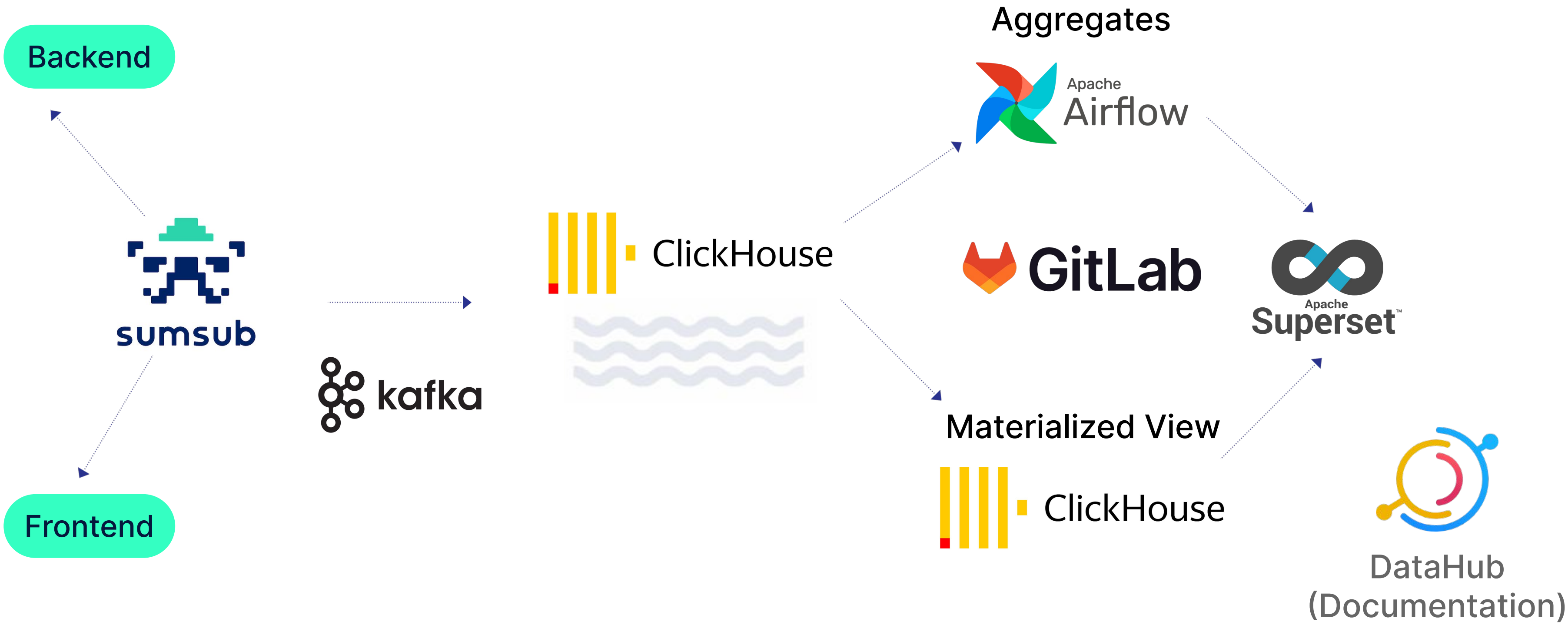
One verification platform to secure the whole user journey

Why use schema-agnostic approach in ClickHouse?

- **Active phase** of feature development without understanding all the needed columns
- **Adding new calculated metrics to the process** (e.g. A/B tests)
- Other technical logs with **no specific requirements**



How do we collect data for product analytics?



Source

Data Lake

Transformation

Product Analytics

Showcase: Frontend logs with Materialized Views

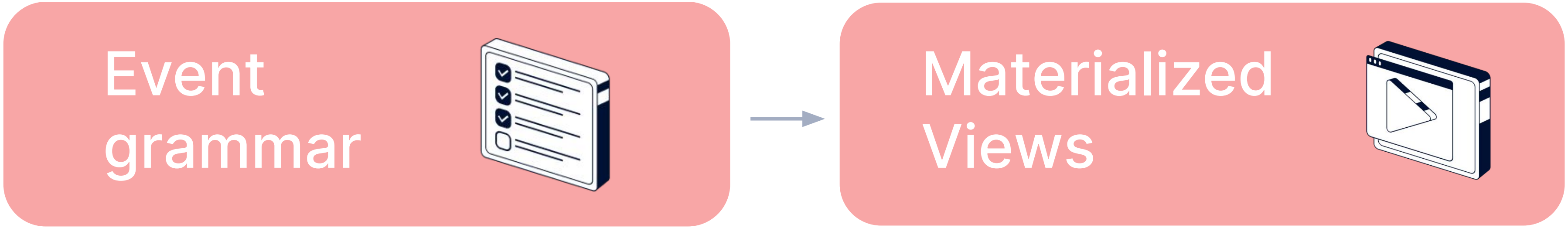
Initial state:

- Every engineer create their **own event logic**
- **One end-point table** for all logs
- Analysts need **lots of context** to calculate simple metrics

```
SELECT dayTs,
      JSONExtractString(metadata,
'customField') AS customField,
      JSONExtractString(metadata, 'statCol')
AS statCol,
      JSONExtractRaw(metadata, 'newCol')
AS newCol,
…
FROM actions
WHERE dayTs >= today() - 30
 AND action = 'random:event:with:diffSize'*
```

* Example of the select request to the event with JSON “metadata” column and action grammar before changes

Materialized View for frontend logs



Event grammar

Materialized Views

Event grammar

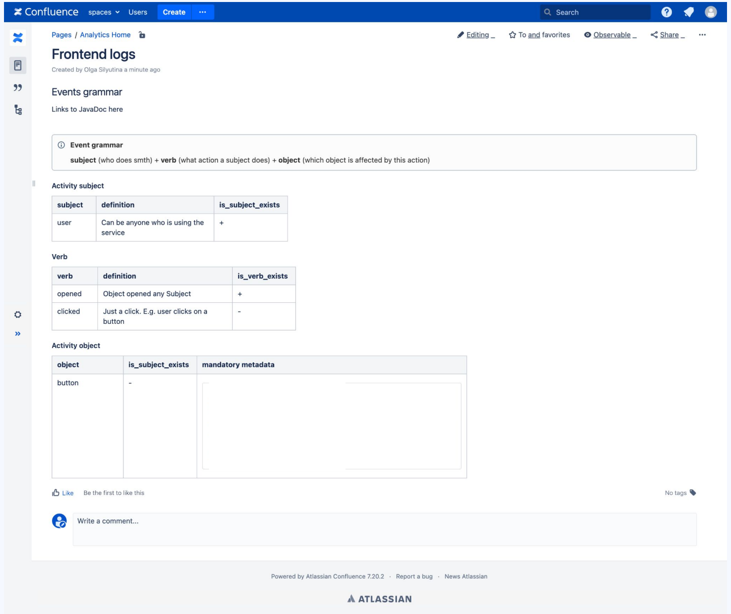
Starting from the documented log structure

- Setting the unified events grammar subject:verb:object
- Documenting the required structure of the JSON (e.g. Confluence)

subject (who does something) +
verb (what action a subject does) +
object (which object is affected by this action)

Confluence + Javadoc

Confluence *



Frontend logs

Created by Olga Silyutina a minute ago

Events grammar

Links to JavaDoc here

Event grammar

subject (who does smth) + **verb** (what action a subject does) + **object** (which object is affected by this action)

Activity subject

subject	definition	is_subject_exists
user	Can be anyone who is using the service	+

Verb

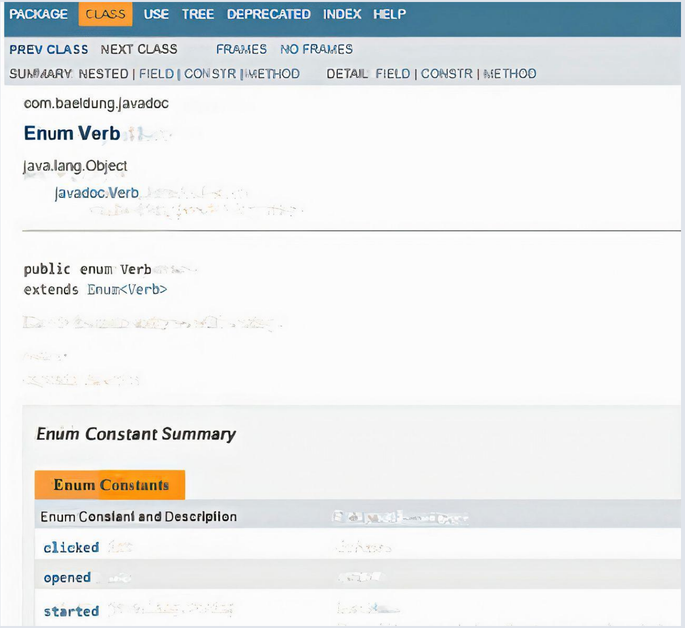
verb	definition	is_verb_exists
opened	Object opened any Subject	+
clicked	Just a click. E.g. user clicks on a button	-

Activity object

object	is_subject_exists	mandatory metadata
button	-	

Like
Be the first to like this
No tags
Write a comment...
Powered by Atlassian Confluence 7.20.2 · Report a bug · News Atlassian
ATLASSIAN

JavaDoc (documentation generator) *



com.baeldung.javadoc

Enum Verb

java.lang.Object

javadoc.Verb

```
public enum Verb
extends Enum<Verb>
```

Enum Constant Summary

Enum Constants

Enum Constant and Description
clicked
opened
started

* Example of Confluence documentation structure for analysts

* Example of JavaDoc interface from devs

Tools and formats for documentation

Documentation generators	Description formats	Description tools
Doxygen	AVRO™	SMARTBEAR SwaggerHub
JavaDoc	YAML	GitLab
JSDoc	{json}	Confluence

More context on user flow

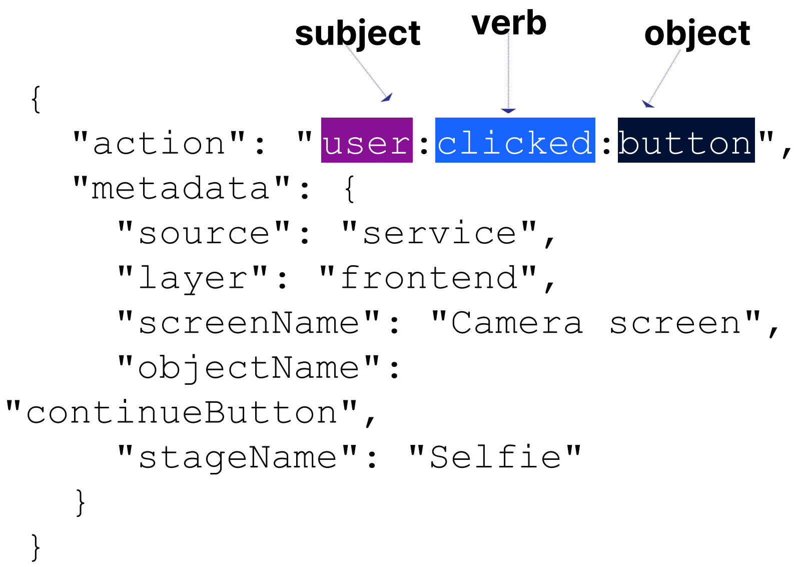
user:started:stage

```
{
  "stageName" : "Selfie",
  "screenName" : "Camera screen",

  ...

  "source" : "service"
}
```

stageName	screenName	source
Selfie	Camera screen	service

Selfie with document
Questionnaire
Make a snapshot
Complete call

JSON structure for metadata of frontend logs



subject verb object

```
 {
   "action": "user:clicked:button",
   "metadata": {
     "source": "service",
     "layer": "frontend",
     "screenName": "Camera screen",
     "objectName":
"continueButton",
     "stageName": "Selfie"
   }
 }
```

Example events in Data Lake

dayTs	userId	action	metadata

2023-04-01 15:06:07	1234567890	user:started:step	{
			"stageName" : "Selfie",
			"screenName" : "Camera options",
			"objectName" : "Selfie",
			"source" : "service",
			"layer" : "frontend",
			...
			}

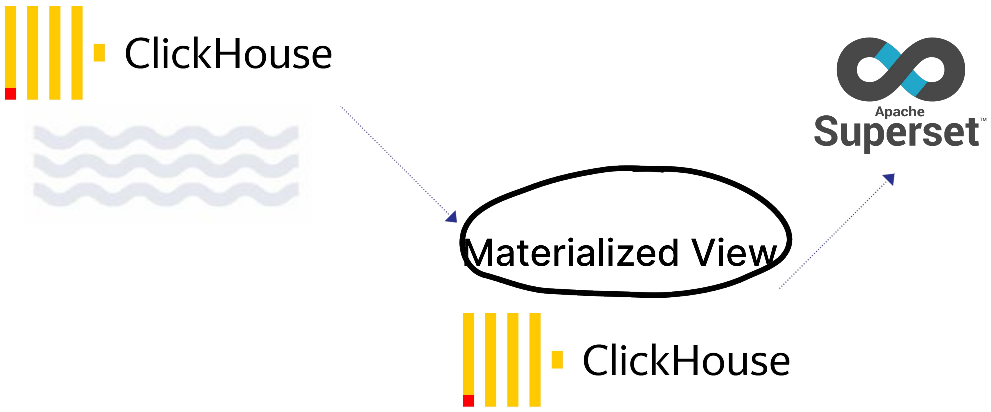
2023-04-01 15:06:12	1234567890	user:clicked:button	{
			"stageName" : "Warning",
			"screenName" : "Warning",
			"objectName" : "Continue button",
			"source" : "service",
			"layer" : "frontend",
			...
			}

Search for clicks by any button

```
SELECT dayTs,
      userId,
      action,
      JSONExtractString(metadata, 'source')          as source,
      JSONExtractString(metadata, 'layer')           as layer,
      JSONExtractString(metadata, 'screenName')      as screenName,
      JSONExtractString(metadata, 'objectName')      as objectName,
      JSONExtractString(metadata, 'stageName')       as stageName
FROM actions
WHERE source = 'service'
 AND layer = 'frontend'
 AND action = 'user:clicked:button';
```

Materialized View

A materialized view is a special trigger that stores the result of a SELECT query on data, as it is inserted, into a target table



Data Lake Transformation Product Analytics

Why use Materialized View?

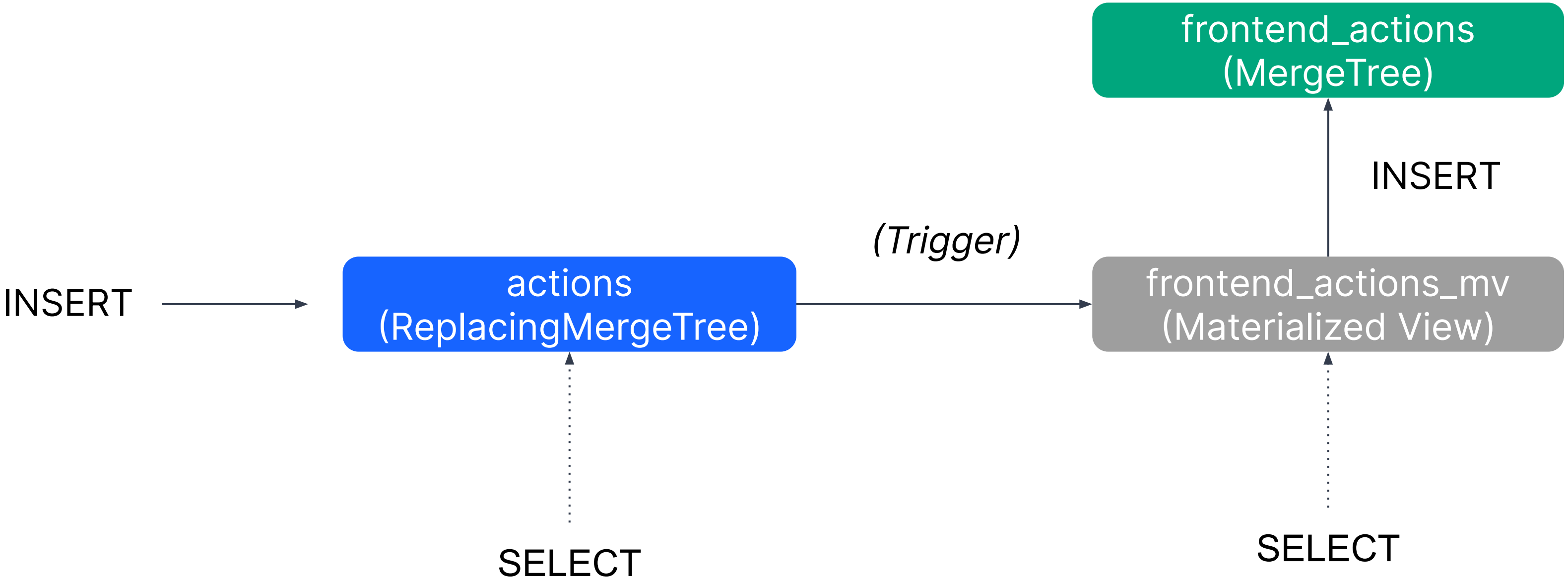
- Democratize the access to data
- Make real-time analytics convenient

- Save time of data engineers
- Make small and readable CH queries
- Answer business questions faster

dayTs	userId	action	metadata
Date	String	String	String

dayTs	userId	action	source	layer	screenName	objectName	stageName
Date	String	String	String	String	String	String	String

Materialized View logic



Creating Materialized View

```
CREATE TABLE actions
(
   `dayTs`     Date,
   `userId`    String,
   `action`    String,
   `metadata`  String
) ENGINE = ReplacingMergeTree()
      PARTITION BY toYYYYMM(dayTs)
      ORDER BY (dayTs, action, userId)
      SAMPLE BY cityHash64(userId)
      SETTINGS index_granularity = 8192;
```

Base table schema
(Data Lake)

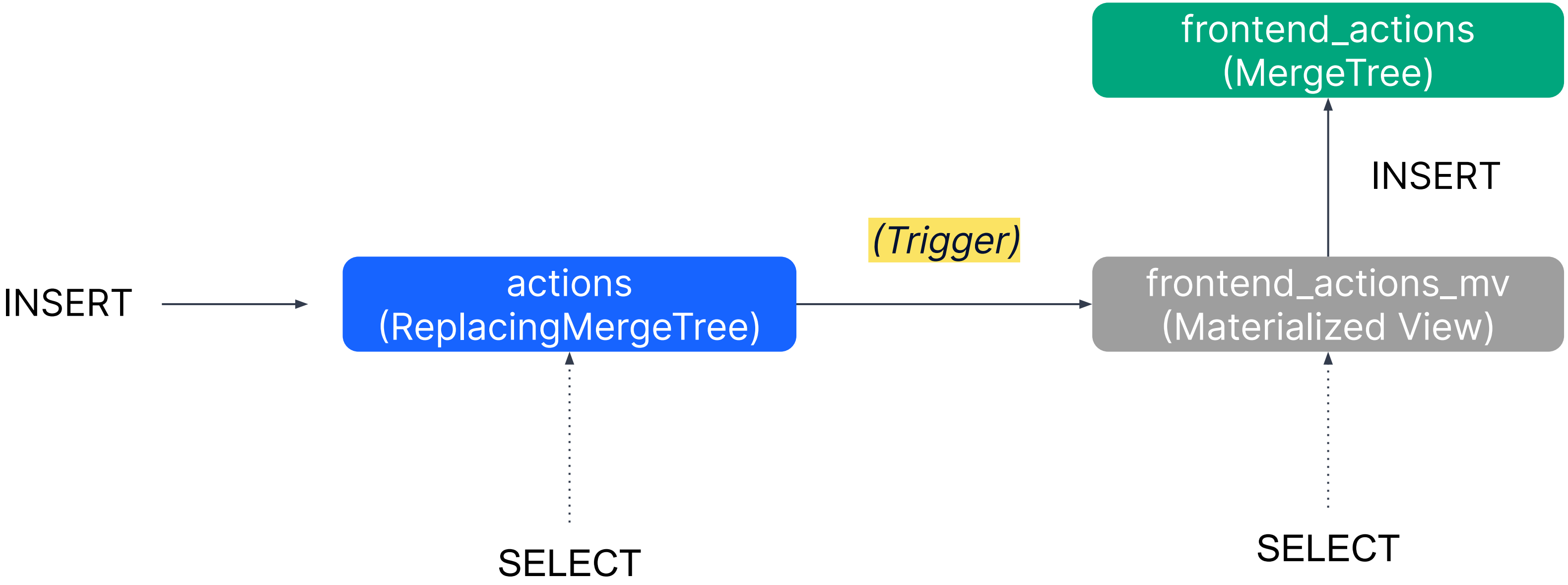
```
CREATE TABLE frontend_actions
(
   `dayTs`        Date,
   `userId`       String,
   `action`       String,
   `source`       String,
   `layer`        String,
   `screenName`   String,
   `objectName`   String,
   `stageName`     String
) ENGINE = ReplicatedMergeTree()
      PARTITION BY toYYYYMM(dayTs)
      ORDER BY (dayTs)
      SETTINGS index_granularity = 8192;
```

Materialized View
schema

Creating Materialized View

```
CREATE MATERIALIZED VIEW frontend_actions_mv TO frontend_actions
AS
SELECT dayTs,
      userId,
      action,
      JSONExtractString(metadata, 'source')          as source,
      JSONExtractString(metadata, 'layer')           as layer,
      JSONExtractString(metadata, 'screenName')      as screenName,
      JSONExtractString(metadata, 'objectName')      as objectName,
      JSONExtractString(metadata, 'stageName')       as stageName
FROM actions
WHERE source = 'service'
 AND layer = 'frontend';
```

Materialized View logic



Insert historical data to Materialized View

```
INSERT INTO frontend_actions
SELECT dayTs,
      userId,
      action,
      JSONExtractString(metadata, 'source')           as source,
      JSONExtractString(metadata, 'layer')            as layer,
      JSONExtractString(metadata, 'screenName')       as screenName,
      JSONExtractString(metadata, 'objectName')       as objectName,
      JSONExtractString(metadata, 'stageName')        as stageName
FROM actions
WHERE source = 'service'
 AND layer = 'frontend'
 AND dayTs >= today()-30;
```

Aggregate

- Requires knowledge and access to Airflow
- Not real-time
- Takes more time to set up and often depends on data engineers

Materialized View

- Do not need Airflow or any cron for inserts
- Requires only a query from analyst
- Real-time
- Could be a smaller table which then can become a part of a larger one
- Takes less memory

Impact of the approach

Time spent by analyst	4 hours → <1 hour
Time of query execution	x2 faster 500 rows retrieved starting from 1 in 8 s 383 ms (execution: 7 s 936 ms, fetching: 447 ms) 500 rows retrieved starting from 1 in 3 s 111 ms (execution: 2 s 769 ms, fetching: 342 ms)
Superset dashboards optimisation	• Faster charts • Single datasource

Showcase: A/B testing results with Aggregates

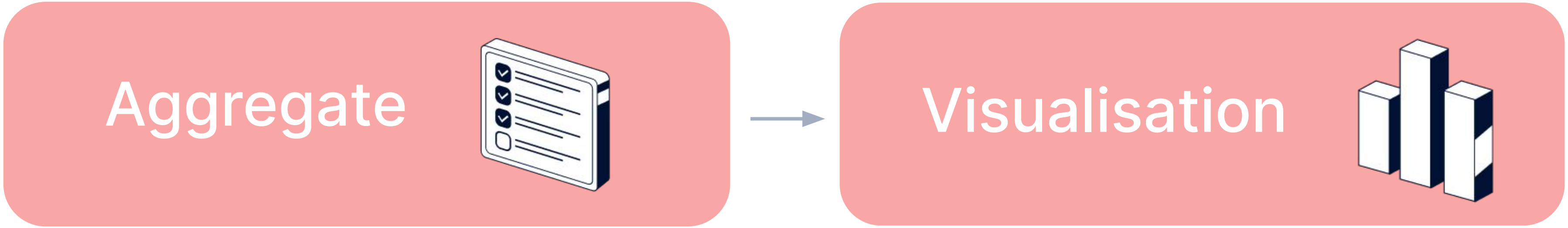
Initial step:

- Analysts aggregate raw data to analyse each experiment
- Calculate same metrics in different ways without synchronization
- Prepare dashboards for each experiment

Product analysts need to:

- Calculate results of A/B tests automatically
- Add new metrics without changing schema every time
- See the results and experimental history in one place

A/B testing results with Aggregates: Solution

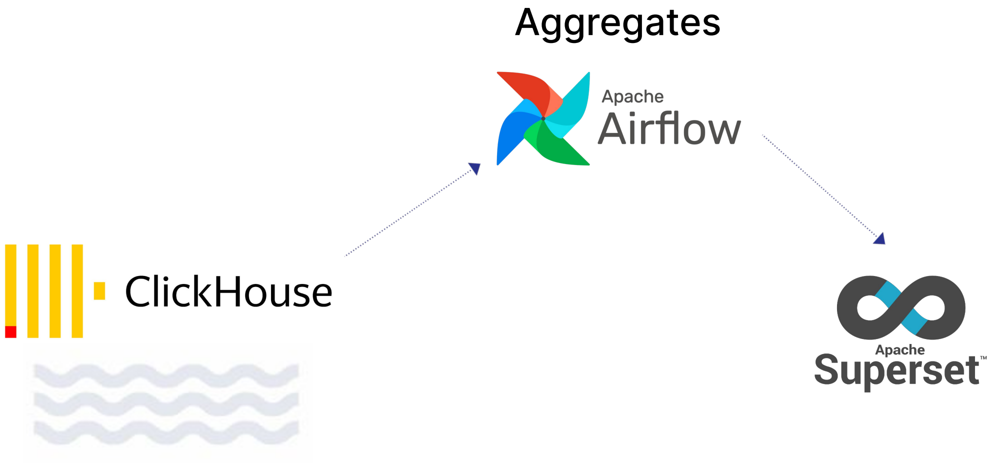


Aggregate

Visualisation

Aggregate

Summarized tables which can be based on several other tables, aggregate functions and other conditions. Inserted to the schema on schedule.



Data Lake

Transformation

Product Analytics

Creating the aggregate with experiment metrics

```
CREATE TABLE experiment_results
(
   dt                DateTime,
   userId            Int64,
   experimentId      Int32,
   experimentalGroup String,
   metricsNames      Array(String),
   metricsValues     Array(UInt64)
) ENGINE = MergeTree()
     PARTITION BY dt
     ORDER BY (dt, userId, experimentId)
     SAMPLE BY cityHash64(userId)
    SETTINGS index_granularity = 8192;
```

Aggregate schema

Creating the aggregate with experiment metrics

```
select dt,
      userId,
      experimentId,
      experimentalGroup,
      ['clicks', 'views'] as metricsNames,
      [clicks, views]     as metricsValues
from (select dt,
            userId,
            experimentId,
            experimentalGroup,
            countIf(event = 'click') as clicks,
            countIf(event = 'view')  as views
     from events
     where dt >= '2023-02-02'
       and experimentId = 1
     group by userId,
              experimentId,
              experimentalGroup,
              dt);
```

You can add metric like clicksMainPage

Query for the schema

Base table

dt	userId	experimentId	experimentalGroup	event
2023-02-02	1234567890	345	control	click
2023-02-02	1234567891	345	test	view
2023-02-02	1234567892	345	control	view

Example code for Airflow DAG

```
from datetime import datetime
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from clickhouse_driver import Client

default_args = {
   'owner': 'airflow',
   'start_date': datetime(2023, 7, 20),
}

dag = DAG('insert_experiment_results' , default_args=default_args, schedule_interval= '0 1 * *
*')

def insert_experiment_results():
   clickhouse_conn = Client( host='your_clickhouse_host' , port='your_clickhouse_port' )
   query = '''query on the next slide'''
   clickhouse_conn.execute( query)


insert_data_task = PythonOperator(
   task_id='insert_data_task' ,
   python_callable=insert_experiment_results,
   dag=dag,
)

insert_data_task
```

Example code for Airflow DAG (query)

```
def insert_experiment_results():
   clickhouse_conn = Client(host='your_clickhouse_host', port='your_clickhouse_port')
   query = '''
       INSERT INTO experiment_results
       (dt, userId, experimentId, experimentalGroup, metricsNames, metricsValues)
       SELECT dt, userId, experimentId, experimentalGroup, ['clicks', 'views'],
[clicks, views]
       FROM (
           SELECT dt, userId, experimentId, experimentalGroup,
               countIf(event = 'click') AS clicks,
               countIf(event = 'view') AS views
           FROM events
           WHERE dt >= '2023-02-02' AND experimentId = 1
           GROUP BY dt, userId, experimentId, experimentalGroup
       )
   '''
   clickhouse_conn.execute(query)
```

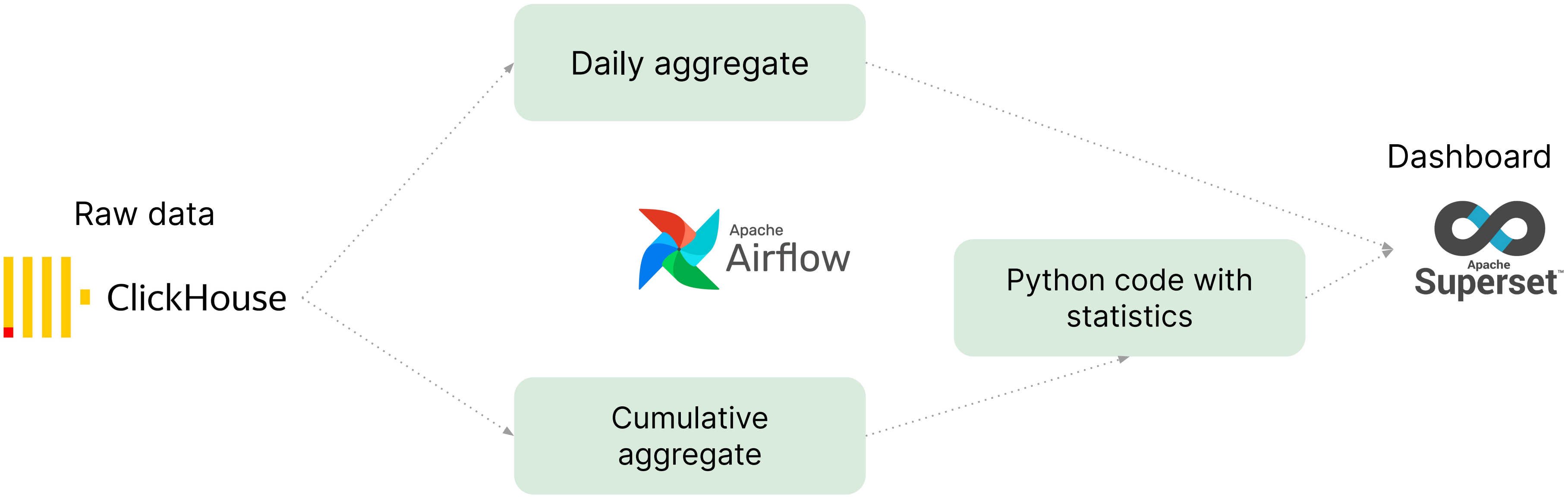
Events in experiment_results

dt	userId	experimentId	experimentalGroup	metricsNames	metricsValues
2023-04-01	1234567890	345	test	[‘clicks’, ‘views’]	[28, 100]
2023-04-01	1234567891	345	control	[‘clicks’, ‘views’]	[10, 90]

```
SELECT dt,
userId,
experimentId,
experimentalGroup,
metricsNames,
metricsValues
FROM experiment_results
WHERE experimentId = ‘345’
ARRAY JOIN  metricsNames,
metricsValues
```

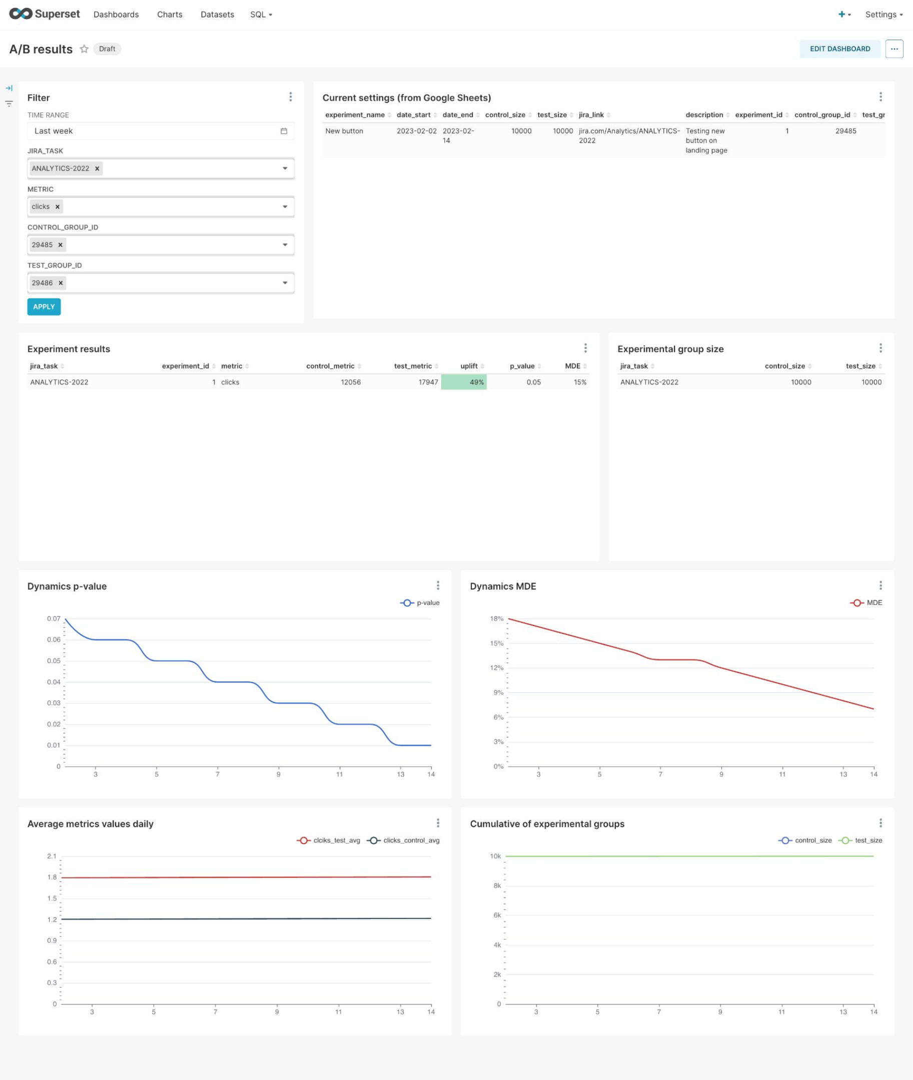
dt	userId	experimentId	experimentalGroup	metricsNames	metricsValues
2023-04-01	1234567890	345	test	clicks	28
2023-04-01	1234567890	345	test	views	100

Overall A/B platform system design



Visualization in Superset

- Description of A/B tests
- Dynamics of metrics and statistics
- Group sizes



A/B results

Filter

TIME RANGE: Last week

JIRA_TASK: ANALYTICS-2022

METRIC: clicks

CONTROL_GROUP_ID: 29485

TEST_GROUP_ID: 29486

APPLY

Current settings (from Google Sheets)

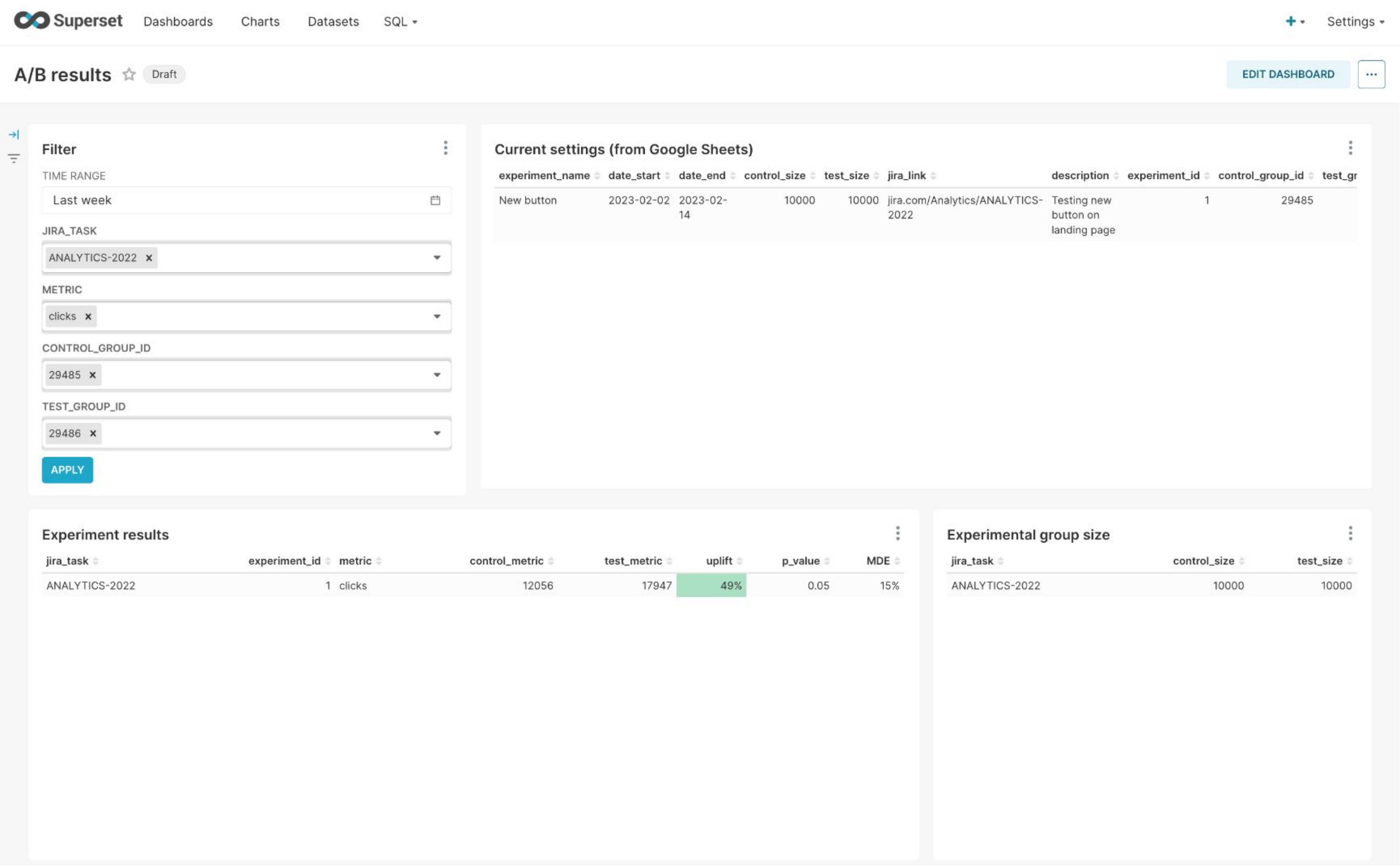
experiment_name	date_start	date_end	control_size	test_size	jira_link	description	experiment_id	control_group_id	test_gr
New button	2023-02-02	2023-02-14	10000	10000	jira.com/Analytics/ANALYTICS-2022	Testing new button on landing page	1	29485	

Experiment results

jira_task	experiment_id	metric	control_metric	test_metric	uplift	p_value	MDE
ANALYTICS-2022	1	clicks	12056	17947	49%	0.05	15%

Experimental group size

jira_task	control_size	test_size
ANALYTICS-2022	10000	10000



A/B results

Draft

Filter

TIME RANGE: Last week

JIRA_TASK: ANALYTICS-2022

METRIC: clicks

CONTROL_GROUP_ID: 29485

TEST_GROUP_ID: 29486

APPLY

Current settings (from Google Sheets)

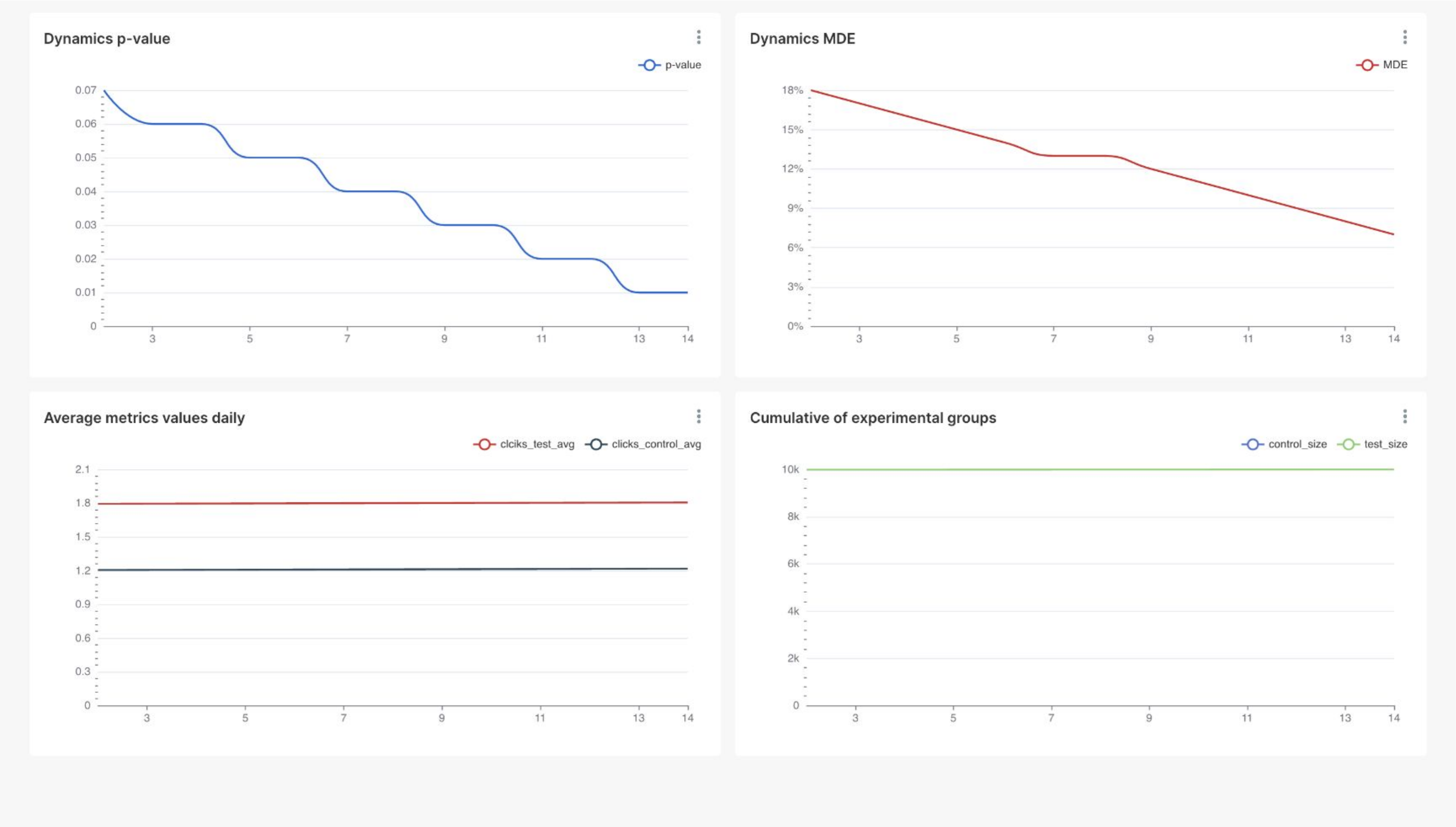
experiment_name	date_start	date_end	control_size	test_size	jira_link	description	experiment_id	control_group_id	test_gr
New button	2023-02-02	2023-02-14	10000	10000	jira.com/Analytics/ANALYTICS-2022	Testing new button on landing page	1	29485	

Experiment results

jira_task	experiment_id	metric	control_metric	test_metric	uplift	p_value	MDE
ANALYTICS-2022	1	clicks	12056	17947	49%	0.05	15%

Experimental group size

jira_task	control_size	test_size
ANALYTICS-2022	10000	10000



Dynamics p-value

Dynamics MDE

Average metrics values daily

Cumulative of experimental groups

Impact of the approach

Historical results of the experiments

Transparency in metrics calculations

Access to unified results for product managers and analysts

Saving time of analytics team

sumsub

Thank you!