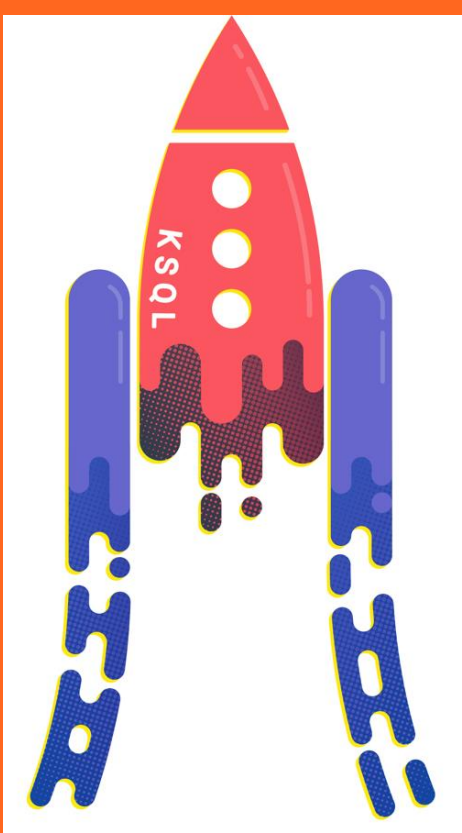


# IoT Sensor Analytics with Apache Kafka, KSQL and TensorFlow

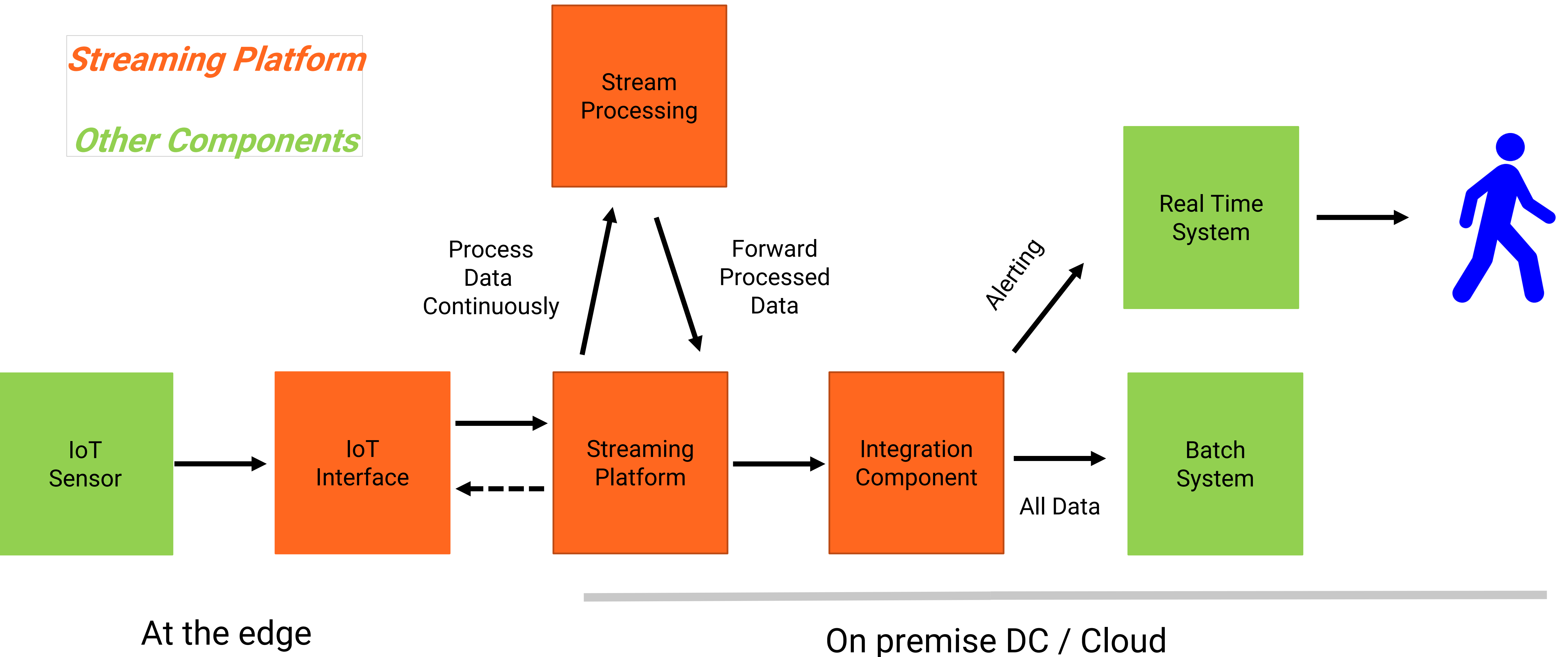
Kafka-Native End-to-End IoT Data Integration and Processing

**Kai Waehner** - Technology Evangelist

kontakt@kai-waehner.de - LinkedIn – Twitter : @KaiWaehner  
[www.confluent.io](http://www.confluent.io)  
[www.kai-waehner.de](http://www.kai-waehner.de)



# Internet of Things – End-to-End Data Processing





# The first analytic model

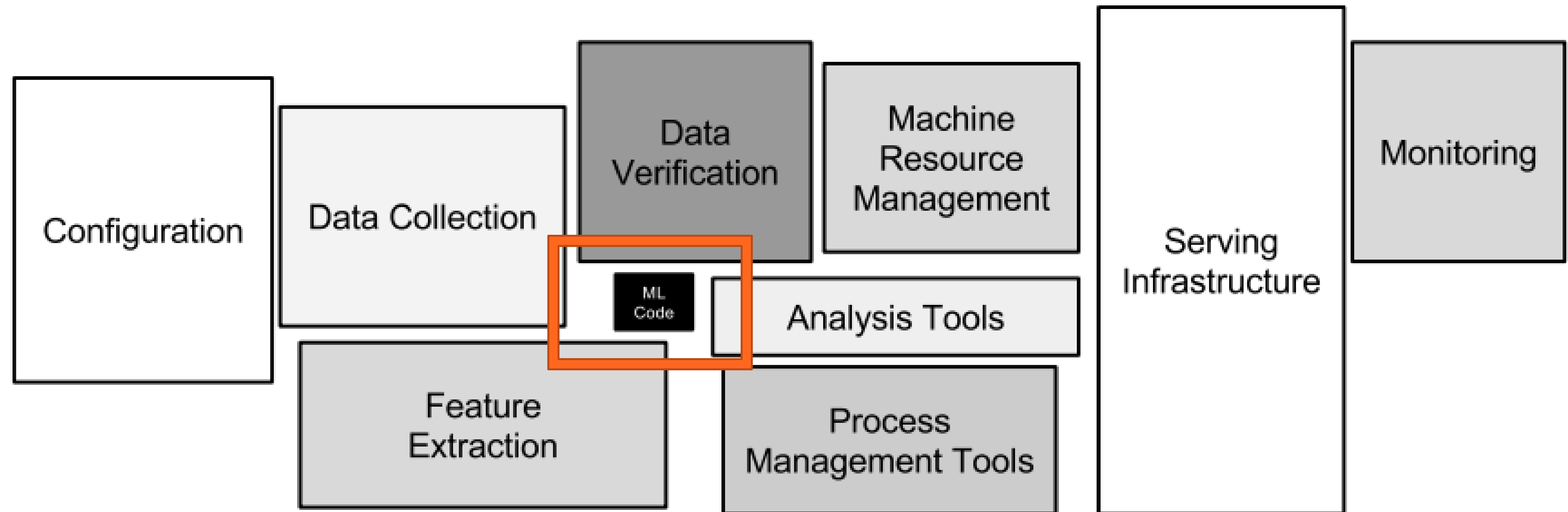


```
def queen(queens):  
    for i in range(BOARD_SIZE):  
        test_queens = queens + [i]  
        try:  
            validate(test_queens)  
            if len(test_queens) == len(queens):  
                return test_queens  
        except:  
            return add_queen(test_queens)
```

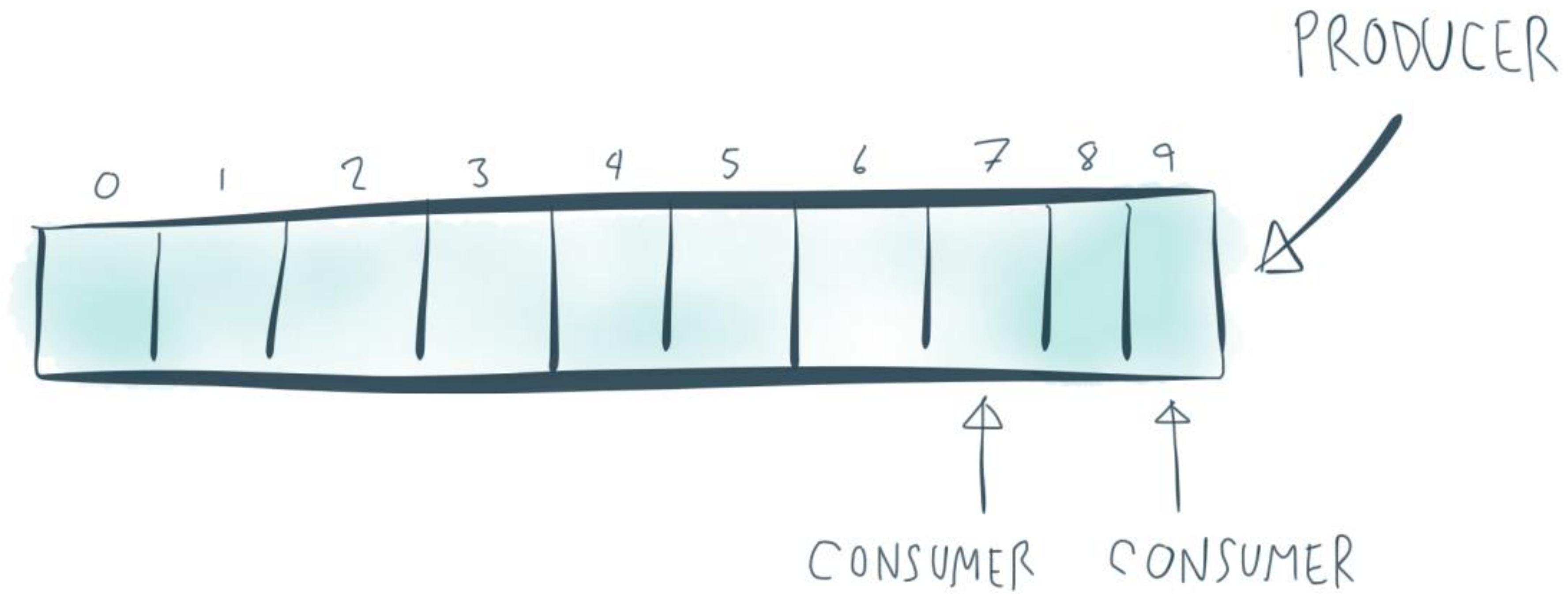


How to deploy the models  
in production?  
...real-time processing?  
...at scale?  
...24/7 zero downtime?

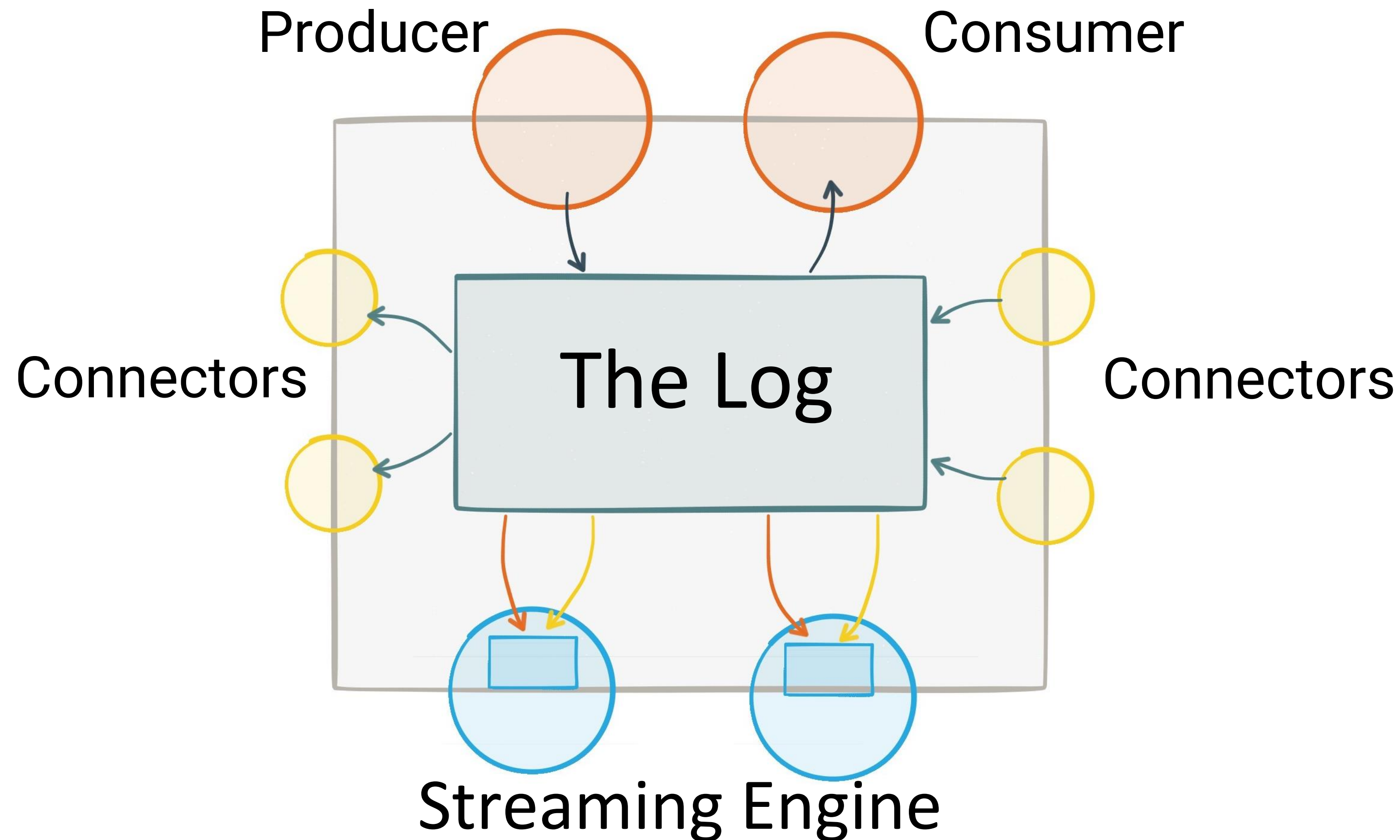
# Hidden Technical Debt in Machine Learning Systems







# Apache Kafka – The Rise of an Event Streaming Platform



1 PUB/SUB  
2 STORE  
3 PROCESS

# Apache Kafka at Scale at Tech Giants



> 4.5 trillion messages / day

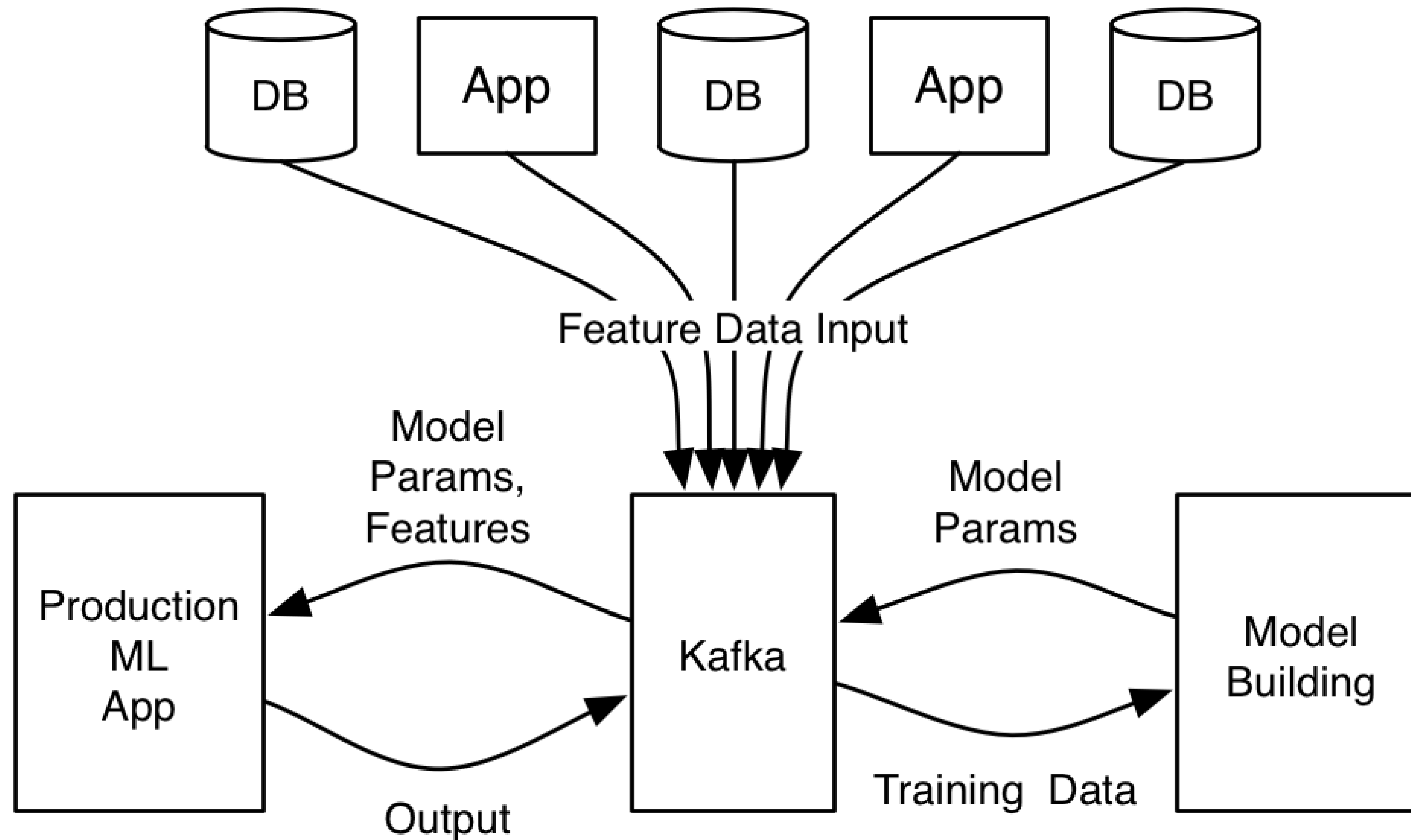
**NETFLIX**

> 6 Petabytes / day



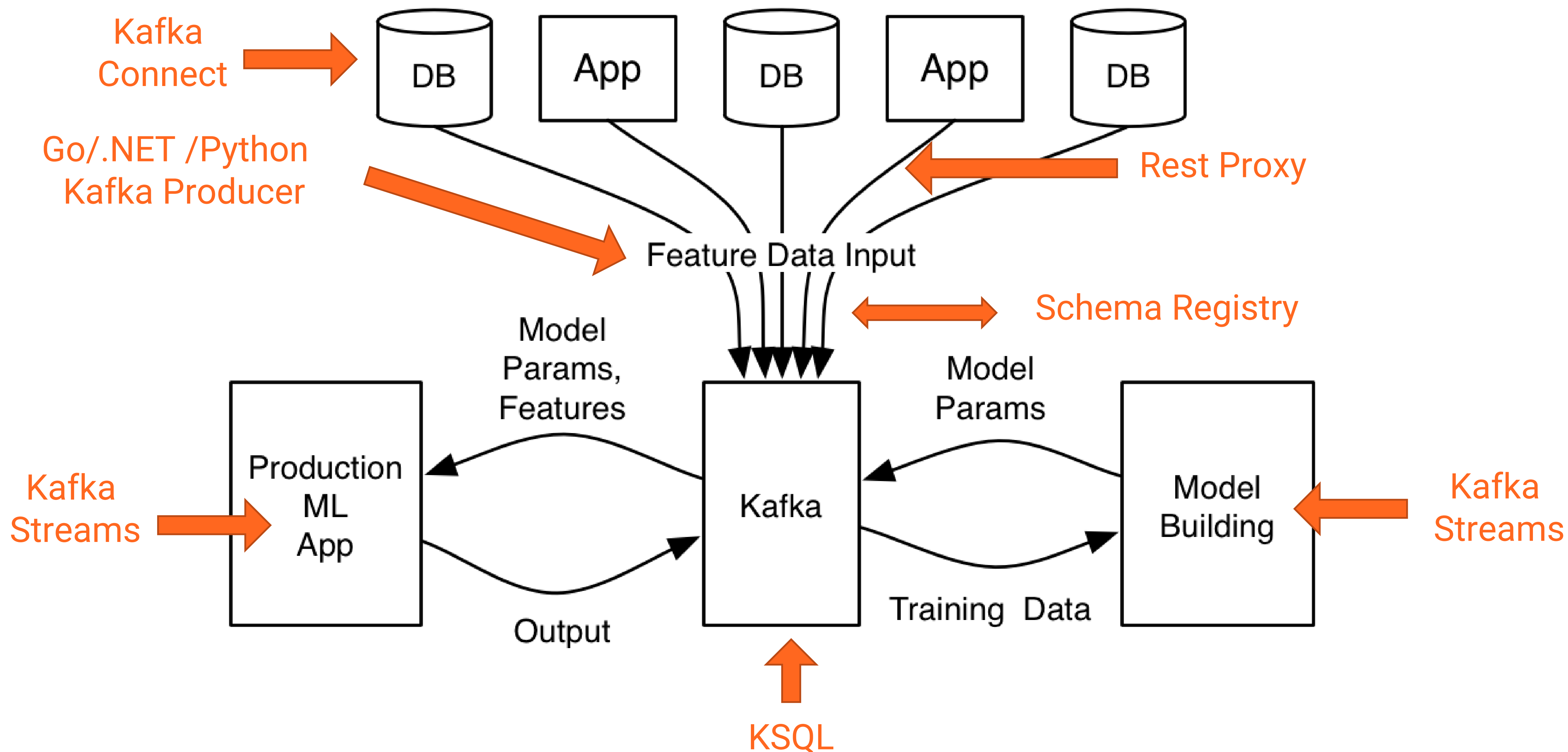
“You name it”

# Apache Kafka as Infrastructure for ML

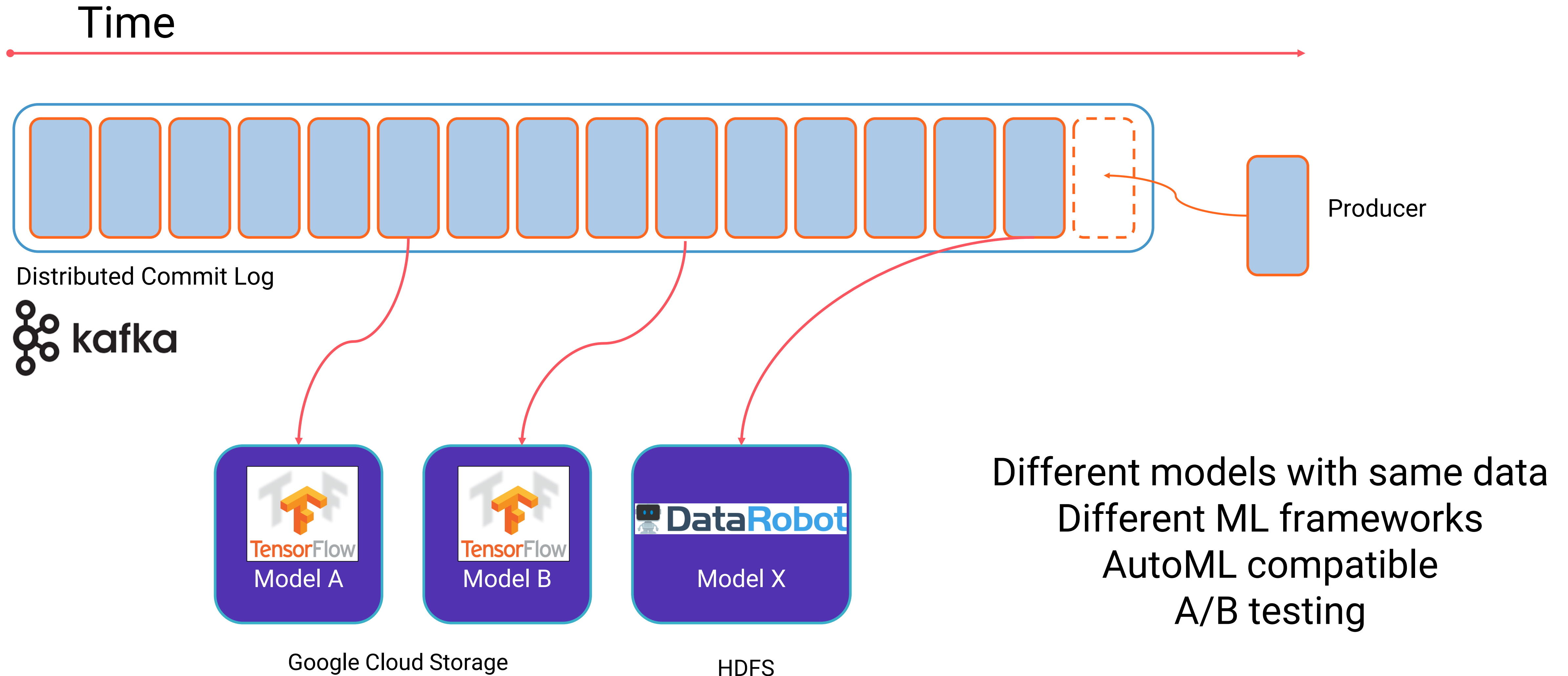




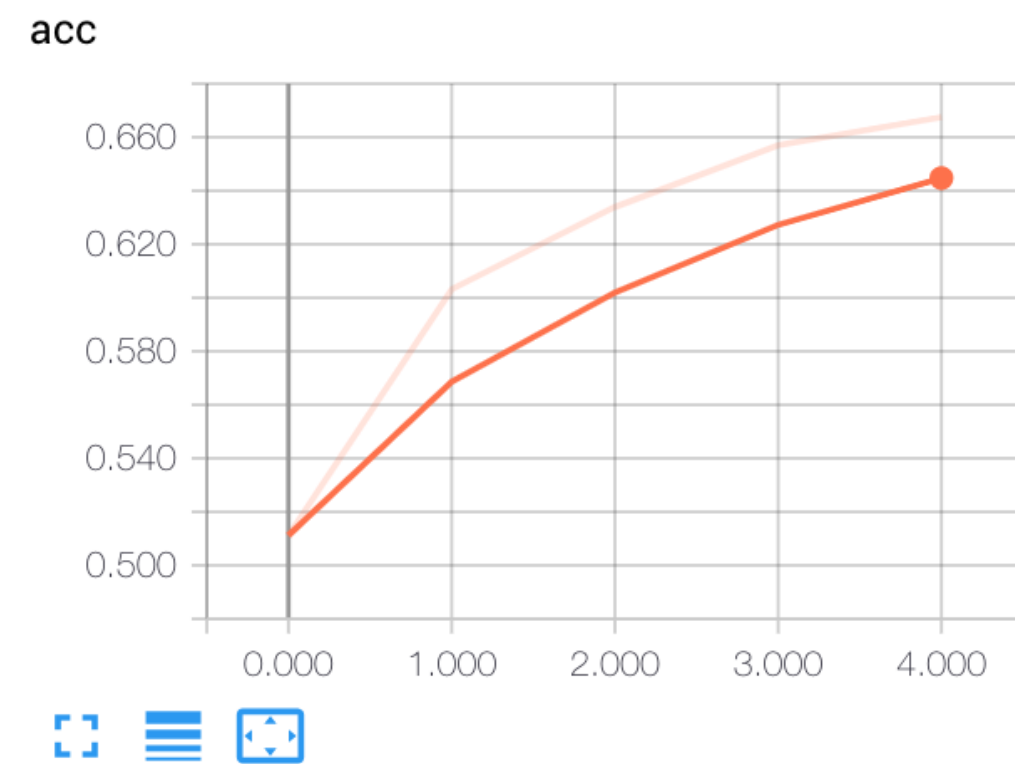
# Apache Kafka's **Open Ecosystem** as Infrastructure for ML



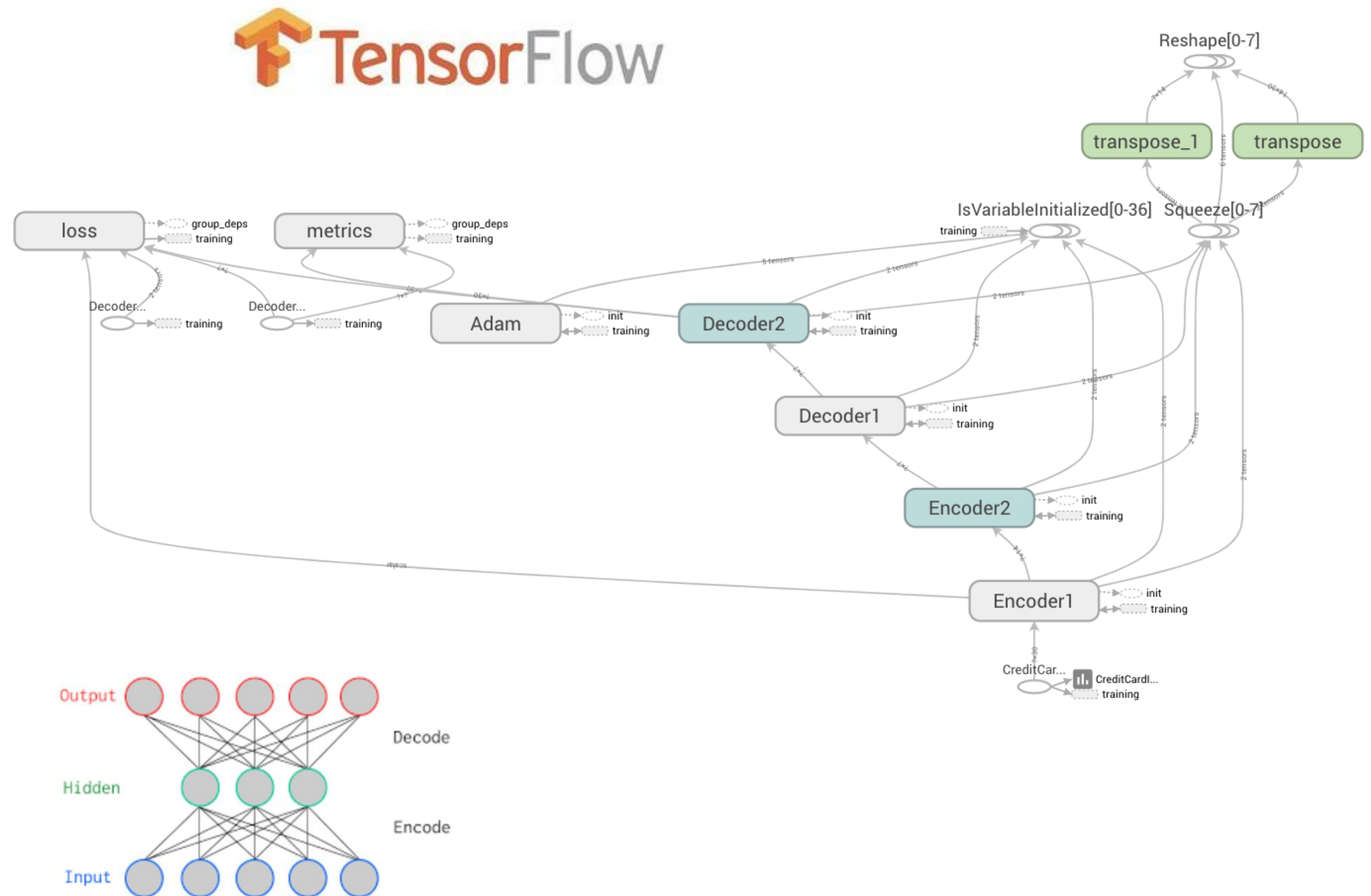
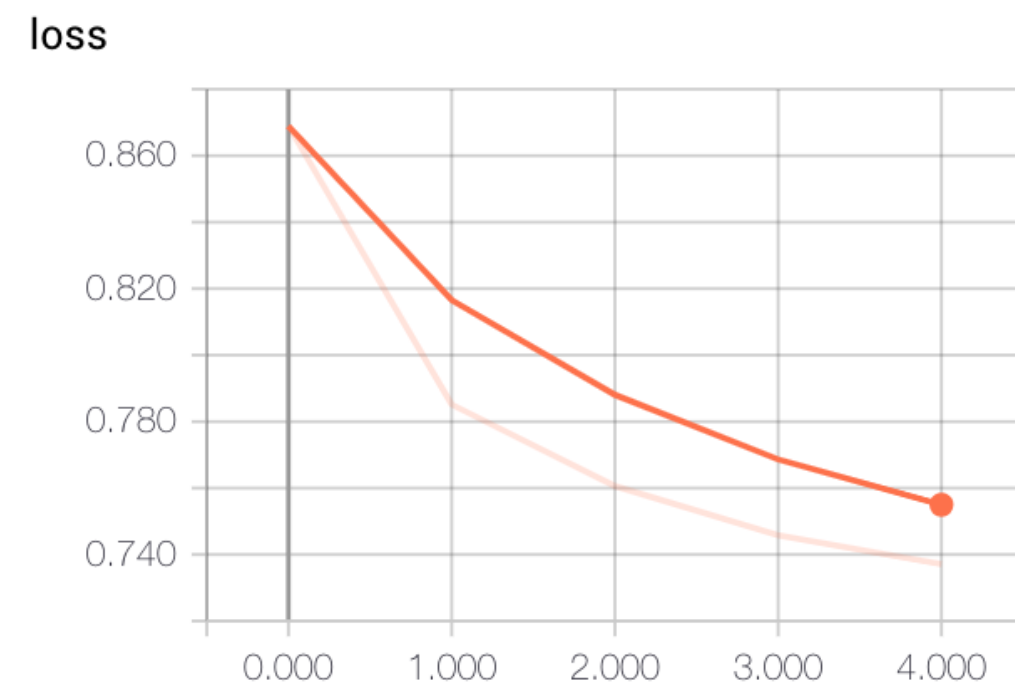
# Replayability — a log never forgets!



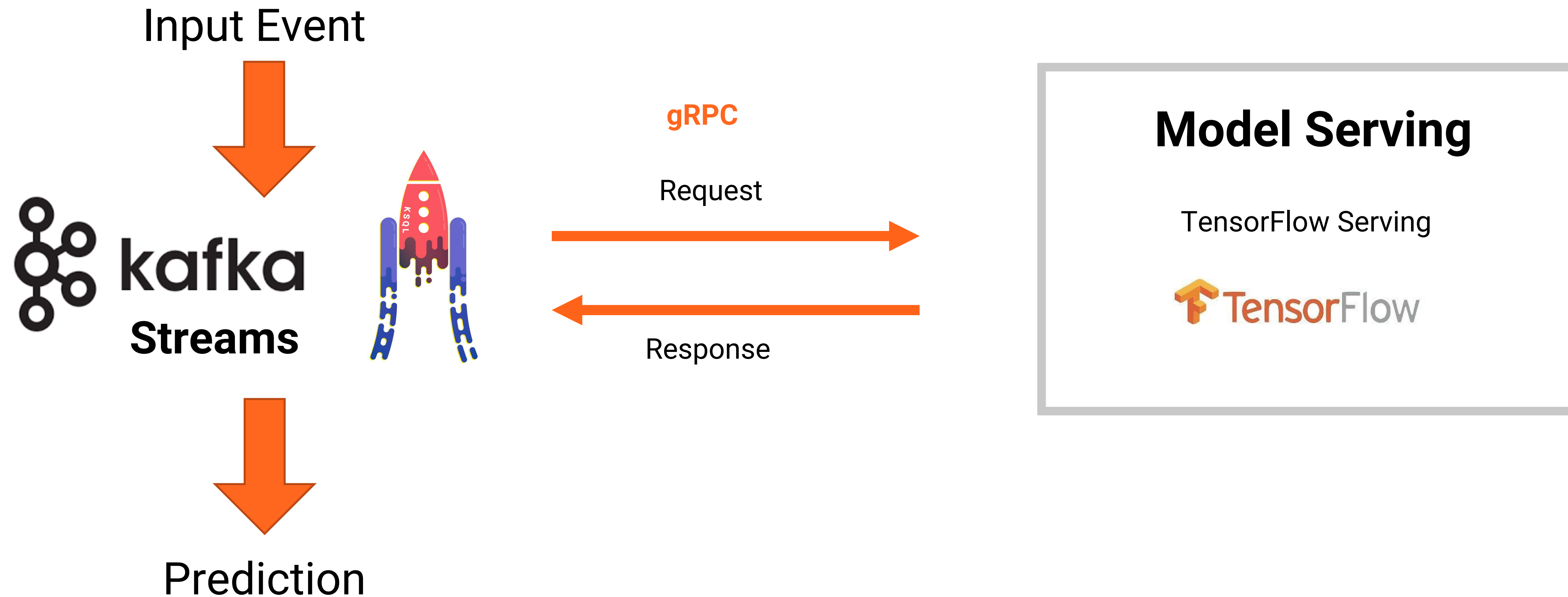
# Analytic Model (Autoencoder for Anomaly Detection)



loss



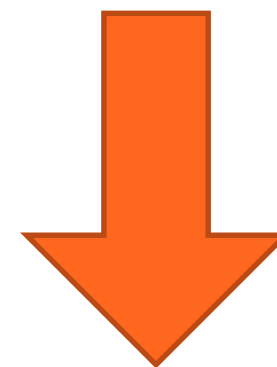
# Model Deployment #1: RPC Communication to do Model Inference





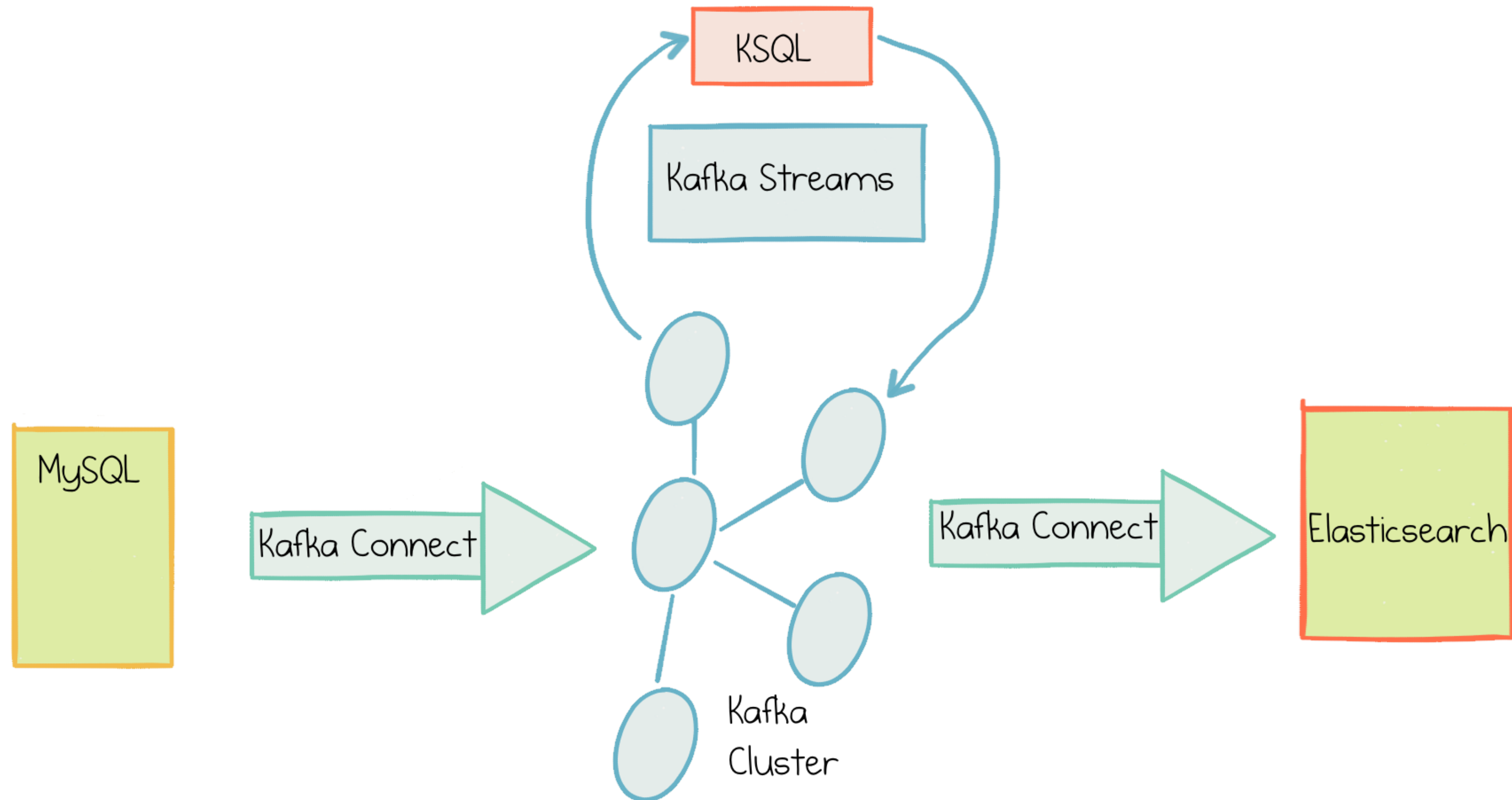
# Model deployment #2: Model interference natively in the App

Input Event



Prediction

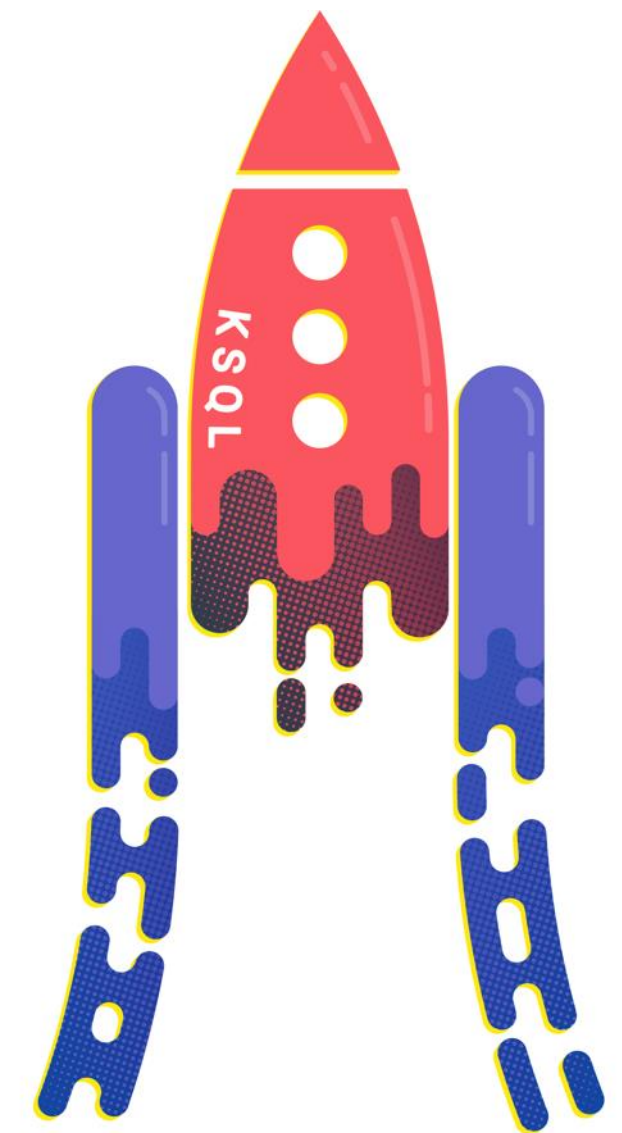
# KSQL – The Streaming SQL Engine for Apache Kafka



# KSQL – Continuous Queries for Streaming ETL / Anomaly Detection

```
CREATE STREAM car_sensor_XYZ AS
SELECT car_ID, car_model, owner_id value FROM car c
LEFT JOIN users u ON c.owner_id = u.user_id
WHERE c.model = 'Luxury Car ABC';
```

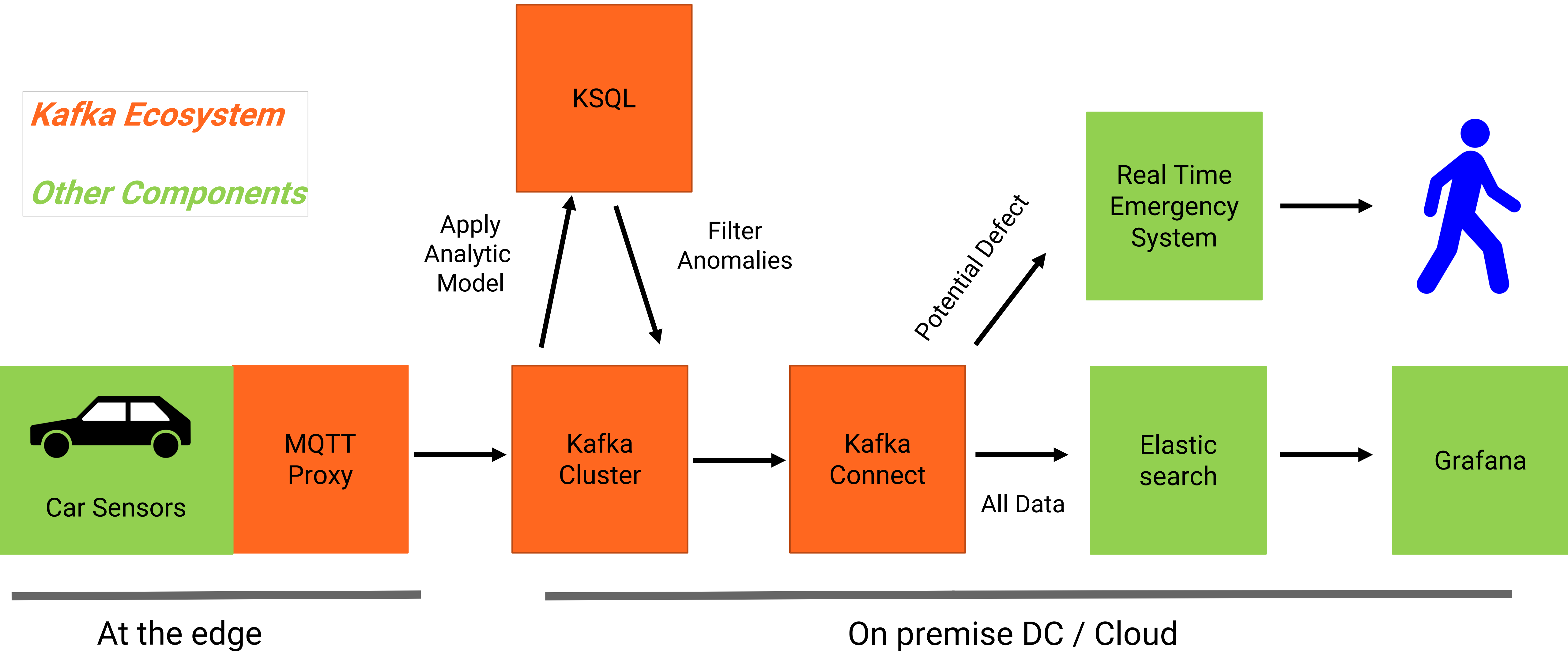
```
CREATE TABLE possible_detect AS
SELECT sensor_value, count(*)
FROM car_sensor
WINDOW TUMBLING (SIZE 120 MINUTES)
GROUP BY car_id
HAVING count(*) > 5;
```



# KSQL and Deep Learning (Auto Encoder) for Anomaly Detection

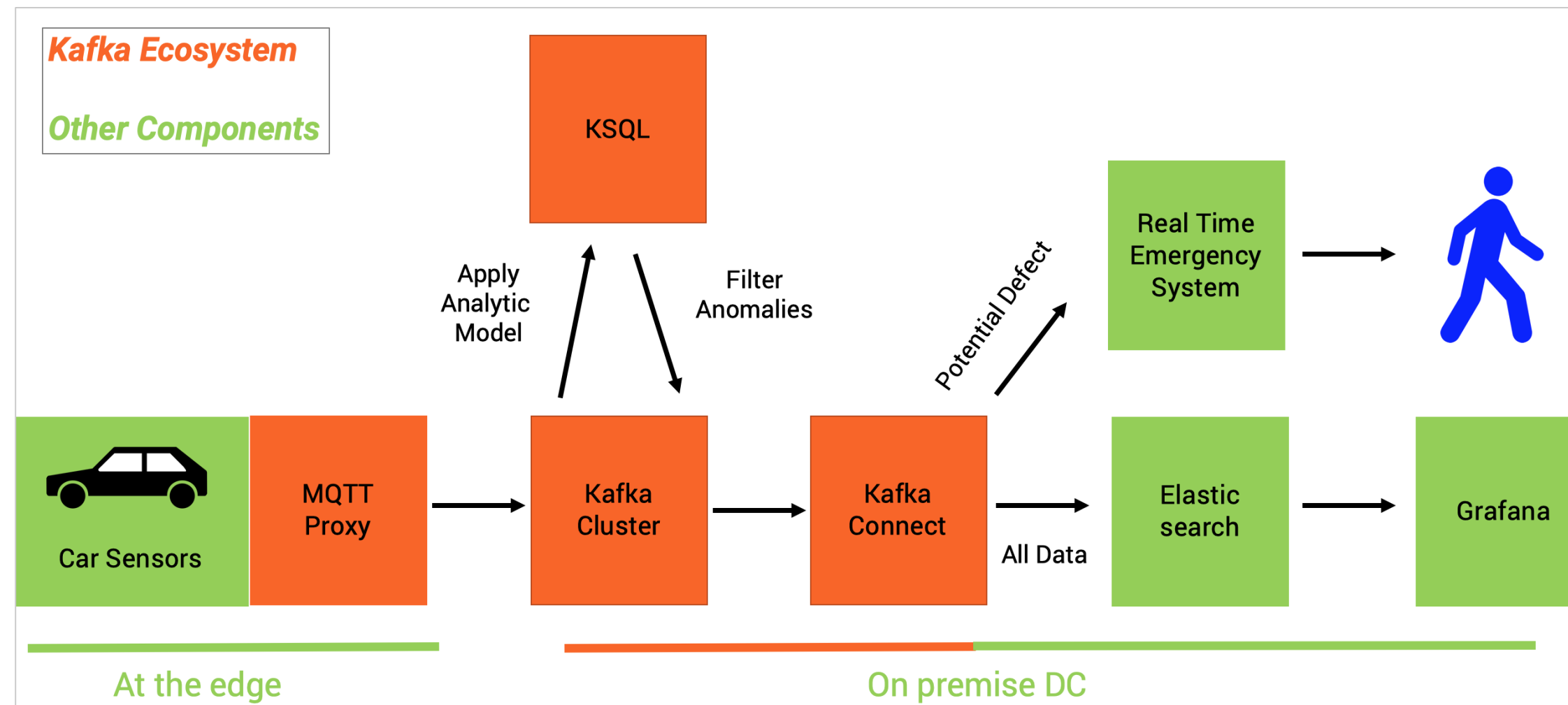
*Kafka Ecosystem*

*Other Components*





# Model Deployment with Apache Kafka, KSQL and TensorFlow



```
“CREATE STREAM AnomalyDetection AS  
SELECT sensor_id, detectAnomaly(sensor_values)  
FROM car_engine;”
```



User Defined Function (UDF)



# Model Training with Python, KSQL, TensorFlow, Keras and Jupyter

```
jupyter python-jupyter-apache-kafka-ksql-tensorflow-keras Last Checkpoint: a few seconds ago (autosaved)

File Edit View Insert Cell Kernel Help Trusted Python 3

Setup for the KSQL API:

In [1]: from ksql import KSQLAPI
        client = KSQLAPI('http://localhost:8088')

In [3]: client.create_table(table_name='users_original',
                           columns_type=['registertime bigint', 'gender varchar', 'regionid varchar', 'userid varchar'],
                           topic='users',
                           value_format='JSON',
                           key = 'userid')

Out[3]: True

In [6]: client.ksql('show tables')

Out[6]: [{'@type': 'tables',
  'statementText': 'show tables;',
  'tables': [{'type': 'TABLE',
    'name': 'USERS_ORIGINAL',
    'topic': 'users',
    'format': 'JSON',
    'isWindowed': False}]]

Execute sql query and keep listening streaming data:

In [4]: query = client.query('select * from users_original')
        for item in query: print(item)

{"row":{"columns":[1543510415238,"User_5",1489124138779,"OTHER","Region_4","User_5"]},"errorMessage":null,"finalMessage":null}
{"row":{"columns":[1543510415361,"User_8",1516543544214,"OTHER","Region_3","User_8"]},"errorMessage":null,"finalMessage":null}
{"row":{"columns":[1543510415897,"User_2",1515464455832,"FEMALE","Region_9","User_2"]},"errorMessage":null,"finalMessage":null}
{"row":{"columns":[1543510416775,"User_3",1514158220288,"OTHER","Region_4","User_3"]},"errorMessage":null,"finalMessage":null}
```

## Start Backend Services (Zookeeper, Kafka, KSQL)

The only server requirement is a local KSQL server running (with Kafka broker ZK node). If you don't have it running, just use Confluent CLI:

```
! confluent start ksql-server
```

This CLI is intended for development only, not for production  
<https://docs.confluent.io/current/cli/index.html>

```
Using CONFLUENT_CURRENT: /var/folders/0s/0xdkb9n12yqdb3fs71926z3c0000gp/T/confluent.0nWtkbhu
Starting zookeeper
zookeeper is [UP]
Starting kafka
kafka is [UP]
Starting schema-registry
schema-registry is [UP]
Starting ksql-server
ksql-server is [UP]
```

## Data Integration and Preprocessing with Python and KSQL

First of all, create the Kafka Topic 'creditcardfraud\_source' if it does not exist already:

```
! kafka-topics --zookeeper localhost:2181 --create --topic creditcardfraud_source --partitions 3 --replication-factor 1
```

<https://github.com/kaiwaehner/python-jupyter-apache-kafka-ksql-tensorflow-keras>

# Deep Learning UDF for KSQL for Streaming Anomaly Detection of MQTT IoT Sensor Data

The screenshot shows the GitHub repository page for 'ksql-udf-deep-learning-mqtt-iot' by user 'kaiwaehner'. The repository has 4 watches, 9 stars, and 0 forks. The main content area displays the repository name and a list of tags including kafka, kafka-connect, kafka-client, confluent, confluent-platform, open-source, deep-learning, machine-learning, tensorflow, h2oai, java, ksql, and mqtt. Below the tags, it shows 23 commits, 1 branch, 0 releases, 1 contributor, and the Apache-2.0 license. A table of recent commits is visible, with the latest commit 'Fixed order of commands' by Kai Waehner and Kai Waehner, dated 14 hours ago. The commit history table includes the following entries:

Commit	Description	Time
df1cc64	Fixed order of commands	14 hours ago
	Added picture with MQTT / Kafka architecture	2 days ago
	Removed Kafka Streams code (not needed for the UDF)	2 days ago
	Initial commit	8 days ago
	Explained that kafkakat is optional and just a comfortable alternativ...	15 hours ago
	Fixed order of commands	14 hours ago
	Removed Kafka Streams dependency (not needed for the UDF)	2 days ago
	Bash script which generates new sensor events continuously	15 hours ago

Below the commit history, the README.md file is shown, containing the title 'Deep Learning UDF for KSQL for Streaming Anomaly Detection of MQTT IoT Sensor Data'.

<https://github.com/kaiwaehner/ksql-udf-deep-learning-mqtt-iot>

# This is just the beginning of new era!

## Confluent Vision for Kafka:

### Global

Automated **disaster recovery**  
Global applications with **geo-awareness**

### Infinite

Efficient and infinite data with **tiered storage**  
Unlimited horizontal scalability for single clusters  
Faster elastic scaling for brokers and partition

### Elastic

Easy Kubernetes- based orchestration and management with Confluent operator  
Faster **elastic scaling** when adding brokers and partitions

### Cloud-Native Apache Kafka





Kai Waehner

Technology Evangelist

[kontakt@kai-waehner.de](mailto:kontakt@kai-waehner.de)

[@KaiWaehner](#)

[www.kai-waehner.de](http://www.kai-waehner.de)

[www.confluent.io](http://www.confluent.io)

[LinkedIn](#)

Questions? Feedback?  
Please contact me!

Come and find out more  
about Kafka & Confluent  
on **Booth A3**

