One Database, Many Solutions: Uzum's Journey with ClickHouse

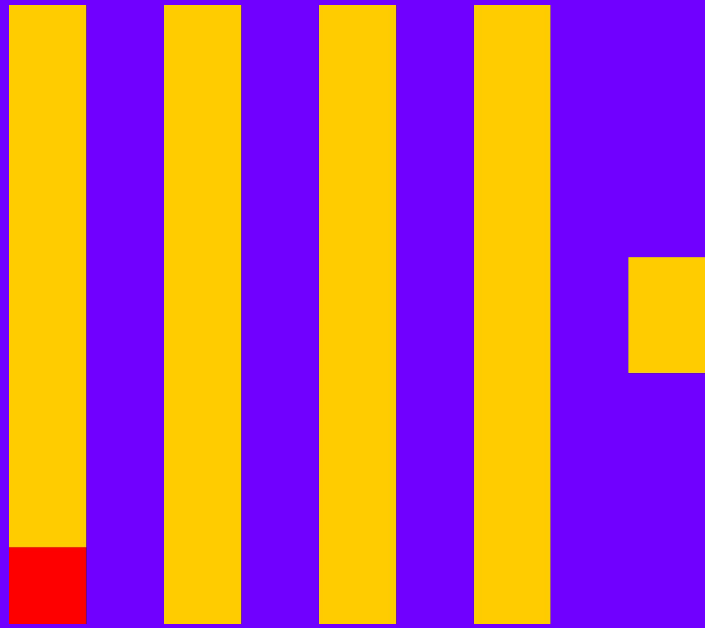
Yuriy Gavrilin for ClickHouse with Grafana meetup / 25.05.2023
CDO Uzum Market & Uzum Bank

Agenda

And the reason to hear me out

1. Introduction and Overview
2. The genesis of ClickHouse @ Uzum
3. Realization about ClickHouse
4. Overall Architecture
5. Conclusion and Q&A

Clickhouse Introduction



Uzum Introduction

We are UZUM

Nation’s #1
Marketplace

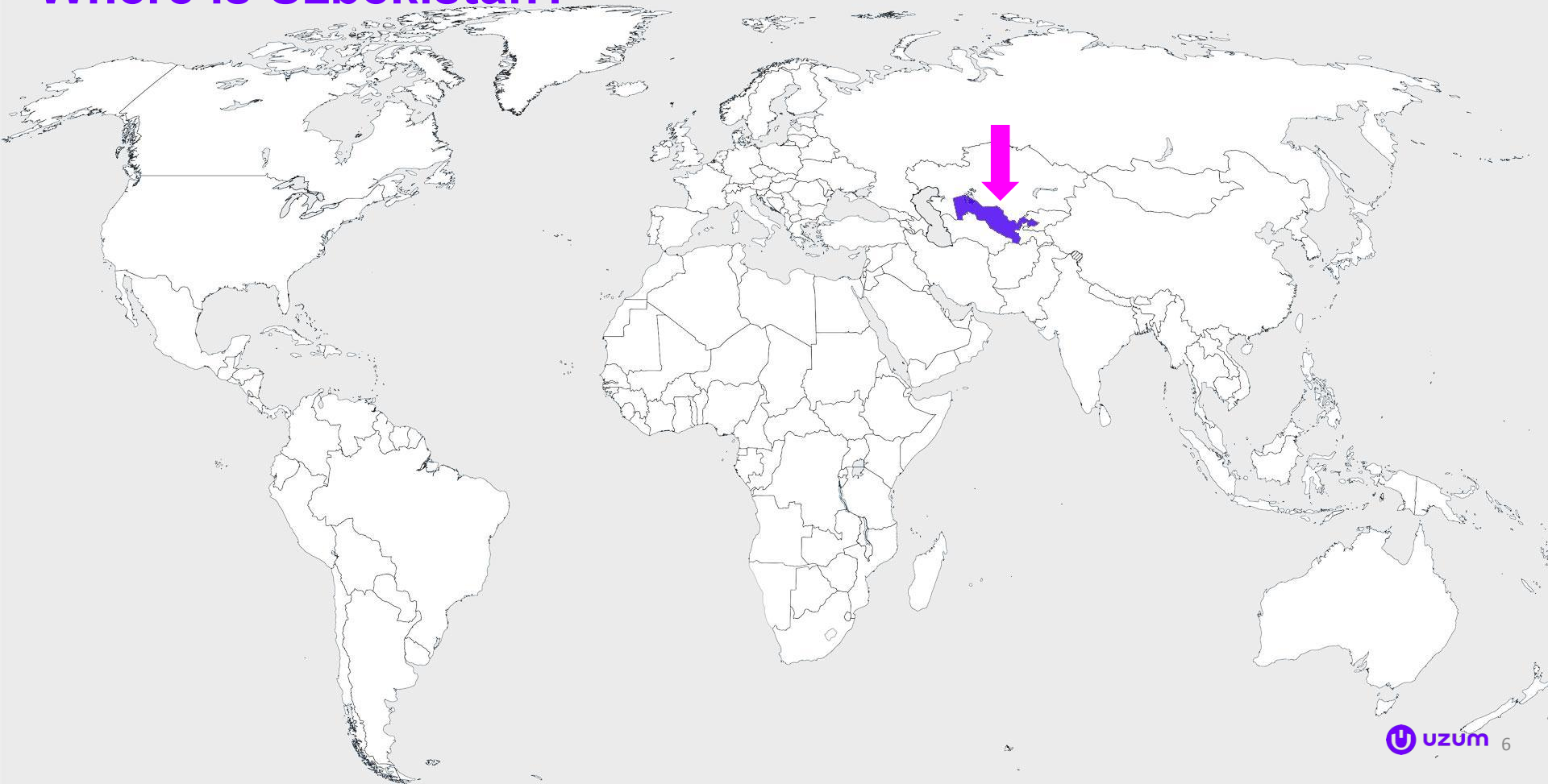
Halal BNPL Service

Digital Bank and
#1 P2P Service

Food Delivery
Service

And We Are Building The Future of Uzbek e-Commerce

Where is Uzbekistan?



Uzum Introduction

More than 350k SKUs available in Uzum Market now and >800 will be at the end of the year

KapitalBank has more than 1 500 000 active individuals

Uzum Market has more than 140 delivery points across 35 cities and >500 will be at the end of the year

Uzum Analytics team is more than 50 data specialists

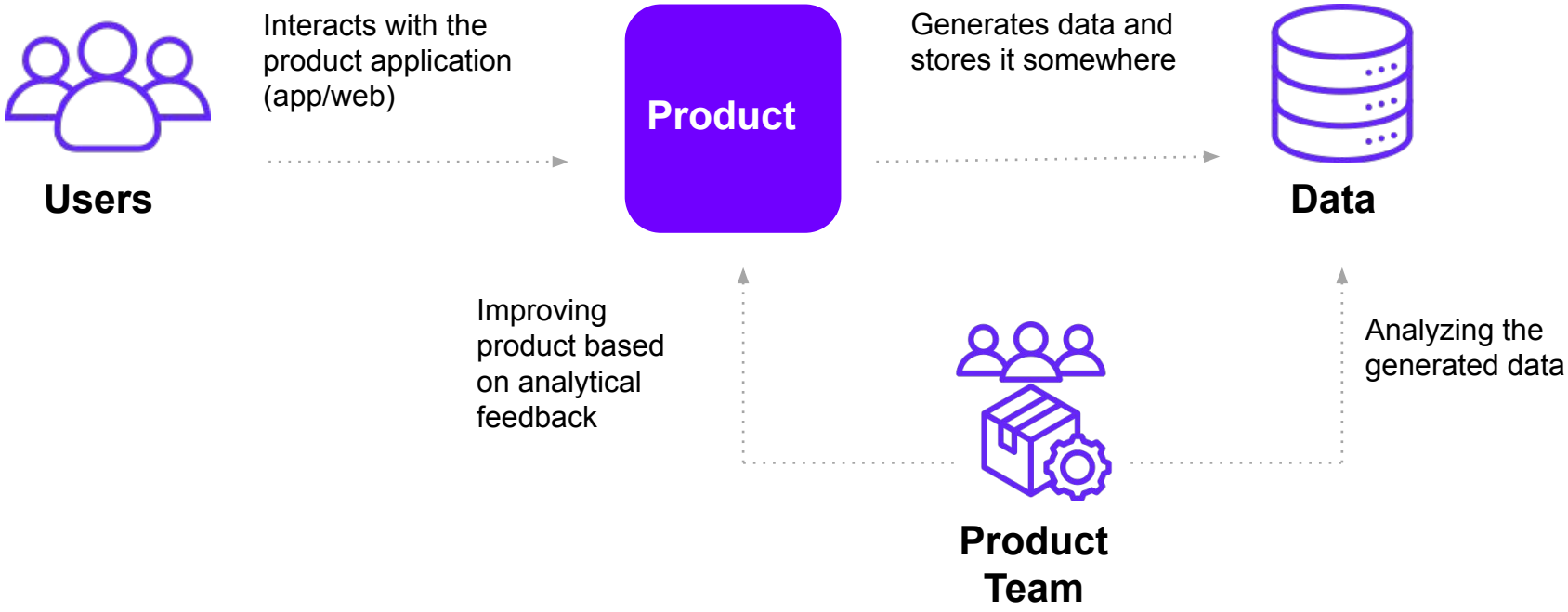
Uzum Ecosystem has more than 5 mln active users

The Genesis of ClickHouse at Uzum

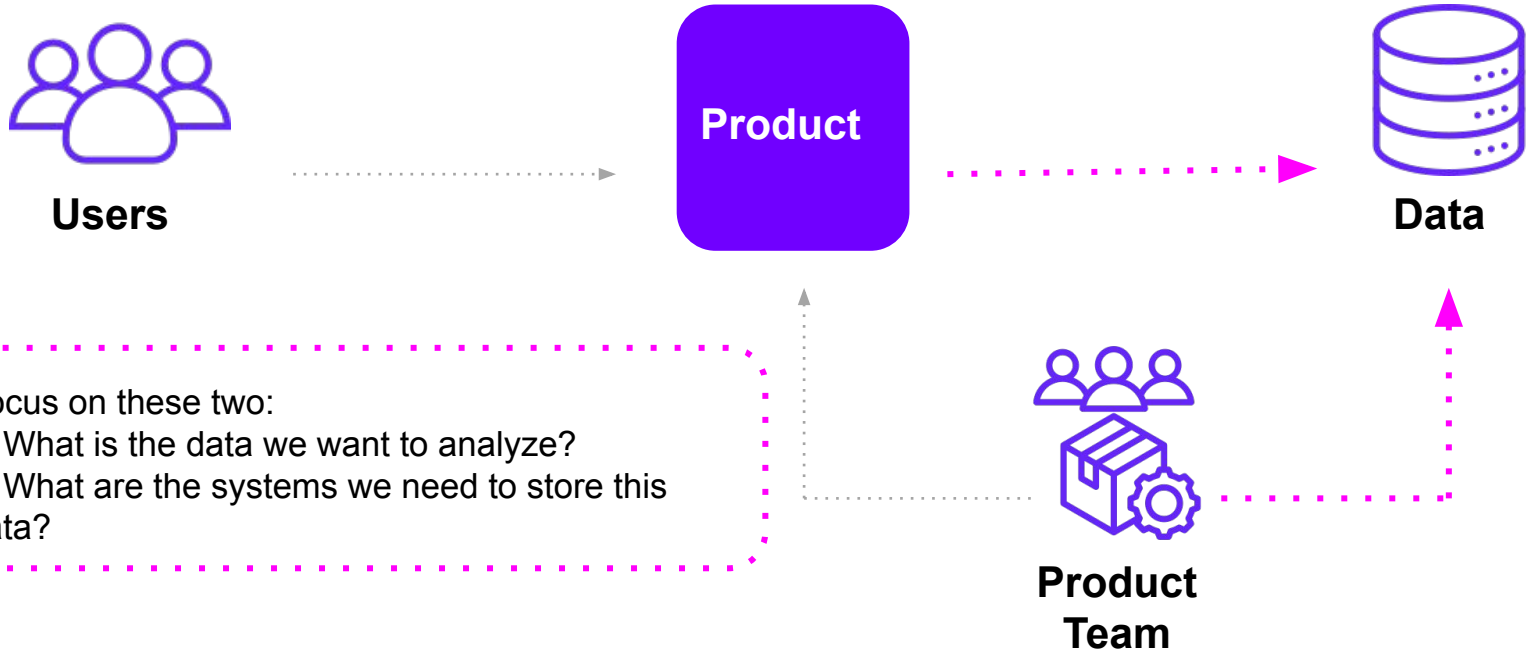
~~The Genesis of Clickhouse at Uzum~~

The Genesis of **Product Analytics** at Uzum

The Genesis of Product Analytics at Uzum



The Genesis of Product Analytics at Uzum



Types of generated data

Product

Product generates many types of data which are needed to be stored and analyzed using different systems and methods

- Operational Data (stored in Transactional Storages like PostgreSQL, MySQL, Oracle)
- Clickstream Data (Product interactions, clicks, etc)
- Service Logs (API requests, text logs)
- Cold Operational Data (stored in cold storage like S3)
- other forms and types of data not or rarely used for analysis

Analytics usually involve multiple data sources and generally read a lot of data

Product Team

Let’s elaborate:

Operational Data (stored in Transactional Storages like PostgreSQL, MySQL, Oracle)

Clickstream Data (Product interactions, clicks, etc)

Service Logs (API requests, text logs)

Cold Operational Data (stored in cold storage like S3)

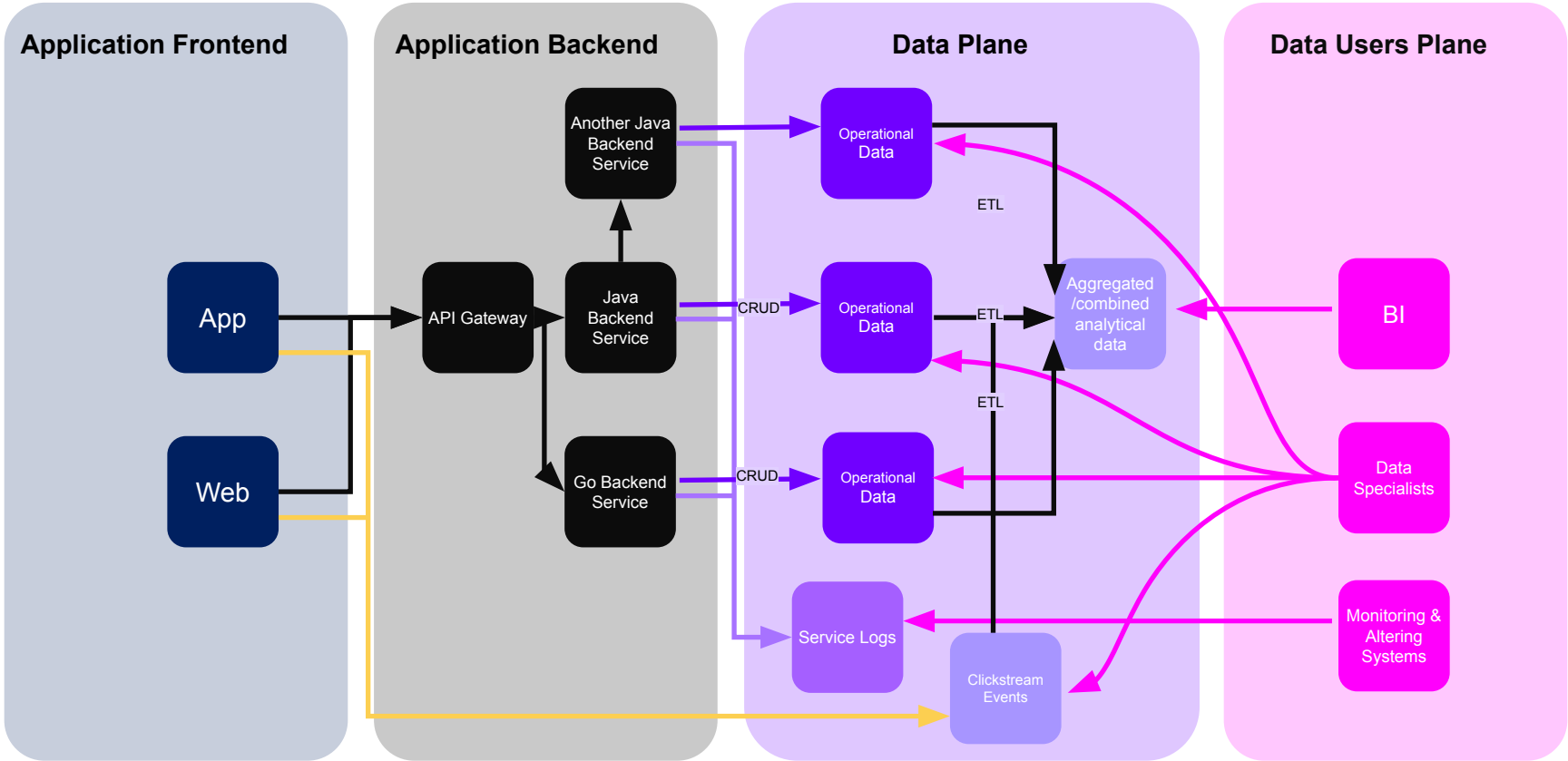
other forms and types of data not or rarely used for analysis

Analysts (ad-hocs & researche

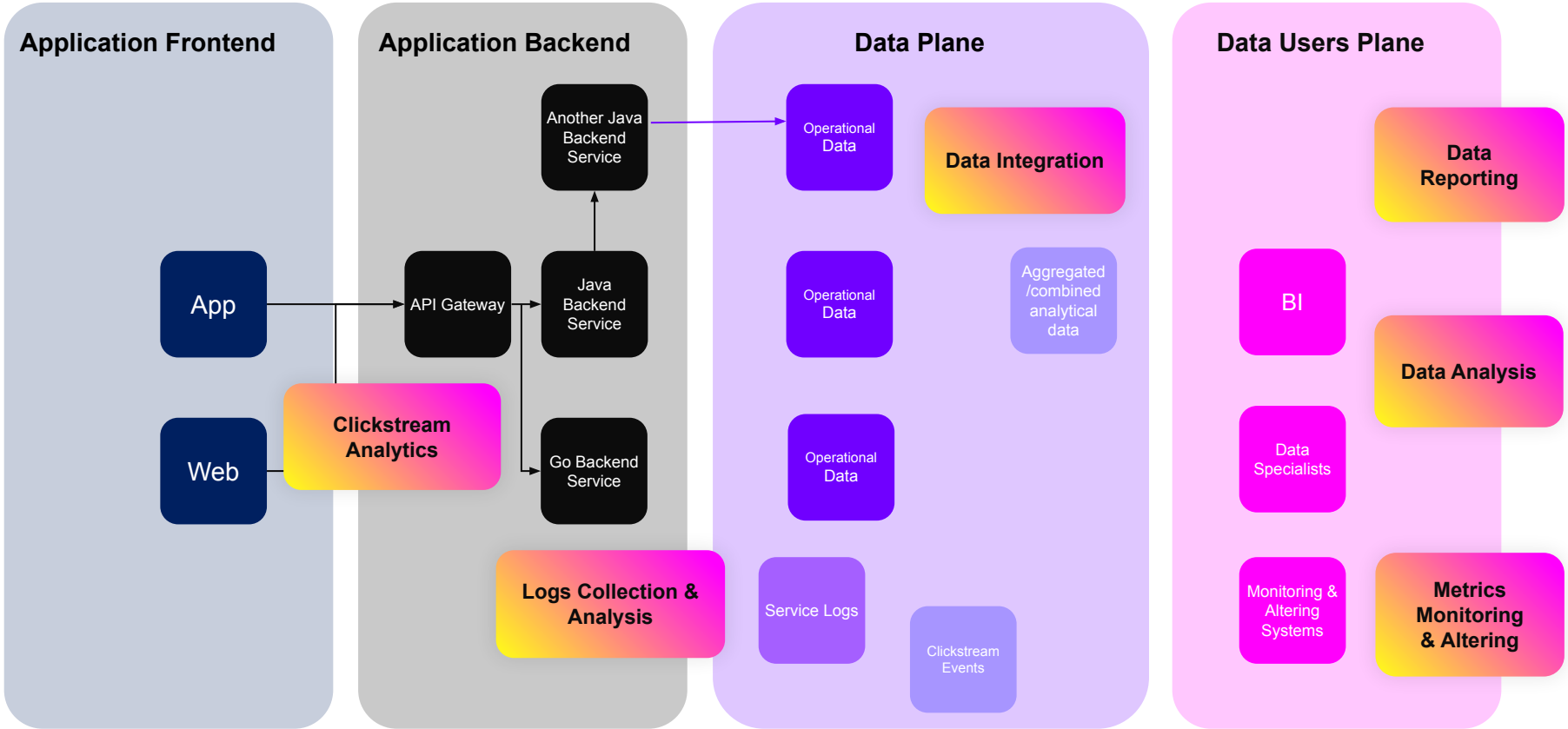
BI Tools for reporting and etc.

OLAP queries from backend services

Monitoring & Alerting


Application Frontend
Application Backend
Data Plane
Data Users Plane
App
Web
API Gateway
Another Java Backend Service
Java Backend Service
Go Backend Service
CRUD
CRUD
Operational Data
Operational Data
Operational Data
ETL
ETL
ETL
Aggregated /combined analytical data
Service Logs
Clickstream Events
BI
Data Specialists
Monitoring & Altering Systems

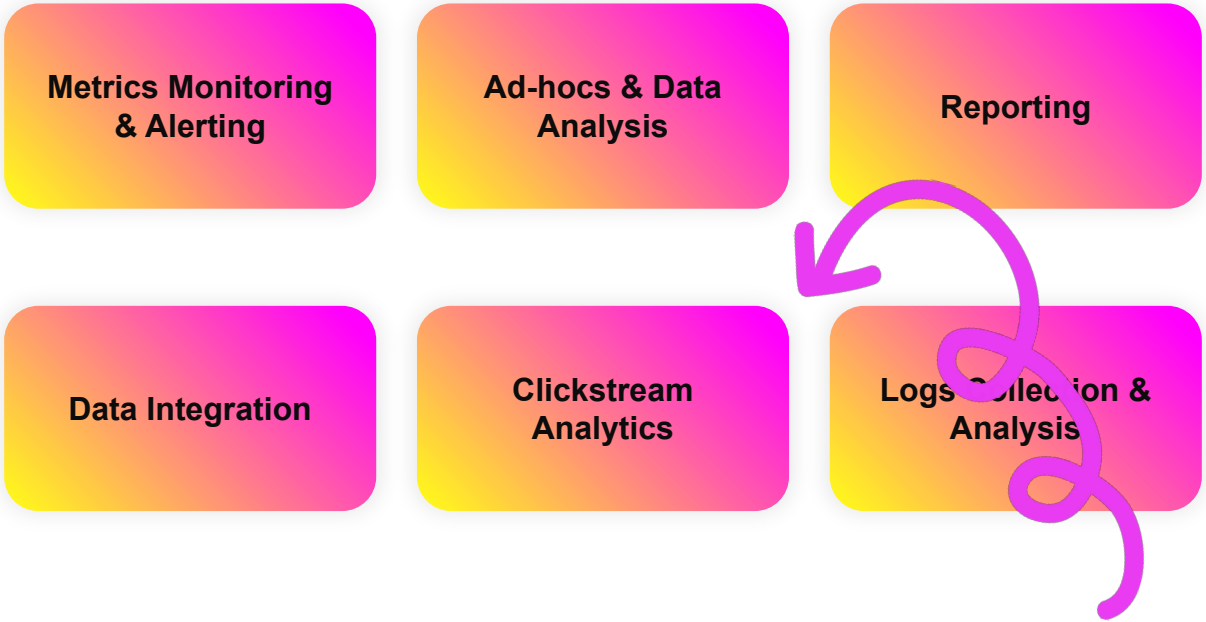
Use cases introduced



User cases we identified

- Metrics Monitoring & Alerting
- Ad-hocs & Data Analysis
- Reporting
- Data Integration
- Clickstream Analytics
- Logs Collection & Analysis

User cases we identified

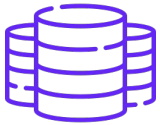


Why ClickHouse for Clickstream Processing?

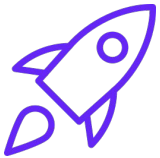
Why ClickHouse for Clickstream Processing?

Scalability

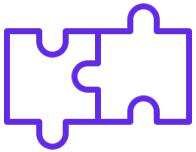
Real-Time Analysis



Efficient Storage



Fast



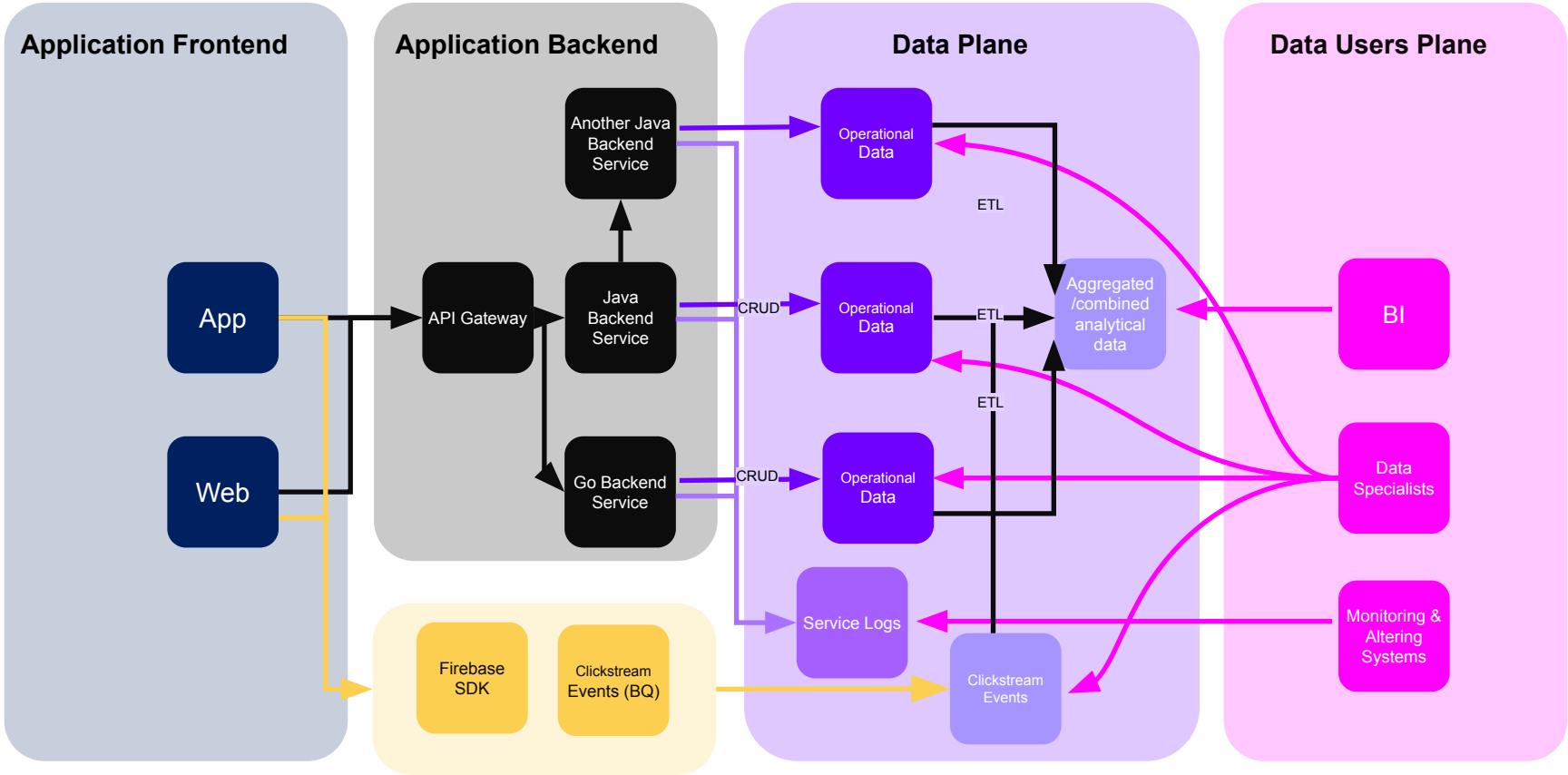
Integration Capabilities

Zero to Billion
Clicks in no* time!

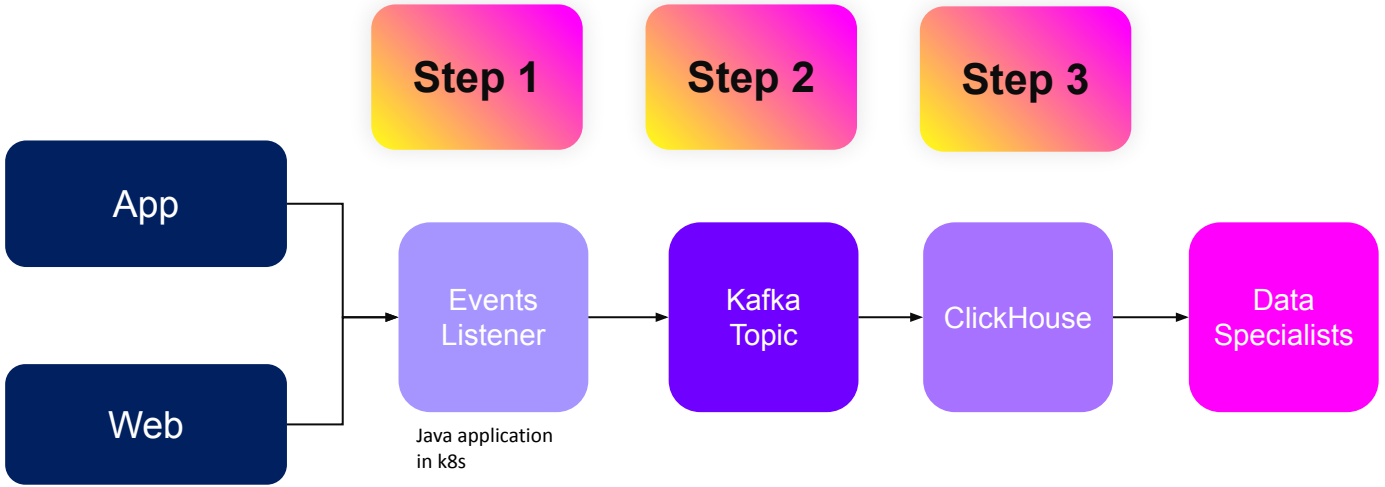
*In a few days

How would you do event analytics?

Let’s start from the baseline



How we did it (oversimplified, but in the same spirit)



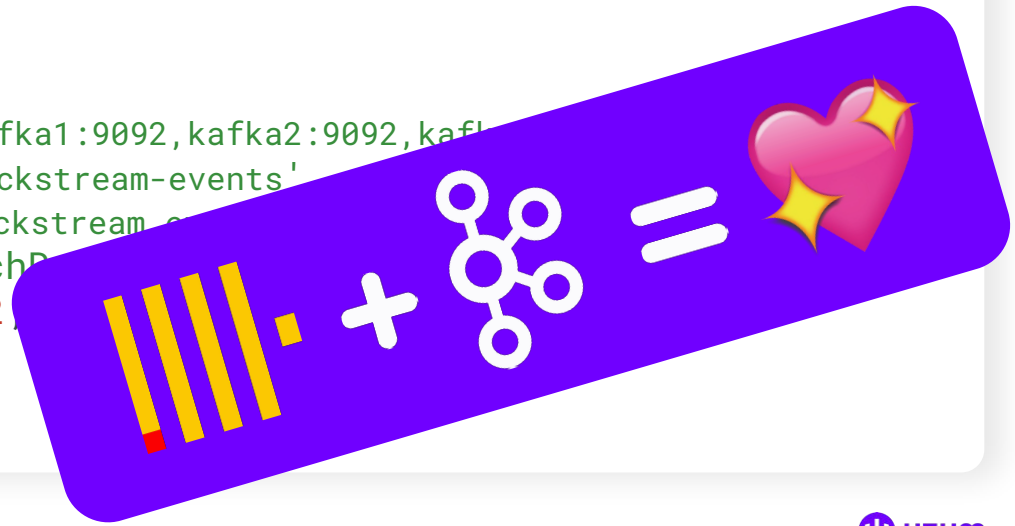
Step 1 – Build a webserver

```python
from flask import Flask, request, jsonify
from kafka import KafkaProducer
import json

app = Flask(__name__)

# Initialize Kafka producer
producer = KafkaProducer(bootstrap_servers='localhost:9092', value_serializer=lambda v: json.dumps(v).encode('utf-8'))

@app.route('/events', methods=['POST'])
def events_listener():

    # Produce the events to Kafka topic
    for event in request.get_json():
        producer.send('clickstream-events', event)

    return jsonify({"message": "Events received and produced to Kafka topic"}), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

***Just ask ChatGPT to do it:** Could you please provide a Python script that initiates a Flask web server? This server should have a single POST endpoint, "/events", which accepts a batch of events and forwards them to a Kafka topic named "clickstream-events". t:

Step 2 read data from Kafka

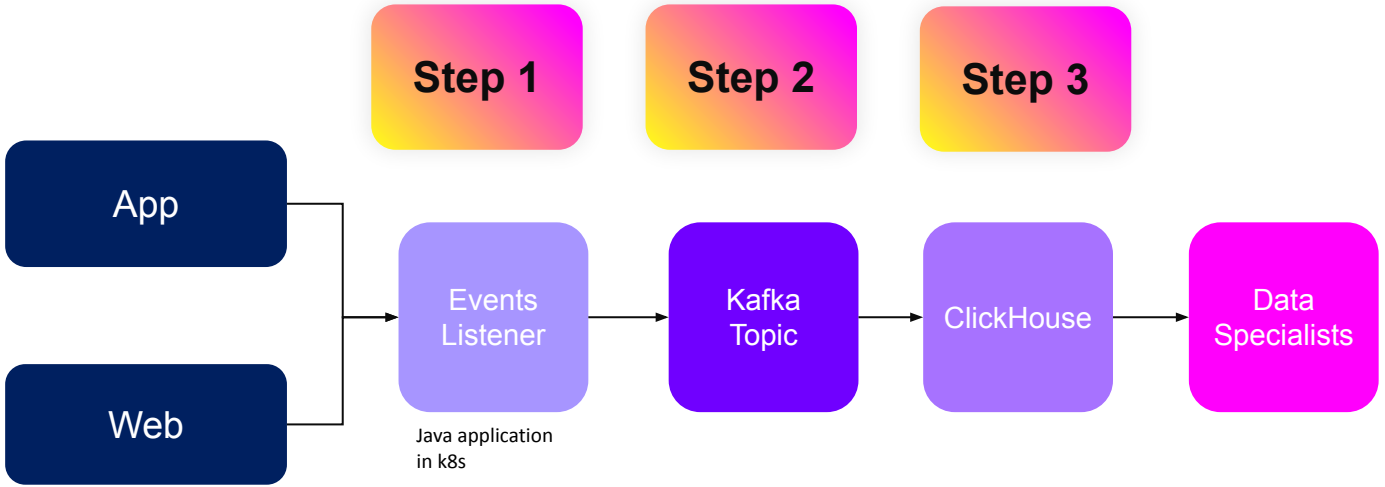
```
CREATE TABLE clickstream_events_kafka
(
    event_date Date,
    event_type String,
    user_id Int32,
    page_id Int32,
    session_id Int32
) ENGINE = Kafka
SETTINGS kafka_broker_list = 'kafka1:9092,kafka2:9092,kaf
         kafka_topic_list = 'clickstream-events'
         kafka_group_name = 'clickstream_c
         kafka_format = 'JSONEachR
         kafka_num_consumers = 2
```



Step 2 / part 2 read data from Kafka

```
CREATE TABLE clickstream_events
(
    event_date Date,
    event_type String,
    user_id Int32,
    page_id Int32,
    session_id Int32
) ENGINE = MergeTree()
ORDER BY (event_date, event_type, user_id);

CREATE MATERIALIZED VIEW clickstream_events_mv TO clickstream_events AS
SELECT *
FROM clickstream_events_kafka;
```

Few lines and you have



>10 000 000 000 events monthly with access in under 4 seconds!

Is it really this simple?

Can ClickHouse save us?

How ClickHouse handles all that kind of stuff?

```
CREATE TABLE clickstream.events
(
    device_id UUID,
    session_id UUID,
    account_id UInt64,
    logged_at DateTime('UTC') CODEC(Delta(4), LZ4),
    ipv4 IPv4,
    …, -- Magic is happening here
    device_properties` String CODEC(ZSTD(5)),
    platform LowCardinality(String),
    event_type LowCardinality(String);
)
ENGINE = ReplicatedReplacingMergeTree('/clickhouse/tables/{shard}/clickstream/events', '{replica}')
PARTITION BY toYYYYMMDD(received_at)
ORDER BY (platform, event_type, logged_at, event_id)
TTL received_at + toIntervalDay(60) TO DISK 'object_storage'
```

Replicate data to reach access SLA

```
CREATE TABLE clickstream_b2c.events
(
    device_id UUID,
    session_id UUID,
    account_id UInt64,
    logged_at DateTime('UTC') CODEC(Delta(4), LZ4),
    ipv4 IPv4,
    …, -- Magic is happening here
    device_properties` String CODEC(ZSTD(5)),
    platform LowCardinality(String),
    event_type LowCardinality(String);
)
ENGINE = ReplicatedReplacingMergeTree('/clickhouse/tables/{shard}/clickstream/events', '{replica}')
PARTITION BY toYYYYMMDD(received_at)
ORDER BY (platform, event_type, received_at)
TTL received_at + toIntervalDay(60) TO DISK 'object_storage'
```

Remove duplicates automatically

```
CREATE TABLE clickstream_b2c.events
(
    device_id UUID,
    session_id UUID,
    account_id UInt64,
    logged_at DateTime('UTC') CODEC(Delta(4), LZ4),
    ipv4 IPv4,
    …, -- Magic is happening here
    device_properties` String CODEC(ZSTD(5)),
    platform LowCardinality(String),
    event_type LowCardinality(String);
)
ENGINE = ReplicatedReplacingMergeTree('/clickhouse/tables/{shard}/clickstream/events', '{replica}')
PARTITION BY toYYYYMMDD(received_at)
ORDER BY (platform, event_type, received_at, event_id)
TTL received_at + toIntervalDay(60) TO DISK 'object_storage'
```

Special Data Types

```
CREATE TABLE clickstream_b2c.events
(
    device_id UUID,
    session_id UUID,
    event_id UUID,
    account_id UInt64,
    logged_at DateTime('UTC') CODEC(Delta(4), LZ4),
    ipv4 IPv4,
    ab Nested (experiment UInt64, variation UInt64),
    …, -- Magic is happening here
    device_properties` String CODEC(ZSTD(5)),
    platform LowCardinality(String),
    event_type LowCardinality(String);
)
ENGINE = ReplicatedReplacingMergeTree('/clickhouse/tables/{shard}/clickstream/events', '{replica}')
PARTITION BY toYYYYMMDD(received_at)
ORDER BY (platform, event_type, received_at, event_id)
TTL received_at + toIntervalDay(60) TO DISK 'object_storage'
```

Codecs to optimize storage and read time

```
CREATE TABLE clickstream_b2c.events
(
    device_id UUID,
    session_id UUID,
    event_id UUID,
    seller_id UInt64 CODEC(T64, LZ
    logged_a                    UTC'),
    ipv4 IPv
    …, -- Ma                    ning here
    device_properties` String CODEC(ZSTD(5)),
    platform LowCardinality(String),
    event_type LowCardinality(String);
)
ENGINE = ReplicatedReplacingMergeTree('/clickhouse/tables/{shard}/clickstream/events', '{replica}')
PARTITION BY toYYYYMMDD(received_at)
ORDER BY (platform, event_type, received_at, event_id)
TTL received_at + toIntervalDay(60) TO DISK 'object_storage'
```

Compression: 200x

Compression: 30x

Compression: 200x

Move old and rarely used data to S3 without hassle

```
CREATE TABLE clickstream_b2c.events
(
    device_id UUID,
    session_id UUID,
    event_id UUID,
    seller_id UInt64 CODEC(T64, LZ4),
    logged_at DateTime('UTC'),
    ipv4 IPv4,
    …,
    device_properties` String CODEC(ZSTD(5)),
    platform LowCardinality(String),
    event_type LowCardinality(String);
)
ENGINE = ReplicatedReplacingMergeTree('/clickhouse/table                tream/events', '{replica}')
PARTITION BY toYYYYMMDD(received_at)
ORDER BY (platform, event_type, received_at, event_id)
TTL received_at + toIntervalDay(60) TO DISK 'object_storage'
```

Cold storage:
20+ Tb

Frequent queries can be optimized through projections/indices

```
CREATE TABLE clickstream_b2c.events
(
    device_id UUID,
    session_id UUID,
    event_id UUID,
    seller_id UInt64 CODEC(T64, LZ4),
    logged_at DateTime('UTC'),
    ipv4 IPv4,
    …,
    device_properties` String CODEC(ZSTD(5)),
    platform LowCardinality(String),
    event_type LowCardinality(String),
    PROJECTION users_by_hour (SELECT toStartOfHour(logged_at) as  hour, uniq(device_id) GROUP BY hour),
    NDEX app_version_numerical app_version_num TYPE(minmax) GRANULARITY 1
)
ENGINE = ReplicatedReplacingMergeTree('/clickhouse/tables/{shard}/clickstream/events', '{replica}')
PARTITION BY toYYYYMMDD(received_at)
ORDER BY (platform, event_type, received_at, event_id)
TTL received_at + toIntervalDay(60) TO DISK 'object_storage'
```

What happened?

With a few tweaks Click house have handled:

1. Large data volume via Codecs, data types and Hybrid storage
2. Data Quality (some of it) via Replacing- Engine
3. Data Access SLA via Replicated- Engine
4. Fast data access via Projections

Let’s revisit:

Logs Collection & Analysis

Ad-hoc Analysis

Reporting

Data Integration

Clickstream Analytics

Metrics Monitoring & Alerting

Let’s revisit:

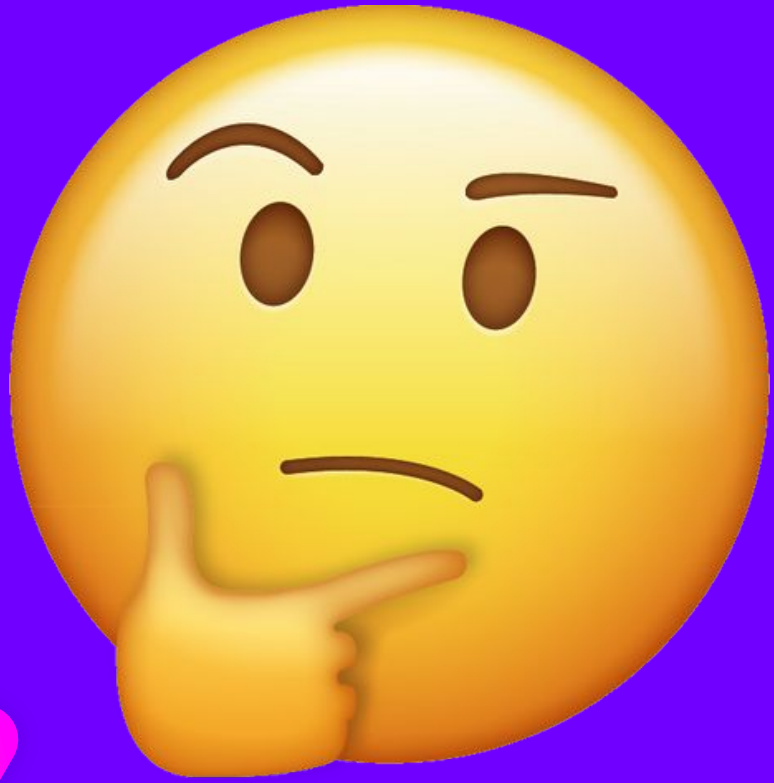
Metrics Monitoring & Alerting

Logs Collection & Analysis

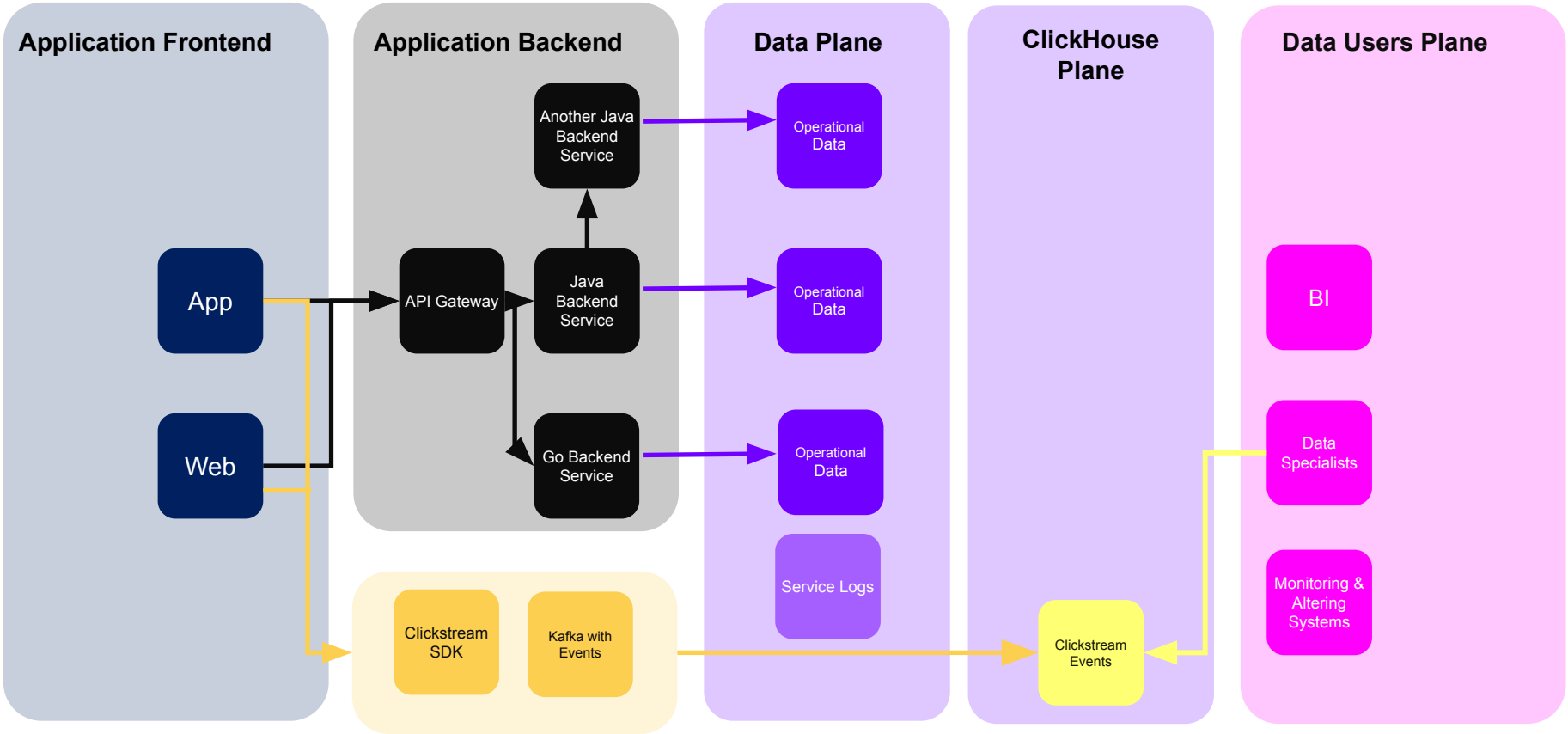
Reporting

Ad-hoc Analysis

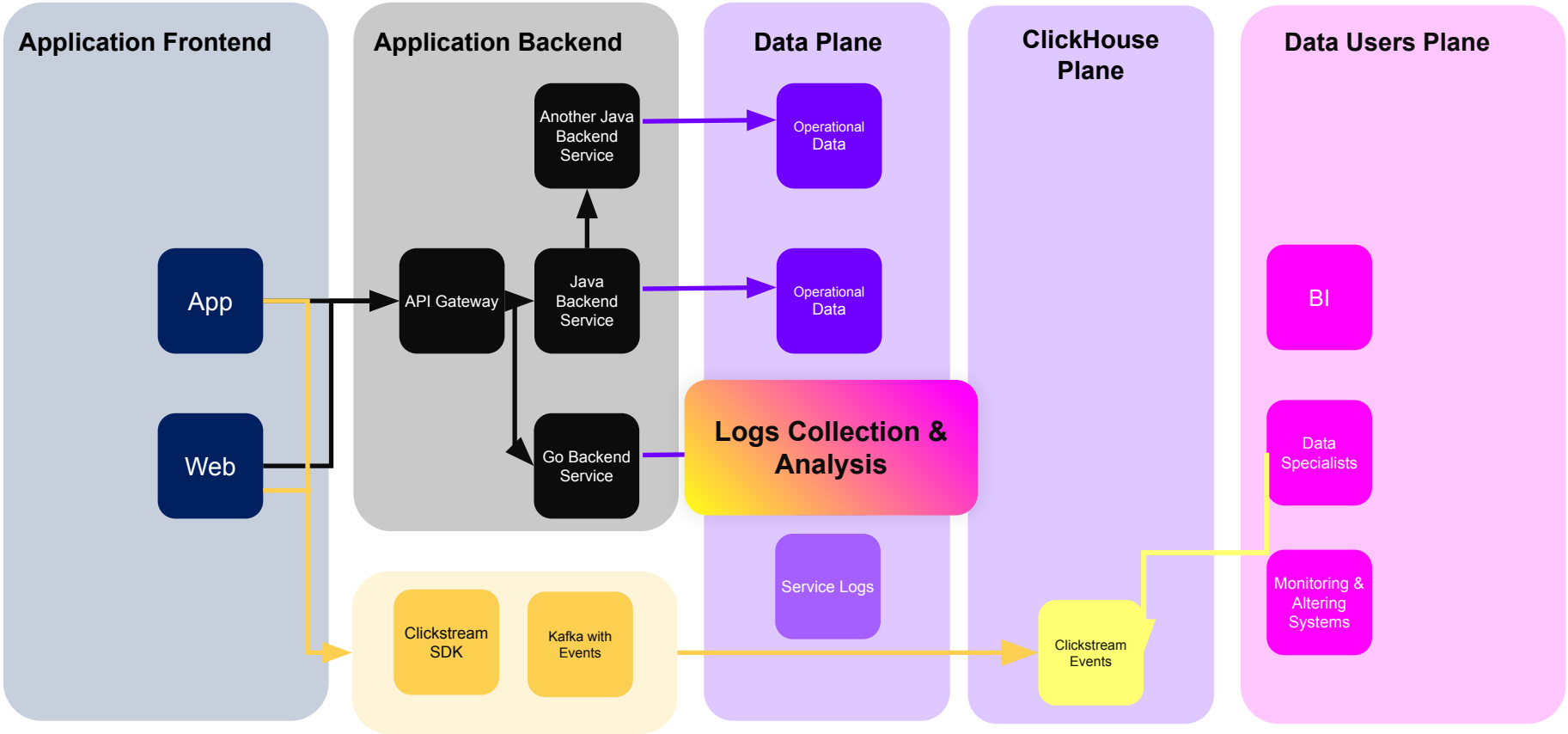
Data Integration



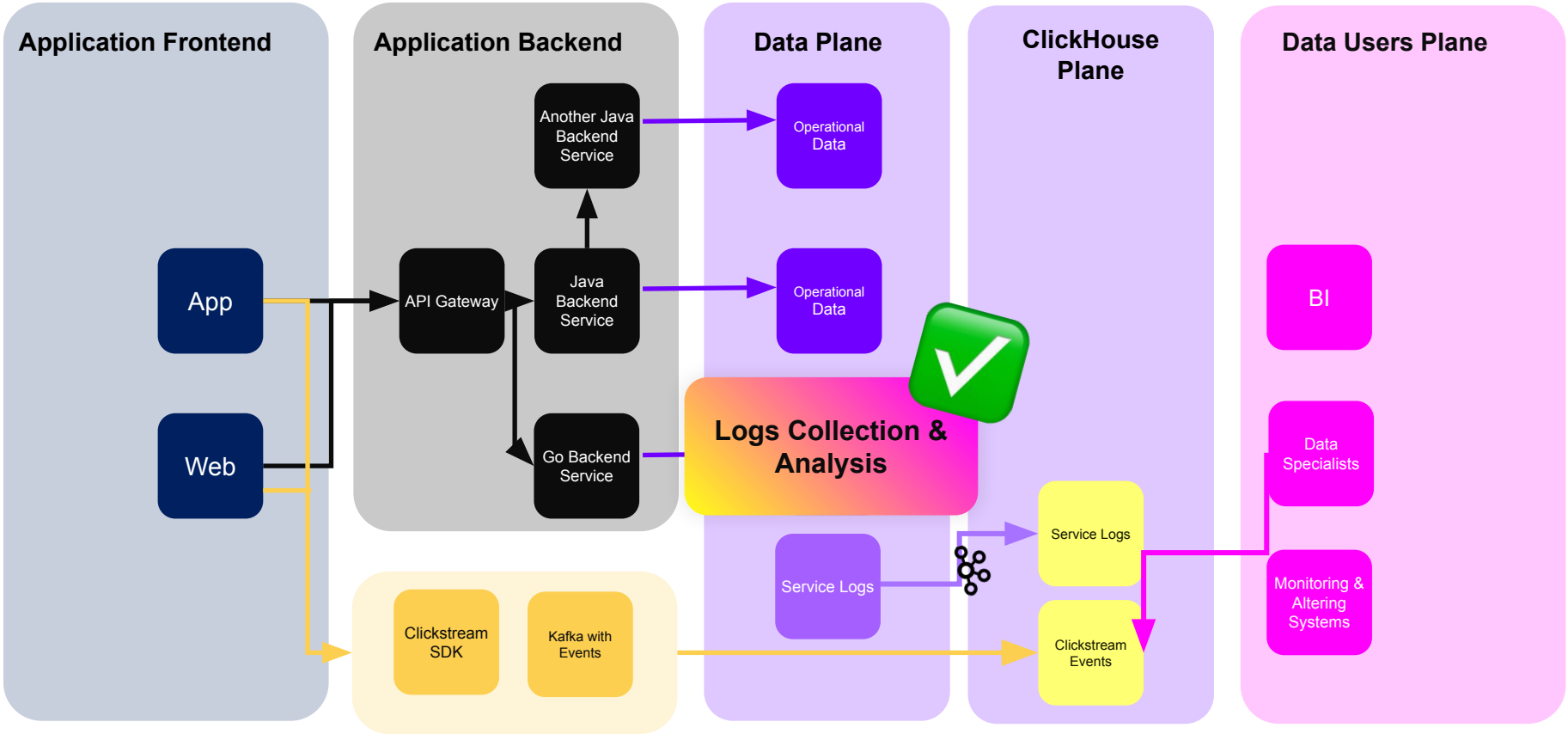
Let’s revisit again…



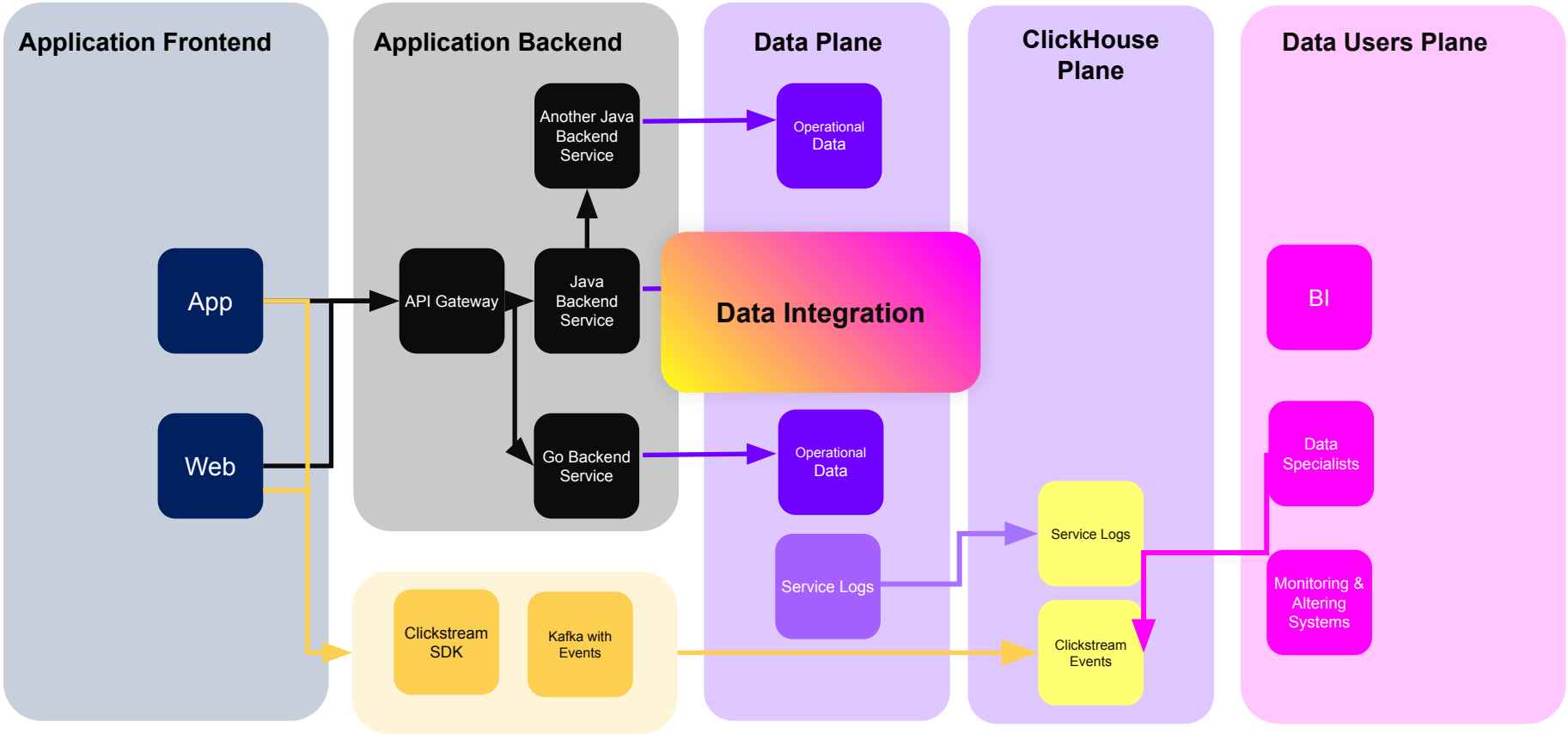
Let’s revisit again…



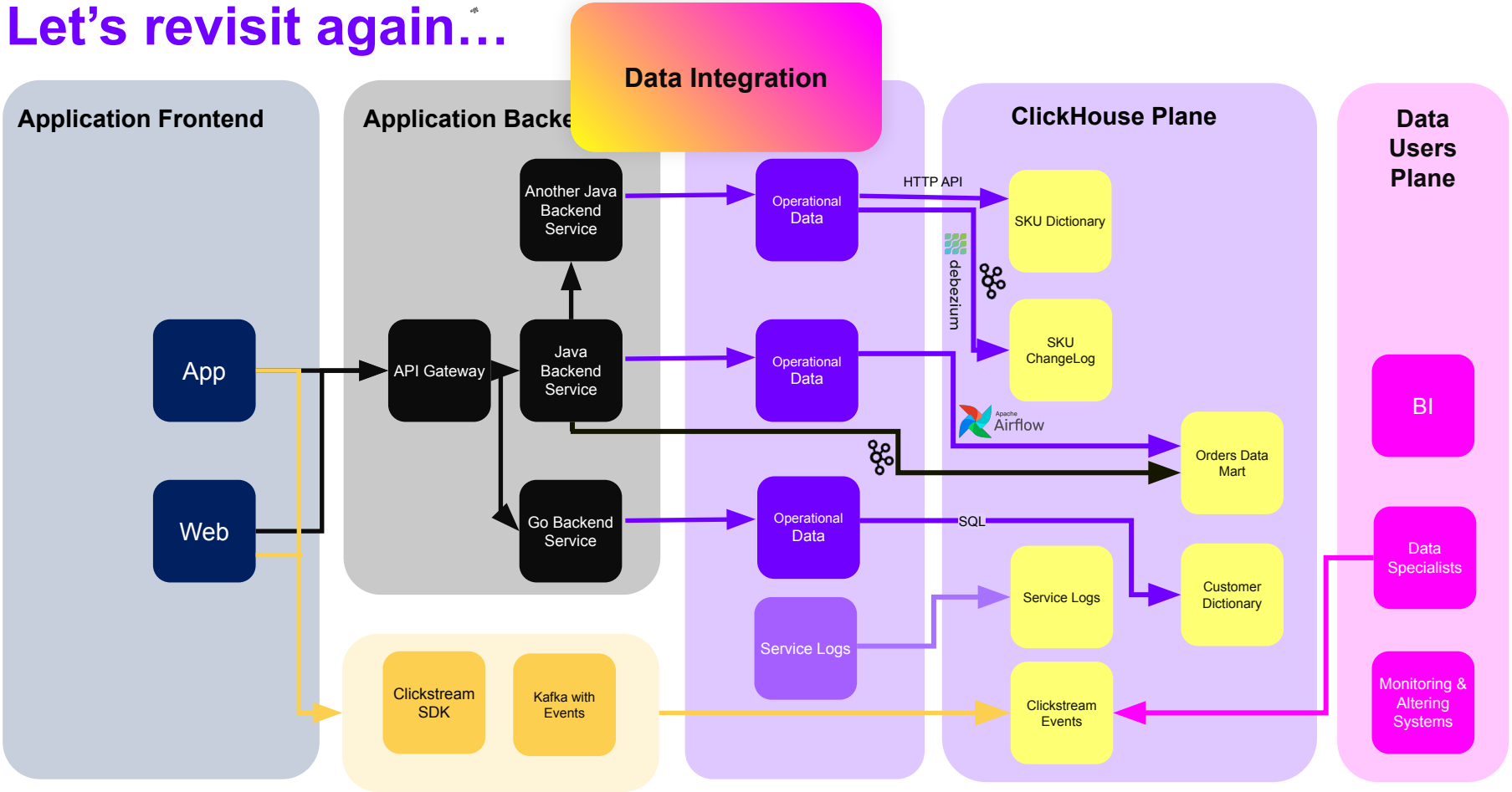
Let’s revisit again…



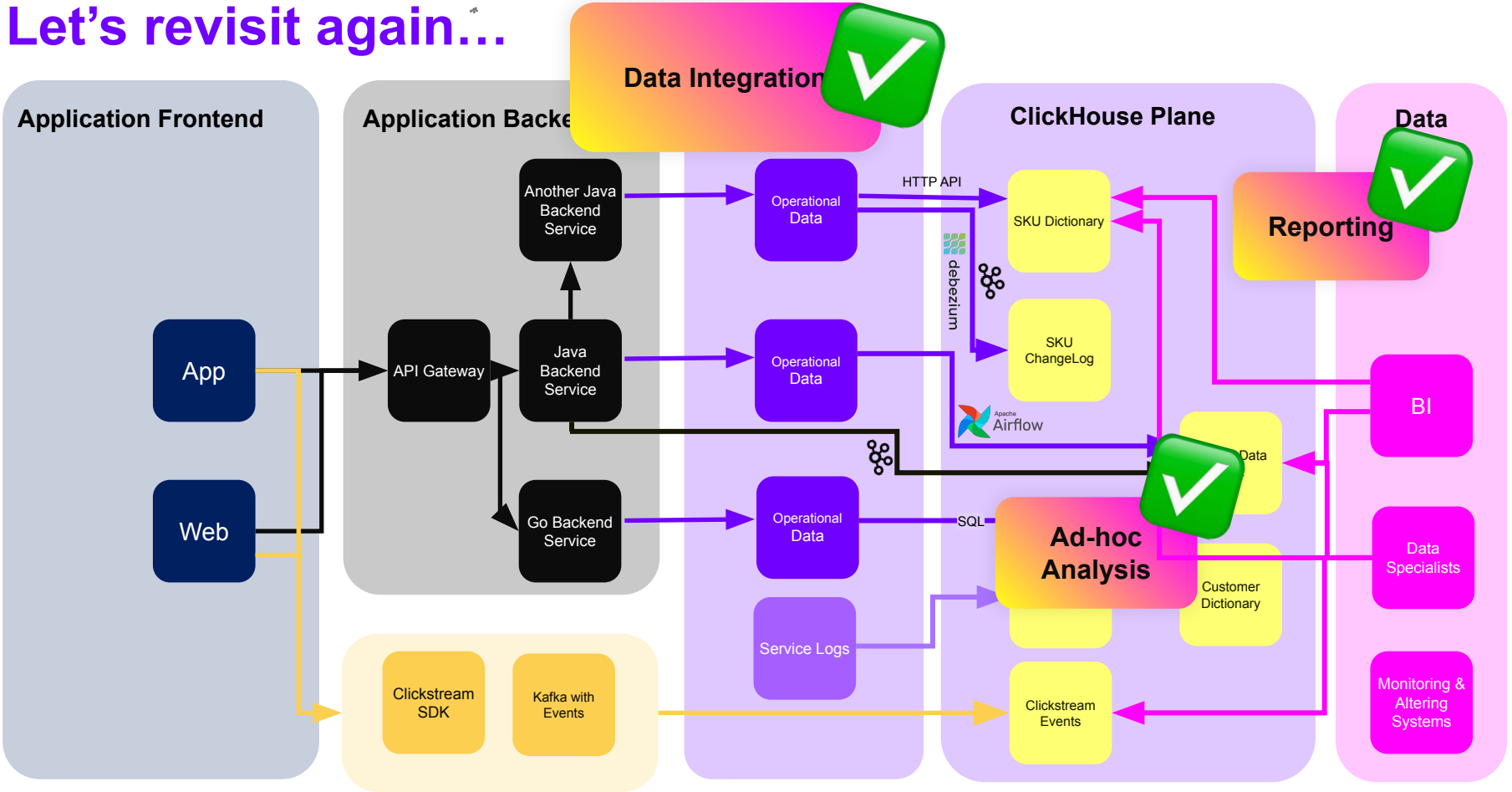
Let’s revisit again…



Let’s revisit again…



Let’s revisit again…



What's left?

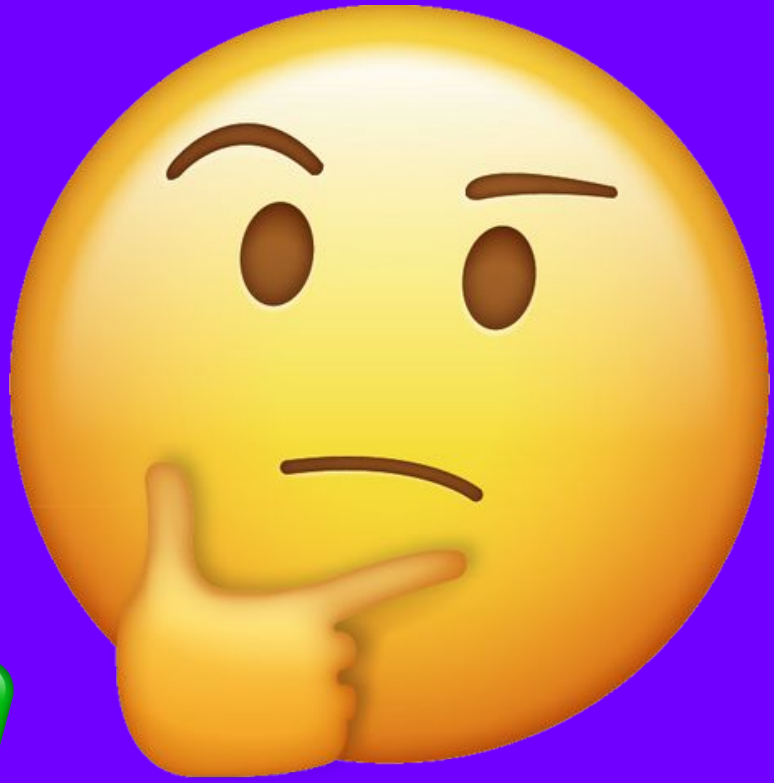
Metrics Monitoring & Alerting

Logs Collection Analysis

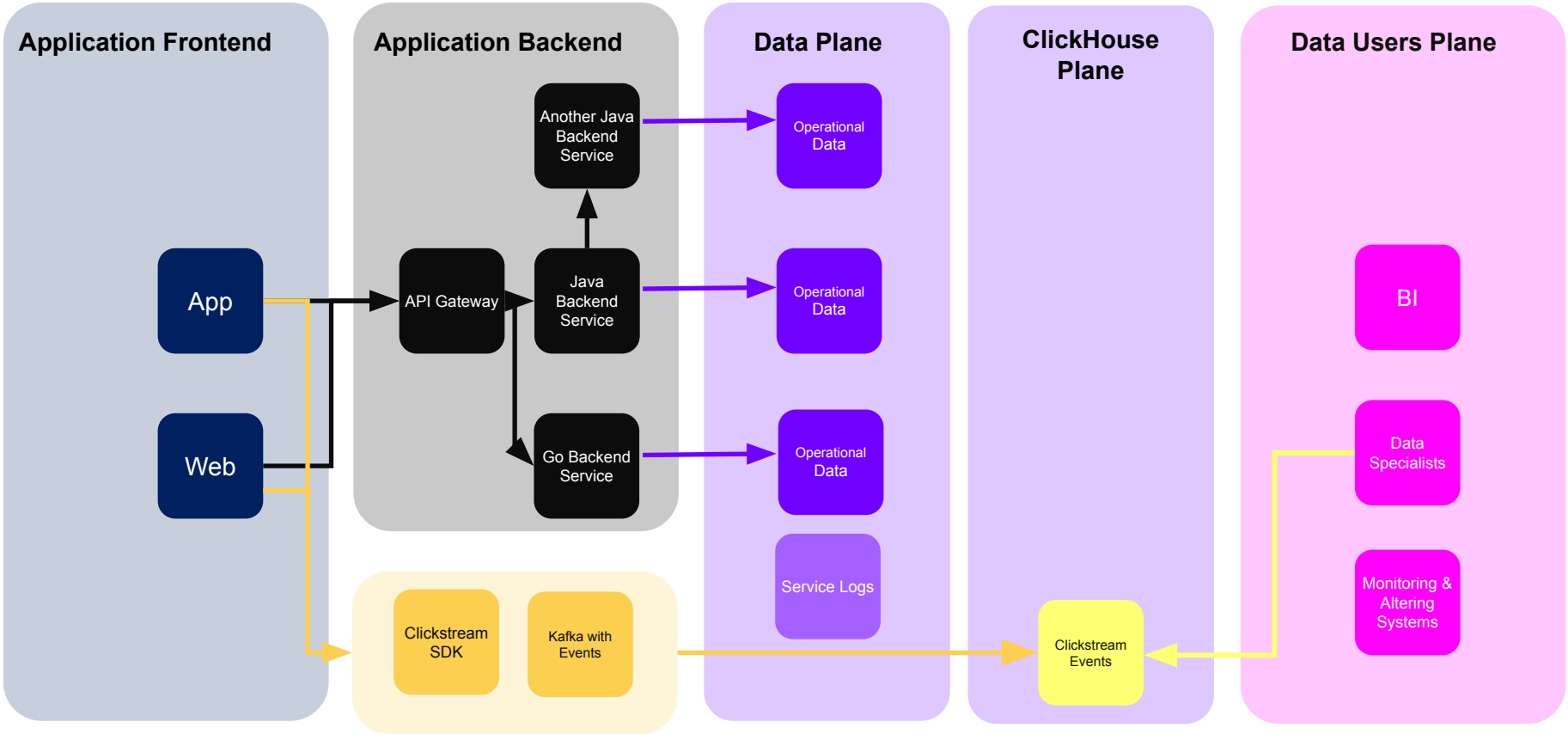
Reporting

Ad-hoc Analysis

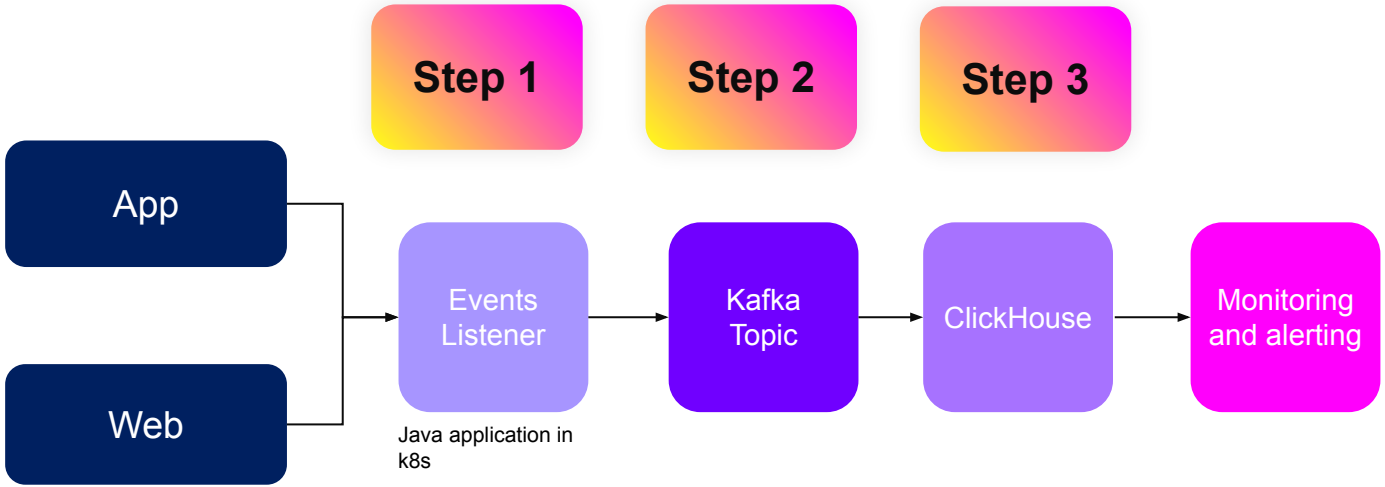
Data Integration



I'll erase some stuff...



How we did it:


Step 1
Step 2
Step 3
App
Web
Events Listener
Java application in k8s
Kafka Topic
ClickHouse
Monitoring and alerting

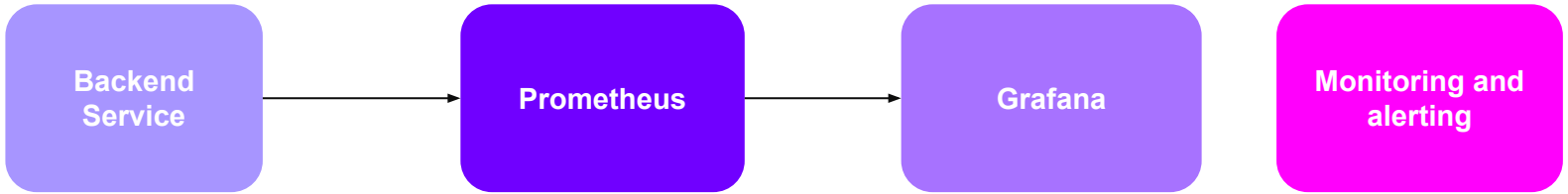
~~How we did it:~~

ClickHouse

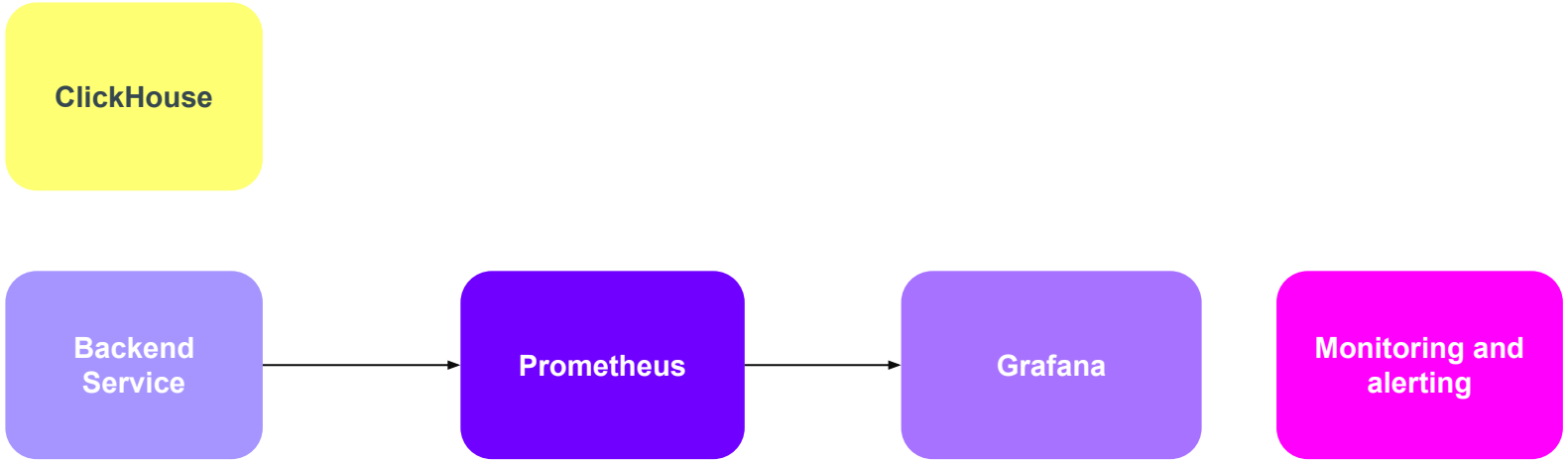
Monitoring and alerting

How do you do monitoring?

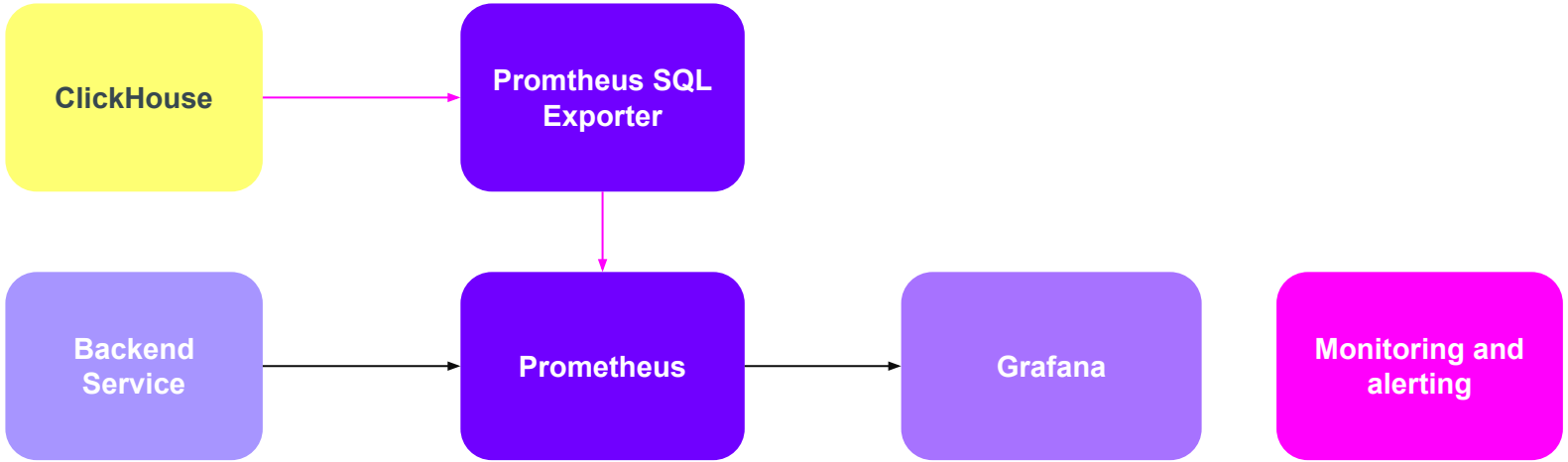
Usually it’s something like this



But not all of our data can come from Backend

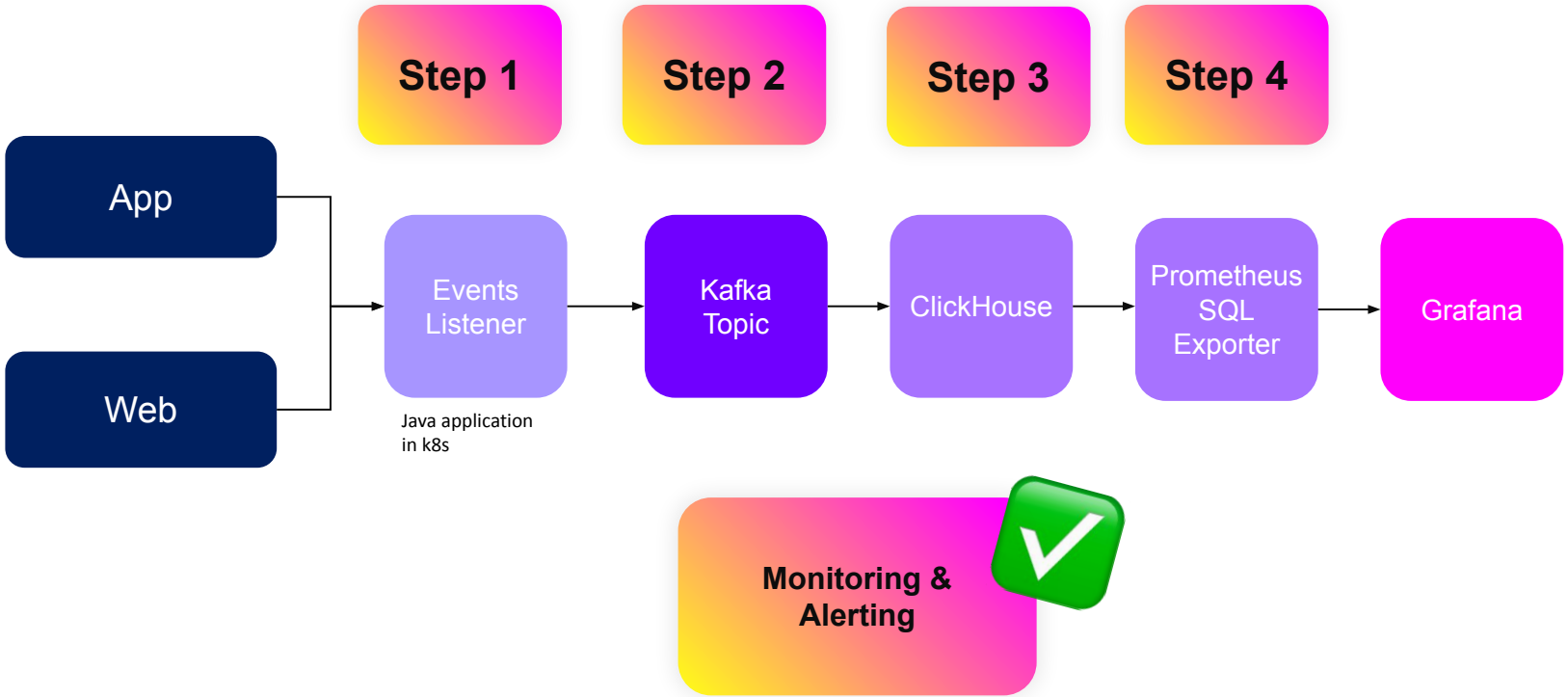


But not all of our data can come from Backend




```
global:
  scrape_timeout_offset: 500ms
target:
  data_source_name: 'clickhouse://host:port?username=user&password=passw&database=clickstream'
collectors: [active_users]
collector_files:
   - "*.collector.yml"
```

```
collector_name: active_users

metrics:
  - metric_name: active_users
    type: gauge
    help: 'Number of active users per minute.'
    values: [users]
    query: |
      SELECT toStartOfMinute(logged_at) m, uniq(device_id) AS users
      FROM clickstream.events
      GROUP BY m
```


Step 1
Step 2
Step 3
Step 4
App
Web
Events Listener
Java application in k8s
Kafka Topic
ClickHouse
Prometheus SQL Exporter
Grafana
Monitoring & Alerting

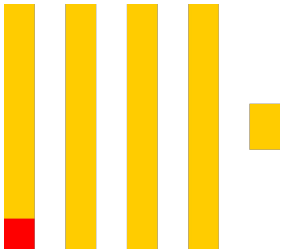

Metrics Monitoring & Alerting
Logs Collection Analysis
Reporting
Ad-hoc Analysis
Data Integration

Architecture recap – ClickHouse POV

Architecture recap – ClickHouse POV

Data Specialists

Clickstream



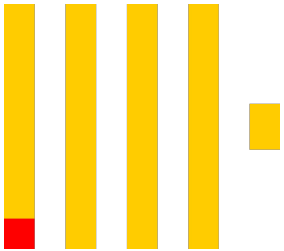
Events Log Table

Architecture recap – ClickHouse POV

Data Specialists

Clickstream

Backend Services API



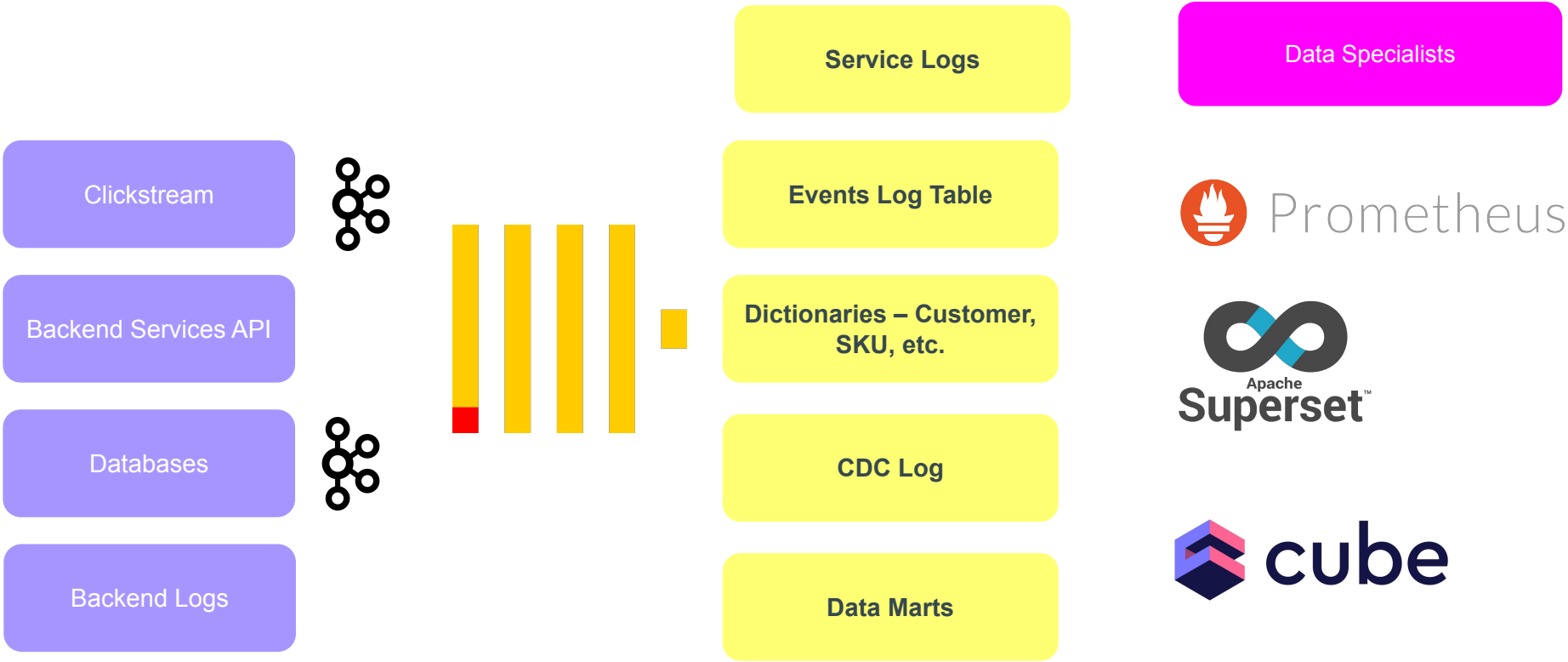
Events Log Table

Dictionaries – Customer, SKU, etc.

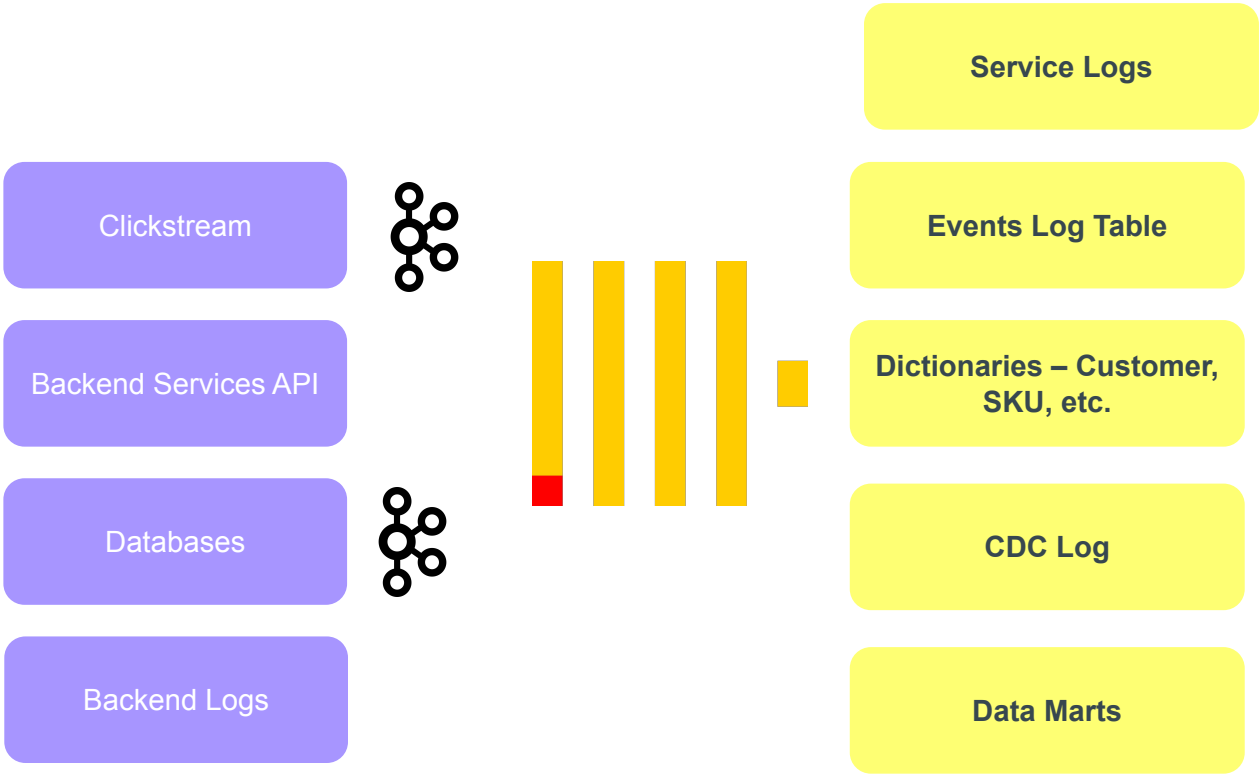
Apache Superset™

cube

Architecture recap – ClickHouse POV

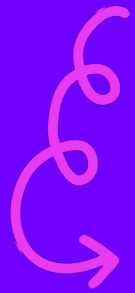


Architecture recap – ClickHouse POV



TIME TO QUESTIONS

Or come find me at networking to ask deeper questions!



Follow me